A-188

Design and Evaluation of MiMaze, a Multi-player Game on the Internet

Laurent GAUTIER and Christophe DIOT
INRIA Sophia Antipolis (http://www.inria.fr/rodeo/)

Abstract: This paper describes the design, implementation, and evaluation of MiMaze, a distributed multiplayer game on the Internet. This work focusses on the transmission control mechanisms that have to be used to make it possible to play a real-time multiplayer game on the Internet. Our major contribution is to have designed and implemented a completely distributed communication architecture based on IP multicast. The major elements of MiMaze architecture are a multicast communication system based on RTP/UDP/IP, and a distributed synchronization mechanisms to guarantee the consistency of the game, regardless network delay. This paper provides an early evaluation of the game on the Mbone, as well as elements of monitoring to compare the "efficiency" of such applications. Missing mechanisms and future work are discussed at the end of the paper.

Keywords: Distributed Architecture, Interactive Applications, Synchronization, Experimental System, Group Communication, Transmission Control, Internet.

1.0 Introduction

This article describes the design and the evaluation of a multi-player (distributed) game over the Internet. MiMaze is a distributed (i.e. serverless) game that uses an unreliable communication system (MiMaze transmission control is based on RTP [9] over UDP/IP multicast [16]). Multi-player games are representative of the new generation of interactive multimedia applications, that also includes Distributed Interactive Simulations (DIS), digital battle fields and Air Traffic Control (ATC)[3][4].

Because of the distributed architecture of this application, we had to design a synchronization mechanism to cope with different transmission delays among the participants. This synchronization mechanism (called bucket synchronization) will be analyzed later in this document. It is the minimum functionality required to play a distributed application on the Internet.

For this work, we have chosen to follow some of the DIS [1][2] rules. In particular, we require that any action issued in the game must be displayed to any participant before 100ms.

Before we move to the technical content of this paper, we define the notions of "object" and "state". The game is made of objects. E.g., avatars are objects; the terrain is an object. When two objects have a common behavior, they are considered as a single object. For example, when an avatar shots another avatar, the bullet remains part of the shoting avatar until it hits the target avatar. The exact description of all the game objects (e.g., position, displacement, nature, etc.) is called the game state. Because the architecture is distributed, each participant computes its own view of the game state.

This article is structured as follows. Section 2 describes MiMaze. We start with a description of the game and of its functional architecture. Then we give a detailed description of the bucket synchronization algorithm. In section 3, we describe and analyze early performance measurements realized on the Mbone. The monitoring tool used to re-synchronize the distributed traces is briefly described. The performance results give us an opportunity to discuss what enhancements are necessary to make MiMaze real-time, interactive, and scalable. We conclude with a discussion of the next MiMaze design and evaluation steps, and with the extension of the game to a more realistic interactive environment.

2.0 Description of MiMaze

The characteristics of distributed games are very similar to those of DIS applications [1][2]. The main difference in the case of Mimaze is that the application CPU requirement is low. One reason for choosing a "simple" game is that choosing a more complex game can make it difficult to analyze the game traffic parameters. The DIS characteristics that apply to MiMaze are:

- Interaction delay: any action issued by any participant must reach other participants within 100ms. If the network delay is more than 100ms, the received action (encoded in an Application Data Unit, or ADU) is late and is not used by the application.
- Large number of participants. As we have no idea of what a "large" group is for an interactive application, we have decided to have a single group in MiMaze until initial experiments tell us the upper limit on a single group size.
- Interactive data are short (few tens of bytes) and frequent (but not necessarily periodic). They differ from other multimedia data (audio and video). Each game object can also transmit its local state which, in the worst case (many articulated parts, complex objects), represent a small amount of information (less than few hundred bytes).
- High level of dynamicity in group structure and topology. Participants join and leave the session dynamically. In this context, the IP multicast model [16] is particularly convenient.
- Information is “continuous”. In most of the cases, the behavior of avatar *X* at time *n+1* is an evolution of its behavior at time *n* (for example the displacement of an avatar, or the trajectory of a bullet). We have consequently decided to use an unreliable communication support based on RTP and UDP.

2.1 MiMaze design characteristics

MiMaze is an evolution of iMaze [8]. iMaze is a bi-dimensional "Pacman" game in which each player has a 3D representation of its view of the game (see screen shot figure 1). Avatars (Pacman) evolve in a labyrinth where they try to kill each other. Each participant, besides having a 3D representation of its vision domain, also has a 2D "global" view of the game (from the top), that shows the location of all players.

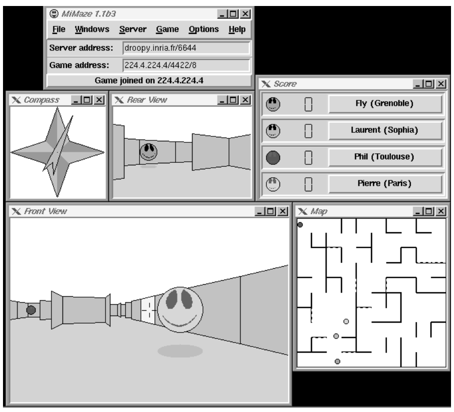

Figure 1. MiMaze screen shot

2.1.1 MiMaze distributed architecture

MiMaze [13] is to our knowledge the only game with *a fully distributed architecture using multipoint communication support* (a server is only used when a new entity joins a session, e.g., to learn the session group address and to download the maze). Fully distributed architectures have many advantages compared to server-based architectures:

- Robustness. In a distributed architecture, the failure of any of the participants has no effect on other participants. Each participant is independent, and locally has all necessary information to compute the state of the game at any time. The architecture being serverless, it is also more simple for an application provider to deploy a game. The quality observed by a participant only relies on the network capacities and on the participant equipment.

- Scalability. We have identified two different factors that limit the scalability of centralized architectures:

  - All data converge to the server. The server is consequently a natural bottleneck. Once its maximum CPU load is reached, the game state computation frequency slows down in order to continue to serve all participants, and to collect all participants data. Interactivity is then lost[1].

  - With a server architecture, the amount of data transferred on the network is generally bigger (the global state is usually bigger than the sum of all participants data) than with a distributed architecture (in which the game state is never carried by the network).

  In this sense, distributed architectures are more scalable than centralized architectures.

- Minimum delays. In a centralized architecture, information takes more time to be delivered than in a distributed architecture. When information leaves a participant's location, it has to reach first the server where the game state is computed. Then, the state is forwarded to all participants to be displayed. In the distributed architecture, any action crosses the network only once to reach its final destinations.

Centralized architectures do have advantages. One of them is that all the participants in the game share the same game state (global consistency). Moreover, the server introduces a natural synchronization among players since they all display the same game state at the same time. Another important feature of centralized architectures is that it allows game companies to make money.

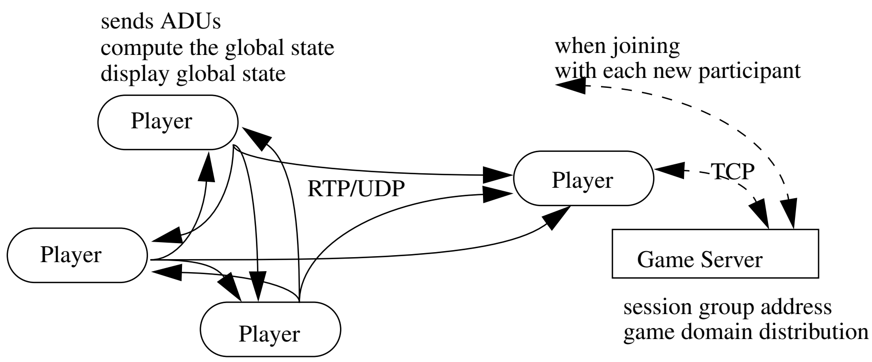

Figure 2. MiMaze architecture

But the main advantage of centralized architectures is that the presence of a server makes cheating difficult. In a totally distributed architecture, each entity makes its own decisions, and there is no authority to identify potential cheaters. In a centralized architecture however, all the information flows to the server, which can authenticate the exactness of the global state. Consequently, the deployment of distributed architectures will require the use of specific distributed mechanisms to deal with the "honesty" of participants [12].

1. This also applies to distributed architectures. However, computational resources saturate at a larger group size than with a single, centralized server.

2.1.2 Object characteristics

The DIS standard defines numerous types of information to be packetized and managed by the application. The DIS standard is an application-level IEEE standard designed by applications experts. It is not optimized for network transmission. Even worse, it is unrealistic to use this protocol over the Internet [3][4].

Most of the DIS packets (called PDU's Protocol Data Units) apply to distributed games. There are more than 25 PDUs defined in the DIS standard [1][2]. The most frequently used is the ES PDU (Entity State PDU) which carries state information describing the game objects. A specific PDU type has also been defined for "exceptions" such as collisions, fire, detonation, and also for logistics, and control.

In MiMaze, packets (called ADU's Application Data Units) are slightly different from those defined by the DIS standard. For network optimization reasons, we use only one packet type (called the Object State ADU - or OS ADU) whose format is described figure 3. An Object State ADU is made of two parts: the header, and the payload. To make it more flexible for network transmission, each information field in MiMaze ADUs is encoded as a descriptor in the packet payload. The generic structure of a descriptor is [type of information/size of information/information].

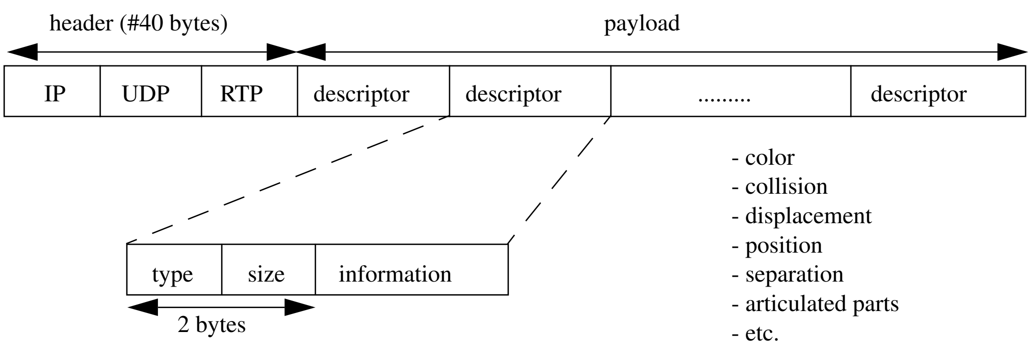

Figure 3. MiMaze ADU format

The OS ADU is "equivalent" to the DIS ES PDU. An object uses OS ADUs to periodically describe its behavior (position, speed, etc.). Exception information such as shooting and collisions are sent as part of the Object State ADU. OS ADUs are transmitted with an unreliable protocol; the frequency of the transmission provides a natural redundancy that minimzes the impact of ADU losses in reception.

The main difference with DIS is that in MiMaze, the bullets remain part of the avatar that shot them. This characteristic comes from the iMaze game; modifying it would have required the complete re-design of the game. But we consider this as an important issue for the next release of MiMaze (from a group management point of view, it is not clear if the articulated parts have to be independent).

In MiMaze, the size of an ADU is 52 bytes (including 16 bytes for MiMaze, 8 for RTP 8 for UDP and 16 for IP). This is an important parameter in the game saclability and consistency analysis.

2.1.3 Related work

Three related pieces of work are of particular interest to us:

- Amaze can be considered as MiMaze ancestor. It was designed by Berglund and Cheriton in 1984 [14] to be played on a LAN (Amaze uses a point to point communication support). Mimaze and Amaze both have a distributed architecture. But states are managed differently in both game (Amaze transmits the game state on the network, and maintains replicated copies of the game state). Amaze also uses dead-reckoning for "state extrapolation".

- "le deuxième monde" [15] ("the second world") is a (French) game commercialized by a satellite TV provider. The game is based on a CD-ROM that stores 3D maps of Paris. Participants can create their own avatar and circulate in Paris, where they meet other avatars, and where they can communicate. The game in neither real-time nor distributed. Each participant is connected to a game server using a TCP/IP connection. But it is interesting to see the commercial success of this game, where up to 150 participants connect to the same session.
- Spline [17][18] is a virtual distributed interactive world with 3D animation and spoken interaction. Spline has a distributed architecture which is based on the DIS standard. Most of the effort in Spline has been done on flow synchronization. But the communication architecture is not using the Internet protocol suit and multicast is emulated using a server.

2.2 The Bucket Synchronization Mechanism

With each MiMaze participant computing its own local view of the global state, synchronization must be introduced to allow ADUs issued at the “same time” to be processed "together" by any participant (see figure 4).

The synchronization mechanism makes sure that ADUs processed to evaluate the global state of the game at time *t* were all issued by the participants during the same interval of time; ideally this interval should be “close to” *t*. In MiMaze, time is divided into fixed length sampling periods and a bucket is associated with each sampling period. All ADUs received by a player that were issued by senders during a given sampling period are stored by the reciever in the bucket corresponding to that interval. When a participant has to deliver an updated global state, it computes all the ADUs available in the "current" bucket.

For example, in figure 4, without synchronization, the local ADU issued at t_3 would be processed together with the ADU issued at t_1 from another location (but received at t_2, which is in the same state processing interval than t_3). Bucket synchronization allows information received at t_0 to be delayed in the bucket *d* (that will be processed at t_d) in order to be synchronized with the ADU issued at t_1 by another participant.

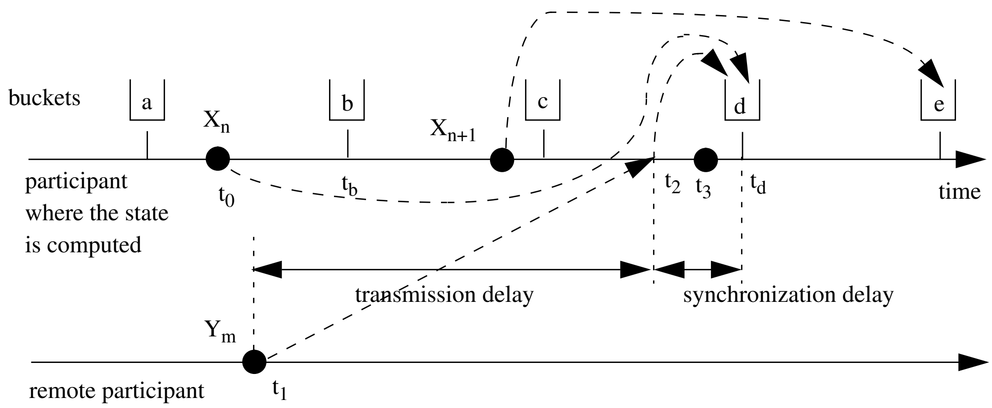

Figure 4. The bucket synchronization mechanism

Another aspect of synchronization is that all participants should display the same game state at the same time. As the display frequency is high enough (equal or above 25 images per second), we did not implement a specific mechanism to perform this synchronization. But such mechanism could be needed for application having sharper time constraints than MiMaze (or applications unable to process buckets frequently enough).

2.2.1 Synchronization delay

The synchronization delay is computed on ADU reception to determine in which bucket the ADU content should be stored. The MiMaze synchronization delay is equal to 100ms minus the network delay. In other words, an ADU issued at absolute time *t* will be rendered at the receiver (if at all) in the interval containing absolute time *t + 100ms*. If the network delay is more than 100ms the ADU is discarded[2]. The bucket mechanism is very close to a playout buffer mechanism [10] used to reduce network jitter effects in packet audio information.

2.2.2 Bucket frequency

The bucket frequency defines the rate at which a new game state is computed (and displayed). This bucket frequency must be chosen carefully. The human vision being continuous for a frequency of 25 images per second, we have chosen to compute 25 buckets per second. The bucket frequency is a receiver parameter that should not be influenced by network parameters. The only reason to reduce this frequency is local CPU congestion.

2.2.3 Global clock mechanism

The bucket synchronization mechanism uses a global clock mechanism to evaluate the delay between participating entities. The clock has to be accurate (a 10ms precision is required [19]) and continuous. In our implementation, we use NTP [6]. In case NTP is not available, we use a NTP-like algorithm based on the evaluation of the round trip time [6]. There are three difficulties with NTP:

- There are 3 levels of NTP servers (stratum) and it is very difficult to maintain good synchronization among participants when level 3 servers are involved. Lower stratum mechanisms (e.g. ntpdate) are not sufficient.
- NTP encodes clock information in 64 bits, while RTP uses a 32 bit clock. MiMaze has to manipulates both clock representations.
- NTP does not provide a reference clock signal and each participant has to compute an offset for every other participant.

In the current implementation, in order to increase the precision of NTP under stratum 2, we use both NTP and our NTP-like mechanisms to compute clock offsets.

3.0 Performance evaluation

This section is organized in five subparts. We begin with the description of the experimental settings. The monitoring tool designed to re-synchronize the distributed traces is briefly presented. We then define the distributed game metrics that are analyzed. The evaluation section begins by providing an analysis of the behavior of MiMaze during an experimental session on the Mbone. Networks and game parameters are analyzed. We conclude this section with a discussion on MiMaze efficiency and consistency.

3.1 Experimental environment

We have performed an evaluation of MiMaze on the Mbone [11] with up to 25 players located in various places in France. The architecture of the experiment multicast tree is given figure 5.

The three delay values given figure 5 are averaged measured end-to-end network delays. The computers participating in the experiment were SUN (SPARC 10, 20, ULTRA), DEC Alphas, and PCs. Note that we have a network with a delay that is, on average, smaller than 100ms.

3.2 Monitoring

During the experiment, participants play MiMaze and each participant collects a trace that is composed of:

- Network level data: for each received ADU, we collect the senders' identity, the emission time-stamp, the reception time-stamp, and the sequence number.

2. Dead-Reckoning can be used to deliver late ADUs.

- Application level data: information contained in all ADUs sent and received is time-stamped and collected in the trace file. These data allow us to reconstruct the state of the game computed by each participant.
- Synchronization data: round trip times and clock offsets with respect to each participant. These data are used to synchronize the traces.

The main difficulty is not in benchmark computation, but in trace re-synchronization. As a result of the system being distributed, there is no absolute clock in a game session, and each participant computes offsets for each other participant. To re-synchronize traces, we apply the least square algorithm to the clock offset measured to compute an approximation of the real offset. This offset approximation is used as a reference value for resynchronization.

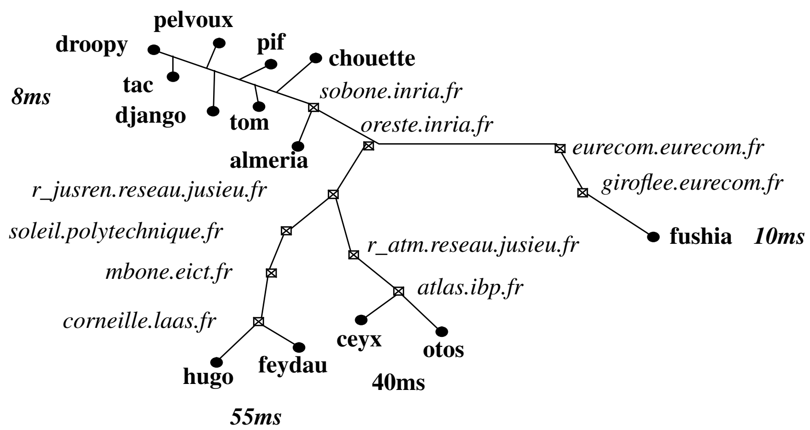

Figure 5. Mbone architecture during the evaluation

This re-synchronization method is only valid when measured offsets are not too far from the reference offset. Consequently, when NTP synchronization was very loose, we were not able to resynchronize traces. For this reason, the following benchmarks have been obtained for those time intervals when it was possible to re-synchronize clocks.

3.3 Distributed game metrics

The first problem is to define objective criteria for multi-player games evaluation. These criteria must reflect how far the distributed game behavior is from the behavior that would have occured if the game was played with no network delay and no network loss. There are three important functions to evaluate the game behavior:

- Scalability. The number of participants can be high, as well as dynamic. The quality of the game should not be influenced by the number of participants. The complexity of game should be at worst *o(N)* where *N* is the number of participants.
- Interaction. The control algorithms provided with a distributed game must hide the network delay that separate participants. Maintaining interaction among participants means that the view of the game at each location is not dependent on the network delay.
- Consistency. In a distributed application, each participant computes its own application's view of the global state. The game is consistent if this view is the same for all participants in a session. We define the reference state as the state obtained with no network delays among participants (it can be computed off-line from each participant's emission trace).

The parameters that influence the previous functions are of two different kinds: network parameters and game parameters. We will not analyze them all in this paper. But it is important to identify them as they can appear in the benchmark discussion.

3.3.1 Network parameters

The network parameters will help us to understand the behavior of our application:

- Network loss is a function of the number of participants *N* and of the underlying network itself.
- Network delay is the delay between any pair of participants (measured application to application, not user to user).

3.3.2 Game parameters

We have identified the following game parameters:

- The game state is defined in the introduction. If the game is consistent, all participants should see the same game state at any time.
- Number of participants (*N*). In a distributed architecture, theamount of traffic carried by the network is $o(N)$.
- Frequency with which each participant sends its ADUs. It is defined by each participant independently, depending on its local performance, and on network parameters.
- Game speed is the game display frequency. The usual game speed is 25 times per second (it corresponds to the bucket frequency). In other experiments we found that it can be reduced to 16 times per second with "tolerable" effect on the game.
- Synchronization delay (*Ds).* As defined earlier, the synchronization delay is the time that should be added to a received ADU to delay it to the appropriate bucket. $Ds = |100 - D|$, where *D* is the network delay.

3.4 Performance analysis

3.4.1 Informal analysis

As it was our first experience with a distributed game over the Internet, it is useful to first informally describe how players "perceived" the game[3]:

- We found that neither the network delay (note that network delays were on average less than 100ms) nor the number of participants had a negative impact on the "quality" of the game. Previous experiments with the original iMaze game were limited to less than 10 participants. The number of participants reached 25 during the experiment with no change in the consistency or in the interaction perception. The displacement of avatars in the labyrinth was smooth and regular at all participant locations.
- The behavior of each participant is independent. Due to the distributed nature of the architecture, few unexpected behaviors occurred during the evaluation session, such as participant disconnection (due to CPU load, network failure, trace memory saturation, etc.) or synchronization loss (NTP re-synchronization or failure). Such a "failure" only annoyed the victim of the problem, but had no effects on the other participants. As the game is designed to remove immediately a participant from the display when a failure is detected, other players can continue to play without inconvenience.
- The game is very sensitive to the host computer activity. MiMaze is installed in the user space of the computer. It can consequently be de-scheduled to run higher priority tasks. If the waiting time is too long (more than 200ms), synchronization is lost, as well as game information (the synchronization algorithm discards packets that cannot be re-synchroni-

3. It is important to recall here that the game architecture is totally distributed (no server), that it uses multicast (RTP/UDP/IP), that transmission is totally unreliable (no error recovery mechanism), and that there is no Dead-Reckoning mechanism used to recover lost or late ADUs.

zed). The visible consequence was that different participants could have different scoring information (the scores being computed locally, cannot be incremented when the participant is not-synchronized).

- The bucket algorithm was efficient, except in the case of potential collision, e.g., when two avatars were “too” close to each other . The visible effect of a collision situation to the participant was that his avatar was unable to kill another “too close” avatar.

3.4.2 Network parameters

We first investigate delay distribution and losses.

Delay distribution and clock

MiMaze uses the sender transmission time and the receiver arrival time of OS ADUs to compute the network delay between any couple of participants. Consequently, each game entity maintains a clock offset information for each session participant. When the delay information is not coherent (when network delay variation between two ADUs is too large), the clock offset is not computed from the time-stamps, but is interpolated from previous offset values. Interpolation is used until new "stable" network delays can be observed.

Hence, the accuracy of the delays computed by MiMaze (most of the synchronization accuracy lays on the accuracy of the delay) is strongly correlated to the accuracy of the clock offsets and to the efficiency of NTP.

For this experiment we were using NTP stratum 3 on most of the computers, and NTP stratum 2 on some of them. Figure 6 shows the delay distribution measured (between hugo and droopy) on the Mbone during the experiment. We have chosen to present the distribution curves only, as the delay vs. time figure would be more difficult to analyze.

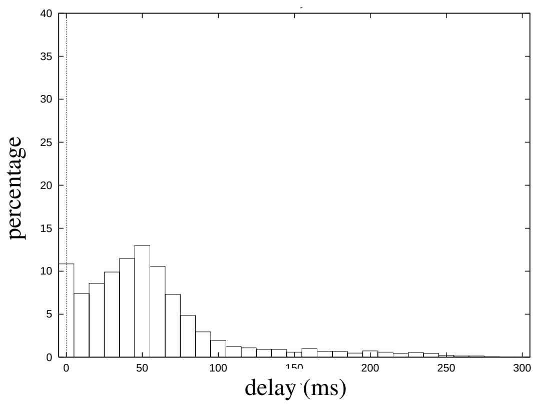

Figure 6. Network delay distribution observed by MiMaze on the Mbone.

We observe in figure 6 that the delay distribution is quite large. The calculation of the mean (μ) and of the standard deviation (σ) gives more accurate information on the measure. For figure 6, σ is 50.44ms (with μ=55.47ms). The mean is very close to the average delay measured during the experiments (55 ms as shown figure 5). But the standard deviation is too large. Using NTP stratum 1 and 2 synchronized on a global clock network (such as GPS) should reduce the standard deviation to less than 10ms [7].

Losses

We only analyze here losses due to the network. ADU that are discarded because they cannot be re-synchronized can be considered as lost; we will analyze losses due to late ADUs in the following section. Figure 7left shows the percentage of OS ADUs lost during a 200 second period of time on the Mbone. Figures 7right gives the loss distribution, i.e. the distribution of the percentage of ADUs lost during a bucket interval. The mean loss rate is 7% on the Mbone (nationwide). The distribution of loss shows that there is no loss in 75% of the buckets.

Losses have no visible effect on the game performance because the ADU emission frequency is high compared to loss rate and to human perception. We will try to analyze in the discussion section the correlation between loss, emission frequency, and game consistency. But it can be noticed that loss rate can significantly increase in case of Mbone congestion (see next section).

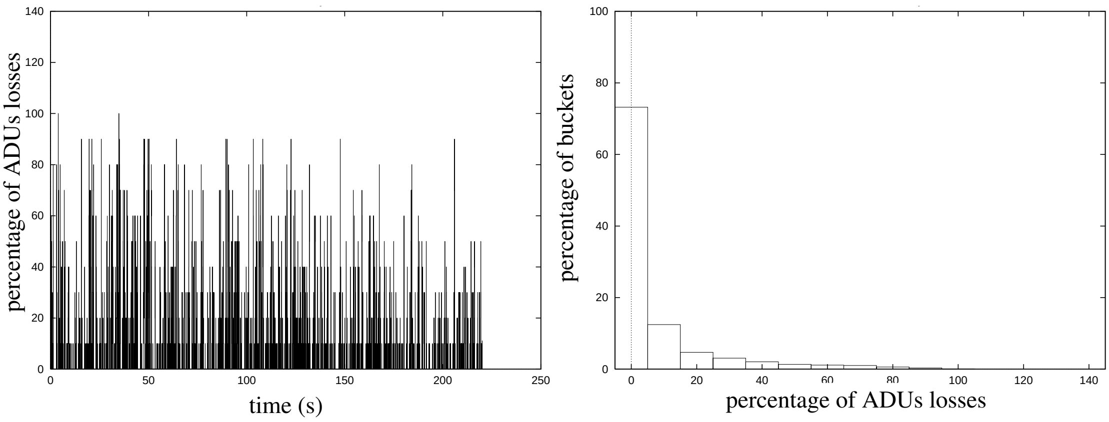

Figure 7. Percentage (left) and distribution (right) of lost ADUs on the Mbone.

3.4.3 Scalability

We have not been able to study scalability in depth for this first evaluation of MiMaze. This is due to the reduced number of participants (not more than 25). There are two possible causes to reduce scalability:

- The Mbone bandwidth being limited to 500 kilo-bit per second, and the size of an OS ADU being approximately 50 bytes, MiMaze would saturate the Mbone for more than 50 participants (each participant transmits 10 kilo-bit per second).
- The computation of the global sate takes approximately *(5 + N) ms*, where *N* is the number of participants. 40ms between two buckets gives plenty of type to compute the global state and CPU congestion has not been a problem in these experiments (it would be for more than 35 participants).

We have consequently decided to present the following results independently of the number of participants (percentage of ADUs).

3.4.4 Bucket synchronization efficiency

To evaluate the efficiency of the synchronization mechanism, we analyze the percentage of ADUs that were not re-synchronized. Figure 8 gives the percentage[4] of OS ADUs that are not re-synchronized by the bucket algorithm because they reached their destination (hugo) after more than 100ms. ADUthat were lost or corrupted do not appear in these figure (they have been analyzed in section 3.4.2). Here again, the same type of observations can be made.

The analysis of the late ADUs distribution (figure 8right) shows that un-synchronized (late) ADUs occur in less than 15% of the buckets, or that in more than 85 % of the buckets, there is no late ADU. This result is coherent with the average delay and standard deviation observed in

4. The maximum should be 100%. Due to delay evaluation inaccuracy, it sometimes goes up to 140%, which can be explained by more than one ADU received from a participant in the same bucket. This phenomenon has no effect on the game as MiMaze keeps only the most recent information to compute the global state.

the previous section. A reduction on the delay standard deviation would also have a significant influence on the proportion of late ADUs (see section 3.5).

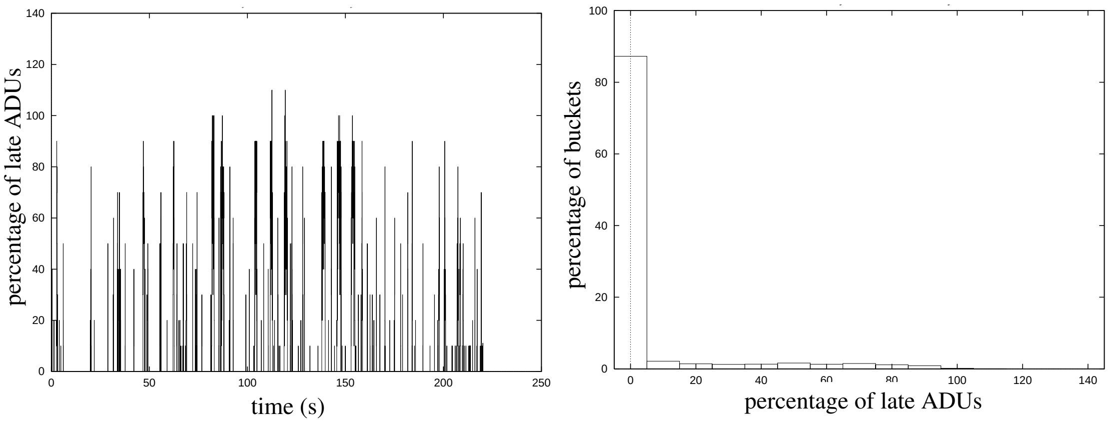

Figure 8. Percentage (left) and distribution (right) of un-synchronized ADUs on the Mbone.

3.4.5 Consistency

We have chosen two different metrics to analyze the consistency of the game. First, we have measured the number of new ADUs that are used by each entity to compute its local evaluation of the global state. With the current game setting, there should be one new ADU for each participant available in a bucket at the time this bucket is processed. If a new ADU is missing, it is replaced by the last ADU received from this participant.

Figure 9 shows that the global game state is most often computed with 70% of new ADUs (μ=67%). The game state is computed with 100% of new ADUs in only 10% of the cases. The standard deviation is again high (σ=27.05ms).

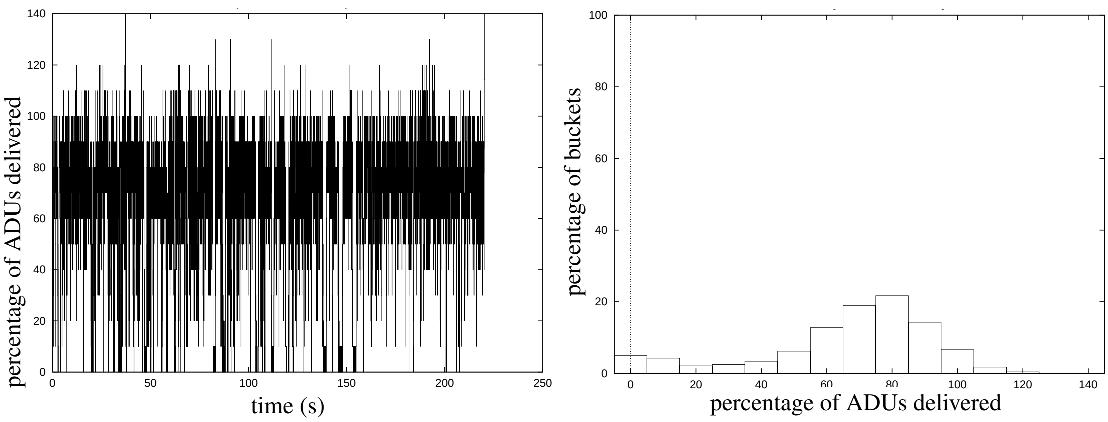

Figure 9. Percentage (left) and distribution (right) of re-synchronized ADUs on the Mbone.

It should be noticed that approximately 5% of the buckets are computed with no new ADU at all (figure 9right). Because of the high bucket frequency and of the human perception, the game participants did not see the resulting inconsistencies.

The second metric compares the global state computed by a participant with the reference state computed by the monitoring tool (and visualized by the 0 line on the vertical axis). In a perfect scenario (or with a server based architecture), all states would be on the reference line. This is obviously not the case for MiMaze.

Figure 10 shows the position of the global state as computed by hugo to the reference state (which is the same for all participants). The vertical axis corresponds to a "drift function" that is calculated as follows: $D = \Sigma_i [(X_{true} - X_i)^2 + (\alpha_{true} - \alpha_i)]$. Where $(X_{true} - X_i)$ is the distance between the real position of an avatar (given by the reference state) and its position computed by participant i; α is the orientation of the avatar in degrees.

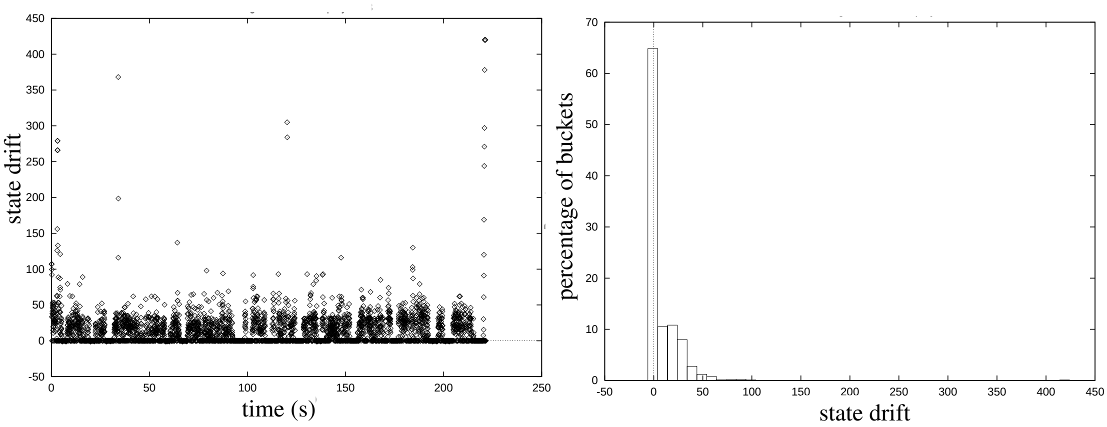

Figure 10. Comparison of the reference state to the global state as observed by Hugo, 43

A unit of 1 on the vertical axis can consequently correspond to a one degree error in the avatar orientation, or to a 1 unit of distance error (a distance unit corresponds to a 40ms constant speed move). A 2 distance units error correspond to 4 units of the drift function, and 3 distance units error correspond to 9 units of the drift function. In other word, 45 units can correspond to:

- an orientation error of 45 degres on a single avatar, or
- 5 avatars having a 3 distance unit error (or 120 ms).

Despite the results observed with the previous metric that indicated that only 10% of the buckets are computed with 100% of new ADUs, we can see on figure 10right that hugo has computed the correct reference state (e.g., has a drift of 0) for more than 65% of the buckets. There seems to be a correlation between the average percentage of delivered ADUs (67% figure 9) and the 65% of buckets that compute the reference state (more experimentation is needed to confirm this observation; but it is credible in our experimental conditions).

The state drift is also less than 50 units in almost all the buckets, and it is less than 20 units in 85% of the buckets. 50 units correspond approximately to an error of 120 ms (3 distance units) on 30% of the participants (from figure 9). This cannot be considered as an important error in the case of human interaction.

Another major observation on figure 10 is that the global state does not diverge, and that an entity will systematically comme back to the reference state, even if it has computed a wrong state for a while.

3.5 Discussion

The most important problem is to render the real-timeliness and the consistency of the game for all participants, whatever their network delay with the other participants is. With the current game and network setting, we have shown that for a human controlled application, there is no need for reliable transmission, and that the game is “consistent” in more than 65% of the time. Therefore, some improvement can be made so that the game consistency should significantly increase.

Network synchronization

Our results have been obtained with relatively loose global clock synchronization (NTP stratum 2 and 3). We expect to reduce the standard deviation on delays to less than 10ms to improve significantly the consistency of the game by using NTP stratum 1 and 2 synchronized on a global clock network (such as GPS).

The reduction of the standard deviation would have a direct influence on the average number of un-synchronized ADUs. Consider our experimetal settings (mean delay of 50ms and standard deviation of 50ms). Assuming normally distributed network delays, approximately 85% of the delays are shorter than 100ms, and 15% are longer than 100ms. For a mean delay equal to 80ms, more than 30% of the ADUs would be discarded. With a standard deviation limited to 10ms, there would be no late ADU in our experimental conditions; and there would be only a few percent of late ADUs with an 80ms average network delay. Limiting the standard deviation to 10ms would significantly improve MiMaze performance.

ADU transmission frequency and bucket frequency

The bucket frequency is mostly independent of the ADU transmission frequency, and should be chosen considering local parameters only, such as the global state computing time (one bucket is processed anytime a new game state is displayed) and the real-time properties of the application. Slowing down the bucket frequency will not help to maintain the real-time properties of the game as long as the global state computing time is less than the interval of time between two buckets.

On the other hand, reducing the frequency of ADU transmission would have a significant impact on the MiMaze consistency. For example, a division of the frequency by 2 (transmission of one ADU every 80ms) would reduce by 50% the number of new ADUs available in buckets. With the same network characteristics, we can expect that 10% of the buckets would be computed with no new ADU, and that the average percentage of ADU per bucket would be 33% instead of 67%.

So, by modifying its ADU tranmission frequency, an entity modifies the level of redundancy of the information it sends to the other entities. A high frequency will minize the impact of losses and increase the network traffic. A low frequency will reduce the consistency and the network traffic. Consequently, the ADU emission frequency has to be chosen depending on the loss rate (due to the network and to late ADUs), and possibily on CPU congestion.

Addition of Dead-Reckoning

Dead-Reckoning (DR) is mandatory in a distributed interactive application such as MiMaze. DR will improve the efficiency of the game in various aspects:

- Lost ADUs can be replaced by the Dead-Reckoned data computed from the last ADU received for the same participant. In MiMaze, that is approximately 7% of ADUs that would have been "replaced" by DR.
- Re-synchronization in case of network delays higher than 100ms. ADUs that are received late (with more than 100ms network delay) will be dead-reckoned to fill the next bucket to be processed. In the current implementation, these ADUs are simply discarded.
- Reduce ADU transmission frequency. DR can help to reduce the network load, depending on the application characteristics and on the network situation. The drawback will be to increase the CPU load of receiving entities.

We can give an approximate evaluation of the game consistency considering that DR would have permitted to deliver each ADU. With DR, 100% of the buckets should be computed with new ADUs. If the trajectory of the avatar doesnt change, the dead-reckoned information corresponds to the real position of the avatar and all participants would see the reference state. When there is a trajectory change, the dead-reckoned information is wrong and game consistency is not improved. Considering direction changes happen in less than 10% of the time (which is high in case of human interaction), inconsistency would only happen in 5% on the buckets.

4.0 Conclusion

The main contribution of this work is to show that with a multicast communication architecture and with a simple synchronization mechanism (the bucket mechanism), a fully distributed interactive application can work properly on the Internet. We have shown that reliable transmission is not mandatory in the case of human interaction. Moreover, we have shown that Mimaze consistency can be significantly improved with simple mechanisms.

We were not able to analyze scalability with this experimentation of MiMaze. This is mostly due to that:

- MiMaze ADUs are short,
- MiMaze CPU requirements are small, and
- the number of participants was limited.

But simple calculations shown that the scalabity will become a major problem as soon as ADU size or global state computing time will increase (from this point of view, MiMaze is relatively unrealistic as 52 bytes is close to the minimum size of RTP/UDP/IP headers).

To our knowledge, MiMaze is the first multi-player game to be designed with a totally distributed architecture, e.g., serverless. Another major contribution of this work is to prove the efficiency of distributed (or multipeer) architectures. Today, the most popular architecture in distributed systems is client-server. We have shown that a distributed approach provides a good level of performance with potential scalability and better real-time properties.

A new release of Mimaze is being designed. The main features of this new implementation are:

- Provide error recovery by Dead-Reckoning. We expect DR used on the received ADUs will increase the reliability, making possible to deliver lost or late ADUs.
- Resolve collision by source anticipation.
- Implement a new ADU type (called EXception ADU) and a new ADU structure for an optimized network transmission and control.

Other improvement such as group management is kept for further studies. We need first, with experiments like this one and with modelling, to understand how to manage efficiently groups from an application perspective in order to minimize cost related to group management. But group management will be a major feature in interactive application scalability and consistency.

We also keep for future work the extension of MiMaze with synchronized 3D VRML objects, video scenes (MPEG4), and 3D audio (that need a higher clock resolution [19]).

5.0 References

[1] IEEE Standard for Distributed Interactive Simulation -- Application Protocols (IEEE Std 1278.1-1995). IEEE Computer Society. 1995.

[2] IEEE Standard for Distributed Interactive Simulation -- Communication Services and Profiles (IEEE Std 1278.2-1995). IEEE Computer Society. 1995.

[3] S. Seidensticker and W. Garth Smith and M. Myjak. "Scenarios and Appropriate Protocols for Distributed Interactive Simulation". Working Internet Draft <draft-ietf-lsma-scenarios-01.txt>. March 1997.

[4] J. M. Pullen and M. Myjak and C. Bouwens. "Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environment". Working Internet Draft <draft-ietf-lsma-limitations-01.txt>, March 1997.

[5] J-C. Bolot, A. Vega Garcia, "Control mechanisms for packet audio in the Internet", Proceedings of IEEE Infocom '96, San Fransisco, pp. 232-239, April 1996

[6] D. L. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC-1305, March 1992.

[7] A. Cox, E. Luiijf, R. van Kampen, R. Ripley. "Time Synchronization Experiments". Proceedings of the 14th DIS workshop (dis-96-14-175). Spring 1996.

[8] J. Czeranski, H-U. Kiel. "Softwarepraktikum Netzwerkprogrammierung unter Unix am Beispiel des Spiels", 1993/94, http://www.tu-clausthal.de/student/iMaze/.

[9] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. "RTP: A Transport Protocol for Real-Time Applications", RFC-1889, January 1996.

[10] R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, "Adaptive playout mechanisms for packetized audio applications in wide-area networks", Proceedings of Infocom '94, Toronto, Canada, pp. 680-688, April 1994.

[11] .H. Eriksson. "MBONE: The Multicast Backbone". Communication of the ACM. Vol. 37. pp. 54-60. August 1994.

[12] A. Goscinsky. "Distributed Operating System, The Logical Design". Addision-Wesley publishing company. 1991.

[13] L. Gautier and C. Diot. "MiMaze, a Multiser Game over the Internet". INRIA Research Report 3248. INRIA Sophia Antipolis (France). September 1997.

[14] E. Berglund and D. R. Cheriton. "Amaze: a multiplayer computer game". IEEE Software. 2(3):30-39, May 1985.

[15] CRYO Interactive. "le deuxieme monde". http ://www.cryointeractive.com/. 1997.

[16] S. Deering. "Host Extensions for IP Multicasting". RFC 1112. 17. August 1989.

[17] D. B. Anderson, J. W. Barrus, D. C. Brogan, M. A. Casey, S. G. McKeown, I. B. Sterns, R. C. Waters, and W. S. Yerazunis. " Diamond Park and Spline: A Virtual Reality System with 3D animation, Spoken Interaction, and Runtime Modifiability". MERL report TR96-02a. 1996.

[18] J. W. Barrus, R. C. Waters and D. B. Anderson. " Locales and Beacons: Efficient and precise Support for Large Scale Multiuser Virtual Environments". IEEE Virtual reality Annual International Symposium. Santa Clara (CA). March 1996.

[19] R. C. Waters. "Time synchronization in Spline". MERL report TR96-09. April 1996.