A Distributed Architecture for Multiplayer Interactive Applications on the Internet

Christophe DIOT[1] and Laurent GAUTIER[2]

[1]SPRINT ATL, 1 Adrian Court, Burlingame, CA 94030, USA. cdiot@sprintlabs.com

[2]INRIA, 2004 route des Lucioles, 06902 Sophia Antipolis, France. lgautier@sophia.inria.fr

Abstract: *This paper describes the design, implementation, and evaluation of MiMaze, a distributed multiplayer game on the Internet, and, more precisely, on the design of dedicated transmission control mechanisms. MiMaze is implemented on a completely distributed communication architecture based on the IP multicast protocol suite (RTP/UDP/IP). This is the first work to analyze a distributed interactive game on the multicast Internet. The major element of the MiMaze architecture is a distributed synchronization mechanism that guarantees the consistency of the game regardless of network delay. This paper provides an evaluation of the MiMaze game on the MBone, and discusses approaches to monitor and evaluate this new type of application. The main contribution of this work is to show, based on an example, the feasibility of this new family of applications on a best-effort network. It is shown that real-time interactivity can be maintained, provided that some level of inconsistency can be tolerated by the application. This work also highlights the role of multicast as an enabling technology for a real-time Internet.*

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

1.0 Introduction

This article describes the design and the evaluation of a multiplayer game over the Internet. MiMaze [22] is a distributed (i.e. serverless) game that uses an unreliable communication system which is based on RTP [9] over UDP/IP multicast [16]. Multiplayer games are representative of the new generation of Distributed Interactive Applications (DIAs) which also includes Distributed Interactive Simulation (DIS), digital battlefields, Air Traffic Control (ATC), cooperative tools, and home interactive applications[3][4]. These applications are expected to represent a major share of the Internet traffic within in the next 5 years.

We have chosen a distributed architecture, which relies on multicast communication, instead of a client/server architecture based on TCP, in order to increase the scalability of the application. As a consequence, we do not use a central entity (or server) to compute the game state (see section 2.1). On the contrary, each participant computes its own view of the game state, increasing the chance of inconsistency[1]. It is therefore the responsibility of each game entity to provide synchronization facilities to cope with distributed state computation and heterogeneous Internet delays. A server exists in MiMaze, but its use is limited to non-real-time tasks (such as a new player joining an ongoing session and session management issues in general).

This distributed architecture can be criticized because multicast is not widely deployed on the Internet (it is experimental with the MBone). Nevertheless, group communication is currently one of the most active research domains in the Internet community. Access via multicast tunneling is available for experimental purpose, and some ISPs already offer multicast in their Internet services. Moreover, the experimental analysis of this new family of applications gives us an

1. Inconsistency occurs when participants have a different view of the game global state. Inconsistency is inherent to distributed networked applications (with heterogeneous delays).

opportunity to learn which multicast service definition best suits this new family of applications.

The central theme of this paper is the evaluation of the synchronization mechanism in the MiMaze game, referred to as bucket synchronization. The synchronization mechanism is the minimum functionality required for a distributed game. Without a synchronization mechanism, the real-time requirements of an interactive game cannot be satisfied. In MiMaze, we require that information must be delivered to its destinations in less than 100ms. We have implemented a very simple dead reckoning algorithm to recover lost or late packets. This mechanism replaces classic ARQ-based error recovery which is not applicable in this context (acknowledgment-based techniques introduce unacceptable delays).

This paper 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 performance measurements on the MBone, and we describe the monitoring tool used to resynchronize the distributed traces. The global clock problem is addressed. The performance results give us an opportunity to discuss what metric could be used to provide a realistic evaluation of the performance of a DIA. We also propose new mechanisms to improve consistency and scalability in DIAs. We conclude with a discussion of the next MiMaze design steps, and with the extension of the game to a more realistic interactive environment (including 3D graphics and virtual worlds).

The main contribution of this work is to show, based on an example, the feasibility of a new family of applications on a best-effort network. We are conscious that MiMaze is a simple application and that the following results are difficult to generalize. However, analyzing the behavior of a simple application on the Internet is an important first step in understanding how DIAs can be deployed on the Internet. Consequently, we do not try to generalize our results; instead, we try to analyze generalizable observations.

2.0 Description of MiMaze

The characteristics of distributed games are very similar to those of DIS applications [1][2]. The main difference between DIS applications and MiMaze is that the CPU requirement in MiMaze is low. One reason for choosing such a "simple" game (in term of rules and graphics) 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, be processed and be displayed to any other participant within the shortest possible delay[2].If the network delay is excessive, the received action (encoded in an Application Data Unit, or ADU) is late and typically cannot be used by the application.
- Participants can join and leave a MiMaze session dynamically. In this context, the IP multicast model [16] is particularly convenient.
- The system architecture is distributed. This is justified, in the case of DIS, by robustness requirements, and by a potentially better scalability (see section 2.1.1).

An important property of DIAs is that the real-time behavior of game objects (also called avatars) is “continuous.” By continuous, we mean that the behavior of avatar *X* at time *n+1* can be extrapolated from its behavior at time *n*. As a consequence, an avatar description lost on the network can be easily recovered from a previous description of the avatar, given that updates on the avatar position are sent often enough. This allows us to use an unreliable communication system based on RTP and UDP, where the effects of lost or overly delayed packets are minimi-

2. the delay range to preserve real-time interaction between participants is 40ms to 200ms, depending on the application characteristics[15][17][5]. In MiMaze, we have chosen 100ms.

zed by the natural redundancy of the interactive data, and by the use of a simple dead reckoning algorithm (see section 2.3).

The problem of number of participants is not addressed in this paper. At this stage of the project, we have no idea of what a "large" session is for an interactive game. We have consequently decided to have a single multicast group in MiMaze until initial experiments tell us more about the feasibility of DIAs on the Internet.

2.1 MiMaze design characteristics

MiMaze is inspired by iMaze [8], a 2-dimensional "Pacman" game in which each player has a 3D representation of its view of the game (see screen shot Figure 1). Avatars (Pacmen) move 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.

2.1.1 MiMaze distributed architecture

To our knowledge, MiMaze [13] is the only game with *a fully distributed architecture using IP multicast* (see Figure 2). A server is only used when a new entity joins a session, to learn the session group address and to download the maze.

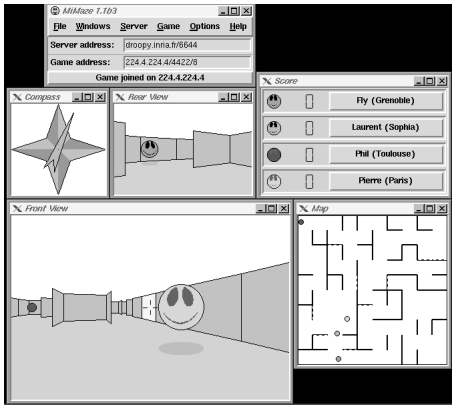

Figure 1. Screen shot showing MiMaze participant's local view: maze view from the top, session information (multicast address, session properties, etc.), participant information (scoring, participants' names), front and rear views from the participant's avatar (3D).

Distributed architectures have many advantages over 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 has locally all necessary information to compute the state of the game at any time. Since a serverless architecture does not require the availability of a server which supports currently active applications, a serverless architecture greatly simplifies the deployment of a game by an application provider. The quality observed by a participant only relies on the network properties (including multicast support) and on the participant equipment.
- Scalability. We have identified two factors which limit the scalability of centralized architectures:

  - Since all data converge at the centralized server, the server becomes a bottleneck. Since a centralized server must collect all participants' data and serve all participants of a game, the frequency at which the game state can be computed slows down once the CPU of the server becomes saturated. In the worst case, overload of the server CPU leads to a loss of interactivity of the game[3].

- With a server architecture, the amount of data transferred on the network is in the best case equal to the amount of data transferred with a distributed architecture. It is higher when multicast is not used, or when TCP is used to collect data.

- Minimum delays. In a centralized architecture, information reaches its destination through the server. Depending on network topology and on the routing tree structure, this can increase the network delay up to two times the delays in a distributed architecture. In a distributed architecture, information crosses the network only once to reach its final destinations.

On the other hand, centralized architectures do have some advantages. First, since all participants in the game receive the same game state from the server, global consistency is guaranteed. Moreover, the server introduces a natural synchronization among players. Another important feature of centralized architectures is that they allow game companies to easily charge players based on the duration of their participation.

Another major 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 not necessarily an authority to identify potential cheaters. In a centralized architecture, however, all information flows to the server, which can 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].

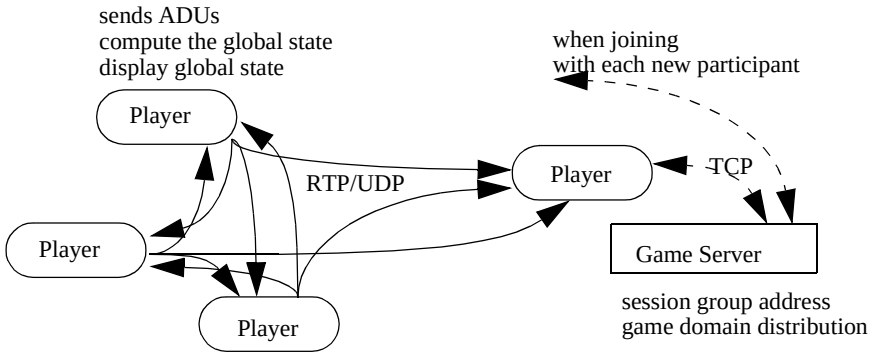

Figure 2. MiMaze communication architecture showing the session management server used by participant when joining a session, and the fully distributed communication once the game has started.

To summarize, *choosing a distributed architecture improves the real-time properties of the application, at the cost of consistency. The motivation of this work is consequently to show that the real-time properties of a DIA can be preserved on the Internet, despite long and heterogeneous delays, provided that an acceptable level of inconsistency can be tolerated in the application. This work shows that a simple distributed synchronization mechanism and a global clock infrastructure (such as NTP) can help to control the consistency*.

2.1.2 Data structures

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, and it may be unrealistic to use this protocol over the Internet [3][4].

The DIS standard defines more than 25 different packet types (called PDU's or Protocol Data Units) [1][2]. Most of the DIS packet types are relevant to distributed games. The most fre-

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

quently used PDU is the Entity State PDU (ESPDU) which carries state information describing the game objects. A specific PDU type has also been defined for "exceptions" such as collisions, fire (shooting), detonation (bullets or projectiles), and also for logistics and exercise control. A reference to a freely available implementation of DIS can be found at [23].

To minimize the network traffic generated by MiMaze, we use only one type of packet which is called an ADU (Application Data Unit). The MiMaze ADU is very similar to DIS's ESPDU. The size of an ADU is up to 52 bytes, including 8 bytes for the RTP header, 8 bytes for the UDP header, 20 bytes for the IP header, and up to 16 bytes for MiMaze application payload. MiMaze ADUs contain a description of the local state of an avatar, consisting of the local position in the game (x position, y position, angle) and the displacement vector (speed, angular speed) of the avatar and of the projectile emitted by the avatar.

A MiMaze entity sends ADUs on a periodic basis. In the current version of MiMaze, the ADU transmission frequency is 25 times per second. For game experts, this value corresponds to a very fast reaction speed of a human player, i.e. 40ms.

2.1.3 Related work

Amaze can be considered as MiMaze's ancestor. Amaze was designed by Berglund and Cheriton in 1984 [14] to be played on a LAN, using point-to-point communication. MiMaze and Amaze both have a distributed architectures but manage state differently. Amaze transmits the game state on the network, and maintains replicated copies of the game state.

Distributed games on the Internet are now a real market for companies. Microsoft, BT, Intel, Sony have their own projects for on-line distributed games or shared virtual worlds. There are also small companies that enhance multiplayer games with more sophisticated transmission infrastructures (see for example [15]). But all commercial on-line games available to date still use client/server architectures based on TCP transmission, simply because this is the only technology fully available and stable on the Internet today. Additionally, closely related works include the following:

- 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 local flow synchronization. But there is no distributed synchronization mechanism to deal with heterogeneous network delays.
- The PARADISE project [21] at Stanford University aims to architect and build a large-scale internetworked simulation environment that supports multi-player interactive, 3D simulations running over a wide-area network. This project has produced very interesting results on group communication, dead reckoning, entity aggregation, and collision detection. Our work here differs from [20] in that our goal is to evaluate the MiMaze architecture from a system standpoint, where the inter-related issues of bucket-synchronization, dead reckoning, and network impairments such as loss and delay are inextricably linked. In contrast, the work in [20] is aimed primarily at aggregation (not considered here) and specific dead reckoning algorithms. It worth noting that dead reckoning is used in [20] primarily to decrease the frequency of state transmission and smooth trajectories between state updates.

2.2 The Bucket Synchronization Mechanism

In a serverless architecture, synchronization must be introduced to make sure that the state displayed by each entity is consistent, i.e. that:

- All ADUs issued at the same time (by various game entities) are computed together to evaluate the state of the game.
- All the session entities display to their own player the same game state simultaneously.

In MiMaze, time is divided into fixed length periods and a bucket is associated with each period. All ADUs received by a player that were issued by senders during a given period are

stored by the receiver in the bucket corresponding to that interval. At the end of every bucket interval, all ADUs in that bucket are used by the entity to compute its local view of the global state. Buckets are computed 100 ms after the end of the sampling period during which ADUs have been issued (100ms is the playout delay[4]). In other words, to compute a new global state, an entity computes all the ADUs available in the "current" bucket.

For example, in Figure 3, without synchronization, the ADU issued by player X at t_3 would be processed together with the ADU issued at t_1 by player Y (but received at t_2 by X, which is in the same state processing interval as 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 Y.

We did not implement a specific mechanism to perform the another aspect of synchronization (inter-entity synchronization). This is provided naturally by the bucket algorithm given that the time interval between two buckets is small enough.

When an ADU is received with a transmission delay which is more than 100ms, its destination bucket has already been computed. But the late ADU is still stored in this bucket. It will be used by the dead reckoning algorithm to eventually replace a missing ADU when computing future buckets (see section 2.3).

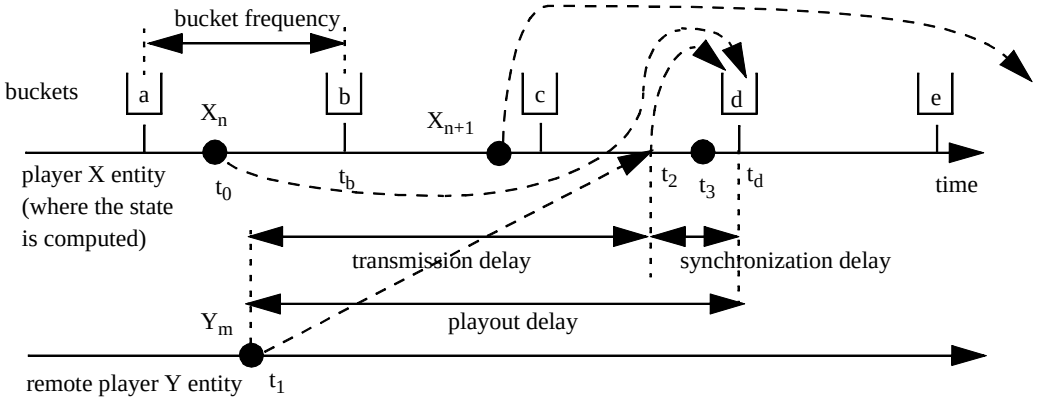

Figure 3. The bucket synchronization mechanism. The horizontal axis represents time. Player X (upper line) is the player where the bucket algorithm is observed. The playout delay and the bucket frequency are static parameters which are defined independently of the network properties.

2.2.1 Bucket frequency

The bucket frequency defines the rate at which a new game state is computed and displayed. Since human vision perceives smooth motion when the frame rate exceeds 10-15 frames per second, we have chosen to compute 25 buckets per second. The bucket frequency is a receiver application parameter that should not be influenced by network parameters.

With MiMaze's current settings, the ADU transmission frequency being equal to the bucket frequency[5], there should be one new ADU per entity at the time a bucket is processed.

4. The bucket mechanism is similar to a playout buffer mechanism [10] used to reduce network jitter effects in packet audio information.

5. It is a coincidence that these two parameters are equal.

2.2.2 Global clock mechanism

The bucket synchronization mechanism uses a global clock system to evaluate the delay between participating entities. A discussion on the properties of the global clock system can be found in [19]. 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 (strata) of NTP servers 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 manipulate both clock representations.
- NTP does not provide a reference clock signal and each participant has to compute an offset for every other participant in a game session.
- NTP makes the assumption that network links used to calculate timing delays are symmetric, which is often not the case in the Internet.

In our current implementation, in order to increase the precision of NTP under stratum 2 and 3, we use both NTP and an NTP-like mechanism which we have specifically designed to compute clock offsets (see [13]).

2.3 Dead reckoning

To deliver a complete view of the game, the bucket algorithm requires at least one ADU per participant to be available in each bucket. However an ADU can be missing for various reasons. It may have been lost by the network or it may be late. Dead reckoning is used to replace missing ADUs.

For each missing ADU, the state computation algorithm goes back to the previous buckets, looking for the most recent ADU received for the missing avatar. Once found, this ADU is dead reckoned to evaluate the position where the avatar should "most probably" be at the current time. The accuracy of the evaluation depends on the dead reckoning algorithm used (there are many possible dead reckoning algorithms available), on the age of the ADU used, and on game characteristics.

Dead reckoning being not the purpose of this paper, we have implemented in MiMaze the simplest possible dead reckoning algorithm. When an avatar position (ADU) is missing at the time we compute a bucket, we simply replay the last known position of this avatar. We believe that dead reckoning is a major mechanism for error correction in DIAs, and more dead reckoning algorithms will be evaluated in future work. But since the principal goal of this work is to gain an understanding of the behavior of DIAs on the Internet, we intentionally decided to have a simple dead reckoning algorithm in order to analyze distributed synchronization with minimal side effects.

3.0 Performance evaluation

This section is organized in five parts. We begin with the description of the experimental settings. The monitoring tool designed to resynchronize the distributed traces is briefly presented. We then define the distributed game metric that we have chosen to evaluate MiMaze’s consistency (the notion of consistency is the one defined in the Introduction section). The evaluation section begins by providing an analysis of the behavior of MiMaze during an experimental session on the MBone. The influence of network parameters on the game consistency is analyzed. We conclude this section with a discussion of the experimental observations.

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 number of participants in experiments was varied in order to vary the loss rate. We make the assumption that all losses are due to network congestion. A total of 1600 traces of 15 to 20 minutes each were collected. The architecture of the experimental multicast tree is given in Figure 4. The delay values given in Figure 4 are average values, on the duration of a session, end-to-end network delays measured from host droopy. Note that average network delays are always less than 100ms. The computers participating in the experiment were SUN (SPARC 10, 20, ULTRA), DEC Alphas, and PCs.

3.2 Monitoring

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

- Network level data: for each received ADU, we collect the senders' identity, the transmission time-stamp, the reception time-stamp, and the sequence number.
- 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: network delays and clock offsets with respect to each participant. These data are used to resynchronize the traces.

The main difficulty in analyzing the collected traces is not the benchmark computation, but the trace resynchronization. Since the system is distributed, there is no absolute clock in a game session, and each participant timestamps its local traces. Since the global clock mechanism is imperfect, we used an algorithm based on least-squares [24] to resynchronize traces. However, this simple method to synchronize traces is only valid when the clocks are not drifting too fast from each other. Consequently, we decided to limit the analysis to those traces where NTP clock synchronization was successful. We therefore omitted time intervals where relative clock information was not interpretable [13].

Hence, the accuracy of the delays computed by MiMaze is strongly correlated to the ability of NTP to maintain a synchronized global clock signal among participants.

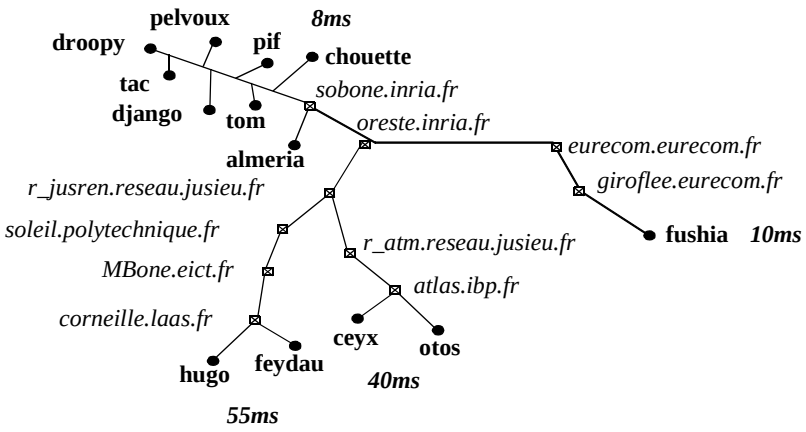

Figure 4. MBone architecture during the evaluation. Network delays are the averages measured from droopy (located at INRIA Sophia Antipolis). All locations are in France.

3.3 Distributed game metrics

A major difficulty when attempting to analyze distributed multiplayer games is the definition of meaningful performance metrics. These criteria must reflect how far the distributed game

behavior is from the perfect behavior that would have occurred if the game had been played with no network delay and no network loss. Some people expect a networked game to behave exactly as a non-networked solution, i.e., without any effects of network delay. Clearly, this is impossible to achieve. The network introduces impairments which are an integral part of the game, e.g., a projectile does not reach its target instantaneously because of its speed. Instead of trying to eliminate the impairments (in particular, network losses and delays), we try to minimize the impact of impairments on the game. The goal is to deliver the same information to each participant, even if this information is slightly different from what it would have been without network delay. Trying to compare a networked game session to the same session played with no network would be wrong, since all player actions are inevitably influenced by the presence of the network.

For evaluation purposes, we have chosen a game metric called "drift distance." The drift distance represents, in distance units, the absolute value of the distance between the position of an avatar as displayed by its local entity, and the position of the same avatar displayed by a remote entity. The game is consistent if the drift distance is zero. Nevertheless, a small drift does not mean that the game is inconsistent. Given that the avatar radius is 32 units and that the avatar constant speed is 32 units per 40 ms, then we assume (based on an informal analysis of player satisfaction) that errors of up to 50 units on a moving avatar are not significant (see next section).

3.4 Performance analysis

In this section, we first provide an informal analysis of MiMaze gaming sessions. Then we analyze network parameters during these sessions. Understanding the network parameters helps us to understand the game behavior, and how to improve its consistency. The final part of this section analyzes MiMaze consistency with regard to network parameters.

In order to keep the figures clear, we present performance observed between only two participants (droopy and hugo). We have verified that any pair of participants on the MBone have roughly the same behavior.

It is important to notice here that most of the results interpretations provided in this paper are conjectures. As discussed in the section on metrics, and due to the nature of the application, where human related factors have a major impact, one has to caution against a generalization of our results. Nevertheless the level of confidence we have in these conjectures is very high, since we have only provided an explanation for those phenomenons that we have observed almost systematically, on more than 1000 experimental traces (after filtering of traces not conforming to synchronization prerequisites).

3.4.1 Informal analysis

Since MiMaze was our first experience with a distributed game over the Internet, it is useful to first informally describe how players “perceived” the game. This informal evaluation is also useful to understand the limits of the consistency metric chosen.

- 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. The displacement of avatars in the labyrinth was smooth and regular at all participant locations. We observed from this informal evaluation that an error of 50 units, i.e. less than the diameter of an avatar, on a moving avatar position did not create any inconvenience to the participants. This result is important, and shows that giving preference to interactivity at the expense of consistency was a good choice.

- The behavior of each participant is independent. Due to the distributed nature of the architecture, few unexpected behaviors, such as participant disconnection (due to CPU load, network failure, trace memory saturation, etc.) or synchronization loss (NTP resynchronization or failure) occurred during the evaluation session. Such "failures" only annoyed the victim of the problem, and had no effects on the other participants.
- The MBone load and topology were stable during the experiments. We ~~also~~ noticed that the paths are not symmetrical on the MBone. This problem makes it difficult to compute network delays based on round trip times (including NTP). We consequently did not use computers connected to a NTP server by an asymmetric link.

3.4.2 Network parameters

Delay distribution and clock

Figure 5 shows the delay distribution measured (between hosts hugo and droopy) on the MBone during the experiment.

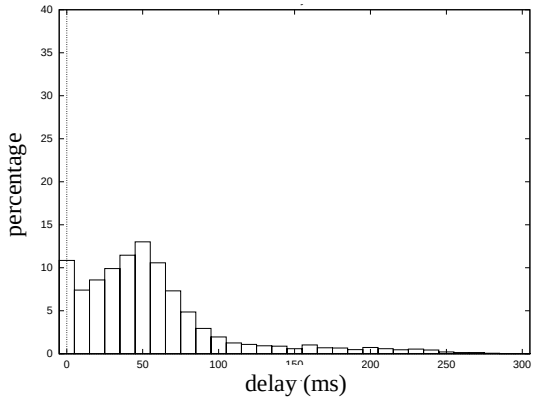

Figure 5. Network delay distribution observed by MiMaze on a region of the MBone (between hosts hugo and droopy).

We observe in Figure 5 that the delay distribution is long tailed. The standard deviation is 50.44 ms; the mean (55.47 ms) is very close to the average delay measured during the experiments (55 ms as shown in Figure 5).

The long-tailed distribution means that a significant part of the ADUs are late (i.e. they are not available at the time the bucket is computed). The dead reckoning algorithm used being quite simple, this should result in a significant drift increase for these late ADUs. Figure 5 indicates that more than 15% of the ADUs experience network delays higher than 100 ms. These ADUs will consequently not be used for on-time bucket computation. Reducing the standard deviation would reduce the proportion of late ADUs with immediate consequences for the game consistency.

To confirm the previous observation, Figure 6 gives two different observations of late ADUs. Late ADUs are ADUs that reached their destination after 150ms, and they result in a missing ADU at the time their respective bucket is computed. ADUs that were lost or corrupted are not considered (they will be analyzed in the next section). Figure 6a shows, over a time period of 220 seconds, the percentage of ADUs which have not arrived the time when the bucket is computed.

Figure 6b plots, on same time interval, the late ADUs distribution. Figure 6b shows that late ADUs occur in less than 15% of the buckets. In other words, in more than 85% of the buckets, there is no late ADU. This result is consistent with the observations from Figure 5.

A reduction in the delay standard deviation would have a significant influence on the proportion of late ADUs. Using NTP strata 1 and 2 synchronized on a global clock network (such as GPS) should reduce the standard deviation to less than 10ms [7], and would shorten the delay and late-ADUs distribution tail.

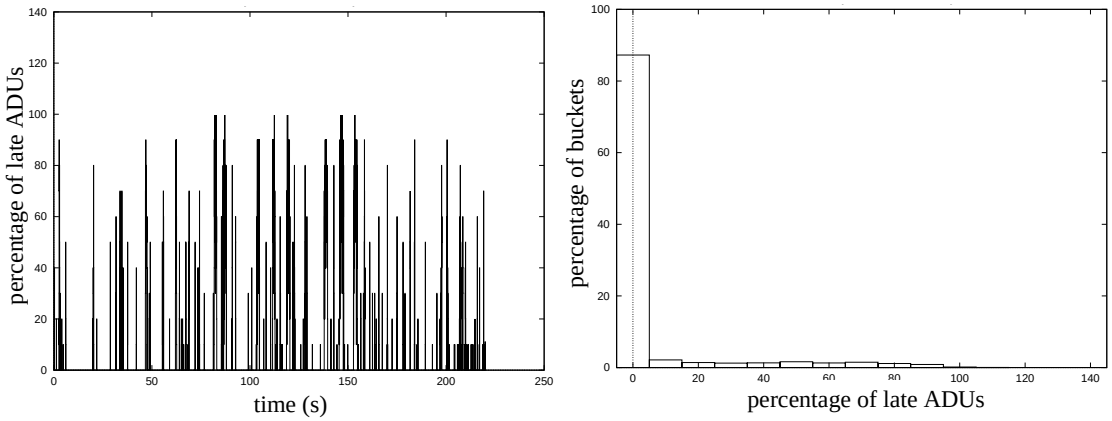

Figure 6. (a) Percentage and (b) distribution of late ADUs on the MBone (between hosts hugo and droopy).

Before addressing the problem of network losses, note that late ADUs have exactly the same application effect as losses on the synchronization algorithm (since corresponding data is not available in the bucket at the time of computation).

Losses

Figure 7 now gives two observations of lost ADUs. Only network losses are observed. Figure 7a shows the percentage of ADUs missing in each bucket due to losses in the Mbone. The observation period is the same as in Figure 6. Figures 7b gives the loss distribution, i.e. the distribution of ADUs lost during a bucket interval. The mean packet loss rate on the MBone was 7% during this experimental session. The loss distribution shows that there are no ADUs missing because of network loss in 75% of the buckets. We discuss the consequences of network losses on the game consistency in the next section.

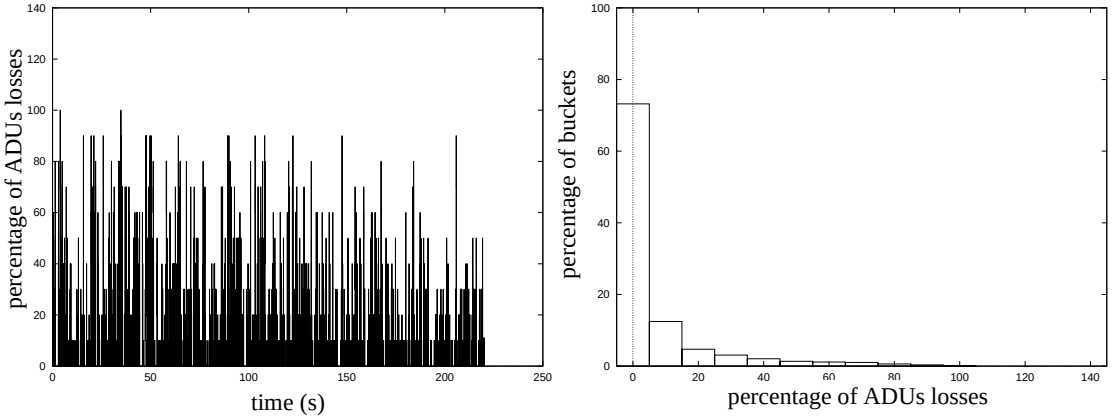

Figure 7. (a) Percentage and (b) distribution of lost ADUs on the MBone (between hosts hugo and droopy).

3.4.3 Bucket synchronization efficiency

Consistency of MiMaze

Recall that the drift is the distance between the actual position of an avatar, as seen by the local entity, and the position of the same avatar seen by a remote entity. In a perfect scenario or with a server based architecture, the overall game drift is zero. This is obviously not the case in MiMaze.

Figure 8 shows the drift distance between hugo (the remote entity) and droopy (the reference position). The drift distance appears on the vertical axis of Figure 8a, and on the horizontal axis of Figure 8b.

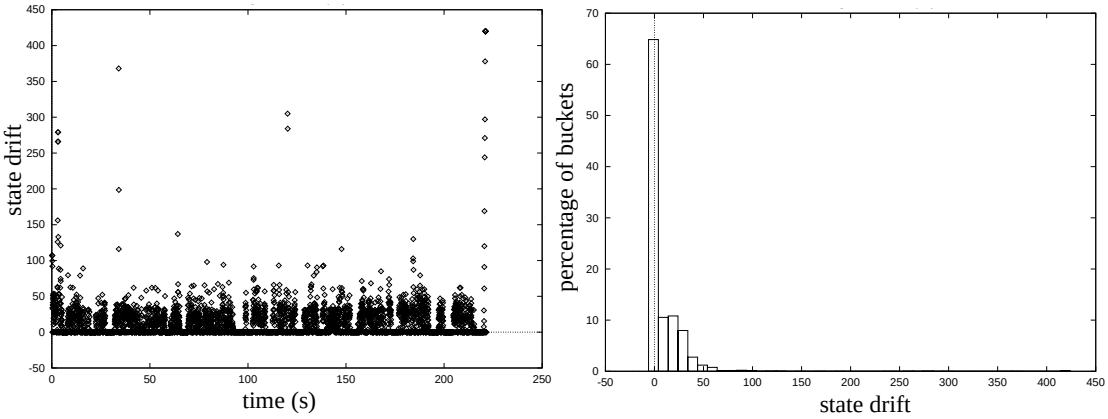

Figure 8. Evaluation of the drift distance between the local position of an avatar (measured at droopy) and its position observed by a remote entity (hugo); including (a) sample and (b) distribution.

The first and immediate observation is that the drift does not diverge, and that the remote entity will systematically come back to compute the real position, even if it has computed a wrong position ~~one~~ for a while. One reason is that an entity just needs to receive a single "on-time" ADU to reset to the correct position. Another reason is that network delays are less than 100ms. With higher delays, we conjecture the same behavior with a higher error. Recall that late ADUs which are late at the time a new bucket is computer are assumed to be lost.

The second observation is that the drift is 97% of the time less than 50 distance units. It is also less than 20 distance units in 85% of the buckets. This error did not result in inconvenience at the player level, since avatar diameter is 64 units. We consequently consider it as an insignificant error.

The drift distribution shows that in 65% of the cases, remote entities display the "exact" position of an avatar. This result was expected from observations on delay and losses (Figures 6 and 7 show that ADUs are missing in up to 40% of the buckets). The drift analysis reveals that only 35% of the buckets are computed with missing ADUs. More experimentation is needed to confirm this observation, but it is credible in our experimental conditions.

Impact of the synchronization mechanism

In order to understand how the distributed synchronization mechanism impacts MiMaze consistency, we have observed MiMaze's consistency with and without bucket synchronization. Figure 9 gives the gain on the drift distance, for drift distances observed. It is based on the distribution of drift distances observed during the experiments. A value of 1 on the vertical axis corresponds to no gain and 2 corresponds to a 100% gain.

We report the average loss rate and delay for each experiment, after filtering of non conforming traces. The session parameters are as described in section 3.1, except for the participants location: speedy is located at UCL (UK), pegase at LIP6 (Paris), elvis and droopy are at INRIA (Sophia Antipolis).

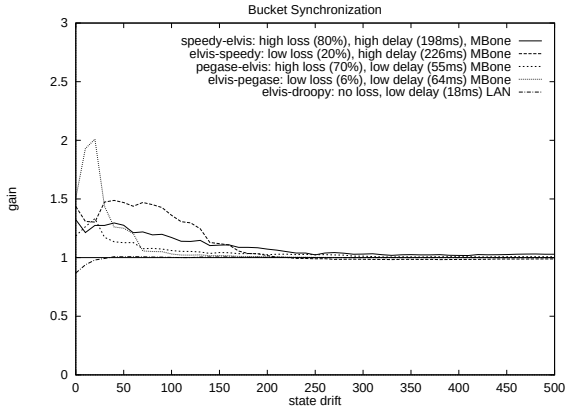

Figure 9. Consistency improvement (gain) with distributed synchronization (for various network loss/delay scenarios).

The first important result is that synchronization significantly reduces the drift for the most frequent drift values, i.e. between 1 and 150 distance units. In the case of large drifts corresponding to large sequences of losses, synchronization can not help to reduce the drift. This is confirmed by the second observation: synchronization reduces the drift for long delays (curves elvis-pegase and elvis-speedy have long delays and low losses); in the case of high losses with similar delays (on the pegase-elvis and speedy-elvis curves) the gain is smaller. Consequently, a new mechanism needs to be added to MiMaze in order to reduce inconsistencies due to losses. Such mechanisms will be discussed in the next section. We also observe in Figure 8 that for local communication (short delays, no loss), synchronization slightly reduces the consistency. We conjecture that this is due to the clock inaccuracy that causes some ADUs to be scheduled in an incorrect bucket (one in advance or one late) in the case of bucket synchronization.

3.5 Discussion

The main problem addressed in this work is the design of a communication infrastructure which that allows distributed interactive applications to be played on heterogeneous networks such as the Internet. We have shown that using a distributed architecture together with synchronization, it is possible to preserve the real-time interactive properties of the application, provided that some level of inconsistency can be accepted.

A major problem solved addressed in this study is the definition of a metric which is suitable to express the satisfaction of session participants in a realistic manner.

The performance analysis provided in the previous section shows that "only" 65% of buckets deliver the exact position of a given avatar. At the same time, players were very satisfied during the entire game session. This indicates that this type of application is more tolerant to network impairments than numerical observations would tend to show. Our finding is that 65% of consistent state evaluation is acceptable by MiMaze game users in the current network conditions. To better reflect player satisfaction, the following parameters should be involved in the consistency evaluation:

- The characteristics of the avatar, in term of speed, acceleration, size, shape can influence the acceptable error on the position of this avatar. E.g., a small error on the position of a very slow avatar can be dramatic. The same error on a very fast avatar would not be visible to the player (i.e. if this error is along the trajectory of the avatar).

- The game nature. An avatar moving in a 3D space with no terrain limits may be more difficult to dead reckon than a MiMaze avatar whose trajectory is constrained by the maze topology. It will consequently be easier to "extrapolate" or dead reckon the avatar trajectory in MiMaze.

Our approach was to deliver on-time a view of the game that is "close" to the real one, and that is "almost" the same at each participant location. This deliberate lack of precision (due to the unreliability of the architecture) allows more scalability, provides real-time interaction between participants, and does not alter participants' satisfaction. We believe that similar results could be observed with more complex games and shared virtual worlds.

The MiMaze design proves that for a simple human controlled application, there is no need for "100% reliable" transmission, and that the game is perceived as being "consistent" even if only 65% of the buckets are filled in time. A more complete analysis of game parameters, such as frequency of ADU transmissions, dead reckoning algorithm, etc., would show how much reliability can be "relaxed." In particular we expect, from the above observations, that dead reckoning is a key technique in increasing the consistency of the game within more demanding network environments, e.g. with high loss rates.

The dead reckoning algorithm we use in MiMaze is the simplest possible. Nevertheless it appears that due to the nature of the application, and due to the ADU transmission frequency (which is high in MiMaze), this algorithm is sufficient to minimize the drift distance in most of the cases. With faster avatars and different game environments, more sophisticated dead reckoning algorithms will be required.

Dead reckoning can help in various situations:

- Replacing lost or late ADUs. There are many dead reckoning algorithms defined in the DIS standard [1][2], some of them being very complex. Choosing (or designing) a dead reckoning algorithm is a complex task that is influenced by the game nature, by the ADU structure, and by the CPU load level.
- Smoothing the trajectory between two received ADUs. If one or more ADUs are missing between two received ADUs, dead reckoning can be run to interpolate the trajectory between the two received positions, thus displaying a smoother trajectory.
- Anticipating collisions between avatars. When a collision happens, it is most of the time too late to compute the consequences of the collision in real time (e.g. in case of digital battle field applications, where very complex changes can happen). Not anticipating collisions can also lead to game inconsistencies. dead reckoning can be used by the sending entities to anticipate potential future collisions.
- Reducing network congestion. By reducing the ADU transmission frequency, dead reckoning can help to reduce the network load, depending on the application characteristics and on the network situation. The drawback is increasing the CPU load at receiving entities and possibly increase inconsistency.

To concluding this discussion, it should be noticed here that bursty losses might significantly affect game consistency, and that a specific mechanism will have to be installed to prevent such losses.

4.0 Conclusion

MiMaze is a first (and necessary) step in understanding what changes a new generation of distributed interactive applications will introduce to the Internet, and how to deploy them safely on what is considered to be a non-real-time network.

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 inte-

ractive application can provide an acceptable level of consistency to distributed interactive applications on the Internet. We have shown that relaxing reliability constraints is possible, given that some level of inaccuracy is introduced in the global state computation. We have also identified the problem of defining a metric for the evaluation of the application consistency.

We were not able to analyze MiMaze's scalability with this experimentation. This is mostly because MiMaze is not a realistic application from a data complexity point of view: ADUs are short, CPU requirements are small, and the number of participants was limited.

Scalability will become a major problem as soon as ADU size or global state computing time increase.

Another major contribution of this work is a proof of feasibility of distributed multipeer architectures on heterogeneous best-effort networks. Today, the most popular architecture in distributed systems is a client-server architecture. We have shown that a distributed approach provides a good level of performance with potentially better scalability and better real-time capabilities.

The advent of distributed games and other DIAs will increase the need for a wide development of multicast on the Internet. Multicast is the only technique capable of reducing transmission delays in a multi-user session in point-to-point networks. More work on distributed games will tell us how to deploy multicast, and provide us with insights in need for specific group semantics, group management, pricing schemes, etc.

The MiMaze display has been recently modified to offer a real 3D view of the game terrain, with possible mapping of MPEG video on the maze walls [22]. This makes MiMaze a more realistic application for the evaluation of interactive virtual worlds. To allow further investigations, MiMaze needs to be improved with:

- Congestion control. The idea here will be to vary the sending entity ADU transmission frequency depending on network congestion feedback information (probably carried by RTCP).
- Avatar collision detection and anticipation. Local states will be dead reckoned at the source in order to anticipate their position in the close future. If a potential collision is detected, specific ADUs will be sent to other entities to announce the collision.
- Session management in subgroups of participants. Dividing a session into subgroups is necessary in order to increase the scalability and the consistency of applications [18].

Extension of MiMaze with synchronized 3D Virtual Reality Modeling Language (VRML) objects, video scenes (MPEG4), and 3D spatial audio (that need a higher clock resolution [19]) will also be a necessary step to increase the complexity of the application. MiMaze is available for evaluation on the MiMaze web site [22].

Acknowledgments

"Remerciements" are going to all volunteer players that made experimental sessions possible; to their Ph.D. advisors who did not say anything during these long game sessions (sometimes playing themselves!); and to Jim Kurose who helped to analyze the MiMaze behavior. Also large thanks to Don Brutzman who has supported our work since the beginning and helped improve this paper. Final thanks to Jorg Liebeherr that has been an extremely efficient and helpful editor.

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. "Design and evaluation of MiMaze, a Multiplayer Game on the Internet". IEEE Multimedia System Conference. Austin. June 28 - July 1, 1998.

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

[15] J. Rothshild, "Designing and Writing Multiplayer Games for the Internet: Technical Considerations", www.mpath.com/news/white_paper.html.

[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.

[20] S. Singhal, "Effective Remote Modeling in Large Scale Distributed Simulation and Visualization Environments," PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, August 1996.

[21] The PARADISE project web site. www-DSG.Stanford.EDU/paradise.html.

[22] L. Gautier, E. Lety, C. Diot. "The MiMaze web page". www.inria.fr/rodeo/MiMaze/.

[23] D. Brutzman. "The Virtual Reality Modeling Language and Java", Communications of the ACM, Vol. 41, No. 6, pp. 57-64, June 1998.

[24] L. Gautier. "Une architecture de communication pour les applications multi-utilisateurs interactives distribuees sur Internet". PhD report (in French language). University of Nice Sophia-Antipolis. September 1998.

Glossary

ADU: Application Data Unit. An ADU is a chunk of data manipulated by the application. For transmission efficiency purposes, it is recommended not to fragment ADUs within the communication stack.

Avatar: Any dynamic object in a game that is controlled either by a participant or automatically by the system.

Dead Reckoning: An extrapolation technique used in the aviation systems to compute an estimate of the current position of a plane based on the knowledge of its position in the past and on its trajectory.

DIA: Distributed Interactive Application are real-time applications where users (i.e. participants) interact in a defined environment. Examples of DIAs are distributed games, digital battlefield, shared virtual worlds, cooperative tools, etc.

DIS: Distributed Interactive Simulation. DIS is an IEEE standard (see references [1][2]) which describes the format of the packets that should be exchanged between simulation entities in a distributed simulation, and that defines the protocol to handle these packets.

IP multicast: An extension of IP to support the construction of trees (instead of point-to-point routes) for the delivery of data to a group of receivers.

Mbone: Virtual overlay installed on the Internet to implement IP multicast.

RTP/RTCP: Real-Time transport Protocol / Real-Time Control Protocol [9]. RTP is an encapsulation format designed to handle realtime data transmission on the Internet. RTP is generally used in conjunction with UDP. RTCP is a control protocol that carries statistic and control information for RTP data flows.

NTP: Network Time Protocol [6]. NTP is a protocol used to synchronize a clock signal over a network (in other words, to provide a global clock in a network). NTP is a client/server protocols where servers are organized in stratum. NTP is sensitive to link asymmetry.

PDU: Protocol Data Unit. PDU is the standard way to describe a packet constructed by a protocol for transmission purposes. In the DIS, the most popular PDU is the Entity State PDU that carries a description of an avatar.

UDP: User Datagram Protocol. UDP is an unreliable transport protocol (as opposed to TCP that guarantees ordered and reliable data transmission). UDP's main functionality is to multiplex/demultiplex data. UDP has been designed to implement real-time applications on the Internet.