End-to-end Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet

Laurent Gautier
INRIA
2004 route des Lucioles
06902 Sophia Antipolis FRANCE
laurent.gautier@sophia.inria.fr

Christophe Diot
Sprint ATL
1 Adrian Court
Burlingame, CA 94010 USA
cdiot@sprintlabs.com

Jim Kurose
Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4610 USA
kurose@cs.umass.edu

Abstract— **This paper reports on the design and the evaluation of transmission control mechanisms specifically designed for multiplayer, distributed (serverless), interactive Internet applications. Distributed synchronization and dead reckoning are the main elements of this transmission control infrastructure. These mechanisms have been implemented in a fully distributed, multiplayer game application, i.e., one in which each entity in a game session computes its own local view of the session. The role of each entity is consequently to periodically send its own state to all other session participants (using RTP/UDP/IP multicast) and to periodically compute its own local view of the global game state using information received from the other participants. A detailed experimental analysis is provided using MBone and LAN experiments We investigate how the "quality" of the game is influenced by the frequency at which players exchange state information, as well as by network impairments such as packet loss and transmission delay.**

Keywords— **Interactive Applications, Distributed Architecture, Internet, Transmission Control, Multipoint Communication.**

I. Introduction

THIS article describes the design and the evaluation of an end- to-end transmission control infrastructure for multiparty interactive applications. Such applications include interactive distributed games and distributed VR environments [22], Distributed Interactive Simulations (DIS) [1][2], command-and-control applications, collaborative tools [27] and Air Traffic Control (ATC)[3][4]. This new generation of applications has data transmission requirements that are significantly different from those of traditional data applications, including a resiliancy to data loss, and the constraint that data be delivered within a certain amount of time in order to preserve real-time interactivity among participants. These new data communication requirements, as well as the possibility of leveraging communication-related, application-specific characteristics [20] of such interactive, multiparty environments (e.g., the fact that participant trajectories are often being computed) suggest that new transmission control mechanisms are required.

We have implemented a multi-player distributed game in order to analyze the transmission control mechanisms we have designed. A distributed game was chosen because it is representative of this new generation of interactive multimedia applications. MIMAZE is inspired from iMaze [8]. iMaze is a bi-dimensional "Pacman" game with a 3D representation. Avatars ("Pacmen") move in a labyrinth in which they try to kill each other. Each player has a 3D representation of its vision domain and a 2D overhead view of the entire game. A detailed description of MIMAZE is given in [13]. From a communication architecture point of view, MIMAZE is serverless, and uses an unreliable communication infrastructure based on RTP [9] over UDP/IP multicast [16]. Because the focus of our work is on transmission control mechanisms, the MIMAZE game rules and graphics are purposefully simple, as a more complex application would have made it difficult to isolate the behavior of the transmission control mechanisms.

In order to guarantee the real-time properties of the application (including interactivity) on the Internet, we have designed a synchronization mechanism to accommodate the heterogeneous transmission delays among the participants. To increase the efficiency of this synchronization mechanism, a dead reckoning based algorithm has also been implemented to cope with error control. We will see that dead reckoning provides a very natural way to recover losses in such a real-time application, where retransmission is impossible. These two mechanisms form the core of our architecture. They are analyzed later in this paper.

The contribution of this paper is consequently to describe original communication-related mechanisms for multiparty interactive applications on the Internet and to provide a detailed analysis of its transmission control parameters. We investigate how the "quality" of the game is influenced by the frequency at which players exchange state information, as well as by network impairments such as packet loss and delay[1]. Our results generally show the transmission control infrastructure we describe can be used to provide an highly interactive, distributed game environment, even in the presence of significant network impairments. Our detailed analysis also reveals important insights into specific aspects of the transmission control infrastructure. We find that the tradeoff between the "quality" of the game and network resource used (rate of transmitting state updates) is such that there is a well-identified point on this tradeoff curve where it is desirable to operate, and that this point

[1]Scalability is not addressed in this paper. We decided that it was more important to first understand the behavior of this new type of application, before designing it to scale to large numbers of participants.

0-7803-5417-6/99/$10.00 ©1999 IEEE.

is relatively independent of loss rate. We show that an increased transmission rate can be used to refine one player's estimate of another player's location only up to a certain point; increasing the transmission beyond that rate only results in the transmission of redundant data. We show that the burstiness of the network packet loss process can effect the quality of the game as much as, or more than, the loss rate itself.

It is clear that we are studying a very specific instance of a set of applications. Even if all results can not be generalized, it was an important first step to understand how these applications will behave on the Internet. Consequently, we do not try to generalize our result, but we try to point out generalizable results.

Before moving to the technical content of this paper, we define here some specific vocabulary introduced from [13]. The representation of a participant in the game is called an avatar. The exact description of an avatar (e.g., position, orientation) is called the avatar state. The exact description of the game (including all avatars, score, terrain, etc.) is called the game's global state. Because the architecture is distributed, each participant computes its own view of the game global state, which we will refer to as the "local" view of the global state. Each participant location is called an entity. A game entity periodically sends its local state to all other entities. It also computes and displays periodically the global state of the game to the participant [2]. The game is perfectly "consistent" if all entities in the game compute and display the same global state. We elaborate further on these definitions in section 2.

The remainder of this paper is structured as follows. Section 2 describes MIMAZE's end-to-end transmission control infrastructure, which is based on distributed synchronization and dead reckoning. In section 3, we describe and analyze performance measurements realized on the Mbone and in a LAN. We then empirically investigate how game quality is influenced by the frequency at which players exchange state information, and by various levels of packet loss and delay. Section 4 concludes the paper and discusses future work.

II. End-to-end transmission control infrastructure

In a multi-participant game played over a network with best effort service (such as the Internet), each player will experience varying and unpredictable network delays and packet loss. The goal of a transmission control infrastructure that supports such an application is to mask this delay and loss, while providing a "consistent" and "timely" view of the game to the distributed participants. By "timely" and "consistent" we mean the following:

- **Timeliness.** When an action is played (or issued) by a player, it should be displayed to all participants within a relatively short amount of time. The DIS standard [1,2,3] recommends a maximum delay of 150 ms, while commercial networked game companies note that the quality of game begins to degrade when the delay is on the order 200 milliseconds [22]. In practice, companies offering networked games services insure that such delay constraints are always met by over-provisioning bandwidth to ensure that congestion and packet loss never occur. Our interest here, on the other hand, is in designing and evaluating robust transmission control mechanisms that operate in the face of variable network delays and packet loss.
- **Consistency.** At any point in time, all players should ideally "see" the same information at the same time, in spite of network delay and losses. That is, if player X takes an action (e.g., makes a move) that should influence the current game view of players Y and Z, then *consistency* requires that Y and Z should display the results of X's action at *approximately* the same time.

Note that consistency concerns the degree of similarity of displayed information, while timeliness concerns how soon that information is displayed at the various game sites after its transmission from the sending site (interaction is an aspect of timeliness). In section 3 we formalize these notions and evaluate transmission control mechanisms with respect to these performance metrics.

MIMAZE adopts a fully distributed architecture. Distributed architectures have a number of advantages for interactive multi-participant games, including robustness (e.g., the failure of one entity has no effect on the others), and scalability (e.g., a distributed architecture more easily allows for natural partitioning of game computation as the number of players increases). A discussion of the advantages and disadvantages of a distributed architecture are beyond the scope of this paper and are covered in detail in [12]. For the purposes of this paper, we simply note that a distributed game has no server that computes a unique global state. Instead, each entity computes its own local view of the global state of the game using information received from other entities. This locally-computed view is that which is displayed to the local participant.

The key challenge then is to provide as timely and as consistent a display as possible within the context of such a distributed architecture. The consistency requirement implies that even though each participant computes its own local view of the games state, a distributed synchronization technique is needed to ensure that the participants compute games states that are as similar as possible at a given time; such a technique is discussed in section 2.1. Entities must also recover from lost or overly delayed messages from other players. The stringent timing constraints and relaxed reliability requirements of interactive games suggest that ARQ-based error recovery techniques developed for traditional data applications are not well-suited to our environment. We will see that the nature of the information exchanged in our application makes it possible to use interpolation and extrapolation techniques to recover missing information. In section 2.2, we discuss such dead reckoning techniques. *Together, the techniques of distributed synchronization and dead reckoning form the cornerstone*

[2] We will see later that the sending frequency and the display frequency are independent parameters.

of a transmission control architecture for distributed, interactive, multi-participant applications.

A. Distributed synchronization

Given a distributed architecture in which each participant computes its own local view of the global state, two complementary aspects of the synchronization problem can be identified:

- All entity local states issued (transmitted in an Application Data Units (ADU) [20] via the network) at nearly the "same time" (by various entities) must be processed "together" by a receiving entity when computing its local view of the global game state.
- All entities should display the same information (i.e., ideally, compute identical views of the global state) at approximately the same time.

The bucket synchronization mechanism described in section 2.1.1 meets these requirements. This approach requires the use of a global clock mechanism, an issue addressed in section 2.1.2.

A.1 The bucket synchronization mechanism

A consequence of the varying network delays between entities is that the timing/ordering of ADU reception at an entity need not (and typically will not) reflect the timing/ordering in which those ADUs[3] were actually sent. Thus, a mechanism is needed to insure that events occurring at "close" to the same time at distributed entity sites are considered together whenever an entity computes its local view of the global state. The principle of the bucket synchronization mechanism is for all players to delay (for an amount of time, Δ) their computation of their local view of the global state so that avatar state descriptions issued at the same time by remote entities (but received with different delays) can be used together in the computation of the local view.

The bucket mechanism (Figure 1) operates by considering time to be divided into intervals of length T, with $1/T$ being known as the bucket frequency. When computing its local view at the end of time interval i (which has a length of $[i, i+T]$), an entity uses all received ADUs that were generated by the remote entities during interval $[i\text{-}\Delta, i\text{-}\Delta+T]$, as well as its own local states during the same interval, to compute this local view. In this sense, actions occurring during the same interval of time (e.g., during interval $[i\text{-}\Delta, i\text{-}\Delta+T]$) at the various sites are grouped together in the same receiver "bucket" (e.g., the bucket at the end of interval i). The added delay, Δ, compensates for network delays – as long as an ADU is received within Δ time units of having been sent, it will be processed by the receiver in its state computation. The placement of ADUs into buckets assures that ADUs generated at approximately the same time (i.e., during the same time interval) are used at the same time at the receiver. An ADU that is received after its bucket has been processed is considered a late arrival; it is stored in his original destination bucket (as discussed in section 2.2). We note that the bucket mechanism is closely related to the buffering mechanisms used in packet audio playout algorithms [10].

Figure 1 illustrates bucket synchronization, showing three entities (A, B, and C). The ADUs sent by entities A and C to entity B at times $t1$ and $t3$, respectively, and B's own local event information at $t2$, are used together by B at the end of interval i to compute its local view of the global state. This synchronization occurs even though the information in the bucket is received by B during different intervals. It is worth explicitly noting that the event generated at $t4$ by C, is not used at the end of interval i, even though it has already been received.

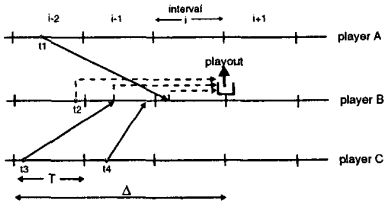

Fig. 1. Bucket Synchronization

The bucket synchronization mechanism has two important parameters that will influence the perceived "quality" of the game:

- **Playout delay.** The value chosen for Δ will determine the playout delay (the time between a packet's generation at a sender and its playout at a receiver) associated with a packet. *In MIMAZE, the value of the playout delay is 150 ms.* The DIS standards [1,2,3] and commercial interactive games vendors [22] claim that game quality begins to degrade when the delay exceeds 200 ms. Note that with a larger playout delay, fewer ADU's will be "lost" due to late arrival (ADU's that arrive after their bucket playout time are considered lost, at least with respect to the computations performed on that bucket). However, with a larger playout delay, the game becomes less "real-time" and "interactive."
- **Bucket frequency.** The bucket frequency $1/T$, controls the time scale over which events are aggregated and considered "simultaneous." It also defines the rate at which the view/display is updated. This should be high enough to provide a natural fluidity in the displayed image. In the case of human perception, a frequency of 25 image refreshes per second (each image refresh corresponds to the computation of a new global state) guarantees the smooth display of objects moving smoothly in the image sequence. Display fluidity is not the only aspect of the game influenced by bucket frequency – the degree of synchronization among participants is affected as well. Even with perfectly synchronized clocks, if the period between two buckets is T, all actions occurring within the same period will processed at the same time, and thus the consequences of two events that are separated by up to T time units will appear to have occurred simul-

[3]With ALF, each local state is encoded and carried as a single ADU. Thus, we often use the term "ADU" instead of "local state".

taneously. This problem can be even more significant when clocks are not synchronized. Finally, the bucket frequency is constrained by the CPU time needed for any message reception, computation, and display of the local view – all of which must be performed during each interval of length T.
Note that the bucket frequency and playout delay are independent of each other, and independent of network characteristics.

A.2 Global clock mechanism

The bucket synchronization mechanism described above assumes a global clock mechanism (equivalently, synchronized clocks) that provides a common timing reference among the distributed entities. For distributed, interactive, real-time environments such as that considered in this paper, accurate and synchronized clocks are extremely important. In [19], it is shown that all entity clocks should be within 10 ms of each other. In our implementation, we use NTP [6], and where NTP is not available, an NTP-like algorithm based on the evaluation of the round trip time [6]. Where possible, we used GPS receivers in our experiments to help with clock synchronization, as discussed in section 3.1.1.

A number of issues arise when using NTP. We have found that it is difficult to maintain good synchronization among participants when level 3 NTP servers are involved. Lower stratum mechanisms (e.g., ntpdate) are also not sufficient. NTP does not provide a reference clock signal; instead, each participant must compute an offset for any other participant. In our 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. In the experiments discussed later in this paper, almost all hosts were synchronized on a GPS system.

B. *Dead reckoning*

Dead Reckoning (DR) is an extrapolation technique initially developed in the aviation domain to estimate the current position of an airplane based on the last known position and on the motion vector. In our application, there are two different uses of DR:

- **For lost or late ADU recovery.** DR is used by an entity whenever it computes a new local view of the global state and finds that the bucket contains no updates from a remote entity. In this sense, DR is performed using information found in the most recent past bucket that contains the missing information by the receiver to evaluate the value of missing information. This information may be missing for several reasons. The ADU containing an update may have been lost or overly delayed in the network. It is also possible that a remote entity did not send an ADU that maps to the current bucket in the first place. For example, that it may be advantageous for an entity to decrease its ADU transmission rate in the presence of network congestion. Additionally, an entity may change its ADU transmission rate due to local (e.g., CPU) resource constraints.
- **For collision detection.** Before sending an ADU to other entities, a local entity dead reckons its position into the future. If a collision with another entity is determined to be likely in the near future, the results of this collision can be determined in advance of the collision itself. This allows the results of the collision to be displayed in as close to real-time as possible at the entity. We do not analyze this use of DR in this paper.

Given the playout delay mechanism described in section 2.1.1, DR can be used not only to extrapolate state information, but to interpolate it as well. In the latter case, the receiver uses ADU's associated with past buckets, as well as already-received ADU's that have been mapped into future (yet-to-be-processed) buckets. to interpolate the state of an entity at a given point in time.

A number of possible DR algorithms can be identified [1][2][26]. The simplest algorithm would be to simply replay the last known position. This algorithm has minimal CPU cost for computing the missing information, but may not provide an accurate extrapolation of the missing position.

In MIMAZE, the multicasted state description of a player includes the following information: its current position, orientation (i.e., the direction the avatar is facing, which defines its instantaneous direction, if it is moving), velocity, and "angular speed" (i.e., the rate at which is direction is changing). With this state description, if a given ADU is received but the following one is lost, the receiver uses the information contained in the received ADU to extrapolate ("dead reackon") the missing information. In the current version of MIMAZE, when an avatar's future position is dead-reckoned based on its last known position, its direction and velocity at that time, and its angular speed are all used. It is assumed that velocity and angular speed remain constant over the period of time that a remote player's position is dead reckoned. Note that by including derivative information (velocity and angular speed), it is possible to dead-reckon a missing position based on a single ADU (a number of simpler DR algorithms requires two or more ADUs to dead-reckon a missing one).

In section 2.1.1 we identified bucket frequency and playout delay as two key parameters of the bucket synchronization mechanism. The dead reckoning algorithms discussed above highlight a third key parameter: the ADU transmission frequency, i.e., the rate at which an entity transmits state update ADUs to the other entities[4]. A high ADU transmission frequency is desirable from a number of standpoints. Because a remote entity is providing frequent updates of its position, its reported motion appears more "fluid." Also, if the transmission frequency is too slow, the receiver may not be able to detect fast time-scale changes in the sender's trajectory. We examine this issue in more detail in section 3.3.2. A final consideration is that if an

[4]The transmission frequency is thus also the rate at which a sender samples and sends its state.

ADU is lost, the change reported from one ADU to the next will be smaller than with a lower transmission frequency and a DR algorithm should consequently be more accurate. However, the higher bandwidth and CPU requirements associated with a higher ADU frequency make it undesirable from a resource consumption standpoint.

In section 3.3 we evaluate the impact of the bucket frequency on the consistency of the game. We will not explicitly consider CPU resource constraints. We note here, however, that ADU transmission frequency is at the center of a number of interesting tradeoffs between CPU processing for DR and for protocol (message reception) processing. For example, while a high transmission frequency means that there is less change in state between ADU's (and hence the quality of DR should be higher), it also results in a higher protocol processing load on a receiver and consequently fewer CPU cycles available for performing the actual DR calculation itself. Limits on the available computation time may, in turn, limit the type of DR algorithm that can be executed.

C. Related work

To our knowledge, MiMaze is unique in the area of Internet games, being based on a serverless architecture together with distributed synchronization and dead-reckoning based error control. It is clear that other applications use synchronization and dead-reckoning, but we could not identify one in a distributed environment context.

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 architecture but manage states 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 private companies. Microsoft, BT, Intel have their own game services. But the approach of these companies is to over-engineer the network in order to maximize the quality of the game. There are also private companies such as Mpath [22] that develop more sophisticated transmission infrastructure (but still with network over-engineering) for distributed games.
- 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 [21,26] at Stanford University aims to architect and build a large-scale internetworked simulation environment that supports multiplayer 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 [26] 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. By contrast, the work in [26] is aimed primarily at aggregation (not considered here) and specific dead reckoning algorithms. It worth noting that dead reckoning is used in [26] primarily to decrease state transmission frequency and smooth trajectories between state updates. This is reflected in their evaluation, which assumes no network loss or delay.

III. Performance Analysis

In this section we experimentally investigate the performance of various aspects of the MIMAZE architecture. Broadly speaking, our goal here is to examine how the "quality" of the game is influenced by the frequency at which players exchange state information (e.g. the so-called transmission frequency), and by network impairments such as packet loss and delay.[5]

We begin by describing the experimental setting and methodology for the evaluation in section 3.1. In later sections we present and analyze various measurements.

Since MIMAZE currently has no congestion control algorithm, we decided to make the following experiments with no more that 5 participants, so that we limit the effect of MIMAZE-induced network congestion on the loss rate[6].

A. Experimental framework

Since the distributed MIMAZE architecture exploits IP multicast capabilities, our evaluation uses the MBone [11]. The general experimental setup for our evaluations was as follows. Players were located at 4 MBone sites running variously on SUNs (SPARC 10, 20, ULTRA), DEC Alphas, and PCs. A single player was located at the UCL (London), LAAS and LIP6 (France), while the number of participants at INRIA was varied, as discussed below. In certain cases (as noted later), subsets of this configuration were used in our experimental evaluation.

The players themselves are so-called "ninjas" – software-directed (rather than human-controlled) avatars that are a standard component of the MIMAZE game. Each ninja's behavior is simple – it chases other ninjas (using the information received from all the other ninjas) and attacks any other ninja that comes within its firing view. Each participating computer runs a single ninja. We made a conscious decisions to primarily use ninjas in our evaluation rather than human-controlled entities since our previous experience [13] indicates that human players can introduce artifacts (e.g., occasionally ignoring an on-going game to focus on another activity, or varying skill levels ranging from a novice that has never played to an expert player) into the

[5] We do not consider here the effects of the changing playout delay or display frequency, as these are primarily non-network-related, human factors considerations.

[6] we remind here that MIMAZE scalability (i.e. varying the number of participants) is intentionally not addressed in this paper).

game. Such considerations can make it difficult to separate out the influence of variable human player behavior from the influence of game parameters and network performance on the quality of the game. This latter influence is of more interest to us here and so we initially chose to perform the evaluation in a *controlled* setting with player (ninja) behavior that can be considered representative of typical human player behavior. We show that ninjas are a good model for human players in [13].

To quantitatively evaluate various aspects of the MIMAZE transmission control infrastructure, we define here the notion of "drift" as follows. Recall that at the simplest level, the game consists of distributed players moving in a maze. Each player displays the view seen by its avatar in the maze, as well as an overhead (global) view of the maze. Thus, each player will have position information about each of the other players. Suppose now that entity A's view of its own spatial location (as computed by A using the bucket synchronization algorithm pictured in Figure 1) at time t is (x, y) and that entity's B's view of A (computed by B using its own local bucket synchronization computation) at time t is (x', y'). Then the drift (or error on the estimated position) associated with B's view of A is simply the Cartesian distance between these two positions: $\sqrt{(x-x')^2 + (y-y')^2}$. To give a feel for the magnitude of the drift values, we note that the MIMAZE game is a 65Kx65K two dimensional grid in which a player moves at 32 units every 40 milliseconds when in motion; the avatar itself has a radius of 32 units.

In the following, we will only consider the drift associated with oneplayer's view of another player's position. That is, we will not average the drift over all players. This is because there will be different network delays and losses between each pair of players and it is precisely the effect of such impairments on game quality that we want to quantify.

A.1 Monitoring, trace synchronization, and trace filtering

In order to evaluate the performance of our transmission control mechanisms, each entity collects a trace of its activity (including application level, network level and synchronization data)during a game session. These traces are analyzed off-line.

The main difficulty in analyzing these traces is trace re-synchronization. Note that as a result of the distributed system architecture, there is no absolute clock in a game session. Instead, each participant computes the clock offset between itself and any other participant in the session. 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 re-synchronization.

We have shown in an early experiment that clock synchronization was an important aspect of game consistency and in determining traces usability [13]. The clock signal of the remote computers was carefully controlled and an acceptable clock synchronization was guaranteed by NTP *stratum 1* and *2* implemented on any computer involved in the experiment.

As discussed in section 3.3, we will occasionally break a single trace (e.g., of a single player in a 15-minute game) into a number of smaller traces, according to a well-defined criteria (e.g., the packet loss or delay experienced during various intervals of time). With a single large trace, widely disparate behaviors in the environment (e.g., a period of extremely high packet loss or delay, versus a period when such values are small) and their impact on performance of our transmission control mechanisms can be "washed out" by averaging performance over a long trace. Since one of the main goals of this paper is to understand the performance of our transmission control mechanisms as a function of specific network conditions, we will occasionally classify trace subintervals according to such conditions (and note when we do so).

B. Illustrating the Impact of Packet Loss

In today's public Internet with its best-effort service, packet loss and delay are a fact of life; this is particularly true in the MBone. The situation is also aggravated by ADUs that reach their destination with a delay which is longer than the playout delay. In that case, the arriving data is stored in a past bucket, and the current bucket will process this information as if it was lost (section 2.1.1).

In this section we illustrate the consequences of such network impairments on MIMAZE by analyzing traces of network and application-level activity.

Previous measurements of MBone traffic [23] have noted periods of "outage" when no packets are received from the network. We also observed such outages throughout the course of this study. In particular, we observed long periods of time (ranging from a few seconds to a few tens of minute) when the Mbone did not deliver any packets. It has been conjectured [23] that such outages result from instability in the underlying (unicast) routing protocols [25]. From a game-playing standpoint, such long periods of time with no ADU reception make dead reckoning difficult. In MIMAZE, based on previous observations [14], we have chosen to dead reckon up to 1 second, and then remove an avatar for which no information has been received for the past second.

Figure 2 illustrates the impact of packet loss on MIMAZE behavior. Figure 2 contains two plots. The upper graph shows the state drift (or distance) between site A's actual avatar position and a remote site (B's) estimate of that position as a function of time. The lower graph in Figure 2 plots the number of consecutive packet losses, i.e., consecutive ADUs not received from A by B. These traces were gathered in a configuration with five game participants sending ADU's[7] at a rate of 25 ADU's/sec.

Several observations can be made from Figure 2:

- First, as discussed above, there are periods of time when the position of the remote avatar can not be computed. These periods of time are indicated by the absence of an "x" (upper graph) during an interval of

[7] Recall that each ADU contains an update description of the avatar position

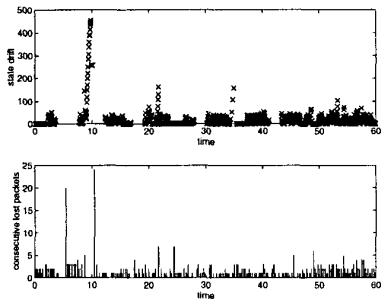

Fig. 2. Drift and consecutive packet loss as a function of time

time. They correspond to periods where the avatar was dead (killed by another avatar).

- Figure 2 also indicates that when the loss rate is small, the drift is generally very small. In the best case, there is no drift (i.e., the estimate of a remote avatar's position is exactly correct). We note that for bursts of 3 to 5 consecutive losses, our DR algorithm succeeds in estimating the remote avatar's position with limited error. Specifically, with an error of less than 50 units in 90% of the cases.

C. Impact of ADU transmission rate on game consistency

In this section we analyze the impact of the frequency with which ADUs are sent (i.e. transmission frequency) on the consistency of the game. This parameter is the most important of the game (at least from a network point of view) as it has influences on game quality, and on the CPU and bandwidth requirements. Our goal here is consequently to understand the tradeoff between the transmission frequency and the consistency of the game under various network conditions.

It is important to notice that in MiMaze, avatars are 32 unit radius spheres, and that their displacement is constant speed, with a trajectory that can be either a straight line or a circle.

C.1 Transmission frequency

Figure 3 plots the average drift in one player's estimation of another player's avatar position as a function of the ADU transmission rate[8]. 90% confidence intervals are also shown in the figure. The drift values shown in Figure 3 were obtained directly from our traces; whenever an avatar added its own state description to the trace file, it also added its computed position of the other players as well.

Three curves are plotted in Figure 3, corresponding to the cases where *(i)* the two players are on the same LAN and hence there is negligible loss and very short delay between the two players (the elvis-droopy curve), *(ii)* the two players are connected by a wide area MBone connection with a packet loss rate that is typically less than 5% and an average delay that is approximately 100 ms (the elvis-speedy curve), and *(iii)* the two players are connected by high loss and high delay MBone connection with a loss rate that averages approximately 40% and an average delay of approximately 175 ms (the speedy-elvis curve). The data for graphs was obtained from a series of 15 minute games (each played for 15 minutes at a given transmission rate) among five hosts. We divided each game into 100-second intervals and removed any interval where the receive rate was less than 20% of the ADU send rate and computed the drift over the remaining intervals (with the exception of the speedy-elvis curve, almost no intervals were filtered out in this manner).

[8]the maximum frequency range of 30 corresponds to the human perception limits

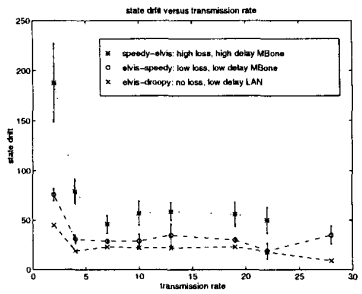

Fig. 3. Average state drift versus transmission rate

Two important observations can be drawn from Figure 3. First, we note that for a given pair of players, the drift remains relatively constant as the transmission rate decreases from 28 ADU's per second down to a rate of approximately 5 ADU's per second. Clearly, the sender's higher sampling frequency and ADU transmission rate are not resulting in a noticeable improvement of the game consistency.

A related observation concerns the state drift associated with the flat portions of each of the curves. First, note that in the flat portions of the curves, the drift remains small (less than 50 units for a 64 units wide avatar moving at 32 units per 40 ms). From a qualitative standpoint, this is a small error indeed. Figure 3 also shows that, as expected, a low loss and low delay network connection results in smaller state drift. It might appear that the increase in drift from one curve to the next results from the higher loss associated with each curve. On the other hand, note that that halving the transmission rate (e.g., decreasing the ADU transmission rate from 20 to 10) might be *roughly* considered to have the same effect as moving from a no loss regime to a 50% loss regime. Yet moving from a transmission rate of 20 to 10 on the elvis-droopy curve has no noticeable effect, while moving from the elvis-droopy curve to the speedy-elvis curve at the same value of 20 ADU's per second has a noticeable change in the drift. We will resolve this issue shortly in section C.3, where we take a deeper look at the effects of loss on game quality.

C.2 Avatar trajectories

Clearly, we would like to choose the transmission frequency so that the drift can be minimized. However, increasing the transmission frequency increases the required

bandwidth and can possibly lead to congestion and increased loss. The compromise is consequently to maintain the transmission rate at a point where receivers have enough information to display a relatively consistent view of the game. This point is not only defined by the operating conditions (loss rate, delay, etc.), but also by the game nature, and in particular by the characteristics of the avatars' trajectory.

It is thus of interest to characterize avatar trajectory within MIMAZE. Figure 4 plots the percentage of ADUs that can be dead reckoned depending on the transmission frequency. An ADU is said to be dead reckonable if it is on the same trajectory (which in MIMAZE must either be a line or a circle) as the previous ADU, and if the two ADUs are consecutive. Figure 4 shows that the transmission frequency has an important influence on the percentage of ADU that can be dead reckoned. As we would expect, the percentage of dead reckonable ADU's increases with an increased transmission rate. Intuitively, the more frequently a trajectory is sampled, the more samples belong to the same trajectory, and hence the more likely it is that the points on that trajectory can be dead reckoned.

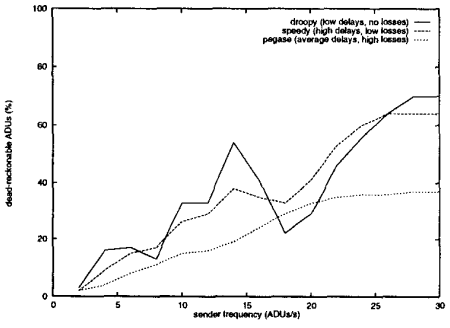

Fig. 4. Percentage of states that are dead reckonable

Figure 4 also shows that the higher loss rate curve (the curve labeled "pegase") results in a smoother increase in the percentage of dead reckonable ADU's. We conjecture this results from the ninja's behavior. Recall that ninjas chase other ninjas. The more information they receive from other entities, the more opportunity there is to detect changes in the other ninjas' trajectories, and consequently the more opportunity there is for a ninja's own trajectory to change. If the number of avatars is high, and the loss rate is low, then ninjas will continuously change their direction. The droopy curve (plain line) illustrate this later point. On the other hand when the loss rate is high at one entity, the local ninja has less information to use in computing the remote ninjas' trajectories (and consequently estimates fewer changes in the remote ninjas' trajectories) and hence itself continues for longer periods of time without changing its own trajectory [9].

We also analyze the peaks in droopy graph (recall that droopy experiences no loss in this graph). We conjecture that the peaks under 18 times per second (around 5 and 14) result from "errors" in the trajectory evaluation.

Consequently, the results in Figure 4 would argue that the transmission frequency required to observe all the trajectory details should be 18 states per second. Sampling at lower frequencies than 18 is also possible, but the observed trajectory would be a "subset" of the real one (some of the trajectory details would not appear, but the overall direction would be visible).

This very important result is specific to MIMAZE. We believe, however that similar types of observations can be made for other type of avatars, i.e., that the minimum transmission frequency depends primarily on the motion properties of the avatar, including its acceleration.

C.3 Loss Analysis

We have shown in the previous experiments that the network parameter that most dramatically affects the game consistency is the loss rate. Heterogeneous delays also increase loss as a side effect (since a late ADU has not yet arrived by the time its bucket is processed). In this section we thus provide a deeper analysis of the influence of losses on the game consistency.

In order to study MIMAZE performance in a more systematic way, we wanted to vary the network conditions in a more controlled manner – something not possible using an operational network such as the MBONE. We do so as follows. We begin with a trace from an actual live game of a player's position, it ADU transmissions, and the receipt (or loss) of its ADUs at a remote player. Given the set of received ADU's and a player's DR algorithm, we can reconstruct its view of each remote player's position as a function of time (indeed, in a real game, all of a player's information about remote player positions is obtained via the received ADU's). Given a trace of the actual positions of the remote player and the reconstructed view of these positions, we can then compute the drift in one player's view of another player's position. We can also systematically vary network behavior by, for example, simulating the loss of transmitted ADUs and determine the reconstructed view given this simulated network behavior. We can also simulate the variation in the transmission rate by "thinning" the set of transmitted ADUs. For example, by removing every other ADU from a trace, we can simulate the situation in which the sending avatar would have transmitted its state updates at half the original frequency. By keeping only every fourth ADU in the trace, we can simulate a transmission rate of one fourth the original rate.

But does such a thinning result in behavior similar to what we empirically observed in Figure 3? Figure 5 addresses this question. The curve labeled "extrapolated: elvis-droopy" (represented by a plain line) is obtained by taking the single set of elvis-droopy trace data at a transmission rate 28 ADUs/sec, thinning the transmissions (as discussed above) and applying the DR algorithm to the received ADUs. *Thus, the entire curve was obtained from a single set of trace data taken at 28 ADUs/sec.* This thinned curve is to compare to the curve that links the

[9] We have observed the same behavior for human players. The figure is not presented in this paper for readability.

x (the curve has not been drawn to keep the figure clear), where x where obtained by playing MiMaze at the different transmission frequencies. We see good agreement between the extrapolated performance curve obtained by thinning a single trace, and the performance computed from the separate individual traces. The agreement is not quite as good for the speedy-elvis curve (the thinned curve is represented by a doted line, and the experimental observations by the "o"), although the thinned data shows the characteristic sharp increase in drift at low transmission rates. We also note that the point-valued data (o's) obtained from the individual speedy-elvis traces is not very "smooth" in the first place – probably due to the high loss rate (see section 3.4.1) between speedy and elvis[10].

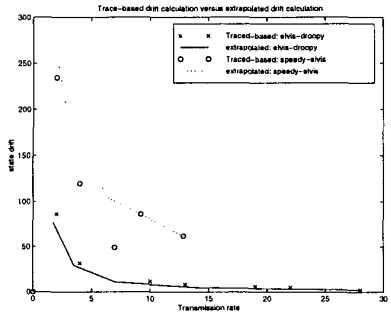

Fig. 5. Calculating drift: thinning one trace versus using separate traces

Given our confidence that thinning will now allow us to more systematically study the effects of loss on performance, we consider the following scenarios. We first begin with a no-loss trace and then thin the trace as discussed above. We introduce loss into the trace by modeling network loss as a memoryless process (that is, each packet is independently lost with a given probability). The results are shown in Figure 6 for no loss, 10% loss and 30% loss. We note that the curves have the same characteristic shape as in Figure 3 – a flat portion of the transmission rate versus state drift tradeoff, with a marked increase in drift as the transmission rate fall below 5 ADU's/sec. We note, however, that the curves are not as separated from each other as in Figure 3.

The curves in Figure 6 were each obtained under a memoryless loss model. That is, the loss probability for a given packet was independent of whether the previously sent packet was lost or not. Recent unicast and multicast measurements [28] suggest that end-end packet losses show correlation for times scales up to eight hundred milliseconds. It is thus of interest to examine performance under a bursty loss model.

We thus next consider a two-state loss model. When in the no loss state, a transmitted packet will be received successfully; when in the loss state, a transmitted packet will be lost. We can vary the "burstiness" of the loss process by varying the transition probability from the loss state back to the loss state.

[10]Note that differences can also arise from the fact that in the case of trace based data points taken from an actual game, networking conditions were different for each game. This was not the case for the thinned curved (that are obtained from a single trace).

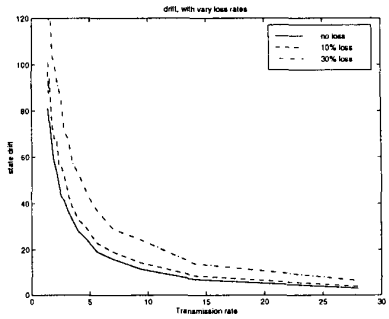

Fig. 6. Drift: varying loss rates

Figure 7 plots the state drift versus transmission rate tradeoff for 10% loss under different loss burstiness scenarios. In one case, whenever a packet is lost, the next packet is successfully delivered (an average loss burst length of one) and in the second case, the average length of the burst of consecutively lost packets is 5. We note that with higher loss burstiness, the drift is significantly higher. In fact, comparing Figures 3 and 7 we see that the results of the bursty 10% loss model are similar to those from the 30% independent loss model – indicating the important role that loss burstiness plays in determining state drift. This also points out the flaw in our earlier conjecture in the discussion of Figure 3, that decreasing the ADU transmission rate in half might be roughly considered to have the same effect as moving from a no loss regime to a 50% loss regime. The former approach smoothly removes packet receptions while the latter approach results in losses (packets being removed) in a more bursty manner. Figure 7 tells us that burstiness plays a key role in determining performance.

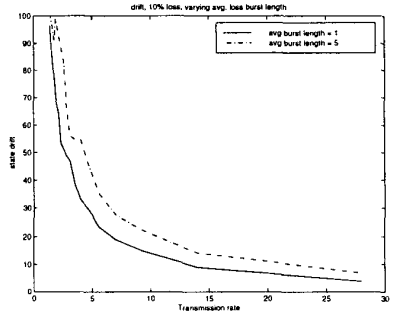

Fig. 7. Drift: identical loss rates but different mean loss burst lengths

Let us conclude with several observations regarding loss burstiness. First, in our distributed architecture, each player sends its own stream of ADU's. While bursty loss may affect some ADU streams (in which case, the drift associated with those players may be high), it will not affect other ADU streams. If game control were centralized, bursty loss from the central site would result in poor game performance for everyone. A second observation is that given the effects of bursty loss, it is important to consider

error recovery (and perhaps congestion control) algorithms that operate in the face of bursty loss.

IV. Conclusion

This paper has described the design, implementation and analysis of a distributed (serverless) interactive multi-participant game on the Internet. The architecture has two key mechanisms for end-to-end transmission control: the bucket algorithm and dead reckoning.

We have shown that the real-time capabilities of the game can be maintainedgiven that a small error can be intriduced on the game consistency. We have also shown that the transmission frequency can be varied with a limited influence on the game consistency. Finally, this paper suggests that a robust mechanism to control bursty losses would be useful.

Several aspects of the our current transmission control infrastructure must still be enhanced to address the important problems of scalability (large number of participants), and application complexity (graphics, video, spatial audio). We are currently working on these problems:

- Incorporate a congestion control algorithm. Distributed interactive application offers a very simple way to control the congestions by varying the local state transmission frequency.
- Developing a new mechanism to detect collisions between avatars by "anticipation". We propose to use source dead reckoning to anticipate collisions.
- Study participant sub-grouping in order to improve the game scalability and quality. Such techniques, generally based on grid subgrouping, are recommended by the DIS community [1][2][18].

Acknowledgments

The authors want to thank Mostafa Ammar, Jon Crowcroft, Serge Fdida and Michel Diaz for having provided the experimental resource.

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.

[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 Multiuser 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". 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.

[20] D. Clark and D. Tennenhouse. "Architectural Considerations for a New Generation of Protocols". In ACM SIGCOMM '90, (pp. 200-208).

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

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

[23] M. Yajnik, J. Kurose, and D. Towsley. "Packet Loss Correlation in the Mbone Multicast Network". IEEE Global Internet Conference. London. November 1996.

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

[25] V. Paxson, "End-to-end Routing Behavior in the Internet," Proc. 1996 ACM SigComm, (Stanford University, California), pp. 25-39.

[26] 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.

[27] M. Handley and J. Crowcroft, "Network Text Editor (NTE): A scalable shared text editor for the MBone", Proc. ACM Sigcomm 1997, Cannes, France, Sept. 1997.

[28]. M. Yajnik, S. Moon, D. Towsley, J. Kurose, "Measurement and Modeling of the Temporal Dependence in Packet Loss," submitted to IEEE Infocom99.