Fairness in Dead-Reckoning based Distributed Multi-Player Games

Sudhir Aggarwal Hemant Banavar
Department of Computer Science
Florida State University, Tallahassee, FL
Email: {sudhir, banavar}@cs.fsu.edu

Sarit Mukherjee Sampath Rangarajan
Center for Networking Research
Bell Laboratories, Holmdel, NJ
Email: {sarit, sampath}@bell-labs.com

ABSTRACT

In a distributed multi-player game that uses dead-reckoning vectors to exchange movement information among players, there is inaccuracy in rendering the objects at the receiver due to network delay between the sender and the receiver. The object is placed at the receiver at the position indicated by the dead-reckoning vector, but by that time, the real position could have changed considerably at the sender. This inaccuracy would be tolerable if it is consistent among all players; that is, at the same physical time, all players see inaccurate (with respect to the real position of the object) but the same position and trajectory for an object. But due to varying network delays between the sender and different receivers, the inaccuracy is different at different players as well. This leads to unfairness in game playing. In this paper, we first introduce an "error" measure for estimating this inaccuracy. Then we develop an algorithm for scheduling the sending of dead-reckoning vectors at a sender that strives to make this error equal at different receivers over time. This algorithm makes the game very fair at the expense of increasing the overall mean error of all players. To mitigate this effect, we propose a budget based algorithm that provides improved fairness without increasing the mean error thereby maintaining the accuracy of game playing. We have implemented both the scheduling algorithm and the budget based algorithm as part of BZFlag, a popular distributed multi-player game. We show through experiments that these algorithms provide fairness among players in spite of widely varying network delays. An additional property of the proposed algorithms is that they require less number of DRs to be exchanged (compared to the current implementation of BZflag) to achieve the same level of accuracy in game playing.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Distributed multi-player games, fairness, dead-reckoning, clock synchronization, network delay

1. INTRODUCTION

In a distributed multi-player game, players are normally distributed across the Internet and have varying delays to each other or to a central game server. Usually, in such games, the players are part of the game and in addition they may control entities that make up the game. During the course of the game, the players and the entities move within the game space. A player sends information about her movement as well as the movement of the entities she controls to the other players using a Dead-Reckoning (DR) vector. A DR vector contains information about the current position of the player/entity in terms of x, y and z coordinates (at the time the DR vector was sent) as well as the trajectory of the entity in terms of the velocity component in each of the dimensions. Each of the participating players receives such DR vectors from one another and renders the other players/entities on the local consoles until a new DR vector is received for that player/entity. In a peer-to-peer game, players send DR vectors directly to each other; in a client-server game, these DR vectors may be forwarded through a game server.

The idea of DR is used because it is almost impossible for players/entities to exchange their current positions at every time unit. DR vectors are "quantization" of the real trajectory (which we refer to as *real path*) at a player. Normally, a new DR vector is computed and sent whenever the real path deviates from the path extrapolated using the previous DR vector (say, in terms of distance in the x, y, z plane) by some amount specified by a threshold. We refer to the trajectory that can be computed using the sequence of DR vectors as the *exported path*. Therefore, at the sending player, there is a deviation between the real path and the exported path. The error due to this deviation can be removed if each movement of player/entity is communicated to the other players at every time unit; that is a DR vector is generated at every time unit thereby making the real and exported paths the same. Given that it is not feasible to satisfy this due to bandwidth limitations, this error is not of practical interest. Therefore, the receiving players can, at best, follow the exported path. Because of the network delay between the sending and receiving players, when a DR vector is received and rendered at a player, the original trajectory of the player/entity may have already changed. Thus, in physical time, there is a deviation at the receiving player between the exported path and the rendered trajectory (which we refer to as *placed path*). We refer to this error as the *export error*. Note that the export error, in turn, results in a deviation between the real and the placed paths.

The export error manifests itself due to the deviation between the exported path at the sender and the placed path at the receiver **(i)** before the DR vector is received at the receiver (referred to as the *before export error*, and **(ii)** after the DR vector is received at the receiver (referred to as the *after export error*). In an earlier paper [1], we showed that by synchronizing the clocks at all the players and by using a technique based on time-stamping messages that carry the DR vectors, we can guarantee that the *after export error* is made zero. That is, the *placed* and the *exported* paths match *after* the DR vector is received. We also showed that the *before export error* can never be eliminated since there is always a non-zero network delay, but can be significantly reduced using our technique [1]. Henceforth we assume that the players use such a technique which results in unavoidable but small overall export error.

In this paper we consider the problem of different and varying network delays between each sender-receiver pair of a DR vector, and consequently, the different and varying export errors at the receivers. Due to the difference in the export errors among the receivers, the same entity is rendered at different physical time at different receivers. This brings in *unfairness* in game playing. For instance a player with a large delay would always see an entity "late" in physical time compared to the other players and, therefore, her action on the entity would be delayed (in physical time) even if she reacted instantaneously after the entity was rendered. Our goal in this paper is to improve the fairness of these games in spite of the varying network delays by equalizing the export error at the players. We explore whether the time-average of the export errors (which is the cumulative export error over a period of time averaged over the time period) at all the players can be made the same by scheduling the sending of the DR vectors appropriately at the sender. We propose two algorithms to achieve this.

Both the algorithms are based on delaying (or dropping) the sending of DR vectors to some players on a continuous basis to try and make the export error the same at all the players. At an abstract level, the algorithm delays sending DR vectors to players whose accumulated error so far in the game is smaller than others; this would mean that the export error due to this DR vector at these players will be larger than that of the other players, thereby making them the same. The goal is to make this error at least approximately equal at every DR vector with the deviation in the error becoming smaller as time progresses.

The first algorithm (which we refer to as the "scheduling algorithm") is based on "estimating" the delay between players and refining the sending of DR vectors by scheduling them to be sent to different players at different times at every DR generation point. Through an implementation of this algorithm using the open source game BZflag, we show that this algorithm makes the game very fair (we measure fairness in terms of the standard deviation of the error). The drawback of this algorithm is that it tends to push the error of all the players towards that of the player with the worst error (which is the error at the farthest player, in terms of delay, from the sender of the DR). To alleviate this effect, we propose a budget based algorithm which budgets how the DRs are sent to different players. At a high level, the algorithm is based on the idea of sending more DRs to players who are farther away from the sender compared to those who are closer. Experimental results from BZflag illustrates that the budget based algorithm follows a more balanced approach. It improves the fairness of the game but at the same time does so without pushing up the mean error of the players thereby maintaining the accuracy of the game. In addition, the budget based algorithm is shown to achieve the same level of accuracy of game playing as the current implementation of BZflag using much less number of DR vectors.

2. PREVIOUS WORK

Earlier work on network games to deal with network latency has mostly focussed on compensation techniques for packet delay and loss [2, 3, 4]. These methods are aimed at making large delays and message loss tolerable for players but does not consider the problems that may be introduced by varying delays from the server to different players or from the players to one another. For example, the concept of *local lag* has been used in [3] where each player delays every local operation for a certain amount of time so that remote players can receive information about the local operation and execute the same operation at the about same time, thus reducing state inconsistencies. The online multi-player game MiMaze [2, 5, 6], for example, takes a static bucket synchronization approach to compensate for variable network delays. In MiMaze, each player delays all events by 100 ms regardless of whether they are generated locally or remotely. Players with a network delay larger than 100 ms simply cannot participate in the game. In general, techniques based on bucket synchronization depend on imposing a worst case delay on all the players.

There have been a few papers which have studied the problem of fairness in a distributed game by more sophisticated message delivery mechanisms. But these works [7, 8] assume the existence of a global view of the game where a game server maintains a view (or state) of the game. Players can introduce objects into the game or delete objects that are already part of the game (for example, in a first-person shooter game, by shooting down the object). These additions and deletions are communicated to the game server using "action" messages. Based on these action messages, the state of the game is changed at the game server and these changes are communicated to the players using "update" messages. Fairness is achieved by ordering the delivery of action and update messages at the game server and players respectively based on the notion of a "fair-order" which takes into account the delays between the game server and the different players. Objects that are part of the game may move but how this information is communicated to the players seems to be beyond the scope of these works. In this sense, these works are very limited in scope and may be applicable only to first-person shooter games and that too to only games where players are not part of the game.

DR vectors can be exchanged directly among the players (peer-to-peer model) or using a central server as a relay (client-server model). It has been shown in [9] that multi-player games that use DR vectors together with bucket synchronization are not cheat-proof unless additional mechanisms are put in place. Both the scheduling algorithm and the budget-based algorithm described in our paper use DR vectors and hence are not cheat-proof. For example, a receiver could skew the delay estimate at the sender to make the sender believe that the delay between the sender and the receiver is high thereby gaining undue advantage. We emphasize that the focus of this paper is on fairness without addressing the issue of cheating.

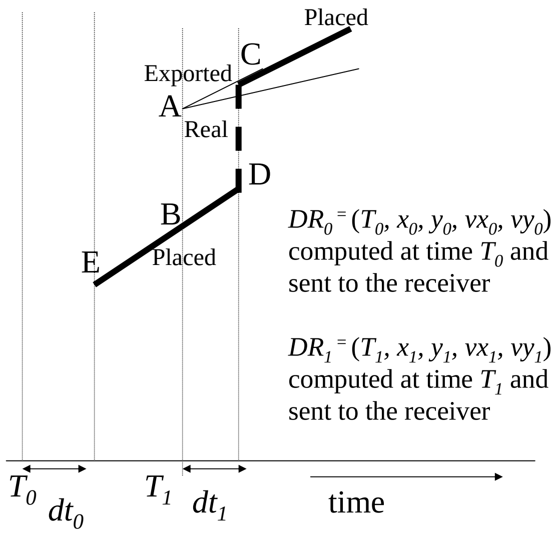
In the next section, we describe the game model that we use and illustrate how senders and receivers exchange DR vectors and how entities are rendered at the receivers based on the time-stamp augmented DR vector exchange as described in [1]. In Section 4, we describe the DR vector scheduling algorithm that aims to make the export error equal across the players with varying delays from the sender of a DR vector, followed by experimental results obtained from instrumentation of the scheduling algorithm on the open source game BZFlag. Section 5, describes the budget based algorithm that achieves improved fairness but without reducing the level accuracy of game playing. Conclusions are presented in Section 6.

3. GAME MODEL

The game architecture is based on players distributed across the Internet and exchanging DR vectors to each other. The DR vectors could either be sent directly from one player to another (peer-to-peer model) or could be sent through a game server which receives the DR vector from a player and forwards it to other players (client-server model). As mentioned before, we assume synchronized clocks among the participating players.

Each DR vector sent from one player to another specifies the trajectory of exactly one player/entity. We assume a *linear* DR vector in that the information contained in the DR vector is only enough at the receiving player to compute the trajectory and render the entity in a straight line path. Such a DR vector contains information about the starting position and velocity of the player/entity where the velocity is constant[1]. Thus, the DR vectors sent by a player specifies the current time at the player when the DR vector is computed (not the time at which this DR vector is sent to the other players as we will explain later), the current position of the player/entity in terms of the x, y, z coordinates and the velocity vector in the direction of x, y and z coordinates. Specifically, the i^{th} DR vector sent by player j about the k^{th} entity is denoted by DR^j_{ik} and is represented by the following tuple $(T^j_{ik}, x^j_{ik}, y^j_{ik}, z^j_{ik}, vx^j_{ik}, vy^j_{ik}, vz^j_{ik})$.

Without loss of generality, in the rest of the discussion, we consider a sequence of DR vectors sent by only one player and for only one entity. For simplicity, we consider a two dimensional game space rather than a three dimensional one. Hence we use DR_i to denote the i^{th} such DR vector represented as the tuple $(T_i, x_i, y_i, vx_i, vy_i)$. The receiving player computes the starting position for the entity based on x_i, y_i and the time difference between when the DR vector is received and the time T_i at which it was computed. Note that the computation of time difference is feasible since all the clocks are synchronized. The receiving player then uses the velocity components to project and render the trajectory of the entity. This trajectory is followed until a new DR vector is received which changes the position and/or velocity of the entity.



Placed
C
Exported
A
Real
D
B
E
Placed
$DR_0 = (T_0, x_0, y_0, vx_0, vy_0)$ computed at time T_0 and sent to the receiver
$DR_1 = (T_1, x_1, y_1, vx_1, vy_1)$ computed at time T_1 and sent to the receiver
T_0
dt_0
T_1
dt_1
time

Figure 1: Trajectories and deviations.

Based on this model, Figure 1 illustrates the sending and receiving of DR vectors and the different errors that are encountered. The figure shows the reception of DR vectors at a player (henceforth called the *receiver*). The horizontal axis shows the time which is synchronized among all the players. The vertical axis tries to conceptually capture the two-dimensional position of an entity. Assume that at time T_0 a DR vector DR_0 is computed by the sender and immediately sent to the receiver. Assume that DR_0 is received at the receiver after a delay of dt_0 time units. The receiver computes the initial position of the entity as $(x_0 + vx_0 \times dt_0, y_0 + vy_0 \times dt_0)$ (shown as point E). The thick line EBD represents the projected and rendered trajectory at the receiver based on the velocity components vx_0 and vy_0 (placed path). At time T_1 a DR vector DR_1 is computed for the same entity and immediately sent to the receiver[2]. Assume that DR_1 is received at the receiver after a delay of dt_1 time units. When this DR vector is received, assume that the entity is at point D. A new position for the entity is computed as $(x_1 + vx_1 \times dt_1, y_1 + vy_0 \times dt_1)$ and the entity is moved to this position (point C). The velocity components vx_1 and vy_1 are used to project and render this entity further.

Let us now consider the error due to network delay. Although DR_1 was computed at time T_1 and sent to the receiver, it did not reach the receiver until time $T_1 + dt_1$. This means, although the exported path based on DR_1 at the sender at time T_1 is the trajectory AC, until time $T_1 + dt_1$, at the receiver, this entity was being rendered at trajectory BD based on DR_0. Only at time $T_1 + dt_1$ did the entity get moved to point C from which point onwards the exported and the placed paths are the same. The deviation between the exported and placed paths creates an error component which we refer to as the export error. A way to represent the export error is to compute the integral of the distance between the two trajectories over the time when they are out of sync. We represent the integral of the distances between the placed and exported paths due to some DR DR_i over a time interval $[t_1, t_2]$ as $Err(DR_i, t_1, t_2)$. In the figure, the export error due to DR_1 is computed as the integral of the distance between the trajectories AC and BD over the time interval $[T_1, T_1 + dt_1]$. Note that there could be other ways of representing this error as well, but in this paper, we use the integral of the distance between the two trajectories as a measure of the export error. Note that there would have been an export error created due to the reception of DR_0 at which time the placed path would have been based on a previous DR vector. This is not shown in the figure but it serves to remind the reader that the export error is cumulative when a sequence of DR vectors are received. Starting from time T_1 onwards, there is a deviation between the real and the exported paths. As we discussed earlier, this export error is unavoidable.

The above figure and example illustrates one receiver only. But in reality, DR vectors DR_0 and DR_1 are sent by the sender to all the participating players. Each of these players receives DR_0 and DR_1 after varying delays thereby creating different export error values at different players. The goal of the DR vector scheduling algorithm to be described in the next section is to make this (cumulative) export error equal at every player independently for each of the entities that make up the game.

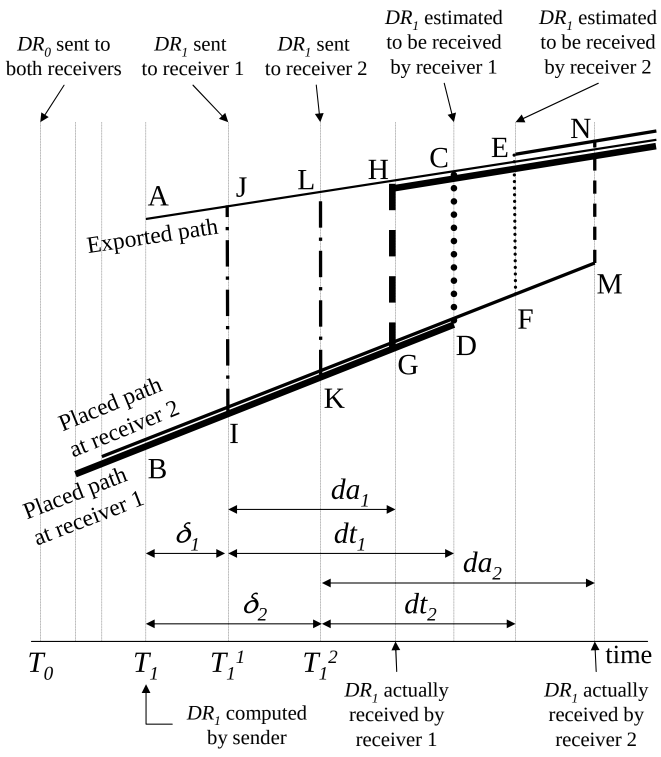
4. SCHEDULING ALGORITHM FOR SENDING DR VECTORS

In Section 3 we showed how delay from the sender of a new DR

[1]Other type of DR vectors include quadratic DR vectors which specify the acceleration of the entity and cubic spline DR vectors that consider the starting position and velocity and the ending position and velocity of the entity.

[2]Normally, DR vectors are not computed on a periodic basis but on an on-demand basis where the decision to compute a new DR vector is based on some threshold being exceeded on the deviation between the real path and the path exported by the previous DR vector.

vector to the receiver of the DR vector could lead to export error because of the deviation of the placed path from the exported path at the receiver until this new DR vector is received. We also mentioned that the goal of the DR vector scheduling algorithm is to make the export error "equal" at all receivers over a period of time. Since the game is played in a distributed environment, it makes sense for the sender of an entity to keep track of all the errors at the receivers and try to make them equal. However, the sender cannot know the actual error at a receiver till it gets some information regarding the error back from the receiver. Our algorithm estimates the error to compute a schedule to send DR vectors to the receivers and corrects the error when it gets feedbacks from the receivers. In this section we provide motivations for the algorithm and describe the steps it goes through. Throughout this section, we will use the following example to illustrate the algorithm.



DR_0 sent to both receivers
DR_1 sent to receiver 1
DR_1 sent to receiver 2
DR_1 estimated to be received by receiver 1
DR_1 estimated to be received by receiver 2
A
J
L
H
C
E
N
Exported path
M
F
D
G
K
I
B
Placed path at receiver 2
Placed path at receiver 1
da_1
δ_1
dt_1
da_2
δ_2
dt_2
T_0
T_1
T_1^1
T_1^2
time
DR_1 computed by sender
DR_1 actually received by receiver 1
DR_1 actually received by receiver 2

Figure 2: DR vector flow between a sender and two receivers and the evolution of estimated and actual placed paths at the receivers. $DR_0 = (T_0, T_0, x_0, y_0, vx_0, vy_0)$**, sent at time** T_0 **to both receivers.** $DR_1 = (T_1, T_1^1, x_1, y_1, vx_1, vy_1)$ **sent at time** $T_1^1 = T_1 + \delta_1$ **to receiver 1 and** $DR_1 = (T_1, T_1^2, x_1, y_1, vx_1, vy_1)$ **sent at time** $T_1^2 = T_1 + \delta_2$ **to receiver 2.**

Consider the example in Figure 2. The figure shows a single sender sending DR vectors for an entity to two different receivers 1 and 2. DR_0 computed at T_0 is sent and received by the receivers sometime between T_0 and T_1 at which time they move the location of the entity to match the exported path. Thus, the path of the entity is shown only from the point where the placed path matches the exported path for DR_0. Now consider DR_1. At time T_1, DR_1 is computed by the sender but assume that it is not immediately sent to the receivers and is only sent after time δ_1 to receiver 1 (at time $T_1^1 = T_1 + \delta_1$) and after time δ_2 to receiver 2 (at time $T_1^2 = T_1 + \delta_2$). Note that the sender includes the sending timestamp with the DR vector as shown in the figure. Assume that the sender estimates (it will be clear shortly why the sender has to estimate the delay) that after a delay of dt_1, receiver 1 will receive it, will use the coordinate and velocity parameters to compute the entity's current location and move it there (point C) and from this time onwards, the exported and the placed paths will become the same. However, in reality, receiver 1 receives DR_1 after a delay of da_1 (which is less than sender's estimates of dt_1), and moves the corresponding entity to point H. Similarly, the sender estimates that after a delay of dt_2, receiver 2 will receive DR_1, will compute the current location of the entity and move it to that point (point E), while in reality it receives DR_1 after a delay of $da_2 > dt_2$ and moves the entity to point N. The other points shown on the placed and exported paths will be used later in the discussion to describe different error components.

4.1 Computation of Relative Export Error

Referring back to the discussion from Section 3, from the sender's perspective, the export error at receiver 1 due to DR_1 is given by $Err(DR_1, T_1, T_1 + \delta_1 + dt_1)$ (the integral of the distance between the trajectories AC and DB over the time interval $[T_1, T_1 + \delta_1 + dt_1]$) of Figure 2. This is due to the fact that the sender uses the estimated delay dt_1 to compute this error. Similarly, the export error from the sender's perspective at received 2 due to DR_1 is given by $Err(DR_1, T_1, T_1 + \delta_2 + dt_2)$ (the integral of the distance between the trajectories AE and DF over the time interval $[T_1, T_1 + \delta_2 + dt_2]$). Note that the above errors from the sender's perspective are only estimates. In reality, the export error will be either smaller or larger than the estimated value, based on whether the delay estimate was larger or smaller than the actual delay that DR_1 experienced. This difference between the estimated and the actual export error is the *relative export error* (which could either be positive or negative) which occurs for every DR vector that is sent and is accumulated at the sender.

The concept of relative export error is illustrated in Figure 2. Since the actual delay to receiver 1 is da_1, the export error induced by DR_1 at receiver 1 is $Err(DR_1, T_1, T_1 + \delta_1 + da_1)$. This means, there is an error in the estimated export error and the sender can compute this error only after it gets a feedback from the receiver about the actual delay for the delivery of DR_1, i.e., the value of da_1. We propose that once receiver 1 receives DR_1, it sends the value of da_1 back to the sender. The receiver can compute this information as it knows the time at which DR_1 was sent ($T_1^1 = T_1 + \delta_1$, which is appended to the DR vector as shown in Figure 2) and the local receiving time (which is synchronized with the sender's clock). Therefore, the sender computes the relative export error for receiver 1, represented using R_1 as

$$\begin{aligned} R_1 &= Err(DR_1, T_1, T_1 + \delta_1 + dt_1) \\ &\quad - Err(DR_1, T_1, T_1 + \delta_1 + da_1) \\ &= Err(DR_1, T_1 + \delta_1 + dt_1, T_1 + \delta_1 + da_1) \end{aligned}$$

Similarly the relative export error for receiver 2 is computed as

$$\begin{aligned} R_2 &= Err(DR_1, T_1, T_1 + \delta_2 + dt_2) \\ &\quad - Err(DR_1, T_1, T_1 + \delta_2 + da_2) \\ &= Err(DR_1, T_1 + \delta_2 + dt_2, T_1 + \delta_2 + da_2) \end{aligned}$$

Note that $R_1 > 0$ as $da_1 < dt_1$, and $R_2 < 0$ as $da_2 > dt_2$. Relative export errors are computed by the sender as and when it receives the feedback from the receivers. This example shows the relative export error values *after* DR_1 is sent and the corresponding feedbacks are received.

4.2 Equalization of Error Among Receivers

We now explain what we mean by making the errors "equal" at all the receivers and how this can be achieved. As stated before the sender keeps estimates of the delays to the receivers, dt_1 and dt_2 in the example of Figure 2. This says that at time T_1 when DR_1 is computed, the sender already knows how long it may take messages carrying this DR vector to reach the receivers. The sender uses this information to compute the export errors, which are $Err(DR_1, T_1, T_1 + \delta_1 + dt_1)$ and $Err(DR_1, T_1, T_1 + \delta_2 + dt_2)$ for receivers 1 and 2, respectively. Note that the areas of these error components are a function of δ_1 and δ_2 as well as the network delays dt_1 and dt_2. If we are to make the exports errors due to DR_1 the same at both receivers, the sender needs to choose δ_1 and δ_2 such that

$$Err(DR_1, T_1, T_1 + \delta_1 + dt_1) = Err(DR_1, T_1, T_1 + \delta_2 + dt_2).$$

But when T_1 was computed there could already have been accumulated relative export errors due to previous DR vectors (DR_0 and the ones before). Let us represent the accumulated relative error up to DR_i for receiver j as R^i_j. To accommodate these accumulated relative errors, the sender should now choose δ_1 and δ_2 such that

$$\begin{aligned} R^0_1 + Err(DR_1, T_1, T_1 + \delta_1 + dt_1) = \\ R^0_2 + Err(DR_1, T_1, T_1 + \delta_2 + dt_2) \end{aligned}$$

The δ_i determines the scheduling instant of the DR vector at the sender for receiver i. This method of computation of δ's ensures that the *accumulated export error* (i.e., total actual error) for each receiver equalizes at the transmission of each DR vector.

In order to establish this, assume that the feedback for DR vector D_i from a receiver comes to the sender before schedule for D_{i+1} is computed. Let S^i_m and A^i_m denote the estimated error for receiver m used for computing schedule for D_i and accumulated error for receiver m computed after receiving feedback for D_i, respectively. Then $R^i_m = A^i_m - S^i_m$. In order to compute the schedule instances (i.e., δ's) for D_i, for any pair of receivers m and n, we do $R^{i-1}_m + S^i_m = R^{i-1}_n + S^i_n$. The following theorem establishes the fact that the accumulated export error is equalized at every scheduling instant.

THEOREM 4.1. *When the schedule instances for sending D_i are computed for any pair of receivers m and n, the following condition is satisfied:*

$$\sum_{k=1}^{i-1} A^k_m + S^i_m = \sum_{k=1}^{i-1} A^k_n + S^i_n.$$

Proof: By induction. Assume that the premise holds for some i. We show that it holds for $i+1$. The base case for $i = 1$ holds since initially $R^0_m = R^0_n = 0$, and the $S^1_m = S^1_n$ is used to compute the scheduling instances.

In order to compute the schedule for D_{i+1}, the we first compute the relative errors as

$$R^i_m = A^i_m - S^i_m, \text{ and } R^i_n = A^i_n - S^i_n.$$

Then to compute δ's we execute

$$\begin{aligned} R^i_m + S^{i+1}_m &= R^i_n + S^{i+1}_n \\ A^i_m - S^i_m + S^{i+1}_m &= A^i_n - S^i_n + S^{i+1}_n. \end{aligned}$$

Adding the condition of the premise on both sides we get,

$$\sum_{k=1}^{i} A^k_m + S^{i+1}_m = \sum_{k=1}^{i} A^k_n + S^{i+1}_n.$$

4.3 Computation of the Export Error

Let us now consider how the export errors can be computed. From the previous section, to find δ_1 and δ_2 we need to find

$$Err(DR_1, T_1, T_1 + \delta_1 + dt_1) \text{ and } Err(DR_1, T_1, T_1 + \delta_2 + dt_2).$$

Note that the values of R^0_1 and R^0_2 are already known at the sender. Consider the computation of $Err(DR_1, T_1, T_1 + \delta_1 + dt_1)$. This is the integral of the distance between the trajectories AC due to DR_1 and BD due to DR_0. From DR_0 and DR_1, point A is $(X_1, Y_1) = (x_1, y_1)$ and point B is $(X_0, Y_0) = (x_0 + (T_1 - T_0) \times vx_0, y_0 + (T_1 - T_0) \times vy_0)$. The trajectory AC can be represented as a function of time as $(X_1(t), Y_1(t) = (X_1 + vx_1 \times t, Y_1 + vy_1 \times t)$ and the trajectory of BD can be represented as $(X_0(t), Y_0(t) = (X_0 + vx_0 \times t, Y_0 + vy_0 \times t)$.

The distance between the two trajectories as a function of time then becomes,

$$\begin{aligned} dist(t) &= \sqrt{(X_1(t) - X_0(t))^2 + (Y_1(t) - Y_0(t))^2} \\ &= \sqrt{((X_1 - X_0) + (vx_1 - vx_0)t)^2 + ((Y_1 - Y_0) + (vy_1 - vy_0)t)^2} \\ &= \sqrt{((vx_1 - vx_0)^2 + (vy_1 - vy_0)^2)t^2 + 2((X_1 - X_0)(vx_1 - vx_0) + (Y_1 - Y_0)(vy_1 - vy_0))t + (X_1 - X_0)^2 + (Y_1 - Y_0)^2} \end{aligned}$$

Let

$$\begin{aligned} a &= (vx_1 - vx_0)^2 + (vy_1 - vy_0)^2 \\ b &= 2((X_1 - X_0)(vx_1 - vx_0) \\ &\quad + (Y_1 - Y_0)(vy_1 - vy_0)) \\ c &= (X_1 - X_0)^2 + (Y_1 - Y_0)^2 \end{aligned}$$

Then $dist(t)$ can be written as

$$dist(t) = \sqrt{a \times t^2 + b \times t + c}.$$

Then $Err(DR_1, t_1, t_2)$ for some time interval $[t_1, t_2]$ becomes

$$\int_{t_1}^{t_2} dist(t)\ dt = \int_{t_1}^{t_2} \sqrt{a \times t^2 + b \times t + c}\ dt.$$

A closed form solution for the indefinite integral

$$\int \sqrt{a \times t^2 + b \times t + c}\ dt =$$

$$\frac{(2at + b)\sqrt{at^2 + bt + c}}{4a} + \frac{1}{2} \ln\Big(\frac{\frac{1}{2b} + at}{\sqrt{a}} + \sqrt{at^2 + bt + c}\Big) c \frac{1}{\sqrt{a}} - \frac{1}{8} \ln\Big(\frac{\frac{1}{2b} + at}{\sqrt{a}} + \sqrt{at^2 + bt + c}\Big) b^2 a^{-\frac{3}{2}}$$

$Err(DR_1, T_1, T_1 + \delta_1 + dt_1)$ and $Err(DR_1, T_1, T_1 + \delta_2 + dt_2)$ can then be calculated by applying the appropriate limits to the above solution. In the next section, we consider the computation of the δ's for N receivers.

4.4 Computation of Scheduling Instants

We again look at the computation of δ's by referring to Figure 2. The sender chooses δ_1 and δ_2 such that $R_1^0 + Err(DR_1, T_1, T_1 + \delta_1 + dt_1) = R_2^0 + Err(DR_1, T_1, T_1 + \delta_2 + dt_2)$. If R_1^0 and R_2^0 both are zero, then δ_1 and δ_2 should be chosen such that $Err(DR_1, T_1, T_1 + \delta_1 + dt_1) = Err(DR_1, T_1, T_1 + \delta_2 + dt_2)$. This equality will hold if $\delta_1 + dt_1 = \delta_2 + dt_2$. Thus, if there is no accumulated relative export error, all that the sender needs to do is to choose the δ's in such a way that they counteract the difference in the delay to the two receivers, so that they receive the DR vector at the same time. As discussed earlier, because the sender is not able to a priori learn the delay, there will always be an accumulated relative export error from a previous DR vector that does have to be taken into account.

To delve deeper into this, consider the computation of the export error as illustrated in the previous section. To compute the δ's we require that $R_1^0 + Err(DR_1, T_1, T_1 + \delta_1 + dt_1) = R_2^0 + Err(DR_1, T_1, T_1 + \delta_2 + dt_2)$. That is,

$$R_1^0 + \int_{T_1}^{T_1+\delta_1+dt_1} dist(t)\ \ dt = R_2^0 + \int_{T_1}^{T_1+\delta_2+dt_2} dist(t)\ \ dt.$$

That is

$$R_1^0 + \int_{T_1}^{T_1+dt_1} dist(t)\ \ dt + \int_{T_1+dt_1}^{T_1+dt_1+\delta_1} dist(t)\ \ dt =$$

$$R_2^0 + \int_{T_1}^{T_1+dt_2} dist(t)\ \ dt + \int_{T_1+dt_2}^{T_1+dt_2+\delta_2} dist(t)\ \ dt.$$

The components R_1^0, R_2^0, are already known to (or estimated by) the sender. Further, the error components $\int_{T_1}^{T_1+dt_1} dist(t)\ \ dt$ and $\int_{T_1}^{T_1+dt_2} dist(t)\ \ dt$ can be a priori computed by the sender using estimated values of dt_1 and dt_2. Let us use E_1 to denote $R_1^0 + \int_{T_1}^{T_1+dt_1} dist(t)\ \ dt$ and E_2 to denote $R_2^0 + \int_{T_1}^{T_1+dt_2} dist(t)\ \ dt$. Then, we require that

$$E_1 + \int_{T_1+dt_1}^{T_1+dt_1+\delta_1} dist(t)\ \ dt = E_2 + \int_{T_1+dt_2}^{T_1+dt_2+\delta_2} dist(t)\ \ dt.$$

Assume that $E_1 > E_2$. Then, for the above equation to hold, we require that

$$\int_{T_1+dt_1}^{T_1+dt_1+\delta_1} dist(t)\ \ dt < \int_{T_1+dt_2}^{T_1+dt_2+\delta_2} dist(t)\ \ dt.$$

To make the game as fast as possible within this framework, the δ values should be made as small as possible so that DR vectors are sent to the receivers as soon as possible subject to the fairness requirement. Given this, we would choose δ_1 to be zero and compute δ_2 from the equation

$$E_1 = E_2 + \int_{T_1+dt_2}^{T_1+dt_2+\delta_2} dist(t)\ \ dt.$$

In general, if there are N receivers $1, \ldots, N$, when a sender generates a DR vector and decides to schedule them to be sent, it first computes the E_i values for all of them from the accumulated relative export errors and estimates of delays. Then, it finds the smallest of these values. Let E_k be the smallest value. The sender makes δ_k to be zero and computes the rest of the δ's from the equality

$$E_i + \int_{T_1+dt_i}^{T_1+dt_i+\delta_i} dist(t)\, dt = E_k, \qquad \forall i\ 1 \le i \le N, i \ne k. \tag{1}$$

The δ's thus obtained gives the scheduling instants of the DR vector for the receivers.

4.5 Steps of the Scheduling Algorithm

For the purpose of the discussion below, as before let us denote the accumulated relative export error at a sender for receiver k up until DR_i to be R_k^i. Let us denote the scheduled delay at the sender before DR_i is sent to receiver k as δ_k^i. Given the above discussion, the algorithm steps are as follows:

1. The sender computes DR_i at (say) time T_i and then computes δ_k^i, and $R_k^{i-1}, \forall k, 1 \le k \le N$ based on the estimation of delays $dt_k, \forall k, 1 \le k \le N$ as per Equation (1). It schedules, DR_i to be sent to receiver k at time $T_i + \delta_k^i$.

2. The DR vectors are sent to the receivers at the scheduled times which are received after a delay of $da_k, \forall k, 1 \le k \le N$ where $da_k \le$ or $> dt_k$. The receivers send the value of da_k back to the sender (the receiver can compute this value based on the time stamps on the DR vector as described earlier).

3. The sender computes R_k^i as described earlier and illustrated in Figure 2. The sender also recomputes (using exponential averaging method similar to round-trip time estimation by TCP [10]) the estimate of delay dt_k from the new value of da_k for receiver k.

4. Go back to Step 1 to compute DR_{i+1} when it is required and follow the steps of the algorithm to schedule and send this DR vector to the receivers.

4.6 Handling Cases in Practice

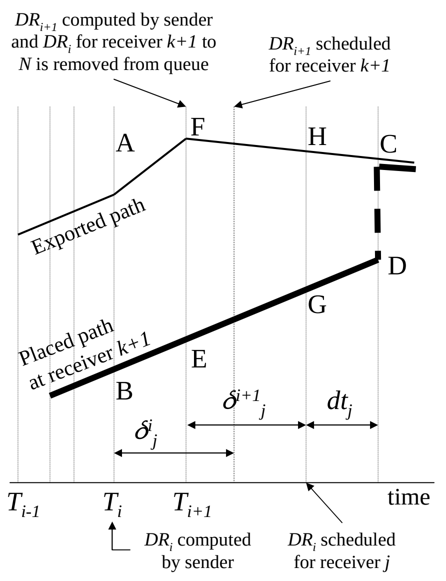
So far we implicity assumed that DR_i is sent out to all receivers before a decision is made to compute the next DR vector DR_{i+1}, and the receivers send the value of da_k corresponding to DR_i and this information reaches the sender before it computes DR_{i+1} so that it can compute R_k^{i+1} and then use it in the computation of δ_k^{i+1}. Two issues need consideration with respect to the above algorithm when it is used in practice.

- It may so happen that a new DR vector is computed even before the previous DR vector is sent out to all receivers. How will this situation be handled?

- What happens if the feedback does not arrive before DR_{i+1} is computed and scheduled to be sent?

Let us consider the first scenario. We assume that DR_i has been scheduled to be sent and the scheduling instants are such that $\delta_1^i < \delta_2^i < \cdots < \delta_N^i$. Assume that DR_{i+1} is to be computed (because the real path has deviated exceeding a threshold from the path exported by DR_i) at time T_{i+1} where $T_i + \delta_k^i < T_{i+1} < T_i + \delta_{k+1}^i$. This means, DR_i has been sent only to receivers up to k in the scheduled order. In our algorithm, in this case, the scheduled delay ordering queue is flushed which means DR_i is not sent to receivers still queued to receive it, but a new scheduling order is computed for all the receivers to send DR_{i+1}.

For those receivers who have been sent DR_i, assume for now that $da_j, 1 \le j \le k$ has been received from all receivers (the scenario where da_j has not been received will be considered as a part of the second scenario later). For these receivers, $E_j^i, 1 \le j \le k$ can be computed. For those receivers $j, k+1 \le j \le N$ to whom DR_i was not sent E_j^i does not apply. Consider a receiver $j, k+1 \le j \le N$ to whom DR_i was not sent. Refer to Figure 3. For such a receiver j, when DR_{i+1} is to be scheduled and



DR_{i+1} computed by sender and DR_i for receiver $k+1$ to N is removed from queue
DR_{i+1} scheduled for receiver $k+1$
A
F
H
C
Exported path
D
G
Placed path at receiver k+1
E
B
δ^{i+1}_j
dt_j
δ^i_j
T_{i-1}
T_i
T_{i+1}
time
DR_i computed by sender
DR_i scheduled for receiver j

Figure 3: Schedule computation when DR_i is not sent to receiver $j, k+1 \leq j \leq N$.

δ_j^{i+1} needs to be computed, the total export error is the accumulated relative export error at time T_i when schedule for DR_i was computed, plus the integral of the distance between the two trajectories AC and BD of Figure 3 over the time interval $[T_i, T_{i+1} + \delta_j^{i+1} + dt_j]$. Note that this integral is given by $Err(DR_i, T_i, T_{i+1}) + Err(DR_{i+1}, T_{i+1}, T_{i+1} + \delta_j^{i+1} + dt_j)$. Therefore, instead of E_j^i of Equation (1), we use the value $R_j^{i-1} + Err(DR_i, T_i, T_{i+1}) + Err(DR_{i+1}, T_{i+1}, T_{i+1} + \delta_j^{i+1} + dt_j)$ where R_j^{i-1} is relative export error used when the schedule for DR_i was computed.

Now consider the second scenario. Here the feedback da_k corresponding to DR_i has not arrived before DR_{i+1} is computed and scheduled. In this case, R_k^i cannot be computed. Thus, in the computation of δ_k for DR_{i+1}, this will be assumed to be zero. We do assume that a reliable mechanism is used to send da_k back to the sender. When this information arrives at a later time, R_k^i will be computed and accumulated to future relative export errors (for example R_k^{i+1} if da_k is received before DR_{i+2} is computed) and used in the computation of δ_k when a future DR vector is to be scheduled (for example DR_{i+2}).

4.7 Experimental Results

In order to evaluate the effectiveness and quantify benefits obtained through the use of the scheduling algorithm, we implemented the proposed algorithm in BZFlag (Battle Zone Flag) [11] game. It is a *first-person shooter* game where the players in teams drive tanks and move within a battle field. The aim of the players is to navigate and capture flags belonging to the other team and bring them back to their own area. The players shoot each other's tanks using "shooting bullets". The movement of the tanks as well as that of the shots are exchanged among the players using DR vectors.

We have modified the implementation of BZFlag to incorporate synchronized clocks among the players and the server and exchange time-stamps with the DR vector. We set up a testbed with four players running the instrumented version of BZFlag, with one as a sender and the rest as receivers. The scheduling approach and the base case where each DR vector was sent to all the receivers concurrently at every trigger point were implemented in the same run by tagging the DR vectors according to the type of approach used to send the DR vector. NISTNet [12] was used to introduce delays across the sender and the three receivers. Mean delays of 800ms, 500ms and 200ms were introduced between the sender and first, second and the third receiver, respectively. We introduce a variance of 100 msec (to the mean delay of each receiver) to model variability in delay. The sender logged the errors of each receiver every 100 milliseconds for both the scheduling approach and the base case. The sender also calculated the standard deviation and the mean of the accumulated export error of all the receivers every 100 milliseconds. Figure 4 plots the mean and standard deviation of the accumulated export error of all the receivers in the scheduling case against the base case. Note that the x-axis of these graphs (and the other graphs that follow) represents the system time when the snapshot of the game was taken.

Observe that the standard deviation of the error with scheduling is much lower as compared to the base case. This implies that the accumulated errors of the receivers in the scheduling case are closer to one another. This shows that the scheduling approach achieves fairness among the receivers even if they are at different distances (i.e, latencies) from the sender.

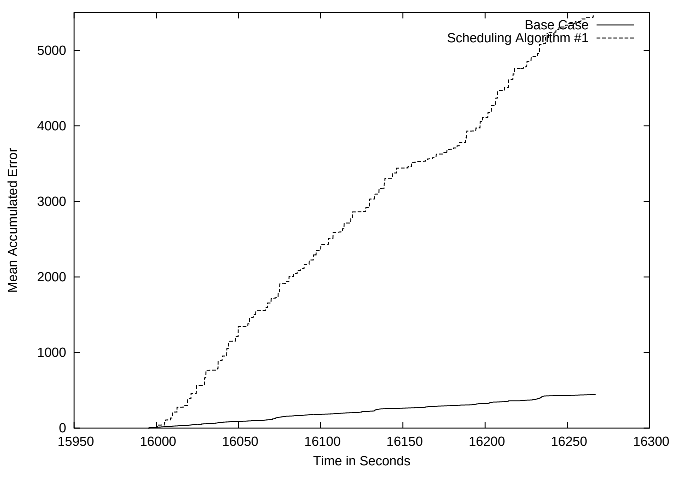
Observe that the mean of the accumulated error increased multifold with scheduling in comparison to the base case. Further exploration for the reason for the rise in the mean led to the conclusion that every time the DR vectors are scheduled in a way to equalize the total error, it pushes each receivers total error higher. Also, as the accumulated error has an estimated component, the schedule is not accurate to equalize the errors for the receivers, leading to the DR vector reaching earlier or later than the actual schedule. In either case, the error is not equalized and if the DR vector reaches late, it actually increases the error for a receiver beyond the highest accumulated error. This means that at the next trigger, this receiver will be the one with highest error and every other receiver's error will be pushed to this error value. This "flip-flop" effect leads to the increase in the accumulated error for all the receivers.

The scheduling for fairness leads to the decrease in standard deviation (i.e., increases the fairness among different players), but it comes at the cost of higher mean error, which may not be a desirable feature. This led us to explore different ways of equalizing the accumulated errors. The approach discussed in the following section is a heuristic approach based on the following idea. Using the same amount of DR vectors over time as in the base case, instead of sending the DR vectors to all the receivers at the same frequency as in the base case, if we can increase the frequency of sending the DR vectors to the receiver with higher accumulated error and decrease the frequency of sending DR vectors to the receiver with lower accumulated error, we can equalize the export error of all receivers over time. At the same time we wish to decrease the error of the receiver with the highest accumulated error in the base case (of course, this receiver would be sent more DR vectors than in the base case). We refer to such an algorithm as a *budget based algorithm*.

5. BUDGET BASED ALGORITHM

In a game, the sender of an entity sends DR vectors to all the receivers every time a threshold is crossed by the entity. Lower the threshold, more DR vectors are generated during a given time period. Since the DR vectors are sent to all the receivers and the network delay between the sender-receiver pairs cannot be avoided, the before export error [3] with the most distant player will always

[3]Note that after export error is eliminated by using synchronized clock among the players.



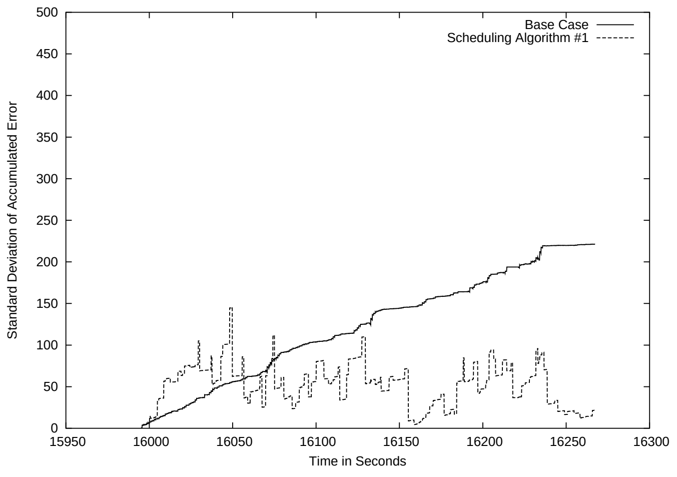


Figure 4: Mean and standard deviation of error with scheduling and without (i.e., base case).

be higher than the rest. In order to mitigate the imbalance in the error, we propose to send DR vectors selectively to different players based on the accumulated errors of these players. The budget based algorithm is based on this idea and there are two variations of it. One is a *probabilistic* budget based scheme and the other, a *deterministic* budget base scheme.

5.1 Probabilistic budget based scheme

The probabilistic budget based scheme has three main steps: a) lower the dead reckoning threshold but at the same time keep the total number of DRs sent the same as the base case, b) at every trigger, probabilistically pick a player to send the DR vector to, and c) send the DR vector to the chosen player. These steps are described below.

The lowering of DR threshold is implemented as follows. Lowering the threshold is equivalent to increasing the number of trigger points where DR vectors are generated. Suppose the threshold is such that the number of triggers caused by it in the base case is t and at each trigger n DR vectors sent by the sender, which results in a total of nt DR vectors. Our goal is to keep the total number of DR vectors sent by the sender fixed at nt, but lower the number of DR vectors sent at each trigger (i.e., do not send the DR vector to all the receivers). Let n' and t' be the number of DR vectors sent at each trigger and number of triggers respectively in the modified case. We want to ensure $n't' = nt$. Since we want to increase the number of trigger points, i.e, $t' > t$, this would mean that $n' < n$. That is, not all receivers will be sent the DR vector at every trigger.

In the probabilistic budget based scheme, at each trigger, a probability is calculated for each receiver to be sent a DR vector and only one receiver is sent the DR ($n' = 1$). This probability is based on the relative weights of the receivers' accumulated errors. That is, a receiver with a higher accumulated error will have a higher probability of being sent the DR vector. Consider that the accumulated error for three players are $a1$, $a2$ and $a3$ respectively. Then the probability of player 1 receiving the DR vector would be $\frac{a1}{a1+a2+a3}$. Similarly for the other players. Once the player is picked, the DR vector is sent to that player.

To compare the probabilistic budget based algorithm with the base case, we needed to lower the threshold for the base case (for fair comparison). As the dead reckoning threshold in the base case was already very fine, it was decided that instead of lowering the threshold, the probabilistic budget based approach would be compared against a modified base case that would use the normal threshold as the budget based algorithm but the base case was modified such that every third trigger would be actually used to send out a DR vector to all the three receivers used in our experiments. This was called as the 1/3 base case as it resulted in 1/3 number of DR vectors being sent as compared to the base case. The budget per trigger for the probability based approach was calculated as one DR vector at each trigger as compared to three DR vectors at every third trigger in the 1/3 base case; thus the two cases lead to the same number of DR vectors being sent out over time.

In order to evaluate the effectiveness of the probabilistic budget based algorithm, we instrumented the BZFlag game to use this approach. We used the same testbed consisting of one sender and three receivers with delays of 800ms, 500ms and 200ms from the sender and with low delay variance (100ms) and moderate delay variance (180ms). The results are shown in Figures 5 and 6. As mentioned earlier, the x-axis of these graphs represents the system time when the snapshot of the game was taken. Observe from the figures that the standard deviation of the accumulated error among the receivers with the probabilistic budget based algorithm is less than the 1/3 base case and the mean is a little higher than the 1/3 base case. This implies that the game is fairer as compared to the 1/3 base case at the cost of increasing the mean error by a small amount as compared to the 1/3 base case.

The increase in mean error in the probabilistic case compared to the 1/3 base case can be attributed to the fact that the even though the probabilistic approach on average sends the same number of DR vectors as the 1/3 base case, it sometimes sends DR vectors to a receiver less frequently and sometimes more frequently than the 1/3 base case due to its probabilistic nature. When a receiver does not receive a DR vector for a long time, the receiver's trajectory is more and more off of the sender's trajectory and hence the rate of buildup of the error at the receiver is higher. At times when a receiver receives DR vectors more frequently, it builds up error at a lower rate but there is no way of reversing the error that was built up when it did not receive a DR vector for a long time. This leads the receivers to build up more error in the probabilistic case as compared to the 1/3 base case where the receivers receive a DR vector almost periodically.

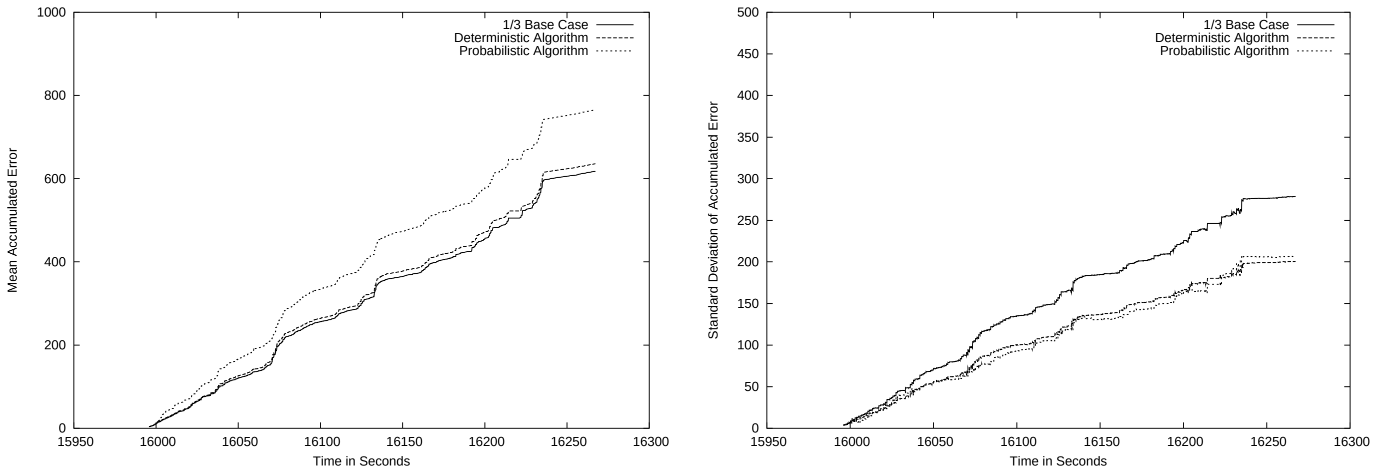


Figure 5: Mean and standard deviation of error for different algorithms (including budget based algorithms) for low delay variance.

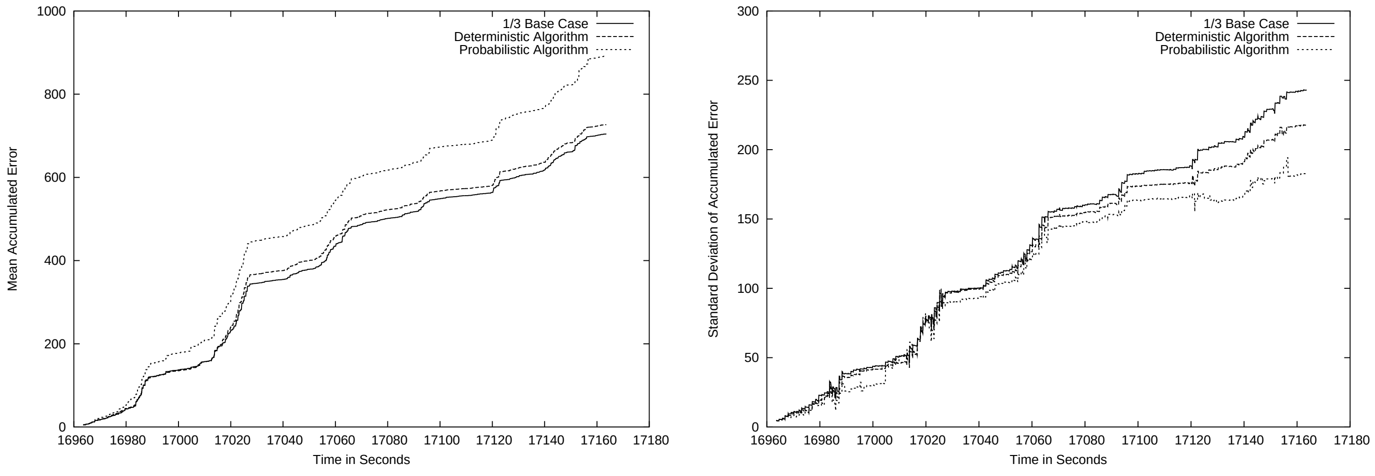


Figure 6: Mean and standard deviation of error for different algorithms (including budget based algorithms) for moderate delay variance.

5.2 Deterministic budget based scheme

To bound the increase in mean error we decided to modify the budget based algorithm to be "deterministic". The first two steps of the algorithm are the same as in the probabilistic algorithm; the trigger points are increased to lower the threshold and accumulated errors are used to compute the probability that a receiver will receiver a DR vector. Once these steps are completed, a deterministic schedule for the receiver is computed as follows:

1. If there is any receiver(s) tagged to receive a DR vector at the current trigger, the sender sends out the DR vector to the respective receiver(s). If at least one receiver was sent a DR vector, the sender calculates the probabilities of each receiver receiving a DR vector as explained before and follows steps 2 to 6, else it does not do anything.

2. For each receiver, the probability value is multiplied with the budget available at each trigger (which is set to 1 as explained below) to give the frequency of sending the DR vector to each receiver.

3. If any of the receiver's frequency after multiplying with the budget goes over 1, the receiver's frequency is set as 1 and the surplus amount is equally distributed to all the receivers by adding the amount to their existing frequencies. This process is repeated until all the receivers have a frequency of less than or equal to 1. This is due to the fact that at a trigger we cannot send more than one DR vector to the respective receiver. That will be wastage of DR vectors by sending redundant information.

4. (1/frequency) gives us the schedule at which the sender should send DR vectors to the respective receiver. Credit obtained previously (explained in step 5) if any is subtracted from the schedule. Observe that the resulting value of the schedule might not be an integer; hence, the value is rounded off by taking the ceiling of the schedule. For example, if the frequency is 1/3.5, this implies that we would like to have a DR vector sent every 3.5 triggers. However, we are constrained to send it at the 4th trigger giving us a credit of 0.5. When we do send the DR vector next time, we would be able to send it

on the 3rd trigger because of the 0.5 credit.

5. The difference between the schedule and the ceiling of the schedule is the credit that the receiver has obtained which is remembered for the future and used at the next time as explained in step 4.

6. For each of those receivers who were sent a DR vector at the current trigger, the receivers are tagged to receive the next DR vector at the trigger that happens exactly schedule (the ceiling of the schedule) number of times away from the current trigger. Observe that no other receiver's schedule is modified at this point as they all are running a schedule calculated at some previous point of time. Those schedules will be automatically modified at the trigger when they are scheduled to receive the next DR vector. At the first trigger, the sender sends the DR vector to all the receivers and uses a relative probability of 1/n for each receiver and follows the steps 2 to 6 to calculate the next schedule for each receiver in the same way as mentioned for other triggers. This algorithm ensures that every receiver has a guaranteed schedule of receiving DR vectors and hence there is no irregularity in sending the DR vector to any receiver as was observed in the budget based probabilistic algorithm.

We used the testbed described earlier (three receivers with varying delays) to evaluate the deterministic algorithm using the budget of 1 DR vector per trigger so as to use the same number of DR vectors as in the 1/3 base case. Results from our experiments are shown in Figures 5 and 6. It can be observed that the standard deviation of error in the deterministic budget based algorithm is less than the 1/3 base case and also has the same mean error as the 1/3 base case. This indicates that the deterministic algorithm is more fair than the 1/3 base case and at the same time does not increase the mean error thereby leading to a better game quality compared to the probabilistic algorithm.

In general, when comparing the deterministic approach to the probabilistic approach, we found that the mean accumulated error was always less in the deterministic approach. With respect to standard deviation of the accumulated error, we found that in the fixed or low variance cases, the deterministic approach was generally lower, but in higher variance cases, it was harder to draw conclusions as the probabilistic approach was sometimes better than the deterministic approach.

6. CONCLUSIONS AND FUTURE WORK

In distributed multi-player games played across the Internet, object and player trajectory within the game space are exchanged in terms of DR vectors. Due to the variable delay between players, these DR vectors reach different players at different times. There is unfair advantage gained by receivers who are closer to the sender of the DR as they are able to render the sender's position more accurately in real time. In this paper, we first developed a model for estimating the "error" in rendering player trajectories at the receivers. We then presented an algorithm based on scheduling the DR vectors to be sent to different players at different times thereby "equalizing" the error at different players. This algorithm is aimed at making the game fair to all players, but tends to increase the mean error of the players. To counter this effect, we presented "budget" based algorithms where the DR vectors are still scheduled to be sent at different players at different times but the algorithm balances the need for "fairness" with the requirement that the error of the "worst" case players (who are furthest from the sender) are not increased compared to the base case (where all DR vectors are sent to all players every time a DR vector is generated). We presented two variations of the budget based algorithms and through experimentation showed that the algorithms reduce the standard deviation of the error thereby making the game more fair and at the same time has comparable mean error to the base case.

7. REFERENCES

[1] S.Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in Dead-Reckoning based Distributed Multi-Player Games," *Proceedings of ACM SIGCOMM 2004 Workshop on Network and System Support for Games (NetGames 2004)*, Aug. 2004.

[2] L. Gautier and C. Diot, "Design and Evaluation of MiMaze, a Multiplayer Game on the Internet," in *Proc. of IEEE Multimedia (ICMCS'98)*, 1998.

[3] M. Mauve, "Consistency in Replicated Continuous Interactive Media," in *Proc. of the ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, 2000, pp. 181–190.

[4] S.K. Singhal and D.R. Cheriton, "Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality," *Presence: Teleoperators and Virtual Environments*, vol. 4, no. 2, pp. 169–193, 1995.

[5] C. Diot and L. Gautier, "A Distributed Architecture for Multiplayer Interactive Applications on the Internet," in *IEEE Network Magazine*, 1999, vol. 13, pp. 6–15.

[6] L. Pantel and L.C. Wolf, "On the Impact of Delay on Real-Time Multiplayer Games," in *Proc. of ACM NOSSDAV'02*, May 2002.

[7] Y. Lin, K. Guo, and S. Paul, "Sync-MS: Synchronized Messaging Service for Real-Time Multi-Player Distributed Games," in *Proc. of 10th IEEE International Conference on Network Protocols (ICNP)*, Nov 2002.

[8] K. Guo, S. Mukherjee, S. Rangarajan, and S. Paul, "A Fair Message Exchange Framework for Distributed Multi-Player Games," in *Proc. of NetGames2003*, May 2003.

[9] N. E. Baughman and B. N. Levine, "Cheat-Proof Playout for Centralized and Distributed Online Games," in *Proc. of IEEE INFOCOM'01*, April 2001.

[10] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties," in *Proc. of ACM SIGCOMM'99*, Sept. 1999.

[11] BZFlag Forum, "BZFlag Game," URL: `http://www.bzflag.org`.

[12] Nation Institute of Standards and Technology, "NIST Net," URL: `http://snad.ncsl.nist.gov/nistnet/`.