# Accuracy in Dead-Reckoning Based Distributed Multi-Player Games

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

Sarit Mukherjee Sampath Rangarajan
Center for Networking Research
Lucent Technologies Bell Laboratories, Holmdel, NJ
Email: {sarit, sampath}@lucent.com

## ABSTRACT

Distributed multi-player games use dead reckoning vectors to intimate other (at a distance) participating players about the movement of any entity by a controlling player. The dead reckoning vector contains the current position of the entity and the velocity components. When a participating player receives a vector, traditionally it puts the entity at the current position specified by the vector and starts projecting the path of the entity from that point using the local clock of the receiver. In this paper we show that this traditional method of usage of dead reckoning vector brings in inaccuracy in the receivers' rendering of the entity. This inaccuracy can be substantial even with low network delay between the sender-receiver pairs and increases with network delay. We propose the use of globally synchronized clocks among the participating players and a time-stamp augmented dead reckoning vector that enables the receiver to render the entity accurately. We modified the popular game BZFlag with this technique, and compared the accuracy seen in game playing using the traditional method and the proposed technique. We conducted several types of experiments varying the frequency of generation of dead reckoning vectors and the delay between the sender and the receivers. The experiments show significant quantitative improvement in accuracy even for 100ms delay between the sender-receiver pairs and appreciable qualitative improvement in game playing experience.

**Categories and Subject Descriptors:** C.2.4 [Distributed Systems]: Distributed Applications

**General Terms:** Design, Experimentation, Human Factors.

**Keywords:** Distributed Multi-Player Games, Accuracy, Dead-Reckoning, Clock Synchronization, Network Delay.


*SIGCOMM'04 Workshops,* Aug. 30+Sept. 3, 2004, Portland, Oregon, USA.


## 1. INTRODUCTION

In a distributed multi-player game, players are normally geographically distributed. 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 technique referred to as *Dead Reckoning* (DR) is commonly used to exchange information about player/entity movement among the players [1, 2, 3]. 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 typically 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 players that form part of the game receives such DR vectors from each other and renders the other players/entities on the local console until a new DR vector is received for that player/entity. Thus, the DR vector is a model of expected future behavior, and is used to "predict" the position of the player/entity until an updated DR vector is received. Normally, a new DR vector is sent whenever the path of the player/entity at the sender deviates from the path corresponding to the previous DR vector (say, in terms of distance in the x, y, z plane) by some amount specified by a threshold. 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 vector is used because it is almost impossible for players/entities to exchange their current positions continuously or at some very small time unit.

Because of the nature of the use of DR vectors to project the trajectory of entities, when a DR vector is received and rendered at a receiver, the original trajectory of the player/entity at the sender may have already changed. Thus, in physical time, there is a deviation at the receiving player between the real trajectory (which we refer to as *real path*) and the rendered trajectory (which we refer to as *placed path*). This is unavoidable unless every coordinate change of a player/entity is constantly sent to all the players, which is impossible; that is why DR vectors are used in the first place. As long as the message delay is negligible, a DR vector that specifies the location of a player/entity sent by a sender can be considered to have been essentially received instantaneously at all the receivers. In this sense, at any point in physical time, the location of each of the players/entities at all the player consoles is the same except for

the rendering of the entity controlled by the player herself. Furthermore, because the deviation between the *real path* and the *placed path* is considered small (note that a new DR vector is sent whenever the deviation is above a tolerance threshold) this discrepancy is acceptable and does not affect game playing.

But with distributed multi-player games involving a large number of players across the Internet, message delays between players are not negligible. When network delay is non-negligible, the trajectory of an entity that the sender expects receivers to follow in physical time may not be followed in physical time *before* and even *after* the receivers receive the DR vectors. This means, there is a deviation in physical time between the *placed path* at the receiver and the path that follows the DR vector exported by the sender (which we refer to as the *exported path*). We refer to this deviation as the *export error*).

In this paper, we explore this problem by considering the following path trajectories: **(a)** the trajectory of the player/entity at the receiver before the DR vector is received, and **(b)** the trajectory of the player/entity after the DR vector is received. We show 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 *placed* and the *exported* paths match *after* the DR vector is received; i.e, the export error after the DR vector is received is zero (we refer to this as the *after export error*). We instrumented the game BZFlag (Battle Zone Flag) [4] with this technique and use an experimental setup to perform **(a)** quantitative experiments to show the reduction in export error, and **(b)** qualitative experiments to show the improvement in game playing experience. Although we use BZFlag to illustrate our results, the idea that we intend to convey through this work is that the game playing accuracy of any distributed multi-player game can be significantly improved using synchronized clocks among players and using a global time to project the trajectory of entities. The trajectory of the player/entity at the receiver *before* the DR vector is received will follow the trajectory specified by the previous DR vector, which means there will be a non-zero *export error* before the DR vector is received (we refer to this as the *before export error*). With non-negligible network delay this is unavoidable. However, we show by experimentation that this error can be significantly reduced by the proposed technique.

Modifications to dead reckoning have been suggested by [5], [6] and [7] but not from the accuracy perspective. Previous work has considered solutions that compensate for different network delays at different players. For example, the concept of *local lag* has been used where each player delays every local operation for a certain amount of time so that remote players can receive information about the local operation (for example in the form of a DR vector) and execute the same operation at the about same time, thus reducing state inconsistencies [2]. The online multi-player game MiMaze [1, 8] takes a static bucket synchronization approach to compensate for the unfairness introduced by variable network delays. There has also been work that focuses on how to either reduce or mask player experienced response time. For client-server based first person shooter games, [9] discusses a number of latency compensating methods at the application level which are proprietary to each game. In [10, 11] synchronized message delivery techniques for client-server game architectures are discussed that aim to compensate for variable message delays from different players to the game server. As far as we are aware, there is no systematic study on the effect of network delay on the *accuracy* of distributed multi-players games that use DR vectors to exchange movement information.

In the next section, we use an example to illustrate the *before* and *after* export errors. In Section 3, quantitative results based on the modified implementation of BZFlag are used to show the reduction in export error compared to the current implementation of BZFlag. Conclusions from our work are presented in Section 4.

## 2. EXPORT ERRORS

In current implementations of multi-player games the clocks at the players are **not** synchronized and the DR vectors are generated and used as follows. Each DR vector sent from one player to another specifies the trajectory of exactly one player/entity. In the description below we assume the use of a *linear DR vector* in that the information contained in the DR vector used by the receiving player is enough 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 entity where the velocity is considered to be constant [1]. Thus, the DR vectors sent by a player specifies 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.

In the following discussion when we consider the export error, we will consider a sequence of DR vectors sent by only one player and for only one entity. We will use $DR_i$ to denote the $i^{th}$ such DR vector. This DR vector will denote the tuple $(x_i, y_i, z_i, vx_i, vy_i, vz_i)$. When this DR vector is received, the receiver will use the $x_i, y_i, z_i$ values to project the entity on the local console and then use the velocity vectors $vx_i$, $vy_i$ and $vz_i$ to continuously project and render the trajectory of the entity. This trajectory will be followed until a new DR vector is received which changes the position and/or velocity of the entity.

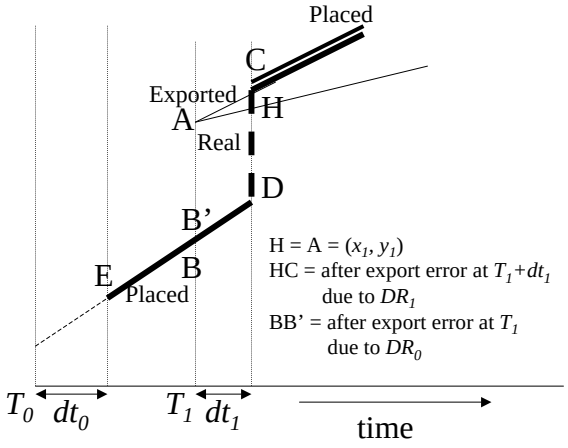


**Figure 1: Trajectories and errors.**

Based on this model, Figure 1 illustrates the sending and receiving of DR vectors and the different errors due to the deviation in the trajectory of the entity at the same physical time between the sender and the receiver. For ease of description, the DR vectors shown in the figure illustrate only two dimensions. The figure shows the reception of DR vectors at a player (henceforth called the *receiver*). The horizontal axis shows the time as per the sender's clock. Also, the sender's clock is used as the reference physical time for the

[1]Other types of DR vectors include vectors defining constant speed circular motion, quadratic DR vectors which specify the acceleration of the entity and cubic spline DR vectors that attempt to smoothly fit a path between a starting position and velocity and an ending position and velocity of the entity.

description below. The vertical axis tries to conceptually capture the two-dimensional position of an entity. Assume a traditional game environment where the player clocks are not synchronized. Assume that at time $T_0$, a DR vector $DR_0 = (x_0, y_0, vx_0, vy_0)$, 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 will place this entity at $(x_0, y_0)$ which are the coordinates specified in $DR_0$ (note that because of delay $dt_0$, this entity will be at a different position at the sender at the same physical time). This will now become the placed path at the receiver for that entity and this path $ED$ is the trajectory of the entity at the receiver. Assume that at time $T_1$ a new DR vector $DR_1 = (x_1, y_1, vx_1, vy_1)$ is computed for the same entity and immediately sent to the receiver. At this time, at the sender, the entity is at location $(x_1, y_1)$ [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 at the receiver. The entity is now moved at the receiver to $(x_1, y_1)$ which is the coordinate specified in the DR. This is shown as point H in the figure. Note that the coordinates of point A and H are the same, as the figure shows the progression in time as the X-axis (and **not** the coordinate axes). At this physical time when the entity is at point H $(x_1, y_1)$ at the receiver, the trajectory of the entity at the sender is following the same path except that the entity is at a different position (shown as point C). Point C is *ahead* in the trajectory because the entity was at point A $(x_1, y_1)$ at time $T_1$ at the sender and only at time $T_1 + dt_1$ has it been put at the same coordinate position at the receiver. This means, the placed path at the sender *leads* the placed path at the receiver although they follow the same trajectory. This error between the position of the entity at the sender (point C) and the position of the entity at the receiver (point H) is the *after export error*. Note that the trajectory of the entity due to $DR_0$ also has a similar after export error where the receiver lags the sender. At time $T_1$, the entity at the sender is at point B (from which it is moved to point A due to the computation of $DR_1$) and the entity at the receiver lags the sender and is at point B'.

Again, consider Figure 1 but now assume that the clocks are synchronized at the players. In addition to maintaining synchronized clocks, the DR vectors sent includes the times $T_0$ and $T_1$ at which they are sent. When $DR_0 = (T_0, x_0, y_0, vx_0, vy_0)$ is received, the receiver will compute 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 receiver can figure out $dt_0$ as the time difference between when it received $DR_0$ and $T_0$ (which has been appended to $DR_0$). Again, the line $ED$ represents the placed path at the receiver. When $DR_1 = (T_1, x_1, y_1, vx_1, vy_1)$ is received, a new position for the entity is computed as $(x_1 + vx_1 \times dt_1, y_1 + vy_1 \times dt_1)$ and the entity is moved to this position (point C). Again, the receiver can figure out $dt_1$ as the time difference between when it received $DR_1$ and $T_1$. The velocity components $vx_1$ and $vy_1$ are used to project and render this entity further. Note that when the entity is moved to point C at the receiver at time $T_1 + dt_1$, the position of the entity at the sender is exactly the same (that is, points H and C are the same). This means, with synchronized clocks, there is no *after export error* and the placed paths at the sender and the receiver after the DR is received at the receiver is exactly the same.

Let us now consider the *before export error*. We mentioned earlier that this error due to network delay is unavoidable even if the clocks are synchronized. 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 based on $DR_0$ at trajectory BD in case of synchronized clocks and B'D in case the clocks are not synchronized. Only at time $T_1 + dt_1$ did the entity get moved to point C in case of synchronized clocks and to point H in case the clocks are not synchronized. This is the *before export error* due to $DR_1$ (that is, the error component due to the use of $DR_0$ to render the entity at the receiver before $DR_1$ is received). A way to represent this error is to compute the integral of the distance between the two trajectories (AC and BD in case of synchronized clocks and AC and B'D in case the clocks are not synchronized) over the time that they are out of sync. Note that there would have been a *before 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.

Although even with synchronized clocks there does exist a *before export error*, in the next section, using a modified version of BZFlag and using experimental measurements from the modified implementation, we show that the total export error (which includes the *before error* and the *after error*) is significantly reduced with our new implementation that uses synchronized clocks compared to the current implementation. The modifications to the BZFlag incorporates synchronized clocks and the time-stamping of the DR vector with the sending time of the DR vector as described above.

## 3. INSTRUMENTATION OF BZFLAG AND NUMERICAL RESULTS

BZFlag (Battle Zone Flag) 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 (players) as well as that of the shots (entities) are exchanged among the players using DR vectors. The current implementation of BZFlag uses local clocks (i.e., not synchronized) for dead reckoning of the players on the screen.

We have modified the implementation of BZFlag to incorporate global clocks (i.e., synchronized) among the players and the server and exchange time-stamps with the DR vector. Our instrumentation traces the real path traversed by a player/entity as a sender of the object. A receiver logs the reception of a DR vector and renders the player/entity to account for the delay between the sender and the receiver so that a single instance of game playing generates placed paths for both global and local clocks.

We set up a testbed with four player stations. One station acts as a sender where the tank is moved, real path is traced and logged, and DR vectors are generated and sent. The other three stations act as receivers. We use NIST Net [12] to insert different but fixed amounts of delay, 100ms, 300ms and 800ms, between the three sender-receiver pairs. Each receiver gets the same DR vector from the sender, computes the placed path based on both local and global clocks, and logs the corresponding placed paths.

We conducted three different sets of experiments, each differing in the frequency at which the DR vectors are generated. In the *Linear Motion* case, the tank is navigated (by human) in a straight line thereby generating very few DR vectors. The *Circular Motion* moves the tank in a circular path. This also generates a few DR vectors as BZFlag detects circular motion and dead-reckons using

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

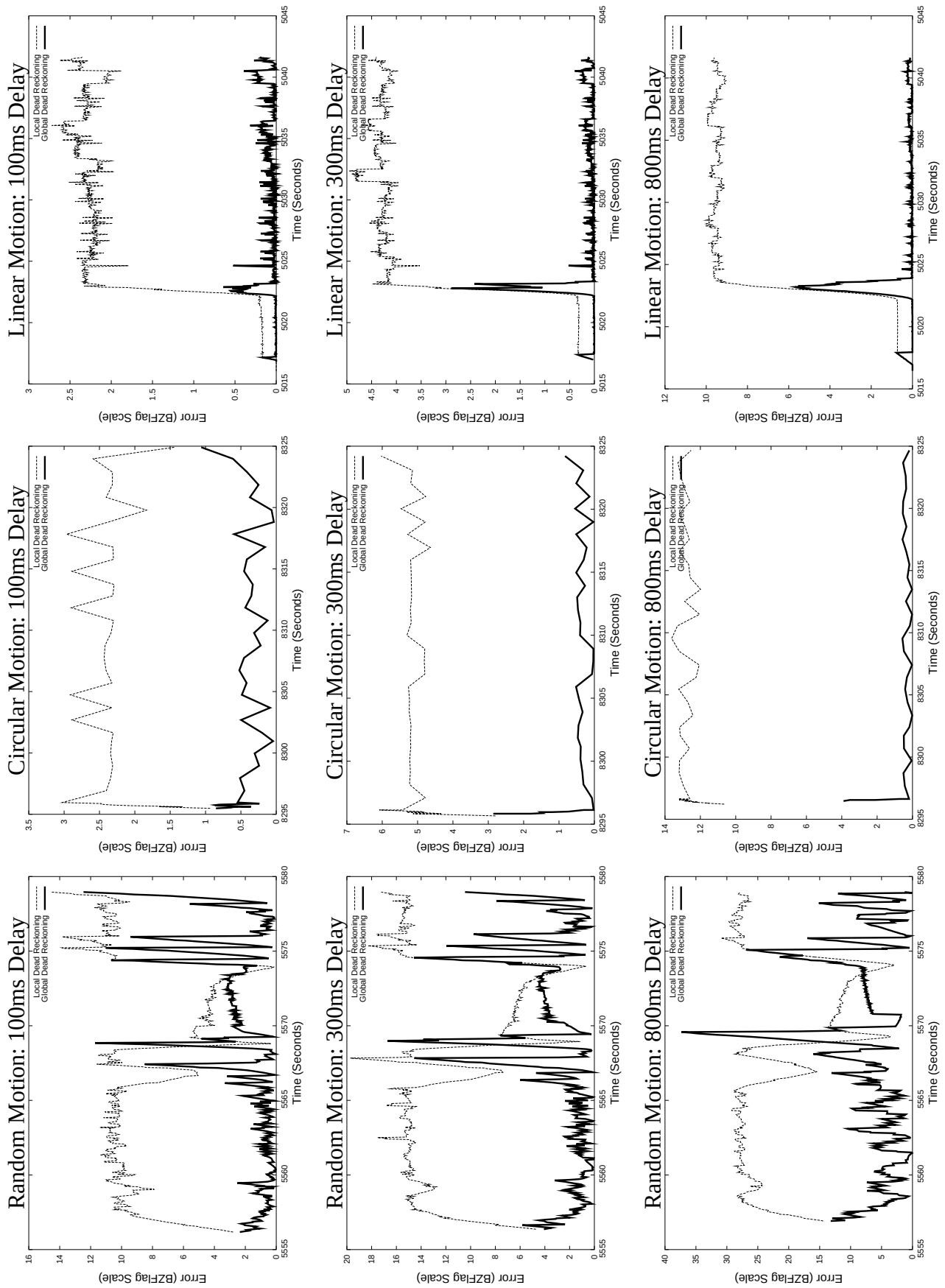


**Figure 2: Error between real and placed paths with local and global clocks.**

a circular motion DR vector. In the *Random Motion* case we move the tank in a random fashion which generates moderate to large numbers of DR vectors. In each experiment, the real and the placed paths using both local and global clocks are logged. The logs are used to compute the instantaneous error between the two paths (i.e., the distance between them at any point in time) using local and global clocks. In Figure 2 we show snapshots between 25 to 30 seconds (in the X-axis) of the entire game play. The Y-axis shows the error between the real and the placed paths in terms of distance between BZFlag coordinate point units. The size of a BZFlag tank in these units is 2.8 units wide and 6 units long. The errors due to local and global clock are henceforth referred to as the local and global errors, respectively.

For the *Linear Motion* case, frequency of generation of DR vectors is low. Due to this, the exported DR vector is rendered "correctly" before the next DR vector is generated. This leads to mostly zero global error as the entity is placed exactly where it would be in the sender's screen. As the delay between the sender and the receiver increases, the possibility of the sender generating a new DR vector before the previous one reaching and getting rendered on the receiver's screen increases. Therefore, we observe relatively higher peaks in the global error. In all cases the global error is much smaller than the local error. For example, with a delay of 300ms between the sender and the receiver (which is quite typical), the local error could be up to 2 tank units, whereas the global error is close to zero. The *Circular Motion* case is similar to the Linear Motion case except that it captures the situation with a different type of DR vector (circular motion DR).

For the *Random Motion* case, the DR vectors are generated with moderate to high frequency. For low delay most of the DR vectors reach and get rendered accurately at the receiver and the global error reaches zero quite often. As the delay increases, the sender generates the next DR vector before the previous one reaches and gets rendered; this increase the global error. There are a few instances where global error is larger than the local error. This happens when a moving player becomes stationary (i.e., a tank stops). A new DR vector indicating the stationarity is sent to all the receivers. Till it reaches the receiver, the receiver computes the placed path using the previous DR vector. With global clock, since the player is placed in the exact position, the trajectory overshoots the stationary point by the time the DR vector arrives. With local clock, the placed path always (time-)lags by the amount of delay between the sender and the receiver. So the player's path does not overshoot but become stationary after a time lag. In general the average global error is substantially less than the average local error leading to the conclusion that globally synchronized clock among the players will substantially increase the accuracy of a distributed multi-player game.

So far we have shown the improvement in accuracy provided by our technique (applied on the players only) using quantitative results. To evaluate the qualitative game playing experience, we applied our technique on the "shots" as well, and played the game between two players with different delays. We played the game several times using both local and global clocks in order to qualitatively assess the differences in the two approaches. We observed that the hit rate (due to the "shots") was better in case of global clock as compared to local clock particularly when the delay was high. This was because the player who was being dead reckoned was more accurately placed on the other players' screen and hence the other player was able to hit her where she could see her as compared to using local clock where the other player would miss her if she tried to hit her where she saw her.

# 4. CONCLUSIONS

This paper discussed the use of synchronized clocks in distributed multi-player games to improve the accuracy of game playing. We showed that the current technique of using a DR vector without any timing information to render an entity at a player brings in inaccuracy in game playing and this inaccuracy is substantial even with low network delay between the sender-receiver pairs. We proposed the use of globally synchronized clocks among the participating players and a time-stamp augmented dead reckoning vector that enables the receiver to render the entity accurately. Using BZFlag as an example, we showed through experiments that the accuracy of game playing using our technique that uses synchronized clocks among the players is much improved compared to the current implementation. In general our results are intended to show that globally synchronized clocks among players should improve game playing accuracy for any distributed multi-player game.

**Acknowledgments:** We would like to thank the developers of BZFlag and especially Tim Riker, Sean Morrison and Tupone Alfredo for their help and insights. We would also like to thank Vikram Aggarwal for help with Perl scripts used for data analysis.

# 5. REFERENCES

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

[2] 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.

[3] 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.

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

[5] Z. B. Simpson, "A Stream Based Time Synchronization Technique for Networked Computer Games," URL: `http://www.mine-control.com/zack/timesync/timesync.html`.

[6] URL: `http://p24.bakadigital.com/p24bb/viewtopic.php?t=14`.

[7] L. Pantel and L. Wolf, "On The Suitability of Dead Reckoning Schemes for Games," in *Proc. of NetGames2002*, 2002.

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

[9] Y. W. Bernier, "Latency Compensation Methods in Client/Server In-game Protocol Design and Optimization," in *Proc. of Game Developers Conference'01*, 2001, URL: `http://www.gdconf.com/archives/proceedings/2001/prog_papers.html`.

[10] 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.

[11] 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.

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