Play Patterns for Path Prediction in Multiplayer Online Games

Jacob Agar
School of Computer Science
Carleton University
Ottawa, Canada
Email: jakeagar@gmail.com

Jean-Pierre Corriveau
School of Computer Science
Carleton University
Ottawa, Canada
Email: jeanpier@scs.carleton.ca

Wei Shi
Faculty of Business and I.T.
University of Ontario Institute of Technology
Oshawa, Canada
Email: wei_shi@scs.carleton.ca

***Abstract*—Traditional dead reckoning schemes predict a player's position by assuming that players move with constant force or velocity. However, because player movement is rarely linear in nature, using linear prediction fails to produce an accurate result. Among existing dead reckoning methods, only few focus on improving prediction accuracy via genuinely non-traditional methods for predicting the path of a player. In this paper, we propose a new prediction method based on *play patterns*. We implemented a 2D top-down multiplayer online game to act as a test harness that we used to collect play data from 44 experienced players. From the data for half of these players, we extracted play patterns, which we used to create our dead reckoning algorithm. A comparative evaluation proceeding from an extensive set of simulations (using the other half of our play data) suggests that our *EKB* algorithm yields more accurate predictions than the IEEE standard dead reckoning algorithm and the recent "Interest Scheme" algorithm.**

I. Introduction

The IEEE standard 1278.1 defines a Distributed Interactive Simulation (DIS) as an infrastructure that links simulations of various types at multiple locations to create realistic, complex, virtual worlds for the simulation of highly interactive activities. DIS exercises are intended to support a mixture of virtual entities with computer controlled behavior (computer generated forces), virtual entities with live operators (human in-the-loop simulators), live entities (operational platforms and test and evaluation systems), and constructive entities (wargames and other automated simulations) [1].

Data messages, known as protocol data units (PDUs), are exchanged on a network between simulation applications. Delay and loss of PDUs are the two major issues facing DISs. *Delay* (or network latency) refers to the time it takes for packets of PDUs to travel from sender to receiver. This delay is usually said to be caused by the time it takes for a signal to propagate through a given medium, plus the time it takes to route the signal through routers. *Jitter* is a term used as a measure of the variability over time of delay across the network [2]. *Loss* (often higher when delay is higher) refers to lost network packets as a result of signal degradation over a network medium, as well as rejected packets and congestion at a given network node. These two factors cause a DIS to suffer from a lack of consistency between remote participants, jittery movement of various entities and a general loss of accuracy in the simulation.

A method of position/orientation estimation called *dead reckoning* is employed a) to minimize the appearance of *delay* and *loss* and b) to minimize network traffic. The former is achieved by having each simulation application maintain a model (from dead reckoning) of the position and possibly the orientation of all entities that are of interest (e.g., within some visibility range). The predicted position/orientation of other entities are used to display their position/orientation in an entity's visual or sensor displays. An entity corrects its dead reckoning model and replaces the estimation with the most recent position, velocity, and acceleration information, upon receiving a new update from one of the entities it is dead reckoning. Smoothing techniques may be used to eliminate *jitter* that may occur in a visual display when the dead reckoned position/orientation of an entity is corrected to the most recently communicated position/orientation [1]. This is achieved by estimating the position/orientation of other entities. With the estimated position, it is not necessary to receive an update of a position and orientation that occurs in the entity's trajectory over time. An update is required only when a change in position and orientation differs by a pre-specified amount (threshold) from the dead reckoned position and orientation.

DISs are used by military, space exploration, medicine organizations. In recent years, another type of DIS, namely Multiplayer Online Games (MOGs), has come to make up a huge chunk of one of the largest entertainment industries on the planet. But dealing with a MOG raises many architectural problems that do not exist in a traditional video game. In particular, players playing in geographical locations thousands of kilometers away from each other need to have their actions appear to be executed in the same virtual space. This can be problematic because there are delays in message sending over large distances, as well as message loss. Delay and loss entail discrepancies between the simulated environment of each player, resulting in unfair scenarios between players, and incorrect perception of events. For example, when a packet is late or lost, it causes objects in the scene to be rendered at out of date or incorrect locations. If an object is simply rendered at its latest known position, then this object's movement is, as a result, jittery and sporadic. Regardless of the internet problems relevant to MOGs, the player still demands the same

978-1-4673-2699-5/12/$31.00 © 2012 IEEE

playing experience s/he would get from a locally played game. In order to maintain playability within video games that are played in distributed fashion over the internet, there is a great need for prediction algorithms to maintain accuracy at high levels of lag (reaching up to three seconds of packet Round Trip Time). In turn, such a requirement suggests the design of a more sophisticated dead reckoning scheme to handle the internet quality problem.

II. Related Work and Our Contribution

Traditional prediction schemes forecast a player's position by assuming each player moves using constant force or velocity. However, because player movement is rarely linear in nature, using linear prediction fails to maintain an accurate result. Furthermore, Pantel et al. explore and discuss the suitability of different prediction methods within the context of different types of video games [3]. They conclude that some prediction schemes are better suited to some types of games than to others. More specifically, they look at five traditional prediction schemes: constant velocity, constant acceleration, constant input position, constant input velocity, and constant input acceleration. Each prediction scheme is compared to the others in the context of a sports game, a racing game, and an action game. As a result of the evaluation of these different prediction methods in each game, these authors demonstrate that different prediction schemes are better suited to different types of games. For example, it is shown that predicting with a constant input velocity is best suited to sports games; a constant input acceleration is best for action games; and predicting with constant acceleration is best suited to racing games [3]. For action games, constant velocity and constant input position predictions also offer a relatively low prediction error [3].

Among existing dead reckoning methods, only a few focus on improving prediction accuracy via genuinely new (i.e., non traditional) methods for predicting the path of a player:

Traditionally, dead reckoning algorithms dictate that the server should send a positional update to clients when an object strays from its predicted path by some threshold. Duncan et al. [4] propose a method, called the Pre-Reckoning scheme, that sends an update just before it is anticipated that an object will exceed some threshold. To anticipate a threshold change, the angle between the current movement and the last movement is analyzed. If this angle is large enough, it is assumed that the threshold will be crossed very soon, and a dead reckoning packet is sent. The Pre-Reckoning algorithm yields better results when variability in player movement is low.

Cai et al. [5] present an auto-adaptive dead reckoning algorithm that uses a dynamic threshold to control the extrapolation errors in order to reduce the number of update packets. The results suggest a considerable reduction in (the number of) update packets without sacrificing accuracy in extrapolation. While having a dynamic threshold for predicting objects does result in less data needing to be sent over the network, it does not eliminate the requirement for increasingly accurate prediction schemes. A dynamic threshold allows farther away objects to not require a high a degree of accuracy, but regardless, closer objects still need to be predicted accurately. Furthermore, the method outlined in [5] assumes a perspective view on the world, such that farther away objects are smaller and less visible. However, in a 2D video game, in which an orthographic view is utilized, all objects in view are of normal size, and therefore almost all of the objects are of interest to the user.

Work has also been done in using neural networks to enhance the accuracy of dead reckoning [6], [7]. In [6], McCoy et al. propose an approach that requires each controlling host to rely on a bank of neural network predictors trained to predict future changes in an object's velocity. Conversely, the approach proposed by Hakiri et al. in [7] is based on a fuzzy inference system trained by a learning algorithm derived from neural networks. This method does reduce network loads. While these methods have been shown to improve performance of dead reckoning, we will not consider here any approach that requires training and imposes extra computation on each host prior to the launching of a game.

Delaney et al. describe a hybrid predictive contract technique, which chooses either the deterministic dead reckoning model or a statistically based model. The claim of these authors is that their approach results in a more accurate representation of the movement of an entity and a consequent reduction in the number of packets that must be communicated to track that movement remotely. The statistical model rests on repeatedly observing players race to a same goal location in order to determine the most common path used. In turn, this path is used to predict the path of a player towards the same goal location.

Finally, Li et al. propose a method called the "Interest Scheme" [9], [10] for predicting the location of a player-controlled object. That approach shows an increased accuracy of path prediction beyond traditional dead reckoning models specifically in a 2D tank game, with levels of lag up to 3000 ms. The strength of the Interest Scheme lies in the way it uses the surrounding entities of a given player-controlled entity to better predict what actions the user will take. The method works on the assumption that a player's directional input is affected by its surroundings, such as items and enemy players. Due to the introduction of an extra computational burden, especially when network conditions are adequate for play, a hybrid method is introduced into the "Interest Scheme". This method involves using traditional dead reckoning algorithms up until a fixed threshold of prediction time. However, "Interest Scheme" is designed for one very specific type of game. Thus, unfortunately, the success of the "Interest Scheme" is not reproducible in a traditional team-based action game, wherein players can move in all directions freely (in contrast to a tank limited to the forward vector, that is, a tank that cannot "strafe" to the left and right of the forward vector, but instead has to rotate to change direction).

In light of these limitations and issues observed in existing work on dead reckoning, we will propose in the rest of this

paper a prediction scheme that takes user play patterns into account. We do so in the context of traditional and typical team-based action games, such as first-person, third-person or top-down shooters. We choose such a context because a player's movement is highly unpredictable in it, and is therefore highly prone to prediction inaccuracies, thus emphasizing the need for a better prediction scheme. Our work can be summarized as follows:

1) We first implemented a 2D top-down multiplayer online game entitled "Ethereal" to act as our test harness. This test harness allows us to record all data (including raw input data from all participants) during each play testing session. Beyond keyboard and mouse input, all world or environment variables are also recorded, such as game object and item positioning, world geometry information, and game events.
2) We then collected data by having 44 experienced players play Ethereal over three play testing sessions. We arbitrarily divided our players into two sets of equal size:
   - group A: those used to create our proposed dead reckoning algorithm
   - group B: those used to evaluate this algorithm against other algorithms

   More specifically, by observing players of group A play and from the subsequent analysis of the data collected from the games of these players, we identified a series of typical player behaviors, which we call *play patterns*. We then used these play patterns to create our *EKB* (Experience Knows Best) algorithm for dead reckoning.
3) Using the key ability of our test harness to playback to itself actual player inputs from the collected data, we played back the games of group B, varying from one experiment to the next the network latency, loss and jitter. We repeated each experiment with each of the three algorithms we considered, namely: our EKB algorithm, the IEEE standard dead reckoning algorithm [1] and the "Interest Scheme" (IS) [9], [10] algorithm.
4) In the end, this allowed us to develop a detailed comparative evaluation of these three algorithms. Our results are reported in section V.

We now elaborate on our proposed algorithm.

III. The EKB Algorithm

A. Main Theoretical Component: Forces

Our approach involves predicting a player's position by predicting the potential behaviours that a player may adopt. To do so, we first collected data from having 44 experienced players play Ethereal over three play testing sessions. From half of this data, we identified several behaviors that are deemed to affect the player's next movement. These behaviours each take the form of a force that is exerted on the players, affecting where they will be located next. These behaviour forces are applied at different strength levels depending on what is occurring in the game from the point of view of the player at hand. These forces are based on the positions and states of other objects in the scene. Forces are applied as either an attraction or repulsion force towards or away from a given position in space. The strength of these forces depends on several factors such as the distance to the object and the strength or weakness of the player. We first separate a player's behaviour into two states: *in battle* and *out of battle*. We then exert different forces based on a player's current state. The following are the forces employed in our work: the *follow* force, the *bravery* force, and the *align* force.

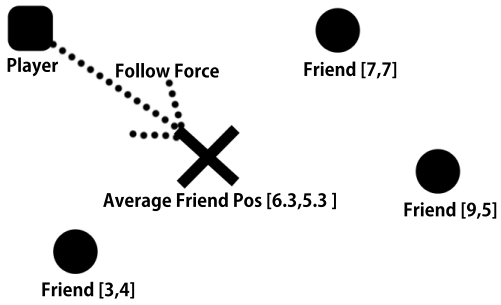

Fig. 1. Follow force

Each force takes into account other players in the game world in order to determine direction and magnitude. They do so only if a given player is within a specified static distance threshold. In our current experiments, we set this distance to the size of the screen. Any player outside of such a region of interest is not considered in the computing of a force.

1) Follow: The follow force arises from our observation that a player tends to move towards and group up with friendly players (e.g., other teammates). It is computed by taking the average position of all friendly players within a specified radius, and having the player move towards that location. Furthermore, the speed of differentiation of this force does not depend on the distance of other players but is instead always set to the maximum speed of the player. From our observations, the follow force is the most important force to exert on a player. This is because, in multiplayer online games, a player's actions generally proceed from a team-based strategy.

$$\vec{E}_{f(or\ e)} = \frac{\sum_{i=1}^{i=n} \vec{P}_{f(or\ e)}}{n} \tag{1}$$

$$\vec{F}_{follow} = \frac{\vec{E_f} - \vec{C}}{|\vec{E_f} - \vec{C}|} \times S \tag{2}$$

Equation 1 calculates the averaged position of all friendly (f) (or enemy e) players within a given threshold, where $\vec{P}_{f(or\ e)}$ is the position of a friend (or enemy), and n is the number of players of that type within a given specified static distance threshold. The averaged position of all enemy players is used for the Bravery force. Equation 2 represents the force from the player to the average position of all friendly players, where $\vec{C}$ is the current position of the player. S is the maximum speed of the player. The follow force is illustrated in Figure 1

2) Align: The align force arises from a player's tendency to travel in a group with friendly players. The align force takes into account all friendly players' velocities within a specified radius. The closer a friendly player is, the more weight it carries in affecting the direction of the align force. The align force's magnitude is a function of how many friendly players are within the specified radius, and how close they are.

$$D_f = \begin{cases} D_{f_{MAX}} & if \quad D_f \geq D_{f_{MAX}} \\ D_{f_{MIN}} & if \quad D_f \leq D_{f_{MIN}} \end{cases} \tag{3}$$

$$\vec{F}_{align} = \sum_{i=1}^{i=n} (\vec{V}_f \times (1 - \frac{D_f - D_{f_{MIN}}}{D_{f_{MAX}} - D_{f_{MIN}}})) \quad (|\vec{F}_{align}| \leq S) \tag{4}$$

Equation 4 outlines the align force. D_f, $\vec{V}_f$, $D_{f_{MIN}}$, $D_{f_{MAX}}$ represent the distance to the friendly player, the velocity of the friendly player, the minimum distance to consider for friendly players, and the maximum distance to consider for friendly players respectively.$D_{f_{MIN}}$, $D_{f_{MAX}}$ are each set to a specified static distance threshold.

3) Bravery: The bravery force arises from the observed behaviour of a player's tendency to fall back when outnumbered by the enemy and the tendency to advance on the enemy while winning. To obtain the bravery force, the total strength of all nearby friends and the total strength of all nearby enemies is calculated. The strength of each team is calculated by adding up the health and ammo values of each player. The relative strength of the friendly army versus the enemy army determines the direction of the resulting force. If the friendly army is stronger, the bravery force is positive, namely, towards the enemy. Otherwise it is negative, consequently, away from the enemy forces. The higher the magnitude of the force, the farther the player will move away or towards the enemy.

$$I_{f,e,c} = \frac{H_{f,e,c}}{MH_{f,e,c}} + \frac{A_{f,e,c}}{MA_{f,e,c}} \tag{5}$$

$$Z_{f,e} = \sum_{i=1}^{i=n} I_{f,e} \tag{6}$$

$$\vec{V}_b = (\vec{E}_f + (\frac{(\vec{E}_e - \vec{E}_f)}{|(\vec{E}_e - \vec{E}_f)|} \times (u \times \frac{(kZ_f + I_c) - Z_e}{max((kZ_f + I_c), Z_e)}))) - \vec{C} \tag{7}$$

$$\vec{F}_{bravery} = \frac{\vec{V}_b}{|\vec{V}_b|} \times S \tag{8}$$

In equation 5, $I_{f,e,c}$ is the influence of a friend, enemy or the current player in terms of its strength. H, MH, A and MA are the health, maximum health, ammo value and maximum ammo of the player respectively. This influence value is then combined into either the enemy or friendly team influence value, depending on which team the player is, represented by $Z_{f,e}$ in equation 6. $Z_{f,e}$ is made up of all the players on the given team that are within a specified static distance threshold. $\vec{V}_b$ in equation 7 is the direction vector that is used for the bravery force; u is a coefficient that is the maximum distance that a player will run away or towards the enemy; and k is a coefficient that modifies the strength of the friendly influence. This is to model the fact that a player will consider their own strength over their allies' strength in a combat situation. The player is either moving towards or away from the enemy, in relation to the averaged friend position. This is illustrated in figure 2. Equation 8 is the actual force used for bravery.

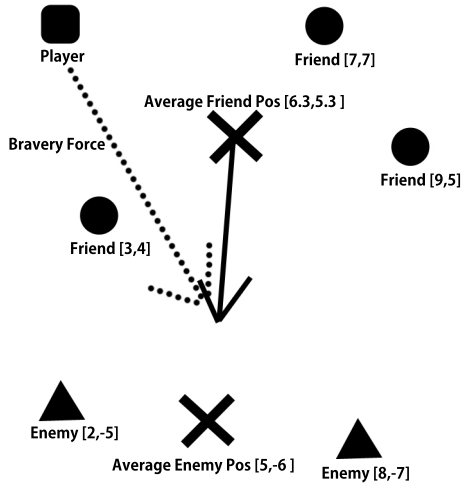

Fig. 2. Bravery force when friendly team is stronger

B. Combination of Forces

As previously mentioned, in order to simplify how the forces interact with each other, we separate player behaviour into two categories: *in battle* behaviours and *out of battle* behaviours. When the player is *in battle*, the player's position is calculated by combining the follow and the bravery forces. When *out of battle*, the players position is calculated by combining the follow and align forces.

Whether the player is in battle or not is chosen as a simple distance check to the closest enemy, outlined in equation 11. If there exists an enemy within the battle threshold W, then the player is said to be *in battle*. Equation 9 calculates the distance from the current player position to a given enemy player.

$$D_e = |\vec{P}_e - \vec{C}| \tag{9}$$

$$closestEnemyDist = min\{D_{e1}, D_{e2}....D_{en}\} \tag{10}$$

$$InBattle = \begin{cases} true & if \quad closestEnemyDist \leq \text{W} \\ false & otherwise \end{cases} \tag{11}$$

Algorithm 1 shows how the forces are handled. Coefficient q and r are static values that are between 0 and 1. They dictate how much each force is used. $\vec{F}_r$ is the final resultant force that is used to predict the player's position. If the player's velocity is immediately set to $\vec{F}_r$, the result is most often an inaccurate account of the player's movement. This is due to ignoring the player's most recent previously known velocity. To alleviate this, we perform a smooth transition of the player from the last known velocity to the one determined by the combined forces (see Equation 12). $\vec{F}_j$ is the force that should be used

as the velocity of the player and j is the number of updates that has occurred since the last known velocity $\vec{V_0}$. The more updates that have passed since the last known velocity was received (ie. the larger the value of j), the larger the value of $\vec{F_r}$ is and the smaller the value $\vec{V_0}$ is. Once j reaches the size of m, then we exclusively use $\vec{F_r}$ to determine $\vec{F_j}$.

$$\vec{F_j} = \begin{cases} \frac{m-j}{m}\vec{V_0} + \frac{j}{m}(\vec{F_r}) & if \quad j \leq m \\ \vec{F_r} & otherwise \end{cases} \tag{12}$$

$$m = \begin{cases} l \min\{closestEnemyDist, closestFriendDist\} & \\ & if \quad m \leq R \\ R & otherwise \end{cases} \tag{13}$$

The calculation for m is shown in equation 13. m is proportional to the distance between the player and the closer of the closest friendly or enemy player. This is due to the following observation: a player is more likely to react to a player that is close to it, and is less likely to continue at the current velocity. l is a coefficient used to modify m so that it is in the right scale. (We found that $l = 0.1$ works best.) R is the upper bound on m.

IV. Further Improvement

A. Using A*

We employ the A* path finding algorithm [11] to further improve the accuracy of the above-mentioned method. The use of the A* algorithm proceeds from observing a player's tendency to avoid walls and to find an efficient path through space to a desired location. This ensures that the path predicted for the player avoids walls objects and is more realistic. The implementation of the A* path finding algorithm in our scheme involves modification to the Follow and Bravery force (whereas the Align force remains the same). Instead of having only a force associated with $\vec{F}_{follow}$ and $\vec{F}_{bravery}$, each of these also has a desired location associated with it. For $\vec{F}_{follow}$, this desired location is the average position of all nearby friendly players. For $\vec{F}_{bravery}$, this desired location is $\vec{V_b} + \vec{C}$. A path to this desired location is then calculated using A* to ensure the player does not collide into any obstacle. Algorithm 1 outlines how A* is used.

B. Hybrid Approach

In order to further improve the prediction accuracy and reduce packets across the network, we adopt a hybrid scheme. Given the game is probed every 300ms: a) below 300ms of lag, EKB is always used (as it outperforms other algorithms) b) if a player has been moving in the same direction for the same amount of time as the network delay, then we assume the player will continue to move in this direction and thus we use TDM for the prediction, c) otherwise, EKB is used.

Algorithm 1 Algorithm *EKB*

1: **if** the desired A* end destination location has changed since the last update **then**
2: recalculate an A* path to the desired location for $\vec{F}_{follow}$ and $\vec{F}_{bravery}$.
3: **end if**
4: **if** the next A* node in the path is reached **then**
5: increment the desired location to the next node in the A* path.
6: **end if**
7: Compute $\vec{F}_{follow}$ and $\vec{F}_{bravery}$ using the next grid node in the path as the desired location.
8: **if** the player is in battle **then**
9: $\vec{F_r} = (\vec{F}_{follow} \times q) + (\vec{F}_{bravery} \times (1 - q))$
10: **else**
11: $\vec{F_r} = (\vec{F}_{follow} \times r) + (\vec{F}_{align} \times (1 - r))$
12: **end if**

V. Comparative Evaluation

We use two metrics to measure the accuracy of our dead reckoning scheme. The first one is the *Average Export Error (AEE)*. *AEE* is the discrepancy between the actual location of the player and the predicted location of the player (in pixels). We take the average of all export errors at fixed intervals of time (e.g., $300ms$, $600ms$, etc.) to determine the general accuracy of an algorithm. Figure 3 shows the AEE introduced by each algorithm. From this figure we can see that EKB greatly lowers the overall prediction error when predicting at large amounts of network delay. Table I shows the detailed comparison results between the three algorithms on AEE when the latency is between 1500 to 3000 milliseconds.

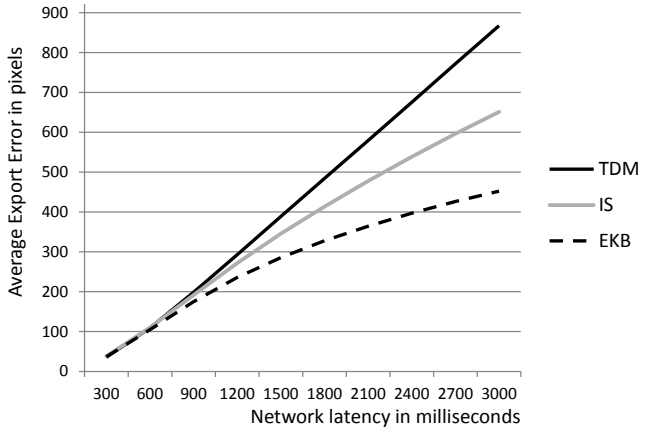

Fig. 3. Average Export Error measured at different network latency

TABLE I
AEE with high latency

	1500ms	1800ms	2100ms	2400ms	2700ms	3000ms
TDM [1]	388.8	484.2	579.5	675.4	771.7	867.5
IS [9], [10]	345.7	413.8	477.7	538.5	596.6	651.4
EKB	285.7	327.8	364.5	397.2	426.4	452.5

We then measure the number of times each algorithm makes an accurate *hit*. A *hit* is defined as when the predicted location

is within a specific threshold (measured in pixels) of the actual position. This threshold should be small enough so that when correcting the prediction player location to the actual location, the correction is minimally visible. In our work, we use 45 pixels as the threshold, given it is less than the width of the player in our multiplayer online game. Figure 4 lays out the number of hits that were recorded at each given time interval. The total number of hits counted for TDM, IS and EKB were 3135574, 2513613 and 3291199 respectively. EKB performed best, followed very closely by TDM. This is because TDM predicts very accurately when a player moving in a single direction, which is often the case. However, TDM is the worst for AEE.

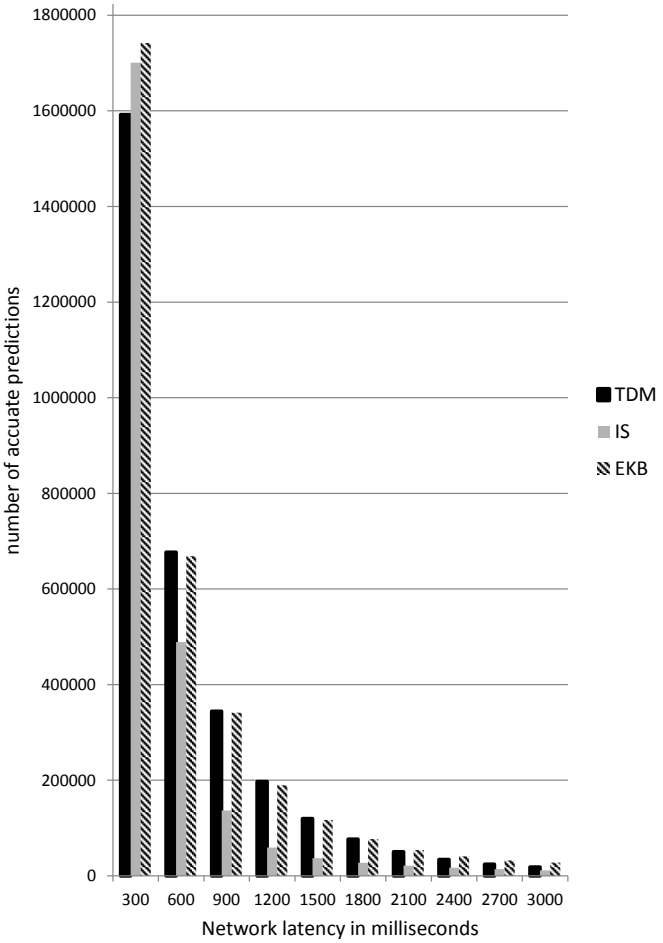

Fig. 4. Number of accurate predictions at threshold 45

We also measure the number of packets that need to be transmitted over the network during each time interval we monitor. Figure 5 shows EKB improves prediction accuracy over TDM and IS while sending as few packets as TDM. The difference in number of packets between TDM and EKB is negligible, whereas IS needs to send a significantly higher number of packets. We also conclude that contrary to IS, EKB improves the prediction accuracy without increasing the network traffic.

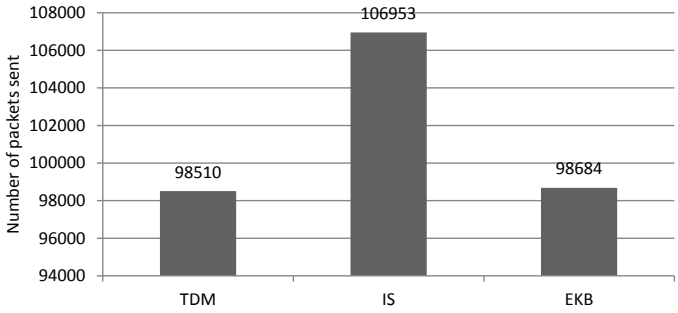

Fig. 5. Number of packets sent by algorithm TDM, IS and EKB

VI. Conclusion

In light of the limitations observed in existing work on dead reckoning, we have proposed here a new prediction scheme that relies on user play patterns. Our research takes place in the context of a 2D top-down multiplayer online game we have developed. In such typical team-based action game, a player's movement is highly unpredictable and is therefore highly prone to prediction inaccuracies, thus emphasizing the need for a better prediction scheme. We have evaluated our algorithm against the IEEE standard dead reckoning algorithm [1] and the recent "Interest Scheme" algorithm [9], [10]. Our simulation results suggest that our algorithm yields more accurate predictions than these two other algorithms.

Acknowledgment

The authors would like to thank the financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] IEEE Standard for Distributed Interactive Simulation— Application Protocols. IEEE Standards Board. Septempber 1995.
[2] D. E. Comer, *Computer Networks and Internets*. Prentice Hall, pp. 476.
[3] L. Pantel and L. C. Wolf, *On the Suitability of Dead Reckoning Schemes for Games*. The 1^{st} workshop on Network and system support for games (NetGames '02), pp. 79-84, 2002.
[4] T. Duncan, D. Gračanin, *Pre-Reckoning Algorithm For Distributed Virtual Environments*, Proceedings of the 2003 Winter Simulation Conference, pp. 1086 - 1093, 2003.
[5] W. Cai and F. Lee and L. Chen, *An Auto-adaptive Dead Reckoning Algorithm for Distributed Interactive Simulation*, Proceedings of the 13^{th} workshop on Parallel and Distributed Simulation, pp. 82 - 89, 1999.
[6] A. McCoy and T. Ward, S. McLoone, D. Delaney, *Multistep-Ahead Neural-Network Predictors for Network Traffic Reduction in Distributed Interactive Applications*, ACM Transactions on Modeling and Computer Simulation, Vol.17(4), September 2007.
[7] A. Hakiri and P. Berthou and T. Gayraud, *QoS-enabled ANFIS Dead Reckoning Algorithm for Distributed Interactive Simulation*. IEEE/ACM 14^{th} International Symposium on Distributed Simulation and Real Time Applications (DS-RT '10), pp. 33-42, 2010.
[8] D. Delaney and T. Ward and S. McLoone, *On Reducing Entity State Update Packets in Distributed Interactive Simulations Using a Hybrid Model*. Applied Informatics (378), pp. 833-838, 2003.
[9] S. Li and C. Chen, *Interest Scheme: A New Method for Path Prediction*. Proceedings of the 5^{th} ACM SIGCOMM workshop on Network and system support for games (NetGames '06), 2006.
[10] S. Li and C. Chen and L. Li., *A new method for path prediction in network games*, Computers in Entertainment, Vol. 5, New York, 2007.
[11] S. Russell and P. Norvig, *Artificial Intelligence - A modern Approach*, Prentice Hall, 1995.