A New Method for Path Prediction in Network Games

SHAOLONG LI, CHANGJA CHEN, AND LEI LI
Beijing Jiaotong University

In almost all multiplayer network games, dead-reckoning (DR) is used to predict the movements of game players, who can then predict the future movements of other players via the DR vectors they received. DR vectors, referred to as network packages, generally contain the position and velocity of game roles controlled by a sender at sending time. To achieve more accurate prediction, some games include the timestamp and acceleration of game roles in DR vectors. However, DR does not work well under bad network conditions. In our previous work [Li and Chen 2006] we proposed a solution called the interest scheme (IS), which proved to be efficient when network latency was unsteady and package loss frequent. Thus, in order to achieve much more accurate prediction, we proposed a hybrid solution. IS assumes that the path prediction for a given player is related to nearby objects or players. That is, that the players' surroundings can affect their movements, and different players may behave differently under the same conditions. In IS, a given player's surroundings are taken into account, and in order to achieve more accuracy, his habitual preferences are also taken into consideration. Experience with network games indicates that the same player will almost always behave in the same way under the same circumstances—for example, use the same fighting style. Moreover, we consider that different prediction methods should be used for different network latencies. So we introduce a hybrid method, which is a combination of IS, DR, and personal preferences. We use a 2D tank game to experiment, and compare the results of our solution with those of traditional methods. To obtain information on the players' habitual movements, we observed each participant for 30 minutes of play. Simulation shows that our method achieves significant improvements in path prediction.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems — *Distributed applications*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Network delay, dead-reckoning, dead-reckoning vectors, interest scheme (IS), hybrid method

ACM Reference Format:
Li, S., Chen, C., and Li, L. A new method for path prediction in network games. *ACM Comput. Entertain.*, 5, 4, Article 8 (March 2008). 12 Pages. DOI = 10.1145/1324198.1324206
http://doi.acm.org/10.1145/1324198.1324206

1. INTRODUCTION

In network multiplayer games, due to limitations in network bandwidth, local machine computation, and so on, players cannot send information on their current positions to others at all times. So clients send DR vectors at very short intervals. Due to network delay, players in multiplayer network games know only the previous positions of others

Authors'address: EIE Department, Beijing Jiaotong University, Shangyuancun, Xizhimenwai, Beijing 100044, China; email: shaolonglichina@yahoo.com.cn; changjiachen@sina.com.cn; leili_china@yahoo.com.cn

from the DR vectors they received. In order to get the current positions of other players, some prediction method must be used. After receiving a DR vector, receivers use a pre-diction method to predict the movements of the senders until a new DR vector is received. Players commonly use a method called dead-reckoning (DR) to exchange information on their positions [Singhal and Cheriton 1995]. DR is the process of estimating a present position by projecting the direction and speed of a player from his last known position. A DR vector typically contains the position of a player in f x, y, and z coordinates as well as the velocity component in each dimension. In some network games, a timestamp [Aggarwal et al. 2004.] and acceleration are also included to achieve more accuracy in path prediction. Using DR vectors, a player renders another player on the local console until a new DR vector is received from that player. DR vectors can usually be described as (*Vx*, *Vy*, *Vz*, *Px*, *Py*, *Pz*, *T*) where *V* is the speed of movement, *P* is the position, and *T* is the physical sending time. The traditional method to predict a path (DR) simply considers the last known position, velocity, and the time slot. DR works well under normal network conditions, but in actual, real game experience, its performance drops sharply with growth of network delay.

In this article we propose a method called the interest scheme (IS) to achieve much more accurate path prediction The traditional method predicts the movements of players according to speed and direction only. Unlike the traditional method, IS takes the players' environments and personal habits into account. Experience gained from playing actual games shows that player behavior is affected by their surroundings and personal preferences: players may confront or evade the object, fight the enemy, or run away. Some players are interested in killing, some are dedicated to obtaining objects. Experience with network games shows that the same player will almost always do the same thing under the same conditions (e.g., use the same fighting style).

There are two attitudes that a given player will take toward his surroundings: attraction or repulsion. The degree of the relationship is denoted by two parameters — interest and distance. The first parameter refers to the attitude of hostility or friendliness, and the second refers to the distance between the player and the nearby object or player in a game world The degree of interest depends on many factors such as the players' current state and their personal preferences. That is, for players who like to acquire objects, interest in objects will be higher than that of players who like to fight against others. To achieve better performance, our method allows time to observe and become familiar with the personal preferences of players. The results of our experiments show that the players' future movements are decided by the two aforementioned parameters.

Extra computation is introduced by IS to ensure accuracy in path prediction, since computing parameters is very complex. It is obviously unnecessary when a network is steady and delays are short, since the traditional method can work well in such circumstances. Hence we propose a hybrid method (HM), in which the advantages of IS and the traditional method are combined and a time threshold is introduced into the prediction process. The threshold is the maximum time below which a traditional method can work well. A traditional method is used when the path prediction process begins. Once the elapsed time is beyond the threshold, IS will replace the traditional method until a new DR vector is received.

The rest of this article is organized as follows: Section 2 introduces the background of prediction methods; Section 3 describes IS and the algorithm; the hybrid method is introduced in Section 4. The experimental method and results obtained from the

instrumentation of the scheduling algorithm on a modified open-source tank game are presented in Section 5; Section 6 concludes and presents possible future work.

2. BACKGROUND

Modifications to DR have been suggested previously [**URL:http://p24.bakadigital.com/p24bb/viewtopic.php?t=14**; Pantel and Wolf 2002b], but not from the perspective of accuracy. Earlier work on path prediction methods has mostly focused on compensation techniques for packet delays and loss [Pantel and Wolf 2002a; Hashimoto and Ishibashi 2006]. These methods aim to make long delays and message loss tolerable for players, but do not consider adopting a new method to replace DR. The concept of a local log was used in Mauve [2000], where, to reduce state inconsistencies, each player delays every local operation for a certain amount of time so that a remote can receive information about the local operation and execute the same operation at about the same time. Some network games (e.g., *MiMaze* [Gautier and Diot 1998; 1999] use a static bucket synchronization method to compensate for network delay. For client-server- based FPS (first person shooter) games, Bernier [2001] discusses a number of latency compensating methods at the application level (these are proprietary to each game). There have been a few articles that have examined the problem of the fairness and security of message-delivery mechanisms [Aggarwal et al. 2005]. Some previous work [Guo et al. 2003] uses more sophisticated message-delivery mechanisms to solve the problem of fairness in a distributed game, but greatly increase network delay for players. It has been shown in Baughman and Levine [2001[that multiplayer games that use DR vectors along with bucket synchronization are not cheat-proof unless additional mechanisms are put in place. In some sense such work is limited because the methods proposed above are based on a DR method in which large export errors cannot be avoided under bad network conditions. We emphasize that the focus of this article is on accuracy and efficiency, it does not address the issue of cheating.

The traditional DR method uses previously received vectors to project a future path. The receiver predicts the movements of the sender on the basis of the latest information on position received from the last vectors. The popular algorithm is

$$P_c = V \times \Delta t + P_p \tag{1}$$

In Eq. (1), P_c, V, Δt, and P_P represent, respectively, the current position, movement speed, elapsed time, and previous position as received from the last and second-to-last DR vectors. Some correlative techniques such as the synchronizing technique [Lin et al. 2002; Simpson 2004] and latency compensation [Bernier 2001] are imported into the traditional method to improve accuracy and to obtain a more accurate P_P.

It is almost impossible for players to exchange their current positions continuously. So the real trajectory (which we refer to as a real path) to a player is quantized to DR vectors. A new DR vector is usually computed and sent whenever the real path deviates from the path extrapolated via the previous DR vector by a specified threshold. We refer to the trajectory that can be computed by using a sequence of DR vectors as the exported path. Due to network delay, when a DR vector is received and rendered to a player, the player's original trajectory may have already changed. Thus, for the receiving player, there is a deviation in real time between the exported path and the rendered trajectory (which we refer to as the placed path). We call this error the export error. Note that the export error, in turn, results in a deviation between the real and placed path.

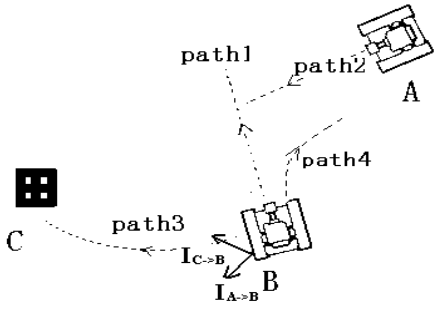


Fig 1 Difference between DR and IS

3. THE INTEREST SCHEME

As mentioned above, with the DR method, large export errors cannot be avoided under bad network conditions. Large export errors result in many more changes in position for the client, which negatively affects the playing experience. Accurate path prediction will make game-play a more satisfactory experience for players.

In most network multiplayer games, there are always some players who experience long network delays in accessing the game server or contacting other players. For such players there is often a large deviation in path prediction due to their use of the traditional method. Hence we propose IS in order to achieve better performance.

3.1 The IS Model

We believe that the players' surroundings and habitual behaviors can affect their movements So we propose our solution, called the interest scheme (IS), which is based on a system in which interaction effects exist among players and objects. A player cannot be isolated in a game space. His future actions are actually influenced by nearby objects and or players. Game experience and experimental results show that a player's movements are influenced by the interaction effect with his or her surroundings. Common sense shows that an entity controlled by a player will change its direction if it wants to pick an object to its side or wants to avoid fighting against a powerful enemy in front of it. Personal preferences also affect a player's behavior. Some players have an interest in killing and some are dedicated to picking objects. We classify the relationship between the player and his surroundings into two categories: attraction and repulsion. Attraction means wanting to get closer, while repulsion means wanting to get away. The intensity of the attraction or repulsion is determined by two parameters: interest and distance. The interest parameter denotes the degree of correlation between the player and a nearby object or player; distance stands for the extent of the space from the entity to the nearby object or player. Interest is related to the current state and the personal preferences of the players. Using these two parameters, we can compute the intensity of attraction or repulsion between two players or from a player to an object. The final direction of a given player's movements can be computed after we gain an understanding of the forces which nearby objects and players have on him.

Figure 1 shows the difference between the DR and IS methods. The direction of a player's movement is affected by nearby objects and other players. We suppose that the

state of tank A is much better than that of tank B. There is no consideration of the influence of personal preferences, so there is an attraction from tank B to object C, and a repulsion from tank A to tank B. As shown in the figure, tank B will change its future direction in order to avoid fighting because its state is not good, so it will change direction to pick object C to enhance its state. Path 1 is the result of using a traditional DR method. After computing the composition of forces inflicted on tank B, path 3 is our IS result. We can see that the deviation between path 1 and path 3 becomes larger as time goes on. If personal preferences are taken into account, and tank B would like to destroy more enemies before picking objects, then there is an attraction from tank B to tank A. After computing the composition of forces under this condition, path 4 is the IS result. The difference between our model and DR is that DR assumes that players will always move according to their previous movements, but in fact they may change their direction because of their surroundings or personal preferences. Our model takes these influences into consideration. In our model, the forces introduced by other players and objects will decide the future direction in which the entity will move. The previous speed and direction will also have an impact on the direction of movement, but their effects will gradually decrease.

3.2 The IS Algorithm

In multiplayer network games, two basic components are involved: players and objects, which are either good or harmful to players. In our model, the direction in which a player is headed is predicated on how the player is affected by nearby players or objects; previous speed and direction will also affect the prediction. In this article, *S* denotes the state of a player or object (*S* has different meanings in different games; here it refers to the life and quantity of bullets in a tank). P_i denotes the position of a player or object at a given time T_i; $\vec{D}$ denotes the distance from the given player to other players or objects; $\vec{F}$ denotes the virtual force among players and objects; and $\vec{F}_r$ denotes resultant force. *I* is the degree of interest between two players or from a player to an object. Incidentally, in our algorithm, personal preferences only affect *I*. Suppose there are two players and an object in a game: A, B, and O. S_A, S_B, S_O, and S_{MAX} denote, respectively, the states of player A, player B, object O, and the entire state of the players. Then we have the degree of interest of player B for player A:

$$I_{B-A} = \frac{S_B - S_A}{S_{MAX}}. \tag{2}$$

The degree of interest of player A or B for object O:

$$I_{A(B)-O} = \begin{cases} \dfrac{S_O}{S_{MAX}} & S_{MAX} - S_{A(B)} \geq S_O \\ \dfrac{S_{MAX} - S_{A(B)}}{S_{MAX}} & ow \end{cases} \tag{3}$$

I can be positive or negative; that is, the force can be one of attraction or repulsion.

And we have

$$\vec{F} = k\frac{I\vec{D}}{\left|\vec{D}\right|^3} \tag{4}$$

where *k* is a coefficient. We can then compose the forces that affect the player.

We assume that our game model uses a synchronizing technique and that the effects of the previous direction decrease linearly. From the view of player B, we can now predict the path of player A. At a given time T_i, player B predicts the position of player A from DR_{i-1} which player A sent at T_{i-1}. We split the time interval between two consecutive DR vectors, denoted by ΔT, into several time slices. During each time slice, denoted by Δt, we predict the direction of movement of the entity to get ΔP_j, which denotes the distance the player traveled in the *j* slice. We set Δt to be a proper value due to the trade-off between predictive complexity and accuracy.

According to the traditional method, P_i is

$$P_i = P_{i-1} + \vec{V}_{i-1} \times \Delta T \tag{5}$$

$$\Delta T = T_i - T_{i-1} \tag{6}$$

In our model, the *j* slice time is denoted by

$$T_j = T_i + j \times \Delta t \tag{7}$$

$$P_i = P_{i-1} + \sum_{j=0}^{j=n-1} \Delta P_j + \Delta P' \tag{8}$$

$$\Delta P_j(\Delta P') = \vec{V}_j \times \Delta t(\Delta t') \tag{9}$$

$$\Delta T = n \times \Delta t + \Delta t'(\Delta t > \Delta t') \tag{10}$$

On T_j, player A is affected by $\vec{F}_j$,

$$\vec{F}_j = \begin{cases} \frac{m-j}{m}\vec{F}_0 + \frac{j}{m}(\vec{F}_r) & j \le m \\ \vec{F}_r & ow \end{cases} (\vec{F}_r = \vec{F}_{B-A} + \vec{F}_{A-O}) \tag{11}$$

If personal preferences are taken into account, the direction of the resulting force must be amended according to the player's habitual behavior. From observation, we classify all players into three categories: those whose first priority is to pick objects; those whose first priority is to kill enemies; and players who play the game normally. E denotes the frequency of a player's habitual behaviors during a period of observation. We set E from -100 to 100, so that player behavior ranges from always picking objects first to always killing enemies first. Personal preferences only affect the angle of the resulting force. We divide the angle of $\vec{F}_r$ by $\vec{F}_{B-A}$ or $\vec{F}_{A-O}$ into 100 even parts. Thus we change the direction of $\vec{F}_r$ in accordance with E to fit personal preferences.

$\vec{F}_0$ is the initial virtual force and F_{B-A} is the virtual force of player B acting on player A. For appending the effects of a previous direction, we import $\vec{F}_0$, which may vary greatly in different games, but in any given game it is almost a constant, even for different players. Then $\vec{F}_0$ is treated as constant while $\vec{F}_{B-A}$ is determined by $\vec{D}_{A-B}$ and I_{A-B}. As shown in Eq. (4), *m* is used to gradually decrease the effect of $\vec{F}_0$ on $\vec{F}_j$.The exact value of *m* is determined by $\vec{D}_{A-B}$ and $\vec{D}_{A-O}$, but it also has an upper bound because

$\vec{F}_0$ will decrease rapidly when $\vec{D}_{A-B}$ and $\vec{D}_{A-O}$ are very small. That is to say, the influence of $\vec{F}_0$ will disappear rapidly when other entities or objects are near. Then we have

$$m = \begin{cases} \left\lfloor l \min\left\{\vec{D}_{A-B}, \vec{D}_{A-O}\right\}\right\rfloor & m \le R \\ R & ow \end{cases} \tag{12}$$

where l is a coefficient and R is the upper bound. $\vec{V}_j$ has the same direction as $\vec{F}_j$, and the velocity of player A is constant; $\vec{V}_j$ is easy to work out. In addition, the direction of $\vec{F}_{B-A}$ is not from P_B to P_A but from $P_B + \Delta\vec{P}_B$ to P_A. $\Delta\vec{P}_B$ is intending distance of player B moving and the length of $\Delta\vec{P}_B$ relates to the distance between players A and B and the speed of player B.

$$\left|\Delta\vec{P}_B\right| = q\frac{\left|\vec{V}_B\vec{D}_{A-B}\right|}{\left|\vec{V}_A\right| \times \left|\vec{V}_B\right|}\vec{V}_B \tag{13}$$

where q is a coefficient. $\Delta\vec{P}_B$ is used because player A cares about the position that player B is heading to. From the viewpoint of A, we can also use this method to obtain the trajectory of player B.

When multiplayers and multiobjects exist in the game (n players, l object), the situation is almost the same as above. The only difference is that when considering personal preferences, angle of $\vec{F}_r$ is related to Max($\vec{F}_A$) or Max($\vec{F}_O$). So at time T_j

$$\vec{F}_j = \begin{cases} \frac{m-j}{m}\vec{F}_0 + \frac{j}{m}(\vec{F}_r) & j \le m \left(\vec{F}_r = \sum_1^n \vec{F}_A + \sum_1^l \vec{F}_O\right) \\ \vec{F}_r & ow \end{cases} \tag{14}$$

$$m = \begin{cases} \left\lfloor l \min\left\{\vec{D}_{A1},....\vec{D}_{An}, \vec{D}_{O1},.....\vec{D}_{Ol}\right\}\right\rfloor & m \le R \\ R & ow \end{cases} \tag{15}$$

where l is a coefficient and R is a threshold. Finally, all values of coefficients mentioned in this section such as k, l, q, and R are identical to matching experimental data.

4. THE HYBRID METHOD

IS works well when packet loss is frequent and network delay is long. However, it introduces an extra computational burden, and seems unnecessary when a network is steady and delay is short. Hence we propose the hybrid method, which combines the advantages of IS and the traditional method. For some time after prediction begins, HM uses the traditional method first. If the new DR vector is not received within a certain time, IS begins to work until new DR data arrives, and the process continues. The steps are as follows:

(1) A fixed time threshold is set taking into account the given network multiplayer game and its network situation.
(2) The sender sends the DR vector at time T.
(3) To predict the path, the traditional method is used first.

(4) Once the traditional method is chosen, attention must be paid to the elapsed time. If the time reaches the threshold, the IS method is used until a new DR vector is received.
(5) Go back to Step 2 to handle the new DR vector.

By following these steps, the computational burden can be sharply decreased, while the exported errors can be kept to a low level.

5. EXPERIMENTAL RESULTS

In this section we modify an open source network multiplayer tank game and use experimental measurements from the modified implementation. In the game map, each player controls a tank that fights against others and can also pick objects to complement its bullets or to lengthen its life. The state of the tank relates to its longevity and the quantity of bullets. In order to obtain enough observational data, we invited 37 expert game participants to play our modified tank game for more than half an hour, as shown in Table I(a). Table I(b) shows the times of the path predictions under different network latencies.

We established our measurement platform on a C/S-based network consisting of clients A, B, and a game server. Client A is a station where the real path is traced and logged. DR vectors are sent from client A with an interval of 500ms. Client B acts as a receiver for the DR vectors and records the projected path of client A. Real paths are traced and logged every 50 ms at client A, which also records the global time. Client B records the received DR vectors and the projected path every 50ms. The global time is logged at both clients. Before our experiment, each player was asked to play the game for 30 minutes in order to provide information on their personal preferences (shown in Figure 2). We used the 30 minutes to compute the export error on the basis of the real path, the projected path, and global time.

Network delays to two clients are controlled by the server; network delay from the server to client A is set to zero. The network latency between the server and client B has three possible values: 1 second, 2 seconds, and 3 seconds. The export errors of the traditional DR method and the IS and HM methods are calculated at client B. Sheldon et al. [2003] show that when network latency is longer than 1 second, the players' experience will be

Table I Data Statistics

Number of players	1	5	11	18	2
Playing Time (minutes)	20~30	31~40	41~50	51~80	81~120

(a)

Network delay (second)	1	2	3
Times of prediciton	13913	13299	12642

(b)

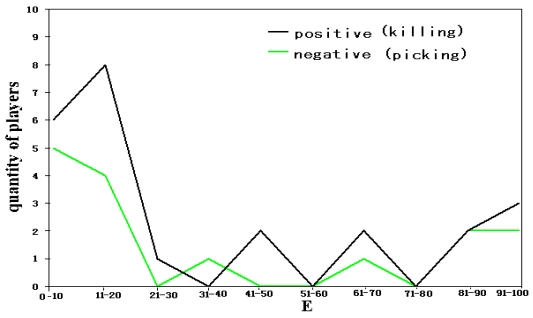
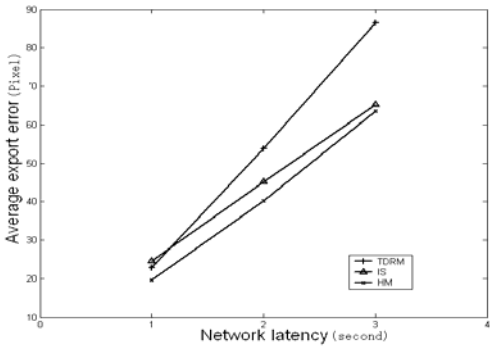


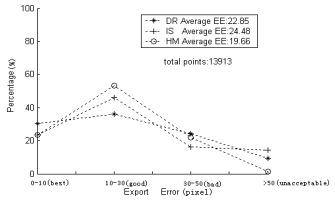
Fig 2 habitual behaviors of players

affected by network latency. When network latency is shorter than 1 second, the traditional prediction method can work well. So, for purposes of measurement, we set network latency from 1 second to 3 seconds.

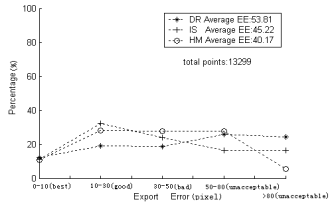
From all the data stored at two clients, offset: $\Delta\vec{P}_B$ can be computed every 50ms, so q is an average of 0.356 in Eq. (13). For a trade-off between predictive accuracy and the computational burden, Δt is set to 250ms. From the information on position recorded at the two clients, entities keep moving for an average of 2.4 seconds when there are no enemies or objects nearby. Thus the influence of $\vec{F}_0$ only lasts 2.4 seconds without the effects of other entities and objects. From Eq. (11), R equals 9.6; because R is an integer, R is set to 10. Tank velocity is 40 pixels per second and the effective time of $\vec{F}_0$ is 2.4 seconds; l is set to 0.1 from Eq. (12). E is obtained from our observational data, which is shown in Figure 2.

In network multiplayer games, relations between $\vec{F}$ in Eq. (4) and $\left|\vec{F}_0\right|$ are complex, having to do with the scale of the map, velocity of the entity, zone of sight, definition of state, and so on. In our proposed method, value $\left|\vec{F}_0\right|$ cannot be obtained from the stored data; we can only decide about the relationship between $\vec{F}$ in Eq. (4) and $\left|\vec{F}_0\right|$ from experimental data. For simplicity, $\left|\vec{F}_0\right|$ is set to 1. We chose more than 3,000 position

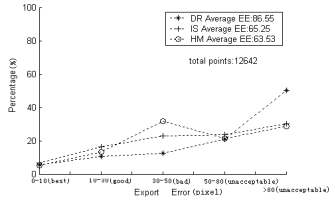




(a)



(b)



(c)

Fig. 4. Comparing the rates of export errors.

Fig 3 Average export error of DR IS and HM

points stored at the second client for matching *k* to experimental data. Considering the computational costs, there is only one nearby enemy or object at these points. We scanned the number of *k* for minimizing the average of the export errors at these points. From the results of the match, when *k* is 4,165, the average export error is at a minimum. Incidentally, the fixed time threshold was simply set to 500 ms in the HM method, taking into account machine computation and the condition of the network.

The game map is 400×400 pixels and the velocity of a tank is 40 pixels per second. We rank the export error (EE) as follows: from 0~10 is best; 10~30 is good, and 30~50 is bad; an EE of more than 50 is unacceptable. We think the best EE and the good EE rankings are acceptable because players find them tolerable. We chose 13,913, 13,299, and 12,642 points with a network latency of 1s, 2s, and 3s, respectively. In Figure 4, the x axis stands for the EE; the y axis stands for the percentage of points at which the EEs are in a given range. We note that in Figure 4(a) the IS and HM have much better acceptable rates of EEs than DR does and that HM has the minimum percent of unacceptable EEs (less than 1%). We can also see that the highest percentages of acceptable EEs are obtained by HM at 77.86% and DR at 62.45% We can also see that the IS has the highest percent of acceptable EEs (77.86%); DR has 62.45%, which is a little lower than the 65.21% for IS.. The average rate of EEs is lowest for HM, only

16.47, while for IS it is 23.25. We compare the two-second delay for the different prediction methods in Figure 4(b). It is clear that when network delay increases, the percentage of unacceptable EEs grows and the percentage of best EEs decreases rapidly. HM has 80 EEs, a small percentage, i.e., 5.31%; the acceptable percentage of EEs in IS is the largest at 46.58%. As to average EEs, HM is still the best (31.23); DR has the largest (53.81). From Figure 4(c), we see that the unacceptable EEs increase continually and the good EEs decrease substantially as network delay increases. By the way, DR has a much larger percentage of unacceptable EEs than IS and HM do. As to the average EE, IS and HM are still much better than DR. In Figure 3, we see that when network delay increases, the average EE rises rapidly when using DR; however, it is almost the same lower increment when using IS and HM. We can see from the figure that the EE for HM gives the best experimental results.

In summary, the export error for the DR method becomes unexpectedly large with long network latency. We confirm that the proposed HM reduces export errors and that the burden of computation on the client side is negligible. With the network delay becoming longer, the improvement will become more obvious.

6. CONCLUSION

In this article we introduced a new path prediction method for multiplayer network games. To test the new method, we modified a tank game and analyzed the results of our experiments. As our experiments involved 37 players, we believe that the results show the advantages of our new prediction method.

It is common sense to say that the players' surroundings and personal preferences will affect their behavior. Experience with network games indicates that the same player will do almost the same thing in the same way (e.g., use the same fighting style). Our results suggest that this influence can be used to improve the accuracy of path prediction. In most network games there are strong correlations among entities controlled by players and objects, so our new prediction method can be used widely. In future work we will investigate other player states (e.g., how scores affect their movements). Furthermore, we can optimize the choice of time threshold into our proposed prediction method.

REFERENCES

AGGARWAL, S., BANAVAR, H., MUKHERJEE, S., AND RANGARAJAN, S. 2005. Fairness in dead-reckoning based distributed multiplayer games. In *Proceedings of the NetGames 05 Conference.*

AGGARWAL, S., BANAVAR, H., KHANDELWAL, A., MUKHERJEE, S., AND RANGARAJAN, S. 2004. Accuracy in dead-reckoning based distributed multi-player games. In *Procceedings of NetGames 2004* (Aug.).

BAUGHMAN, N. E. AND LEVINE, B.N. 2001. Cheat-proof playout for centralized and distributed online games. In *Proceedings of the INFOCOM 01Conference.*

BERNIER, Y.W. 2001. Latency compensation methods in client server in-game protocol design and optimization. In *Proceedings of the Game Developers Conference '01*.

DIOT, C. AND GAUTIER, L. 1999. A distributed architecture for multiplayer interactive applications on the Internet. *IEEE Network Mag. 13*, 6-15.

GAUTIER, L. AND DIOT, C.1998. Design and evaluation of MiMaze, a multiplayer game on the Internet. In proceedings of *ICMCS 98.*

HASHIMOTO, Y. AND ISHIBASHI, Y. 2006. Influences of network latency on interactivity in networked rock-paper-scissors. In *Proceedings of NetGames 06*.

LI, S. AND CHEN, C. 2006. Interest scheme: A new method for path prediction. In *Proceedings of the NetGames 06 Conference.*

LIN, Y., GUO, K., AND PAUL, S. 2002. Sync-MS: Synchronized messaging service for real-time multi-player distributed games. In *Proceedings of the Tenth IEEE International Conference on Network Protocols* (ICNP, Nov.).

MAUVE, M. 2000. Consistency in replicated continuous interactive media. In *Proceedings of the CSCW 2000 Conference*.

PANTEL, L. AND WOLF, L.C. 2002a. On the impact of delay on real time multiplayer games. In *Proceedings of the NOSSDAV Conference*. ACM, New York.

PANTEL , L. AND WOLF, L.C. 2002b. On the suitability of dead reckoning schemes for games. In *Proceedings of NetGames 02*.

SHELDON, N., GIRARD, E., S.BORG, S., CLAYPOOL, M., AND AGU, E. 2003. The effect of latency on user performance in Warcraft II. In *Proceedings of NetGames 2003*.

SIMPSON, Z.B. 2004 A stream based time synchronization technique for networked computer games. http://www.mine-control.com/zack/timesync/timesync.html.

SINGHAL, S.K. AND CHERITON, D.R.1995. Exploiting position history for efficient remote rendering in networked virtual reality. *Presence:Teleoperators and Virtual Environments 4*, 2, 169-193.

URL:http://p24.bakadigital.com/p24bb/viewtopic.php?t=14

GUO, K., MUKHERJEE, S., RANGARAJAN, S., AND PAUL, S. 2003. A fair message exchange framework for distributed multi-player games. In *Proceedings of the NetGames 2003.*

Received May 2007; revised August 2007; accepted August 2007