Energy-Efficient Gaming on Mobile Devices using Dead Reckoning-based Power Management

R. Cameron Harvey, Ahmed Hamza, Cong Ly, and Mohamed Hefeeda
School of Computing Science
Simon Fraser University
Surrey, BC, Canada

***Abstract*—We address the issue of how to reduce energy consumption of the wireless interface during multiplayer gaming sessions on mobile devices. Reducing energy consumed by the wireless interface is achieved by putting it in a low-energy state when it is not used. We leverage the dead reckoning technique used in existing games. In dead reckoning, future locations of objects in a game are estimated based on their current locations and velocities. The difference between the extrapolated and true locations is known as the dead reckoning error. We propose an algorithm that employs the dead reckoning error rate to dynamically control the state of the wireless interface. We implement our algorithm into a dead reckoning simulator that is based on a real open-source game. Our experimental results show that the proposed algorithm can achieve up to 36% energy savings for mobile devices. Our proposed algorithm is practical because it does not require much additional code and it allows easy integration with existing games.**

I. Introduction

The last few years have seen a tremendous growth in wireless networking enabled devices. PDAs and smart phones are becoming increasingly powerful and are now used for much more than basic communications. Wireless capability and powerful CPUs have led to the mobile gaming industry capturing a progressively larger portion of the gaming market. Mobile gaming is projected to reach $18 billion globally in 2014 [8]. This includes multiplayer games which rely heavily on wireless radios to communicate with other players. One example of a commercially available multiplayer game available for download to mobile devices is Quake [2].

Gaming uses a lot of power and one of the most significant drains on the batteries of mobile devices is the wireless network interface card. This can account for up to 70% of the total power consumed by the device [12]. Lowering the power used by the wireless interface would greatly decrease the overall power consumed by the mobile device. This is achieved by spending as much time as possible in low-power states. At a first glance, we may think that we can maximize energy savings by turning off the wireless interface when it is not in use. This naive approach presents difficulties. If wireless interface is in sleep-mode, it may not know when to wake up and will miss an incoming packet, but if we let the wireless interface stay awake too long we miss the opportunity to save energy.

This work is partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

In this paper, we address the issue of how to reduce energy consumption of the wireless interface during a multiplayer gaming session by leveraging dead reckoning (DR) techniques used in existing games. Our key idea is to use dead reckoning error rates to dynamically devise a power saving schedule for the wireless interface. We use these rates to predict how long to put the wireless interface into its low-power consumption state. Using a multiplayer game simulator [11], we show that our system can achieve up to 36% energy savings without severely degrading the player's quality of experience.

The rest of this paper is organized as follows. We give a brief overview of multiplayer games and the dead reckoning technique in Section III. We present our proposed solution in Section IV, and we evaluate its performance in Section V. Section VI concludes the paper.

II. Related Work

Our work is closely related to several techniques to make gaming devices with wireless interfaces more energy efficient. Common techniques such as [4], [13] use the power save mode (PSM) introduced in the IEEE 802.11 standard for infrastructure mode. The power save mode allows a wireless client to transition to lower-power state when it is not active. Incoming data destined for clients in power save mode are buffered by the access point. The access point will periodically send out a traffic indication map inside the beacon frame to indicate that the access point has buffered data for a particular client in sleep mode. It may seem like using power save mode obviates the need for additional energy saving techniques. However, power save mode is only available in infrastructure mode and does not work in ad-hoc mode where there is no access point to buffer data. Additionally, the gaming traffic has real-time constraints and each packet must reach the client by a certain deadline [7]. Clients using power save mode may not know when it is necessary to wake up, causing intolerable delay in gaming packets. Our proposed solution is well-suited as an enhancement to PSM-based techniques in infrastructure mode and it also works in the ad-hoc mode.

Krashinsky et al. [10] present an alternative to power save mode called the Bounded-Slowdown (BSD) protocol that dynamically adapts sleep periods to past network activity. This technique requires making changes to existing and widely accepted network protocols and standards. In contrast, methods such as [14]–[16] seek to minimize energy consumption

978-1-4244-8355-6/10/$26.00 ©2010 IEEE

by turning off the wireless interface. These methods rely on scheduling algorithms to determine the sleep periods. We have adapted similar strategies to [16] for use in our system and evaluated its effectiveness in a realistic gaming environment. It is worth noting that while the energy reduction strategy is similar, our solution does not seek to formulate a complex scheduling algorithm. Instead, we leverage the dead reckoning mechanisms present in existing multiplayer games to establish the prediction model.

To the best of our knowledge, the use of dead reckoning error rate was first proposed by Aggarwal et al. [3]. The context of their problem was however quite different. Their goal was to use a distributed dead reckoning error rate in order to improve fairness in game play. In our work, we use the dead reckoning error rate to dynamically devise a sleep period for the wireless interface.

III. Background and Overview

A. Multiplayer Games

Multiplayer games are roughly classified into two types: avatar games and omnipresent games [7]. In avatar games, a player controls a single character, which exists at a precise location in the virtual space, and can only interact with near-by objects. Avatar games include shooter games, role-playing games, action games, and sports games. These games are further categorized into first-person avatar games in which a player's view is through the character's eyes, and third-person avatar games in which a player sees the character from a distance. In omnipresent games, a player concurrently controls a group of characters, and can interact with objects that are close to any of these characters. Omnipresent games include real-time strategy games and simulation games.

We consider a fairly general model for multiplayer avatar games in wireless environments, where several players form a group and exchange many game-state updates. This model is general because it can be readily mapped to different types of games. After agreeing upon the game settings such as the map and rules, several players form a *gaming session*. In order to maintain session consistency, one client is chosen as the *authoritative host*. The host acts as a pseudo server for the gaming session. Game-state updates are sent to the host for verification before being forwarded to players in the session.

B. Dead Reckoning

Dead reckoning is the process of estimating the future position of an object using its current position and velocity. Traditionally, dead reckoning is used to hide network latency in multiplayer games and to reduce network traffic. Using dead reckoning [5], clients can extrapolate the behavior and state of gaming objects and thus can continue rendering frames even if game-state updates are late. A dead reckoning vector is used to provide game-state updates about player and object movements. The vector typically contains the current position of the game-object in terms of x, y, and z coordinates as well as its trajectory.

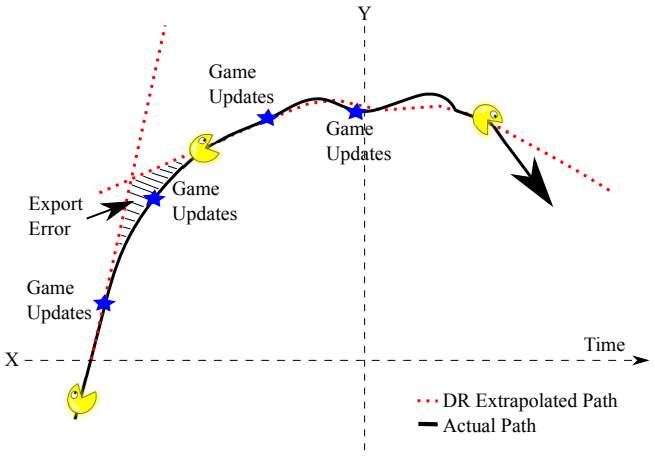


Fig. 1. Dead Reckoning.

As shown in Figure 1, clients are required to maintain two parallel models, an extrapolated path often called ghost model and an actual path that represents the true state as computed from player inputs. The ghost model represents an approximation of the true entity state. Game-state updates are used to ensure consistency between the two models. Within a game session, clients agree on a predictive contract mechanism. Locally, the predictive contract also ensures the two models do not deviate beyond a threshold. In most gaming environments, dead reckoning mechanisms often rely on additional convergence techniques to smoothly correct the inconsistencies which arise due to inaccurate extrapolation.

While dead reckoning may be invisible to the players, in real time there is a deviation at the receiving player between the actual and the extrapolated trajectories. This deviation is known as the *dead reckoning error*. It is calculated by periodically comparing the deviation between the actual and extrapolated paths. The local clients compare the dead reckoning error to a suitably defined error threshold to dynamically determine whether the game object deviates too far from reality and thus requires a game-state update.

IV. DRS: Dead Reckoning Sleep Algorithm

The idea of the proposed Dead Reckoning Sleep (DRS) algorithm is to exploit dead reckoning to predict periods of inactivity in the wireless device during game play. Figure 2 illustrates typical game interactions with the wireless network interface. As can be seen in the figure, quiet periods exist for which the device is not transmitting updates. Putting the wireless interface into a lower energy state during these periods would realize considerable energy savings and allow extended play times. Wireless radios operate in different modes. Common modes include *transmit*, *receive*, *idle*, and *sleep*. From Table I, one can see that the wireless radio in its idle state consumes about 23 times more power than it would in sleep mode.

Since updates occur once the dead reckoning error exceeds a predefined threshold, we can use how close our error is to this threshold as a predictor for how long it will take before

Power Mode	Power Consumption (W)
Sleep Mode	.045
Idle Mode	1.04
Receive Mode	1.3
Transmit Mode	1.88
Sleep to Idle transition Mode	2.08[1]
Idle to Sleep transition Mode	2.08[1]

TABLE I
POWER CONSUMPTION FOR CISCO AIR-PCM350 CARD.

[1]It is suggested in [9] that transitioning uses twice the idle power consumption.

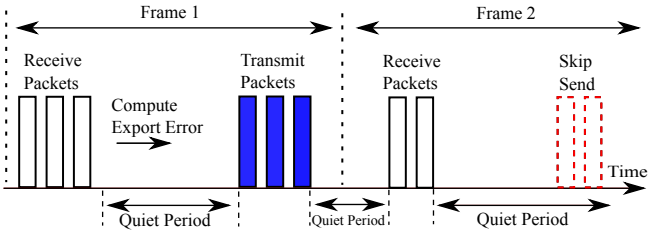


Fig. 2. Potential sleep periods.

the next update will occur. Each frame of the game produces a dead reckoning error. We track the time between this error and the time the next update occurs and use a weighted moving average to estimate how long the interface can be put to sleep. The objective is to put the wireless device into a low-energy sleep state for as long as possible, but to have it awake and ready to transmit before an update is necessary.

We assume that if we are running in ad-hoc mode, that one of the clients will act as an authoritative host. The authoritative host will cache transmissions from other players until the time at which their wireless is awake. Using an authoritative host means that game play for the ad-hoc mode is similar to that using an access point and we do not need to differentiate between these two modes. The player cast in the role of authoritative host would not be able to put his wireless interface to sleep while acting in this role. Fairness would dictate that the role be shared among players, or structured in such a way that players without battery limitations serve in this capacity. A simple mechanism for rotating the role of the authoritative host in a round-robin manner can be used.

The intuition behind DRS is that the interval for which the wireless in not needed is longer when the dead reckoning error is far from the threshold. A game client usually keeps track of multiple dead reckoning variables for the game objects under its control. For example, the tank game BZFlag [6] uses dead reckoning variables for the tanks orientation, which is the direction it is facing, as well as its position. We can divide the threshold value for each dead reckoning variable into n regions. Each region will act as a storage bin for the statistical information used to predict when the wireless interface will be needed. For each dead reckoning variable we have an array of n storage bins indexed based on the dead reckoning error, x_{error}. For example, if n = 10, the dead reckoning variable x will have the following bins:

- $0\% < x_{error} \leq 10\%\ of\ threshold = Bin\ 0$
- $10\% < x_{error} \leq 20\%\ of\ threshold = Bin\ 1$
- :
- $90\% < x_{error} \leq 100\%\ of\ threshold = Bin\ 9$

We expect when the error falls within the range of bin 9 that the wireless will be needed sooner than when the error falls within the range of bin 0.

The DRS algorithm is summarized in Figure 3. In each frame of game play, the dead reckoning error is calculated for each dead reckoning variable. If none of the variables exceeds its threshold, we find the bin with an error range spanning the calculated error and place both the bin and the current timestamp into a queue. We use an exponentially weighted moving average of the potential sleep times from this bin to predict a sleep time. If one of the variables does exceed its threshold, an update event is triggered. The bins are dequeued and the weighted averages, $estST_i$ where $i = 0, ..., n-1$, are updated according to (1). We define $currentInterval_i$ as the time when the dead reckoning error value fell within the range of bin i to the time of the next dead reckoning update. The weighting factor, α, is used to give more weight to recent sleep duration samples. If the wireless interface is sleeping, a command is issued to begin waking up the card. Otherwise, a state update is sent.

$$estST_i = (1-\alpha) \cdot estST_i + \alpha \cdot (currentInterval_i) \quad (1)$$

We obtain a prediction for the amount of time we can safely put the wireless device to sleep using (2), where $DevST_i$ is a measure of the variability of sleep times and is calculated in (3) using β as a weighting factor. We introduce γ_i as a conservative offset factor to mitigate the variability and to ensure we do not sleep too long. Accounting for variability in our prediction gives a more cautious sleep estimate.

$$sleepTime_i = estST_i - \gamma_i \cdot (DevST_i) \quad (2)$$

$$DevST_i = (1-\beta) \cdot DevST_i + \beta \cdot |estST_i - currentInterval_i| \quad (3)$$

We want the sleep time predictions to be increasingly more conservative as the threshold is approached. Ideally, the wireless should be awake before it is needed. We accomplish this by varying the γ-factor used in each bin. The bins which are furthest from the threshold need not be as conservative as those closer. Thus, we apply a smaller γ-factor to them. If the factor is large enough near the threshold, the sleep prediction will result in a negative sleep time. In this situation, the wireless is not put to sleep and is able to transmit when the threshold is exceeded.

For multiple dead reckoning variables, each variable will produce a sleep duration estimate. We choose the sleep duration of the bin with the smallest estimate as we would expect that threshold to be exceeded first. Figure 4 shows an example where two dead reckoning variables are utilized and each bin covers a 10% range of the threshold.

Algorithm 1: DRS: DR Sleep Algorithm

Input: *N:* Number of DR variables
Input: *error[], threshold[]:* DR errors and thresholds
Input: *PSP:* Power saving profile
Input: Wireless state
Input: *Q:* Queue for DR error bins
for $i \leftarrow 0$ **to** $N-1$ **do**
  if $error[i] < threshold[i]$ **then**
    Add bin corresponding to $error[i]$ to Q;
    $sleepTime[i] \leftarrow 0$;
  else
    Update weighted averages of queued bins;
    Empty Q;
    if *wireless is sleeping* **then**
      Wake wireless;
    else
      Send update;
    end
  end
end
Put wireless to sleep for $PSP \cdot \min_{0 \leq i \leq N-1}(sleepTime[i])$;

Fig. 3. The proposed algorithm.

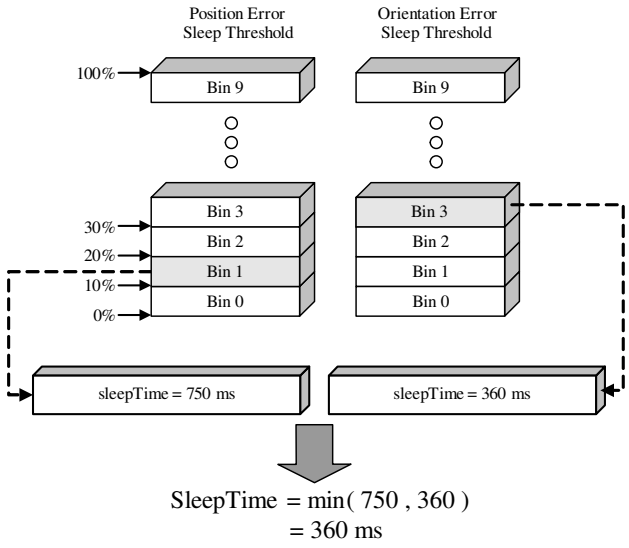


Fig. 4. Threshold partitioning.

Our algorithm will also allow the player to control his power savings profile. The DRS algorithm returns a sleep time for the wireless device. The user can adjust the aggressiveness of the algorithm by applying a factor to the returned sleep time. The factor ranges from 0 (wireless never sleeps) to 1 (the full estimated sleep time is applied). In this way, the player is able to optimize the trade off between performance and battery savings. In our analysis, we set this factor to 1.

V. Evaluation of DRS Algorithm

In this section, we conduct several simulation experiments to quantify the energy savings potential of our algorithm and the effect it has on the quality of game play.

A. Setup and Performance Metrics

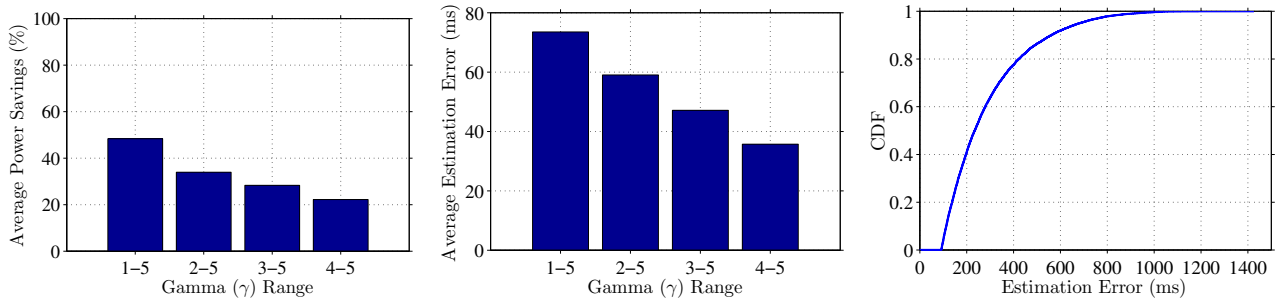
We have modified the Game Latency Simulator (GLS) system [11] by adding a wireless controller module which implements our DRS algorithm and uses a power consumption model based on the characteristics of the Cisco AIR-PCM350 [1] wireless network adapter. The power characteristics of the chosen card are summarized in Table I. Transitions between the idle mode and the sleep mode require a small duration of time (250 μsec) and consume additional energy. However, a considerable amount of delay (up to 300 ms) is encountered when the wireless interface wakes up due to reassociation with the access point. GLS is based on BZFlag [6], a multiplayer tank game in which several players drive tanks in a battle field and shoot each other for as many kills as possible.

The controller module receives the dead reckoning errors from the simulator at a rate equal to the frame rate of the game. Upon receiving these errors, it updates the sleep duration estimates and decides whether or not to put the interface to sleep (and for how long). We conducted three sets of experiments. In the first set, we study how the γ-range choice affects the performance of the algorithm. In the second set of experiments we vary the granularity of the threshold partitions between 5 bins to 20 bins. And in the third set, we vary the frame duration from 20 to 120 ms. For each set, we simulate a two hour game session between two players. We repeat the simulation 10 times for every value of the parameter under consideration using different seeds and obtain the averages. The chosen values for the parameters α and β are 0.125 and 0.25, respectively. The default values for the parameters if not under consideration are: 40 ms frame duration and 10 bins per threshold partition.

The three metrics used in our evaluation are the *energy savings*, *average estimation error*, and *average position deviation*. Energy savings is the percent-difference between the energy consumed by the wireless interface always idling and the energy used by our DRS algorithm. Average estimation error is a measure of how good our estimated values are. An estimation error occurs when the game needs to send an update but is not able to do so because the wireless interface is either in sleep mode or in one of the transition modes. This error is the difference between the time the sleeping interface awakes and the time the dead reckoning error was first exceeded during this sleep period. Unless otherwise specified, the average is taken over all the cycles in which we put the interface to sleep. Average position deviation is an objective criterion of the quality of a gaming session and measures how far the predicted position of a player deviates from his real location. The unit of measure for the position deviation depends on the game's geometry assumptions. Without loss of generality, we assume here that the deviation is in meters.

B. Results

To determine the effect that the γ-range has on the performance of DRS, we chose a number of ranges starting from $\gamma^{initial}$ for the first bin, and ending at γ^{final} for the last bin. We fixed γ^{final} at 5 and ran the simulator using the



(a) Average power savings (b) Average estimation errors (c) Estimation errors CDF for range 3-5

Fig. 5. Results for different γ ranges.

ranges shown in Figure 5(a). As shown, the γ-range 1 to 5 yielded 48.35% energy savings while range 4 to 5 resulted in a saving of 22.19%. Since we get higher energy savings when we start from a smaller $\gamma^{initial}$, we might be tempted to choose that range. However, 26.8% of all sleep periods were in error for the 1 to 5 γ-range because we needed to send an update sooner than the estimated sleep duration and the average position deviation was 0.471, Figure 5(b), compared to only 12.5% errored sleep cycles for the 4 to 5 range and an average position deviation of 0.315. Improved accuracy from a smaller $\gamma^{initial}$ is a result of longer sleep times as the lower bins become less conservative. We now take a closer look at those sleep durations for which our estimate turned out to be incorrect. We plot the cumulative distribution function (CDF) for the estimate errors for the 3 to 5 range case, considering only the sleep cycles with errors, in Figure 5(c). As shown in the figure, about 60% of the estimation errors are 300 ms or less and only 10% of those errors are greater than 600 ms. This shows that our algorithm is fairly accurate in estimating how long the interface can be put sleep. For the remaining experiments, we use the 3 to 5 γ-range.

Next, we focus on the first metric in our evaluation and study the achievable energy savings using DRS. By varying the granularity of the partitions, we did not notice a significant change in the achieved power savings which remained around 28% with the frame duration set to 40 ms. However, as can be seen in Figure 6, where *Base* is the baseline case without using DRS, the energy savings achieved are more pronounced at higher frame rates. The savings percentage dropped from 36.5% at a frame duration of 20 ms to 12.76% for a frame duration of 120 ms. This is to be expected since smaller frame times give finer grained control of the wireless interface. The DRS algorithm is run in each frame. If the frame duration is larger, the algorithm runs less frequently.

Figure 7(a) shows the average sleep time estimation errors using four different partition granularities. As we increase the granularity from 5 to 20 bins, the average estimation error is reduced from 49.58 ms to 39.53 ms. However, as shown in Figure 7(b), we noticed that the average sleep time estimation error increased almost exponentially as the framerate is increased. The error was 7.91 ms at a frame duration of 120 ms and climbed to 70.62 ms for a 20 ms

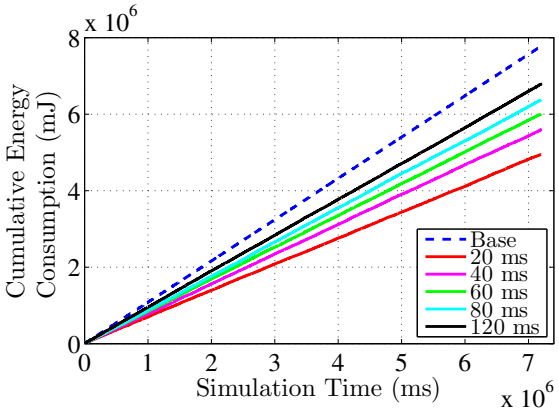
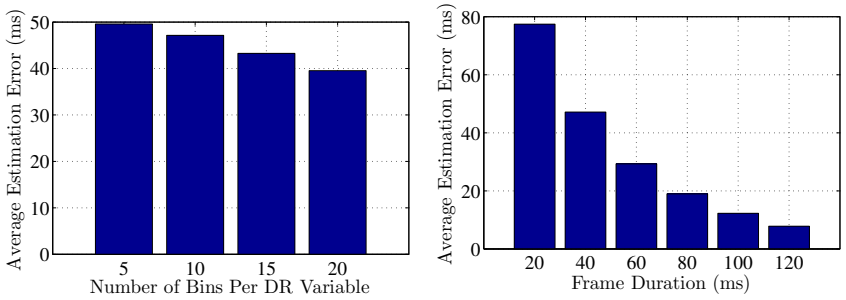


Fig. 6. Cumulative energy consumption at various frame durations.

frame duration. This is because at higher framerates, sleep durations will span more frames with the first frame being closer to the beginning of the sleep cycle. We also observed a similar behavior with the average position deviation. Because the position deviation is calculated every frame, at high frame rates if the wireless interface is sleeping and the threshold gets exceeded, it may take several frames for interface to wake up. For those frames, maximal errors will be added to the calculated average which explains the increase in position deviation. If the frame duration is longer, the interface can wake up before the next frame and send the update with only one high deviation value added to the average.

Finally, we consider how running the DRS algorithm would affect the quality of experience from the players' perspective. We first run the simulation without using the DRS algorithm and then repeat the same session using DRS and compare the average position deviation for the two cases. As shown in Figure 8, the average position deviation using DRS does not vary significantly as we increase the number of bins. The values obtained are roughly double the average deviation for the case where DRS is not used. Clearly, there is a trade off between the amount of achievable energy savings and the average position deviation experienced because of less frequent updates. However, most games can tolerate such deviations by applying smoothing techniques that transition



(a) Threshold partition granularity (b) Frame rates

Fig. 7. Average estimation errors.

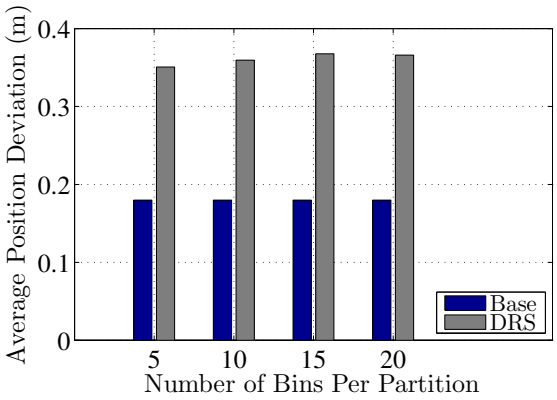


Fig. 8. Average position deviation vs. granularity of partitions.

the game object from the incorrect extrapolated position to the actual position in a smooth manner. We report here that we encountered similar results when varying the framerate in the simulation but omit the details due to space limitations.

VI. Conclusions and Future Work

In this paper, we proposed a new power saving algorithm, which uses dead reckoning to predict the wireless interface sleep cycles. Our simulation results show that power savings up to 36% can be achieved in most gaming sessions using our algorithm. We also showed the impact our algorithm had on quality of game play. We found that the effect of the algorithm on the quality of experience remains within an acceptable range even with high energy savings.

For future work, we intend to study the implications of cheating during game play on our power-management algorithm. We are also currently in the process of developing a test bed by integrating our DRS algorithm into the BZFlag game and experimenting with game sessions in all wireless and hybrid settings. Our test bed will enable us to measure real world energy savings. It will also enable us to further assess the interaction between energy saving schemes and gaming quality. In addition, we plan to extend our implementation to mobile devices such as Google's Nexus One phone.

References

[1] Cisco Aironet 350 series client adapters. http://www.cisco.com/en/US/prod/collateral/wireless/ps6442/ps4555/ps448/product_data_sheet09186a0080088828.html.

[2] id Software's Quake. http://www.idsoftware.com/games/quake/quake/.

[3] S. Aggarwal, H. Banavar, S. Mukherjee, and S. Rangarajan. Fairness in dead-reckoning based distributed multi-player games. In *Proc. of ACM SIGCOMM Workshop on Network and System Support for Games (NetGames'05)*, pages 1–10, Hawthorne, NY, October 2005.

[4] G. Anastasi, M. Conti, E. Gregori, and A. Passarella. 802.11 power-saving mode for mobile computing in Wi-Fi hotspots: limitations, enhancements and open issues. *Wireless Networks*, 14(6):745–768, December 2008.

[5] Y. Bernierr. Latency compensating methods in client/server in-game protocol design and optimization. In *Proc. of Game Developers Conference (GDC'01)*, San Jose, CA, March 2001.

[6] BZFlag Web page. http://bzflag.org/.

[7] M. Claypool and K. Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, November 2006.

[8] A. Goyal. Mobile gaming industry to reach $18 billion by 2014. August 2009. http://www.medianewsline.com/news/138/ARTICLE/4900/2009-08-06.html.

[9] E.-S. Jung and N. H. Vaidya. Improving IEEE 802.11 power saving mechanism. Technical report, Dept. of Electrical and Computer Engineering, University of Illinoi, 2004.

[10] R. Krashinsky and H. Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. *Wireless Networks*, 11(1-2):135–148, February 2005.

[11] W. Palant, C. Griwodz, and P. Halvorsen. Evaluating dead reckoning variations with a multi-player game simulator. In *Proc. of ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'06)*, pages 1–6, Newport, RI, 2006.

[12] T. Pering, Y. Agarwal, R. Gupta, and R. Want. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proc. of ACM Mobile Systems, Applications and Services (MobiSys'06)*, pages 220–232, Uppsala, Sweden, 2006.

[13] D. Qiao and K. Shin. Smart power-saving mode for IEEE 802.11 wireless lans. In *Proc. of IEEE INFOCOM'05*, pages 1573–1583, Miami, FL, March 2005.

[14] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *Proc. of ACM Mobile Computing and Networking (MobiCom'02)*, pages 160–171, Atlanta, GA, September 2002.

[15] M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications*, E80-B(8):1125–1131, August 1997.

[16] T. Zhang, S. Madhani, P. Gurung, and E. van den Berg. Reducing energy consumption on mobile devices with WiFi interfaces. In *Proc. of IEEE Global Telecommunications Conference (GLOBECOM'05)*, pages 5–9, St. Louis, MO, November/December 2005.