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

Ahmed Hamza
aah10@cs.sfu.ca

Network Systems Laboratory
Simon Fraser University

September 28, 2010

Outline

Outline

Mobile Gaming

- Mobile gaming revenues are estimated to reach $1.5 billion in the US by 2014 [eMarketer]
  - 64 million people will play mobile games at least monthly, a number that will rise to 94.9 million by 2014
- Mobile gaming market is predicted to reach $18 billion by 2014 (%16.6 annual growth rate) [Pyramid Research]
- In 2010, factory unit shipments of game-capable mobile phones are forecasted to reach 1.27 billion [iSuppli Corp]
- In addition to commercially available games, many games have been ported to Android-based phones/devices (e.g. Kwaak3)

Motivation

- Gaming uses a lot of power
  - screen always on
  - CPU used more intensively (calculations and rendering)
  - wireless network interface for communication
- Wireless network interface card can account for up to 70% of total power consumption in mobile devices
- Muliplayer games need to send state updates to maintain game state consistency among players
- Power Consumption vs. Consistency
  - How can we reduce energy consumption of wireless interface without greatly affecting consistency?

Outline

SFU

Related Work

- Power Saving Mode (PSM)-based
  - only available in infrastructure mode
  - gaming traffic has real-time constraints [CC'6]
- Bounded-Slowdown
  - dynamically adapts sleep periods to past network activity
  - requires making changes to existing protocols and standards
- Minimize energy consumption by turning off the wireless interface [SBS'02] [ZMG'05]
  - scheduling algorithms to determine sleep periods
  - formulate a complex scheduling algorithm

Outline

Dead Reckoning

- Multiplayer games
  - **avatar games** (player controls a single character)
    - *first-person avatar:* player's view is through the character's eyes
    - *third-person avatar:* player sees the character from a distance
  - **omnipresent games** (player concurrently controls group of characters)
    - can interact with objects close to any of the characters
    - include real-time strategy games and simulation games
- After agreeing on game settings (e.g. map and rules), players form a gaming *session*
- One client is chosen as the authoritative host (to maintain consistency)

SFU

Dead Reckoning

- Dead Reckoning
  - process of estimating the *future position* of an object given its *original position*, intended course, velocity, and amount of time passed
- DR is used to hide network latency in multiplayer games
  - extrapolate behavior and state of gaming objects $\rightarrow$ can continue rendering frames even if game-state updates are late.
- A *dead reckoning vector* typically contains:
  - current position of the player (in terms of x, y, and z coordinates)
  - trajectory
- Clients agree on a predictive contract mechanism, and ensure the two models do not deviate beyond a threshold
- Deviation between actual and extrapolated trajectories is known as the *dead reckoning error*

Dead Reckoning

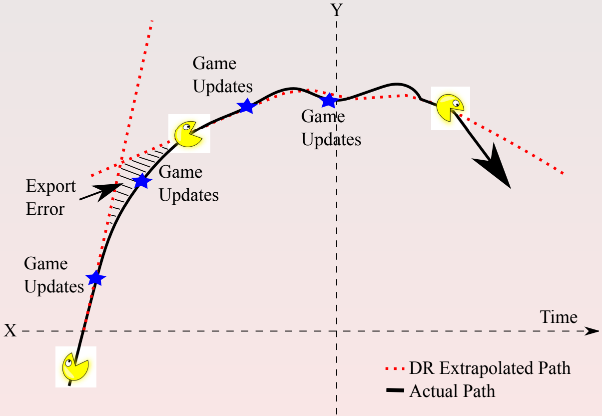


Figure: Dead reckoning

Potential Sleep Periods

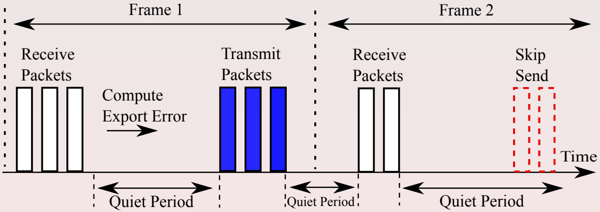


Figure: Game interactions with the wireless interface

Outline

Dead Reckoning Sleep (DRS) Algorithm

- **Idea:** exploit dead reckoning to predict periods of inactivity in the wireless device during game play
- Predict how long it will take before the next update will occur
  - based on how close the current DR error is to the threshold
- Divide threshold value for each DR variable into *n* regions
  - each region has a corresponding storage bin for the statistical information used to predict when the wireless interface will be needed
- A bin maintains a weighted moving average for the time duration until threshold is exceeded
- Caching the state update if receiver is sleeping

SFU

Dead Reckoning Sleep (DRS)

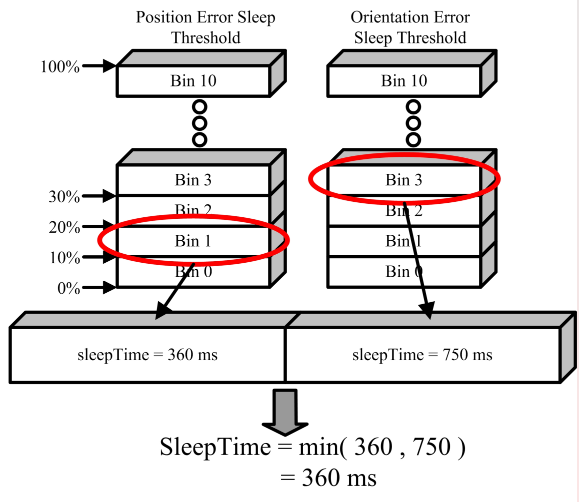


Figure: Threshold partitioning

Dead Reckoning Sleep (DRS) Algorithm

Estimated Sleep Time

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

Variability Estimation

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

Sleep Time

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

- γ conservative offset factor to mitigate the variability and to ensure we do not sleep too long

SFU

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 ← 0 to N − 1 do
    if error[i] < threshold[i] then
        Add bin corresponding to error[i] to Q;
        sleepTime[i] ← 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 $PSP \cdot \min_{0 \leq i \leq N-1}(sleepTime[i])$;

Outline

Evaluation of DRS Algorithm

- Modify the Game Latency Simulator (GLS) from University of Oslo
  - wireless controller module which implements DRS
  - power consumption model based on the characteristics of Cisco AIR-PCM350
- Simulate a *two hour* game session between *two players*
- Chosen values for α and β are 0.125, 0.25, respectively
- *Defaults:*
  frame duration = 40 ms, PSP = 1.0, granularity = 10, threshold factor = 0.8
- Evaluation Metrics
  - *energy savings*, *average estimation error*, and *average position deviation*.

SFU

BZFlag

Figure: Screen capture from BZFlag

SFU

Game Latency Simulator

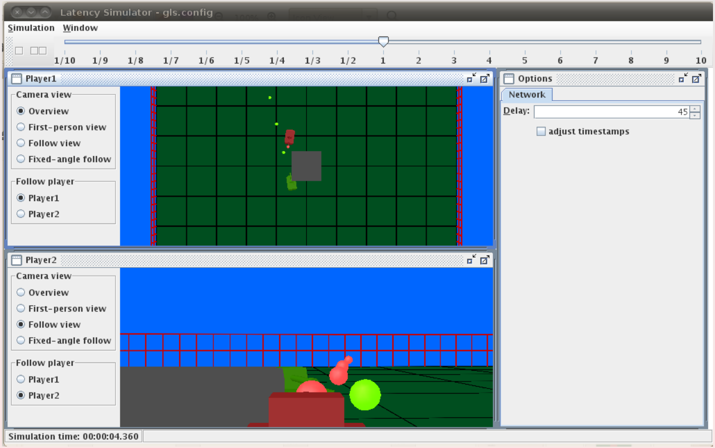
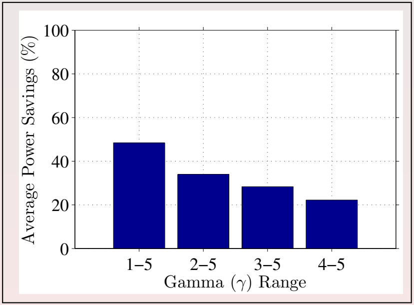
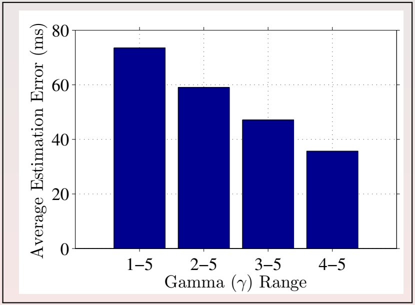


Figure: Screen capture from GLS

Gamma Effect





- Tradeoff:
  - wider γ range $\rightarrow$ more power savings
  - narrower γ range $\rightarrow$ less estimation errors

Gamma Effect

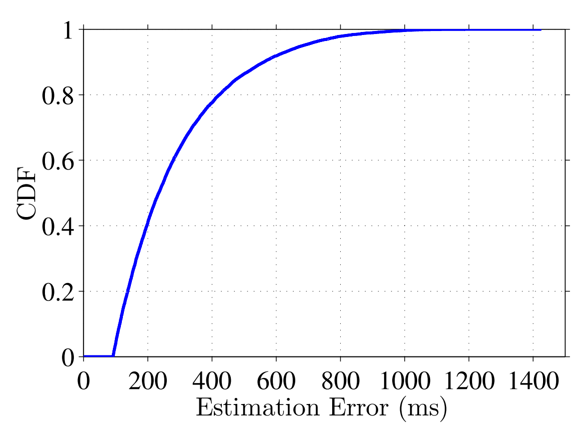


Figure: Cummulative Distribution Function of Estimation Errors ($\gamma : 3 \rightarrow 5$)

Power Savings

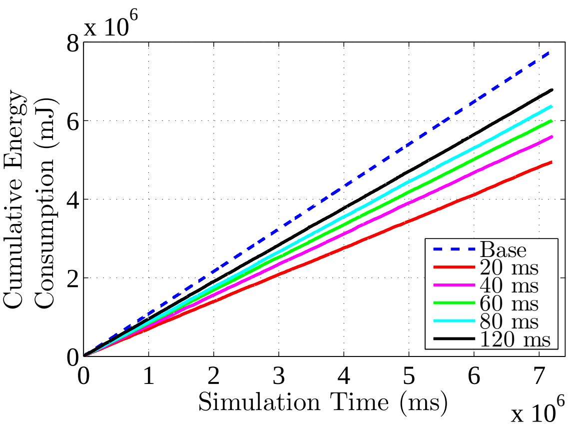
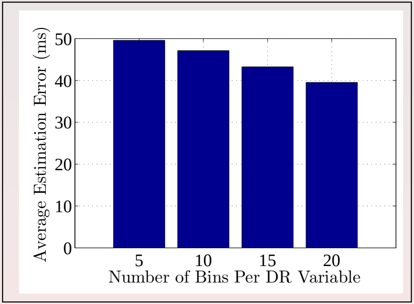
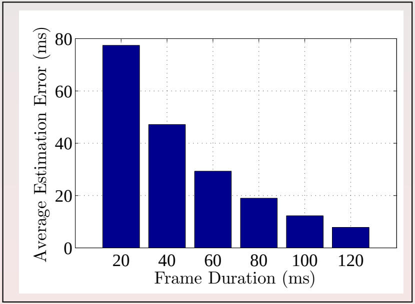


Figure: Cumulative energy consumption at various frame durations

- energy savings are more pronounced at higher frame rates

Average Sleep Time Estimation Errors





- average sleep time estimation error increases almost exponentially as the framerate is increases
    - higher framerates → sleep durations span more frames, with the first frame being closer to the beginning of the sleep cycle

Average Position Deviation

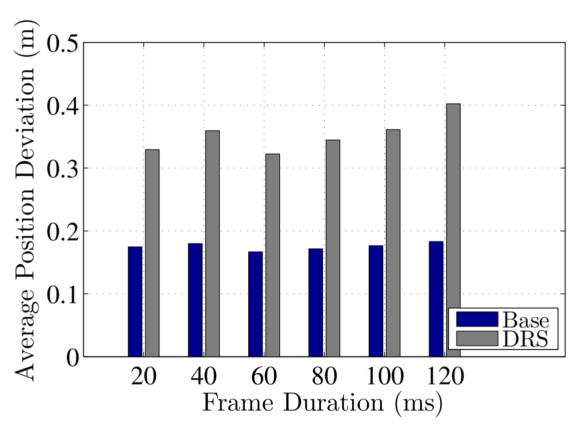


Figure: Average position deviation vs. frame duration

Average Position Deviation

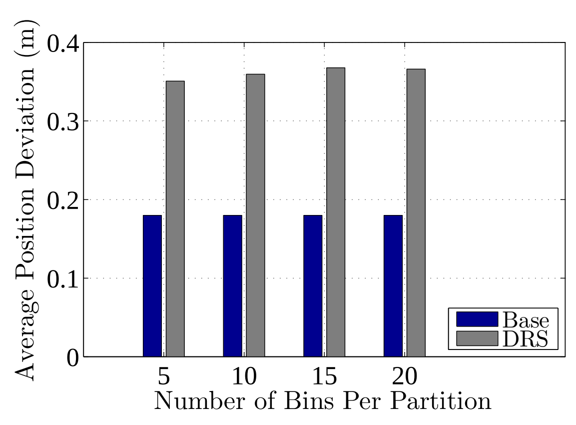


Figure: Average position deviation vs. granularity of partitions

Outline

Conclusions

- Mobile gaming is gaining popularity with a rapidly growing market
- Wireless network interface is one of the main sources of power drain in mobile devices
- Proposed a new power saving algorithm utilizing dead reckoning to predict wireless interface sleep cycles
- Simulation results show that power savings up to 36% can be achieved in most gaming sessions using the DRS algorithm
- Power savings come at some cost in terms of game state consistency

Future Work

- Reduce average position deviation using periodic checks

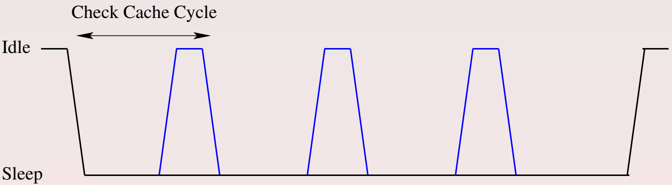


Figure: Periodic checking for cached updates

- Develop a testbed and implement DRS into the BZFlag code
- Extend our implementation to mobile devices such as the Google Nexus One phone

References

- [SBS'02]
  E. Shih, P. Bahl, and M. J. Sinclair, *Wake on wireless: an event driven energy saving strategy for battery operated devices*, MobiCom'02, 2002.
- [ZMG'05]
  T. Zhang, S. Madhani, P. Gurung, and E. van den Berg, *Reducing energy consumption on mobile devices with WiFi interfaces*, GLOBECOM'05, 2005.
- [CC'06]
  M. Claypool and K. Claypool, *Latency and player actions in online games*, Communications of the ACM, 2006.

Thank You

Questions?