Dynamic Lookahead Mechanism for Conserving Power in Multi-Player Mobile Games

Karthik Thirugnanam†, Bhojan Anand‡, Jeena Sebastian‡, Pravein G. Kannan‡
Akhihebbal L. Ananda‡, Rajesh Krishna Balan†, and Mun Choon Chan‡
†Singapore Management University and ‡National University of Singapore

***Abstract*—As the current generation of mobile smartphones become more powerful, they are being used to perform more resource intensive tasks making battery lifetime a major bottleneck. In this paper, we present a technique called *dynamic AoV lookahead* for reducing wireless interface power consumption upto 50% while playing a popular, yet resource intensive, mobile multiplayer games.**

I. Introduction

Modern smartphones are increasingly being used to play games. Games use large amount of CPU and display resources in addition to large amount of network resources for multi-player games. These are the resources that consume significant portion of the battery capacity of a smartphone — with the display and network resources each consuming close to 45% of the total battery power when fully operational [1].

In this paper, we present our solution for reducing the power consumption of a smartphone's network interface when playing mobile multiplayer games without impacting the game quality noticeably. Our solution looks for game situations where it is safe to turn off the network interface.

AoV is generally defined as the visible area around each player – with the assumption that almost all player interactions in a game are with other players that they can actually see. In general, AoV is a large circular disc, centred on the player, However, in many cases, the actual AoV is constrained by in-game obstacles such as walls and buildings. By estimating the minimum time it takes for a player to reach other player's AoV, we can accurately estimate how long we can turn-off the player's network interface.

We call this approach *dynamic AoV lookahead* and present its algorithm, implementation in a real game, and evaluation results in the rest of this paper. To test our approach, we used the Quake III Arena first person shooting (FPS) game. FPS games are the hardest to save power in as real-time player interaction events dominate in these types of games.

We tested our approach using both WiFi and cellular 3.5G (HSPDA) network interfaces, with different maps and numbers of players. Our results show that our approach is able to save, using a trace-based simulation, between 20% to 50% of the power consumed by the network interface and between 15% to 36% using a real implementation with off-the-shelf WiFi hardware. Finally, we tested the real implementation with end users and determined that our power conservation method has very little impact on perceived game quality.

This work is supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 under the research grant T208B2301. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the granting agency, National University of Singapore or Singapore Management University.

II. Related Work

There is a large set of related work in the area of power conservation and interest management. We briefly discuss them below.

Power Conservation Various authors have proposed saving power by observing and exploiting the statistical correlations between game application characteristics and load (eg. network traffic, processor cycles) [2], [3], [4].

Interest Management There are two widely used categories of techniques for Area of Interest (AoI) or Interest Management: Distance and Visibility based techniques. There are various algorithms for determining the AoI [5] such as Euclidean distance, square tile distance, and hexagonal tile distance [6], [7], more complex visibility and orientation-based algorithms [8].

The main difference of our work is that we are focusing on client-side wireless interface power conservation instead of overall game scalability. In addition, our approach has to work in real-time for a modern multiplayer FPS game that demands high frame rates.

III. Algorithm

A. Background

Modern commercial multi-player games are based on the classic client-server paradigm. The client is typically an enhanced graphic device with minimum intelligence that simply accepts inputs from the user, and advances the game state according to the server's instructions. In particular, the game server has all the information and controls all logic. Such a design is adopted for various reasons including better manageability and security, which are important issues for modern commercial games.

As explained earlier, we use the notion of AoV to determine if two players can see each other in the game. To successfully use the AoV, we need to answer two related questions. Given the real-time requirements of a mobile game application, can using AoVs tell us when it is possible to put the network interface to sleep. In addition, can it also tell us how long to sleep the interface? Our solution to these questions uses the following observations: 1) Players cannot interact with each other unless they can see each other. 2) Within a given time interval, a player can (usually) only travel a limited distance. 3) When a player is not interacting for a certain time interval, the player's network interfaces can be put to sleep during that time interval without affecting game quality.

However, when the network interface is put to sleep, the player receives no updates during that period. Problems can arise if during the sleep period, two players that were not visible to each other before sleeping become visible (and thus interacting) during the sleep period. This is the source of error/inaccuracy that we would like to avoid/minimise. Hence, the key problem is to determine when the players may become visible in the future. If this period is sufficiently large, we can save power without affecting game quality.

B. Visibility and Future Location Matrix

We assume that for any game, the maximum speed of a player is fixed. For a given interval, say β milliseconds (ms), a player can only move so far from their current location. Conceptually, our approach does the following:

- For each player, it identifies all the possible areas the player can be in β ms if they are moving at the maximum speed.
- For all pairs of players, check if two players can be visible to each other in β ms.
- For each player, if they are not visible to all other players in β ms, then a sleep interval of β ms is a "safe" interval for this player to sleep.

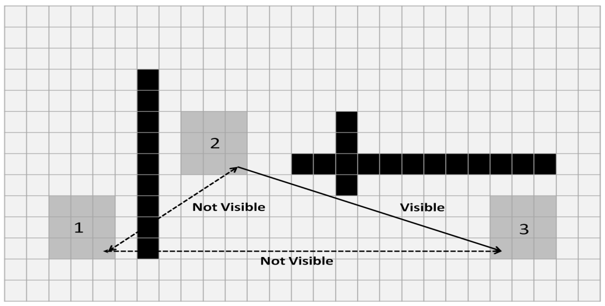


Fig. 1. Inter-player Visibility With Obstacles

Figure 1 shows an example on how visibility varies with the players' positions over time. The shaded areas around each player's number indicate the possible areas that that player can move to in β ms while the black areas indicate obstacles that obstruct visibility. From the figure, we see that Player 1 is not visible to Players 2 and 3 even after β ms — no matter how each player moves. As a result, player 1 can sleep.

In order to perform this computation efficiently, we only consider time quantum of β ms for time efficiency and we discretized the game map for space efficiency. We divide the game space into hexagonal or square grids, where the grid element width (p) is the maximum distance a full-speed player can cover in β ms. In the rest of this section, we use a 2D hexagonal grid, even though our approach works for 3D maps as well. However, for evaluation we use square grid due to its ease of implementation.

Using the grid defined above, we identify the possible future locations in multiple time steps (of β ms). Figure 2(b) shows the possible locations from the current player's position of (0) (centre of figure) that can be reached up to 3 β time-steps later. The darkest shading shows the area reachable in 1 time-step, followed by 2 steps (lighter shading), and then 3 steps (lightest shading). Note: the player could stay still and not move in each time-step. The value inside each hexagon represents direction-weight, which is explained in next section.

The visibility computation can then be performed off-line as the map layout (where the walls are etc.) and grid representation is mostly static. The visibility computation generates a visibility matrix V. The matrix entry is 1 if two grids (in particular the centre positions of those 2 grids) are visible to each other, and 0 otherwise. For maps that change dynamically, we can pre-compute matrices for the different layouts and use the appropriate matrix at runtime.

C. Dynamic Lookahead

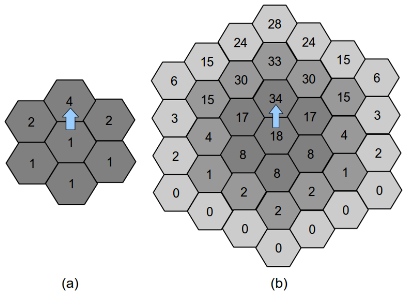


Fig. 2. Scaling Grid Weights (1 and 3 Time-steps)

To effectively use these grids, we need to also incorporate the player's movement patterns. In particular, we need to assign weights to all the likely positions that a player could be at β ms later. We can then use these weights to determine the probability of the player being able to see other players located at specific grids.

In our approach, we exploit the player's velocity to set the grid weights. This is a reasonable assumption as velocity is often part of the the game state. To illustrate how we incorporate velocity, Figure 2(a) shows the case where the player is moving up from the centre. Let w_i be the weight of moving to grid i from the current position. The grid weight for moving in the forward direction is 4. The forward plus sideways directions have weights of 2, and the rest of the directions have their weights set to 1. We do not claim that our chosen weights are optimal, only that velocity information can be utilised to improve prediction.

The weights shown above are for just 1 time-step. We used a small grid size to reduce the grid-to-grid visibility check inaccuracy. To find possible long sleep periods we extended our weights to consider multiple β time intervals. Figure 2(b) shows an example of a possible weighted grids for 3 times steps and for a player moving forward.

D. Determining Sleep Times and Intervals

We now present our final algorithm that determines when and for how long to sleep a player's network interface. Consider two players 1 and 2 at grids x_1 and x_2 respectively.

In t time steps, let the grids reachable for players 1 and 2 be L_t^1 and L_t^2 respectively.

For each point $l_1 \in L_t^1$, check its visibility to each point in L_t^2 (using the visibility grid V). Set the value $v(l_1, l_2)$ to 1 if their positions are visible to each other and 0 otherwise. For each pair (l_1, l_2), the cost is computed as $w_{l_1} * w_{l_2} * v(l_1, l_2)$. We compute the likelihood of player 1 seeing player 2 as the following normalised sum:

$$S_t(1,2) = \frac{\sum_{l_1 \in L_t^1, l_2 \in L_t^2} w_{l_1} * w_{l_2} * v(l_1, l_2)}{\sum_{l_1 \in L_t^1, l_2 \in L_t^2} w_{l_1} * w_{l_2}} \tag{1}$$

Let N be the set of all players, T be the set of all discrete sleep time intervals considered and α be a control parameter. The parameter α, $0 \leq \alpha \leq 1$, controls the aggressiveness of the power saving mechanism, with 0 being the most conservative and 1 the most aggressive (always sleep). By varying α, we can obtain different trade-offs in power consumption versus accuracy. The algorithm is outlined below:

```
For each player $i$
    For each time slot $t \in T$
        For each player $j \in N \setminus i$
            compute $S_t(i, j)$
        $t$ is feasible if $S_t(i, j) \leq \alpha, \forall j$
    Select the largest feasible $t$ as user $i$'s sleeping interval,
    otherwise user $i$ does not sleep.
```

This algorithm is executed every β ms on the sever, but control data is transmitted to the client only when the client is awake.

IV. Quake III Implementation

We used the commercially successful Quake III Arena [9] 3D FPS game as our test game.

On average, the traffic rate from the server to clients is 16Kbps (≈20packet/sec) and 21Kbps (≈40 packet/sec) in the reverse direction. The key information sent from a client to the server is the player's current position and action (e.g. shooting). The key consequence of sleeping a client's network interface is that the player's position updates could be delayed significantly. As discussed previously, we consider this delay to be a problem only if it results in state inconsistency between two or more players.

In our implementation, we added a new sleep command that the server uses to tell specific clients how long to sleep. The client will put the wireless interface in sleep mode. When it wakes up, it must stay awake for a minimum interval to ensure that it receives at least one update packet from the server, allowing the client to re-synchronise correctly with the server. We experimentally determined that this interval should be at least 100ms for WiFi as it needs time to re-establish a connection with the access point.

V. Methodology

We performed three kinds of evaluation, namely: trace-based simulations, actual system performance measurements, and a small scale user study. Our evaluation aimed to answer the following questions: 1) Game state varies from run-to-run and no two games runs are identical. However, in order to compare different algorithms, we need to evaluate different algorithms under the "same" condition. How can this be done? 2) How do we evaluate quality objectively and how do we relate the objective metric used to a player's perception of the gameplay? 3) How much power can we save in a realistic game playing environment with minimum impact on gameplay? 4) How much of the error is actually visible to the user?

A. Trace-based Simulation Methodology

1) Using Traces for Repeatability: To address the first question, we separated the trace collection from the algorithm evaluation. In the trace collection phase, the game was played "normally" and the player location, player-to-player visibility information (a record of which players are visible to each player at that point in time), and direction information was logged every 100ms. We then compared different lookahead mechanisms using the same collected traces.

2) Defining a Accuracy Metric: For the second question, we defined a miss or an error as the case when a player, on waking up from sleep, finds that it is visible to some other player.

The quality of the game is thus computed as

$$Accuracy = 1 - \frac{Total\ sleep\ time\ with\ errors}{Total\ sleep\ time} \tag{2}$$

This *accuracy* metric has the advantage that it can be easily measured using our visibility matrix. However, to support this objective metric, we evaluated the impact of accuracy levels on human quality perception through a small scale user study (described in Section VI-F).

3) Running the Simulations: We conducted repeatable experiments using a custom-built two-stage Java processor. The first stage takes the player traces and generates a predicted visibility map for various β values (200, 400, and 600ms).

In the second stage, we analyse the predicted visibility map and determine the expected power savings and accuracy for different algorithm variants for that particular trace. The power savings is computed as:

$$\%\ Power\ Saved = \frac{total\ sleep\ period}{total\ game\ period} \tag{3}$$

This percentage can be converted to actual power numbers by multiplying it with the actual power consumption of the specific wireless interface.

4) Realistic Network & Game Characteristics: We collected our game traces using two different wireless interfaces that are commonly found on modern smartphones — 1) 802.11n WiFi, and 2) two different 3.5G cellular HSPDA interfaces (2Mbps and 7.2Mbps).

We used laptops to run the Quake III Arena game client as we did not want the limited graphics capabilities of cell phones to hinder our network-interface focused experiments. Our latencies to the game server were as follows: 3.55ms average RTT (stdev. of 3.58ms) for WiFi with a 0% packet loss, 84.35ms average RTT (stdev. of 34.63ms) for a 2Mbps

3.5G connection with a 4% packet loss, and 62.29ms average RTT (stdev. of 8.91ms) for a 7.2Mbps 3.5G connection with a 1% packet loss.

We used three different game maps for all our experiments; one created by the Quake III Arena developers themselves (*q3dm1*), and two developed by us (*longroom, bigroom2*). *q3dm1* is a comparatively small map, while *longroom* is roughly twice its size, and *bigroom2* is twice as big as longroom.

B. Experimental Evaluation Methodology

In order to address the third question, we measured the actual power consumption of the network interface while the game was being played for 5 minutes on a Lenovo T61 laptop.

C. Small Scale User Study Methodology

We answered the final question through a small scale user study with 12 players to evaluate how the accuracy values defined by our metric affects a human player's perceived game play experience. The participants in our study were all graduate students in the computer science department. We first trained the players on an unmodified game, and then had them play the modified game several times with different α values. For each game play, we asked them to rate how noticeable, if at all, were any network related artifacts in the game, compared to the unmodified version, on a 5-point Likert scale.

VI. Evaluation Results

A. Baseline - No Prediction

We first provide baseline power consumption values, using trace-based evaluation, for the range of scenarios we considered. The client goes to sleep, for fixed sleep intervals of 200ms, 400ms, and 600ms, when no player is visible. The simulation results are shown in Figure 3 for the *q3dm1* map. Substantial amounts of power (up to 50%) can be saved when the number of players is small and the sleep interval is large. However, this results in low accuracy value ($< 70\%$) when the number of players increases. We show, in the rest of this section, that our dynamic lookahead algorithm can achieve large power savings with low accuracy loss.

B. Experiments over Various Networks

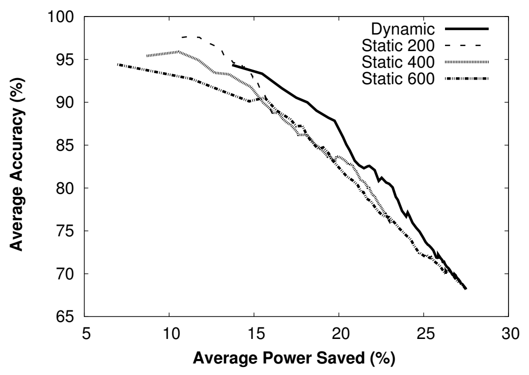


Fig. 4. Power Savings (3.5G Networks)

We performed the measurements using both the WiFi and 3.5G network traces while varying the parameter α (Section III-D) from 0 to 1.

Figure 4 shows the results for 3.5G cellular networks for the map *q3dm1* using 4 players. "Dynamic" refers to our dynamic algorithm where the maximum possible sleep interval is selected. Each line shows a smooth variation of α from 0 (best quality and the leftmost point) to 1 (worst quality and the rightmost point). The power saved is 14%, 11%, 9% and 7% for dynamic, static 200ms, static 400ms and static 600ms respectively. The accuracy for all 4 cases is above 94%. For almost all α values, the dynamic approach provides the highest power savings with the highest accuracy.

The results for WiFi, which is not shown, is similar. Finally, we also obtained similar patterns for both interfaces when using the other two maps and player numbers (2, 8, and 16). We omit those results due to space constraints.

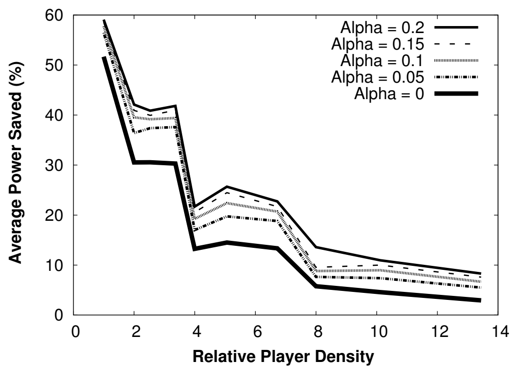


Fig. 5. Effect of Player Density

C. Effect of player density

The player density value is computed as follows:

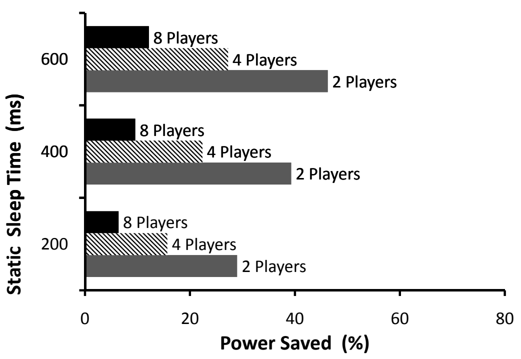
$$\%\ Player\ Density = \frac{no.\ of\ Players}{Map's\ Visibility\ Grid\ Area} \quad (4)$$

Figure 5 shows our results. To make the plot more readable, we normalised the player density by setting the lowest density to 1. Hence, the scenario with the highest density has about 14 times more players per unit area than the scenario with the lowest density.

The 5 plots represent 5 different α values. While it is clear that a more aggressive setting (larger α) always saves more power (at the expense of accuracy as shown earlier) and vice versa, there are clear regions where the power saved percentage falls into different effectiveness zones. For example, player densities between 2 to 3 can save up to 42% of the interface power while densities between 4 to 7 can save at most 25% of the interface power.

D. Benefit of Our Dynamic Algorithm

Figure 6 plots the power savings achievable by our dynamic algorithm and the 3 static algorithms for the full range of player densities (α is set to 0 (best accuracy) for all four algorithms and player densities are normalised as mentioned previously). The dynamic algorithm performs consistently the best over the entire range ($\approx$50% savings at the lowest density).



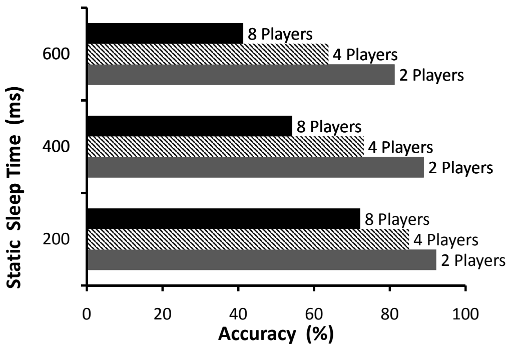


Fig. 3. Results with no Prediction (*q3dm1* Map)

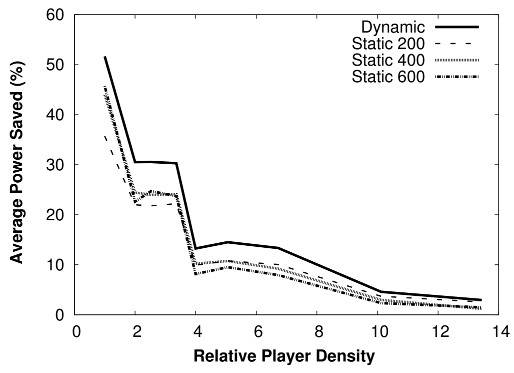


Fig. 6. Dynamic Versus Static Algorithms

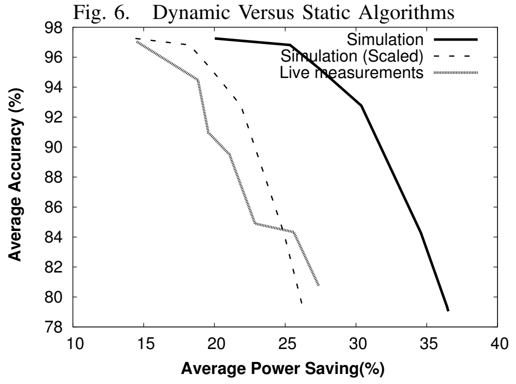


Fig. 7. Actual Versus Simulation Results

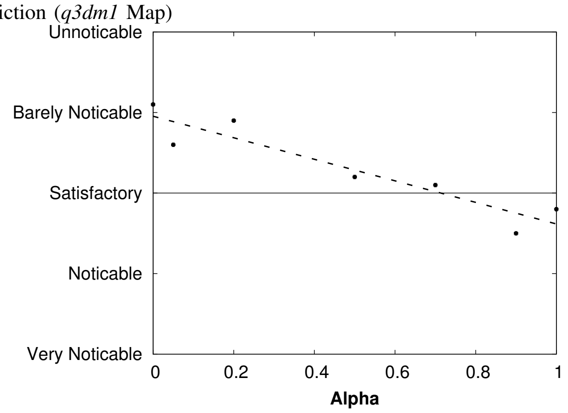


Fig. 8. User Study - Quality Loss Versus Alpha

E. Actual Power Measurements

In this section, we evaluate the correctness and accuracy of the simulation by comparing it with actual power measurements. Figure 7 shows the results. The actual power savings is less than the predicted savings as fraction of the sleep interval is taken for turning the interface on/off and that consumes energy.

F. Impact of Errors on Perceived Quality

Figure 8 shows the user study results. We excluded two players' results that had outlier ratings (always good or contradictory). The results show that most users found the system very playable even when the α is around 0.5.

VII. Discussion and Conclusion

Our algorithm assumes that other players are moving in a linear regular fashion. We are currently investigating various mechanisms to reduce the impact of non-linear movements (eg. teleportation/spawn events). Also, we are studying various mechanisms to improve the memory and processing efficiency of our algorithm.

To conclude, we have presented our *dynamic AoV lookahead* algorithm and demonstrated that it can save significant amounts of power without compromising quality. In the future, we plan to test this algorithm with other game genres.Finally, we are in the process of integrating the various power savings mechanisms for display, network and processing into a single integrated framework.

References

[1] B. Anand, K. Thirugnanam, J. Sebastian, P. G. Kannan, A. L. Ananda, M. C. Chan, and R. K. Balan, "Adaptive display power management for mobile games," in *ACM Mobisys*, 2011.

[2] Y. Gu and S. Chakraborty, "A Hybrid DVS Scheme for Interactive 3D Games," in *IEEE RTAS*, 2008.

[3] B. Anand, A. L. Ananda, M. C. Chan, L. T. Long, and R. K. Balan, "Game action based power management for multiplayer online games," in *MobiHeld*, 2009.

[4] B. Anand, A. L. Ananda, M. C. Chan, and R. K. Balan, "Arivu: Making networked mobile games green," *MONET, Springer*, 2011.

[5] J.-S. Boulanger, J. Kienzle, and C. Verbrugge, "Comparing interest management algorithms for massively multiplayer games," in *NetGames*, 2006.

[6] R. K. Balan, M. Ebling, P. Castro, and A. Misra, "Matrix: Adaptive middleware for distributed multiplayer games," in *Middleware*, 2005.

[7] I. Kazem, D. T. Ahmed, and S. Shirmohammadi, "A visibility-driven approach to managing interest in distributed simulations with dynamic load balancing," in *DS-RT*, 2007.

[8] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: enabling large-scale, high-speed, peer-to-peer games," in *ACM SIGCOMM*, 2008.

[9] *Quake 3 Arena Source Code*, http://ioquake3.org/, Id Software, Jul. 2010, (Version 3.21).