Energy Efficient Multi-player Smartphone Gaming using 3D Spatial Subdivisioning and PVS Techniques

Bhojan Anand, Zeng Qiang, Ananda L Akkihebbal
School of Computing, National University of Singapore
banand@comp.nus.edu.sg, zen1986@gmail.com, ananda@comp.nus.edu.sg

ABSTRACT

With the advent of feature rich smartphone platforms such as Android and iOS, people can now enjoy a wide variety of applications on-the-go. Among these applications, games are one of the most desired types. However, as bigger screens, faster CPUs and interfaces supporting higher bandwidth (WiFi, 3G, LTE) consume more power, battery lifetime becomes a bottleneck on such devices.

In this paper, we present novel techniques that combine *3D spatial subdivisioning, Potentially Visible Set (PVS)* and *Visual Perception based Localisation (VPL) methods* to estimate the non-critical game states to save wireless interface energy with minimum processing penalty. Our techniques and algorithms are realised in a commercial game and can save up to 57% of wireless interface energy while retaining game play quality.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Wireless Communication; K.8 [**Personal Computing**]: Games

General Terms

Algorithms, Design, Experimentation, Human Factors

Keywords

Energy; Power; Wireless; Game; Mobile; Smartphone

1. INTRODUCTION

Energy. Growth of battery technology is not on par with the rest of the technologies in modern smartphones. As most of us are aware, very few systems today fulfill Mark Weiser's vision of *"several days of continuous use"* for the *"Computer for the 21st Century"*. The three main sources of power consumption in smartphones are, 1) the processor (CPU/GPU), 2) the display, and 3) the network interfaces. For the HTC Magic (Android samrtphone), the measurements show 35-40%, 45-50% and 4-15% power consumption by the wireless interface, the LCD and the rest (CPU, memory...) respectively [3]. In this paper, we focus on conserving power at the network interface level.

Games. Games are one of the most-downloaded categories of smartphone applications. Mobile games are resource-intensive applications and they consume more power than other common applications. For example, in our measurements on the HTC Desire HD (Android smartphone), the battery provides 8 hrs of GSM talk time, while it lasts for only 1hr 50min when the 3D game Kwaak3 [11] is played [1]. In this paper, we focus on conserving power at the network interface level in networked multiplayer 3D gaming environments.

Network Interface. The key mechanism to reduce the power consumption of wireless interfaces is to put the interface into sleep mode whenever possible. The challenge is to decide, 1) *when to put the wireless interface into sleep mode?* and, 2) *how long can it be in sleep mode without affecting the quality of the game play adversely?*. An obvious answer would be to *sleep when the game state is not important*, from the player's perspective. Almost all player interactions in a game are with other players who are in a perceptible range. Hence, the game state of a client is important when there are other players in its perceptible range. However, *predicting the near-future game state for a client is challenging*.

In this paper, we introduce a novel technique that uses the 3D rendering engine's knowledge to determine the importance of the game state. Modern game engines use occlusion culling algorithms [18, 9] to compute PVS for efficient rendering. We use this PVS data to determine visibility between two players. In contrast to previous methods that overlay simple 2D squares or hexagons on the 3D maps [10] to compute distance or visibility between players, we propose using the 3D data directly from the renderer. Hence our approach is efficient (minimum computations are required), accurate (we handle the 3D world as it is without overlaying any 2D structures) and easy to integrate with modern game engines (PVS is already implemented in most of these modern game engines). In addition, we use VPL, to filter players based on their distance to improve scalability. For VPL, we compute Maximum Perceptible Distance (MPD) which is the distance up to which a player can view other players or bots and interact. As human behaviour is hard to predict, we use a feedback based error controlling mechanism to keep the errors within a given threshold.

To test our algorithms, we used the commercially successful Quake III [14] First Person Shooting (FPS) game and its Android port (Kwaak3 [11]). We used Quake III because it is open-sourced. As a fast paced FPS game, Quake III brings maximum challenges to our approach by demanding lower processing time and network latency.

2. RELATED WORK

Power Conservation A large body of prior work has also looked at saving wireless interface power. Meyer et. al [17] show that power can be saved through better link utilisation while Anastasi et. al [4] propose a proxy assisted power saving mechanism. Various authors have also proposed saving power by observing and exploiting the statistical correlations between applications and the

[1] Kwaak3 is an Android port of the game Quake III [14]

MM'13-IMMPD'13, October 22, 2013, Barcelona, Spain.

http://http://dx.doi.org/10.1145/2505483.2505491.

network traffic [6, 22, 23]. These statistical methods have also been applied to games [12, 13, 1].

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 [10] such as Euclidean distance, square tile distance, hexagonal tile distance [19, 8, 2, 15] and more complex visibility and orientation-based algorithms [7]. However, in all these works the authors overlay a 2D grid over a 3D game map handling the 3D environment inaccurately. Such basic approaches, resulted in more errors as the number of players increased. Moreover, the pure visibility or AoV (Area-of-Visibility) based algorithms fail to work for huge open maps where the players are always visible to each other. In addition, pure AoV based approaches [19] are not scalable. If there are n players, for each player p_i, we always need to check the remaining $n-1$ players for visibility. Hence the computational cost will be $O(n^4)$ for checking and dynamically predicting visibilities between each other.

There are two main differences in our work. Firstly, we consider distance and visibility in 3D for improved accuracy. Secondly, we use the 3D rendering engine's data directly to avoid additional computing overhead and to increase the efficiency of our algorithms.

3. POWER MANAGEMENT ALGORITHMS

Our power management algorithms are run in two phases, namely *macro scanning* and *micro scanning*. Macro scanning filters the players by distance and passes only relevant players for micro scanning. Micro scanning queries the PVS from the current area of a player and puts his wireless interface to sleep mode for a period of time.

The pseudo code presented in *Algorithm 1* outlines the main algorithm.

Algorithm 1 Main

```
for each server frame do
    for each active client do
        if client is sleeping then
            continue
        else
            do t = Macro_scan()
            if t > 0 then
                sleep for t msec
            else
                do t = Micro_scan()
                if t > 0 then
                    sleep for t msec
                end if
            end if
        end if
    end for
end for
```

Area, Cluster and Portal in a Game World. *Area* is a space, partially enclosed by geometric surfaces. Adjacent *areas* that share same surfaces are grouped together to form a *cluster* [21]. Clusters are separated by a *portal*. For example, door is a portal, because player can travel from one cluster to another through it. Thus, a map is logically divided into clusters through portals. This graph structure together with areas forms a hierarchical network.

3.1 Macro Scanning Algorithm

We use *path-distance* to measure the distance from the position of the current player to the clusters containing the MPD end point as shown in the Figure 1, so only necessary clusters are added to the queue for processing. Breadth-First Search (BFS) is used to traverse clusters in nearest-first order and stopped at the MPD of the current player. *Current player* is the player or client who is currently under consideration for power management. Detailed pseudo

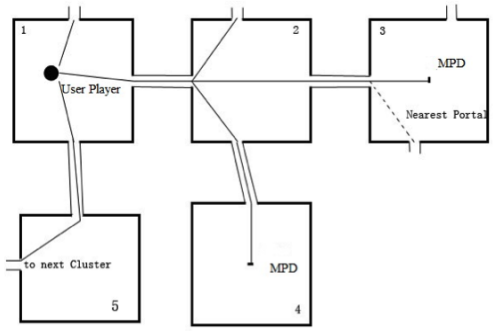


Figure 1: BFS follows the Path Distance of each Cluster

code for macro scanning is presented in *Algorithm 2*. The algorithm initialises sleep time to maximum sleep time. Maximum sleep time is game specific. Setting the wireless interface to sleep mode beyond this duration produces noticeable latency in game play. It is about 180ms [5] for fast paced FPS games and 400ms for slower RPG games.

Algorithm 2 Macro_scan

```
clusterQueue: initialize to contain the current player's (player_i) cluster
playerQueue: initialize to empty, will pass to micro scan
sleepTime: initialize to maximum sleepTime (game dependent value)

while clusterQueue is not empty do
    cluster = pop clusterQueue
    for each player_j in cluster; j ≠ i do
        if player_j's distance to player_i > MPD then
            calculate time player_j takes to MPD of player_i using MS
            if time < sleepTime then
                sleepTime = time
            end if
        else
            put player_j in playerQueue
            sleepTime = 0
        end if
    end for
    for each portal_k of cluster that has path-distance to player_i < MPD do
        clusterOther = portal_k->otherCluster
        add clusterOther to clusterQueue
    end for
    for each portal_q of cluster that has path-distance to player_i > MPD do
        calculate the time taken from the portal_q to MPD
        if time < sleepTime then
            sleepTime = time
        end if
    end for
end while
return sleepTime
```

The distance from any player to the current player's MPD can be easily calculated from the positions of the players. We compute the sleep time for the current player by dividing this distance by the players' relative movement speed as shown in Equation (1). As human behaviour in a game play is complex, it is hard to predict when and at what speed (and direction) the user will move his character. Hence, computing relative movement speed for each pair of players (n^2 combinations in worst case) at every frame is complex and would incur too much overhead. To keep it practically feasible, we use *estimated average movement speed* (here after we simply refer it as *movement speed*). If this estimated movement speed is lower than actual movement speed of the players, there are more possibilities for errors. Hence, the estimated movement speed is dynamically tweaked based on error rate, which we describe in the section 3.3. *Error rate* is a ratio of *total sleep time with error* to *total sleep time* in a given period.

We stop traversing the clusters after reaching the MPD. Once the MPD is reached, we conservatively assume that there could be a player in the next (unexplored) cluster, just behind the portal (door), and he could enter the current cluster (and the MPD of current player) in no time. To cater for such incidents, the algorithm takes the distance between the MPD of the current player to the nearest portal (excluding the portal through which the BFS enters the cluster) in the current cluster and computes the sleep time based on this distance. The shortest of all the sleep times is returned by the algorithm. Note, if the BFS finds another player within the MPD distance of current player, it will return zero as sleep time.

$$Sleep\ Time = \frac{MPD_{distance}}{2 \times MS} \quad (1)$$

where, $MPD_{distance}$ *- is distance from a player to the MPD of the current player; MS - is estimated movement speed of players.*

As BFS scans the clusters in incremental order and stops at MPD, our macro scanning is naturally scalable. In huge maps (as in MMOG games), it is highly efficient.

3.2 Micro Scanning Algorithm

As our spatial subdivision scheme (eg. BSP tree in Quake III game, Octree in Ogre3D engine) is pre-computed by 3D renderer and stored together with the data structure of the game map, micro scanning becomes simple and efficient. For the current player under consideration we get his PVS from the renderer and check whether any of the players in the player queue (created by macro scan) is present in the PVS. The PVS data is usually stored in the form of bit-array. For example, in Quake III, *Area x is visible to area y if* $(1 << y\%8)$ *bit of* $PVS[x*no.of\ areas+y/8]$ *is set*. Assuming the bit-array structure for PVS, the pseudo code for micro scanning is described in *Algorithm 3*.

Algorithm 3 Micro_scan

```
playerQueue: passed as parameter
currentPlayer: index no. of current player

for each player inside playerQueue do
    if (1<<player%8) & PVS[currentPlayer * no. of areas + player/8]==1 then
        return NO_SLEEP
    end if
end for
return FIXED_SLEEP_TIME
```

In micro scanning, the time to sleep is always fixed. We call it Fixed Sleep Time (FST). When there are no other players in the current player's PVS the client is put into sleep for a FST time. If FST is too high, there are more possibilities for errors. The FST value is dynamically tweaked based on the error rate, which we describe in Section 3.3.

3.3 Feedback based Error Controller

There is a trade off between the error rate and amount of power saved. The error rate always grows together with the total power saved. The error threshold (maximum acceptable error rate) really depends on how much an user can bear with the affected game experience. At the time of low battery level, he may want to sacrifice a bit to save more power.

To control the errors according to the Error Threshold (ET) set by the user, we need to tweak two complex parameters - Movement Speed (MS) described in Section 3.1 and Fixed Sleep Time (FST) described in Section 3.2 to optimal levels for maximum energy saving. In our current implementation we have used Additive Increase and Multiplicative Decrease (AIMD) method to make our system more conservative on quality. Once the error threshold is passed we enforce large changes (multiplicative) to FST and MS to bring down the error as fast as possible. The key advantage of multiplicative decrease is that it enables all other clients to get the correct state (eg. position) information about the current player much faster, resulting in rapid global game state synchronisation among all clients.

As the game activities are very random and human behaviours are hard to predict, such feedback based control systems are very robust and adaptive to different requirements. The pseudo code presented in *Algorithm 4* outlines the AIMD based error control procedure. Average Error Rate (AER) in the algorithm is the running average error rate for the past 't' time period. Running average is used to avoid any abrupt changes to FST and MS. Note: After several sets of experiments we have arrived at the values *2* and *1/4 of the current value* for increment or decrement. These values gave reasonably fast convergence and a well controlled error level.

Algorithm 4 AIMD Based Run-Time Error Controller

```
- Initialise: FST = 200ms, MS = 500 game unit/ms. Game unit is pixels in most
games. Movement speed here is the normal average movement speed in game. 500
pixels/ms is obtained from multiple game play experiments.
- Let the user set an ET between 1%, 3% and 5%. Higher threshold results in lower
quality and higher power saving. It determines the trade-off between quality and
power.
- Assumption : Main algorithm is running (macro and micro scanning).
while game is not end do
    update AER
    if AER < ET then
        FST = FST + 2
        MS = MS − 2
    else
        FST = FST + 1/4 * FST
        MS = MS − 1/4 * MS
    end if
end while
```

4. IMPLEMENTATION

The power management algorithms were implemented and tested in a community maintained copy of Quake III source called ioquake3 [14].

We implemented our algorithms in the Quake III server as the server knows the entire state information of the clients. We used Ubuntu 10.10 with Eclipse as our development platform. The server estimated the client's state and the time period that the client could sleep without affecting the quality of game play adversely.

In our implementation, we added a new *sleep command* on top of the existing client/server communication in the game. The server used this sleep command to tell specific clients how long to sleep. Upon receiving this command, the client sent a signal to the wireless hardware to make it sleep for the specified time period.

To put the wireless card into sleep mode and wake up fast we used Madwifi driver [16], which is an advanced Linux based WiFi driver for WiFi cards with Atheros chipset. MadWifi comes with *Atheros Hardware Access Layer* (HAL). The HAL provides hardware support for wireless network adapters. More details on this can be obtained through the ubuntu man pages [20].

5. EVALUATION METHODOLOGY

Figure 2 shows the testbed used for our experiments. The game server was implemented and ran on a Dell Precision T7500 computer which came with Intel(R) Xeon(R) E5520 2.26GHz/8MB L3 Cache/12 GB RAM. The game clients were connected through a wired network and different wireless networks (WiFi, 3G) to the game server. We used a variety of Lenovo Thinkpad laptops (Model T61/T60/W500) and Android phones (HTC Magic/Dream/Nexus One/HTC Desire HD) as game clients. To obtain accurate power measurements, we used a high-speed multifunction Data Acquisi-

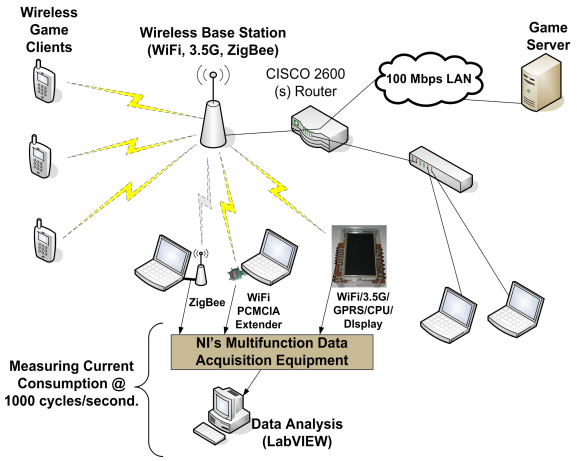


Figure 2: Testbed: Game Play and Power Measurement

tion Equipment (DAQ) USB-6251 and a Signal Conditioning Equipment (SC-2345) from National Instruments (NI).

5.1 Defining Evaluation Metrics

In this section we describe our evaluation metrics.

Definition of Error. We define *a miss* or *an error* as the case when a client, on waking up from sleep, finds that there is some other client in its *PVS* and *MPD*.

Sources of Errors. There are 3 scenarios for an error to occur: 1) Other players teleport near to the current player (instantly appear). 2) Other players run fast towards the MPD region during the sleep period (Estimated average movement speed in Equation (1) is too low). 3) The boundary area in PVS test is too small and the other player is just near to that area and moving towards the current player (fixed sleep time in micro-scan is too long).

The error rate and quality of the game are computed as,

$$ErrorRate = \frac{Total\ sleep\ time\ with\ errors}{Total\ sleep\ time} \quad (2)$$

$$Accuracy = 1 - ErrorRate \quad (3)$$

Open and Closed map. We define Open Map as the map which consists of only a few loosely distributed buildings and most of the areas in map are flat planes. (Eg. - Simpson's map of Quake III). Closed Map is defined as the map that is mainly constructed by architectural structures, such as the interiors of a castle. It consists of many rooms separated by door-ways. (Eg. - Q3DM7).

5.2 Small Scale User Study - Methodology

We did a small scale user study with 21 non-author students from our school to study the effects of the power saving algorithm on human players' game play experience. There were 16 male and 5 female students in the age group 22-30.

The study procedure was as follows: First, the participants filled up basic demographic information in the survey form. Then, they played the original Quake III game for 10 minutes to get some training. After that, they played six randomly chosen games from eight versions (including the original and our modified versions with ET values 1%, 2%, 3%, 4%, 5%, 6%, 7%). The users compared the game plays and gave their observation on the quality of the game play by filling up the questions in the survey form. All the evaluations were done with Quake III game in a networked environment with up to 10 concurrent players.

5.3 Experiments

For each of our experiments, we ran the game client and played for around five minutes to capture the data. We varied the maps, number of players (2, 4, 8, 16...) and networks (WiFi, 3G-WCDMA) in each of these experiments. We used up to 10 concurrent human players in each run. To ensure high interactivity and interesting game play we added a few moderately intelligent bots with the real human players. We used three different maps developed by the Quake III Arena team (*q3dm1, q3dm7, Simpsons*). *q3dm1 and q3dm7* are highly occluded closed maps and *Simpsons* is a open map with fewer occlusions.

Though we varied the number of clients in each run, to have a good mix, only some of them ran in power saving mode while the rest ran in normal mode. Such a combination is realistic and it enabled us to study the effect of power saving clients on normal clients. We measured the WiFi interface power consumption of the mobile clients that were connected to the DAQ.

6. EVALUATION RESULTS

We measured the power (WiFi interface) and error convergence rate for various map types, player density and error thresholds in Quake III game.

6.1 Effects of Map Type

Our first experiment was to measure the effects of map type on amount of energy saved. We set all other variables to fixed values except the map type. The settings for the experiment are shown in the Table 1.

Table 1: Experiment Variables Setup for Section 6.1

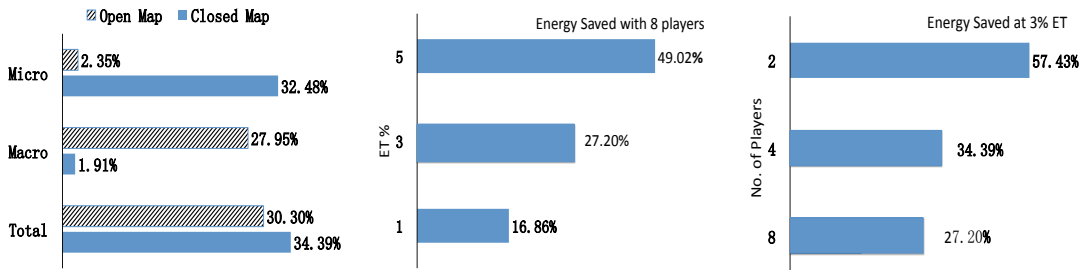
	Open Map	Closed Map
Map:	Simpsons	Q3DM7
Player no.:	4	4
Error Threshold:	3%	3%
MPD:	2500 game units	2500 game units
Fixed Sleep Time:	200	200
Movement Speed:	500	500
Game Length:	15 minutes	15 minutes

The graph in Figure 3a shows the test results. The values represent the total energy (in percentage) saved per client. For closed maps we saved 34.39% energy with more than 97% accuracy. It is much higher than 20% energy saving (with same level of accuracy and player density) reported in our previous work for closed maps [19].

The contributions of Macro Scan and Micro Scan were logged at runtime. Overall, in closed maps we saved 4% more energy than in open maps. This shows our algorithm can function well on both open maps and closed maps. Most of the energy saved with the open map came from macro scan while most of the energy saved with the closed map came from micro scan. This is expected, because the PVS approach doesn't gain much on an open map as there are very few geometric occlusions. Visible distance between players in an open map is more likely to be greater than the MPD, thus macro scan can save more energy. But in a closed map, the players are within the MPD most of time, thus, it is not possible for macro scan to return a valid sleep time. This experiment shows the importance of using both macro scan and micro scan to adapt to different map types and to save the maximum possible energy.

6.2 Effects of Energy Threshold

From the previous experiments we can see, the amount of energy saved with different map types differs only with respect to the contribution of macro scan and micro scan. Since they are equivalently important, in the following measurements we are concerned only about the total amount of energy saved. We chose to



a. MapType vs. Energy Saved **b. Error Threshold vs. Player Density** **c. Player Density vs. Energy Saved**

Figure 3: Effect of map type, error threshold and player density on energy saving

use the closed map for the remaining experiments thus eliminating the macro scan. With other parameters unchanged, we had two variables, namely number of players and Error Threshold (ET) to consider.

Firstly we measured the effects of different error thresholds on the amount of energy saved. We fixed the number of players to 8, and ran the experiments with ET of 1%, 3% and 5%. The result in Figure 3b shows the percentage of energy saved for each value of ET. The general trend is that for higher ET, more energy is saved. The rationale behind the increase is that the higher ET results in higher Fixed Sleep Time. From ET 1% to 3%, there is a 10.34% increase in the amount of energy saved. From ET 3% to 5%, there is a 21.82% increase. The non-linearity of increase is explained by unpredictable fluctuations in players' interactions.

6.3 Effects of Player Density

To measure the effects of the number of players (player density) on the amount of energy saved, we fixed ET to 3%, and changed the number of players to 2, 4 and 8. Note, for FPS games with highly occluded maps (eg. Quake III), the average number of concurrent players is around 6, though the game can accept up to a maximum of 16 players for certain maps [10].

As shown in Figure 3c, fewer players resulted in more energy savings. For a 2-player game, the current player is in fact alone for most of the time. Intuitively, the client should sleep up to 80% of time given that the map Q3DM7.pk3 is quite big. Due to the overhead imposed by the wireless interface card, only 57.43% of energy is saved. As the card is switched on/off often, the heat in the hardware increases, resulting in slow response. It takes about 50-200 milliseconds to wake-up from sleep mode. This is higher than the normal 10-50 milliseconds lag. During these mode transition (sleep-to-wakeup mode) period it is not possible to save energy.

We used *off-the-shelf WiFi hardware* in our experiments. With better WiFi cards (with 10-50 milliseconds ON/OFF lag) , we have estimated that it is possible to achieve close to 80% power saving for 2 players and up to 50% for 8 players.

6.4 Optimisation of Fixed Sleep Time

The Error Controller is supposed to continuously adjust the Fixed Sleep Time (FST) and Movement Speed (MS) to optimal values to save maximum energy while meeting the quality requirement (Error Rate). The graph in Figure 4a shows how the FST is adjusted to optimal values over time when playing with 8 players and 5% ET. (This data is from the same game play used to plot Figure 3c for 8 players). Again , from the same game play data, we have also plotted the graph in Figure 4b which shows the corresponding Average Error Rate (AER) over the time.

From the Figure 4a we can see, the FST increases linearly for the first 20 samples (at every tenth frame we took a sample). Then it jumps down from about 550ms to about 300ms. This is because the AER at this instant is higher than the threshold 5%; the FST needs to be shortened in order to keep the AER within the ET. We shorten FST to 75% of its current value. The AIMD method allows AER to reduce quickly. The flat portion of the graph means the AER is close to the ET. At this moment, FST stays relatively stable. Whenever another error occurs, the FST is reduced again until the AER is lower than the ET. The graph in Figure 4b shows the effect of this adjustment. The AER starts from 0. It then increases until slightly higher than 5%, where FST is forced to be multiplicatively reduced. The AER is well controlled within the 5% threshold which proves our AIMD method is effective.

The graphs in Figure 4c and Figure 4d show the convergence of AER to ET level for game plays with 3% and 1% ET respectively. The average convergence time is less than 6 seconds. This can be shortened further by optimising the sample size and using weighted average for AER.

6.5 Energy Overheads

There was a linear increase in processor load and memory requirement with number of clients in power save mode. However, with up to 8 players in power save mode, there was less than 3% increase in processor load and memory. It resulted in 4% additional energy consumption. This additional energy consumption is negligible as the processor and memory consume only 4-15% of overall system energy whereas wireless interface in mobile devices consume up to 40% of system energy. We obtained such efficiency by relying on the existing PVS data rather than recomputing.

6.6 Small Scale User Study

After training, the users played a set of six randomly chosen games from eight versions as described in Section 5.2. They were not informed about the meaning of each version. For each game play, we asked them to rate how noticeable, if at all, were any network related artefacts in the game, compared to the unmodified version, on a 5-point Likert scale. Figure 5 shows the user study results. The results show that the artefacts are almost unnoticeable up to ET=3% and ET=3% is good enough to save significant amount of energy as shown in Figure 3c.

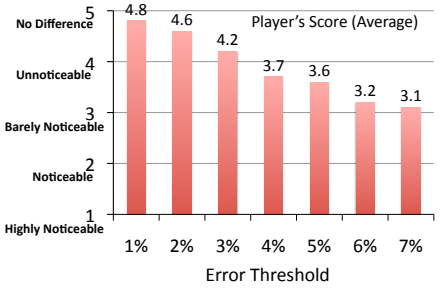
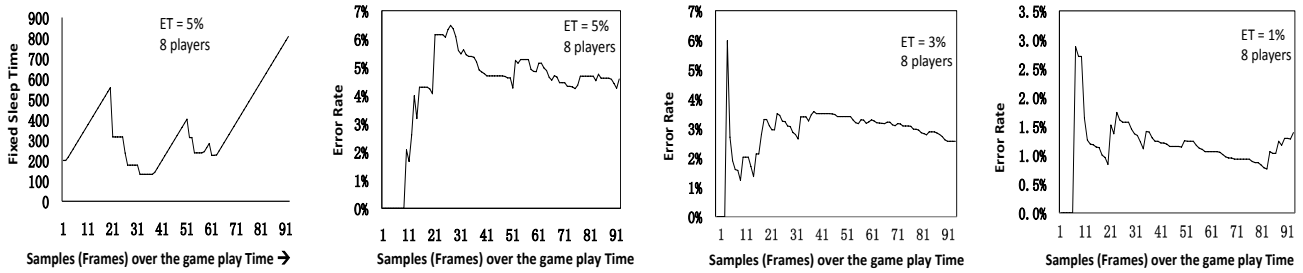


Figure 5: User Study - ET Varies (1% - 7%)



FST over Game Play Time | **Error Threshold 5% vs. Energy Saved** | **Error Threshold 3% vs. Energy Saved** | **Error Threshold 1% vs. Energy Saved**

Figure 4: Effect of feedback based control on FST and Error Rate

Objective Study. Offline analysis of game play data revealed that there was no significant difference in performance (number of opponents killed/fragged) of the users between original and power efficient versions. It varied only by 1.8%. Experts always killed more opponents than normal users in both normal and power efficient versions.

7. CONCLUSION AND FUTURE WORK

We have presented a set of techniques which used novel and scalable algorithms to save wireless client's energy. The techniques can also be applied to reduce game server's bandwidth requirement. We can save up to 57% of Wireless Interface Energy (which is, about 22.8% overall system energy) per mobile client without affecting the quality of the game play adversely using off-the-shelf network hardware. Our future work will include, extensive studies on different feedback based controlling mechanisms, large scale user studies and integration into commercial game engines.

8. REFERENCES

[1] Anand, B., Ananda, A. L., Chan, M. C., Long, L. T., and Balan, R. K. Game action based power management for multiplayer online games. *Proceedings of the 1st ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld)*, Barcelona, Spain, Aug. 2009.

[2] Anand, B., Thirugnanam, K., Long, L. T., Pham, D. D., Ananda, A. L., Balan, R. K., and Chan, M. C. Arivu: Power-aware middleware for multiplayer mobile games. *Proceedings of the Ninth IEEE NetGames*, Teipei, Taiwan, Nov. 2010.

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

[4] Anastasi, G., Passarella, A., Conti, M., Gregori, E., and Pelusi, L. A power-aware multimedia streaming protocol for mobile users. *Proceedings of the International Conference on Pervasive Services*, Santorini, Greece, July 2005.

[5] Armitage, G. An experimental estimation of latency sensitivity in multiplayer quake 3. *Networks, 2003. ICON2003. The 11th IEEE International Conference on*, pages 137 – 141, sept.-1 oct. 2003.

[6] Bertozzi, D., Benini, L., and Ricco, B. Power aware network interface management for streaming multimedia. *Proceedings of the IEEE Wireless Communications and Networking Conference*, 2002.

[7] Bharambe, A., Douceur, J., Lorch, J. R., Moscibroda, T., Pang, J., Seshan, S., and Zhuang, X. Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. *Proceedings of ACM Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, Seattle, WA, USA, Aug. 2008.

[8] Bhojan, A., Akhihebbal, A., Chan, M., and Balan, R. Arivu: Making networked mobile games green. *Mobile Networks and Applications*, pages 1–8, may 2011. 10.1007/s11036-011-0312-8.

[9] Bittner, J., Mattausch, O., Wonka, P., Havran, V., and Wimmer, M. Adaptive global visibility sampling. *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 94:1–94:10, New York, NY, USA, 2009. ACM.

[10] Boulanger, J.-S., Kienzle, J., and Verbrugge, C. Comparing interest management algorithms for massively multiplayer games. *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 6, New York, NY, USA, 2006. ACM.

[11] Google Open Source Project. *A port of Quake3 to Android.* `http://code.google.com/p/kwaak3/`, 2010.

[12] Gu, Y. and Chakraborty, S. A Hybrid DVS Scheme for Interactive 3D Games. *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2008.

[13] Gu, Y. and Chakraborty, S. Power Management of Interactive 3D Games Using Frame Structures. *VLSI Design*, 2008.

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

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

[16] MadWifi Organisation. *MAdWiFi Wireless Driver for Linux, Release v0.9.4*, Feb. 2008. `http://madwifi-project.org/`.

[17] Meyer, M., Sachs, J., and Holzke, M. Performance evaluation of a tcp proxy in wcdma networks. *Proceedings of the Eighth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, 2002.

[18] Teller, S. J. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, Technical Report UCB/CSD-92-708, EECS Department, University of California, Berkeley, Oct 1992.

[19] Thirugnanam, K., Anand, B., Sebastian, J., Kannan, P. G., Ananda, A. L., Balan, R. K., and Chan, M. C. Dynamic lookahead mechanism for conserving power in multi-player mobile games. *Proceedings of the 31st IEEE INFOCOM 2012*, Orlando, USA, Mar. 2012.

[20] Ubuntu. *Atheros Hardware Access Layer*. `http://manpages.ubuntu.com/manpages/maverick/en/man4/ath_hal.4freebsd.html/`.

[21] Waveren, J. P. V. The quakeiii arena bot. *http://www.kbs.twi.tudelft.nl/docs/MSc/2001/Waveren_Jean-Paul_van/thesis.pdf*. University of Technology Delft, Faculty of ITS, 2001.

[22] Wei, Y., Chandra, S., and Bhandarkar, S. A statistical prediction-based scheme for energy-aware multimedia data streaming. *Proceedings of the Wireless Communications and Networking Conference (WCNC)*, Atlanta, GA, Mar. 2004.

[23] Yang, S.-R. Dynamic power saving mechanism for 3g umts system. *ACM Mobile Networks and Applications*, 12(1):5–14, 2007.