Mobile Netw Appl
DOI 10.1007/s11036-011-0312-8

ARIVU: Making Networked Mobile Games Green

A Scalable Power-Aware Middleware

Anand Bhojan · Ananda L. Akhihebbal · Mun Choon Chan · Rajesh Krishna Balan

Abstract With the improved processing power, graphic quality and high-speed wireless connection in recent generations of mobile phones, it looks more attractive than ever to introduce networked games on these devices. However, these games consume higher levels of energy. While device features and application resource requirements are rapidly growing, the battery technologies are not growing at the same pace. Therefore, the main concern is the limitation of the battery power of such portable devices to support potentially long-hour of game play. In this paper we present ARIVU, a scalable power aware middleware that dynamically controls the energy consumption of wireless interface based on the game and system state while maintaining the user experience. The middleware is able to save up to 60% of the total energy consumed by the 802.11 g wireless interfaces for First Person Shooting (FPS) games and up to 35% of the total energy for Massively Multiplayer Online Role Playing Games (MMORPG).

Keywords energy middleware · power middleware · game engine · mobile game · wireless game · server scalability · area of interest · area of visibility · green middleware

1 Introduction

In this paper, we present an extended version of a preliminary scheme that appeared in ACM/IEEE Netgames 2010 conference [2]. ARIVU is an adaptive middleware that uses set of novel techniques and algorithms to conserve energy consumed by wireless interface, display and processor of mobile devices without affecting the quality of game play. For mobile game clients, the resources that can be managed to save power include the network, processor, and display. Our measurements in modern Google Android Phones such as, HTC Hero and HTC Magic show 45%, 40% and 15% of phone energy consumption by LCD, Wireless Interface (WiFi or 3G) and CPU respectively, at their peak levels. In this paper, we will focus on reducing the energy consumption of the wireless network interfaces. Display and processor energy conservation can also be performed based on game state using the same middleware framework, but we defer that extension to future work. Most of the recent works on saving wireless interface power [4, 11] try to put the mobile client's wireless interface in sleep mode when there is

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.

A. Bhojan (✉) · A. L. Akhihebbal · M. C. Chan
School of Computing, National University of Singapore, Singapore, Singapore
e-mail: banand@comp.nus.edu.sg

A. L. Akhihebbal
e-mail: ananda@comp.nus.edu.sg

M. C. Chan
e-mail: chanmc@comp.nus.edu.sg

R. K. Balan
School of Information Systems, Singapore Management University, Singapore, Singapore
e-mail: rajesh@smu.edu.sg

enough data in the client's buffer to process. Access points or servers buffer the packets addressed to the mobile client until the client wakes-up. For latency sensitive applications such as real-time media streaming and networked games these schemes may even tend to increase overall power usage [3]. Hence we pose the following question:

How do we minimise the energy requirements of latency sensitive applications, especially Multiplayer Mobile Games without affecting the user experience adversely?.

In this work we present a novel approach which exploits the game state and user type to optimise power consumption as well as the bandwidth.To test our middleware, we used the Quake III [8] first person shooting (FPS) game and the Ryzom [9] massively multiplayer online role playing game (MMORPG). We chose these two game as (1) the code for both game have been open sourced, (2) both are popular games from popular game genres [10], (3) FPS and MMORPG games are the hardest to save power, as they involve large amounts of real-time player interaction (players are constantly trying to shoot or interact with each other) and (4) there exist an Android port for Quake III (called kwaak3). Our results show that we are able to save up to 60% of wireless interface power with no perceptible quality loss. Overall, the main contributions of this paper are as follows: (1) We present a set of general and systematic approaches to conserve power consumed by Wireless Interface while playing multiplayer mobile games. (2) We analyse and identify the key parameters that determine the trade-off between game quality and power saved. (3) Our approach has been realized on commercial games. We show that power savings can be achieved even in games with very fast, constant interaction and high frame rate. (4) We provide a simple API for integrating existing games with our middleware and building new power aware green games.

2 Design of the ARIVU middleware

The component layout of the ARIVU middleware is shown in Fig. 1a. ARIVU's goal is to reduce the game client's energy consumption by adaptively varying resource consumption while maintaining the user's game play experience—we do not do server power management as the server is usually connected to a power source. To collect necessary information about the game and make power saving decisions, we created two specific sub-components of ARIVU. The Resource Data Collector (RDC) collects the raw data, discretizes the game map into cells and stores distance and visibility information from cell to cell. Data from RDC are used by the Resource Controller (RC) for power management. We now explain each sub-component in detail.

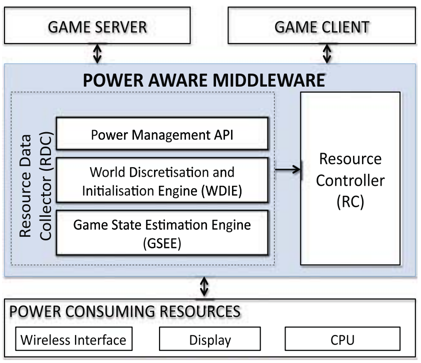
2.1 Resource Data Collector (RDC)

The major part of the RDC is a generic API through which the RDC collects the necessary information about the game and client resources. The RDC collects two types of data: (1) Client's game state and (2) Client's resource state (client hardware type, network state, battery state).

2.1.1 API for game state

The API is the central part of the RDC. The API is simple enough for easy integration into existing and new games. It has only five primary functions for MMORPG and FPS games which are described in Fig. 1b. The initialisation function (*init*), which is overloaded and it differs for each game genre. The next function (*updateEntityPosition*) collects information about each 'interactive entity' in the game. An interactive entity (or shortly 'entity' in general) is an

Fig. 1 Architecture and API


GAME SERVER
GAME CLIENT
POWER AWARE MIDDLEWARE
Resource Data Collector (RDC)
Power Management API
World Discretisation and Initialisation Engine (WDIE)
Game State Estimation Engine (GSEE)
Resource Controller (RC)
POWER CONSUMING RESOURCES
Wireless Interface
Display
CPU

a. Architectural Components

Function	Description	Frequency / Game genre
void init(uint32 worldMapSizeX, uint32 worldMapSizeY, uint32 visionDistance, unit32 maxMoveSpeed)	Initialize the world map, divide the world map into grid, save the vision distance and move speed.	Once per map/ RPG
void init(areaList areas, portalList portals, graph adjecenyGraphofClustersPortals, bitArray PVS, unit32 maxMoveSpeed)	Initialize the world map using the in-game discretization of the game world and the visibility from one area to other areas, save move speed.	Once per map/ FPS
void updateEntityPosition (CWorldEntity entity)*	Update the position, orientation and environment of an entity.	every frame/ RPG and FPS
Void visionDistance(uint32 visionDistance)	Save new vision distance. (triggered when the player moves to new Environment).	When there is a change/ RPG and FPS
Void playerHostileEnvironment (Boolean)	Players current environment hostile or free friendly environment	When there is a change/ RPG and FPS

b. API

entity in a game which can interact with a player such as, another player (or game client), a Non-Player Character (NPC) or Artificial Intelligence (AI) character controlled by the game server, etc. Modern MMORPG games define 'vision range' (also known as, 'vision distance', 'visibility radius' or 'Area of Interest-AoI'). It is a range up to which an entity can actively view with higher level of detail and interact. Our middleware makes use of this information. In FPS games the game world is basically made up of set of convex hulls (called areas). Group of adjacent convex hulls make a cluster and clusters are connected by portals. Portals themselves are areas which provide path or connection from one cluster to another. Set of visible areas from current area is called Potentially Visible Set (PVS) of areas. PVS is also known as Area of Visibility (AoV). Games without visionRange, PVS and maxMoveSpeed information can set some of these values to '0' and it will be computed by World Discretisation and Initialisation Engine (WDIE) described in the following section.

2.1.2 World Discretisation and Initialisation Engine (WDIE)

For MMORPG games and FPS games without area based discretisation data, WDIE discretises the world map into hexagonal tiles (Fig. 2a) and pre-computes visibility and distance between the tiles and stores them in two matrices for use by Game State Estimation Engine (GSEE) and Resource Controller (RC) described in the following sections. AoI and AoV are computed at the game server and is based on hexagonal tile distance, area or the tile visibility algorithm [6] and the player's environment. In ARIVU, *vision radius (AoI radius)* is bounded to the player's environment. Vision radius is small in safe friendly zone (no monsters or enemies) and high in hostile environments. The actual radius values are game dependent. In Fig. 2a, the visibility radius of an entity *a* could be *r*1 or *r*2 depending on its current environment. *PVS (or AoV)*- A tile (or area) is considered visible from another tile if there exists a point in each of the two tiles that can be connected by a line segment that does not intersect an obstacle. A set of visible tiles from current tile forms PVS of current tile. PVS is symmetric. That is, *PVS(tile x) includes tile y* ⇔ *PVS(tile y) includes tile x*.

2.1.3 Game State Estimation Engine (GSEE)

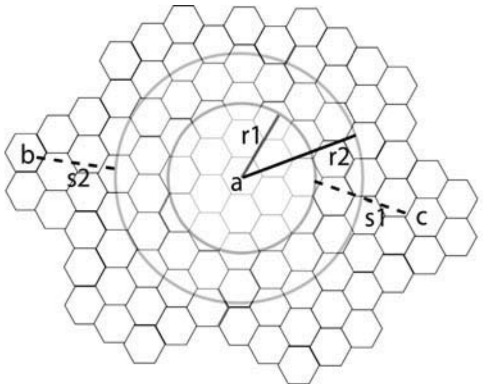
Based on the run-time position update data (through function *updateEntityPosition*) for each entity (player or NPC) at the server side, the GSEE dynamically subscribes the entity to the tile or area in which it is currently residing. We refer the client currently under consideration for saving power as the 'current client' (or current player). If there is no other interactive entities inside the current client's vision radius, the state is considered non-critical for the current client. We call this as *game state* in the rest of the paper.

2.1.4 Resource state—network state & device state

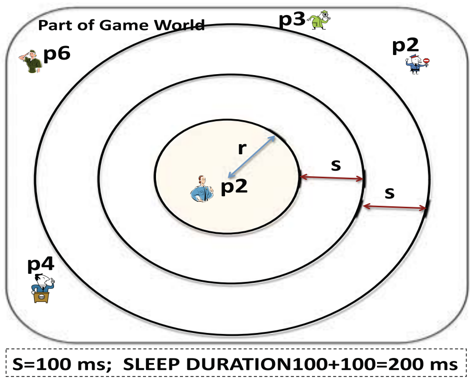
The network state comprises of the round trip time (RTT), available bandwidth, and packet drop rate between the client and the server. In our current implementation, we only consider the RTT using ping probes. Current implementation of ARIVU uses only current battery state, calculated from the remaining battery life, to decide on an appropriate aggressiveness (trading off game-play quality for increased power savings) setting.

2.2 Resource controller (RC)

The resource controller (RC) follows client/server architecture. A key consideration for the RC is determin-



a. Tile Distance/Visibility



b. Dual Ring or Incremental Look ahead

Fig. 2 World discretisation and macro algorithms

ing how long to put the wireless network interface in sleep mode. First the RC checks the current client's game state (GSEE). If game state is not critical, the RC enters Macro Level (has larger sleep duration) power management for the current client; Otherwise, it enters Micro Level (sleep duration is limited) power management. These are explained below.

2.2.1 Macro power management

At the macro level, the server side RC makes power management decision and sends *sleep and aggressive_level* commands to the clients. It's decision relies on the position of all entities and resource states of all clients provided by the RDC. Depending on the game genre and number of entities in the game, it can use either single-ring (suitable for MMORPGs) or dual-ring algorithms (suitable for FPS games) given below.

Single Ring Algorithm (SRA) Single ring algorithm is based on the *relative velocity* between the interactive entities. In Fig. 2a, it takes $s1$ time for client c to reach client a's AoI if $AoI\ radius = r1$. Similarly, it takes $s1$ time for client b to reach client a's AoI if $AoI\ radius = r2$. Here, $s1$ and $s2$ are time-distance values (that is, time to travel a distance) computed based on relative velocity between the clients. Single ring algorithm estimates the *Potential_Sleep_Duration* (PSD) of a client as given below.

The algorithm finds the nearest *n* interactive entities around the current client and estimates the time required for them to reach the current client's AoI range. The smallest of these reach-time values is set as the PSD. The PSD is the safest duration and there is very less chance for important game state changes during this period. For a game with *m* interactive entities, the algorithm takes $O(m^2)$ time to find the nearest n entities. To make it scalable for Massively Multiplayer Online Games (MMOGs), RC uses *interaction recency* as the key strategy for games in which some kind of grouping or clustering semantics is employed. RC maintains list of recently interacted entities for each client in *Most Recent Interaction Table (MRIT)* of size $m \times p$, where m is number of clients and p is number of interactive entities a client is interested in. Each table row i maintains identification of p most recently interacted players with client i. For each client RC computes and compares the distance with only p other clients or entities. Here the tradeoff is, as p grows the accuracy increases at the cost of computation. Games without grouping semantics may have completely random order of interaction between the entities and MRIT approach

Algorithm *Single_Ring_Algorithm* **(SRA))**

```
for each entity i do
    //get current proximity of all interactive entities
    $currentProximity_i = getEuclideanDistance$
     $(currentClient, entity_i)$;

    // adds current value to history and removes oldest
    value $pastProximity_i.add(currntProximity_i)$;

//get n nearest entities; we are interested only on n nearest
entities
$interestingEntities_{1...n} = getNearest(n\ entities)$;

for each interestingEntity j do
    // compare the historical proximity to determine new
       relative
    // velocity (bi_directional). entities coming closer or
       going away?
    $relativeVelocity_j = calculateRelativeVelocity$
       $(pastProximity)$;

    //calculate potential sleep time. That is, s1 or s2 in
       Fig. 2a
    $PSD = (currentProximity_j - AoI\_Visibility\_$
       $Radius)/relativeVelocity_j$;

    //If $PSD <= 0$, return 0 and exit SRA. Entities
       found inside the ring
    $if(PSD <= 0) return(0)$;

//return the PSD of the entity which is expected to reach
the client's AoI Visibility Radius first
return $smallest(PSD_{1..n})$
```

becomes ineffective. For such games, Dual Ring algorithm is used.

Dual Ring Algorithm (DRA) Dual ring algorithm is based on *incremental lookahead* mechanism. It is highly scalable than SRA. ARIVU enters this part when there is no entities/players inside the vision range of the current player. The DRA algorithm looks beyond the vision range by gradually increasing the search area. It checks for other entities in the area from vision range of the current player to distance s as shown in Fig. 2b. Where, s is 100 ms time step from the vision range of the current player (p1) and it is computed as, $s = visionRange + (estimatedAveragePlayerVelocity \times 100\ \text{ms})$

If there is no other entity in the range s, then the algorithm increases s by another 100 ms time step and checks again. This is repeated until an entity is found or max sleep threshold (game dependent parameter [1]) is reached. PSD is set to either the value of s prior to finding the first entity or max sleep threshold. We have selected 100 ms time-step as the smallest possible value for s as sleep duration below 100 ms are too small to save any significant energy [1].

2.2.2 Micro power management

If game state (AoI state) is critical, the RC enters Micro Level (sleep duration is limited) power management. At micro level the RC relies on either visibility from area-to-area or orientation of the players. Area-to-area visibility is suitable for games with maps having several walls and obstacles (eg. common maps of FPS games). Player orientation is suitable for game with maps having huge open areas (eg. common maps of MMORPG games).

Player orientation based approach In most games the orientation of a player is defined by a float value (range from $-\Pi$ to Π) and the field of view of the player is $2\Pi/3$. The entities which are inside the vision range of a current player still cannot be seen if they are not in the field of view of the current player as shown in Fig. 3a for players p and entity e. Micro power management exploits this property to improve the efficiency of our middleware. We compute the duration required for the current player to reach other entities' field of view as given below.

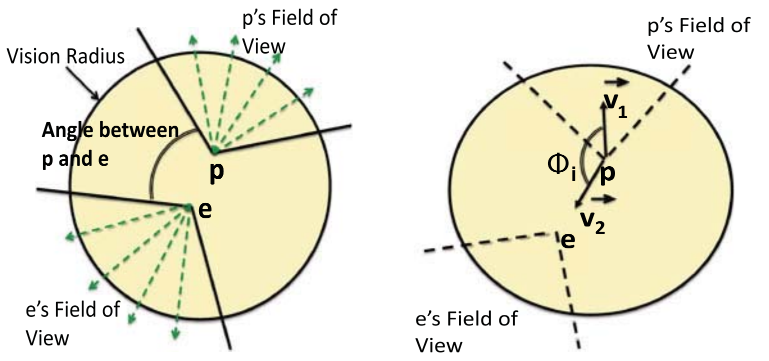
1. As shown in Fig. 3b, v_1 is the direction vector of the current player and v_2 is the direction vector from current player p to entity e. We calculate the angle $\phi 1$ between v_1 and v_2 . From which we compute, $(\phi 1 - \Pi/3)$ which is the angle the current player need to turn to see this entity.
2. Similarly we calculate the angle $\phi 2$, which the entity need to turn to see the current player. We calculate $\phi 2$ only if the entity has vision. The $min(\phi 1, \phi 2)$ will be the angle displacement between the character and the entity.
3. After calculating the angle between the current player and all other entities insides vision range or distance, we can find the minimum angle distance, $\phi_{\min}$.
4. Based on this value, we find the *Potential_Sleep_Duration* PSD as follows.

   $PSD = \phi_{\min}/\alpha$ where, α is the mean angle velocity of players.

Visibility based approach In visibility based approach we check the visibility from the current area of the player to all other areas. Set of visible areas from current area is called Potentially Visible Set (PVS) of areas. It is not from the current position of the player but from the current area (any position or point in current area) of the player and hence, we call it PVS. If there is no other entities in PVS of current player, ARIVU assigns a fixed time (100 ms) to PSD. Otherwise, PSD is set to 0.

2.2.3 Resource controller—client side

The server side RC sends two types of messages to the RC client—A *sleep* message with PSD and a *resource state* message with AGGRESSIVE_LEVEL flag (high, medium or low). On receiving sleep message the client side RC computes the Effective Sleep Duration (ESD) as given below and puts the wireless interface into sleep mode. As described in our previous work [1], there are two constraints for sleep duration: *maximum sleep duration* that a game can tolerate (using techniques such as Dead Reckoning) and *minimum sleep duration* that is really required to save energy (due to *mode switch*



a. Field of View b. Angle distance between players

Fig. 3 Micro algorithms depicted

power cost and *mode switch latency* of the wireless interface).

```
//Computing Effective Sleep Duration (ESD)
if (PSD < minSleepDuration)
    ESD = 0;
elseif (PSD > maxSleepDuration)
    ESD = maxSleepDuration;
else
    ESD = PSD;
```

In both macro and micro power management cases the client ensures that it receives at least one complete update before the next sleep to avoid longer inconsistencies.

3 Results

3.1 Testing methodology

Figure 4 shows the testbed used for our experiments. To obtain accurate power measurements, we used a custom made cell phone test board, a high-speed multifunction Data Acquisition Equipment (DAQ) USB-6251 and a Signal Conditioning Equipment (SC-2345) from National Instruments (NI). For more details on measurement methods the readers may refer to our previous work [1]. We have assumed, availability of more than 80% battery level in the mobile device (hence, we just used the default algorithm, without increasing the AGGRESSIVE level) and good network conditions in all our experiments. Our test scenarios for wireless interface power saving were designed to show the following:

1. **Base Effectiveness**: What is the potential power savings achievable by ARIVU? What is the effect of player density on power saving? We present these results in Section 3.2.
2. **Effect of Game Type**: How effective is ARIVU in saving power for different game types? We show the results for this experiment in Section 3.3.

The main quality measure we used is the number of important packets that were either dropped or missed their deadlines as a result of the power saving technique.

3.2 Base effectiveness

The results for Ryzom game (MMORPG game) are depicted in Fig. 5a and b. As this is a MMORPG game, ARIVU selects Dual Ring Algorithm for macro mode power saving and Player Orientation Approach for micro mode power saving. The amount of power saved ranges from 35 to 20% for 802.11g wireless interface. ARIVU guarantees the quality of game play with less than 2% loss of important packets. An additional information shown in this graphs is, the effect of *estimated average player velocity (EAPV)* of dual ring algorithm. If EAPV is lower than actual velocity, then there is no possibility for errors. Figure 5c shows the contribution of macro and micro methods in the amount of power saved.

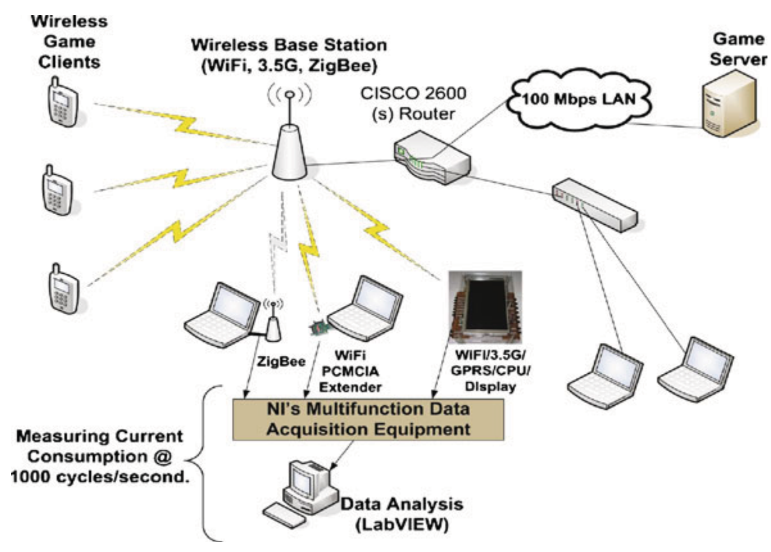
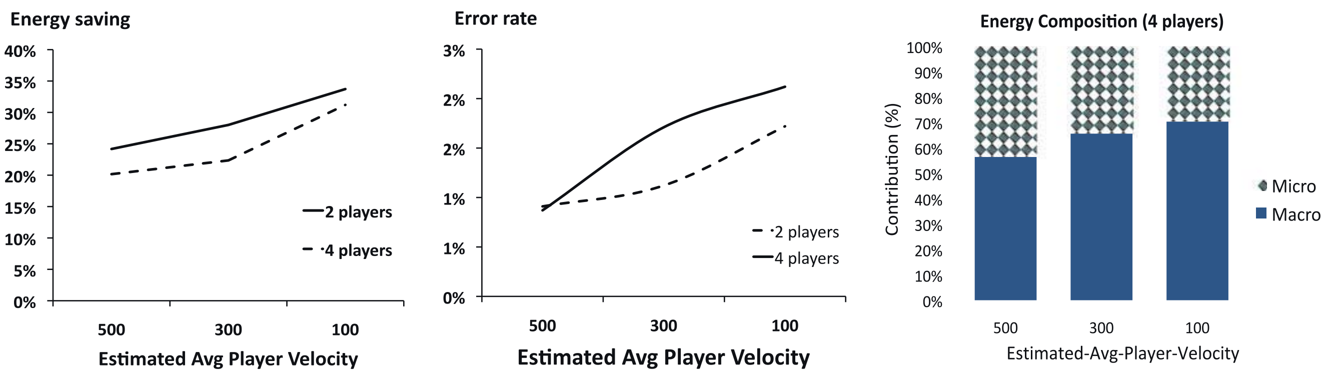

Wireless Game Clients
Wireless Base Station (WiFi, 3.5G, ZigBee)
CISCO 2600 (s) Router
100 Mbps LAN
Game Server
ZigBee
WiFi PCMCIA Extender
WIFI/3.5G/ GPRS/CPU/ Display
NI's Multifunction Data Acquisition Equipment
Measuring Current Consumption @ 1000 cycles/second.
Data Analysis (LabVIEW)

Fig. 4 Testbed used for experiments

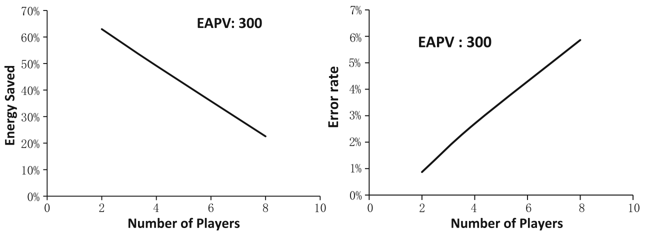


a. Power Saved (RPG) b. Drop Rate (RPG) c. Macro vs. Micro

Fig. 5 Results in MMORPG

3.3 Game type effects

The above results are based on slow speed MMORPG games with outdoor open area maps. We evaluate the results for Quake III game which is a high speed shooting game with indoor maps. As it is FPS game and usually has up to maximum 16 players, ARIVU selects Single Ring Algorithm for macro mode power saving as scalability is not required here and Visibility based Approach for micro mode power saving as the map has many walls and obstacles. The amount of power saved ranges from 60 to 20% for wireless interface for this game with maximum error level of 6%. The results are depicted in Fig. 6a and b. We set EAPV to 300 game units which is average player velocity in Quake III. ARIVU saves more power with FPS games than MMORPG games due to the following facts: The FPS game map is a indoor map and visibility approach is used for micro power management. Though the players are nearby, they are separated by walls and obstacles hence they cannot interact with each other which results in more opportunities for saving power. Also, it is important to observe that the graphs depict the performance of ARIVU is highly sensitive to number of players in FPS games.



a. Power Saved (FPS) b. Drop Rate (FPS)

Fig. 6 Results in FPS games

4 Related work

There is a rich body of related work in the area of power conservation and interest management.

Power conservation Mobile devices today comes with various power management features for its processor, LCD display and wireless interface. Previous works present different techniques for wireless interface power management such as, better link utilization and throughput [7], using proxies to allow clients to sleep [4], and looking for statistical correlations that allow power savings [1, 12]. In our work, we apply the lesson and techniques derived from these past works to the context of latency-critical interactive games.

Interest management There are various algorithms for determining AoI [5, 6]. They are either distance based or visibility based technique for improving scalability. We use a combination of both techniques by taking their good features.

5 Future work and conclusion

The results on our efforts in building a scalable integrated middleware to manage overall system power consumption while playing FPS and MMORPG games is presented in this paper. Through macro and micro power management modules ARIVU tries to capture both longer possible sleep duration and shorter ones for efficient power management of wireless interface. We have demonstrated that ARIVU can save highly significant amount of energy for both smart phone and laptop games: 60% for wireless interface. Saving 60% wireless interface energy translates to saving 24% of overall system energy in HTC Magic and HTC Hero smartphones. ARIVU uses the same game state information to manage CPU and LCD/OLED Display power. We are continuously enhancing our middleware to include more game genres to make it highly generic.

References

1. Anand B, Ananda AL, Chan MC, Long LT, Balan RK (2009) Game action based power management for multiplayer online games. In: Proceedings of the 1st ACM SIGCOMM workshop on networking, systems, and applications on Mobile Handhelds (MobiHeld). Barcelona, Spain
2. Anand B, Thirugnanam K, Long LT, Pham DD, Ananda AL, Balan RK, Chan MC (2010) Arivu: Power-aware middleware for multiplayer mobile games. In: Proceedings of the ninth IEEE NetGames. Teipei, Taiwan
3. Anand M, Nightingale EB, Flinn J (2003) Self-tuning wireless network power management. In: Proceedings of the 9th international conference on Mobile Computing and networking (Mobicom). San Diego, CA
4. Anastasi G, Passarella A, Conti M, Gregori E, Pelusi L (2005) A power-aware multimedia streaming protocol for mobile users. In: Proceedings of the international conference on pervasive services. Santorini, Greece
5. Bharambe A, Douceur J, Lorch JR, Moscibroda T, Pang J, Seshan S, Zhuang X (2008) Donnybrook: Enabling large-scale, high-speed, Peer-to-Peer games. In: Proceedings of ACM conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM). Seattle, WA
6. Boulanger JS, Kienzle J, Verbrugge C (2006) Comparing interest management algorithms for massively multiplayer games. In: NetGames ’06: proceedings of 5th ACM SIGCOMM workshop on network and system support for games. ACM, New York, p 6. doi:10.1145/1230040.1230069
7. Meyer M, Sachs J, Holzke M (2002) Performance evaluation of a tcp proxy in wcdma networks. In: Proceedings of the eighth annual ACM/IEEE international conference on Mobile Computing and networking (MOBICOM)
8. Quake III (2010) Quake 3 Arena Source Code. Id Software, (Version 3.21)
9. Ryzom (2010) RYZOM. Winch Gate Property Limited
10. Schonfeld E (2010) When it comes to iphone games, what sells is action, adventure, and arcade
11. Wei Y, Chandra S, Bhandarkar S (2004) A statistical prediction-based scheme for energy-aware multimedia data streaming. In: Proceedings of the Wireless Communications and Networking Conference (WCNC). Atlanta, GA
12. Yang SR (2007) Dynamic power saving mechanism for 3g umts system. ACM Mobile Netw Appl 12(1):5–14