ARIVU: Power-Aware Middleware for Multiplayer Mobile Games

Bhojan Anand[‡], Karthik Thirugnanaman[†], Le Thanh Long[‡], Duc-Dung Pham[‡],
Akhihebbal L. Ananda[‡], Rajesh Krishna Balan[†], and Mun Choon Chan[‡]
[‡]National University of Singapore and [†]Singapore Management University

***Abstract*—With the improved processing power, graphic quality and high-speed wireless connection in recent generations of mobile phone, it looks more attractive than ever to introduce networked games on these devices. While device features and application resource requirements are rapidly growing, the battery technologies are not growing at the same pace. Networked Mobile games are a class of application, which consume higher levels of energy, as they are naturally more computationally intensive and use hardware components including audio, display and network to their fullest capacities. Therefore, the main concern is the limitation of the battery power of such portable devices to support the potentially long-hour of game play.**

In this paper we present ARIVU, a power aware middleware that dynamically controls the energy consumption of wireless interface based on the game and system state while maintaining the user experience. ARIVU provides the relevant API for game developers to easily integrate the middleware. We measure power consumption of game play over different wireless interfaces including 3.5G (HSPA), 802.11g and ZigBee. The middleware is able to save up to 40% of the total energy consumed by the wireless interfaces (802.11g and ZigBee). In addition, we show the efficiency of ZigBee interface as potential low power interface for networked game applications.

I. Introduction

Wireless interface, display and CPU are the major power consuming components in modern smartphones. A wide range of recent research focuses on power aware protocols, middleware and techniques for mobile environments. Most of these works [3], [10] try to put the mobile client's wireless interface in sleep mode whenever possible. They buffer the packets addressed to the mobile client until the client wakes-up. These schemes will work well for latency tolerable applications such as file transfer, web surfing, email and stored media streaming. However, for latency sensitive applications such as real-time media streaming and networked games these schemes may even tend to increase overall power usage [2]. 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?

Self tuning middlewares suggested in previous works are based on the network access patterns of the applications and the application's intent to transfer data [2]. As games intent to transfer data continuously (20-40 pps), these schemes fail to save significant energy. In this work we present a novel approach which goes one level deeper and exploits the internal behavior of the application and user type to optimise power consumption as well as the bandwidth.

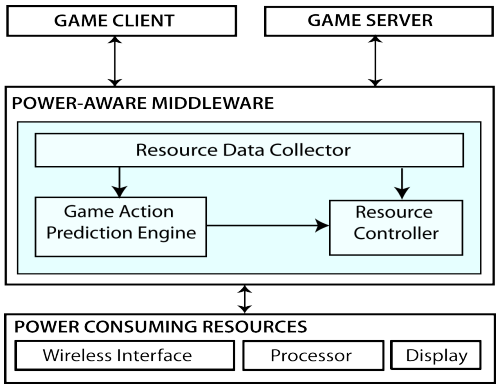

Fig. 1. Main Architectural Components of ARIVU

II. Design of the ARIVU Power-Aware Multiplayer Game Middleware

The component layout of the ARIVU middleware is shown in Figure 1. 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. For mobile game clients, the resources that can be adjusted to save power include the network, processor, and display. In this paper, we will focus on reducing the energy consumption of the wireless network interfaces. We believe that process and display power conservation can also be performed using the same middleware framework, but we defer that extension to future work.

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 do this without affecting the game play. Therefore, the basic decision made by ARIVU is: *When and for how long can the wireless interface be put into sleep mode without affecting game play?*

An obvious answer would be to sleep when there is minimum game state change and/or during unimportant, from the player's perspective, game events. However, in order to find these situations, ARIVU needs to gather sufficient information to estimate the current game state and then decide on the appropriate action. To do this, we created three specific subcomponents of ARIVU. The Resource Data Collector (RDC) collects the raw data that is used by the Game Action Prediction Engine (GAPE) to estimate the future game state. Data from RDC and GAPE are used by the Resource Controller (RC) for power management. We now explain each sub-

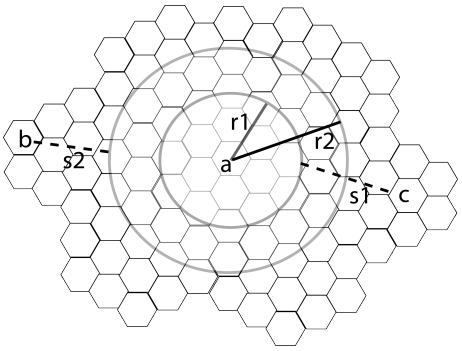

Fig. 2. AoI - Dynamic Hexagonal Tile Visibility

component in detail.

A. Resource Data Collector (RDC)

The RDC collects four types of data: 1) Client's area of interest, 2) Game action data, 3) Player activity level, and 4) Network state. Except the first item (client's area of interest), the rest are computed at the client side as they do not demand more memory and computational resources and do not adversely affect the power saving objective.

1) Area of Interest (AoI): AoI is a common technique used in massively multiplayer games to reduce the required bandwidth for each client. Each client receives update only on a part of the game world in which the client is currently engaged. In ARIVU AoI is computed at the game server and is based on hexagonal tile visibility algorithm [5] and the player's environment. Each player is dynamically subscribed to the tile in which he is currently residing. The visibility between tiles is precomputed for efficiency reasons. A tile 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 and does not cross the visibility radius. The visibility radius is dynamically adjusted based on the player's environment. For example, if the player is in safe zone (like a sanctuary area), there won't be any enemy object and, the player can interact only with the friendly objects. The interactions (such as, chatting, weapon exchange) with friendly objects happen mostly when they are in close proximity to the player. But, the interaction(such as, shooting) with enemies in hostile environments may happen even when their distance is longer. Visibility radius is set to a larger value when the player is in hostile environment when compared to friendly environment. The actual radius is game dependent. In Figure 2, the visibility radius of the client *'a'* could be r1 or r2 depending on its current environment. If there is no interactive object (other players) inside a client's visibility radius, the game state (AoI state) is considered non-critical for that client.

2) Game Action Data and Player Activity Level (PAL): The game action is the current action the player is engaging in such as walking, running, shooting, etc.. These actions closely correlate with the amount of game-state updates required to maintain an acceptable end-user game experience. Our experiments and analysis about the effect of the game actions on the game-state updates with Quake-mobile is described in our previous work [1].

As a FPS, Quake requires players to think and act quickly to win. However, even in this type of game, *non-critical activities* such as walking (27% of the time) and hiding (21% of the time) happen more often than important game changing activities (that cannot experience any quality loss; hereafter referred as *critical activities*) such as shooting (20%). The Quake experiment, thus suggests that substantial power saving, with minimal game-play impact, is achievable if we can accurately predict the current game state.

A FPS game can be considered a worst-case scenario for our approach as they usually have the lowest non-critical events count (due to the fast nature of the game). On the other hand, players in slower RPG games such as World of Warcraft, Lineage II, and Raganarok Online, have more non-critical actions which results in greater power saving potential.

ARIVU currently captures the following game events for RPG games (game events are captured only on the client): *Idle, Attacking, Moving, Accessing Menu, Dead, Chat (Text or Voice), Trading, Item Interaction (Interacting with items in the environment), Interacting with other avatars, Interaction with NPC (Non-playing character).*

RDC also collects the *Player Activity Level* in a separate routine. PAL is defined as the number of keyboard or mouse (or other UI devices) activities per second. Higher PAL rates indicate more critical actions.The effect of PAL on game state is described in more detail in our previous work [1].

3) Network State: The network state comprises of the 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. If the network state is good and there is a steady low-jitter stream of update packets between the client and server, it is possible to sleep (and save energy) without affecting the game-play quality as the client will still receive enough packets when it wakes up.

B. Game Action Prediction Engine (GAPE)

GAPE attempts to predict the game action for the next few frames using historical data and game environment provided by RDC. GAPE data is used by resource controller for micro power management as described in next section. We observed that the historical game action data and game environment tend to correlate quite well with the player's future game actions. We first filter the actions by environment where the player is currently in, then we take the past n actions of the player and do linear prediction of the next game action as shown below.

$$GameAction(i+1) = w_j * GameAction(i-j); \quad for j = 0 \quad to \quad n-1$$

where, $w_0 > w_1 > w_2 > w_3 > w_4 > .....w_{n-1}$ to ensure that older events carry less weight. Through set of experiments with players of different skill levels we found that past 5 actions are sufficient to achieve more than 90% accuracy. We currently use initial weights of $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$ and $\frac{1}{16}$ for w_0 to w_4 respectively so that $\sum_j w_j = 1$. The weights can be dynamically adjusted using the prediction error to improve

the prediction accuracy. We defer this weight tuning to future work. If the predicted game action is critical, then the game state is important

C. Resource Controller (RC)

The resource controller (RC) follows client/server architecture. A key consideration for the RC is determining how long to sleep the wireless network interface. RC has two levels of power management: Macro Level (has larger sleep duration) and Micro Level (sleep duration is limited). These are explained below.

1) Macro Power Management: At the macro level, the server side RC makes power management decision and sends *sleep* command to the client. It's decision relies on client's current AoI state, network data and the position of all players provided by the RDC.

First it checks the client's AoI state. If AoI state is not critical, it estimates the *Potential_Sleep_Duration* (PSD) using on the following algorithm.

Algorithm: *Potential_Sleep_Duration* **(PSD)**

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 Figure 2
$PSD = (currentProximity_j - AoI\_Visibility\_Radius)/relativeVelocity_j;$

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

The algorithm finds the nearest n interactive entities and estimates the time required for them to reach the current client's AoI. The smallest 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.

Computing $Euclidean$ distance requires a square root operation, which even on modern computers is expensive. As our computation is concerned only with comparing distances, comparing square of $Euclidean$ distances is equivalent comparing the distances. Hence, we compute square of $Euclidean$ distances to all other players using the formula below and pick the n nearest players. For these n nearest players we compute the actual distance. $Distance^2 = (X^2+Y^2+Z^2)$ - for three dimension; $Distance^2 = (X^2+Y^2)$ - for two dimension; where, X,Y and Z are $abs(x_1-x_2)$, $abs(y_1-y_2)$, $abs(z_1-z_2)$; (x_1,y_1,z_1) and (x_2,y_2,z_2) define the position of the client and an interactive entity.

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 Games (MMOGs), RC uses *interaction recency* as the key strategy. 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.

TABLE I
POWER CHARACTERISTICS OF DIFFERENT INTERFACES

Interface	Current (mA)		Mode Switch Penalty	
	ON	SLEEP	Current (mA)	Latency (ms)
old 802.11b	**600**	10	900	875
new 802.11b	**300**	6	350	625
802.11g	**260**	4	280	300
802.11g*	**260**	4	280	60
3.5G(HSPA)	**201**	8	300	1500
ZigBee	**50**	2.5	52	8

* - using atheros chipset; Current - Current Consumed

The server side RC sends two types of messages to the RC client - A *sleep* message with PSD and a *AoI state* message with AoI_STATE_CRITICAL flag. 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 power cost* and *mode switch latency* of the wireless interface). Our measurements on the average current consumption in ON and SLEEP states for various wireless interfaces along with the corresponding mode switch penalties are presented in Table I.

//Computing Effective Sleep Duration (ESD)

if (PSD $<$ *minSleepDuration)*
  ESD = 0;
elseif (PSD $>$ *maxSleepDuration)*
  $ESD = maxSleepDuration;$
else
  ESD = PSD;

2) Micro Power Management: At the micro level, RC relies on PAL value and data from GAPE. It is fully implemented at the client side. The client side RC invokes micro level power management once it receives $AoI\_STATE\_CRITICAL$ message from the server RC. Micro power management is based on the notion that the clients may not always interact when they are in each other's AoI visibility radius. If the predicted game action state is not important and PAL level is low then the ESD is set to $minSleepDuration$ otherwise 0. The wireless interface is put into sleep mode for ESD time. Note that as the mode switch penalties (Table I) are very low for modern interfaces (Atheros WiFi and ZigBee), $minSleepDuration$ is very low ($<= 100ms$) for these interfaces. Hence, GAPE prediction errors due to swift change in actions will not adversely effect the playability of the game.

In both macro and micro power management cases the client ensures that it receives at least one complete update before the

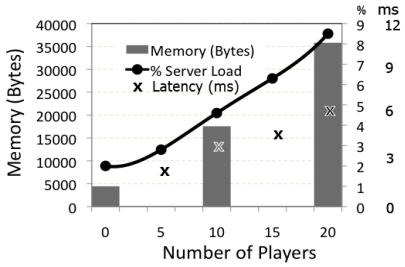

Fig. 3. Server Side Additional Memory and CPU

TABLE II
CLIENT SIDE IMPACT

Metric	MW Enabled	MW Disabled	Overhead
Current Drawn by CPU(mA)	213.39	207.29	6.1
Memory Used(KB)	2294.6	2293.1	1.5

next sleep to avoid longer inconsistencies.

D. Building ARIVU

The entire Middleware has been written in the Java Programming Language, and compiled to the JVM on the server side and Androids Dalvik VM on the clients side. The code size of the game is 7340 lines (Server 3285 + Client 4055) and ARIVU increases the size only by 1237 lines (Server 930 + Client 307).

It is important when designing server side computations, to take care that they do not add too much load so that the increased latency becomes noticeably large. The chart in Figure 3 shows that the load added by the middleware is linear or lower as the number of players increase. Average additional processing latency due to the middleware is less than 8ms for 20 player in a Intel Core 2 Duo (2.26 Ghz) server which is, relatively much smaller when compared to wireless network latency.

When designing such systems there is always the question of memory use on the server. To be truly scalable to large numbers of clients the incremental memory use per client should be optimized and be as low as possible. The chart in Figure 3 shows that the memory use of the middleware is small enough(1.5KB per player) that it will not be a problem.

On the client side the main concerns are power consumption and to a lesser degree, memory use. These should be minimized to be usable on a smartphone. Table II shows us that the memory use and power consumption incurred by using the middleware is small enough to be feasible.

Table I, shown above, compares the relative strengths with respect to current consumption in ON and SLEEP modes. Here we can see, from the current consumption stand point, Zigbee is the best interface to use. Among the commercial interfaces: 3.5G shows some promise, however time taken to turn on (from sleep state) is over 1.5 seconds, and thus infeasible for our purpose. Wifi on the other hand is a very suitable interface to use when we want to save power while gaming.

III. TESTING METHODOLOGY

In this section, we describe the hardware and software components as well as the process used to evaluate ARIVU.

Fig. 4. Armageddon Test Game

A. ARIVU Variants

We developed three variants of ARIVU. They differ in their performance and intrusiveness level (i.e., how much of the existing game client and server logic needs to be modified). We present the relative performance of each variant in Section IV-A. The three variants are:

1 **Full**: This is the full implementation of ARIVU, described in Section II, and includes all the client and server components. This variant requires the most changes to the game code to interface with ARIVU (using the APIs provided).

2 **Secured**: This is a variant of ARIVU that only implements the client-side changes (GAPE value and PAL). This variant may be more appealing to game companies initially as it is more secure and will not modify the server code – minimising the possibility of bugs or security holes entering the most ”mission-critical” component of the game.

3 **Blackbox**: In this variant, we make no changes to the client or server game components and easiest to deploy. It uses only PAL and network data.

In all the three modes the developer can set AGGRESSIVE_LEVEL which defines the *level of aggressiveness in saving power*. Higher value for AGGRESSIVE_LEVEL results in more power saving by trading-off quality. This can be exposed to the end-user as a tunable knob.

B. Test Game

To test ARIVU, we developed a simple Android-based RPG game, called Armageddon (screenshots shown in Figure 4). Armageddon is not a fully developed commercial game, but it has all the basic features of a RPG game including player movement within and across maps, fighting battles with monsters, and in-game communication and collaboration between players. The main goals of the game are to explore the various maps and to kill the enemies discovered. It has two main locations - friendly town areas and hostile areas containing monsters. The effective AoI visibility radius for friendly environment is 125 pixels and hostile area is 250 pixels.

We developed our own game instead of using a commercial game as we needed to tweak the parameters of the game precisely to test specific aspects of ARIVU. In particular, by building our own game, we are able to tweak the game

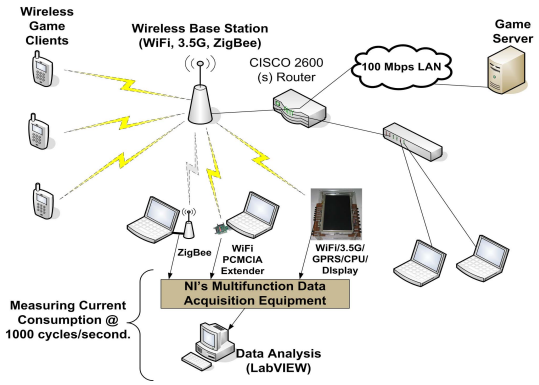

Fig. 5. Testbed Used for Experiments

interaction parameters so that we can simulate both role-playing as well as first-person shooting games — our basic game is an RPG while a variant that involves a lot of fighting closely in hostile area simulates a FPS game.

C. Testing Environment and Devices

Figure 5 shows the testbed used for our experiments. The game clients are connected through wired network and different wireless networks to the game server. We used a variety of Android phones (HTC Magic, HTC Dream, and Google Nexus One phones) as game clients. To obtain accurate power measurements, we used a custom made cell phone test borad, a high-speed multifunction Data Acquisition Equipment (DAQ) USB-6251 and a Signal Conditioning Equipment (SC-2345) from National Instruments (NI).

D. Testing Scenarios

Our test scenarios were designed to show the following:

1 **Base Effectiveness**: What is the potential power savings achievable by the three different variants (Section III-A) of ARIVU? In addition, what is the effect on game quality of each variant? We present these results in Section IV-A.

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 IV-B.

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 various power saving techniques. In all the scenarios, we varied the number of players in the game from 3 to 18 players to understand the effects of increasing the player density. Among these 6 are *human players* and the rest are *bots*. The map size for RPG and FPS variants is the same.

Due to lack of space we skip our analysis and results on efficiency of individual components of ARIVU . The results focus only on total power saving achieved by ARIVU while maintaining the quality of the game play and scalability of the server. There is a tradeoff between game quality and p (number of interactive entities the client is interested in) in MRIT based scalability enhancement. The results are shown for $p = 5$ entities.

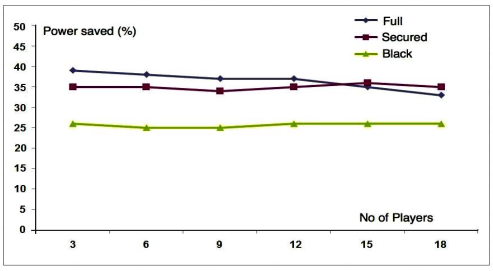

Fig. 6. Percentage of Power Saved (RPG game)

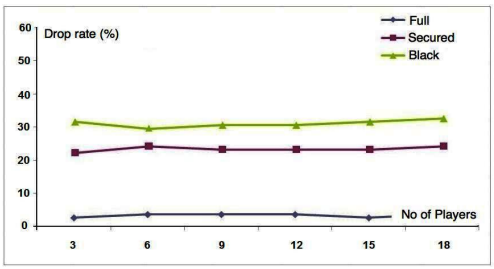

Fig. 7. Drop Rate of Important Packets (RPG game)

IV. Results

A. Results: Base Effectiveness

In the experiment, we consider two main factors: (%) of power saved and (%) of important packet lost. Loss of important packets is used to measure quality of the game. A packet is important for a client if when the packet is transmitted, there is at least one interactive object within its AoI with which there is at least one interaction. The results of 3 different modes are depicted in Figure 6 and Figure 7. These figures need to be considered together. When more information about the game state is available, ARIVU scores high in both the amount of power saved and quality of game play.

In **full mode**, the amount of power saved ranges from 40 to 35%. As ARIVU has necessary information about the game state the quality of the game play is guaranteed. Full mode is not highly affected by the false-positive errors of GAPE prediction as most of the power saving opportunities are contributed by macro power management algorithm. Full mode is relatively more sensitive to player density (number of players in the map). In **secured mode** the amount of power saved is close to values of full mode but the drop rate of important packets is very high. This is because secured mode depends only on the client side game action data. In **blackbox mode** the drop rate is close to the values of secured mode but, the the power saved is less than secured mode. The high drop rate of secured and blackbox modes can be reduced by decreasing the value of AGRESSIVE_LEVEL. These results prove that, we can save power while guaranteeing the quality of the game play when more information about the game is available. Making more information available to ARIVU middleware increases security risk. We have plans to tackle security risks in our future work.

B. Results: Game Type Effects

The above results are based on slow speed RPG games. We evaluate the results for high speed shooting games (such

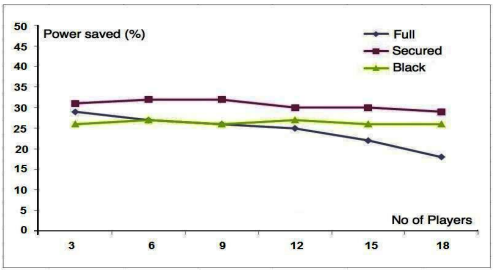

Fig. 8. Percentage of Power Saved (FPS game)

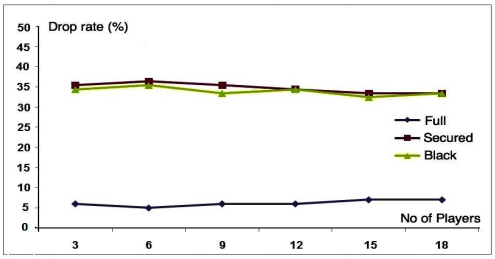

Fig. 9. Drop Rate of Important Packets (FPS games)

as Quake) by increasing the movement speed and densities of interactive objects in the game by 3 times and 5 times respectively. The results are depicted in Figure 8 and Figure 9. The results are similar to RPGs, except that the amount of power saved is lower due to the nature of the game play (player density and movement speed are high). These results shows that our algorithm can work well for high speed shooting games as well which is another extreme test case (worst case) for ARIVU.

V. 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 [8], using proxies to allow clients to sleep [3], and looking for statistical correlations that allow power savings [1], [10], [11]. In our work, we apply the lesson and techniques derived from these past works to the context of latency-critical interactive games. Dynamic voltage and frequency scaling is increasingly being used to reduce the CPU energy requirements in embedded and real-time applications – which maintains the application's real-time characteristics [9], [12], [13]. Accurately predicting run-time workload is the key for successfully using these techniques. These methods have also been applied to games [6], [7]. We apply some of those lessons in the design of ARIVU.

Interest Management. There are various algorithms for determining AoI [4], [5]. They are either distance based or visibility based technique for improving scalability. We use a combination of both techniques by taking their good features.

VI. Future work and Conclusion

The basic results on our efforts in building a complete generic middleware to manage overall system power consumption while playing massively multiplayer online mobile games is presented in this paper. Through macro and micro power management modules ARIVU tries to capture both longer possible sleep times and shorter ones for efficient power management. As ARIVU gathers information about the game state based on already existing information (eg. AoI, Game Actions, Player's Positions, Interactions) in the modern game implementations, these games need very minimum level modifications to use ARIVU middleware. We are currently enhancing our middleware to manage CPU and LCD-display power consumption to achieve higher integrated power efficiency. For example, when game state is not important the number of frames rendered by the CPU/GPU per second can be reduced. We are planing to test our middleware on different types of commercial games and improve our power management techniques, algorithms and API library to make it more generic for all types of mobile games.

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, M., Nightingale, E. B., and Flinn, J. Self-tuning wireless network power management. *Proceedings of the 9th International Conference on Mobile Computing and Networking (Mobicom)*, San Diego, CA, Sept. 2003.

[3] 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.

[4] 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.

[5] 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.

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

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

[8] 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.

[9] Poellabauer, C., Singleton, L., and Schwan, K. Feedback-based dynamic voltage and frequency scaling for memory-bound real-time applications. *Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)*, 2005.

[10] 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.

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

[12] Yuan, W. and Nahrstedt, K. Practical voltage scaling for mobile multimedia devices. *Proceedings of the 12th Annual ACM international conference on Multimedia*, 2004.

[13] Zhu, Y. and Mueller, F. Feedback edf scheduling exploiting dynamic voltage scaling. *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004.