Extending the Lifetime of a Network of Battery-Powered Mobile Devices by Remote Processing: A Markovian Decision-based Approach

Peng Rong and Massoud Pedram
Dept. of Electrical Engineering
University of Southern California
{prong,Pedram}@usc.edu

Abstract

This paper addresses the problem of extending the lifetime of a battery-powered mobile host in a client-server wireless network by using task migration and remote processing. This problem is solved by first constructing a stochastic model of the client-server system based on the theory of continuous-time Markovian decision processes. Next the dynamic power management problem with task migration is formulated as a policy optimization problem and solved exactly by using a linear programming approach. Based on the off-line optimal policy derived in this way, an on-line adaptive policy is proposed, which dynamically monitors the channel conditions and the server behavior and adopts a client-side power management policy with task migration that results in optimum energy consumption in the client. Experimental results demonstrate that the proposed method outperforms existing heuristic methods by as much as 35% in terms of the overall energy savings.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *wireless communication*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *client/server*.

General Terms

Algorithms, Design, Experimentations.

Keywords

Client-server system, remote processing, network lifetime, Markovian decision processes.

1 Introduction

Extending the battery lifetime is one of the most critical and challenging problems in mobile battery-powered systems. Dynamic power management (DPM), which refers to a selective shut-off or slow-down of the idle or underutilized components, has proven to be a very effective technique in reducing power consumption of such systems. However, an implicit assumption in all of the previous DPM works [11][12] is that local tasks of a mobile device are executed on the device itself. This is true if the mobile device has no communication capabilities with other mobile devices. However, when we consider a mobile host within a mobile network, which carries a wireless LAN card and can interchange data with other mobile hosts or fixed base stations over a wireless channel, the situation becomes quite different. A host with heavy workload may ask other hosts or the base station to help it reduce its workload by dispatching local tasks to these remote sites for processing. In this way, the mobile host may save power and extend its service lifetime.

Many key applications running on mobile platforms can benefit from task migration and remote processing. These applications include image processing, e.g., target detection and recognition used in robot control [1], voice recognition [2], and large-scale numerical computations [3].

The effectiveness of the remote processing technique is limited by the fact that data transmission over wireless channel results in additional power consumption. Energy savings on the local host is achieved only if the total energy for transmitting a task and receiving the result is less than the energy consumed for local execution of that same task. The rather large energy dissipation cost of wireless communication in mobile network of battery-powered devices makes the problem of deciding whether to execute a local task on the local host or to dispatch it to another mobile host for remote processing a very important one. In effect, energy-conscious policies must carefully consider the energy tradeoff between communication and computation and the task execution time from the viewpoint of the local host as well as the total energy dissipation for executing a task from the viewpoint of the network of mobile hosts.

A number of research results related to this problem have been reported in the literature. Experiments performed in [2][3] demonstrated the potential of remote processing for significant power savings in a number of real time tasks. Based on CPU measurements, reference [5] proposed an adaptive decision-making policy for a repetitive task. A remote processing framework was proposed in [4], which supports process migration at the operating system level. This adaptive policy differs from that proposed in [5] in that it can filter out the transient noise. Reference [1] proposed a compilation framework for remote processing, which can identify candidate remote computations within a single program. Unfortunately, these works do not consider any timing constraints on the tasks and assume that the user must be able to cope with any level of additional delay that may be introduced by remote processing. This limitation makes these techniques unsuitable for real-time applications, where violation of timing constraints may cause unacceptable loss in quality of service.

IEEE 802.11 protocol supports two types of mobile networks: peer-to-peer architecture (ad-hoc mode) and client-server architecture (infra-structure mode). In ad-hoc network, there is no base station and communication among mobile hosts takes place without the need for a base station. In this case, the major issue is to balance the remaining energy resources of all mobile hosts so as to maximize the ad-hoc network lifetime. This problem – although interesting - is different from the problem that we are addressing here for the infra-structure mode and is beyond the scope of the present paper. This paper targets a mobile device providing real time services in a client-server wireless network. The mobile battery-powered device (client) can communicate with and possibly migrate tasks to the "wall-powered" base station (server). The paper first presents a new Markovian Decision Process-based DPM framework for such a network. The proposed stochastic

This research was supported in part by DARPA PAC/C program under contract DAAB07-02-C-P302 and by NSF under grant no. 9988441.

DAC 2003, *June 2-6, 2003, Anaheim, California, USA.*

model is used for minimizing the power consumption of the mobile host by using remote processing while meeting real-time constraints.

The remainder of this paper is organized as follows. Related work and background are provided in Section 2. In Section 3, details of the proposed DPM framework are described. In Section 4, stochastic models of the client, the wireless channel, and the server are provided. In Section 5, the energy optimization problem is formulated as a mathematical program and two DPM policies are presented. Experimental results and conclusions are given in Sections 6 and 7, respectively.

2 Background

Research on wireless communication has demonstrated that the multi-path fading and shadowing (slow fading) effects in a wireless channel may significantly degrade the signal-to-noise ratio, increase the error rate, and thus cause a large amount of delay and energy consumption for re-transmitting the corrupted packets. So when determining an optimal policy for the client, a detailed and accurate model of a wireless channel should be constructed and used.

A (controllable) continuous-time Markov decision process [13] (CTMDP) is defined with a discrete state space; a generator matrix, where an entry represents the transition rate from one state to another; an action set; and a reward function. In CTMDP, the generator matrix is a parameterized matrix that depends on the selected action. An irreducible CTMDP has a unique limiting distribution that is independent of the initial conditions.

A complete system may consist of several components, each modeled by a CTMDP. The state set of the complete system is obtained as the Cartesian product of the state set of each component minus the set of invalid states. By using the method of [12], the generator matrix of the whole system can be generated from the generator matrices of its components by using the tensor sum and/or product operations.

3 DPM Framework

The proposed framework consists of three major components: the clients (mobile hosts), a server (base station) and a wireless channel which carries the communication packets between the client and the server. It is assumed that the server is AC-powered and has a much larger computational capability than the client. We also assume that the client services only its own local tasks and receives no request for remote processing from the server. This is a reasonable assumption since the AC-powered high-performance server is much more powerful from a processing point of view and has no energy limit, and thus it will execute its own tasks (in addition, it will execute tasks sent to it by the mobile hosts.) This also means that the server has all the hardware and software resources required to execute the tasks that are sent to it by the remote clients. Furthermore, for the same reasons, the server does not turn down any request for remote processing.

When a client desires to execute a task on the server, it sends a *remote process request* (RPR) to the server with a required real time constraint. Because the server may be busy executing other tasks (some local, other remote tasks that have previously arrived), it may reject the RPR from the mobile host because it may have determined that it cannot meet the required time constraint for the remote task. This is the only case in which the server rejects a RPR, that is, the server never turns down a RPR for reasons of server-side energy saving. When the RPR is rejected by the server, the client will have to perform the task locally. However, at that point, the client has wasted some valuable resources (energy and time) trying to migrate a task to the server and because it has failed in doing so, it still has to perform the requested task locally. It is therefore essential for the energy efficiency of the client and for its performance to minimize the probability for its RPR's to be rejected by the server.

The procedure/protocol for remote processing is explained next.

1. The client decides to migrate a task to the server. This task is called a *remote execution candidate* (REC). The client calculates the timing constraint for the execution of the REC.
2. The client sends a RPR to the server containing workload and timing constraint information about the REC.
3. When the server receives the RPR, it checks the status of the tasks waiting on the server to see whether the timing constraint for the REC can be satisfied. If so, the server will accept the application, otherwise, it will reject the application. Whether or not the application is accepted, the server will inform the client of its decision by sending an *acknowledgment* (ACK) back to the client. Included in the ACK response are the decision to accept or reject the RPR, and current status information about the server, i.e., the average incoming request rate and the average execution time of the tasks on the server side.
4. If the client receives a positive (acceptance) response from the server, then it will start to migrate the REC to the server. Otherwise, the client will proceed to execute the task locally.
5. When the client finishes the task migration step, it can immediately start processing a new task if one has arrived.
6. When the server completes the task, it will store the result in its own memory and immediately inform the client that the computation result is ready by sending a *task done* (DONE) message to the client.
7. If the client receives the DONE message from the server, then it will immediately contact the server and collect the computation result (RES). If the server does not see any activity from the client, then it will resend the DONE message at the next conference time. At that time, the client is guaranteed to be awake and therefore will receive the DONE message and will pick up the RES from the server. At the same time, if the client does not receive any message from the server and has not had a conference with the server since the last REC was sent off, then before the deadline for REC is expired, it will contact the server to pick up the RES. [1]

During the process of *RPR negotiation* (steps 1-3), *REC handoff* (step 4,5), and *RES computation* (step 6), and *RES delivery* (step 7), the client counts the number of packets that had to be re-transmitted due to unrecoverable errors in the received packets. This information will enable client to determine the wireless channel condition in real time.

4 Modeling

Because this paper focuses on the client-server architecture (i.e., the infra-structure mode of the IEEE 802.11b), we can assume that the mobile hosts (clients) in the network are independent of each other[2] and therefore when a client learns about the status of the server, it has all that it needs to make local decisions as to how it can improve its energy efficiency and thereby extend its battery lifetime. The client-server system is thus modeled by a joint CTMDP model, which is composed of CTMDP models of only three components: a single client, a wireless channel, and the server.

4.1 Model of the Client

We consider a (mobile) client that is continuously executing some real-time service processes for each incoming task. The QoS requirements for the client service are: 1) the average task delay is less than a predetermined value *D*; and 2) the task loss rate is less than a threshold

[1] This case really means that the server finished the RES computation, send a DONE message to the client who was sleep and thus missed the server ACK.

[2] Other mobile hosts affect the target mobile host only because of packet collision in the wireless channel. In this paper, we treat this collision effect (which is transparently handled and minimized by MAC layer) as noise in the wireless channel.

T_h. Different tasks differ in the task size which is exponentially distributed. It is assumed that the relationships between the task size and its execution time on the client and the migration time over an error-free wireless channel are known in advance (for example, through profiling).

The model of the client is illustrated in Figure 1. It has three processors: Service Provider (SP), Conference Processor (CP), and Issue Processor (IP).

The SP represents the CPU of a real mobile device and can provide service for the service requests (SR). The CP is in charge of negotiation with the server for remote processing and task migration. When a REC is selected, the CP first sends a request for remote processing to the server, which includes the basic information about the REC, such as its expected computational workload and the relevant timing constraint. When receiving a RPR, the server checks its own resources and workload to see whether or not it can finish the task in the required time. If the timing constraint can be met, the request is accepted; otherwise, it is rejected. If the CP receives an "Accept" response from the server, it starts to send off (migrate) the task to the server. After completing the task migration, the CP can immediately start a new negotiation with the server for the next REC. When the server finishes the required job, it stores the RES in its own memory and waits for the client to get them back. If the CP receives a "Reject" response, it moves the rejected REC out of the Conference Queue (CQ) and inserts it into the Priority Queue (PQ). The tasks in the PQ have a higher priority in receiving service from the SP compared to other tasks in the normal Service Queue (SQ). This makes sense because these tasks have already been held back because of the "failed" attempt to migrate them to the server. A typical CP is a WLAN Card with Direct Memory Access (DMA) capability. Since it can transmit and receive data with very little CPU intervention, we assume that the CP and the SP can work independently of one another. When a new task is generated, the IP decides whether to service it locally or make it a REC, and therefore insert the incoming task into the SQ or CQ, respectively. The IP is a low complexity and power-efficient processor (e.g., a PIC processor). We assume its power consumption can be neglected in comparison to the SP and the CP. The IP is always awake waiting for the arriving tasks and deciding whether to treat them as local or REC's.

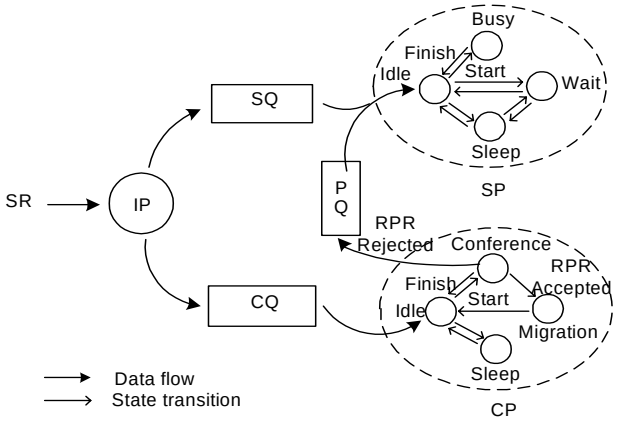

Figure 1. CTMDP model of a client.

The definitions of the states of the SP are as follows.

Busy: a working state, where the SP service the tasks waiting in the SQ or CQ.

Idle: a full-power but non-functional state, during which the Power Manager (PM) may issue any of the following commands to the SP: Go-to-Busy, Go-to-Wait, Go-to-Sleep, Stay-in-Idle.

Wait and **Sleep**: low power states. The SP in the Wait state has a higher power consumption compared to the Sleep state, but its transition to Busy or Idle state requires more time and energy.

The detailed states of the CP are explained as follows:

Idle: State reached when a RPR negotiation is concluded with a "Reject" response, or when the RPR is accepted by the server and the client has completed the task migration step. It is also the state in which the CP receives commands from the PM to determine whether to start a new negotiation, go to sleep, or stay in idle.

Conference: In this state, the client sends the RPR's to the server, waiting for a server response indicating acceptance or rejection of the current RPR. If the request is rejected, the CP goes to the Idle state and the REC is fetched out of the CQ and inserted into the PQ. If the REC is accepted, the CP goes to the Migration state.

Migration: This state is reached after a RPR is accepted by the server. In this state, the client sends all the data necessary for performing the task to the server through the wireless channel. When the data-sending process is concluded, the CP goes back to the Idle state and at the same time the migrated task is removed from the CQ.

Sleep: State reached when the PM decides to put the CP into the lowest power mode to save energy. In this state the front-end of the wireless LAN card is turned off, thus no communication from the server can be received.

It is worth noting that in the CP model, we do not explicitly create a state for receiving the data RES of an RPR that has been serviced by the server. The reasons are: 1) The remote processing protocol/procedure described previously guarantees the transmission of computation RES from the server to the client. 2) It is more convenient from a modeling point of view to account for the time and power consumption overhead of receiving the data RES of a RPR in the Migration state.

All of the state transitions of the CP are assumed to be either exponentially distributed (e.g., the transition from the Migration state to the Idle state) or instantaneous (the only case is for the transition from the Idle state to the Conference state.)

4.2 Model of the Wireless Channel

The Markovian chain model has proven to be a very successful mathematical tool to describe a wireless channel. A lot of Markovian chain based models have been proposed, from two-state "Gilbert Elliot" model [7] to hierarchical hidden Markov model [8]. Complex models work better in terms of capturing the higher order statistics of the wireless channel, but result in a nearly exponential increase in model complexity [9]. A real wireless channel is usually exposed to both fast and slow fading effects. The study in [10] suggests that a two-state Markov chain model is quite accurate and remains insensitive to different coding/modulation schemes when the fading is slow, whereas independent, identically distributed (i.i.d.) processes are suitable for describing the fast fading effects. Based on this study and other similar published results, in this paper, we adopt a two-state continuous-time Markov process to model the slow fading effect. We assign a constant packet error rate *PER* to each state. These rates represent the expected packet error rate of the i.i.d. processes for describing the fast fading effect. The wireless channel model is shown in Figure 2, where $1/\nu(1,2)$ and $1/\nu(2,1)$ represent the expectation time that the wireless channel remains in state $w1$ and $w2$, respectively. Notice that it is straightforward to extend the two-state model to a higher-order model with more states to achieve higher accuracy, but a two-state wireless channel model is sufficient for our purpose.

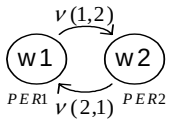

Figure 2. Two-state CTMDP model of wireless channel.

Let's define the average packet error rate (*PER, $0 \le PER \le 1$)* as the ratio of the number of un-recoverable packets, in spite of error-correction techniques such as CRC coding, to the total number of packets. We assume that any packet that is corrupted during transmission and for which error correction circuitry on the receiver side cannot fix the error

must be re-transmitted. Let t denote the time required for transmitting an n-packet data over an error-free wireless channel. The expected time t_a for transmitting the same data over an error-prone wireless channel can be calculated as follows:

$$t_a = n \cdot \sum_{m=0}^{\infty} t_0 \cdot PER^m = \frac{nt_0}{1-PER} = \frac{t}{1-PER},$$

where, t_0 denote the time for transmitting a single packet over an error-free wireless channel, and m is the number of re-transmissions.

4.3 Model of the Server

The server can be represented as an infinite M/M/1 queue [6] with a multi-state task generator as shown in the Figure 3. Usually a server connects to a number of clients and has to perform a large amount of local computations. So we can assume that, in the stationary state condition, 1) the rate of incoming tasks to the server is independent of any particular client and 2) this rate changes slowly. From the client's viewpoint, what is important is the rejection probability of its RPR's. Thus we can reduce the order of the model as explained below.

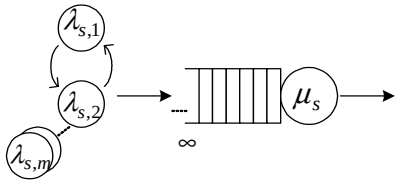

Figure 3. Queuing model of the server.

Assuming that the incoming task rate of the server is λ_s and its average service time is $1/\mu_s$. the probability that the number of waiting tasks in the server queue equals n, is computed as:

$$p_n = \left(\frac{\lambda_s}{\mu_s}\right)^n \left(1 - \frac{\lambda_s}{\mu_s}\right).$$

Suppose the service time constraint of a RPR is a constant factor c larger than its execution time on the server, where $c \geq 1$. The execution time of an RPR on the server can be approximated by an exponential distribution with a mean value k/μ_c, where $1/\mu_c$ is average service time of the client, k is the ratio of processing speed of the client to the server, $k \leq 1$. So the RPR rejection probability is calculated as:

$$p_{reject} = f(\lambda_s, \mu_s) = 1 - \sum_{n=0}^{\infty} p_n \cdot \Pr\{t_s > \frac{t_w}{c-1}\},$$

$$= \frac{\lambda_s}{\mu_s} \cdot \frac{\mu_c}{(\mu_s - \lambda_s)k(c-1) + \mu_c}, \qquad c > 1$$

$$p_{reject} = 1 - p_0, \qquad c = 1.$$

where t_s is the execution time of the RPR on the server and t_w denotes the waiting time of the RPR on the server. $LST^{-1}[\cdot]$ represents the reverse Laplace-Stieltjies Transform. Consequently, the model of the server can be reduced to a two-state Markovian process with rejection probabilities attached to each server state as shown in Figure 4. The rejection probability $P_{reject,1}$, $P_{reject,2}$ and $P_{reject,m}$ correspond to the incoming task rate states, respectively.

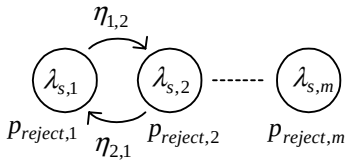

Figure 4. CTMDP model of the server with rejection probabilities.

5 Policy Optimization

We describe two policies: an *off-line optimal policy* and an *on-line adaptive policy*. The off-line optimal policy is computed based on the joint stochastic model of the client, the wireless channel and the server by using a linear programming approach. If the key characteristics of the wireless channel and the server are stable, then using the offline policy will result in the optimum energy saving solution. In practice, however, the channel conditions and the server workloads vary in time. For this time-varying situation, an on-line adaptive policy is devised to handle this time-varying situation. This on-line policy is based on dynamic lookup of pre-computed off-line optimal policies from a Cached Policy Table [14][15]. The key into this cache table is the parameter set that describes the channel conditions (packet error rate) and the server status (rejection probability for RPR's.) The value is the optimal policy that should be put to practice. Although the optimality of the on-line policy cannot be guaranteed (because of the client-side error and/or latency in determining the channel and server parameters), it has proven to be a satisfactory solution in a varying environment, especially if the dynamics of the network change are not too fast (cf. the results.)

5.1 Off-line optimal policy

Our goal is to find an optimal policy for minimizing the energy consumed by the client based on the characteristics of the client, the wireless channel, and the server. To consider the QoS requirements for the real applications, the optimal policy is solved under hard constraints on the expected task service time and task loss rate. A task is lost in (or dropped by) any of the client queues (SQ, CQ or PQ) if the queue is full when the task arrives. We formulate the policy optimization problem as a linear program as described next.

Let x represent the state of the whole power-managed system and a_x denote an action enabled in state x. The constrained energy optimization problem is formulated as a linear program as follows:

$$\text{Minimize}_{\{x^{a_x}\}} \left(\sum_x \sum_{a_x} f_x^{a_x} \gamma_x^{a_x} \right) \tag{5-1}$$

where, $f_x^{a_x}$ is the frequency that state x is entered in and action a_x is chosen in that state; $\gamma_x^{a_x}$ is the expected cost, which represents the expected energy consumed when the system is in state x and action a_x is chosen. It is calculated as:

$$\gamma_x^{a_x} = \tau_x^{a_x} pow(x, a_x) + \sum_{x' \neq x} p_{x,x'}^{a_x} ene(x, x'), \tag{5-2}$$

where, $\tau_x^{a_x} = 1 \Big/ \sum_{x' \neq x} \sigma_{x,x'}^{a_x}$ denotes the expected duration of time that the system will stay in state x when action a_x is chosen.

Subject to:

$$\sum_{a_x} f_x^{a_x} - \sum_{x'' \neq x} \sum_{a'_x} f_{x'}^{a'_x} p_{x',x}^{a'_x} = 0, \quad \forall x \in X \tag{5-3}$$

$$\sum_x \sum_{a_x} f_x^{a_x} \tau_x^{a_x} = 1 \tag{5-4}$$

$$f_x^{a_x} \geq 0 \tag{5-5}$$

$$\sum_x \sum_{a_x} f_x^{a_x} \tau_x^{a_x} (sq_x + pq_x) \leq \lambda_m D \tag{5-6}$$

$$\sum_x \sum_{a_x} f_x^{a_x} \tau_x^{a_x} (cq_x + pq_x) \leq \lambda_m D \tag{5-7}$$

$$\sum_x \sum_{a_x} f_x^{a_x} \tau_x^{a_x} \cdot cq_x \leq \lambda_m D_c \tag{5-8}$$

where λ_m is the average rate of incoming tasks for the client.

$$\sum_x \sum_{a_x} f_x^{a_x} \tau_x^{a_x} (\delta(SQ\ full) + \delta(CQ\ full) + \delta(PQ\ full)) \leq Th \tag{5-9}$$

where $\delta(x) = \begin{cases} 1, & if\ x\ is\ true; \\ 0, & otherwise. \end{cases}$.

In the above inequalities, sq_x, cq_x and pq_x represent the length of waiting tasks in the queue SQ, CQ and PQ. Notice that, the inequalities (5-6,7,8), which are based on the Litter's theorem [6], impose constraints on the expected task delay. In (5-6), pq_x is added because the tasks in the PQ have a higher priority and can block the tasks in the SQ, which means that a task waiting in the SQ will be serviced only when all tasks in the PQ have been serviced. It is thus necessary to account for the task delays in the PQ when considering the delay of a task in the SQ.

For delay calculation, we only consider the locally executed tasks, because we assume: 1) The timing constraint of each REC, which is assigned by the client and provided to the server during the RPR negotiation, equals c ($c \geq 1$) times its expected execution time on the server (cf. Section 4.3). 2) If the server accepts a RPR, it will definitely complete the task before its timing constraint. Therefore, the average delay of the RPR's that are executed on the server will be less than D, if the condition in the following theorem is satisfied.

Theorem: *Let $1/\mu_c$ and $1/\mu_m$ represent the expected value of the task execution time for locally processed tasks (this includes the task service time on the client SP) and for remotely executed tasks (this includes the RPR negotiation time, the REC migration time, and the RES delivery time), respectively; k represents the ratio of processing speed of the client processor to the server processor. If $D \geq D_c + 1/\mu_m + k/\mu_c$, $c = (D - D_c - 1/\mu_m)/(k/\mu_c)$, where D_c is defined in equation (5-8) then the expected value of the delay of remotely executed tasks is less than D.*

5.2 On-line policy

For the on-line policy, we assume that the status of the wireless channel and the server is not known a priori. So we must first construct a cache table of M×N entries off-line. Each entry (i,j) in this table corresponds to an optimal DPM policy computed based on the method proposed in Section 5.1 under the condition that the packet error rate of the wireless channel is PER_i and the rejection probability of the server is $P_{reject,j}$. The sets $\{PER_i\}$ and $\{P_{reject,j}\}$ for which an optimal policy is precomputed and stored in the table are determined by monitoring the channel and the frequency of RPR rejections over a moving window measurement. These indices of the cache table are arranged in an increasing order, i.e., $PER_i < PER_{i+1}$ and $P_{reject,j} < P_{reject,j+1}$.

In contrast to the off-line optimal policy, if during a predetermined period there are no RPR's, the on-line policy will arbitrarily select a task as a REC and send a corresponding RPR to the server. This is needed in order for the client to learn about the condition of the wireless channel and the status of the server.

The client uses profiling and regression to estimate the value of PER and P_{reject}. Let $APER^{(n)}$ denote the percentage of corrupted packets during the n^{th} conference with the server. The predicted value of the packet error rate $PER^{(n)}$ is calculated as:

$$PER^{(n)} = \alpha \cdot APER^{(n)} + (1-\alpha) \cdot PER^{(n-1)} .$$

where α is a coefficient and $0 \leq \alpha \leq 1$. α should be set to a larger value in a fast-changing wireless channel and to smaller value in a slow-changing wireless channel.

Let RR_N denote the rejection ratio of the last N RPR's. Let $\lambda_s^{(n)}$ and $1/\mu_s^{(n)}$ denote the incoming task rate and the average task service time in the server side (these are provided to the client by the server during the n^{th} conference time). Thus the predicted server rejection probability $P_{reject}^{(n)}$ is:

$$P_{reject}^{(n)} = \beta \cdot f(\lambda_s^{(n)}, \mu_s^{(n)}) + (1-\beta) \cdot RR_N .$$

where $0 \leq \beta \leq 1$ is a coefficient. β should be large if the workload status of the server changes rapidly; otherwise it should be small.

If one of the two conditions take place:

$(PER_{i-1}+PER_i)/2 < PER^{(n)} \leq (PER_i+PER_{i+1})/2 < PER^{(n-1)}$ or

$PER^{(n-1)} \leq (PER_{i-1}+PER_i)/2 < PER^{(n)} < (PER_i+PER_{i+1})/2$,

then the policy corresponding to the entry $(i, \cdot)$ will be activated. Similarly, if condition

$(P_{reject,j-1}+P_{reject,j})/2 < P_{reject}^{(n)} \leq (P_{reject,j}+P_{reject,j+1})/2 < P_{reject}^{(n-1)}$ or

$P_{reject}^{(n-1)} \leq (P_{reject,j-1}+P_{reject,j})/2 < P_{reject}^{(n)} < (P_{reject,j}+P_{reject,j+1})/2$,

is satisfied, the policy corresponding to entry $(\cdot, j)$ will be activated. Note that "$\cdot$" represents the unchanged index or index changed based on other conditions. The index i and j are calculated independently.

The flow diagram of the on-line policy is shown in Figure 5.

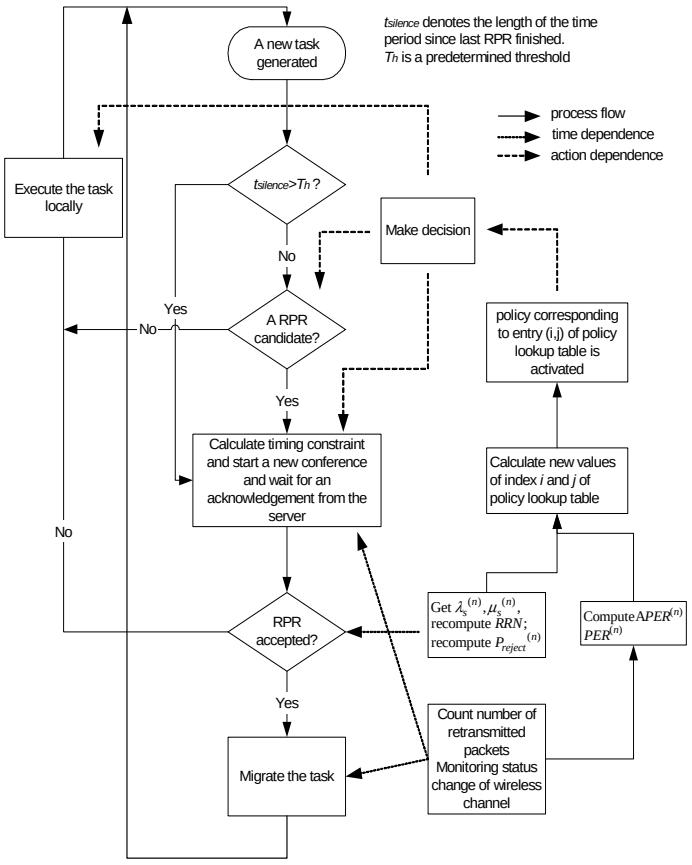

Figure 5. Flow diagram of the on-line policy.

6 Experimental Results

In our simulations, we used a StrongARM SA-1110 processor as the SP in the mobile host. The StrongARM processor was running at a clock frequency of 206MHz. The CP in the host was Orinoco WLAN PC card. The power consumption and state transition times of the StrongARM processor and the Orinoco WLAN PC card are reported as:

Table 1. Features of StrongARM SA1110 and Orinoco WLAN.

StrongARM SA1110	Busy	Wait	Sleep
Power (mW)	600 (with MEM)	100	0.2
Transition Time	Wait to Busy Busy to Wait	10 us	
	Sleep to Busy	160 ms	
	Busy, Wait to Sleep	90 us	
Orinoco WLAN card	Transmit	Receive	Sleep
Power (mW)	1400	900	50
Transition Time	Wake-up time	34 ms	
	Sleep-down time	62 ms	

In the simulation, we assumed that the average task execution time on the mobile host is 400ms, the conference time is 40ms, and the average RPR data migration time plus the RES pick up time is 80ms. The task incoming rate is 0.625 per second. The Maximum task loss rate is 0.1%. The average task delay constraint is less than 0.8s.

We compare the results of both our offline policy and online policy with two baseline techniques. These two baseline methods are:

M1: No RPR. The client will execute every task locally.

M2: Always try RPR first. For every incoming task, the client will first send a RPR to the server. The client will execute the task locally only if the server rejects the RPR.

Off-line policy

In Figure 6, Assume the state of the wireless channel and the server are unchanged. MDPBP, stands for Markov decision processes based policy, is our proposed method.

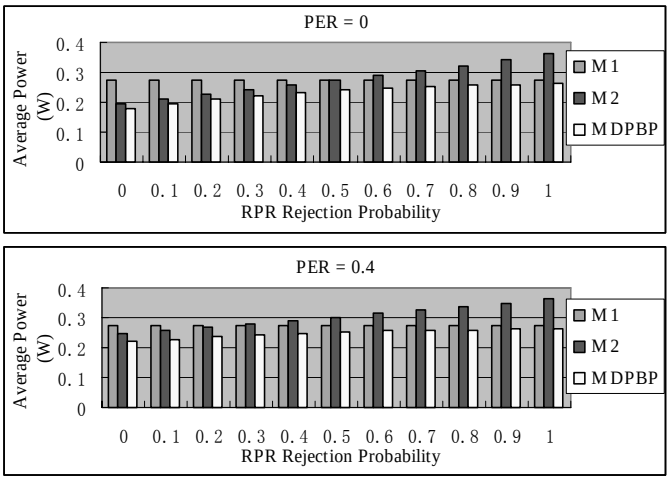

Figure 6. Comparison of simulation results of the three policies with an invariable wireless channel and server.

When the PER and RPR rejection probabilities are small, the M2 method results in large power savings compared with the M1 method. However, as the PER and RPR rejection probabilities increase, the average power consumption of M2 increases and finally significantly outweighs M1. This occurs because of both the energy wasted by the CP during the RPR negotiations and the extra energy consumed by the SP due to more stringent timing constraints (since an amount of time has been wasted for RPR negotiations.) The MDPBP always consumes the least power and achieves power savings as much as 34.9%.

Now consider a wireless channel and a server with time-varying characteristics. In this simulation, the server is simulated as an infinite queue with a Markovian process based task generator (task incoming rates $\lambda_{s,1}$ and $\lambda_{s,2}$). We assumed that the average task execution time on the server is 25ms and the processing speed of the server is 20 times faster than the client. Set D_c, c.f. inequality (5-8), was set equal to 0.65s. The rest model parameters are shown in Table 2. Results of the off-line policy are compared with the two baseline polices in Table 3.

Table 2. Model parameters of wireless channel and server.

PER1	PER2	$v(1,2)$	$v(2,1)$
0%	20%	1/15000	1/10000
$\lambda_{s,1}$	$\lambda_{s,2}$	$\eta(1,2)$	$\eta(2,1)$
16 per sec.	24 per sec.	1/20000	1/20000

Table 3. Simulation results of the off-line policy.

Policy	M1	M2	MDPBP
Average Power (W)	0.2742	0.2746	0.2412
MDPBP Improvement	12.0%	12.2%	--

On-line policy

In this simulation, the server is simulated as an infinite queue with a randomly generated task trace; the parameters of the wireless channel is slowly and randomly increased or decreased. The on-line policy is based on a 5×5 decision table. Due to the limited space, we cannot include this table here. Simulation results are shown in Table 4.

Table 4. simulation results of the on-line policy.

Mode	M1	M2	MDPBP
Average Power (W)	0.2742	0.2570	0.2310
MDPBP Improvement	15.8%	10.1%	--

7 Conclusion

A new mathematical framework for extending the lifetime of a mobile host in a client-server wireless network by using remote processing was proposed. The client-server system was modeled based on the theory of continuous-time Markovian decision processes. The DPM problem was formulated as a policy optimization problem and solved exactly by using a linear programming approach. Based on the off-line optimal policy computation, an on-line adaptive policy was developed ad employed in practice. Experimental results demonstrated the effectiveness of our proposed methods.

8 References

[1] U. Kremer, J. Hicks, and J. Rehg, "A compilation framework for power and energy management on mobile computers," *International Workshop on Languages and Compilers for Parallel Computing*, Aug. 2001.

[2] A. Smailagic, M Ettus, "System design and power optimization for mobile computers," *VLSI on Annual Symposium, IEEE Computer Society ISVLSI* 2002, pp. 15 –19, 2002.

[3] A. Rudenko, P. Reiher, G. Popek, and G. Kuenning, "Saving portable computer battery power through remote process execution," *Mobile Computing and Communications Review*, 2(1):19–26, 1998.

[4] A. Rudenko, P. Reiher, G. Popek, and G. Kuenning, "The remote processing framework for portable computer power saving," *ACM Symposium on Applied Computing*, San Antonio,TX, Feb. 1999.

[5] M. Othman and S. Hailes, "Power conservation strategy for mobile computers using load sharing," *Mobile Computing and Communications Review*, 2(1):44–50, 1998.

[6] J. H. Dshalalow, *Frontiers in queueing: models and applications in science and engineering*, Boca Raton, Fla.: CRC Press, 1997.

[7] E. O. Elliot, "Estimates of error rates for codes on burst-noise channels," *Bell Syst. Tech.J.*, 42:1977-1997, Sep. 1963.

[8] H. Yang, Alouini M. -S, "A hierarchical Markov model for wireless shadowed fading channels," *Vehicular Technology Conference*, pp. 640-644, 2002.

[9] O. Haggstrom, *Finite Markov chains and algorithmic applications*, Cambridge Univ. Press, Cambridge, New York, 2002.

[10] M. Zorzi, R. R. Rao, L. B. Milstein, "Error statistics in data transmission over fading channels," *IEEE Transactions on Communications*, vol. 46, No. 11, Nov. 1998.

[11] L. Benini, G. Paleologo, A. Bogliolo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. Computer-Aided Design*, pp. 813–833, Jun. 1999.

[12] Q. Qiu, Q. Wu and M. Pedram, "Stochastic modeling of a power-managed system-construction and optimization," *IEEE Transactions on Computer-Aided Design*, pp. 1200-1217, Oct. 2001.

[13] E. A. Feinberg, A. Shwartz, *Handbook of Markov decision processes: methods and applications*, Kluwer Academic, 2002.

[14] C.-H. Hwang and A. C.-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 28-32, 1997

[15] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu and G. De Micheli, "Dynamic power management for nonstationary service requests," *IEEE Transactions on Computers*, pp. 1345-1361, Nov. 2002.