ADAPTIVE ALGORITHMS FOR SENSOR ACTIVATION IN RENEWABLE ENERGY-BASED SENSOR SYSTEMS

A Thesis by

Sandeep Reddy Mereddy

Bachelor of Technology, J.N.T.U, 2006

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

December 2009

ADAPTIVE ALGORITHMS FOR SENSOR ACTIVATION IN RENEWABLE ENERGY-BASED SENSOR SYSTEMS

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

_____
Neeraj Jaggi, Committee Chair

_____
Ravi Pendse, Committee Member

_____
M. Edwin Sawan, Committee Member

_____
Hamid M. Lankarani, Committee Member

DEDICATION

To my family members and friends

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Neeraj Jaggi for his guidance, valuable suggestions, and continuous support throughout my research work at Wichita State University. I would also like to thank Dr. Ravi Pendse, and Dr. M. Edwin Sawan, for their support in my thesis research, and Dr. Hamid M. Lankarani, for reviewing my thesis.

I appreciate all my friends and research associates for their support and providing me with valuable suggestions during my stay in Wichita.

Finally, I am very grateful to my family members for their love and support.

ABSTRACT

Future sensor networks would comprise of sensing devices with energy harvesting capabilities from renewable energy sources such as solar power. A key research question in such sensor systems is to maximize the asymptotic event detection probability achieved in the system, in the presence of energy constraints and uncertainties. This thesis focuses on the design of adaptive algorithms for sensor activation in the presence of uncertainty in the event phenomena. Ideas from increase/decrease algorithms used in TCP congestion avoidance are applied to design an online and adaptive activation algorithm that varies the subsequent sleep interval according to additive increase and multiplicative decrease based upon the sensor's current energy level. In addition, the proposed algorithm does not depend on global system parameters, or on the degree of event correlations, and hence can easily be deployed in practical scenarios. Through extensive simulations, it is demonstrated that the proposed algorithm not only achieves near-optimal performance, but also exhibits more stability with respect to sensor's energy level and sleep interval variations.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

Figure Page

LIST OF ABBREVIATIONS

AIAD	Additive Increase Additive Decrease
AIMD	Additive Increase Multiplicative Decrease
AW	Aggressive Wakeup Activation Policy
CW	Correlation-dependent Wakeup Policy
DES	Discrete Event Simulation
EB-CW	Energy Balancing Correlation-dependent Wakeup policy
L	Energy Level
MIAD	Multiplicative Increase Additive Decrease
MIMD	Multiplicative Increase Multiplicative Decrease
MAIAD	Multiplicative/Additive Increase Additive Decrease
MAIMAD	Multiplicative/Additive Increase Multiplicative/Additive Decrease
MSS	Maximum Segment Size
RTT	Round-trip Time
SI	Sleep Interval
TCP	Transmission Control Protocol

CHAPTER 1

INRODUCTION

Sensor networks have profound implications for environmental surveillance and monitoring, health-care, and defense. For long-term monitoring of targeted environments, sensors are envisioned to be deployed with rechargeable batteries, which are capable of harnessing energy from renewable sources in the environment. For instance, Heliomote [1], [2], [3] a solar energy-harvesting platform, demonstrates the self-sustaining capability of a sensing device. These devices are extremely limited in resources, particularly energy and computational power. In addition, they operate in the presence of uncertainties and under dynamically changing and hostile environmental conditions. These factors necessitate the design of adaptive and distributed algorithms for the efficient operation and management of sensor networks. The performance achieved in a renewable energy-based sensor system is measured using the quality of event detection and reporting attained in the system. The goal of this thesis is to design sensor activation algorithms that can provide performance guarantees in the presence of resource constraints and varying environmental conditions. In particular, the algorithm should be implementable through low computational overhead and in an online manner requiring only information that is currently available to the sensor node. In other words, the algorithms should not depend on global system parameters and should be able to withstand the variations in those parameters and other operational conditions.

Typically, sensors are tiny, energy-constrained devices, with low recharge rates (dependent upon energy harvesting) and may have to spend a significant fraction of their time in an inactive (sleep) state. The application of specific events, which the sensor system is required to detect (and report), would occur randomly in the region of interest and could potentially

exhibit temporal correlations across their occurrences. The overall objective is to maximize the time-average event detection probability achieved by the system. The discharge of an active sensor depends on the activation algorithm as well as on the event occurrence process, while the recharge is based upon harnessing renewable energy sources. The following important question relative to a renewable energy-based sensor system is addressed: How should a sensor be activated (deactivated) in order to maximize the overall event-detection probability? A dynamic algorithm is desired, since predetermined scheduling is not feasible due to the randomness in the system.

In this thesis, the sensor energy model first proposed by Jaggi et al. [4] is used to model the renewable energy-based sensor system. Jaggi et al. [4] also discusses various sensor activation algorithms in the presence of temporal correlations in the event phenomena. In particular, this work shows that a Correlation-dependent Wakeup (CW) policy, which employs a constant sleep interval derived appropriately using system parameters, achieves near-optimal performance. However, this activation algorithm depends on all global system parameters, about which, in practice, the sensor may not have knowledge and may not be able to estimate with sufficient ease and accuracy. Also, these parameters could change over time, due to the variability in environmental conditions. Hence, there is a need to design efficient algorithms for sensor operations, which could easily be deployed in practice.

This thesis involves the design of an activation algorithm that does not depend on global system parameters but is still able to achieve performance that is close to optimal. It borrows insights from Transmission Control Protocol (TCP) congestion-avoidance algorithms, which allow the end host to vary the congestion window size dynamically, probing the level of uncertainty (network congestion) in the process. It has been shown [5] that Additive Increase

Multiplicative Decrease (AIMD) of congestion window size at the end hosts leads to the best performance among the linear increase-decrease algorithms in terms of throughput as well as fairness. In wired networks, the feedback or signal to trigger window increase or decrease could be the binary feedback from the network or the packet losses or the queue backlogs at the individual nodes. Similarly, in a renewable energy-based sensor system, the sensor's current energy level is a strong indicator of whether the sensor should act aggressively, by decreasing, or conservatively, by increasing, its subsequent sleep interval. In this thesis, an algorithm that varies its sleep interval dynamically based upon the sensor's current energy level is designed, and it is shown that the proposed algorithm performs close to optimal. Since the algorithm does not rely on global systems parameters and its decisions are based solely upon the sensor's current energy level, it easily can be implemented in practice.

The contributions of this thesis are the following:

- Design of an adaptive algorithm for sensor activation to maximize event detection probability in the presence of temporally correlated event phenomena.
- Comparison of various linear increase/decrease schemes to vary the sensor's subsequent sleep interval.
- A study of the performance of the proposed algorithm through extensive simulations to demonstrate its near-optimality, stability, and feasibility towards practical deployment.

This thesis is organized as follows. Chapter 2 discusses related work in activation algorithms in renewable energy-based sensor systems, and also work in increase/decrease algorithms for congestion avoidance. Chapter 3 describes the discrete event simulation of a queueing system, which is later extended to simulate the sensor system and to evaluate the performance of a sensor system under various algorithms. Chapter 4 describes modeling and the

discrete event simulation of a renewable energy-based sensor system. Chapter 5 discusses the design of adaptive algorithms for sensor activation. The performance of proposed algorithms through extensive simulations are evaluated in Chapter 6, and the conclusions and future research goals are summarized in Chapter 7. The discrete event simulation code is provided in the appendix.

These research results have also been accepted for publication [6].

CHAPTER 2

LITERATURE REVIEW

2.1 Related Work

Various researchers [7], [8], [9] have considered the dynamic activation question in the context of energy harvesting sensor systems. Banerjee et al. [7] considers the single sensor and models it as a closed three-queue system, obtaining Norton's equivalent of the system to evaluate the structure of the optimal rate control policy. Banerjee and Kherani [8] consider the sensor activation problem under a sensor energy model similar to one developed by Kar et al. [10] and demonstrate the optimality of threshold-based policies for a broad class of utility functions and state dynamics. Gatzianas et al. [9] consider the resource allocation problem in the presence of rechargeable nodes, proposes a policy which decouples admission control and power allocation decisions, and shows that it achieves asymptotically optimal performance for sufficiently large battery capacity to maximum power ratio.

Sensor node activation algorithms for rechargeable sensor systems in the presence of temporally correlated event phenomena have been previously considered [4]. In particular, Jaggi et al. [4] propose a Correlation-dependent Wakeup policy, wherein the sensor employs a constant, deterministic sleep interval, which is derived using energy balance during a renewal interval of sensor operation. The proposed algorithm is shown to achieve near-optimal performance. However, this activation algorithm depends explicitly on global system parameters, about which, in practice, the sensor may not have accurate knowledge. Moreover, these parameters may not be easy to estimate in practice, particularly because they are susceptible to sudden changes due to variable environmental conditions. Therefore, this thesis focuses on the

design of algorithms that rely only on available information, such as a sensor's current energy level, in making activation decisions.

Chiu and Jain [5] considered different linear increase and decrease algorithms for congestion avoidance in computer networks, and showed that AIMD of the congestion window converges to an efficient and fair state, regardless of the starting state of the network. The AIMD-based approach has been previously applied in sensor networks for power control [11], for achieving fair wake-up rates among equivalent nodes [12], and for low power listening [13]. In this thesis, however, activation algorithms were designed for a single sensor node, wherein the sensor varies its subsequent sleep interval according to linear increase and decrease algorithms, depending upon the sensor's current energy level.

2.1.1 Drawbacks of Correlation-Dependent Wakeup Policy

The Correlation-dependent Wakeup policy [4] applies a constant sleep interval, which is derived using system parameters such as q, c, p_c^{on}, p_c^{off}, δ_1, and δ_2. The drawback here is that, in practice, the sensor may not know about these parameters.

2.1.2 Objectives of New Activation Policy

The proposed activation algorithms should achieve the following criteria: the sleep interval should depend only on energy level and be independent of other system parameters. These algorithms should dynamically calculate the sleep interval. The performance (event detection probability) of these activation policies should be close to optimal.

2.2 Motivation

2.2.1 Transmission Control Protocol

The Transmission Control Protocol (TCP) is a transport layer protocol that corresponds to Layer 4 of the Open System Interconnection Reference (OSI) model. TCP provides reliable data transmission between end hosts. Different services that are provided by TCP are stream data transfer, reliability, efficient flow control, full-duplex operation, and multiplexing. The TCP groups streams of unstructured bytes from the application layer into segments and passes that information to the Internet Protocol (IP) for delivery. This process avoids the applications to chop the data before handing it off to the TCP. The TCP offers reliability by connection-oriented, end-to-end reliable packet delivery.

2.2.2 Congestion in TCP

When data travels from a fast local area network (LAN) to a slower wide area network (WAN), congestion occurs. Congestion also occurs when a large number of packets arrive at the input queue buffer of a router. A congestion-avoidance mechanism attempts to decrease the packet loss. When congestion occurs between the end hosts, the TCP [14] must slow down its transmission rate of packets into the network and then invoke a slow start.

2.2.3 TCP AIMD Mechanism

For congestion avoidance, the TCP uses the additive increase with the multiplicative decrease mechanism [5], [15]. When congestion occurs, the congestion window size is decreased exponentially. Figure 1 depicts a long-lived TCP connection congestion window. The AIMD algorithm states that initially the congestion window increases additively by one maximum segment size (MSS) for every round-trip time (RTT) until there is a packet loss. When a packet loss is detected, the congestion window will be multiplicatively decreased. This means that it

decreases its congestion window size by half, which results in a saw-tooth behavior, as shown in Figure 1.

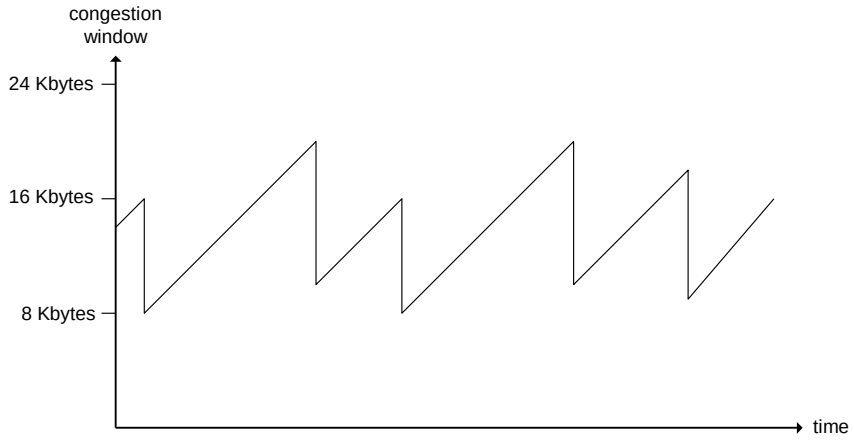


Figure 1. TCP AIMD mechanism [16].

Other algorithms for fairness in congestion control include Additive Increase Additive Decrease (AIAD), Multiplicative Increase Additive Decrease (MIAD) and Multiplicative Increase Multiplicative Decrease (MIMD).

Figure 2 depicts the AIMD mechanism of a congestion window [5]. Initially Host A sends one segment to Host B. If Host A receives acknowledgement, then Host A increases its window size by sending two segments to Host B. If Host B sends only one acknowledgement for the two segments that it received, then Host A decreases the window size multiplicatively, which means that this time it sends only one segment and then the process is repeated.

2.2.4 Dynamic Sleeping

Figure 3 depicts the allocation of bandwidth between two users with varying congestion windows.

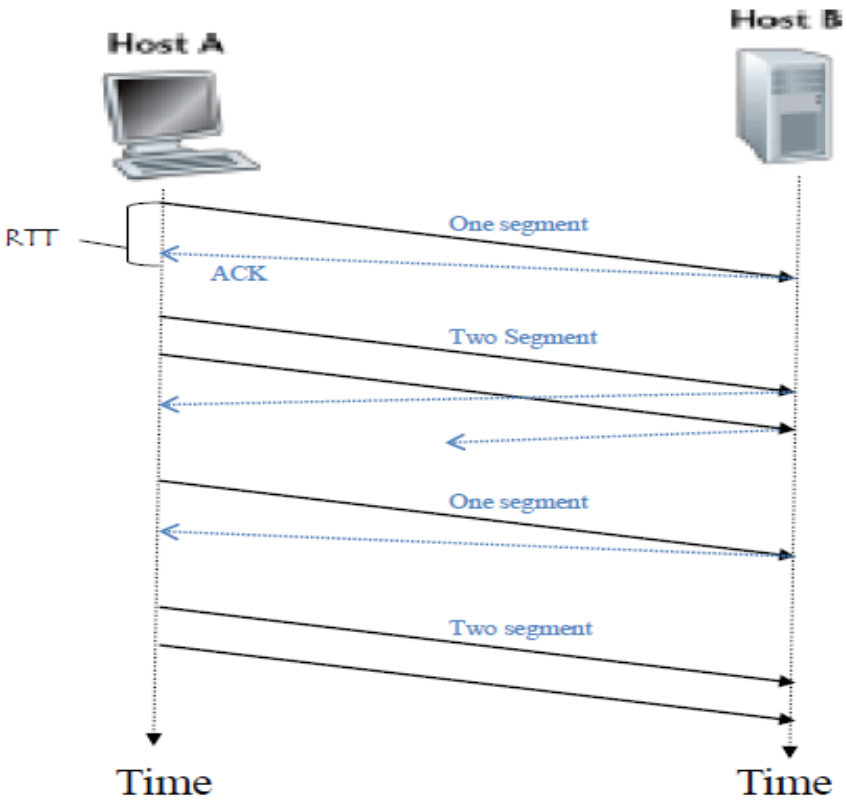


Figure 2. Congestion window of TCP AIMD mechanism [16].

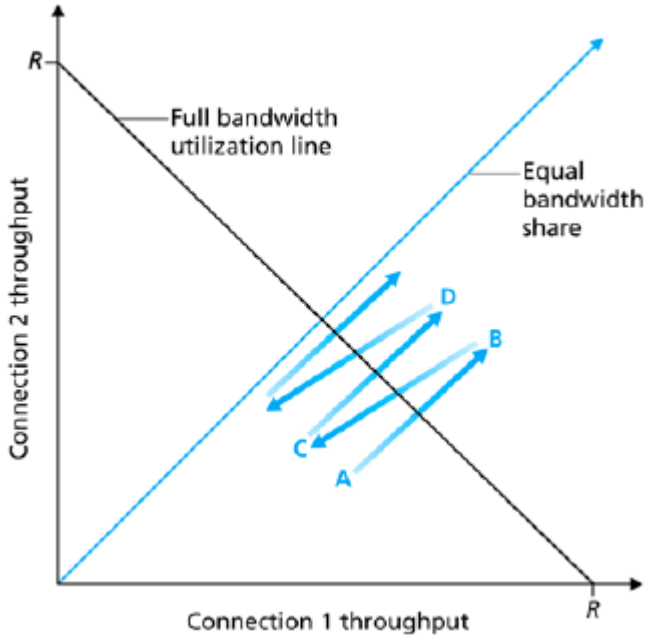


Figure 3. User's allocation in TCP AIMD mechanism [16].

Optimal performance can be defined as the point where the bandwidth utilization (efficiency) line and the equal bandwidth share (fairness) line intersect. The AIMD policy states that the congestion window oscillates (A, B, C, D in Figure 3 are oscillating points), which makes the performance of the allocation between two users to oscillate near the optimal point. This means that the TCP does not know the amount of congestion in the network and tries to estimate or probe the level of congestion using the AIMD mechanism. It does not know the best value of the congestion window to use. Similarly, a sensor may not know about system parameters and their values. It tries to estimate the best value of the sleep interval (SI) using the AIMD-based approach. This is the basis of the algorithm designed in this thesis. Since the SI is computed online dynamically, the adaptive algorithms designed in this thesis exhibit dynamic sleep scheduling of the sensor node.

2.3 Summary

In this chapter, related work, existing activation policies, the need for new activation algorithms, and the objectives of new activation algorithms were discussed. The chapter also talked about the motivation for a new activation algorithm based on similarities between the TCP congestion window choices and the sensor's sleep interval choices.

CHAPTER 3

SIMULATION OF QUEUING SYSTEM

3.1 Simulation of M/M/1 Queueing System

3.1.1 Introduction

Simulation is a process through which a model can be numerically evaluated. Analytical solutions for the discrete event system (DES) are not easy to achieve. A DES is a system whose state changes entirely depending on the asynchronous discrete events occurring over time. Three basic elements comprise a queueing system: (a) *entities*, which do the waiting in their quest for resources; these can be referred to as customers; (b) *resources* are those things for which the waiting is done; these can be referred to as servers; and (c) *queue* is the space where the waiting is done.

3.1.2 Simple Queueing System

A simple queueing system is a system with a queue and a server, where each arriving customer will either wait in the queue until the server is available or will proceed to the server and get served. After receiving service, the customer departs from the system.

Figure 4 represents a simple queueing system, where a circle represents the server. A server is a "delay block," which holds the customer for a certain amount of service time. The open box with slots represents the queue and the number of slots represents the waiting customers in the queue.

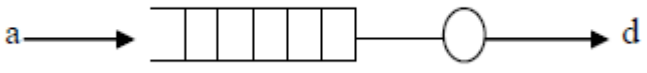


Figure 4. Simple queueing system.

Two types of events can change the state of the system: (a) customer arrival and (b) customer departure.

3.1.3 Designing a Queueing Model

Three components are needed for designing a queuing system model:

- Stochastic models for arrival and service processes.
- Specification of structured parameters of the system (e.g., number of queues, servers).
- Operating policies or conditions for accepting arriving customers, giving preference to certain customers, and so on.

3.1.4 Performance of Queueing System

Different parameters involved in studying a queueing system include the following:

- Waiting Time: Denotes the time taken by the customer from arrival instant until beginning of service.
- System Time: Denotes the time taken by the customer from instant of arrival until departure.
- Queue Length: Denotes the length of the queue.
- Utilization of the System: Defined as the fraction of time that the server is busy.
- Throughput of the System: Defined as the rate at which the customer leaves the system after service.

3.1.5 Event Scheduling Scheme

Cassandras and Lafortune [17] describe the event scheduling scheme, which helps generate system evolution over time. All components of the event scheduling scheme simulator for generating a sample path are described as follows:

The state stores a list of all state variables. Time is a variable that stores the current simulation time. A scheduled event list stores a list of scheduled events and their occurrence time. Data registers are variables that store data that is useful for estimation purposes. The initialization routine initializes all simulation data structures at the beginning of a simulation run. The time update routine identifies the occurrence of the next event and advances the simulation time to the occurrence time of that event. The state update routine updates the state based on the next event to occur. Random variant generation routines are a collection of routines that transform computer-generated random numbers into random variants according to user-specified event lifetime distributions. The report generation routine computes estimates of various quantities of interest based on all data collected during a simulation run. The main program is responsible for the coordination of all components by, first, calling the initialization routine; then repeatedly calling the time, state and report generation update routines; and then updating the scheduled event list. It is also responsible for termination of a simulation run.

3.1.6 Algorithm of Event Scheduling Scheme

First, remove the first entry (e_1, t_1) from the SCHEDULED EVENT LIST. Update the simulation TIME with the new event occurrence time t_1. Update the STATE to the new state x' by using the state transition function:

$$x' = f(x, e_1) \tag{3.1}$$

Delete any entries corresponding to infeasible events in the new state from the SCHEDULED EVENT LIST. Add any feasible event that is not yet scheduled to the SCHEDULED EVENT LIST. The scheduled time for such i^{th} event is given by (TIME + v_i), where TIME was set previously, and v_i is a lifetime obtained from the RANDOM VARIATE GENERATOR. Rearrange the SCHEDULED EVENT LIST based on a smallest-scheduled time first scheme.

Repeat the above procedure until the end of the simulation time, which will be specified by the user.

3.1.7 M/M/1 Queueing System

The M/M/1 queueing system assumes that the inter-arrival and service times are exponentially distributed with a single server and a queue with infinite capacity. Figure 5 depicts the M/M/1 queuing system.

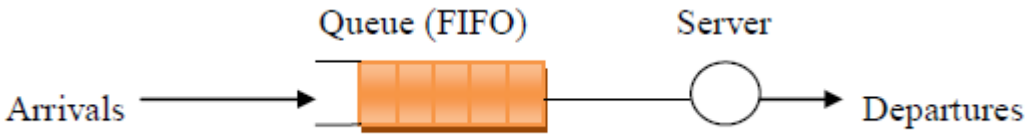
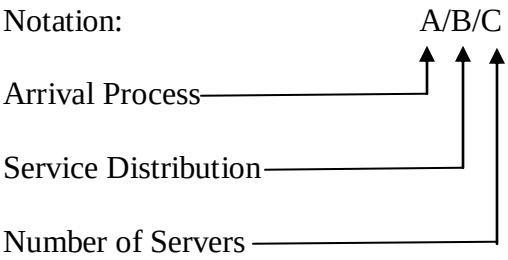


Figure 5. M/M/1 queueing system.



M stands for the Markov process, a memory less system whose transition probabilities depend only on the current state. Equations for the M/M/1 system describe the M/M/1 queuing system as straight forward and easy to use [17]. First, traffic intensity "ρ," the ratio of the average arrival rate to that of average service rate, is defined as

$$\rho = \lambda / \mu \qquad (3.2)$$

The stability condition for the M/M/1 system is given by traffic intensity $\rho < 1$, which means that the average service rate should be greater than average arrival rate. The average queue length is given by

$$E[X] = \rho / (1-\rho) \qquad (3.3)$$

Using Little's Law, the average system time is given by

$$\lambda E[S] = E[X] \tag{3.4}$$

Substituting equation (3.4) into equation (3.3) yields

$$E[S] = 1/ (\mu - \lambda) \tag{3.5}$$

Utilization is given by ρ, and throughput is given by λ.

3.1.8 Modeling of M/M/1 Queueing System

The methodology adopted for modeling the M/M/1 queuing system is the event scheduling scheme. The set of events of the M/M/1 queueing system are arrival event and departure event. State is defined as the number of customers in the system and varies from zero to infinity. The M/M/1 system can be simply modeled as a sample path with arrival and departure events occurrence. Between an arrival and departure of an event, there can be several arrival events.

All components of the M/M/1 queueing system simulator are described as follows:

- Queue List: A List of Arrival times for each customer in the system.
- Arrival Process: Generates arrival time when there is arrival event and adds this event to the queue.
- Departure Process: Generates service times for each arrival and departs the customer from the queue which obeys FIFO discipline.

3.1.9 Algorithm for M/M/1 Queueing System

The algorithm for designing the M/M/1 queueing system is described in the following steps:

Step 1: Declaration of variables: EVENT LINKED LIST and QUEUE LINKED LIST. The EVENT LINKED LIST stores the arrival time of each customer, and a pointer with each list

points towards the next customer's arrival time. The QUEUE LINKED LIST is a collection of list of arrivals.

Step 2: Declaration of initialization, random number generator, arrival, and service functions. The initialization function initializes all declared variables. The random number function generates exponentially distributed arrival and service times.

Step 3: Establishment of the arrival function, which calls the random number generator function, obtains the arrival times of each arriving customer, and adds each arrival events to the LIST OF ARRIVAL EVENTS QUEUE.

Step 4: Establishment of the service function serves and dequeues events from the LIST OF ARRIVAL EVENTS QUEUE in the first come-first served discipline. If there are any arrivals between arrival time and departure time of an event, then the service function executes next arrivals by calling the arrival function.

Step 5: The main function initially calls the arrival function since in the initial state, no departures occur; then it repeatedly calls the service function until the end of simulation.

3.1.10 Simulation Results

The simulation program for the M/M/1 queuing system is written in C programming language, which is provided given Appendix A. Simulations are performed for various system parameters for the M/M/1 queueing system. The system performance is measured using the average system time, average queue length, and utilization of the M/M/1 queueing system.

It is assumed that the queue capacity of the M/M/1 system is infinite for all simulations. In addition, it is assumed that service rate, "μ," is a constant value and arrival rate, "λ," is varied from one to nine. Figures 6, 7, and 8 depict the steady state values and DES outputs of system parameters of the M/M/1 queueing system. Steady state values are the values obtained using

equations (3.3), (3.5), and (3.2), respectively. Steady state represents the performance achieved theoretically. DES output represents the discrete event simulation output of the M/M/1 queueing system. Figures 6, 7, and 8 show that the discrete event simulation-generated outputs are close to that of steady state values. This verifies the accuracy of the discrete event simulation approach, which is used later to evaluate sensor system performance.

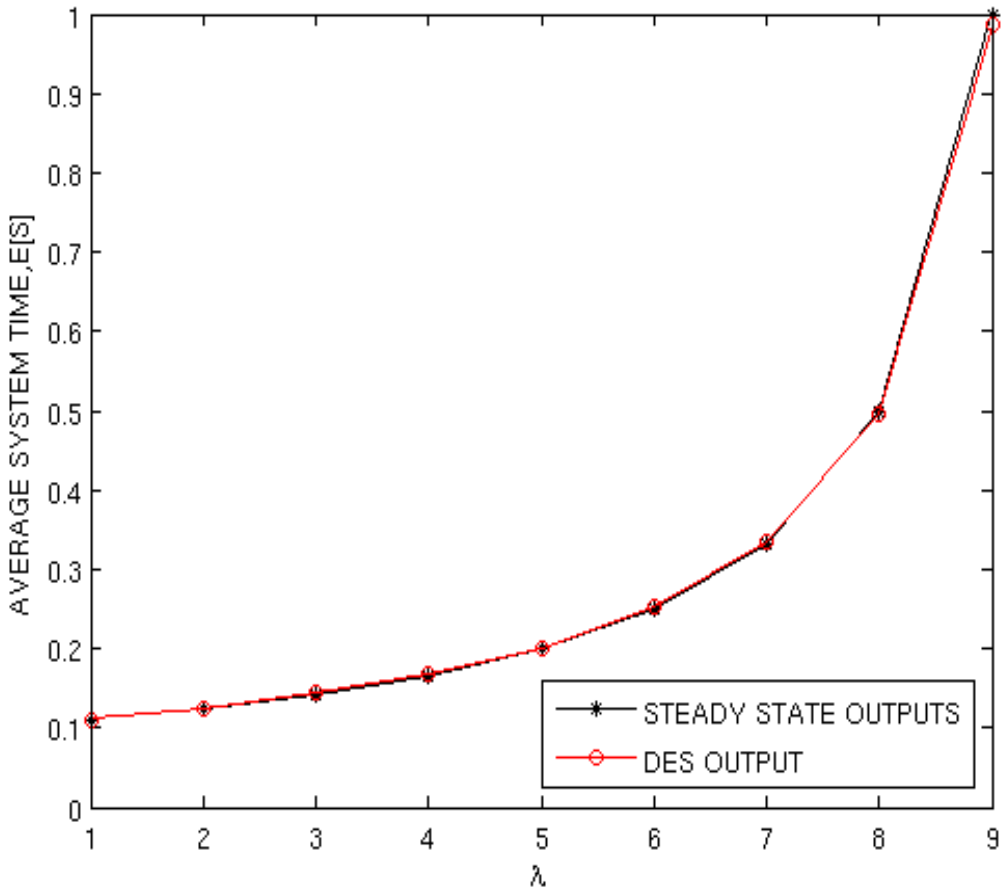


Figure 6. Average system time vs. λ.

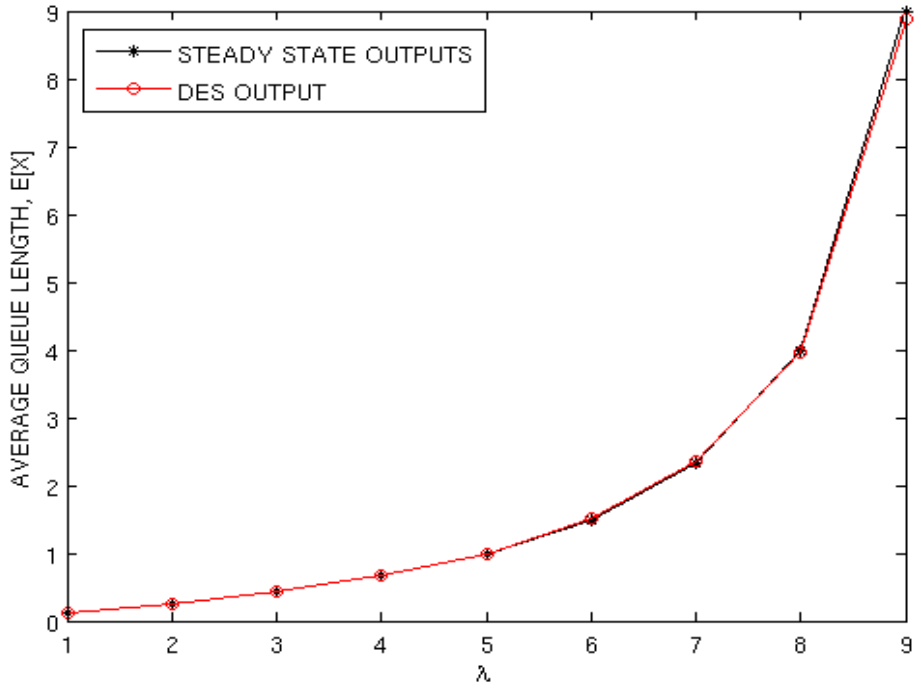


Figure 7. Average queue length vs. λ.

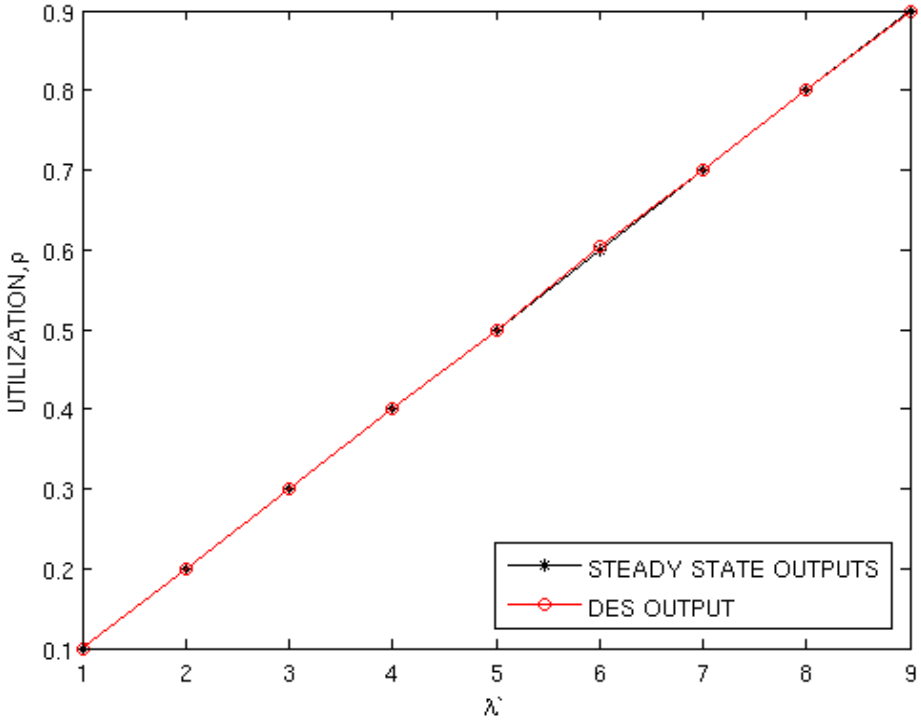


Figure 8. Utilization vs. λ.

3.2 Summary

This chapter discussed the simple queueing system and how to simulate it. The event scheduling scheme was also discussed, as well as the M/M/1 queueing system. An algorithm for simulating M/M/1 queueing system was developed.

CHAPTER 4

RENEWABLE ENERGY-BASED SENSOR SYSTEMS

4.1 Sensor System Modeling

4.1.1 Problem Statement

Rechargeable sensors are deployed to detect interesting events. These events may exhibit temporal correlations across their occurrences. The aim of this thesis was to design activation algorithms for a renewable energy-based sensor system in order to maximize the overall event-detection probability. These algorithms should allow the sensor to make dynamic decisions, such as when to stay active, when to sleep, and for how long. And these algorithms should be able to achieve good performance and be adaptable to varying system parameters.

4.1.2 Problem Formulation

The renewable energy-based sensor system model is described below. The energy bucket of a sensor stores energy in units of a quantum. A discrete time model was assumed, whereby in each time slot, a recharge event occurs with a probability "q" and charges the sensor with a charge of "c" quanta. The size of the sensor energy bucket is denoted by K.

The discharge process at the sensor depends upon its (activation) state as well as on the state of the event phenomena. The sensor having a non-zero energy level is said to be in an active (sleep) state if it has been activated (deactivated) in the current time slot. The sensor discharges energy only when it is in the active state. The sensor expends a charge amount of "δ_1"quanta (operational cost) during each time slot that it is active. In addition, if an event occurs and is detected by the active sensor, the sensor expends an additional charge amount of δ_2 quanta (detection and transmission cost). It can be assumed that if the sensor is active and an event occurs, the event is detected with probability 1. It can also be assumed that the sensor is available

for activation as long as it has sufficient energy to operate for at least one time slot, i.e., its energy level is at least "$\delta_1 + \delta_2$."

The event phenomena, which the sensor system is required to detect and report, is modeled as a correlated stochastic process in order to characterize the inherent randomness and temporally correlated event occurrence. The extent of temporal correlations is specified using correlation probabilities p_c^{on} and p_c^{off}, such that $\frac{1}{2} \leq p_c^{on}, p_c^{off} \leq 1$. If an event occurs during time slot t, then in the next time slot (t + 1), a similar event occurs with probability p_c^{on}, while no such event occurs with probability $1 - p_c^{on}$. Similarly, if no event occurs during the current time slot, no such event occurs in the next time slot with probability p_c^{off}. The event occurrence process used to model the event phenomena is comprised of an alternating sequence of periods where events occur (ON period) and do not occur (OFF period). In practice, events of interest occur rarely; therefore, the OFF periods are expected to be significantly larger than the ON periods (i.e., $p_c^{off} \geq p_c^{on}$). Figure 9 depicts the energy discharge-recharge model of a typical sensor, where the recharge rate depends on q, c, and the discharge rate depends on δ_1, δ_2, the state of event process, and the activation algorithm. The objective is to maximize the asymptotic event-detection probability achieved in the system. By letting $\varepsilon_o(T)$ denote the total number of events that occur in the region during time interval [0 . . . T] and $\varepsilon_d(T)$ denote the total number of event detected during this interval by the sensor operating under the activation algorithm Π, then the asymptotic event detection probability, denoted by U (Π), is measured in terms of time-average fraction of events detected or

$$U(\Pi) = \lim_{T \to \infty} \frac{\varepsilon_d(T)}{\varepsilon_o(T)} \tag{4.1}$$

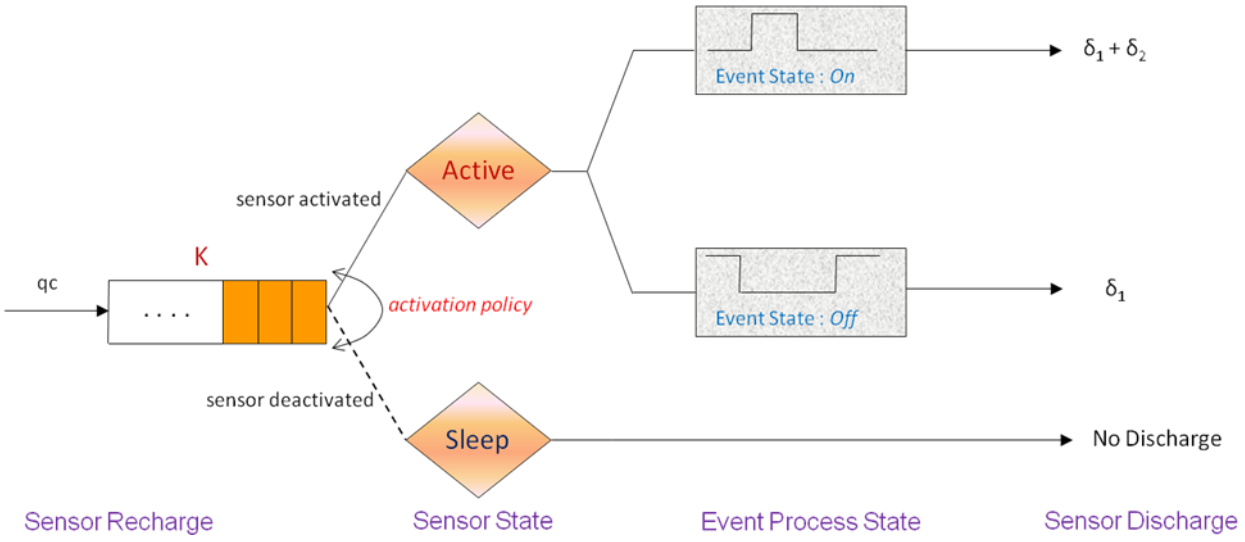


Figure 9. Energy discharge-recharge model of the sensor.

The decision problem is that of finding the activation algorithm Π^* such that $\Pi^* = \arg \max U(\Pi)$. In this thesis, activation algorithms were designed for a single sensor node, which makes dynamic decisions (when to stay active, when to sleep, and for how long), which are independent of global system parameters including the temporal correlation probabilities and are able to achieve near-optimal performance.

4.2 Discrete Event Simulation of Sensor System

4.2.1 Introduction

In recent years, the major evolution of hardware technologies has led to the development of small, low-cost sensor devices. The major application of sensor networks includes gathering data from the deployment region. There are numerous data-gathering applications in military surveillance, environmental and health monitoring, and disaster recovery.

Due to the miniaturized nature of sensors, they are heavily constrained in terms of energy. Sensors are generally powered by battery. Sensors deployed in remote locations are envisioned to be powered using a rechargeable battery system. A rechargeable battery recharges

by using renewable energy sources such as solar energy. In long-term monitoring applications sensor battery lifetimes will play a key role. For effective energy management of sensors, a sensor should be activated when it is expected to improve the system performance significantly; during the remaining times, the sensor should be kept in an inactive (sleep) state to conserve energy.

4.2.2 Renewable Energy Based Sensor Node

A rechargeable sensor can be described as a simple queue that stores energy in terms of quanta. The arrival process corresponds to "c" quanta for each slot whenever there is a recharge event occurrence with probability "q," and the queue discharges as per the activation policy.

The sensor can be in different states such as "active," "sleep," or "dead," depending on the energy level and the activation policy used.

4.2.3 Sensor System Simulation

The algorithm for rechargeable sensor node is developed in C programming language, which is provided in Appendix B. The event scheduling scheme model described in section 3.1.5 is used to develop the algorithm for the sensor system. This sensor system algorithm mainly consists of three modules:

- Recharge process
- Event occurrence process
- Discharge process

Recharge Process

The recharge process generates recharge events and recharges the sensor. The recharge event occurs with probability "q." The recharge event occurs and recharges the sensor with a

charge of "c" quanta. Random numbers can be generated by random number generator function (drand48) in C programming language.

Let $X \approx UNIF(0,1)$, then

$$P_r[X \leq q] = q \tag{4.2}$$

A recharge event occurs when the random number generated using the random number generated function is less than or equal to an event occurrence probability, "q." Any charge in excess of "K" is discarded.

Event Occurrence Process

The event occurrence process generates events of interest, which have some temporal correlation across their occurrences. This correlation can be described by using correlation probabilities "p_c^{on},", " p_c^{off}." An event will occur in the following cases:

Let $X \approx UNIF(0,1)$. If the event process state is on, then the event occurs with probability of p_c^{on} i.e. if $X \leq p_c^{on}$. If the event process state is off, then the event occurs with probability of $(1- p_c^{off})$ i.e. if $X \leq 1- p_c^{off}$.

Discharge Process

Whenever an event occurs during a particular time slot and if the sensor is active, then the sensor detects that event and discharges an amount of its energy "$\delta_1 + \delta_2$." The activation policy makes the decision for the sensor to be activate or to be asleep. Activation algorithms for the Aggressive Wakeup (AW) and Correlation-dependent Wakeup policies [4] are described in sections 4.2.4 and 4.2.5. Algorithms for online activation algorithms are explained in Chapter 5.

4.2.4 Algorithm for Aggressive Wakeup Policy

If the rechargeable sensor node has sufficient energy to operate, i.e., the energy level of the sensor is greater than or equal to "$\delta_1 + \delta_2$," then the sensor activates itself. If the sensor node

does not have sufficient energy, i.e., the energy level is less than "$\delta_1 + \delta_2$," then the sensor will be in a dead state, and the sensor will not detect any interesting events. The sensor activates itself whenever possible and ignores the temporally correlated nature of event process.

4.2.5 Algorithm for Correlation-Dependent Wakeup Policy

The CW algorithm satisfies the following criteria. Active sensor with sufficient energy remains active if an event occurred during the previous time slot. The active sensor goes to sleep (for an appropriately derived sleep interval) if no event has occurred during the previous time slot. The sensor activates itself at the end of the sleep interval, if it has sufficient energy ($\delta_1 + \delta_2$). An energy-balancing CW algorithm corresponds to the sleep interval, which allows the sensor to spend as much energy as is gained through recharge in the long run. The optimal performance of CW activation policy is denoted as "U_{cw}."

4.3 Summary

This chapter discussed the problem statement and problem formulation. An explanation was provided for how to perform discrete event simulations of a sensor system and algorithms for the Aggressive Wakeup activation policy and the Correlation-dependent Wakeup policy.

CHAPTER 5

DESIGN OF ADAPTIVE ACTIVATION ALGORITHMS

5.1 Adaptive Activation Algorithms

5.1.1 Introduction

Sensors are deployed for detecting interesting events. A sensor needs an activation policy for detecting events. A sensor may not know about the state of the event occurrence process, i.e. it does not know when the interesting events will occur or about the recharge/discharge parameters such as q, c, δ_1, δ_2, p_c^{on}, and p_c^{off}. For these reasons, the activation algorithm should be adaptive in nature and should perform near optimal without the knowledge of system parameters and a state–of-event occurrence process. This chapter explains the design of different adaptive activation algorithms for the sensor node.

5.2 Activation Algorithms for Renewable Energy-Based Sensor System

This chapter describes the intuition behind the design of a class of adaptive algorithms for sensor activation in a renewable energy-based sensor system. Here, the performances of the algorithms in this class are compared with that of the Energy Balancing Correlation-dependent Wakeup (EB-CW) policy, which has been shown to have near-optimal performance [4].

For the sensor to employ the EB-CW algorithm, it must have the knowledge of all global system parameters. In the EB-CW algorithm, the sleep interval is derived using energy balance during a renewal interval of sensor operation [4], given by

$$SI_{EB-CW} \approx \frac{\delta_1}{qc} + \frac{(1-p_c^{off})(\delta_1+\delta_2-qc)}{qc(2-p_c^{on}-p_c^{off})(1-p_c^{on})} - 1 \qquad (5.1)$$

In practice, it may not be possible for the sensor to know or estimate the value of these system parameters. Therefore, a class of adaptive and online algorithms that do not depend on the knowledge of these system parameters and are able to dynamically converge to the behavior of the EB-CW algorithm, are proposed, thus achieving near-optimal performance.

This class of activation algorithms, denoted Π (c_1, c_2), satisfies all of the criteria of the CW policy, as described in section 4.2.5. However, each time the sensor goes to sleep, it calculates its new sleep interval based on the sleep interval previously used and its current energy level. If the sensor's current energy level is $< K/2$, it acts conservatively by choosing its subsequent sleep interval to be larger than its previously employed sleep interval. Otherwise, the sensor acts aggressively by choosing its subsequent sleep interval to be smaller than its previously employed sleep interval. Only linear (additive or multiplicative) increase and decrease algorithms for subsequent sleep interval computation are considered.

Let SI_{prev} denote the sleep interval last employed by the sensor, and let SI_{next} denote the sleep interval to be employed the next time the sensor decides to go to sleep. If the sensor's current energy level is $L < K/2$, then it increases its sleep interval as follows:

- Additive Increase: $SI_{next} = SI_{prev} + c_1$, or
- Multiplicative Increase: $SI_{next} = SI_{prev} * c_1$

If the sensor's current energy level is $L \geq K/2$, then the sensor decreases its sleep interval as follows:

- Additive Decrease: $SI_{next} = SI_{prev} - c_2$, or

- Multiplicative Decrease: $SI_{next} = SI_{prev} * \frac{1}{c_2}$

Considering all possible combinations of the increase and decrease functions results in the following algorithms:

- Additive Increase Multiplicative Decrease (AIMD)
- Additive Increase Additive Decrease (AIAD)
- Multiplicative Increase Multiplicative Decrease (MIMD)
- Multiplicative Increase Additive Decrease (MIAD)

The additive parameter could take any value greater than 0, while the multiplicative parameter could take any value greater than 1. The additive (increase or decrease) parameter of 1 and multiplicative parameter of 2 are considered when describing the algorithms. However, other choices of parameters are also feasible.

5.2.1 Additive Increase Multiplicative Decrease Activation Algorithm

This activation policy has the same structure described in section 4.2.5, but the algorithm for the sleep interval is as follows: If the energy level of the rechargeable sensor node is less than half of "K," then the sleep interval value is incremented additively (incremented by 1):

Input: SI_{prev}, L, K

Output: SI_{next}

if L < K/2 **then**

$SI_{next} = SI_{prev} + 1;$

else

$SI_{next} = SI_{prev}/2;$

end if

Algorithm 1: Adaptive Sleep Interval Computation for AIMD Algorithm

If the energy level of the rechargeable sensor node is greater than (or) equal to half of "K," then the sleep interval value is decremented multiplicatively (divided by half) the next time the sensor goes to sleep. The choice of K/2 is arbitrary, and other similar choices are also possible.

5.2.2 Additive Increase Additive Decrease Activation Algorithm

This activation policy has the same structure as described in section 4.2.5, but the algorithm for the sleep interval is as follows: If the energy level of the rechargeable sensor node is less than half of "K," then the sleep interval value is incremented additively (incremented by 1) the next time the sensor goes to sleep.

Input: SI_{prev} , L, K

Output: SI_{next}

if L **<** K/2 **then**

$SI_{next} = SI_{prev} + 1;$

else

$SI_{next} = SI_{prev} - 1;$

end if

Algorithm 2: Adaptive Sleep Interval Computation for AIAD Algorithm

If the energy level of the sensor node is greater than (or) equal to half of "K," then the sleep interval value is decremented additively (decremented by 1) the next time the sensor goes to sleep.

5.2.3 Multiplicative Increase Additive Decrease Activation Algorithm

This activation policy has same structure as described in section 4.2.5, but the algorithm for the sleep interval is as follows: If the energy level of the rechargeable sensor node is less than half of "K," then the sleep interval value should be increased multiplicatively (multiplied by factor of 2) the next time the sensor goes to sleep.

Input: SI_{prev}, L, K

Output: SI_{next}

if L **<** K/2 **then**

$SI_{next} = SI_{prev} * 2;$

else

$SI_{next} = SI_{prev} - 1;$

end if

Algorithm 3: Adaptive Sleep Interval Computation for MIAD Algorithm

If the energy level of the rechargeable sensor node is less than half of "K," then the sleep interval value is decremented additively (decremented by 1) the next time the sensor goes to sleep.

5.2.4 Multiplicative Increase Multiplicative Decrease Activation Algorithm

This activation policy has same structure as described in section 4.2.5, but the algorithm for the sleep interval is as follows: If the energy level of the rechargeable sensor node is less than half of "K," then the sleep interval value is incremented multiplicatively (multiplied by factor 2) the next time the sensor goes to sleep.

Input: SI_{prev}, L, K

Output: SI_{next}

if L **<** K/2 **then**

$SI_{next} = SI_{prev} * 2$;

else

$SI_{next} = SI_{prev}/2$;

end if

Algorithm 4: Adaptive Sleep Interval Computation for MIMD Algorithm

If the energy level of the sensor node is less than (or) equal to half of "K," then the sleep interval value is decremented multiplicatively (divided by factor 2) the next time the sensor goes to sleep.

5.3 Hybrid Activation Algorithms

Other feasible hybrid algorithms include the Multiplicative Additive Increase Multiplicative Decrease (MAIMD), wherein the increase of the sleep interval is multiplicative if the sensor's current energy level **<** K/4 (say) and is additive if the sensor's current energy level

lies in the range of [K/4 . . . K/2]. Similarly, the Multiplicative Additive Increase Multiplicative Additive Decrease (MAIMAD) employs multiplicative as well as additive increase and decrease functionalities, with multiplicative decrease employed if the sensor's current energy level lies in the range of [3K/4 . . . K].

5.3.1 Multiplicative/Additive Increase Multiplicative Decrease Activation Algorithm

The MAIMD policy has the same structure as described in section 4.2.5, but the algorithm for the sleep interval is as follows: If the energy level of the rechargeable sensor node is less than 1/4 of "K," then the sleep interval value is incremented multiplicatively (multiplied by factor 2) the next time the sensor goes to sleep.

Input: SI_{prev}, L, K

Output: SI_{next}

if L < K/4 **then**

$SI_{next} = SI_{prev} * 2$;

else

if L < K/2 && L ≥ K/4 **then**

$SI_{next} = SI_{prev} + 1$;

else

$SI_{next} = SI_{prev}/2$;

end if

end if

Algorithm 5: Adaptive Sleep Interval Computation for MAIMD Algorithm

If the energy level of the rechargeable sensor node is less than half of "K" and greater than (or) equal to 1/4 of "K," then the sleep interval value is incremented additively (incremented by factor 1) the next time the sensor goes to sleep.

If the energy level of the rechargeable sensor node is greater than (or) equal to half of "K," then the sleep interval value is decremented multiplicatively (divided by factor 2) the next time the sensor goes to sleep.

5.3.2 Multiplicative/Additive Increase Multiplicative/Additive Decrease Activation Algorithm

This activation policy has same structure as described in section 4.2.5, but the algorithm for the sleep interval is as follows: If the energy level of the rechargeable sensor node is less than 1/4 of "K," then the sleep interval value is incremented multiplicatively (multiplied by factor 2) the next time the sensor goes to sleep.

If the energy level of the rechargeable sensor node is less than half of "K" and greater than (or) equal to 1/4 of "K," then the sleep interval value is incremented additively (incremented by 1) the next time the sensor goes to sleep.

Input: SI_{prev}, L, K

Output: SI_{next}

if L **<** K/4 **then**

$SI_{next} = SI_{prev} * 2$;

else

if $L < K/2 \;\&\&\; L \geq K/4$ **then**

$SI_{next} = SI_{prev} + 1;$

else

if $L < 3K/4 \;\&\&\; L \geq K/2$ **then**

$SI_{next} = SI_{prev} - 1;$

else

$SI_{next} = SI_{prev}/2;$

end if

end if

end if

Algorithm 6: Adaptive Sleep Interval Computation for MAIMAD Algorithm

If the energy level of the rechargeable sensor node is less than 3/4 of "K" and greater than (or) equal to half of "K," then the sleep interval value is decremented additively (decremented by 1) the next time the sensor goes to sleep.

If the energy level of the rechargeable sensor node is greater than (or) equal to 3/4 of "K," then the sleep interval value is decremented multiplicatively (divided by 2) the next time the sensor goes to sleep.

5.4 Advantages of Dynamic Sleeping in Sensor Systems

The advantages of dynamic sleeping in sensor systems are as follows:

- The sensor system can still achieve performance close to optimal.
- This mechanism can be implementable in practice.

- The mechanism is adaptive in nature. Since the online algorithms dynamically vary the sleep interval values with respect to varying energy levels of the sensor system, these algorithm adapt well to dynamic changes in system parameters.

5.5 Comparison with AIMD Mechanism in TCP

Additive increase and multiplicative decrease is common in both the TCP and in sensor systems. The TCP varies the congestion window according to the AIMD mechanism, and in online activation algorithms, the sensor varies its sleep interval using the AIMD mechanism. In the TCP, the level of congestion in the network is uncertain; similarly in the sensor system, the state of the event process and system parameter values are uncertain.

5.6 Summary

In this chapter, algorithms for online activation of a sensor were developed. An explanation was provided on how the TCP congestion window varying mechanism is applicable in developing these algorithms. The advantages of dynamic sleeping in sensor systems and similarities and differences of using dynamic and adaptive mechanism such as AIMD in both TCP-based and sensor systems were also discussed.

CHAPTER 6

SIMULATION RESULTS

6.1 Simulations of Activation Algorithms

6.1.1 Introduction

The code describing the rechargeable sensor node with different activation policies was written using C programming language, as shown in Appendix B. Chapter 6 describes simulations results obtained using discrete event simulation of the sensor system for different activation policies. Simulations were performed for different system parameters with different activation policies used for the rechargeable sensor. Section 6.1.2 shows the simulation of the Energy Balancing Correlation-dependent Wakeup policy, and a comparison of the theoretical and DES-generated optimal sleep intervals. Section 6.2 shows the simulation of the sensor code and compares the performance of different online activation algorithms to existing activation policies. The energy levels and sleep interval variations of different activation algorithms are also compared in section 6.2.

Section 6.3 explains the simulation of the online activation algorithm (AIMD activation algorithm). The increment and decrement factors of the AIMD activation algorithm were varied and the performance was observed. In this section, the stability region of the CW activation algorithm is plotted and compared to the AIMD activation algorithm.

The system parameters used were $q = 0.5$, $c = 1$, $\delta_1 = 1$, $\delta_2 = 6$, $K = 3000$. The initial energy level of the sensor was assumed to be zero. In all the experiments, the values of increase/decrease parameters were chosen so that the additive parameter equals 1 and the multiplicative parameter equals 2, and the correlation probabilities are given by $p_c^{on} = 0.7$ and

p_c^{off} = 0.9, unless stated otherwise. Similar performance trends were observed with other choices of system parameters as well.

6.1.2 Simulations of Correlation-Dependent Wakeup Policy

Simulations were performed for the EB-CW activation policy. Theoretical and DES-generated sleep intervals were compared. Figure 10 depicts the sleep intervals of the EB-CW activation policy for varying correlation probability, "p_c^{on}." Theoretical values are the values of sleep intervals from the sleep interval equation (equation [5.1]) described in section 5.2. DES outputs are the optimal sleep interval values obtained from the simulation. In this simulation, it was assumed that the correlation probability, "p_c^{off}," is 0.9. From Figure 10, it can be seen that EB-CW is indeed near optimal, as claimed by Jaggi et al. [4].

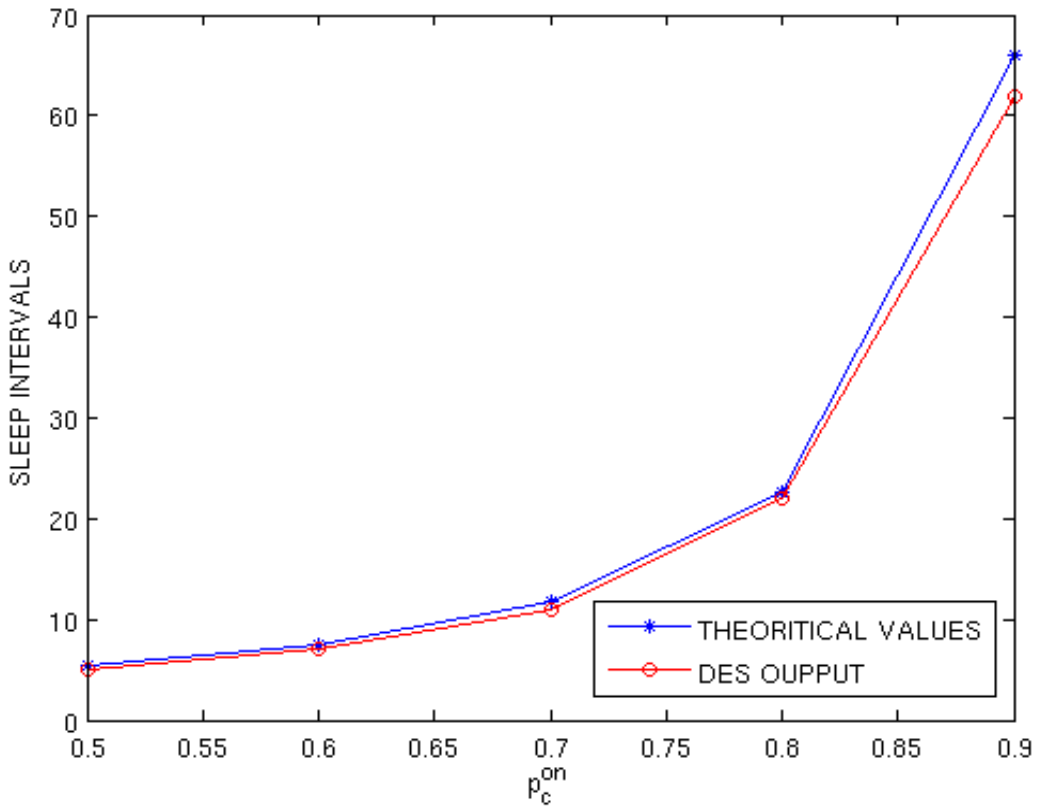


Figure 10. Comparison of theoretical and DES-generated optimal sleep intervals w.r.t "p_c^{on}".

Figure 11 depicts the sleep intervals of the EB-CW activation policy for the varying correlation probability, "p_c^{off}". Theoretical values are the values of sleep intervals from the sleep interval equation (equation [5.1]) described in section 5.2. DES outputs are the optimal sleep interval values obtained by the simulating sensor code.

In this scenario, it is assumed that correlation probability, p_c^{on}, is 0.9, i.e., ON periods are longer than OFF periods. From Figure 11, it can be seen that the theoretical EB-CW sleep interval values are different from that of simulation-generated optimal sleep interval values. Thus, the EB-CW sleep interval is close to, but not, optimal.

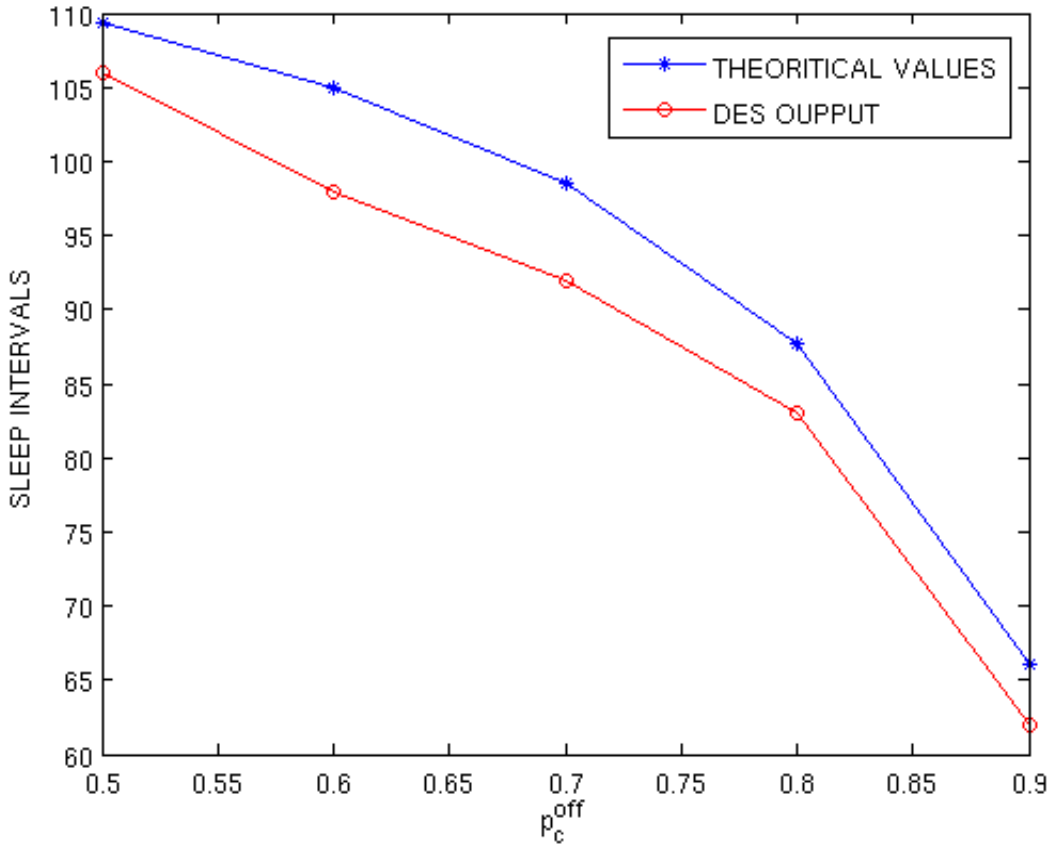


Figure 11. Comparison of theoretical and DES-generated optimal Sleep intervals w.r.t "p_c^{off}".

6.2 Simulations of Different Activation Algorithms

6.2.1 Performances of Different Online Activation Algorithms

Performance was computed using equation (4.1). Figure 12 plots the steady-state performance of the four types of algorithms outlined in section 5.2, with $p_c^{off} = 0.9$. As p_c^{on} increases from 0.5 to 0.9, more events occur during the interval [0 . . . T], and since the recharge system parameters are kept constant, the performance in terms of fraction of events detected decreases for all algorithms. It can be observed that the AIMD algorithm achieves the best performance among these algorithms, closely followed by AIAD and MIMD, whereas MIAD achieves the worst performance. Even though AIAD and MIMD perform well, the sensor behavior has a more stable w.r.t. energy level and sleep interval variations under AIMD.

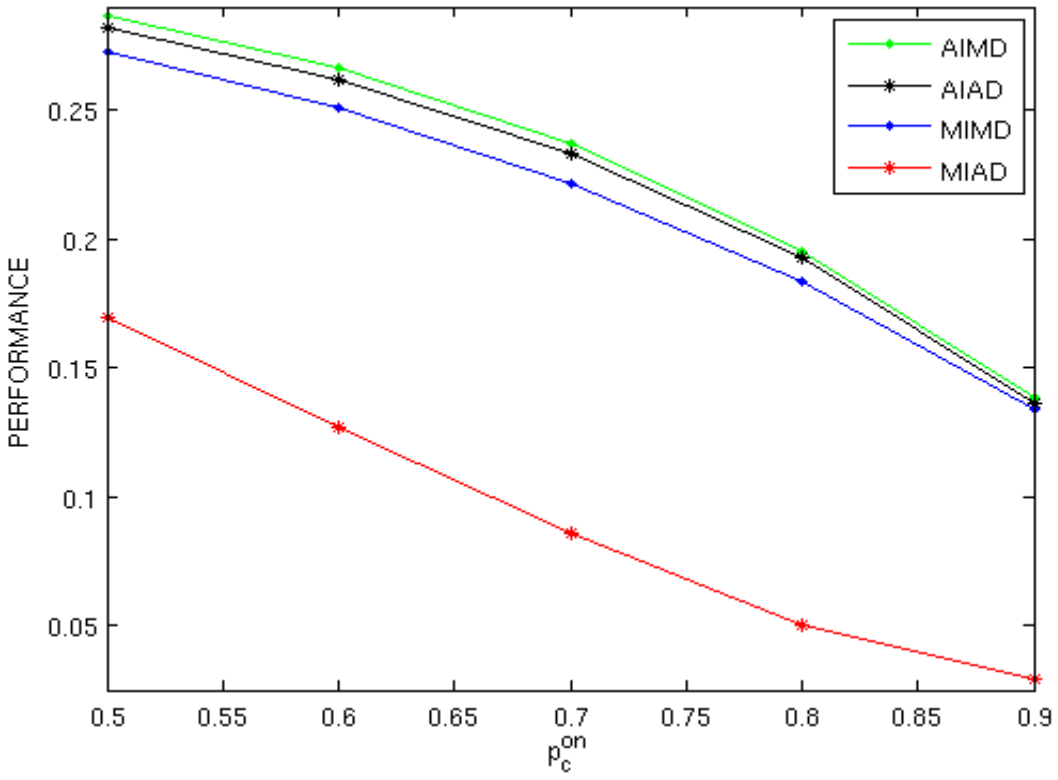


Figure 12. Performance comparison of different adaptive algorithms w.r.t "p_c^{on}."

Figure 13 plots the steady-state performance of the four types of algorithms outlined in section 5.2, with p_c^{on} = 0.9. As p_c^{off} increases from 0.5 to 0.9, fewer events occur during the interval [0 . . . T], and since the recharge system parameters are kept constant, the performance increases with "p_c^{off}". It can be observed that the AIMD algorithm achieves the best performance among these algorithms, closely followed by AIAD and MIMD, whereas MIAD achieves the worst performance.

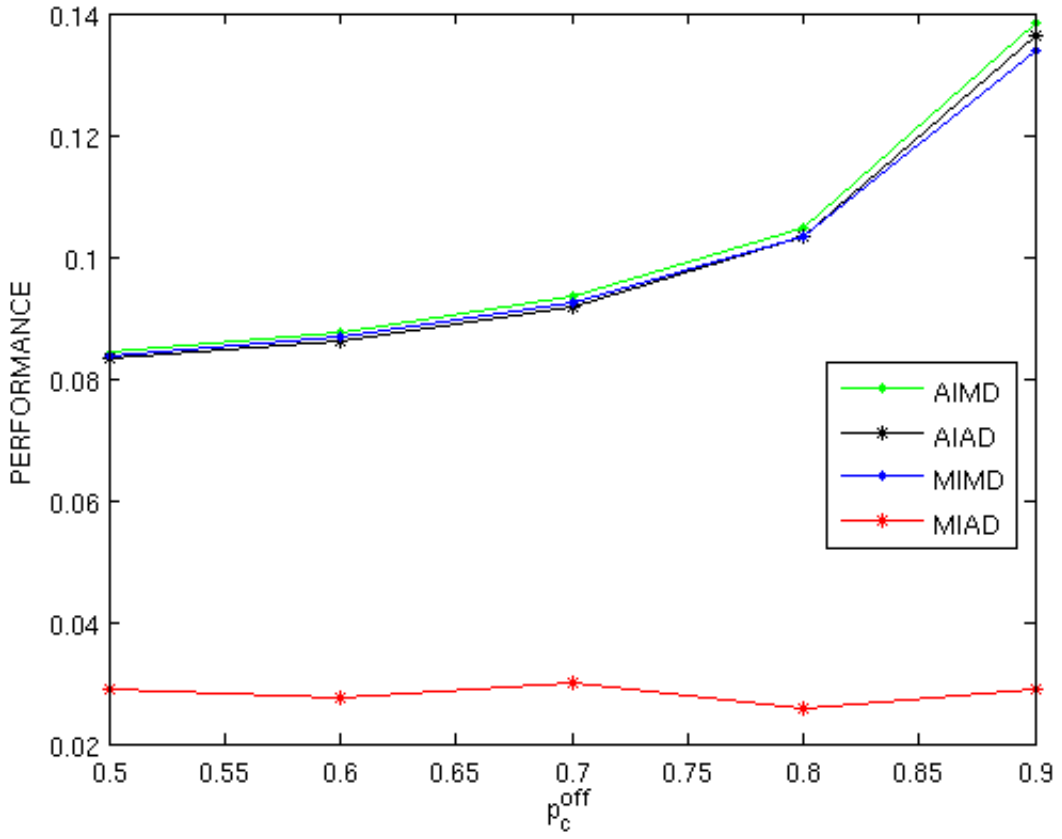


Figure 13. Performance comparison of different adaptive algorithms w.r.t "p_c^{off}."

MIAD algorithm performance is much lower than that of the other algorithms. Since the initial energy level of the sensor is zero, and it takes some time for it to reach above K/2, the sensor operating under the MIAD algorithm increases its sleep interval to a very large value

using a multiplicative increase of two each time. Even when the energy level becomes greater than the above threshold, the sleep interval is decremented slowly (additively by one each time). Therefore, the sensor spends most of its time in the sleep state and detects a lower fraction of events. On the other hand, AIMD is able to perform better since the sleep interval of the sensor operating under AIMD converges to that of EB-CW faster, as will be shown in section 6.2.4.

6.2.2 Comparing Performances of Different Activation Algorithms

This section compares the performances of different online activation algorithms with the AW and EB-CW algorithms. The performance of hybrid algorithms is also compared. Figure 14 compares the performance of some of the hybrid algorithms with that of AIMD.

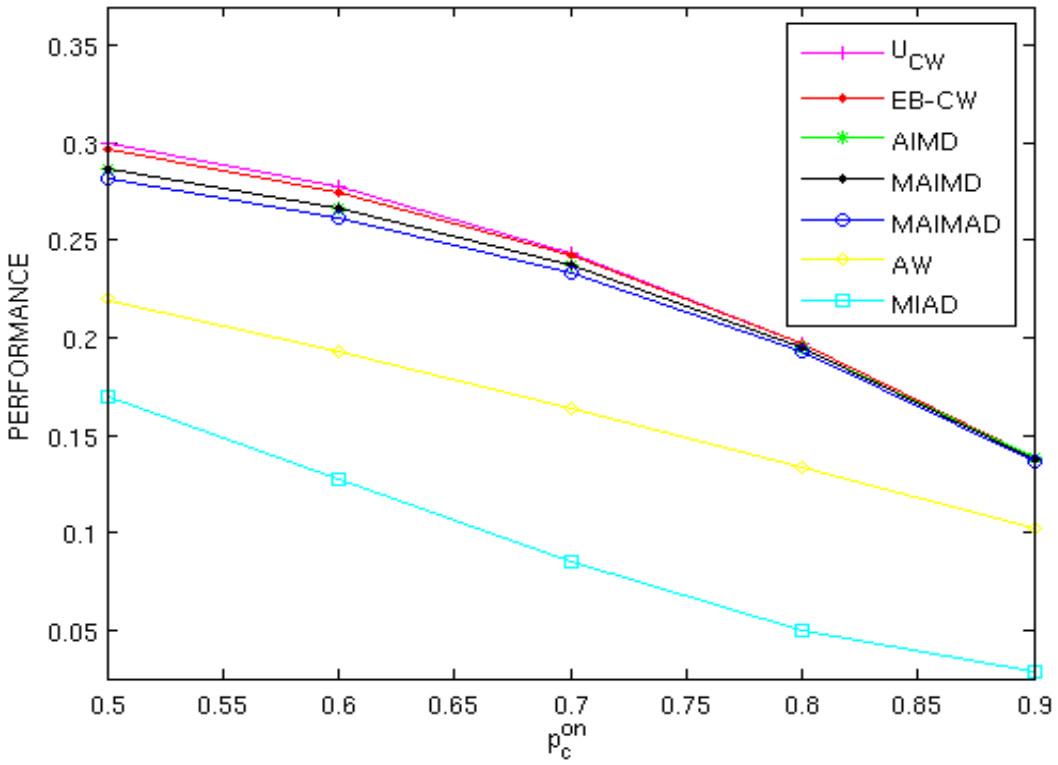


Figure 14. Performance comparison of AIMD with other hybrids, AW and EB-CW algorithms w.r.t "p_c^{on}."

The optimal performance of the CW activation policy is denoted as "U_{cw}." U_{cw} and the performance of the EB-CW and AW algorithms are also plotted for comparison. The correlation probability, p_c^{off}, is 0.9, and the correlation probability, p_c^{on}, is varied from 0.5 to 0.9. Figure 14 shows that the AIMD algorithm performs very well compared to AW, and it performs quite closely to EB-CW. AIMD, AIAD, MIMD, MAIMD, and MAIMAD perform very closely to each other, when $p_c^{on} = 0.9$. Figure 15 depicts the performances of different rechargeable sensor activation policies when $p_c^{on} = 0.9$ and p_c^{off} is varied from 0.5 to 0.9. It can be observed that all the activation policies except AW and MIAD activation policies perform near optimal. MIAD performs worse compared with other activation policies.

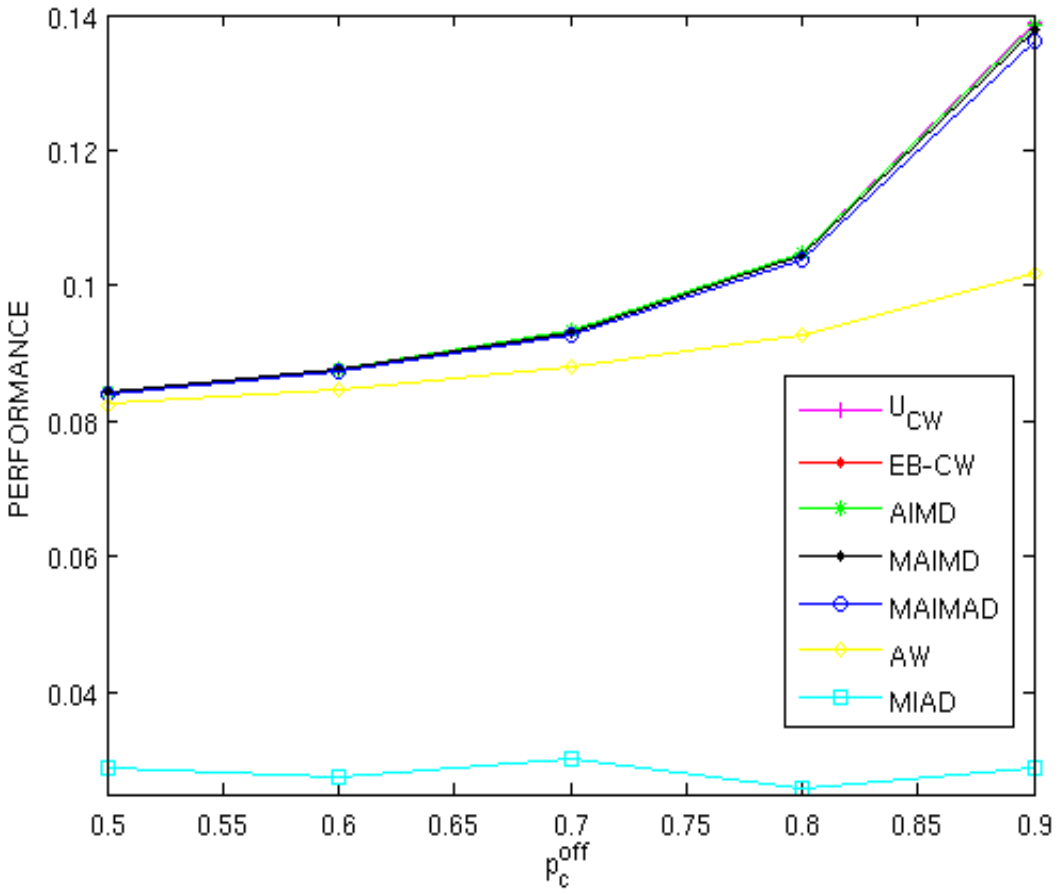


Figure 15. Performance comparison of AIMD with other hybrids, AW and EB-CW algorithms w.r.t "p_c^{off}."

6.2.3 Energy Levels of Different Activation Policies

Figure 16 plots the energy level of the sensor operating under various activation algorithms. Here p_c^{off} is 0.9, and p_c^{on} is 0.7. "$L_{EB\text{-}CW}$" in the plot describes the energy levels of the sensor under EB-CW activation policy for best sleep interval value (SI = 11). $SI_{EB\text{-}CW}$ (computed using equation [5.1]) is around 11 for the chosen set of system parameters.

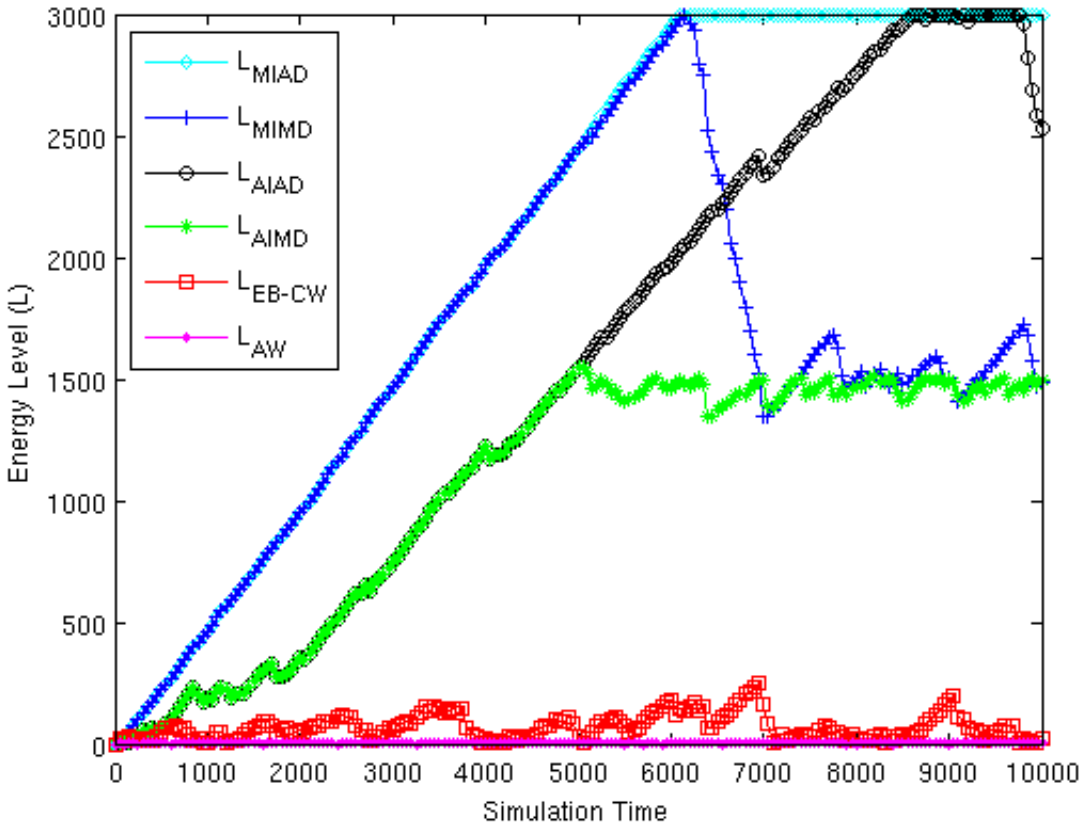


Figure 16. Energy level evolution for different adaptive activation algorithms.

The energy level of the sensor under the AIMD algorithm converges to around half of the energy bucket size, i.e., in steady-state $L_{AIMD} \rightarrow K/2$. The energy level of the sensor operating under algorithms AIAD and MIMD also converge close to K/2 , albeit after a longer time and with larger oscillations. This behavior is a direct consequence of the fact that all of the adaptive algorithms increase the sensor's sleep interval when $L < K/2$ and decrease it when $L \geq K/2$.

Since the sensor's energy level stabilizes close to K/2 quite rapidly operating under the AIMD algorithm, the use of hybrid algorithms like MAIMD and MAIMAD does not provide additional performance improvement over that of AIMD (as the sensor's energy level never reaches close to 0 or K).

Figure 17 depicts the sensor's energy level in steady-state for AIMD and EB-CW activation algorithms.

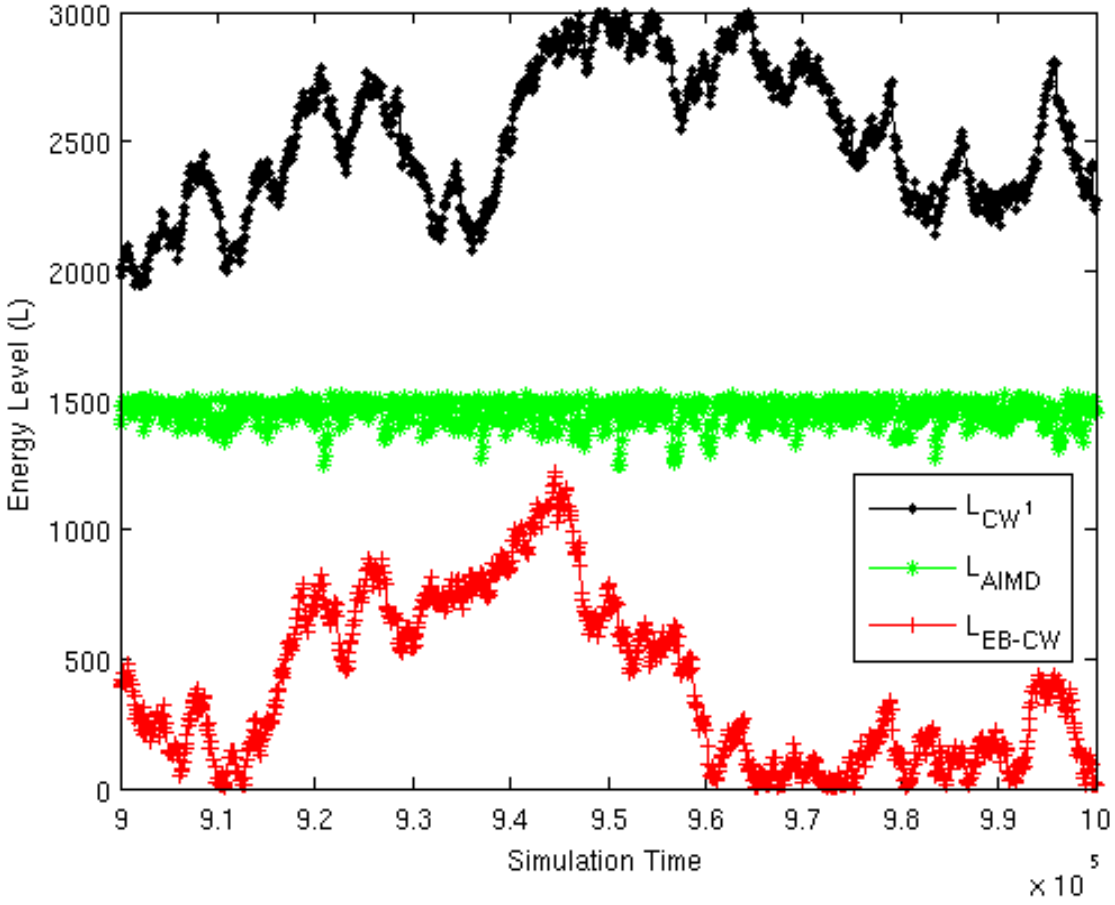


Figure 17. Energy level under AIMD and CW algorithms in steady-state.

"$L_{EB\text{-}CW}$" in the plot describes the energy levels of the EB-CW activation policy for the best sleep interval value (SI = 11). "L_{CW}^{1}" in the plot describes the energy levels of the CW

activation policy for sleep interval (SI = 12). Here can be seen the energy level of the sensor for two different CW algorithms, one with SI = 11 (EB-CW) and another with SI = 12 (denoted CW^1). The sensor operating under the AIMD activation algorithm maintains its energy level close to K/2, whereas under the CW algorithms, the energy level of the sensor either stays close to full (or) close to empty, and this algorithm also exhibits larger variations.

This suggests that the AIMD algorithm is able to stabilize the sensor's energy level more than the CW algorithm. Note that a probabilistic scheme could be used with the CW algorithm to choose the sleep interval 11 w.p. p and 12 w.p. 1 − p, so as to stabilize the sensor's energy level. However, such a scheme would still exhibit larger variations in energy level and would also depend heavily on global system parameters.

6.2.4 Sleep Intervals of Different Activation Policies

Figure 18 plots the sensor's sleep interval over time, operating under various activation algorithms. Here, p_c^{off} is 0.9, and p_c^{on} is 0.7. $SI_{EB\text{-}CW}$ (computed using equation [5.1]) is around 11 for the chosen set of system parameters.

Figure 18 shows that the sleep interval values under various adaptive algorithms oscillate around (and try to converge to) $SI_{EB\text{-}CW,}$ except for MIAD, where the sleep interval diverges to a very large value. Among these, the AIMD exhibits the fastest and more stable convergence. The sleep interval employed by the sensor operating under the AIMD activation algorithm oscillates around $SI_{EB\text{-}CW}$ in steady state, as shown in Figure 19. Note that the AIMD algorithm is able to estimate the value of $SI_{EB\text{-}CW}$ automatically, by dynamically adapting the sleep interval in such a way so as to keep the sensor's energy level close to K/2. As a result, the sensor's behavior operating under the AIMD algorithm approaches the behavior under the EB-CW, resulting in near-optimal performance achieved by the AIMD algorithm.

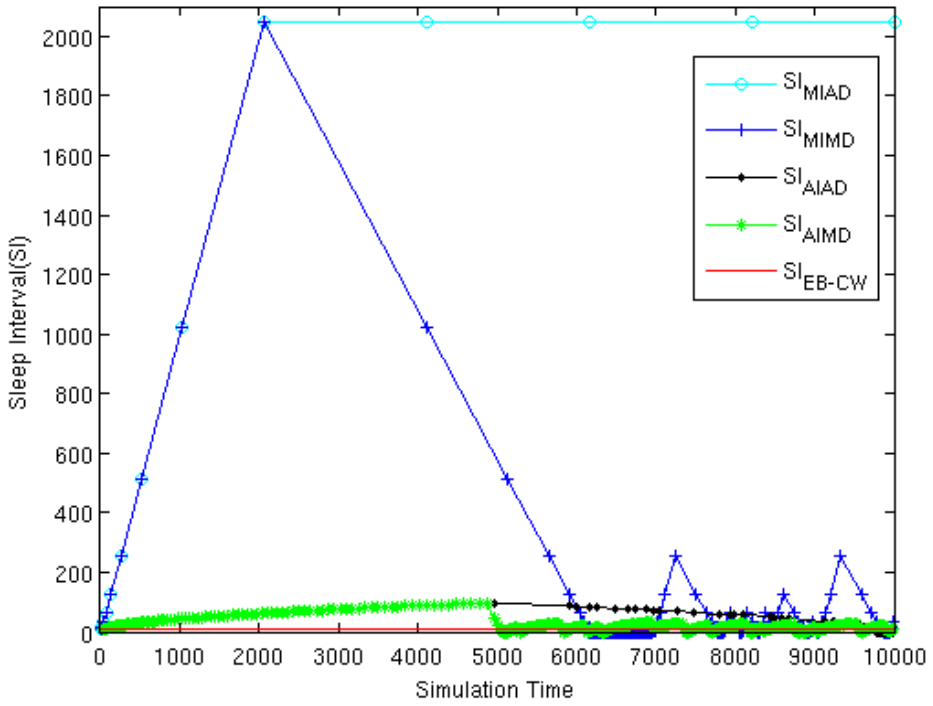


Figure 18. Sleep interval evolution for different adaptive activation algorithms.

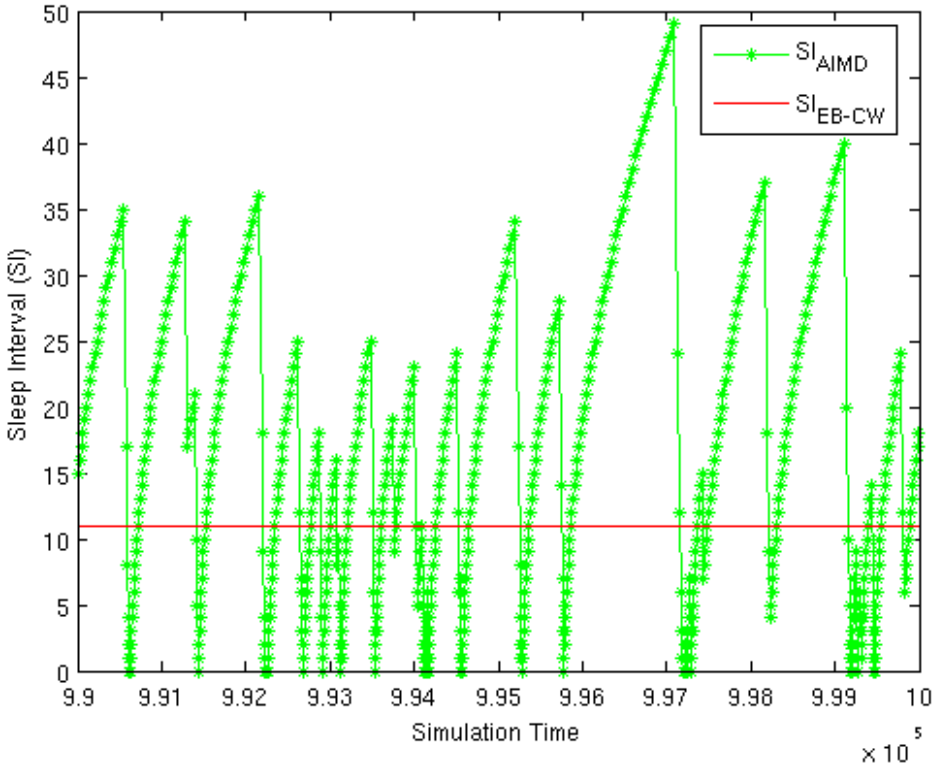


Figure 19. Sleep interval oscillations with AIMD algorithm.

Figure 19 depicts the sleep interval of the energy-balanced Correlation-dependent Wakeup and AIMD activation algorithm. The sleep interval of the EB-CW activation algorithm is a constant value and calculated using equation (5.1). Sleep interval values of the AIMD activation algorithm oscillate around the EB-CW sleep interval ($SI_{EB\text{-}CW}$).

6.3 Simulations of Increase/Decrease Activation Algorithms

6.3.1 Performances of Activation Algorithms for Varying Increase, Decrease Parameters

Simulations were performed for different activation algorithms, and their performances were observed. The term "c_1" defines the increment parameter, and "c_2" defines the decrement parameter for all the policies. Here, $p_c^{on} = 0.7$, and $p_c^{off} = 0.9$.

Figure 20 depicts the performance of the additive increase multiplicative decrease activation algorithm. The performance is observed for values of $c_1 \in [1 \ldots 10]$, and $c_2 = 2^i \ \forall\, i \in [1 \ldots 10]$. From Figure 20, it can be observed that the performance is maximum at $c_1 = 1$, $c_2 = 2$.

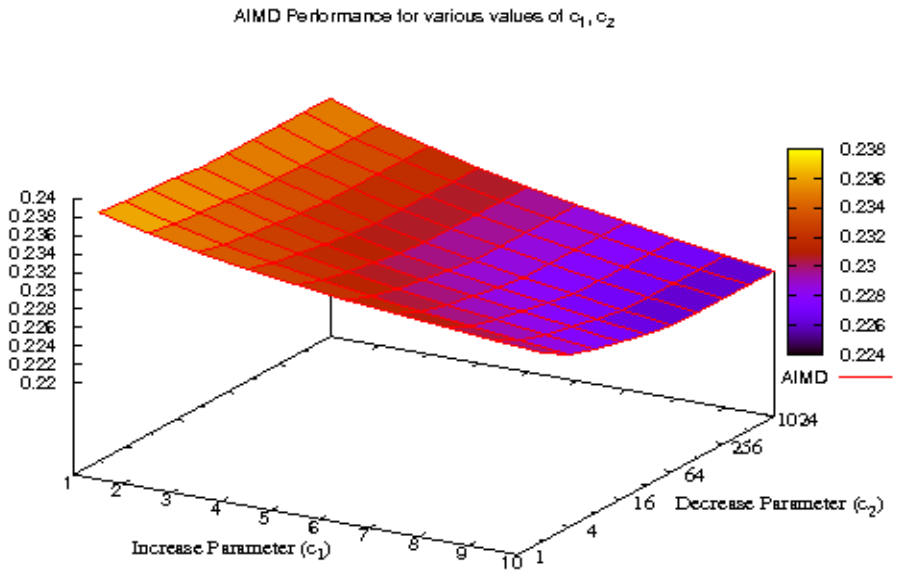


Figure 20. AIMD performance for various values of c_1, c_2.

Figure 21 depicts the performance of the additive increase additive decrease activation policy. The performance is observed for values of $c_1 \in [1 \ldots 10]$ and $c_2 \in [1 \ldots 10]$. From Figure 21, it can be observed that the performance is maximum at $c_1 = 1$, $c_2 = 10$.

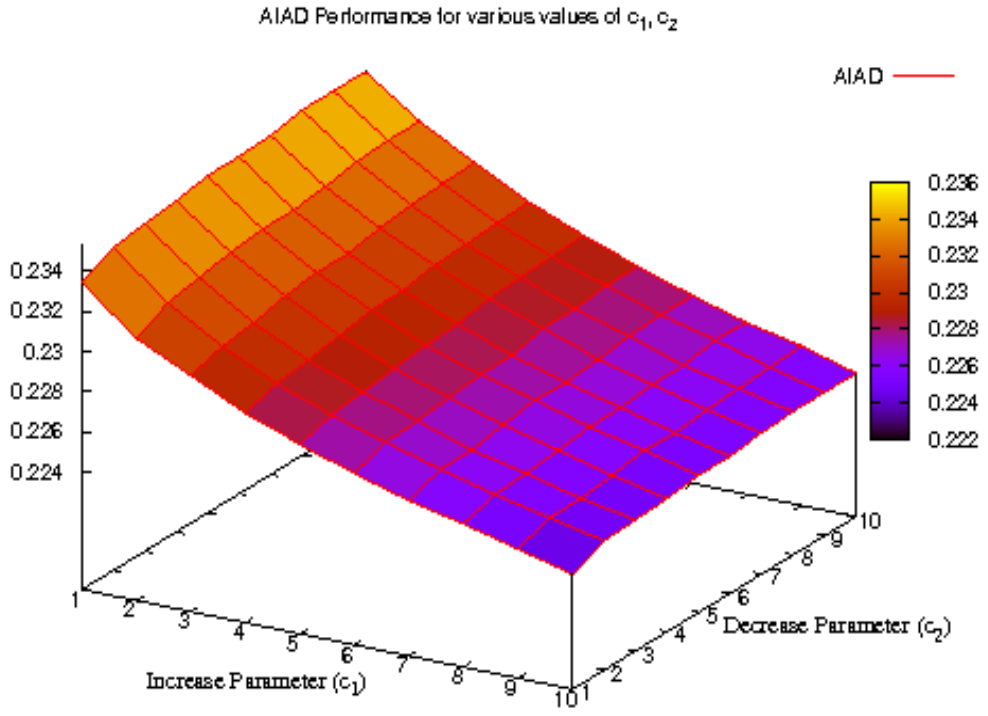


Figure 21. AIAD performance for various values of c_1, c_2.

Figure 22 depicts the performance of multiplicative increase additive decrease activation algorithm. The performance is observed for values of $c_1 = 2^i \ \forall \ i \in [1 \ldots 10]$ and $c_2 \in [1 \ldots 10]$. From Figure 22, it can be observed that the performance is maximum at $c_1 = 2$, $c_2 = 10$.

Figure 23 depicts the performance of multiplicative increase multiplicative decrease activation algorithm. The performance is observed for values of $c_1 = 2^i \ \forall \ i \in [1 \ldots 10]$ and $c_2 = 2^i \ \forall \ i \in [1 \ldots 10]$. From Figure 23, it can be observed that the performance is maximum at $c_1 = 2$, $c_2 = 2$.

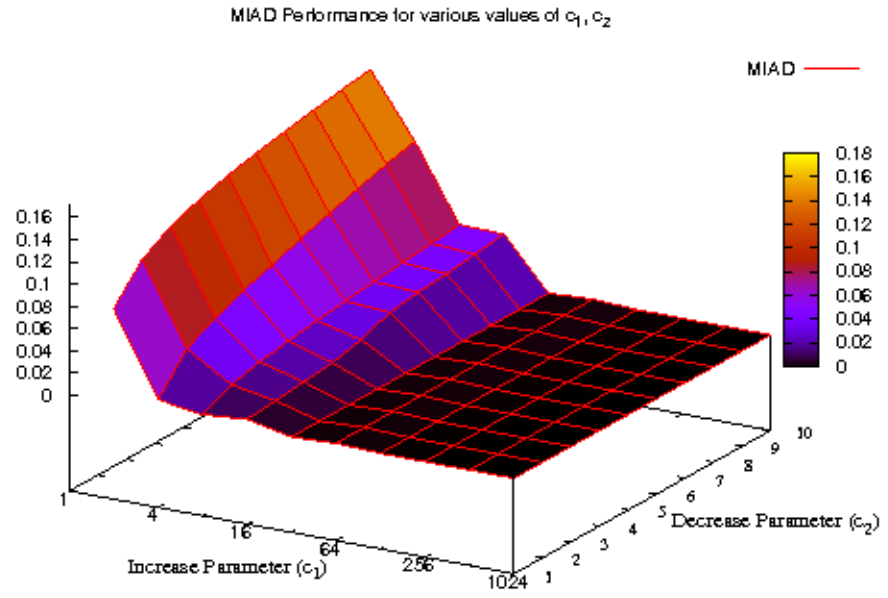


Figure 22. MIAD performance for various values of c_1, c_2.

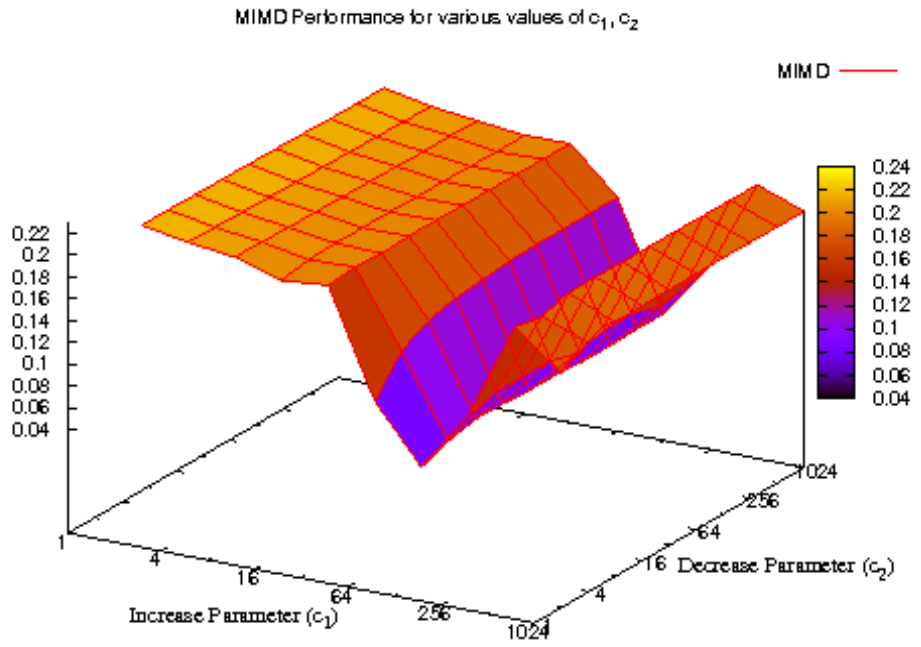


Figure 23. MIMD performance for various values of c_1, c_2.

From Figure 20 to 23, it can be observed that when $c_1 = 1$ and $c_2 = 2$, the best performance among all AIMD algorithms can be achieved. Similar performance trends with respect to c_1 and c_2 can be observed for other adaptive algorithms as well. In particular, the additive (increase or decrease) parameter of 1 and the multiplicative parameter of 2 achieve the best performance, except for algorithms with additive decrease (where $c_2 = 1$ is not the best choice).

Additionally, the AIMD algorithm with $c_1 = 1$ and $c_2 = 2$ is observed to achieve the best performance among all the algorithms in the class Π (c_1, c_2).

6.3.2 Stability Graph of AIMD Activation Policy

Figure 24 depicts the stability of the sensor's behavior, and the performance achieved, operating under the AIMD activation algorithm, in comparison with the CW activation algorithm. The performance of the CW algorithm is shown for different values of sleep intervals. The performance plot of the AIMD activation policy is a straight line, which is a constant value observed at the steady state during the simulation of the AIMD activation policy.

The performance achieved by the AIMD algorithm is quite close to the performance of the best CW algorithm. Figure 24 also depicts the steady-state energy levels (denoted L^{ss}) of the sensor under the AIMD and CW algorithms.

It can be observed that the energy level of the sensor under the AIMD algorithm converges close to K/2, whereas under CW algorithms, the energy level of the sensor is either close to K or close to 0.

"L_{cw}^{ss}" in the plot describes the mean energy levels of Correlation-dependent Wakeup activation policy for varying sleep intervals. "L_{AIMD}" in the plot describes the mean energy level of the AIMD activation policy at steady state. The parallel lines in the plot describe the stability

region for the CW policy and clearly depict that the AIMD performance and energy level lies in that region.

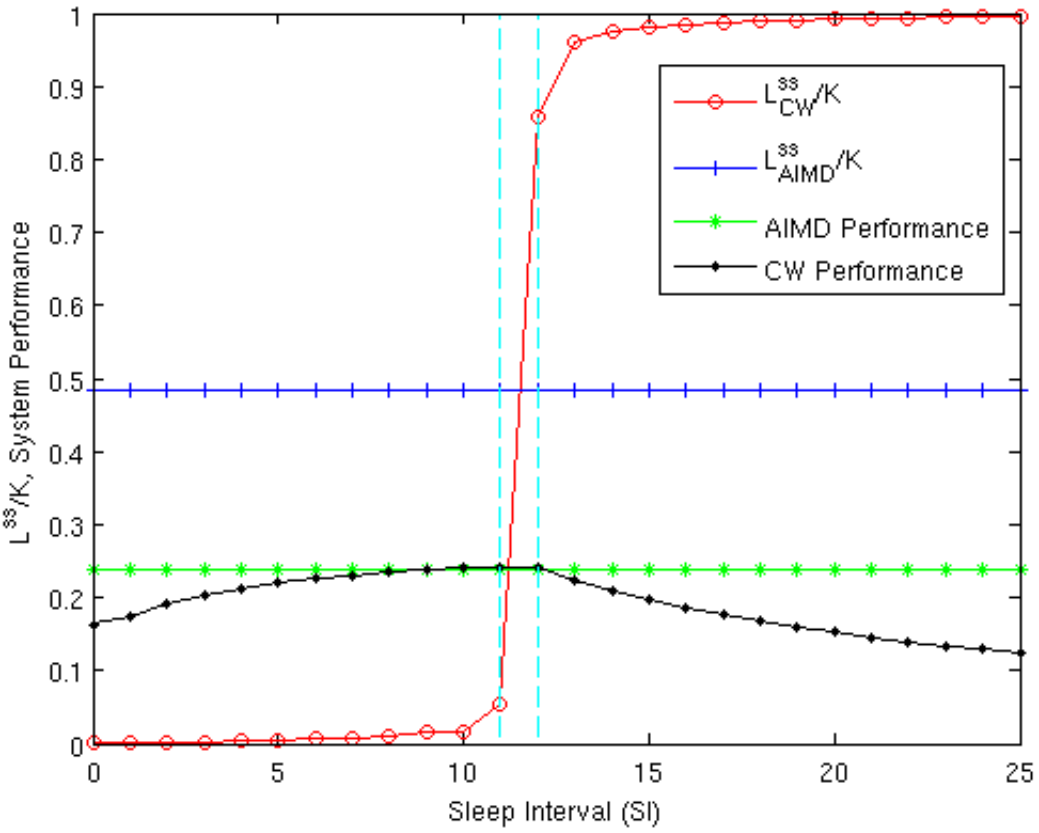


Figure 24. Stability and performance of AIMD activation algorithm.

6.3.3 Comparison of EB-CW and AIMD Algorithms

The AIMD algorithm performs well because it tries to quickly estimate and converge the sensor's behavior to that under the EB-CW algorithm. Thereafter, the AIMD algorithm maintains the sensor's sleep interval value oscillating around $SI_{EB\text{-}CW}$ and its energy level close to K/2. Moreover, the AIMD algorithm is more adaptive to dynamic changes in the environment. For instance, if one of the system parameters changes, the AIMD algorithm would automatically converge to the new high-performance and stability region, whereas the sensor operating under

the EB-CW algorithm would need to estimate the new value of the changed system parameter, and recompute $SI_{EB\text{-}CW}$ in order to achieve good performance. In addition, the AIMD algorithm does not depend on the system parameters (unlike EB-CW) and thus can easily be deployed in practice. The sensor's behavior (with respect to energy level and sleep interval variations) operating under the AIMD algorithm is also more stable compared to the EB-CW and other adaptive algorithms.

6.3.4 Advantages of AIMD Activation Algorithm

Advantages of the AIMD activation algorithm are as follows:

- Is adaptive in nature, dynamically learning the sleep interval values without the knowledge of the recharge/discharge parameters of the sensor system.
- Performs close to optimal.
- Has an energy level that is more stable than that of the Correlation-dependent Wakeup policy.
- Has a sleep interval that converges dynamically and fast to a near-optimal value.

6.4 Summary

In this chapter, the discrete event simulation of sensor system was performed, and the results were observed. Different online activation algorithms were compared, and it was observed that the AIMD activation policy performs the best among all adaptive algorithms. Online activation algorithms with existing activation algorithms were compared, and it was observed that the AIMD activation algorithm performs close to that of the EB-CW activation policy, which has been shown to be near optimal [4]. The energy level and sleep interval variation under different activation policies were discussed. Finally, the stability region operating under CW activation algorithm was discussed. It was observed that the energy level of the AIMD

activation policy is more stable than the CW activation policy, and the performance of the AIMD activation policy is close to that of the EB-CW activation policy. Advantages of the AIMD activation algorithm were also discussed.

CHAPTER 7

CONCLUSION AND FUTURE WORK

A class of online and adaptive activation algorithms was designed for sensor operation in a renewable energy-based sensor system. The performance of the sensor was measured in terms of the asymptotic event-detection probability achieved in the presence of temporally correlated event phenomena. It was shown that an algorithm which increases the sensor's sleep interval conservatively (additive increase) and decreases the sleep interval aggressively (multiplicative decrease) performs close to optimal. In particular, the proposed AIMD algorithm not only achieves near-optimal performance but also maintains a stable energy level for the sensor with a finite energy bucket size. Since the AIMD algorithm does not rely on global system parameters, it is more adaptive to changes in these parameters and is also more suitable for deployment. Also, the sleep interval converges fast to that under the EB-CW [4] policy and oscillates around this near-optimal value.

In the future, it might be interesting to mathematically study the properties of the equilibrium achieved by AIMD and other adaptive algorithms. Another direction of future research might include the distributed implementation of these algorithms in multi-sensor systems.

REFERENCES

REFERENCES

[1] J. Hsu, A. Kansal, J. Friedman, V. Raghunathan, and M. Srivastava, "Energy Harvesting Support for Sensor Network," in Proc. of IEEE IPSN Demo, 2005.

[2] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design Considerations for Solar Energy Harvesting Wireless Embedded Systems," in 4th Intl. Symposium on Information Processing in Sensor Networks (IPSN) - Special Track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS), pp. 457-462, April 2005.

[3] A. Kansal and M. B. Srivastava, "Energy harvesting aware power management (book chapter)," in Wireless Sensor Networks: A Systems Perspective. Norwood, MA: Artech House, July 2005.

[4] N. Jaggi, K. Kar, and A. Krishnamurthy, "Rechargeable Sensor Activation under Temporally Correlated Events," in Wireless Networks, vol.15, no. 5, pp. 619-635, July 2009.

[5] D-M. Chiu, and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks", in Computer Networks and ISDN Systems, vol. 17, Issue 1, pp. 1-14, June 1989.

[6] S. R. Mereddy, N. Jaggi, and R. Pendse, "An Adaptive Algorithm for Sensor Activation in Renewable Energy based Sensor Systems," Accepted for publication in Proc. of 5th International Conference on Intelligent Sensors, Sensor Networks and Information Processing(ISSNIP), December 2009.

[7] T. Banerjee, S. Padhy, and A. A. Kherani, "Optimal Dynamic Activation Policies in Sensor Networks", in Proc. of 2nd International Conference on Communication Systems Software and Middleware (COMSWARE), pp. 1-8, Bangalore, January 2007.

[8] T. Banerjee, and A. A. Kherani, "Sensor Node Activation Policies using Partial or No Information", in Proc. of 5th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), pp. 1-7, Limassol, April 2007.

[9] M. Gatzianas, L. Georgiadis, and L. Tassiulas, "Asymptotically Optimal Policies for Wireless Networks with Rechargeable Batteries," in Proc. Of Wireless Communications and Mobile Computing Conference (IWCMC), pp. 33-38, August 2008.

[10] K. Kar, A. Krishnamurthy, and N. Jaggi, "Dynamic Node Activation in Networks of Rechargeable Sensors," in IEEE/ACM Transactions on Networking, vol. 14, no. 1, pp. 15-26, February 2006.

[11] H. Hu, and Z. Yang, "The study of power control based cooperative opportunistic routing in wireless sensor networks," in International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), pp. 345-348, 2007.

[12] J. V. Gruenen, D. Petrovic, A. Bonivento, J. Rabaey, K. Ramchandran, and A. Sangiovanni-Vincentelli, "Adaptive Sleep Discipline for Energy Conservation and Robustness in Dense Sensor Networks," in IEEE ICC, Vol. 6, pp. 3657-3662, June 2004.

[13] J. Na, S. Lim, and C-K. Kim, "Dual Wake-up Low Power Listening for Duty Cycled Wireless Sensor Networks," in EURASIP Journal on Wireless Communications and Networking, 2008.

[14] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control", Network Working Group: RFC 2581, April 1999.

[15] S. Floyd, M. Handley, and J. Padhye, "A comparison of equation-based and AIMD congestion control" http://www.aciri.org/floyd/papers.html," May 2000.

[16] Figure# 1, 2, 3: http://www.ecs.umass.edu/ece/wolf/courses/ECE697AA/lectures/ECE697AA-08-10-07-Congestion%20Control%20AIMD.pdf [date retrieved Nov 30, 2009].

[17] C. G. Cassandras, and S. Lafortune, Introduction to Discrete Event Systems, Second Edition, Springer Publications, 2008.

APPENDIXES

APPENDIX A

M/M/1 QUEUEING SYSTEM CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Endofsim 100000
#define Lambda 9
#define mu 10

int num_departures;
int Qlength;
double avg_system;   /* Average System Time */
double current_time; /* This stores all the Arrival times*/
double Departure_Time;/* Departure time of last departure num_departures*/
double at_of_next_dep;   /* Contains arrival time of next departure */
double st_of_next_dep;/* Contains service start time of next departure*/
double X;/*Summation of product of queue length and queue length occupacy times*/
double T1;/*Queue Length Occupancy times start time*/
double T2;/*Queue Length Occupancy times end time*/
double time;/*Which represents  event occurrence times(arrivals and departures)*/
double U;/* utilization*/

//Declaration of Event structure
struct Event
 {
 double Arrival_Time;
 struct Event *next;
 } ;

//decleration of Queue structure
// Linked list of arrivals events ONLY
struct Queue
 {
 int num_customers;
 double last_arrival;
 struct Event *head;
 };
struct Queue sys;   /* LOA */

//This Function generates exponential distributed random numbers and returns these values.
double getexp(double mean)
 {
 double x,y;
```

```
 x = drand48();// will generates random number between (0,1]
 y = -(double)(mean*log(1-x));
 return y;
 };

//Initialization of the variables used in the code
int initialization()
 {
 sys.num_customers = 0;
 sys.last_arrival = 0;
 sys.head = NULL;
 srand48(10);// seeds the input to Random Number Generator
 current_time = 0.0;
 num_departures = 0;
 Departure_Time = 0.0;
 at_of_next_dep = 0.0;
 st_of_next_dep = 0.0;
 avg_system = 0.0;
 Qlength = 0;
 X  = 0.0;
 T1 = 0.0;
 T2 = 0.0;
 time = 0.0;
 U = 0.0;
 return 0;
}

// This Function is used dequeue a event from List Of Arrival Events Queue
struct Event *dequeue ()
 {
 struct Event *e;
 struct Event *head = sys.head;
 if (!head)
 return NULL;
 e = head;
 head = head->next;
 sys.head = head;
 return e;
 }

//This Function is used to add an event to List of Arrival Events Queue
int add_event(struct Event *e )
 {
 struct Event *follow;
 struct Event *lead;
 struct Event *head = sys.head;
```

```
 if(!head)
 {
 head = e;
 head->next = NULL;
 sys.head = head;
 return 0;
 }
 if(e->Arrival_Time < head->Arrival_Time)
 {
 e->next = head;
 head = e;
 sys.head = head;
 return 0;
 }
 lead = head;
 follow = NULL;

while(1)
 {
 follow = lead;
 lead = lead -> next;
 if(lead == NULL)
 {
 follow->next = e;
 e->next = NULL;
 break;
 }
 if ((follow->Arrival_Time <= e->Arrival_Time)&&(lead->Arrival_Time > e->Arrival_Time))
 {
        follow->next = e;
        e->next = lead;
        break;
 }
 }
 return 0;
}

// This Function generates the customer arrivals and adds arrival event to List of Arrival Events
Queue
int generate_arrivals()
 {
 double g;
 struct Event *e;
       T1 = sys.last_arrival;
       g = getexp(1.0/ Lambda);
```

```
        sys.last_arrival += g;
        if (sys.head == NULL)
          {
            at_of_next_dep = sys.last_arrival;
          }
        e = malloc(sizeof(struct Event));
        e->Arrival_Time = sys.last_arrival;
        e->next = NULL;
        current_time = sys.last_arrival;
        add_event(e);
        sys.num_customers++;

 return 0;
 }

//This Function generates service time, serves and De queue an event from the List of Arrival
Events Queue
int service()
 {
 double service_time;
 struct Event *e;
        e = dequeue();
        sys.num_customers--;
        service_time = getexp(1.0/mu);
 if(sys.head == NULL )
  {
        st_of_next_dep = at_of_next_dep;
  }
  else
        st_of_next_dep = Departure_Time;
        Departure_Time = st_of_next_dep + service_time;
   avg_system = avg_system + ( Departure_Time - at_of_next_dep );
        num_departures++;

  if(sys.head == NULL )
     {
     }
  else
         at_of_next_dep = sys.head->Arrival_Time;

 if(Departure_Time <= Endofsim)
  {
       while(current_time <= Departure_Time)
       {
          if(Departure_Time > sys.last_arrival)
          {
```

```
            T1 = time;
            time = sys.last_arrival;
            T2 = time;
            X = X + (Qlength * (T2 - T1));
            if(Qlength == 0)
              U = U + (T2 - T1);
            Qlength++;
            }
else
           {
            T1 = time;
            time = Departure_Time;
            T2 = time;
            X = X + (Qlength * (T2 - T1));
            if(Qlength == 0)
               U = U + (T2 - T1);
            Qlength--;
           }
            generate_arrivals();
       }

  }
 else
 {
  printf("Average System Time = %f\n,"avg_system/num_departures);
        printf("Mean Queue Length = %f \n,"X/Endofsim);
        printf("Utilization = %f \n,"(1-(U /Endofsim)));
        abort();
 }
 return 0;
 }

// main program that calls all the above decelared functions
int main()
 {
         initialization();
         generate_arrivals();

    while(current_time <= Endofsim)
            {
               service();
               if(Departure_Time < sys.last_arrival);
               {
                T1 = time;
                time = Departure_Time;
                T2 = time;
```

```
                X = X + (Qlength * (T2 - T1));
                if(Qlength == 0)
                  U = U + (T2 - T1);
                Qlength--;
               }
            }
 printf("Average System Time = %f\n,"avg_system/num_departures);
 printf("Mean Queue Length = %f \n,"X/Endofsim);
 printf("Utilization = %f \n,"(1-(U/Endofsim)));
  return 0;
}
```

APPENDIX B

SENSOR SYSTEM CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Pcon   0.7
#define Pcoff  0.9
#define Q      0.5
#define C      1
#define del1   1
#define del2   6
#define K      3000 /*Energy Bucket Size*/
#define Endofsim 10000
#define POLICY     6  /*POLICY = 1 for AW policy:: POLICY = 2 for CW policy :: POLICY =
3 for MAIMAD policy:: POLICY = 4 for MAIMD policy::POLICY = 5 for MIAD policy ::
POLICY = 6 for AIMD policy*/
#define SLEEPINTERVAL 11
int EO;/* Denotes Event Occurrence "1" represents Event Occured,"0"
represents Event Did not occured*/
int EL;/* Represents Energy Level of the sensor*/
//int SLEEPINTERVAL;/* Sleep Interval of the sensor in CW POLICY*/
int SL;/* Sleep Interval of the sensor*/
int sl;/* Sleep Interval of the sensor*/
int T;/* Current Time*/
int num_events_occured;
int num_events_detected_AW;
int num_events_detected_CW;
int num_events_detected_MAIMAD;
int num_events_detected_MAIMD;
int num_events_detected_MIAD;
int num_events_detected_AIMD;
/*Function which generates uniform distributed random numbers and returns this values*/
double Randgen()
 {
   double x;
   x = drand48();// generates random number between (0,1]
   return x;
 };

double U_CW()
{
double y,pion;
```

```
pion = (1-Pcoff)/(2-Pcon-Pcoff);

y = (Q*C)/((pion*(del1 + del2))+ (del1*(1 - Pcon)));
return y;

}

double U_AW()
{
double z,Pion;

Pion = (1- Pcoff)/(2- Pcon - Pcoff);

z = (Q*C)/(del1 + (del2*Pion));
return z;
}

double Sleep()
{
double s,x,y,z,Pion;

Pion = (1 - Pcoff)/(2 - Pcon - Pcoff);

x = (del1/(Q*C));
y = Pion*(del1 + del2 - (Q*C));
z = (Q*C)*(1 - Pcon);
s = x + (y/z) - 1;

return s;
}

/*Initilization of the variables used in the program*/
int initialization()
 {
   num_events_occured = 0;
   num_events_detected_AW = 0;
   num_events_detected_CW = 0;
   num_events_detected_MAIMAD = 0;
   num_events_detected_MAIMD = 0;
   num_events_detected_MIAD = 0;
   num_events_detected_AIMD = 0;
   EO =  0;
   EL =  0;
   SL =  0;
   sl =  0;
   srand48(10);// seeds the input to Random Number Generator
```

```
 return 0;
}

/*This Function generates Recharge Events*/
int recharge_sensor()
{
 double g;
 g = Randgen();
    if(EL < K)
     {
      if(g <= Q)
        {
          EL = EL + C;
          if(EL > K)
            {
              EL = K;
            }
          else
             EL = EL;
        }
      else
          EL = EL;
     }
    else
      EL = EL;

  return 0;
}


/* This Function Generates Event Occurrence Process*/
int event_occurence()
{
  double P,l;
  l = Randgen();
                if (EO == 0)
                  {
                    P = Pcoff;
                    if (l <= P)
                       EO = 0;
                    else
                      {
                       EO = 1;
                       num_events_occured++;
                      }
```

```
                }
              else
                {
                  P = Pcon;
                  if (l <= P)
                    {
                      EO = 1;
                      num_events_occured++;
                    }
                  else
                      EO = 0;
                }
    return 0;
 }

/* Aggressive Wakeup Activation Policy*/
int activation_policy_AW()
 {

   if( EL >= (del1 +del2))
     {
       if(EO == 1)
         {
           EL = EL - (del1 + del2);
           num_events_detected_AW++;
         }
       else
          EL = EL - del1;
     }
  else
     {
       EL = EL;
     }
  return 0;
 }

/*CW Activation Policy*/
int activation_policy_CW()
   {

          if(EL >= (del1 + del2))
             {
                if(SL == 0)
                  {
                    if(EO == 0)
                      {
```

```
                    EL = (EL - del1);
                    SL = SLEEPINTERVAL;
                   }
                 else
                   {
                    EL = EL - (del1+ del2);
                    num_events_detected_CW++;
                   }
                }
               else
                  SL--;
           }
          else
           {   if(SL == 0)
              {
               SL = SLEEPINTERVAL;
              }
              else
              {
               SL = SL - 1;
              }
           }
    return 0;
   }

/* MAIMAD Activation policy*/
int activation_policy_MAIMAD()
   {

         if(EL >= (del1 + del2))
           {
              if(SL == 0)
                {
                 if(EO == 0)
                   {
                    EL = (EL - del1);
                    SL = SLEEP_INTERVAL_MAIMAD();
                   }
                 else
                   {
                    EL = EL - (del1+ del2);
                    num_events_detected_MAIMAD++;
                   }
                }
               else
                  SL--;
```

```
            }
          else
            {
                if(SL==0)
                  SL = SLEEP_INTERVAL_MAIMAD();
                else
                  SL = SL--;
            }
 return 0;
   }

int SLEEP_INTERVAL_MAIMAD()
{
        if(EL < K/4)
            {
              if(sl == 0)
               {
                 sl = 1;
               }
              else
                 sl = sl*2;
            }
        else
        if((EL >= K/4)&&(EL < K/2))
            {
                 sl = sl + 1;
            }
        else
        if((EL >= (K/2)) && (EL < ((3*K)/4)))
            {
              if(sl == 0)
                sl = sl;
              else
                sl = sl - 1;
            }
          else
            {
               sl = sl/2;
            }
return sl;
}


/* MAIMD Activation policy*/
int activation_policy_MAIMD()
   {
```

```
        if(EL >= (del1 + del2))
          {
            if(SL == 0)
             {
              if(EO == 0)
               {
                EL = (EL - del1);
                SL = SLEEP_INTERVAL_MAIMD();
               }
              else
               {
                EL = EL - (del1+ del2);
                num_events_detected_MAIMD++;
               }
             }
            else
              SL--;
          }
         else
          {
            if(SL==0)
              SL = SLEEP_INTERVAL_MAIMD();
            else
              SL = SL--;
          }
 return 0;
  }

int SLEEP_INTERVAL_MAIMD()
{
        if(EL < K/4)
          {
          if(sl == 0)
          {
            sl = 1;
          }
          else
            sl = sl*2;
          }
        else
          if((EL >= (K/4)) && (EL < K/2))
              {
              sl = sl + 1;
              }
          else
```

```
                {
                sl = sl/2;
                }
return sl;
}

/* MIAD Activation policy*/
int activation_policy_MIAD()
  {

        if(EL >= (del1 + del2))
          {
            if(SL == 0)
              {
               if(EO == 0)
                 {
                  EL = (EL - del1);
                  SL = SLEEP_INTERVAL_MIAD();
                 }
               else
                 {
                  EL = EL - (del1+ del2);
                  num_events_detected_MIAD++;
                 }
              }
            else
              SL--;
          }
         else
          {
             if(SL==0)
               SL = SLEEP_INTERVAL_MIAD();
             else
               SL = SL--;
          }

 return 0;
  }

int SLEEP_INTERVAL_MIAD()
{
        if(EL < K/2)
            {
            if(sl == 0)
              {
               sl = 1;
```

```
            }
          else
            sl = sl*2;
          }
        else
          {
           if(sl == 0)
            {
            }
           else
             sl = sl -1;
          }
return sl;
}

/* AIMD  Activation policy*/
int activation_policy_AIMD()
  {

        if(EL >= (del1 + del2))
          {
            if(SL == 0)
             {
              if(EO == 0)
                {
                 EL = (EL - del1);
                 SL = SLEEP_INTERVAL_AIMD();
                }
              else
                {
                 EL = EL - (del1+ del2);
                 num_events_detected_AIMD++;
                }
              }
            else
              SL--;
          }
         else
          {
            if(SL==0)
              SL = SLEEP_INTERVAL_AIMD();
            else
              SL = SL--;
          }

 return 0;
```

```
  }

int SLEEP_INTERVAL_AIMD()
{
        if(EL < K/2)
            {
            sl = sl +1;
            }
        else
            {
            sl = sl/2;
            }
return sl;
}

// main program that calls all the above declared functions
int main()
 {
double z,y,s;

       if(POLICY == 1)
         {
            initialization();
            for(T = 0; T<=Endofsim; T++)
              {  recharge_sensor();
                 event_occurence();
                 activation_policy_AW();
              }
             z = U_AW();
         }
       else
       if(POLICY == 2)
         {
//for(SLEEPINTERVAL= 0;SLEEPINTERVAL <= 25 ; SLEEPINTERVAL++)
//{
            initialization();
            for(T = 0; T<=Endofsim; T++)
              {
                 recharge_sensor();
                 event_occurence();
                 activation_policy_CW();
              }

  //}
             y = U_CW();
             s = Sleep();
```

```
        }
    else
    if(POLICY == 3)
      {
        initialization();
        for(T = 0; T<=Endofsim; T++)
          {
            recharge_sensor();
            event_occurence();
            activation_policy_MAIMAD();
          }
       }
    else
    if(POLICY == 4)
      {
        initialization();
        for(T = 0; T<=Endofsim; T++)
          {
            recharge_sensor();
            event_occurence();
            activation_policy_MAIMD();
          }
        }
    else
    if(POLICY == 5)
      {
        initialization();
        for(T = 0; T<=Endofsim; T++)
          {
            recharge_sensor();
            event_occurence();
            activation_policy_MIAD();
          }
       }
    else
      {
        initialization();
        for(T = 0; T<=Endofsim; T++)
          {
            recharge_sensor();
            event_occurence();
            activation_policy_AIMD();
          }
     }
 return 0;
}
```