12-1-2003

Maximizing the capability of wireless sensor networks

Cory Cress

Rochester Institute of Technology

MAXIMIZING THE CAPABILITY OF WIRELESS SENSOR NETWORKS

A Thesis

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Industrial Engineering

in the

Department of Industrial & Systems Engineering

Kate Gleason College of Engineering

by

Cory D. Cress

B.S., Industrial Engineering, Rochester Institute of Technology, 2003

December, 2003

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

KATE GLEASON COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of Cory D. Cress

has been examined and approved by the

thesis committee as satisfactory for the

thesis requirement for the

Master of Science degree

Approved by:

_____

Dr. Moises Sudit

_____

Dr. Shanchieh Jay Yang

MAXIMIZING THE CAPABILITY OF WIRELESS SENSOR NETWORKS

ACKNOWLEDGEMENTS

I would like to extend sincere thanks to my thesis committee members, Dr. Moises Sudit and Dr. S. Jay Yang. Dr. Sudit was very helpful with my thesis in providing me with his expertise in Operations Research and Discrete Optimization. His superior knowledge and experience in Operations Research was inspiring. Dr. Yang's dedication, patience, and attention to detail were incredible. His hard work and enthusiasm about this thesis motivated me to achieve a level of success beyond my expectations.

Also, I would like to thank Jacqueline Mozrall, Marilyn Houck, and the rest of the faculty and staff in the Industrial and Systems Engineering Department at RIT; your dedication to your students is unparalleled.

Finally, I would like to thank my family for always being supportive of me in all of my academic endeavors. I cannot put into words, the gratitude I have for you. Thank you so much for your sacrifices and for keeping me focused on my goals.

Abstract

Wireless micro-sensors introduce a new frontier in sensing devices and data acquisition capabilities. These sensors, capable of sensing, processing data, and short-range communication, can be spread over regions to form ad hoc wireless sensor networks (WSN) so as to deliver aggregate information from geographically diverse areas. This aggregate data gathering and processing induces a synergistic effect and enables a sensor network to complete sensing tasks that may never be feasible using a single, perhaps powerful, sensor. This new paradigm in sensing devices is not without many fundamental challenges, one being a constrained energy resource, which first need to be solved before the true capabilities of these networks may be realized.

This thesis will discuss the models and techniques developed as an attempt to maximize the capability of a WSN. The premise used in the research is that the capability of a WSN can be maximize by developing a scheme that can duplicate the optimal energy efficient behavior of individual wireless sensors in a contention dominated, distributed decision-making, network environment. This optimal energy efficient behavior as determined by an analytically derived model and a mixed integer programming model will be presented. The analytical model enables the optimal sensor behavior to be calculated given a contention-less environment, and the integer programming model determines the optimal ON/OFF/transmission schedule for each sensor in a contention dominated network, over time. Finally, the optimal behavior found in the two models has been converted into a preliminary heuristic protocol that coordinates sensors in "real time." The key aspects of this protocol along with its effectiveness, as compared to the optimal, are also presented.

TABLE OF CONTENTS

TABLE OF FIGURES

1. Introduction

Wireless micro-sensors introduce a new frontier in sensing devices and data acquisition capabilities. These sensors, capable of sensing, processing data, and short-range communication, can be spread over regions to form ad hoc Wireless Sensor Networks (WSN) that deliver aggregate information from geographically diverse areas. This aggregate data gathering and processing induces a synergistic effect and enables a sensor network to complete sensing tasks that may never be feasible using a single, perhaps powerful, sensor.

Technological advances in Integrated Circuit (IC) fabrication and Micro-Electronic Mechanical Systems (MEMS) have enabled wireless sensors to be made at the micro scale. Meanwhile, the price of making these devices has also been reduced. These reductions in size and cost have made self-organizing ad hoc networks of thousands of sensors theoretically possible. However, unlike the mechanical and electronic devices that make up wireless micro-sensors, an inexpensive small scale power supply, i.e., a battery, capable of sustaining a micro-sensor for many years is still under development. In order to continue making smaller and less expensive sensors, smaller batteries supplying less energy are used since there are no other options. The use of these batteries restricts the total energy supply of each sensor, which, in turn, constrains the energy supply that is available to the entire network. With energy at a premium, the key to achieving the maximum capability from a WSN is to determine how to optimize the energy usage at the individual sensor level such that the maximum amount of data can be retrieved at the network level. Two aspects impact how network capability can be maximized subject to individual sensor resource constraints: the architecture of individual micro-sensors and the network level operations. The following

two sections provide an overview of wireless micro-sensors and a summary of existing network level approaches that attempt to minimize energy consumptions.

1.1 Overview of Micro-sensor Architecture

Well known, modern sensing devices, or nodes, include COTS Dust and Smart Dust developed by UC Berkeley, UCLA's wireless sensing node termed WINS, Rockwell's node also named WINS, and Sensor Webs by JPL [17]. These developers typically use off-the-shelf components to design their nodes. This strategy makes the nodes modular and easily adjustable; however the choice of component can drastically alter the performance, energy consumption, etc., of the final device. Most nodes are comprised of a power module, radio frequency module, sensing module, and processing module [1], as shown in Figure 1.

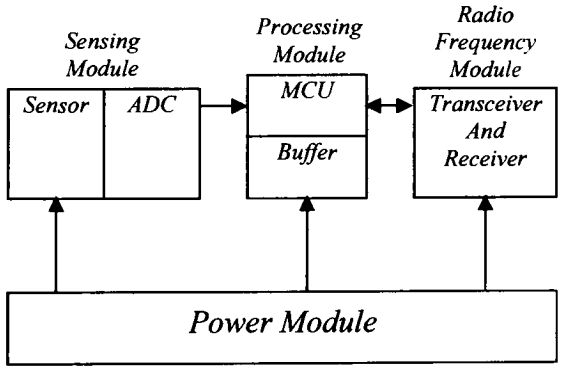

Figure 1.1 Architecture of a wireless sensor.

The power module shown in Figure 1.1, supplies the energy to each of the other modules. The sensing module continuously collects data, converts it from an analog to a digital signal, and then passes it to the processing module. The data is processed by the Micro Controller Unit (MCU) and may either be buffered or passed to the Radio Frequency (RF) module.

The RF module is the only part the sensing device that is directly affected by other sensors contending for access to a communication channel or a data collection gateway. It also consumes a significant amount of the overall energy provided by the power module. In fact, studies in the past have shown that transmitting 1 bit over 100 meters consumes roughly the same amount of energy as executing 3000 instructions by the MCU [10], which is highly significant. A plus side to the RF module is that its complexity, transmission rate, transmission range, reliability, etc., may all be adjusted to meet specific requirements and therefore enhanced capabilities may be sacrificed in order to conserve energy. Many RF modules also have the capability of operating in different energy consumption modes, including sleep, idle, receive, and transmit modes [13].

The sleep mode, which will also be referred to as the OFF mode, consumes a negligible amount of energy and is typically used during inactive periods to conserve energy. The idle mode is the mode in which the RF circuit is turned ON yet the transceiver is not receiving or transmitting data. To switch from OFF to ON, an amount of spike energy is consumed and has been shown to be quite significant [18]. As a result, one must use the energy conserving OFF mode cautiously since switching between ON and OFF may result in a larger overall consumption due to the spike energy. In the ON mode, however, the sensor can switch between transmit and receive modes without consuming any spike energy.

The transmit mode is used by the transceiver to send data, and consumes a significant amount of energy in addition to that consumed in the ON mode. Finally, the receive mode, similar to the transmit mode, consumes energy in addition to the ON mode consumption, however

consumes only a small amount of addition energy [19]. Also note that the receive mode and the transmit mode both require the use of the transceiver and, therefore, cannot be preformed simultaneously.

Over the lifetime of a sensor, the majority of the sensor's energy is typically consumed by the RF module, thus the focus of a sensor or network operational model should aim at understanding how this module consumes energy during the OFF, ON, and transmit modes. Also, in a "multi-hop" network, where data is relayed through multiple sensors before reaching the final destination, the receive mode should also be considered in the model. Finally, the spike energy needs to be considered in a model so that switching between the OFF mode and the ON mode can be investigated under realistic energy consumption conditions.

1.2 Literature Review on Energy Efficient WSN Operations

Many methods for reducing a sensor's energy consumption have been proposed in literature. In general, these approaches can be divided into one of two categories. The first set of approaches indirectly reduces the amount of energy spent by the RF module by routing data throughout a network using the most energy-efficient paths. They focus on "multi-hop" networks in which data may jump from sensor to sensor until it reaches its final destination. In essence, this set of work looks at ways to balance data flows over the entire network in the *steady state,* such that the amount of energy consumed by each sensor's RF module is roughly the same. Optimal steady state solutions have been found for models seeking to maximize the time until the first sensor is completely drained of energy [2][14][15][21], or

for models aiming at maximizing the data retrieved over the entire network [6][12]. These *steady state* models provide information about how data should be disseminated throughout the network on average over the network lifetime and provide innovative methods for how WSN optimization models may be developed. Yet, they do not provide adequate information that would enable sensors to make decisions in real time and do not account for the spike energy since the model is developed for the steady state, i.e., no switching between OFF and ON modes.

The second set of approaches taken to reduce a sensor's energy consumption is to develop efficient Medium Access Control (MAC) protocols that create more efficient transmission schedules by utilizing the different energy modes of the RF module. These protocols [13][18][19][20] mostly make decisions with regard to switching the RF module of a sensor between transmit mode and OFF mode and generally achieve significant performance improvements when compared to schemes that keep the sensor in ON mode or transmit mode at all times. This set of work, however, does not account for the data that has been collected by the sensing circuitry and often overlooks the spike energy consumed. In addition, these protocols focus on heuristically finding the compromise among the various energy consumption factors but lack a theoretical foundation to compare actual performance with maximum performance capabilities.

2. Problem Description

The approaches reviewed in the previous section offer invaluable lessons on how energy can be conserved, either directly or indirectly, by the RF module. Yet they are not capable of determining how each networked sensor should turn ON, OFF, or transmit over time, such that the network capability is maximized. For instance, the first approach, least-energy data flows, provides a good method for modeling a sensor network environment, i.e., the optimization model. Yet, the optimal results obtained are limited since they lack information about the behavior of the sensors over time. As for the second approach, the MAC protocols are capable of conserving energy by making the sensors behave more efficiently over time. However, these protocols are based on single sensor power consumption models and therefore do not accurately incorporate the impact of contending sensors in a network environment. The inadequacy of both of the above approaches is that they attempt to solve a specific problem rather than looking at the broad challenge.

The research in this thesis attempts to maximize the capability of a WSN by determining how the optimal energy efficient behavior[1] of individual wireless sensors can be duplicated in a contention dominated network environment. Using the contributions of other research as reference, various modeling approaches are used to determine the optimal sensor behavior. First, an analytically derived model is developed to determine the optimal single sensor energy efficient behavior given an optimal environment, i.e., no contention among sensors when transmitting. Two schemes: first, sensors are kept ON at all times and second, sensors are switched between ON and OFF modes, are examined in this model. For the second

[1] Refers to how the RF module switches between ON mode, OFF mode, or transmit mode over time.

scheme where sensors are turned ON and OFF, the effect of buffering data for a variable period of time and allowing sensors to transmit at the most efficient time is explored. In addition, the spike energy and other realistic energy parameters are accounted for in this model.

This single sensor behavior model, due to the fact that it represent the ideal case, serves as the theoretical bounds in comparison to results obtained in a network environment and results obtained via simulation models. This thesis proposes a mixed integer programming model in a dynamic regime. This model, which accounts for realistic energy consumption parameters and buffer, determines the optimal ON/OFF/transmission schedule, or behavior, of each sensor ***over time.*** These results may then be used to better understand how a contention dominated network environment affects the optimal, contention-less, single sensor behavior. After developing a baseline contention model solution, the effect of independent factors including buffer size, arrival pattern, service capacity, network connectivity, and receiving duty cycle are evaluated with respect to the baseline model capability.

Knowing the optimal transmission schedules over time enables this behavior to be converted to a distributed decision making heuristic. A heuristic of this type is a set of rules that enables a sensor to independently schedule when to turn ON, turn OFF, and transmit in ***real time***. It is an important extension, as it will enable a sensor to be capable of adjusting to real world changes on-line. Future research may include the development of a MAC protocol. To conclude this research, however, simulation results for sensors controlled by a preliminary on-line heuristic are presented as compared to the optimal results.

3. Optimal Sensor Behavior

The analytically derived model used to investigate the optimal single sensor energy efficient behavior will be discussed in this section. This model is developed assuming an optimal communication scenario. Therefore, the discussion begins with a description of the model's optimal communication scenario, variables, and energy parameters. This will be followed by a comparison of two schemes, one in which the RF module of the sensor is ON at all times, and a second scheme in which the RF module is turned ON and OFF. The goal of this model is to investigate the optimal sensor behavior over time when buffering of sensed data is allowed.

The best communication scenario for a sensor in a sensor network is to always have full access to a data collection gateway whenever it is needed and to have every bit of data transmitted reach the destination successfully. Full access to a data collection gateway could be achieved through other sensors; however, this multi-hop capability is ignored in this model. This best scenario is assumed for this model and the amount of data a sensor can sense and transmit to a data collection gateway (or gateway in short) given the total energy, e_{tot}, available for the sensor, will be investigated. Let the total amount of data retrieved by the gateway be $R = \int_t U(t)dt$, where $U(t) \leq \mu$ is the amount of data transmitted by the sensor at time t, and μ is the maximum transmission capacity of the sensor. Also let $\lambda \leq \mu$ be the fixed arrival rate of the sensed data. For the different energy consumption parameters, keeping the RF circuit ON for one time unit is designated, e_o, the energy for transmitting one unit of data is designated, e_t, the energy for sensing data for one time unit is designated, e_s,

and the energy consumed for switching the RF circuit from OFF to ON is designated, e_p, also referred to as the spike energy.

3.1 Continuous ON Scheme

First consider the case where a sensor has its RF circuit remain ON at all times. The following claim shows that the total data retrieval under this scheme, R^*_{ContON}, is independent of the amount of data that is retrieved at each time period.

Theorem 1: The maximum amount of data a sensor can retrieve when a sensor is kept Continuously ON over its entire lifetime can be represented as:

$$R^*_{ContON} = \frac{\lambda}{e_o + e_s + e_t\lambda} \bullet e_{tot} . \qquad (3.1)$$

Proof: Let T be the lifetime of a sensor. Then the total energy consumed, e_{tot}, and the amount of data to be retrieved, R_{ContON}, are

$$e_{tot} = e_s T + e_o T + e_t \int_{k=0}^{T} U(k)dk \qquad \text{and} \qquad R_{ContON} = \int_{k=0}^{T} U(k)dk$$

Based on the assumption that no sensed data is lost and the buffer may not be over filled, one can determine that

$$\lambda T - B \le \int_{k=0}^{T} U(k)dk \le \lambda T ,$$

where, B, is the buffer size of the sensor. Let $\int_{k=0}^{T} U(k)dk = \lambda T - \delta$, where $0 \le \delta \le B$. One can therefore rewrite, $e_{tot} = e_s T + e_o T + e_t(\lambda T - \delta)$, which leads to

$$T = \frac{e_{tot} + e_t \delta}{e_s + e_o + e_t \lambda}$$

Now, one can rewrite R_{ContON} as a function of δ,

$$R_{ContON} = \lambda T - \delta$$

$$= \frac{e_{tot}\lambda + e_t \lambda \delta}{e_s + e_o + e_t \lambda} - \frac{e_s \delta + e_o \delta + e_t \lambda \delta}{e_s + e_o + e_t \lambda}$$

$$= \frac{e_{tot}\lambda - \delta(e_s + e_o)}{e_s + e_o + e_t \lambda}$$

Clearly, R_{ContON} is maximized when, $\delta = 0$ and the maximum data retrieval

$$R^*_{ContON} = \frac{\lambda}{e_s + e_o + e_t \lambda} \bullet e_{tot}$$

■

This result is due to the simple fact that the cumulative amount of data that can be collected at any point in time cannot be greater than what has been sensed. Apparently, one way to achieve this optimal performance is by simply letting $U(t) = \lambda, \forall t > 0$, i.e., transmitting whatever is sensed immediately upon arrival. This policy is referred to as the "Continuous ON" scheme.

Figure 3.1 Buffer utilization for a sensor turning ON and OFF periodically through its lifetime.

3.2 ON/OFF Scheme

In contrast to the Continuous ON scheme, one may utilize the buffer space, to temporarily store the sensed data while the RF circuit is OFF, and then turn the RF circuit ON to transmit, perhaps at full capacity for one or many consecutive time periods, as shown in Figure 3.1. To understand what constitutes an optimal behavior, the energy expenditure of a general network without any contentions, i.e., sensor can transmit data whenever it desires, is analyzed, given a fixed amount of data, R, to be retrieved. The following equation represents the lower bound on the total energy, e_{LB}, consumed under this setting:

$$e_{LB} = e_p + e_s \frac{R}{\lambda} + e_o \frac{R}{\mu} + e_t R$$

To minimize the spike energy, a single OFF/ON cycle is assumed, which leads to a single spike in energy, e_p, i.e., the first term of the above equation above. The second term of the equation, $e_s \frac{R}{\lambda}$, represents the minimum amount of sensing energy that could be used, by assuming that the sensor will completely drain its buffer and completely consume all of its energy at precisely the same time. Assuming the maximum transmit capacity, μ, is utilized

for every transmission, the third term, $e_o \frac{R}{\mu}$, represents the minimum ON energy consumed by the sensor. Finally, the amount of energy to transmit R units of data is simply a factor of the amount of data and the energy consumed per bit, $e_t R$.

Based on the equation above, one can derive the following result for the optimal amount of data that can be retrieved, $R^*_{ON/OFF}$, by an ON/OFF scheme in a network without contentions.

Theorem 2: The largest amount of data that could be retrieved by a sensor using and ON/OFF scheme, $R^*_{ON/OFF}$, is

$$R^*_{ON/OFF} = \frac{\left(e_{tot} - e_p\right)}{\left(\frac{e_s}{\lambda} + \frac{e_o}{\mu} + e_t\right)} \qquad (3.2)$$

Proof: Consider the lower bound energy described above,

$$e_{tot} \geq e_{LB} = e_p + e_s \frac{R_{ON/OFF}}{\lambda} + e_o \frac{R_{ON/OFF}}{\mu} + e_t R_{ON/OFF}$$

$$e_{tot} \geq e_p + R_{ON/OFF}\left(\frac{e_s}{\lambda} + \frac{e_o}{\mu} + e_t\right)$$

$$R^*_{ON/OFF} = \frac{\left(e_{tot} - e_p\right)}{\left(\frac{e_s}{\lambda} + \frac{e_o}{\mu} + e_t\right)}$$

■

A logical progression from this finding is to determine whether there exists a simple transmission schedule that is capable of achieving this upper bound of data retrieval. The

answer is yes, by having the sensor complete one and only one "full ON/OFF cycle." A full ON/OFF cycle starts with a sensor being OFF, storing any data it senses in its buffer, and then turning ON and transmitting continuously until the buffer is empty. Note that the optimal $R^*_{ON/OFF}$ is achieved if the sensor's energy is completely expended at exactly the same time the buffer is expended. To complete a full ON/OFF cycle, a buffer of sufficient capacity is required.

Although the cost of memory has become relatively inexpensive, it is still necessary to investigate the actual amount of storage required to complete a full ON/OFF cycle. In fact, a buffer size of this magnitude may lead to an unacceptable amount delay. A natural progression of Theorem 2 allows the derivation of the buffer size required to achieve $R^*_{ON/OFF}$.

Corollary 1: Let B^* represent the smallest buffer size required by a sensor to achieve its maximum data retrieval capability in a contention-less environment, and $e_p > 0$, then,

$$B^* = \frac{(\mu - \lambda)(e_{tot} - e_p)}{e_o + \mu e_t + \frac{\mu}{\lambda} e_s} \tag{3.3}$$

Proof: Consider a sensor that has a buffer of size B, and follows a full ON/OFF transmission cycle. The total data retrievable, R, achievable by this sensor is

$$R = \mu\left(\frac{B}{\mu - \lambda}\right)$$

Given that $R \le R^*_{ON/OFF}$ and Theorem 2, it can be concluded that

$$\frac{e_{tot} - e_p}{\frac{1}{\lambda}e_s + \mu e_t + e_o\frac{1}{\mu}} = R^*_{ON/OFF} \ge R = \mu\left(\frac{B}{\mu - \lambda}\right)$$

$$\Rightarrow \frac{(e_{tot} - e_p)(\mu - \lambda)}{\frac{\mu}{\lambda}e_s + \mu e_t + e_o} = B^* \ge B$$

∎

Based on Theorem 1 and Theorem 2, one can also determine the condition (as a function of the energy parameters, arrival rate, and service capacity) for which the ON/OFF scheme will retrieve a greater amount of data than the Continuous ON scheme, or $R^*_{ON/OFF} \ge R^*_{ContON}$.

Corollary 2: A sensor using the optimal ON/OFF scheme will retrieve a greater amount of data than the optimal Continuous ON scheme if and only if

$$e_p \le e_{tot}\left(\frac{\mu - \lambda}{\mu}\right)\frac{e_o}{e_o + e_s + e_t\lambda} \qquad (3.4)$$

Proof: Consider the optimal Continuous ON scheme (3.1) and the optimal ON/OFF scheme (3.2).

$$\frac{(e_{tot} - e_p)}{\left(\frac{e_s}{\lambda} + \frac{e_o}{\mu} + e_t\right)} = R^*_{ON/OFF} \ge R^*_{ContON} = \frac{\lambda}{e_o + e_s + e_t\lambda} * e_{tot}$$

$$\Rightarrow \frac{(e_{tot} - e_p)}{\left(\frac{e_s}{\lambda} + \frac{e_o}{\mu} + e_t\right)} \ge \frac{\lambda}{e_o + e_s + e_t\lambda} * e_{tot}$$

$$\Rightarrow -e_p \ge \frac{\lambda}{e_o + e_s + e_t\lambda} * e_{tot} * \left(\frac{e_s}{\lambda} + \frac{e_o}{\mu} + e_t\right) - e_{tot}$$

$$\Rightarrow e_p \le e_{tot}\left(\frac{\mu - \lambda}{\mu}\right)\frac{e_o}{e_o + e_s + e_t\lambda}$$

∎

3.3 Comparison of Continuous ON Scheme and ON/OFF Scheme

The above results can be used to determine whether employing an optimal ON/OFF scheme is beneficial (as compared to the Continuous ON scheme), and to determine the maximum amount of data that may be retrieved by using this ON/OFF scheme. However a significant, or perhaps unrealistic, buffer space may be required to construct a *single* ON/OFF cycle throughout the sensor lifetime. Moreover, as the buffer size increases, the queuing delay of the sensed data also increases, which may pose yet another problem from the application's perspective. One way to accommodate a smaller buffer size than the one shown in Figure 3.2 is to have multiple ON/OFF cycles for an ON/OFF scheme. One can easily derive the total amount of retrieved data for a multiple ON/OFF cycle scheme based on (3.2) and (3.3). Figure 3.2 exhibits the amount of data that can be retrieved by using the two schemes: Continuous ON and ON/OFF with respect to the number of ON/OFF cycles. The maximum queuing delay associated with using such ON/OFF schemes is also shown in Figure 3.2 and is also plotted with respect to the number of ON/OFF cycles.

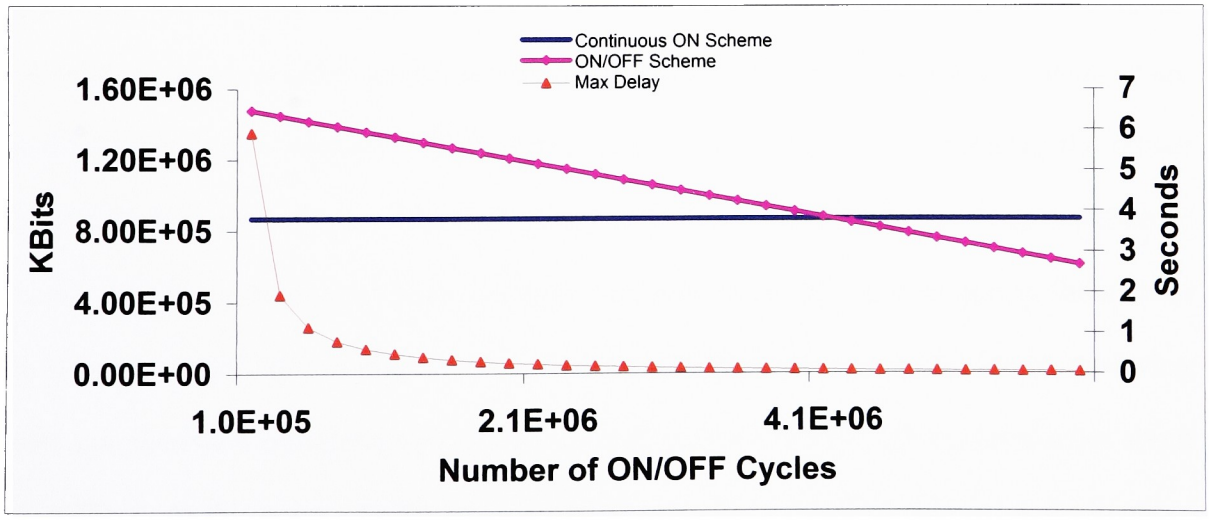

Figure 3.2 Data retrieved by using the Continuous ON scheme and the ON/OFF scheme with different numbers of ON/OFF periods; also shown is the maximum delay associated with the ON/OFF scheme.

Parameter	Value	Units
λ	2	Kbit/sec
μ	10	Kbit/sec
e_s	.01	J/sec
e_p	.001	J
e_t	.0005	J/Kbit
e_o	.012	J/sec
e_{tot}	10,000	J

Table 3.1 Model Parameters

To plot Figure 3.2, the set of parameters shown in Table 3.1 were considered. These parameters were derived based on the information provided in [4][13], for the purpose of exhibiting the realistic performance achievable by a wireless micro-sensor. Sensors for different sensing purposes, which have a different RF implementation, may have a different set of parameters. However, the general performance trend is found to remain consistent for a large variety of parameter choices. Given the set of parameters shown in Table 1, the maximum possible retrievable amount of data is found to be about 1500 Mbits for the ON/OFF scheme with a single ON/OFF cycle (not shown in Figure 3.2), and about 870 Mbits (the constant line shown in Figure 3.2) for the Continuous ON scheme. The buffer size and the queuing delay associated with the single ON/OFF cycle scheme however are unrealistically large – about 1.2 Gbits and 7 hours worth of delay. By stretching the sensor operation into more and more ON/OFF cycles, it is found that the retrievable data decreases linearly, while the maximum delay decreases much more quickly as a reciprocal function to the number of ON/OFF cycles. Notice that the ON/OFF scheme is capable of retrieving more data than the Continuous ON scheme until the number of ON/OFF cycles reaches about 4.2 million. At this point a buffer of only 160 bits would be required, which results in about 80 milliseconds of maximum queuing delay. An attractive operation point of the ON/OFF

scheme is at, perhaps, 1 million ON/OFF cycles, where a significant amount of data (1340 Mbits out of possible 1500 Mbits) can be retrieved while the delay is dropped significantly to about 500 milliseconds with the buffer size being about 1 Kbit. This set of numerical results suggests the superiority of an ON/OFF scheme over the Continuous ON scheme, even with a quite large number of ON/OFF cycles.

4. Networked Sensor Behavior

The analysis in the previous chapter has suggest that a sensor should store as much sensed data as possible and then transmit at full capacity, as long as the sensor is capable of transmitting to a data collection gateway whenever it desires. In a network setting, however, multiple sensors may contend for the same wireless communication channel of each gateway; thus, the full ON/OFF cycle described in the previous section may not be feasible. In this chapter, the optimal, contention dominated, networked sensor behavior is investigated by considering a mixed integer programming model. This task is accomplished by analyzing the behavior of the sensors in a baseline model. With the behavior of the baseline contention model solution determined, the effect of the buffer size, arrival pattern, network connectivity, and service capacity, are evaluated with respect to the baseline model behavior and capability.

The interaction among gateway sharing sensors is modeled using a bipartite network, as shown in Figure 4.1, in which one set of nodes represents the sensors and are connected to the other set of nodes representing the gateways. This simple bipartite structure allows the focus to be placed on the ON/OFF behavior of the sensors that are contending for a common gateway as well as those that may be serviced by multiple gateways. Although it seems like a simplified sensor network, the bipartite network model can be viewed as the building block of any conceivable network model.

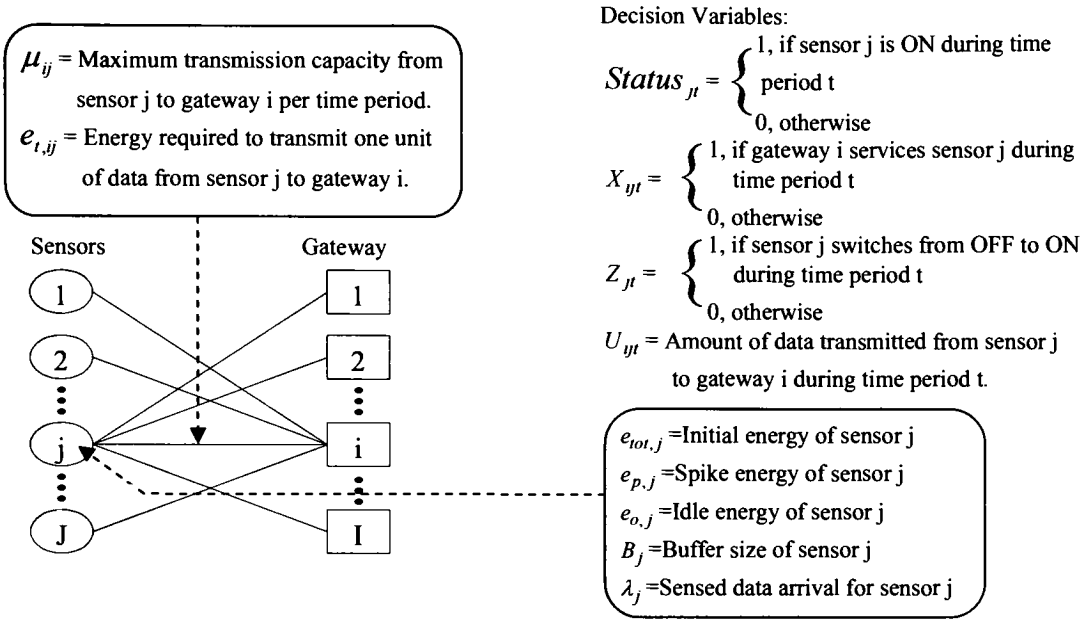

Figure 4.1 The sensor-gateway bipartite network model.

To understand in depth how contending sensors should behave over time, "time" is a discrete variable in the mixed integer programming model. The notation defined in the previous section is followed, and extended with subscripts to indicate the associated sensor $\{j;\ j = 1,...J\}$, gateway $\{i; i = 1,...I\}$, and time period $\{t; t = 1,...T\}$. Figure 4.1 shows the variables and constants that define all data that will be represented in the model. Figure 4.2 defines the objective and constraints of a fully comprehensive mixed integer program. Notice that the arrival rate is now generalized to λ_{jt}, enabling a variable amount of data to arrive at different time periods. In addition to creating a variable U_{ijt} that corresponds to the amount of data transmitted from sensor j to gateway i at time t, three binary decision variables are introduced. The first decision variable, $Status_{jt}$, is used to represent whether sensor j is ON (1) or OFF (0), at time t. In order for a sensor to transmit data, the sensor j must be ON, $Status_{jt} = 1$, at time t and it must have access to a gateway which means one of

its data paths, X_{ijt}, must equal 1 and the rest must be $X_{ikt} = 0, \quad \forall k \neq j$. To account for the spike energy, the value of Z_{jt} is set $Z_{jt} = 1$ when the sensor j turns ON at time t after being OFF at time t-1, and remains 0 otherwise. It should be noted that the combination of the binary variables with the linear variables U_{ijt} in a single model, results in the mixed integer properties of the problem.

The mixed integer programming model is presented in Figure 4.2. Notice that the objective function of the model is to maximize the total data retrieved by the network given a maximum time T. This will result in creating a binary search process over time T, between feasible and non-feasible models with differing values of T.

$$Max \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{t=1}^{T} U_{ijt},$$ This objective maximizes the total data transmitted.

such that,

$$\sum_{j=1}^{J} X_{ijt} \leq 1, \quad \forall i \,\&\, \forall t,$$ Sensors are limited to access 1 gateway at a time.

$$Status_{jt} \geq \sum_{i=1}^{I} X_{ijt}, \quad \forall j \,\&\, \forall t,$$ Sensors must be ON to access a gateway.

$$Z_{jt} \geq Status_{jt} - Status_{j(t-1)}, \quad \forall j \,\&\, \forall t,$$ This decides the spike energy status of all sensors at time t.

$$U_{ijt} \leq \mu_{ij} X_{ijt} \quad \forall i \,\&\, \forall j \,\&\, \forall t,$$ Sensors cannot transmit more than the capacity of the link.

$$\sum_{i=1}^{I} U_{ijS} \leq \sum_{t=1}^{S} \lambda_{jt} - \sum_{i=1}^{I}\sum_{t=1}^{S-1} U_{ijt}, \quad \forall j \,\&\, \forall S = 1,2,...T,$$ Sensors cannot transmit more than what has been sensed.

$$\sum_{t=1}^{S} \lambda_{jt} - \sum_{i=1}^{I}\sum_{t=1}^{S} U_{ijt} \leq B_j, \quad \forall j \,\&\, \forall S = 1,2,...T,$$ Sensors cannot over fill the buffer.

$$e_{p,j}\sum_{t=1}^{T} Z_{jt} + \sum_{i=1}^{I}\sum_{t=1}^{T} e_{t,ij} U_{ijt} + e_{o,j}\sum_{t=1}^{T} Status_{jt} \leq e_{tot,j}, \quad \forall j,$$ Sensors cannot use more energy than the initial total energy.

$$Status_{jt} \in \{0,1\}, \forall j \,\&\, \forall t; \; X_{ijt}, Z_{ijt} \in \{0,1\}, \forall i \,\&\, \forall j \,\&\, \forall t; \; U_{ijt} \geq 0, \forall i \,\&\, \forall j \,\&\, \forall t.$$

Figure 4.2 The mixed integer programming model for bipartite sensor-gateway networks.

4.1 Solution of Small Networks

At first sight, the problem presented in the previous section has a lot of similarities to a Bipartite Matching Problem which has been dealt with in literature by using different types of polynomial-time algorithms [9]. In reality, however, the problem is much more complex, as sensors are contending for resources and buffer overflows are not allowed. This contention and finite storage resource creates a NP-Hard MIP problem, i.e., optimality can be achieved in an exponential number of steps based on the input, which encompasses the generalized knapsack problem [7]. Finding the globally optimal solution for the MIP model as a comprehensive solution approach is computationally prohibitive.

Due to the difficulty and time required to solve the MIP model, a very small network, 4 sensors and 2 gateways, was chosen for the base scenario topology. The sensors in this scenario were set up such that they were fully connected, meaning all sensors shared access to both gateways, and the same energy parameters as shown in Table 3.1 were considered for all sensors with the following assumptions:

1. The sensing energy consumption is assumed to be zero since it is consumes a negligible amount of energy constantly over time.
2. The buffer is fixed at 24 Kbits, which was determined from the single sensor behavior model to be a "reasonable" buffer size.
3. The energy is scaled down to 0.2 joules to control the size of the model.

The spiked curve in Figure 4.3 shows the transmission schedule of a single senor from a 4-2 network. The sensor, whose transmission schedule is shown, was capable of retrieving the

maximum amount of data over its maximum lifetime. Figure 4.3 shows the transmission schedule over the maximum possible lifetime of a sensor on one exemplary simple network that has 4 sensors and 2 gateways. The lines plotted in Figure 4.3 represent the transmission rate used by one of the sensors (regardless of which gateway the data is sent to), when the ON\OFF scheme and the Continuous ON scheme are used. Notice that by using the ON/OFF scheme, the sensor has a much longer lifetime, 61 time periods as compared to the 15 achieved by the sensor using the Continuous ON scheme. Further comparing the total amount of data retrieved via this sensor, which is the sum of the transmitted data over time, it is found to be 110 Kbits for the ON/OFF scheme – a 267% improvement over the Continuous ON scheme that is capable of collecting 30 Kbits. A key reason that the ON/OFF scheme is able to retrieve much more data given the same energy constraints is the use of buffer space, which allows transmission at full capacity almost every time the RF circuit is turned ON.

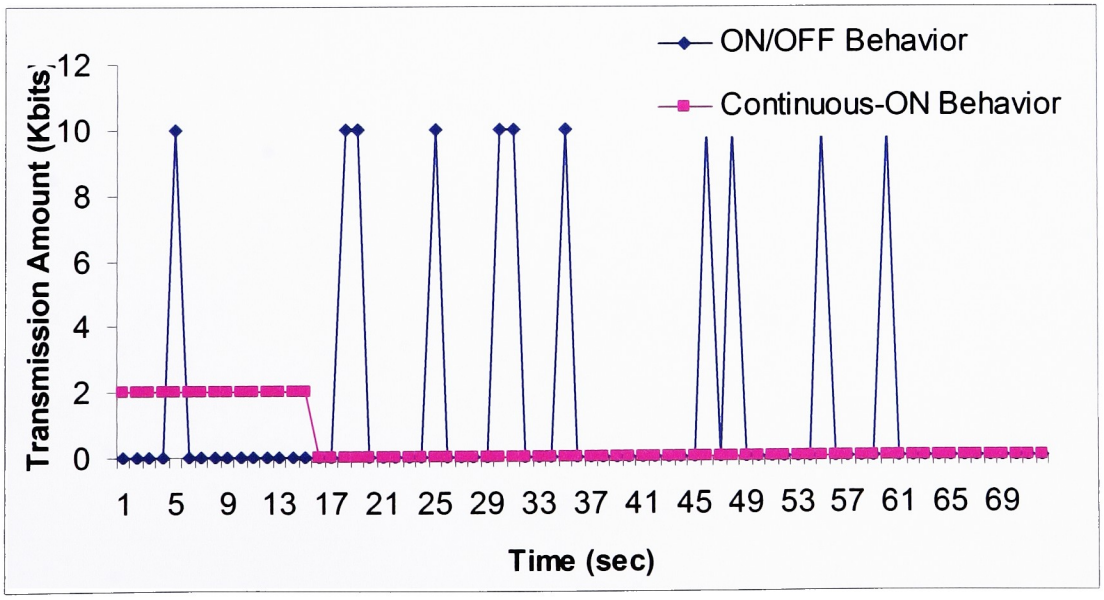

Figure 4.3 Data transmitted over time for one of the sensors on a 4-2 (Sensors to Gateways) bipartite network, when the ON/OFF scheme and the Continuous ON scheme are used.

Figure 4.4 displays the buffer usage over time, corresponding to the optimal ON/OFF transmission schedule that was plotted in Figure 4.3. Notice on two occasions (at time 15 and time 29), the sensor's buffer comes nearly full and is followed by two consecutive transmissions, which significantly reduces the buffer's content. This behavior does not exactly duplicate the ON/OFF behavior found analytically, however, it does provide evidence that the sensor is seeking to behave in the optimal fashion, i.e., full ON/OFF cycles. Full ON/OFF cycles may not be completed, however, due to contention in the network. Also notice that the buffer is not completely emptied before the sensor turns OFF. This behavior may be attributed to the sensor being incapable of completely empting its buffer during a single ON period unless it transmits below its maximum transmit capacity for at least one of its transmissions. As a result, rather than transmitting at less than full capacity, which is inefficient, the sensor buffers the data.

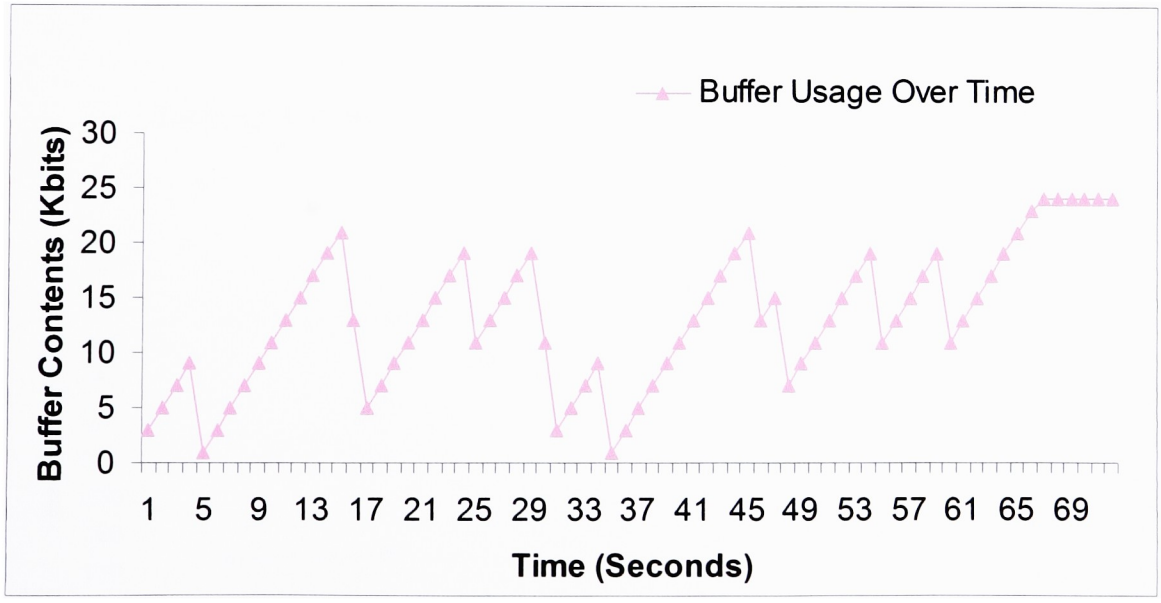

Figure 4.4 Buffer utilization over time for one of the base model sensors in a bipartite network demonstrating ON/OFF behavior.

4.2 Effect of Arrival Pattern

Wireless sensor networks may be used in many different applications, each of which may have a considerably different arrival patterns over time. For a given application, one possible arrival pattern may consist of periods in which a large amount of data is sensed followed by long inactive periods. The results above show that an ON/OFF scheme can achieve significant data retrieval improvements over the Continuous ON scheme, assuming all data arrives constantly. To determine how the arrival patterns would affect the amount of total data retrieved, the different arrival patterns show in Table 4.1 are considered for the same 4-2 network. The resulting optimal data retrieval amounts by the ON/OFF scheme and the Continuous ON scheme are plotted in Figure 4.5.

Trial	Arrival Pattern (Kbits)
Base	2-2-2-2-2-2-2-2-2-2-2-2-...
1	4-0-4-0-4-0-4-0-4-0-4-0-...
2	6-0-0-6-0-0-6-0-0-6-0-0-...
3	10-0-0-0-0-10-0-0-0-10-...
4	14-0-0-0-0-0-0-14-0-0-0-...
5	18-0-0-0-0-0-0-0-0-18-0-...

Table 4.1 Arrival Patterns with same average arrival rate (2 Kbits/Sec).

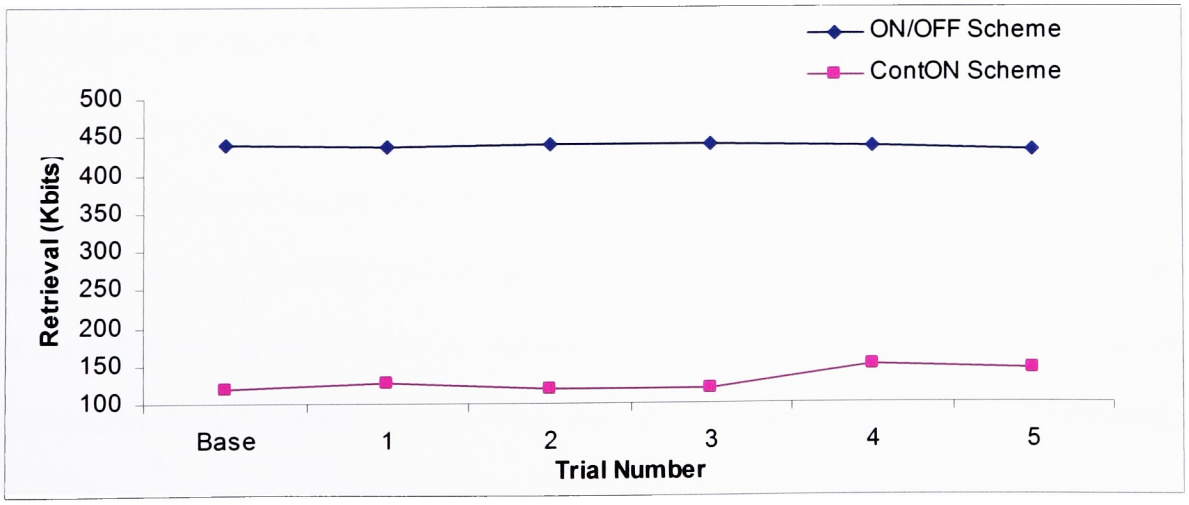

Figure 4.5 The amount of data retrieved by a 4-2 bipartite network with respect to the various arrival patterns.

As shown in the figure above, the amount of data that can be retrieved is relatively insensitive to the different traffic patterns considered. For the ON/OFF scheme, this is due to the buffer being capable of absorbing the arrivals. Therefore, as long as the amount of data that arrived in a single period does not exceed the buffer size, the transmission schedule won't change much, and hence, similar amounts of total data are retrieved. For the Continuous ON scheme, where data is transmitted immediately upon arrival[2], the total amount of data retrieved is equal to the total amount of data arrived throughout the network lifetime, with one exception when the data sensed in the last time period is larger than the maximum transmission capacity. The difference in total data retrieval due to the exception will be marginal as long as the sensor lifetime is much larger than the inactive period of the traffic pattern. This is indeed the case in the scenario considered in Figure 4.5. This set of results exhibits that, for at least the base scenario considered, the ON/OFF scheme provides robust performance with respect to fluctuations in the data arrival pattern, and consistently outperforms the Continuous ON scheme. An interesting extension related to this study is to consider stochastic arrival rates.

4.3 Effect of Buffer Size

Figure 4.3 and Figure 4.5 demonstrate the significant performance improvements of the ON/OFF behavior as compared to the Continuous ON behavior. These results however are based on a network in which each sensor has a fixed buffer size of 24 Kbits; thus the impact of the buffer size in these figures is unknown. Figure 4.6 contains a plot of the total data retrieved by the 4-2 bipartite network when one uses an increasing buffer size. Interestingly,

[2] In cases where more data arrives during a time period than can transmitted, the excess data is assumed to be buffered and transmitted during the subsequent time period(s).

the total data retrieval increases significantly as the buffer size increases when the buffer is small (less than 18 Kbits), and saturates to about 430-440 Kbits after that. Notice that in Figure 4.4 the actual buffer space utilized, i.e. the buffer size minus the residual buffered data, is calculated to be approximately 17- 20 Kbits, which is consistent with the buffer size required to retrieve the maximum amount of data from the optimal behavior section. Several other small networks were examined and the same performance trend with respect to the buffer size was observed. This leads to a logical question of whether the total data retrieval of 440 Kbits is indeed the most one can collect no matter how large the buffer size is. This question was answered using the analytical results from the optimal behavior section which determined this amount of retrieval to be very close to optimal.

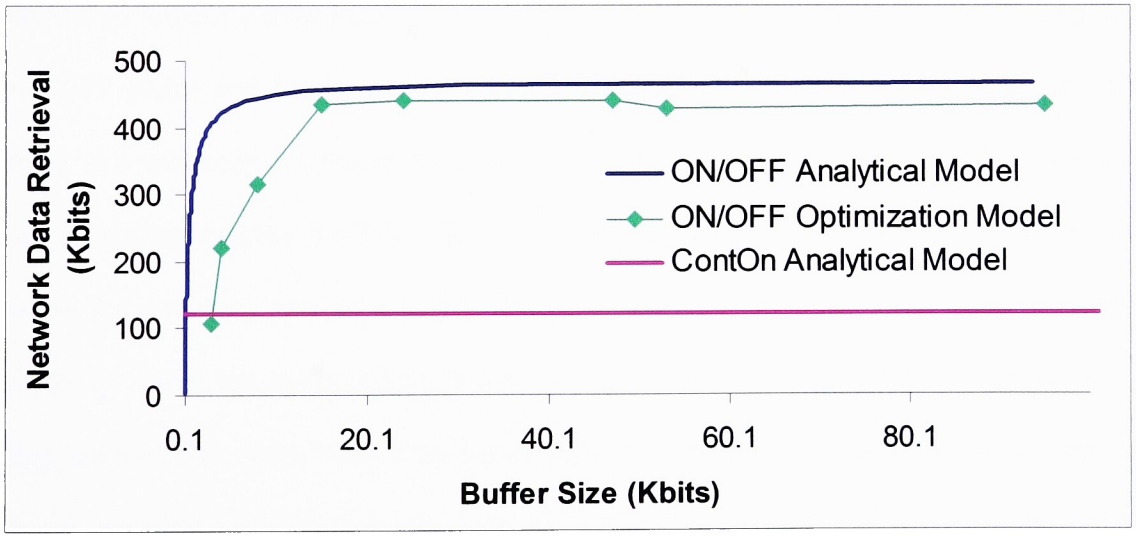

Figure 4.6 Data retrieved by a 4-2 bipartite network with respect to buffer size solved optimally for ON/OFF scheme, and analytically for both ON/OFF scheme and Continuous ON scheme.

From Figure 4.3 it was determined that the networked sensors are not capable of achieving full ON/OFF cycles due to contentions with other sensors. If, however, all of the sensors were able to transmit whenever they wanted to, the transmission schedule certainly would be

better than the networked sensor scenario in terms of the energy usage and the total data retrieval. A characteristic curve is drawn based on the optimal behavior, in Figure 4.6, to exhibit the performance that could have been achieved by the four sensors if there were no contention. The characteristic curve is derived based on equation (3.2) and equation (3.3) as done for Figure 3.2, yet here it is scaled up to correspond to four sensors. Also plotted in Figure 4.6 is the total data retrieval achievable by the Continuous ON scheme as a reference.

Many interesting observations can be made from Figure 4.6. Foremost, one shall note that, as long as the buffer size is not too small, the total data retrieval by the four sensors in reality (with contention) is very close to what they could have retrieved if there were no contention, i.e., the best they can do. In fact, by examining the transmission schedule of the sensors, the contention among the sensors is found to be severe (sensors frequently switch between ON and OFF states and hardly utilize the full transmission capacity) when the buffer size is small, and gradually lessens as the buffer size increases. Increasing the buffer size reduces the contention because it allows the sensors to stay OFF longer, allowing other sensors to access the gateway, and in a sense, alternates the transmissions. The same performance trends, as above, are again observed when considering other simple bipartite networks. This suggests that one needs only a moderate buffer to retrieve an amount of data close to the largest possible retrieval amount even for contending networked sensors. The analytical results seem to serve as a tight bound and a good approximation to what networked sensors can achieve using an ON/OFF scheme.

4.4 Effect of Network Service Capacity

The total network retrieval capability with respect to the maximum network service capability per second is displayed in Figure 4.7 for a scenario which had five fully connected sensors and three gateways. Maximum network service capability per second is defined as the maximum amount of data that can be transmitted out of the network per time period, i.e., the total number of gateways times the maximum transmission capability of the sensors. In each of the trials, the maximum network service capacity was adjusted by increasing the maximum transmission capacity of each sensor by one Kbit. All of the same energy consumption parameters as the previous 4-2 models are again considered except for the total energy of the sensors, which was increased to 0.5 Joules. Also, to ensure that the effect of sensor contention is demonstrated (at least for the low maximum network service capacity levels) the buffer was reduced to 12 Kbits, and the arrival rate was increased to 5 Kbits per second.

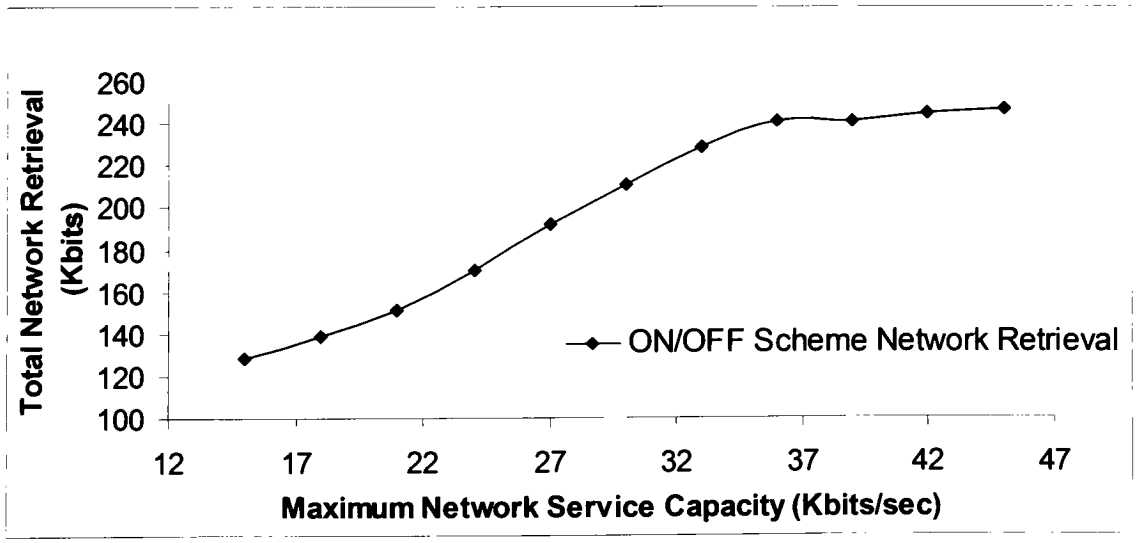

Figure 4.7 Network retrieval capability of a 5-3 bipartite network as a function of the network's maximum retrieval capability per second.

In Figure 4.7, when the service capacity is low (15-18 Kbits) the sensor is capable of retrieving 128 Kbits - 139 Kbits which is very low compared to the expected retrieval, ~250 Kbits, which was calculated by the optimal behavior model. This low retrieval is due to the large arrival rate and the low service capacity requiring the sensor to alternate every other transmission in order to keep its buffer from over flowing. When the sensor's status over time was investigated, it was determined that a Continuous ON scheme was actually utilized due to the frequency of the sensor's transmissions, i.e., every other second. Once the total service capacity reaches 20 Kbits, the ON/OFF behavior is utilized by the sensor and results in a linear increase in total network retrieval before leveling off once the total service capacity reaches the mid thirties. Increasing the total network service capacity no longer increases the total amount of data retrieved because the total network retrieval is now constrained by the buffer size. Running the same scenario yet with larger buffer sizes results in the same curve except that it does not level off until the total network service capacity reaches a higher amount.

4.5 Effect of Network Connectivity

Network connectivity refers to the number of gateways that each sensor is capable of transmitting to. In networks covering a small area, it is feasible that each sensor would be capable of transmitting to every data collection gateway; this type of connectivity will be referred to as "fully connected." Along with a fully connected network, Figure 4.8 shows two additional networks within which sensors may not be able to reach all gateways.

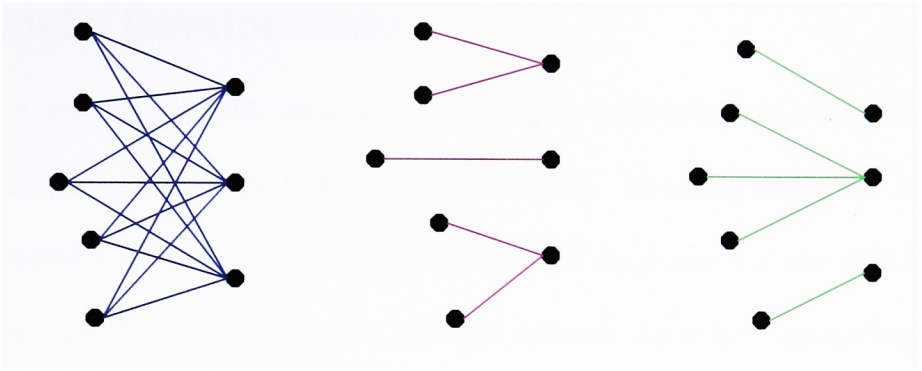

Figure 4.8 From the left, a fully connected network, a (2,2,1) network, and a (3,1,1) network.

Figure 4.9 shows the total data retrieval by the afore-mentioned three networks as the individual sensor sizes increase. The separations between the plots in Figure 4.5 illustrates the impact of the different connectivity levels as well as the effectiveness of even a relatively small buffer, i.e., 18 (Kbits), in eliminating this difference. Moreover, this plot demonstrates that a network that is not fully connected is capable of retrieving the same amount of data as a network that is fully connected. This is a significant finding as it suggests that a network's connectivity can be reduced, so as to reduce the difficulty of coordinating the sensors, yet the network may still be capable of achieving its maximum capability.

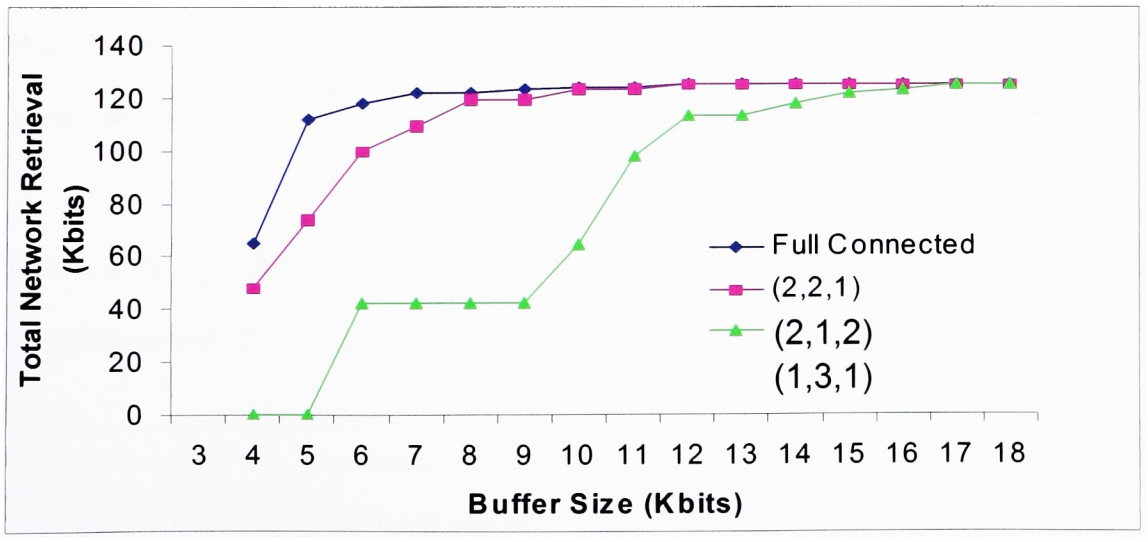

Figure 4.9 Total network retrieval with respect to buffer size for networks with differing connectivity levels.

5. Heuristic Development

In Chapter 4, the optimal behavior of a sensor in a network environment was determined and many key characteristics of this behavior were discussed. The term "behavior" refers to how a sensor schedules its RF circuits to turn ON, turn OFF, and transmit over time in a network environment. Ideally, each sensor in a physical network could be preprogrammed with its optimal behavior; yet this cannot be done due to the difficulty of solving the mixed integer problem for even small networks[3]. More importantly, this predetermined behavior requires a prior knowledge on the data arrivals and, therefore, is infeasible in practice where hard-to-predict sensed events happen in real time. Consequently, a preliminary heuristic has been developed to mimic the optimal sensor behavior. The proposed heuristic consists of two sets of criteria that will be used by the sensors and the gateways[4] to make decisions on the transmission and the receiving behavior, respectively, in an on-line fashion. The idea is to have the sensors independently decide when to turn the RF circuit ON and request for transmission, while each of the gateways will decide which requesting sensor it will grant access to. The goal of this distributive process is to have the network achieve close to the optimal overall capability as exhibited in Chapter 4. The flow charts of the two decision criteria are shown in Figure 5.1.

Several decision criteria determine the sensor behavior (the flow chart on the left of Figure 5.1) and how well the heuristic performs. First, a sensor will decide when to "Turn ON?"

[3] An optimal solution for a 4-2 network with the parameter T =50 was not found by the CPLEX mixed integer optimizer after 10.23 hours.

[4] The heuristic developed in this thesis assumes a 2-level hierarchical network, where the sensors will be sensing and transmitting data while the gateways will be only receiving data. A straightforward generalization of the heuristic is to implement the two set of decision criteria on each sensor so that a sensor will behave as both a transmitter and a receiver as it may be in practice.

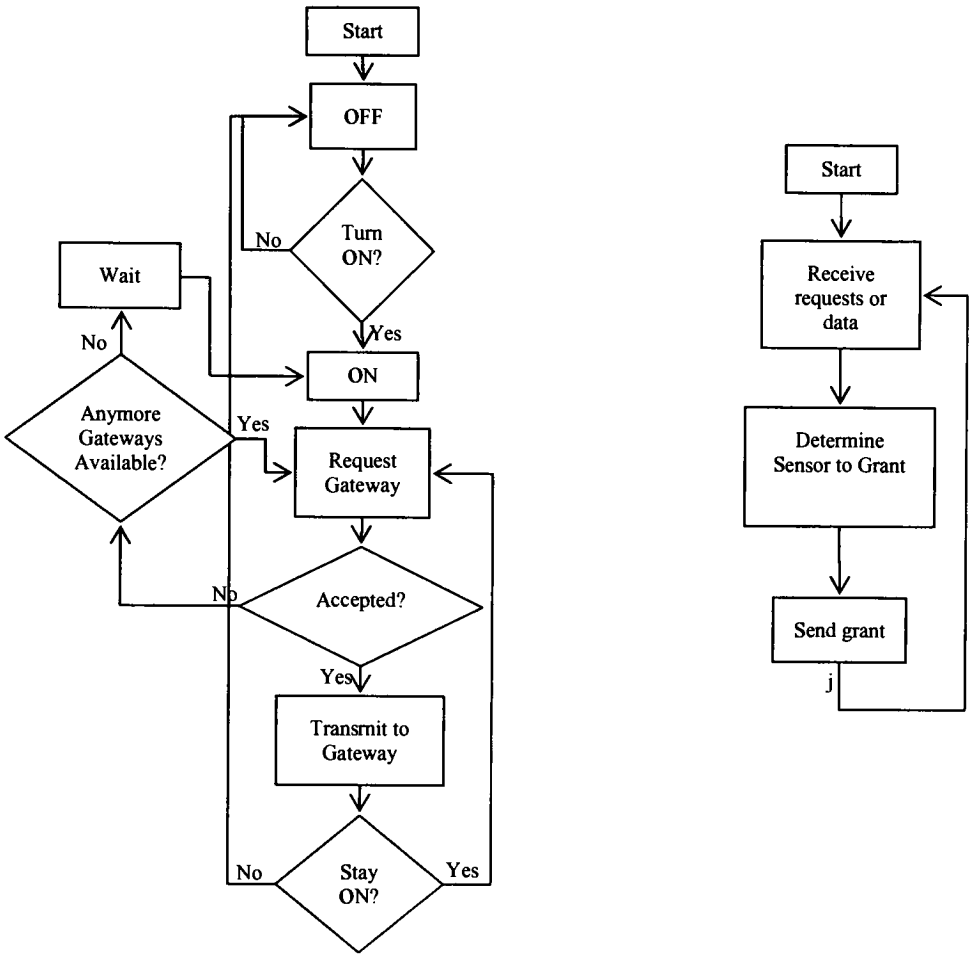

Figure 5.1 Sensor heuristic flow chart (left) and gateway heuristic flow chart (right).

(the first decision box) based on the buffer utilization, i.e., how full the buffer is. Since the sensor is OFF when it reaches this block, the heuristic is designed to remain in the OFF state so as to conserve as much energy as possible. Only when the buffer's utilization has surpassed the Upper Buffer Limit (UBL), i.e., the buffer will soon be full, will the sensor turn ON and begin requesting access to a gateway. Similarly, the decision block, "Stay ON?," is designed to keep the sensor in the ON state as long as possible so as to distribute the spike energy, consumed when turning from OFF to ON, over the maximum number of transmitted bits. It is also necessary to stay ON so that the buffer may continue to be emptied. Once the

buffer is nearly empty, indicated by examining whether the buffer utilization has fallen below the Lower Buffer Limit (LBL), the sensor will stop requesting access to gateways and switch to the OFF mode. The values of the UBL and LBL are determined based on the average arrival rate of the sensor and the maximum transmission capacity of the sensor, respectively.

If the decision is "yes" for either the "Stay ON?" or "Turn ON?" blocks, the sensor will determine the highest priority neighboring gateway, and send it a transmission request. The highest priority gateway is the gateway that is expected to cause the sensor consuming the least amount of energy if the sensed data were transmitted to it. Determining the highest priority gateway and sending it a request are the tasks performed by the sensor in the "Request Gateway" block. The highest priority gateway of sensor j is the gateway i that has the lowest value of $cStat_{ij}$ based on the following equation.

$$cStat_{ij} = U_{ij} * e_{t,ij} + (B_j - U_{ij}) * e_{t,-j} \qquad \forall i \,\&\, \forall j \qquad (5.1)$$

The variable $cStat_{ij}$ estimates the amount of energy that will be consumed by sensor j, if it completely empties its buffer, B_j (Kbits), in a single ON time period and makes its transmission of the first, U_{ij} Kbits to gateway i, with the rest equally transmitted to all neighboring gateways. The parameter $e_{t.ij}$ is the energy consumed by transmitting one Kbit of data from sensor j to gateway i, and $e_{t,-j}$ represents the average energy consumption per Kbit transmitted by sensor j over all neighboring gateways. This process of determining the highest priority gateway and sending it a request is performed repetitively until sensor j is granted access or until no more gateways are available. Notice that if the sensor is not

granted after requesting access to all of its neighboring gateways, the sensor will back off for a short period of time and begin requesting again starting with its highest priority gateway.

The decision of whether the sensor is granted access to a gateway takes place in the "Accepted?" decision block. This decision is actually made by the gateway. In fact, the sensor will wait for the gateway to send back a "grant" acknowledgement, and if received, the senor will transmit data until the utilization of the buffer falls below the LBL, at which time the sensor will turn off. The actual decision of accepting the sensor performed by the gateway is explained subsequently.

On the right side of Figure 5.1 above, the flow chart for the heuristic residing at the gateway is illustrated. The purpose of this part of heuristic is to allow only a single sensor j to transmit to a particular gateway i at any point in time. In addition, the total channel access allotted to sensor j over time, by all of its neighboring gateways combined, must be sufficient to ensure that only a minimal amount of data is lost. Since the sensors determine when they should begin to transmit without knowing other sensor's status, each gateway needs to choose the sensor that is expected to lead to more efficient energy consumption. This decision is made by comparing the values of $tStat_{ij}$ based on the following equation for each sensor j that have requested access to gateway i.

$$tStat_{ij} = \frac{C_j}{q_{-j} - \lambda_j} \qquad \forall i; \forall j \tag{5.2}$$

The sensor j with the largest $tStat_{ij}$ will be granted for access to gateway i, since it is the sensor that will potentially require the most time to completely empty its buffer, hence will

have the longest ON period to disperse the spike energy. The parameter, C_j (Kbits), is the current buffer utilization, q_{-j} (Kbits/sec) is the average service capacity of sensor j, to all its neighboring sensors, and λ_j (Kbits/sec) is the average arrival rate for sensor j.

The heuristic described above essentially pairs up sensors and gateways, so that each sensor once turned ON, will stay ON for the longest time possible. Notice that in the real world this process of pairing up may take a few signaling exchanges among the sensors and gateways, and therefore consumes time and energy. The proposed heuristic, however, assumes such process consumes much less time and energy than those used for transmitting the sensed data, and therefore ignores both factors when conducting the simulations. Consequently, whenever there is at least one sensor request, the simulator will determine the "maximal match" between the requesting sensors and the gateways.

5.1 Heuristic Capability Assessment through Simulation

To investigate the effectiveness of the heuristic above, the behavior of a sensor in a network controlled by the heuristic and a sensor whose behavior was found optimally by the MIP model, were compared. To make this comparison, a 4-2 network was simulated and run with all of the sensors having identical power consumption parameters, buffer sizes, data arrival rates and service capacities to those sensors used to generate the results shown in Figure 4.4. The results of this comparison are discussed below.

Figure 5.2 demonstrates the buffer utilization over time for a sensor in a 4-2 network based on the MIP model and for a sensor in a 4-2 simulated network controlled by the heuristic. In

Figure 5.2, notice the similar behavior exhibited by one of the sensors under the two experimental setups. In fact, the sensor, under both setups, transmits a total of 11 times, and transmits at full capacity for every transmission, and as a result, retrieves a total of 110 Kbits of data.

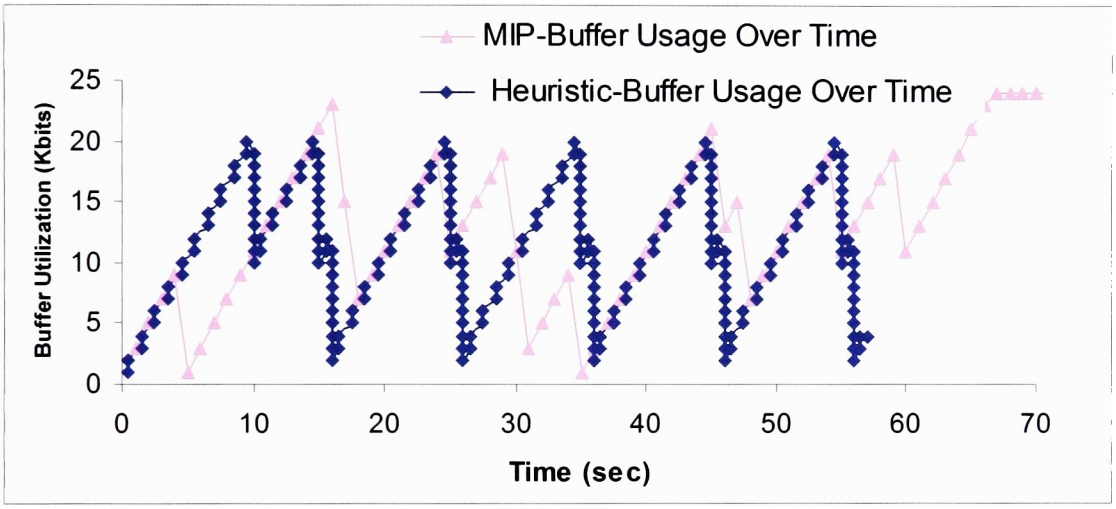

Figure 5.2 Buffer utilization over time of a single sensor in a 4-2 bipartite network based on the MIP model and the simulated network under heuristic control.

The other network sensors under the two setups also exhibit similar results, but may not achieve exactly the same data retrieval, which is obviously expected. The overall performance of the respective networks is also similar, where a total of 440 Kbits is retrieved under the MIP model, and it is 420 Kbits for the simulated network based on the heuristic. The full ON/OFF behavior seems to be more prominent in the plot for the simulated sensor. One should however note that the buffer utilization for the simulated sensor never exceeds 20 Kbits, while it can reach 24 Kbits – the full buffer size, under the MIP model. This discrepancy between the two setups leads to the slightly higher performance achieved using the optimal network solution.

The above results show an example of how a sensor behavior, similar to that of the optimal behavior can be reproduced dynamically by a sensor controlled by a heuristic. It is also interesting to investigate whether the performance trends of the wireless sensor network previously exhibited in Chapter 4 can be also observed for the simulated network. In particular, this thesis will focus on investigating the performance impact due to the size of the buffer at each sensor. For comparison, this set of results will be plotted in Figure 5.3, along with the data points already shown in Figure 4.6. As expected, the network based on the proposed heuristic exhibits a similar performance trend as the buffer size increases. More importantly, the total data retrievals under three different setups are relatively close, especially when buffer size is big enough to provide close to optimal data retrieval – recall the discussion in Section 4.3.

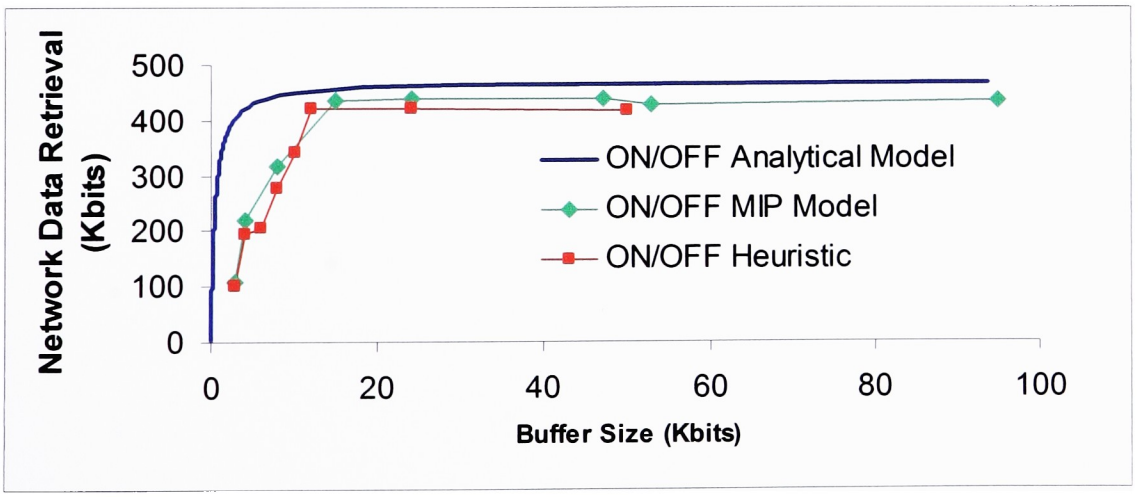

Figure 5.3 Network data retrieval with respect to the buffer size based on the heuristic, the optimal behavior analytical model, and the MIP model.

5.2 Long Term Heuristic Goals

The preliminary results achieved by the heuristic are very promising. There are, however, adjustments that can be made and factors that need to be incorporated into the heuristic before it can be implemented to control actual wireless sensing devices. To make the heuristic more realistic, it will need to account for the time and energy it takes for the sensors to transmit requests and wait for grants to be received. In addition, collisions on the control message also need to be accounted for. One possible solution would have the gateways transmitting a confirmation message after every successfully transmitted message by a sensor. The final adjustment to the heuristic will be to extend the use of the criteria to serve a multi-hop network. In this situation, a sensor will need to decide when to turn ON, turn OFF, transmit, and receive over time. Even with these small inefficiencies, the heuristic does show a promising avenue to reproduce the optimal behavior in sensors and to coordinate a network of sensors in a nonbiased fashion.

6. Conclusions and Future Work

This thesis has investigated various models to determine and maximize the capability of a WSN. These models include an optimal behavior model, a mixed integer programming model, and a simulated heuristic model. Each of these models provided insightful results, which lead to further research opportunities.

The analytical optimal behavior model, which assumes no contention among sensors provides closed form results to exhibit not only the achievable maximum capability, but also how to achieve such maximum capability. In fact, under an ideal situation, it is suggested that a sensor should fully utilize its buffer so that it may turn OFF as long as possible to conserve energy, and utilize its full transmission capability when turned ON. The use of buffer and an ON/OFF scheme is clearly advantageous as compared to keeping the sensor Continuously ON, when realistic energy parameters, including spike energy, are considered.

The mixed integer model provides a novel approach to reflect networked sensor contention over time in the discrete time regime. The model determines the actual ON/OFF/transmission schedule of a sensor at every time instance. This allows one to analyze the impact of network contention on overall data retrieval. In particular, the performance impacts due to buffer size, traffic pattern, service capacity, and connectivity were investigated. It was concluded that increasing the size the buffer size can effectively reduce the contention in a network, especially when the buffer size is small. A similar trend was noticed when one increases the service capacity. The two factors, buffer size and service capacity, however, are interrelated, which affects the effectiveness of each other in terms of the amount of data retrieval improved per unit buffer and transmission rate added,

respectively. When the buffer size is too small, hence becoming the performance bottleneck, it helps tremendously to add more buffer space, but not much is gained by increasing the sensor's transmission rate. A similar phenomenon can be observed when the transmission capacity is limited. Finally, the overall data retrieval by a WSN, based on experimental results, seems to be relatively insensitive to the traffic pattern and network connectively, as long as these factors are "reasonable."

A heuristic has also been proposed to investigate how, in reality, a WSN may achieve the maximum capability as exhibited by the MIP model. The proposed heuristic consists of a set of decision criteria to be used for sensors and gateways to make distributed decisions about their transmission schedule over time. A WSN under heuristic control displays similar behavior and network capabilities as compared to the optimal behavior and network capability derived based on the MIP model. The heuristic also provides a foundation for research dedicated to developing an energy efficient MAC protocol, for WSN's.

The research discussed in this thesis is the foundation for many potential areas of future work. A logical next step is to incorporate one or many new complexities into the MIP model:

- Spatial and Temporal Effects: This research has mainly considered the capability of a WSN over time. Connecting the sensors with a physical location and determining the capability of the network over time may have different results.
- Relaxations of Sensor/Gateway Functionality: A number of assumptions have been made in the model, i.e., gateways and sensors can only communicate with a

single partner. The repercussions of relaxing some of these restrictions could be investigated.

- Multiple Level Bipartition: At first glance the assumption of having two bipartite layers in the network (one for sensors and one for gateways) seems very restrictive. In reality this could be extended to multiple layers under certain conditions (i.e. any graph with no odd cycles). Nevertheless, to generalize the results, a more generic topology of the network could be considered to show that either the current results hold true, or that a new approach can be developed from the simple bipartite case.
- Large Base Line models: To compare any new heuristic approaches in the future a much richer set of base line models needs to be obtained. Given the computational complexity of the problem, a methodology that can determine the optimal or close to optimal solution with a reasonable performance guarantee, for much larger problems, needs to be created. Generic procedures such as Simulated Annealing, Tabu Search and Genetic Algorithms could be considered as part of this process.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "Wireless sensor networks: a survey," Computer Networks, vol. 38, pp. 115-125, 2002.

[2] J.-H. Chang and L. Tassiulas, "Energy Conserving Routing in Wireless Ad-Hoc Networks," Proc. IEEE INFOCOM'00, Mar. 2000.

[3] D. Culler, J. Hill, "Mica: A Wireless Platform for Deeply Embedded Networks," IEEE Micro., vol. 22(6), Nov/Dec 2002, pp 12-24.

[4] Energizer: Technical Information. {Cited September 2, 2003}. Available via http://data.energizer.com.

[5] ILOG, "ILOG CPLEX 8.0: Getting Started," France, July 2002.

[6] K. Kar, M. Kodialam, T. V. Lakshman, and L. Tassiulas "Routing for Network Capacity Maximization in Energy-Constrained Ad-hoc Networks," Proc. IEEE INFOCOM'03, April 2003.

[7] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Revised edition Nov. 1990.

[8] R. Min and A. Chandrakasan, "Energy-Efficient Communication for Ad-Hoc Wireless Sensor Networks," 35th Asilomar Conference on Signals, Systems, and Computers, vol. 1, November 2001, pp. 139-143.

[9] C. H. Papadimitriou, *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, January 1982.

[10] J. Pottie and W. J. Kaiser, "Embedding the Internet Wireless Integrated Network Sensors," Communications of the ACM, vol. 43, no. 5, pp. 51-58, May 2000.

[11] J. Rabaey, J. Ammer, J. L. DaSilva Jr., and D. Patel, "PicoRadio: Ad-hoc Wireless Networking of Ubiquitous Low-Energy Sensor/Monitor Nodes," Proc. WVLSI, April 2000.

[12] B. Radunovic and J.Y. Le Boudec, "A Unified Framework for Max-Min and Min-Max Fairness with Applications," Proc. Allerton Conference, Oct., 2002.

[13] V. Raghunathan, C. Schurgers, S. Park, and M.B. Srivastava, "Energy-Aware Wireless Microsensor Networks," IEEE Signal Processing Magazine, vol. 2, March 2002.

[14] C. Schurgers and M. B. Srivastava, "Energy Efficient Routing in Wireless Sensor Networks," Proc. MILCOM'01, Vienna, VA, October, 28-31, 2001.

[15] A. Srivastava, J. Sobaje, M. Potkonjak, and M. Sarrafzadeh, "Optimal Node Scheduling for Effective Energy Usage in Sensor Networks," Proc. IEEE Workshop on Integrated Management of Power Aware Communications Computing and Networking, 2002.

[16] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, "A Taxonomy of Wireless Micro-Sensor Network Models," ACM Mobile Computing and Communications Review (MC2R), vol. 6, no. 2, April 2002.

[17] C. Ulmer, "Wireless Sensor Networks," Georgia Institute of Technology, presentation.

[18] A.Y. Wang, S.H. Cho, C.G. Sodini, and A.P. Chandrakasan, "Energy Efficient Modulation and MAC for Asymmetric RF Microsensor Systems," Proc. ICASSP 2001, May 2001.

[19] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed Energy Conservation for Ad Hoc Routing," Proc. Mobicom, 2001, pp. 70-84.

[20] W. Ye, J Heidemann, D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," Proc. IEEE INFOCOM'02, June 2002.

[21] G. Zussman and A. Segall, "Energy Efficient Routing in Ad Hoc Disaster Recovery Networks," Proc. IEEE INFOCOM'03, April 2003.

Appendix

```
**************************************************************************
*Simulation of heuristic model developed using ProModel Simulation Software
*This model is capable of simulating 13 sensors and 4 gateways; however, the current
*number of sensors and gateways are set at 4 and 2, respectively.
**************************************************************************
Time Units:                Seconds
Distance Units:            Meters
Initialization Logic:      int j = 1
                           int i = 1

                           WHILE j <= numSensor DO
                           {
                           i = 1
                           NumArrivalj[j] = 2
                           SSj[j] = .001
                           ESj[j] = .2
                           COj[j] = .012
                           WHILE i <= numGateway DO
                           {
                           SGi[i] = 3
                           EGi[i] = 2500
                           qji[j,i] = 10
                           CSji[j,i] = .0005

                           i = i + 1
                           }
                           j = j + 1
                           }
                           //NumArrivalj[1] = 8
                           //NumArrivalj[2]=8

                           //ESj[1]= .8
                           //ESj[2]=.8
                           EG1 = EGi[1]
                           EG2 = EGi[2]

                           ES1 = ESj[1]
                           ES2 = ESj[2]
                           ES3 = ESj[3]
                           ES4 = ESj[4]
                           ES5 = ESj[5]
                           ES6 = ESj[6]
                           ES7 = ESj[7]
                           ES8 = ESj[8]
```

ES9 = ESj[9]
ES10 = ESj[10]
ES11 = ESj[11]
ES12 = ESj[12]

Termination Logic:

****
Locations
****

Name	Cap	Units	Stats	Rules	Cost
Sensor1	50	1	Time Series Oldest, , First		
Sensor2	50	1	Time Series Oldest, ,		
Sensor3	50	1	Time Series Oldest, ,		
Sensor4	50	1	Time Series Oldest, ,		
Sensor5	5	1	Time Series Oldest, ,		
Sensor6	24	1	Time Series Oldest, ,		
Sensor7	24	1	Time Series Oldest, ,		
Sensor8	24	1	Time Series Oldest, ,		
Sensor9	24	1	Time Series Oldest, ,		
Sensor10	24	1	Time Series Oldest, ,		
Sensor11	24	1	Time Series Oldest, ,		
Sensor12	24	1	Time Series Oldest, ,		
Sensor13	24	1	Time Series Oldest, ,		
Gateway1	inf	1	Time Series Oldest, ,		
Gateway2	inf	1	Time Series Oldest, ,		
Gateway3	inf	1	Time Series Oldest, ,		
Gateway4	inf	1	Time Series Oldest, ,		
DummyArrival	3	1	Time Series Oldest, ,		
Unrelated_Location	3	1	Time Series Oldest, ,		
ArrivalS1	3	1	Time Series Oldest, ,		
ArrivalS2	3	1	Time Series Oldest, ,		
ArrivalS3	3	1	Time Series Oldest, ,		
ArrivalS4	3	1	Time Series Oldest, ,		
ArrivalS5	3	1	Time Series Oldest, ,		
ArrivalS6	3	1	Time Series Oldest, ,		
ArrivalS7	3	1	Time Series Oldest, ,		
ArrivalS8	3	1	Time Series Oldest, ,		
ArrivalS9	3	1	Time Series Oldest, ,		
ArrivalS10	3	1	Time Series Oldest, ,		

```
ArrivalS11      3  1    Time Series Oldest, ,
ArrivalS12      3  1    Time Series Oldest, ,
ArrivalS13      3  1    Time Series Oldest, ,
Man1Arrival      1  1    Time Series Oldest, ,
Man2Arrival      1  1    Time Series Oldest, ,
Man3Arrival      1  1    Time Series Oldest, ,
MenExit         3  1    Time Series Oldest, ,
Loc2            1  1    Time Series Oldest, ,


************************************************************************
Entities
************************************************************************

 Name      Speed (mpm)  Stats       Cost
 ---------- ------------ ---------- ------------
 Bit1      50          Time Series
 Bit2      50          Time Series
 Bit3      50          Time Series
 Bit4      50          Time Series
 Bit5      50          Time Series
 Bit6      50          Time Series
 Bit7      50          Time Series
 Bit8      50          Time Series
 Bit9      50          Time Series
 Bit10     50          Time Series
 Bit11     50          Time Series
 Bit12     50          Time Series
 Bit13     50          Time Series
 Dummy     50          Time Series
 Man1      50          Time Series
 Man2      50          Time Series
 Man3      50          Time Series


************************************************************************
*                      Path Networks
************************************************************************

 Name     Type       T/S             From     To       BI   Dist/Time  Speed Factor
 -------- ---------- ---------------- -------- -------- ---- ---------- ------------
 DummyNet Passing    Speed & Distance N1       N2       Bi   0          1


************************************************************************
*                      Interfaces
```

```
**************************************************************************

 Net       Node      Location
 ---------- ---------- ------------------
 DummyNet   N1        DummyArrival
            N2        Unrelated_Location


**************************************************************************
*                        Resources
**************************************************************************

                  Res    Ent
 Name     Units Stats    Search  Search Path      Motion       Cost
 -------- ----- -------- ------- ------ ---------- ------------ ------------
NONE


**************************************************************************
*                        Processing
**************************************************************************

                          Process                  Routing

 Entity  Location          Operation          Blk  Output  Destination      Rule       Move
Logic
 -------- ------------------ ------------------  ---- -------- ------------------ ---------- ------------
 Bit1    ArrivalS1

                                    1   Bit1    Sensor1          FIRST 1    MOVE FOR .5
 Bit2    ArrivalS2                    1   Bit2    Sensor2          FIRST 1    MOVE FOR .5
 Bit3    ArrivalS3                    1   Bit3    Sensor3          FIRST 1    MOVE FOR .5
 Bit4    ArrivalS4                    1   Bit4    Sensor4          FIRST 1    MOVE FOR .5
 Bit5    ArrivalS5                    1   Bit5    Sensor5          FIRST 1    MOVE FOR .5
 Bit6    ArrivalS6                    1   Bit6    Sensor6          FIRST 1    MOVE FOR .5
 Bit7    ArrivalS7                    1   Bit7    Sensor7          FIRST 1    MOVE FOR .5
 Bit8    ArrivalS8                    1   Bit8    Sensor8          FIRST 1    MOVE FOR .5
 Bit9    ArrivalS9                    1   Bit9    Sensor9          FIRST 1    MOVE FOR .5
 Bit10   ArrivalS10                    1   Bit10   Sensor10         FIRST 1    MOVE FOR .5
 Bit11   ArrivalS11                    1   Bit11   Sensor11         FIRST 1    MOVE FOR .5
 Bit12   ArrivalS12                    1   Bit12   Sensor12         FIRST 1    MOVE FOR .5
 Dummy   DummyArrival      HoldMen = 0
                    Tnow = Tnow + 1     1   Dummy   Unrelated_Location FIRST 1
MOVE ON DummyNet
```

```
Dummy   Unrelated_Location int terminate = 1

                WHILE terminate <= numSensor DO
                {
                    IF ESj[terminate] <= 0.014 AND StatusS[terminate] = 0 OR
ESj[terminate] <= .013  THEN
                    {
                            //View "Statistics View"
                            Stop "One of the sensors has drained all of its energy"
                    }
                    terminate = terminate + 1
                }

                terminate = 1

                WHILE terminate <= numGateway DO
                {
                    IF EGi[terminate] <= 0 THEN
                    {
                    View "Statistics View"
                    Stop "One of the Gateways has drained all of its energy"
                    }
                    terminate = terminate + 1
                }
                int promptNum = 1
                int i = 1
                int j = 1
                int TRUE = 1
                int FALSE = 0
                int NA = -1
                HoldMen = 1
                //PROMPT "Prompt num is ", promptNum

                // Start Routine 1
                WHILE j <= numSensor Do
                {
                    i = 1
                    WHILE i <= numGateway DO
                    {
                            CONji[j,i] = 1

                            i = i + 1
                    }
                    j = j +1

                }// End Routine 1
```

```
//CONji[1, 2] = 0
//CONji[2, 2] = 0
//CONji[3, 2] = 0
//CONji[10, 1] = 0
//CONji[11, 1] = 0
//CONji[12, 1] = 0



int count = 1
int currPriority = 1

i = 1
j = 1
int temp = 0
int aveCStemp = 0
int numLinks = 0

// This loop determines the value of aveqj[j] for all j as well as aveCS
//Start Routine 2
WHILE j <= numSensor Do
{

    i = 1
    numLinks = 0
    temp = 0
    aveCStemp = 0
    WHILE i <= numGateway DO
    {
            temp =  temp + (CONji[j,i] * qji[j,i])
            aveCStemp = aveCStemp + (CONji[j,i] * CSji[j,i])
            IF CONji[j,i] <> 0 THEN
            {
                    numLinks = numLinks + 1
            }
            cStatji[j,i] = -1
            i = i + 1
    }
    aveqj[j] = temp / numLinks
    aveCSj[j] = aveCStemp / numLinks
    tStatj[j] = -1  // Initializing all tStatValues to -1 so they all start with
lowest value possible
    j = j +1
} // End Routine 2
i = 1
j = 1
```

```
//Start Routine 3
WHILE j <= numSensor Do
{

    tStatj[j] =  CONTENTS(Loc(j), ENT(j)) /  (aveqj[j]   NumArrivalj[j])
// NumArrivalj
    tStatTemp[j] = tStatj[j]
    tStatIdOrder[j] = j

    j = j + 1
}// End Routine 3


//_____Gateway Priority List
Routine_____
i = 1
j = 2
int tempt
int tempID

IF tieBreak1 = 0 THEN
{
    WHILE i <= (numSensor - 1)  DO
    {
    j = 2

        WHILE j <= numSensor DO
        {
            IF tStatTemp[j] > tStatTemp[j-1] THEN
            {
                tempt = tStatTemp[j]
                tempID = tStatIdOrder[j]

                tStatTemp[j] = tStatTemp[j-1]
                tStatIdOrder[j] = tStatIdOrder[j -1]

                tStatTemp[j-1] = tempt
                tStatIdOrder[j-1] = tempID
            }
        j = j + 1
        }

    i = i + 1
    tieBreak1 = 1
```

```
    }
}
ELSE
{
    WHILE i <= (numSensor - 1)  DO
    {
    j = 2

            WHILE j <= numSensor DO
            {
                    IF tStatTemp[j] >= tStatTemp[j-1] THEN
                    {
                            tempt = tStatTemp[j]
                            tempID = tStatIdOrder[j]

                            tStatTemp[j] = tStatTemp[j-1]
                            tStatIdOrder[j] = tStatIdOrder[j -1]

                            tStatTemp[j-1] = tempt
                            tStatIdOrder[j-1] = tempID
                    }
            j = j + 1
            }

    i = i + 1
    tieBreak1 = 0
    }
}

i = 1
j = 1

WHILE i <= numGateway   DO
{

    j = 1
    WHILE j <= numSensor DO
    {
            IF CONji[j,i] = -1 THEN
            {
                    GatewayPrioorityip[i,j] = -1
            }
            ELSE
            {
                    GatewayPrioorityip[i,j] = tStatIdOrder[j]
            }
```

```
                j = j + 1
            }
            i = i + 1
        }
        //_____________End Gateway Priority List Routine

        // This routine builds the Vji values for each sensor to each gateway it is
connected to
        i = 1
        j = 1

        WHILE j <= numSensor DO
        {
            i = 1
            WHILE i <= numGateway DO
            {
                IF CONji[j,i] <> 0 THEN
                {

                    IF Contents(LOC(j)) > qji[j,i] THEN
                    {
                        Vji[j,i] = qji[j,i]
                    }
                    ELSE
                    {
                        Vji[j,i] = Contents(LOC(j))
                    }
                }
                i = i + 1
            }
            j = j + 1
        }

        i = 1
        j = 1
        // This routine builds the list of cStats for all sensors to all gateways
        WHILE i <= numGateway DO
        {
            j = 1
            WHILE j <= numSensor DO
            {
                IF CONji[j,i] <> 0 THEN
                {
                    cStatji[j,i] = (Vji[j,i]  * CSji[j,i]) + ((Contents(LOC(j)) -
Vji[j,i]) *  aveCSj[j])
                    cStatIdOrder[j,i] = i
```

```
                    }
                    ELSE
                    {
                            cStatji[j,i] = NA
                            cStatIdOrder[j,i] = NA
                    }

                j = j +1
                }
                i = i + 1
            }

            //_____ Sensor Priority List
Routine_____
            j = 1
            i = 1

            // THIS replaces the negative cStatji values with a very large value and
makes the index -1 so it is never chosen
            WHILE j <= numSensor DO
            {
                i = 1
                WHILE i <= numGateway DO
                {

                    IF CONji[j,i] = 0 THEN
                    {
                            cStatIdOrder[j,i] = -1
                            cStatji[j,i] = 100000
                    }
                    i = i + 1
                }
                j = j + 1
            }
            //PROMPT "Prompt num is ", promptNum
            //Once the negative values are replaced and the indices are replaced, the
cStats are sorted below ascending
            j = 1
            i = 1
            int x = 2
            int tempc = 0
            int tempcID = 0
            int test = numGateway - 1

            IF tieBreak2 = 0 THEN
```

```
{
    WHILE j <= numSensor DO
    {
        i = 1
        WHILE i <= numGateway - 1 DO
        {
            x = 2
            WHILE x <= numGateway DO
            {
                IF cStatji[j,x] < cStatji[j,x-1] THEN
                {
                    tempc = cStatji[j,x]
                    tempcID = cStatIdOrder[j,x]

                    cStatji[j,x] = cStatji[j,x-1]
                    cStatIdOrder[j,x] = cStatIdOrder[j,x-1]

                    cStatji[j,x-1] = tempc
                    cStatIdOrder[j,x-1] = tempcID
                }
                x = x +1

            }
            i = i + 1
        }
        j = j + 1
        tieBreak2 = 1
    }
}
ELSE
{
    WHILE j <= numSensor DO
    {
        i = 1
        WHILE i <= numGateway - 1 DO
        {
            x = 2
            WHILE x <= numGateway DO
            {
                IF cStatji[j,x] <= cStatji[j,x-1] THEN
                {
                    tempc = cStatji[j,x]
                    tempcID = cStatIdOrder[j,x]

                    cStatji[j,x] = cStatji[j,x-1]
                    cStatIdOrder[j,x] = cStatIdOrder[j,x-1]
```

```
                                        cStatji[j,x-1] = tempc
                                        cStatIdOrder[j,x-1] = tempcID
                                    }
                                    x = x +1

                                }
                                i = i + 1
                            }
                            j = j + 1
                            tieBreak2 = 0
                        }
                    }
                    promptNum = 3
                    //PROMPT "Prompt num is ", promptNum
                    // Now that the index list is generated, we will make the priority lists only
if a sensors wants to turn on
                    j = 1
                    i = 1

                    WHILE j <= numSensor DO
                    {
                        i = 1
                        WHILE i <= numGateway DO
                        {
                                                // 2
//2 (1 * numArrivalj[j]))
                            IF (StatusS[j] = 1 AND Contents(LOC(j)) >= (4 *
numArrivalj[j])) OR (StatusS[j] = 0 AND CAP(LOC(j))- Contents(LOC(j)) <= (10 *
numArrivalj[j])) THEN
                            {
                                SensorPriorityjp[j, i] = cStatIdOrder[j,i]
                                NumTotal_Requests =  NumTotal_Requests + 1
////////////////////////////New CODE ADDED LAST NIGHT

                            }
                            ELSE
                            {
                                SensorPriorityjp[j, i] = 0
                            }
                            i = i + 1
                        }
                        j = j + 1
                    }
```

```
                //_____ END Sensor Priority List
Routine_____
                j = 1
                i = 1

                WHILE i <= numGateway DO
                {
                    GrantSID[i] = NA
                    i = i +1
                }

                j = 1
                WHILE j <= numSensor DO
                {
                    CurrentPriorityID[j] = 0
                    FinalGID[j] = NA // -1
                    j = j + 1
                }

                i = 1
                j = 1
                int allSensorMatched = FALSE
                int numSensorOK = 0

                WHILE allSensorMatched = FALSE DO
                {
                    //PROMPT "This is the value of allSensorMatched",allSensorMatched
                    //PROMPT "This is the value of numSensorOK", numSensorOK
                    j = 1
                    WHILE j <= numSensor DO
                    {
                            RequestGID[j] = NA
                            j = j + 1
                    }


                    int y = 1
                    int entered = FALSE

                    WHILE y <= numSensor DO
                    {
                            CurrentPriorityID[y] = CurrentPriorityID[y] + 1
                            IF FinalGID[y] = NA THEN  // This will cause the sensor to
request a gateway
                            {
```

```
                                WHILE SensorPrioorityjp[y,CurrentPriorityID[y]] = - 1
AND  CurrentPriorityID[y] <= numGateway DO  // Not sure about this logic
                                {
                                        //PROMPT
"SensorPrioorityjp[y,CurrentPriorityID[y]]", SensorPrioorityjp[y,CurrentPriorityID[y]]
                                //      PROMPT "CurrentPriorityID[y]",
CurrentPriorityID[y]
                                        CurrentPriorityID[y] = CurrentPriorityID[y] + 1
                                }

                                IF CurrentPriorityID[y] > numGateway THEN
                                {
                                        RequestGID[y] = 0
                                }
                                ELSE
                                {
                                        RequestGID[y] =
SensorPrioorityjp[y,CurrentPriorityID[y]]
                                }
                        }
                        y = y + 1
                }

                j = 1
                int oldSID = 0
                int newSID = 0
                int currentGID = 0
                int k = 1
                WHILE j <= numSensor DO
                {
                        IF RequestGID[j] = 0 THEN
                        {
                                FinalGID[j] = 0  // This means sensor has requested all
gateways in its list and no longer will request
                                numSensorOK = numSensorOK + 1
                        }
                        ELSE IF RequestGID[j] > 0 THEN
                        {
                                oldSID = GrantSID[RequestGID[j]]
                                newSID = j
                                currentGID = RequestGID[j]
                                k = 1
                                IF GrantSID[currentGID] = NA THEN
                                {
                                        GrantSID[currentGID] = newSID    //Grants
current sensorID to current gatewayID
```

```
                                        FinalGID[newSID] = currentGID
                                        numSensorOK = numSensorOK + 1
                                }
                                ELSE
                                {
                                        WHILE k <= numSensor DO
                                        {
                                                IF GatewayPrioirityip[currentGID, k] =
newSID THEN
                                                {
                                                        FinalGID[oldSID] = NA
        //-1    TEll oldSID to request again
                                                        GrantSID[currentGID] =
newSID    //Grants current sensorID to current gatewayID
                                                        FinalGID[newSID] = currentGID
                //Tell newSID not to request in next request time
                                                        BREAK
                                                }
                                                ELSE IF GatewayPrioirityip[currentGID,
k] = oldSID THEN
                                                {
                                                        BREAK
                                                }
                                                k = k + 1
                                        }
                                }
                        }
                        j = j + 1
                }

                IF numSensorOK = numSensor THEN
                {
                        allSensorMatched = TRUE
                }
        }

        // The grants for this iteration have been made now need to transmit the
data to the apropriate gateway
        promptNum = 4
        //PROMPT "Prompt num is ", promptNum
        j = 1
        i = 1

        WHILE j <= numSensor  DO
        {
```

```
        IF FinalGID[j] = 0 THEN
        {
                StatusS[j] = 0
        }
        ELSE
        {
                i = FinalGID[j]
                IF StatusS[j] = 0 THEN
                {
                        StatusS[j] = 1
                        SpikeYNj[j] = 1
                }
                IF StatusG[i] = 0 THEN
                {
                        StatusG[i] = 1
                        SpikeYNi[i] = 1
                }
                //PROMPT "We got pretty far baby!",
// int k = 1
//WHILE k <= Vji[j,i]
                SEND Vji[j,i] ENT(j) TO LOC(i + 13)
                NumTotal_Granted = NumTotal_Granted + 1



        }
        j = j + 1
}
```

```
                1   Dummy   DummyArrival     FIRST 1    MOVE FOR 1


Bit1    Sensor1         BitSource = LOCATION()
                Uj[1] = CONTENTS(Sensor1, Bit1)
                S1_Buffer = Uj[1]

                        1   Bit1    Gateway1        SEND 1     MOVE FOR .9
                                                    S1_Buffer = Uj[1]
                            Bit1    Gateway2        SEND       MOVE FOR .9
                                                    S1_Buffer = Uj[1]
                            Bit1    Gateway3        SEND       MOVE FOR .9
                                                    S1_Buffer = Uj[1]
                            Bit1    Gateway4        SEND       MOVE FOR .9
                                                    S1_Buffer = Uj[1]
Bit2    Sensor2         BitSource = LOCATION()
                Uj[2] = CONTENTS(Sensor2, Bit2)
```

```
                S2_Buffer = Uj[2]
                        1   Bit2    Gateway1      SEND 1     MOVE FOR .9
                                                  S2_Buffer = Uj[2]
                            Bit2    Gateway2      SEND       MOVE FOR .9
                                                  S2_Buffer = Uj[2]
                            Bit2    Gateway3      SEND       MOVE FOR .9
                                                  S2_Buffer = Uj[2]

                            Bit2    Gateway4      SEND       MOVE FOR .9
                                                  S2_Buffer = Uj[2]
Bit3    Sensor3     BitSource = LOCATION()
                    Uj[3] = CONTENTS(Sensor3, Bit3)
                    S3_Buffer = Uj[3]
                        1   Bit3    Gateway1      SEND 1     MOVE FOR .9
                                                  S3_Buffer = Uj[3]
                            Bit3    Gateway2      SEND       MOVE FOR .9
                                                  S3_Buffer = Uj[3]
                            Bit3    Gateway3      SEND       MOVE FOR .9
                                                  S3_Buffer = Uj[3]
                            Bit3    Gateway4      SEND       MOVE FOR .9
                                                  S3_Buffer = Uj[3]
Bit4    Sensor4     BitSource = LOCATION()
                    Uj[4] = CONTENTS(Sensor4, Bit4)
                    S4_Buffer = Uj[4]
                        1   Bit4    Gateway1      SEND 1     MOVE FOR .9
                                                  S4_Buffer = Uj[4]

                            Bit4    Gateway2      SEND       MOVE FOR .9
                                                  S4_Buffer = Uj[4]
                            Bit4    Gateway3      SEND       MOVE FOR .9
                                                  S4_Buffer = Uj[4]
                            Bit4    Gateway4      SEND       MOVE FOR .9
                                                  S4_Buffer = Uj[4]
Bit5    Sensor5     BitSource = LOCATION()
                    Uj[5] = CONTENTS(Sensor5, Bit5)
                    S5_Buffer = Uj[5]
                        1   Bit5    Gateway1      SEND 1     MOVE FOR .9
                                                  S5_Buffer = Uj[5]

                            Bit5    Gateway2      SEND       MOVE FOR .9
                                                  S5_Buffer = Uj[5]

                            Bit5    Gateway3      SEND       MOVE FOR .9
                                                  S5_Buffer = Uj[5]

                            Bit5    Gateway4      SEND       MOVE FOR .9
```

```
                                                    S5_Buffer = Uj[5]

Bit6    Sensor6         BitSource = LOCATION()
                        Uj[6] = CONTENTS(Sensor6, Bit6)
                        S6_Buffer = Uj[6]
                                1   Bit6    Gateway1        SEND 1      MOVE FOR .9
                                                            S6_Buffer = Uj[6]

                                    Bit6    Gateway2        SEND        MOVE FOR .9
                                                            S6_Buffer = Uj[6]
                                    Bit6    Gateway3        SEND        MOVE FOR .9
                                                            S6_Buffer = Uj[6]
                                    Bit6    Gateway4        SEND        MOVE FOR .9
                                                            S6_Buffer = Uj[6]
Bit7    Sensor7         BitSource = LOCATION()
                        Uj[7] = CONTENTS(Sensor7, Bit7)
                        S7_Buffer = Uj[7]
                                1   Bit7    Gateway1        SEND 1      MOVE FOR .9
                                                            S7_Buffer = Uj[7]

                                    Bit7    Gateway2        SEND        MOVE FOR .9
                                                            S7_Buffer = Uj[7]

                                    Bit7    Gateway3        SEND        MOVE FOR .9
                                                            S7_Buffer = Uj[7]

                                    Bit7    Gateway4        SEND        MOVE FOR .9
                                                            S7_Buffer = Uj[7]

Bit8    Sensor8         BitSource = LOCATION()
                        Uj[8] = CONTENTS(Sensor8, Bit8)
                        S8_Buffer = Uj[8]
                                1   Bit8    Gateway1        SEND 1      MOVE FOR .9
                                                            S8_Buffer = Uj[8]

                                    Bit8    Gateway2        SEND        MOVE FOR .9
                                                            S8_Buffer = Uj[8]

                                    Bit8    Gateway3        SEND        MOVE FOR .9
                                                            S8_Buffer = Uj[8]

                                    Bit8    Gateway4        SEND        MOVE FOR .9
                                                            S8_Buffer = Uj[8]

Bit9    Sensor9         BitSource = LOCATION()
                        Uj[9] = CONTENTS(Sensor9, Bit9)
```

```
            S9_Buffer = Uj[9]
                        1   Bit9    Gateway1        SEND 1     MOVE FOR .9
                                                   S9_Buffer = Uj[9]

                            Bit9    Gateway2       SEND       MOVE FOR .9
                                                   S9_Buffer = Uj[9]

                            Bit9    Gateway3       SEND       MOVE FOR .9
                                                   S9_Buffer = Uj[9]

                            Bit9    Gateway4       SEND       MOVE FOR .9
                                                   S9_Buffer = Uj[9]

Bit10   Sensor10        BitSource = LOCATION()
            Uj[10] = CONTENTS(Sensor10, Bit10)
            S10_Buffer = Uj[10]
                        1   Bit10   Gateway1        SEND 1     MOVE FOR .9
                                                   S10_Buffer = Uj[10]

                            Bit10   Gateway2       SEND       MOVE FOR .9
                                                   S10_Buffer = Uj[10]

                            Bit10   Gateway3       SEND       MOVE FOR .9
                                                   S10_Buffer = Uj[10]

                            Bit10   Gateway4       SEND       MOVE FOR .9
                                                   S10_Buffer = Uj[10]

Bit11   Sensor11        BitSource = LOCATION()
            Uj[11] = CONTENTS(Sensor11, Bit11)
            S11_Buffer = Uj[11]
                        1   Bit11   Gateway1        SEND 1     MOVE FOR .9
                                                   S11_Buffer = Uj[11]

                            Bit11   Gateway2       SEND       MOVE FOR .9
                                                   S11_Buffer = Uj[11]
                            Bit11   Gateway3       SEND       MOVE FOR .9
                                                   S11_Buffer = Uj[11]
                            Bit11   Gateway4       SEND       MOVE FOR .9
                                                   S11_Buffer = Uj[11]
Bit12   Sensor12        BitSource = LOCATION()
            Uj[12] = CONTENTS(Sensor12, Bit12)
            S12_Buffer = Uj[12]
                        1   Bit12   Gateway1        SEND 1     MOVE FOR .9
                                                   S12_Buffer = Uj[12]
```

```
                    Bit12   Gateway2        SEND      MOVE FOR .9
                                            S12_Buffer = Uj[12]

                    Bit12   Gateway3        SEND      MOVE FOR .9
                                            S12_Buffer = Uj[12]

                    Bit12   Gateway4        SEND      MOVE FOR .9
                                            S12_Buffer = Uj[12]

ALL     Gateway1        int sensorNum
                sensorNum = ENTITY() // This logic will need to be updated if we make
some sensor jumping nodes


                Uj[sensorNum] = CONTENTS(LOC(sensorNum), ENT(sensorNum))
                IF SpikeYNj[sensorNum]  =  0 THEN
                {
                    ESj[sensorNum] = ESj[sensorNum] - CSji[sensorNum,1]-
COj[sensorNum]/ Vji[sensorNum,1]
                }
                ELSE
                {
                    ESj[sensorNum] = ESj[sensorNum] - CSji[sensorNum,1] -
SSj[sensorNum]- COj[sensorNum]/ Vji[sensorNum,1]
                    SpikeYNj[sensorNum] = 0

                }

                IF SpikeYNi[1] = 0 THEN
                {
                    EGi[1] = EGi[1] - CSji[sensorNum,1]
                    }
                ELSE
                {
                    EGi[1] = EGi[1] - CSji[sensorNum,1] - SGi[1]
                    SpikeYNi[1] = 0
                }

                EG1 = EGi[1]
                EG2 = EGi[2]

                ES1 = ESj[1]
                ES2 = ESj[2]
                ES3 = ESj[3]
                ES4 = ESj[4]
                ES5 = ESj[5]
```

```
ES6 = ESj[6]
ES7 = ESj[7]
ES8 = ESj[8]
ES9 = ESj[9]
ES10 = ESj[10]
ES11 = ESj[11]
ES12 = ESj[12]
```

```
                    1   ALL     EXIT            FIRST 1
ALL     Gateway2        int sensorNum
        sensorNum = ENTITY() // This logic will need to be updated if we make
some sensor jumping nodes


        Uj[sensorNum] = CONTENTS(LOC(sensorNum), ENT(sensorNum))
        IF SpikeYNj[sensorNum]  =  0 THEN
        {
            ESj[sensorNum] = ESj[sensorNum] - CSji[sensorNum,2] -
COj[sensorNum]/ Vji[sensorNum,2]
        }
        ELSE
        {
            ESj[sensorNum] = ESj[sensorNum] - CSji[sensorNum,2]
SSj[sensorNum] - COj[sensorNum]/ Vji[sensorNum,2]
            SpikeYNj[sensorNum] = 0

        }

        IF SpikeYNi[2] = 0 THEN
        {
            EGi[2] = EGi[2] - CSji[sensorNum,2]
            }
        ELSE
        {
            EGi[2] = EGi[2] - CSji[sensorNum,2] - SGi[2]
            SpikeYNi[2] = 0
        }
```

```
                EG1 = EGi[1]
                EG2 = EGi[2]

                ES1 = ESj[1]
                ES2 = ESj[2]
                ES3 = ESj[3]
                ES4 = ESj[4]
                ES5 = ESj[5]
                ES6 = ESj[6]
                ES7 = ESj[7]
                ES8 = ESj[8]
                ES9 = ESj[9]
                ES10 = ESj[10]
                ES11 = ESj[11]
                ES12 = ESj[12]
                        1   ALL     EXIT            FIRST 1
ALL     Gateway3        int sensorNum
                sensorNum = ENTITY() // This logic will need to be updated if we make
some sensor jumping nodes


                Uj[sensorNum] = CONTENTS(LOC(sensorNum), ENT(sensorNum))
                IF SpikeYNj[sensorNum]  =  0 THEN
                {
                    ESj[sensorNum] = ESj[sensorNum] - CSji[sensorNum,3] -
COj[sensorNum] / Vji[sensorNum,3]
                }
                ELSE
                {
                    ESj[sensorNum] = ESj[sensorNum] - CSji[sensorNum,3] -
SSj[sensorNum] - COj[sensorNum]/ Vji[sensorNum,3]
                    SpikeYNj[sensorNum] = 0

                }

                IF SpikeYNi[3] = 0 THEN
                {
                    EGi[3] = EGi[3] - CSji[sensorNum,3]
                    }
                ELSE
                {
                    EGi[3] = EGi[3] - CSji[sensorNum,3] - SGi[3]
                    SpikeYNi[3] = 0
                }
```

```
EG1 = EGi[1]
EG2 = EGi[2]

ES1 = ESj[1]
ES2 = ESj[2]
ES3 = ESj[3]
ES4 = ESj[4]
ES5 = ESj[5]
ES6 = ESj[6]
ES7 = ESj[7]
ES8 = ESj[8]
ES9 = ESj[9]
ES10 = ESj[10]
ES11 = ESj[11]
ES12 = ESj[12]
                    1   ALL     EXIT            FIRST 1
ALL     Gateway4        int sensorNum
sensorNum = ENTITY() // This logic will need to be updated if we make
some sensor jumping nodes


Uj[sensorNum] = CONTENTS(LOC(sensorNum), ENT(sensorNum))
IF SpikeYNj[sensorNum]  =  0 THEN
{
    ESj[sensorNum] = ESj[sensorNum] - CSji[sensorNum,4] -
COj[sensorNum]/ Vji[sensorNum,4]
}
ELSE
{
    ESj[sensorNum] = ESj[sensorNum] - CSji[sensorNum,4] -
SSj[sensorNum] - COj[sensorNum]/ Vji[sensorNum,4]
    SpikeYNj[sensorNum] = 0

}

IF SpikeYNi[4] = 0 THEN
{
    EGi[4] = EGi[4] - CSji[sensorNum,4]
    }
ELSE
{
    EGi[4] = EGi[4] - CSji[sensorNum,4] - SGi[4]
    SpikeYNi[4] = 0
}

EG1 = EGi[1]
```

```
EG2 = EGi[2]

ES1 = ESj[1]
ES2 = ESj[2]
ES3 = ESj[3]
ES4 = ESj[4]
ES5 = ESj[5]
ES6 = ESj[6]
ES7 = ESj[7]
ES8 = ESj[8]
ES9 = ESj[9]
ES10 = ESj[10]
ES11 = ESj[11]
ES12 = ESj[12]
                    1  ALL   EXIT          FIRST 1
Man1   Man1Arrival              1  Man1   ArrivalS3      FIRST 1   IF Tnow >
100 THEN
                                          {
                                          GRAPHIC 2
                                          }
                                          MOVE FOR 1.8
Man1   ArrivalS3      ORDER 1 Bit3 TO ArrivalS3
            //WAIT UNTIL HoldMen = 1
                    1  Man1   ArrivalS2     FIRST 1   IF Tnow > 100
THEN
                                          {
                                          GRAPHIC 2
                                          }
                                          MOVE FOR 1.8
Man1   ArrivalS2      ORDER 1 Bit2 TO ArrivalS2
            //WAIT UNTIL HoldMen = 1
                    1  Man1   ArrivalS1     0.500000 1  IF Tnow > 100
THEN
                                          {
                                          GRAPHIC 2
                                          }
                                          MOVE FOR 1.8
                       Man1   ArrivalS4     0.500000   IF Tnow > 100 THEN
                                          {
                                          GRAPHIC 2
                                          }
                                          MOVE FOR 1.8
Man1   ArrivalS1      ORDER 1 Bit1 TO ArrivalS1
            //WAIT UNTIL HoldMen = 1
                    1  Man1   MenExit       FIRST 1   IF Tnow > 100
THEN
```

```
                                                    {
                                                    GRAPHIC 2
                                                    }
                                                    MOVE FOR 1.8
Man1    ArrivalS4       ORDER 1 Bit4 TO ArrivalS4
                //WAIT UNTIL HoldMen = 1
                        1   Man1    MenExit         FIRST 1     IF Tnow > 100
THEN
                                                    {
                                                    GRAPHIC 2
                                                    }
                                                    MOVE FOR 1.8
ALL     MenExit                 1   ALL     EXIT            FIRST 1
Man2    Man2Arrival             1   Man2    ArrivalS6       FIRST 1     IF Tnow >
100 THEN
                                                    {
                                                    GRAPHIC 2
                                                    }
                                                    MOVE FOR .18
Man2    ArrivalS6       ORDER 1 Bit6 TO ArrivalS6
                        1   Man2    ArrivalS5       0.500000 1  IF Tnow > 100
THEN
                                                    {
                                                    GRAPHIC 2
                                                    }

                                                    MOVE FOR 1.8
                        Man2    ArrivalS8       0.500000    IF Tnow > 100 THEN
                                                    {
                                                    GRAPHIC 2
                                                    }
                                                    MOVE FOR 1.8
Man2    ArrivalS5       ORDER 1 Bit5 TO ArrivalS5
                        1   Man2    ArrivalS4       FIRST 1     IF Tnow > 100
THEN
                                                    {
                                                    GRAPHIC 2
                                                    }
                                                    MOVE FOR 1.8
Man2    ArrivalS4       ORDER 1 Bit4 TO ArrivalS4
                        1   Man2    MenExit         FIRST 1     IF Tnow > 100
THEN
                                                    {
                                                    GRAPHIC 2
                                                    }
                                                    MOVE FOR 1.8
```

```
Man2    ArrivalS8       ORDER 1 Bit8 TO ArrivalS8
                            1   Man2    ArrivalS11      FIRST 1    IF Tnow > 100
THEN
                                                        {
                                                        GRAPHIC 2
                                                        }
                                                        MOVE FOR 1.8
Man2    ArrivalS11      ORDER 1 Bit11 TO ArrivalS11
                            1   Man2    ArrivalS10      FIRST 1    IF Tnow > 100
THEN
                                                        {
                                                        GRAPHIC 2
                                                        }
                                                        MOVE FOR 1.8
Man2    ArrivalS10      ORDER 1 Bit10 TO ArrivalS10
                            1   Man2    MenExit         FIRST 1    IF Tnow > 100
THEN
                                                        {
                                                        GRAPHIC 2
                                                        }
                                                        MOVE FOR 1.8
Man3    Man3Arrival                 1   Man3    ArrivalS9       0.500000 1  IF Tnow >
100 THEN
                                                        {
                                                        GRAPHIC 2
                                                        }
                                                        MOVE FOR 1.8
                                Man3    ArrivalS12      0.500000   IF Tnow > 100
THEN
                                                        {
                                                        GRAPHIC 2
                                                        }
                                                        MOVE FOR 1.8
Man3    ArrivalS12      ORDER 1 Bit12 TO ArrivalS12
                            1   Man3    ArrivalS11      FIRST 1    IF Tnow > 100
THEN
                                                        {
                                                        GRAPHIC 2
                                                        }
                                                        MOVE FOR 1.8
Man3    ArrivalS11      ORDER 1 Bit11 TO ArrivalS11
                            1   Man3    ArrivalS10      FIRST 1    IF Tnow > 100
THEN
                                                        {
                                                        GRAPHIC 2
                                                        }
```

```
                                                    MOVE FOR 1.8
 Man3    ArrivalS10      ORDER 1 Bit10 TO ArrivalS10
                           1   Man3    MenExit       FIRST 1    IF Tnow > 100
THEN
                                                    {
                                                    GRAPHIC 2
                                                    }
                                                    MOVE FOR 1.8
 Man3    ArrivalS9       ORDER 1 Bit9 TO ArrivalS9
                           1   Man3    ArrivalS8     FIRST 1    IF Tnow > 100
THEN
                                                    {
                                                    GRAPHIC 2
                                                    }
                                                    MOVE FOR 1.8
 Man3    ArrivalS8       ORDER 1 Bit8 TO ArrivalS8
                           1   Man3    ArrivalS7     FIRST 1    IF Tnow > 100
THEN
                                                    {
                                                    GRAPHIC 2
                                                    }
                                                    MOVE FOR 1.8
 Man3    ArrivalS7       ORDER 1 Bit7 TO ArrivalS7
                           1   Man3    MenExit       FIRST 1    IF Tnow > 100
THEN
                                                    {
                                                    GRAPHIC 2
                                                    }
                                                    MOVE FOR 1.8
 Bit1    Gateway2
 Bit1    Loc2


*************************************************************************
*                        Arrivals
*************************************************************************

 Entity  Location   Qty each       First Time Occurrences Frequency  Logic
 -------- ------------ --------------- ---------- ----------- ---------- ------------
 Bit1    ArrivalS1   NumArrivalj[1] 0        INF        1
 Dummy   DummyArrival 1             0        1          1
 Bit2    ArrivalS2   NumArrivalj[2] 0        INF        1
 Bit3    ArrivalS3   NumArrivalj[3] 0        INF        1
 Bit4    ArrivalS4   NumArrivalj[4] 0        INF        1
 Bit5    ArrivalS5   NumArrivalj[5] 0        INF        1
 Bit6    ArrivalS6   NumArrivalj[6] 0        0          4
```

Bit7	ArrivalS7	NumArrivalj[7]	0	0	4	
Bit8	ArrivalS8	NumArrivalj[8]	0	0	4	
Bit9	ArrivalS9	NumArrivalj[9]	0	0	4	
Bit10	ArrivalS10	NumArrivalj[10]	0	0	4	
Bit11	ArrivalS11	NumArrivalj[11]	0	0	4	
Bit12	ArrivalS12	NumArrivalj[12]	0	0	4	
Man1	Man1Arrival	1	0	0	4	
Man2	Man2Arrival	1	3	0	4	
Man3	Man3Arrival	1	8	0	4	

****
* Attributes
****

ID	Type	Classification
BitSource	Integer	Entity

**********V
ariables (global)
****

ID	Type	Initial value	Stats
numGateway	Integer	2	Time Series
numSensor	Integer	4	Time Series
S1_Buffer	Integer	0	Time Series
S2_Buffer	Integer	0	Time Series
S3_Buffer	Integer	0	Time Series
S4_Buffer	Integer	0	Time Series
S5_Buffer	Integer	0	Time Series
S6_Buffer	Integer	0	Time Series
S7_Buffer	Integer	0	Time Series
S8_Buffer	Integer	0	Time Series
S9_Buffer	Integer	0	Time Series
S10_Buffer	Integer	0	Time Series
S11_Buffer	Integer	0	Time Series
S12_Buffer	Integer	0	Time Series
S13_Buffer	Integer	0	Time Series
HoldMen	Integer	0	Time Series
Tnow	Integer	0	Time Series
tieBreak1	Integer	0	Time Series
tieBreak2	Integer	0	Time Series
ES1	Real	0	Time Series

ES2	Real	0	Time Series
ES3	Real	0	Time Series
ES4	Real	0	Time Series
ES5	Real	0	Time Series
ES6	Real	0	Time Series
ES7	Real	0	Time Series
ES8	Integer	0	Time Series
ES9	Integer	0	Time Series
ES10	Integer	0	Time Series
ES11	Integer	0	Time Series
ES12	Integer	0	Time Series
ES13	Integer	0	Time Series
EG1	Integer	0	Time Series
EG2	Integer	0	Time Series
EG3	Integer	0	Time Series
EG4	Integer	0	Time Series
NumTotal_Requests	Integer	0	Time Series
NumTotal_Granted	Integer	0	Time Series

**********A
rrays
****

ID	Dimensions	Type
SensorPriorityjp	15,15	Integer
GatewayPrioityip	15,15	Integer
RequestGID	15	Integer
GrantSID	15	Integer
FinalGID	15	Integer
CurrentPriorityID	15	Integer
tStatIdOrder	15	Integer
tStatTemp	15	Integer
tStatj	15	Real
cStatIdOrder	15,15	Integer
cStatji	15,15	Real
CONji	15,15	Integer
SSj	15	Real
SGi	15	Real
ESj	15	Real
EGi	15	Real
RemaindingIDsj	15	Integer
Vji	15,15	Real
Uj	15	Real
qji	15,15	Real
StatusS	15	Integer

StatusG	15	Integer
CSji	15,15	Real
aveCSj	15	Integer
aveqj	15	Integer
SpikeYNj	15	Integer
SpikeYNi	15	Integer
Uhatj	15	Integer
NumArrivalj	15	Real
COj	15	Real

****
*Macros
****

ID	Text
Mac1	None

****
*Subroutines
****

ID	Type	Parameter	Type	Logic
Sub1	None			