Energy Efficient Scheduling of Wireless Sensor Networks

A Thesis submitted in
partial fulfillment of the requirements for the degree
of

Bachelor of Technology
in
Computer Science and Engineering

Gurinder Raju, 2001111

Rajpal, 2001130

Under the guidance of

Prof. Bijendra N. Jain

Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

July, 2005

Abstract

Wireless Sensor Networks are an emerging technology with potential application in areas as wide ranging as habitat monitoring and industrial applications. Sensors monitor the changes in a physical attribute of the surroundings, say temperature, and observed data is collected and analyzed. These sensors are mostly unattended, and their limited battery life makes energy a precious resource that must be utilised wisely.

In this thesis we present a distributed scheduling protocol for energy-efficient media access in many to one data collection applications, where the destination of all the data packets in the network is a central data collector, denoted as the Access Point. Energy efficiency is achieved by avoiding retransmissions, idle listening and overhearing. The system implemented over this scheduling protocol uses no notion of global time and works in three phases. The first phase is the topology learning phase, where each node gets to know about its parent and each parent about its children. The second phase is the schedule formation phase. The Third phase is the data collection and routing phase, that continues to re-use the schedules calculated in the previous phase.

We analyse the protocol through simulations in TOSSIM, a simulator for TinyOS applications. The results show significant improvements in energy consumption over the contention based access scheme.

Certificate

This is to certify that the thesis titled *"Energy Efficient Scheduling of Wireless Sensor Networks"* being submitted by Gurinder Raju & Rajpal is a record of bona-fide work carried out by them under my supervision.

The matter and results presented in this thesis are original and have not been submitted elsewhere, wholly or in part, for the award of any degree or diploma.

Prof. Bijendra N. Jain
(Project Supervisor)
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Acknowledgements

We are grateful to our project supervisor *Prof. Bijendra N. Jain* for providing us continuous support throughout the year. His inputs, suggestions, criticisms, and undying patience go great lengths in helping us realize this project. We are also grateful to *Prof. Huzur Saran*, for his encouragement and suggestions during the presentations.

We thank *Mr. Surendra Negi* for helping us out in acquiring systems in the Ernet Lab and also all the fellow networking group members who gave us their full co-operation during the project.

Gurinder Raju
2001111

Rajpal
2001130

Contents

List of Tables

List of Figures

1 Introduction

A Wireless Sensor Network consists of group of nodes called sensor nodes. Each one of these has an embedded processor, a radio and one or more sensors. These nodes operate together in the area being monitored and collect physical attributes of the surroundings, say temperature or humidity. Data gathered by these sensor nodes can be utilised by various top level applications such as habitat monitoring, surveillance systems and systems monitoring various natural phenomenon.

Sensor nodes have limited battery life. In some applications the sensors are placed in difficult-to-reach locations, expecting manual intervention for renewal of battery is impractical. In fact, with advances in technology we can expect that in near future these sensor nodes will be disposable and will only last until their energy drains away. The node has to sustain itself on its battery's limited energy resources and without power management it can last only for a short period of time.

All these power limitations in the sensor networks require the application to have adhoc power saving mechanism to extend the life of the node. The packet transmission and packet receive over the radio consume the maximum power in sensor nodes. In a contention based system, nodes waste energy in retransmitting the packets due to collisions. The nodes also waste energy in overhearing conversations not meant for them and in listening to the idle network by being awake, *i.e.*, keeping the radio ON, all the time.

A media access scheduling mechanism is required for creating sleep/wake schedule for the radio of each node, informing the nodes when to sleep, wake up, sense and transmit their own data and when to relay the data of the other nodes.

1.1 Objectives

Our objective is to design a monitoring application that uses the temperature sensor on each node to monitor the temperature of the surroundings and report it to the Access Point. The application makes use of the scheduling protocol incorporated above the MAC layer in conjuction with the network layer protocol designed to decide on the routes to the Access Point, to highlight the power gains achievable by using this scheduling protocol and its successful use in an application.

We will also ensure that the application dynamically ensures new nodes to become part of the network and adapts routes to the Access Point in case of some nodes dying due to harsh physical conditions or power drainage. In addition to that, we will ensure that requirement of tight time synchronisation is not there. We will also ensure that the scheduling protocol is scalable to a large number of nodes.

1.2 Organisation of the Report

The rest of the report is organised as follows. In Chapter 2 we describe the previous work done in this field and describe how various design issues have been handled. Chapter 3 describes the design approach of the scheduling protocol and the application. In Chapter 4 we present and analyse the simulation results. In Chapter 5 we discuss the implementation details. In Chapter 6 we conclude the thesis and give directions for future work.

2 Previous Work

In this chapter we discuss the previous work done in this field and various approaches to make the MAC in wireless sensor networks energy-efficient. Prior to that, we discuss the hardware platform all these protocols have been deployed on or simulated for, *i.e.*, the Berkeley motes.

2.1 Berkeley Motes

Berkeley motes are popular in the sensor network research community for their open source software development and commercial availability. Majority of the work being done in this area uses these motes and TinyOS as the application development platform. TinyOS is an open-source operating system designed especially for wireless embedded sensor networks. We will see more about it in Chapter 5. There are a range of motes available in the Berkeley family and in Table 2.1 we look at the infrastructural constraints in the one of the latest members of this family, the mica2 mote.

MCU	ATmega128L
Processor	8MHz
Program Memory	128 KB
RAM	4 KB
Battery	2xAA
RF Channel	916 MHz
Transmission Speed	38.4 KBps

Table 2.1: Infrastructure Constraints in mica2 mote

The Table 2.2 shows the energy consumption of various components on a mica mote.

The radio receive mode in comparison to transmit mode comsumes much lesser power, but in a setting where the radio receiver is ON all the time listening to incoming packets, the energy spent in this state will be much more. Transmitting a radio packet of 20 bytes would take 4ms and the same energy spent in sending this packet can keep the radio receiver ON for 27 ms. Going by the numbers, in a typical setting of a monitoring system, where the duty cycle can be as low as 1%, the energy spent in the receive mode will be an order greater than that spent in transmissions. Thus the radio must be supsended, whenever possible.

Component	Rate	Startup Time	Current Consumption
MCU active	4 MHz	N/A	5.5 mA
MCU idle	4 MHz	1μA	1.6mA
MCU suspend	4 MHz	4 ms	$< 20\mu$A
Radio transmit	40 KHz	30 ms	12 mA
Radio receive	40 KHz	30 ms	1.8 mA
Photoresister	2000 Hz	10 ms	1.235mA
Accelerometer	100 Hz	10 ms	5mA/axis
Temperature	2 Hz	500 ms	.150 mA

Table 2.2: Power Consumption in mica motes

2.2 Previous Systems

Energy efficiency in wireless sensor networks has been an area of intense research in the recent past. Achieving energy efficiency at the hardware level has its own fundamental physical limitations. As such, most of the research of late has focussed on MAC and network layer protocols. In each system, the design decisions are specific to the application in mind, as the network stack for sensor networks is not generic and customised according to the application. Apart from the systems that target similar application and media access scheduling, we also review other energy-aware systems and approaches. The following points help us analyse all the designs and their suitability to the application and the scheduling protocol that we target.

- Whether the scheduling algorithm is centralised or distributed.
- What is the level of time synchronisation needed in the network.
- How much and what kind of information does one node store about its neighbours. In general the per-node memory requirements.
- What is the extent to which the control packets are required.
- What is the latency in reporting the variable being monitored.
- How robust is the design in adapting to the changes in topology.
- What trade-off decisions have been made in the above parameters.

2.2.1 Centralised Systems

Time Synchronisation

Nodes' local times may differ from each other due to phase differences or the clock drifts over time. For time synchronisation they must communicate to each other their local times, but the following delays make tight time synchronisation difficult.

1. *Send Time* - The time spent at the sender to construct the message.
2. *Access Time* - The delay in getting hold of the channel to transmit the message.
3. *Propagation Time* - Time needed by the message to transit from the sender to the receiver via the media.
4. *Receive Time* - Time required in processing the message and knowing the contents.

In centralised systems, where the nodes synchronise with the time of the AP as reference [3], the send time and access time is specific to the AP only, and propagation time and receive time is same for all the nodes, as AP can reach all the nodes in one hop. As such, when the nodes synchronise their time with the AP, all of them get almost the same time stamp.

But tight time synchronisation is difficult when attempted over multi-hop. Over the hops, the time differences increase and thus in a situation where the notion of global time is needed to calculate the schedules, the transmission slots for individual nodes could overlap leading to collisions. Also the sleep slots would go out of sync leading to chaos in the network.

Time synchronisation has been achieved up to the level of μs [14], but it involves lot of packet transmissions meant just for time synchronisation and it doesn't serve well with the aim of minimising control packets.

TDMA based systems

In [3] and [4], TDMA(Time Division Multiple Access) based MAC protocols have been suggested. In [3], the Access Point itself synchronises the nodes and schedules their transmissions. In this system, the AP communicates with each of the nodes in one hop. Based on the topology information collected by the nodes and provided to the AP over multi-hop, it computes the sleep/wakeup schedule for the whole network and communicates it to all the nodes directly, and also its local time for the nodes to synchronise, together with the time when it will transmit the next packet because the nodes should leave transmission or come out of their sleep and listen to it at that time. Any scheduling algorithm can be utilised even if it is computationally intensive.

The assumption of unlimited energy and transmission power with the AP is unreasonable and maps more to a situation where the network is working in a friendly and accessible location. Also in case of large networks, the assumption that the Access Point will be able to reach even farthest nodes in one hop is unrealistic. Thus scalability is an issue here.

In [4] also, a similar TDMA based system has been suggested. It recommends the formation of clusters, where the gateway of each cluster is located within the communication range of each of its cluster sensors. These gateways use long-haul communication to send data to further gateways and finally to a base station similar to the one mentioned in the previous system.

The problem with the system is the overhead of formation of these clusters and also the inter-cluster communication and interference.

PEDAMACS

PEDAMACS [12], is another centralised system similar to the first system with respect to the high-powered AP. But it also targets the protocol to be delay-aware. This optimisation problem is NP-complete and the solution proposed is based on graph coloring. We don't have delay awareness as our goal and thus can achieve a much simpler protocol. The kind of application that we target can tolerate latency. Instead, we fully target on doing the power scheduling above the MAC layer for power conservation.

2.2.2 Distributed Systems

We so far discussed centralised systems that have the advantage that all the decisions are shifted to the AP. As compared to the distributed systems, they carry the disadvantage that for an effective schedule, the base station must know about $O(n^2)$ links, where n is the number of nodes in the network. This requires a lot of link probing and thus energy.

TRAMA

TRAMA [11] is a distributed protocol and it assumes time to be slotted. It also tries to achieve fairness in the system in addition to power conservation. It uses a distributed election scheme based on traffic patterns on each node to determine that which nodes can transmit in a particular time slot. Thus it avoids assigning time slots to nodes with no data to send. Each node randomly decides on its schedule and then adapts the schedule based on the traffic patterns of the neighbours.

The overhead of achieving fairness is that it has to keep record of its two-hop neighbours and their traffic and it has to send special schedule packets regularly. We don't target fairness and thus can avoid that. We instead target a system where the schedules are calculated in schedule formation phase and in the data routing phase, which is very large as compared to it, no changes are required in the schedules.

PAMAS & S-MAC

PAMAS[13] and S-MAC[10] are contention based protocols using RTS(Request to Send) and CTS(Clear to Send) packets to gain media access and transmit the data, similar to IEEE 802.11. PAMAS[13] uses an entirely separate signalling channel for control packets. S-MAC [10] is inspired by PAMAS[13], but it doesn't use any separate signalling channel. It requires much looser time synchronisation among the neighbouring nodes as compared to the TDMA based schemes. In this protocol, the neighbouring nodes can synchronise among themselves forming virtual clusters, sleeping and waking up to listen at the same time, and there is no need of real clusters as mentioned in [4]. Here each node is free to choose its sleep/wakeup schedule and maintains a schedule table that stores the schedule information of all its neighbours. Based on a randomly timed broadcast, a node initiates by sending its schedule, *i.e*, the time after which it will go to sleep. The receiver that has not formed its schedule yet, sets it schedule to be the same and the receiver that has already set its schedule, follows both schedules. The updation of the schedules is required due to clock drifts, and is done by special synchronisation packets.

The problem with these protocols is the overhead of the RTS and CTS packets for each data packet transmitted, resulting in significant overhead. Other control packets like the synchronisation packet further add to the overhead. Also, as the protocols are contention based, nodes spend long times listening idly to the network.

Ideally, the protocol should only use the contention approach during the schedule formation phase and that schedule should be used in the data collection phase for media access without contention.

Flexible Power Scheduling

Flexible Power Scheduling [17] is another distributed system and like TRAMA [11], it also computes and adapts the schedules based on *supply and demand*. It also divides time in to slots. The slot numbering is modulo m, where the big timer called the power schedule cycle is m slots long. Each slot is big enough to accommodate one receive/transmit exchange between the parent and the child node. The initiation is done by the AP, by advertising in a random slot for a random slot among its idle slots. Initially, with each node, all slots are idle slots and each node listens for advertisements for at least one cycle. In the cycle after the one in which, say node A receives the advertisement from, say node B for, say slot number n, node A requests for slot n for transmission to B by sending the request to B in the slot n itself. In this case, node B is the parent of node A. A node chooses its parent to be the one having minimum *demand*. After choosing the parent, each node synchronises its current time slot and slot number with that of the parent and this synchronisation is done periodically. Once a node acquires the required time slots to meet its demands, it advertises itself and turns off the radio during idle slots. Here the calculation of demand for each node is bottom-up and the schedule allocation is top-down.

Cross Layer Scheduling

Cross Layer scheduling [18], on the other hand, takes the bottom-up approach for schedule calculation. Each node here maintains a scheduling table that contains the entries for *Receive Time* and *Transmit Time* for the nodes, whose route to the AP goes through this node and also the entries for its own *Sense Time* and *Transmit Time*. These tables are formed on the basis of a special *route-setup* packet that is forwarded on the node's route to the AP. The routes are provided by a separate routing protocol. The node sending the *route-setup* packet enters the steady phase only after it receives the *rack* packet from the AP. It may turn out that a particular flow cannot be carried out by the network.

This system suggests use of special time synchronisation protocols like [23] for improving on the synchronisation. Integrating a separate time synchronisation protocol has a lot of overhead.

3 Design Approach

In this chapter, we describe the working of our system. Along with the routing protocol implemented for enabling the whole application to work and for using the scheduling protocol over it, our system works in three phases.

- The first phase is the initiation phase where the whole topolgy is built-up and parent-child relationships are formed.
- The second phase is the schedule formation phase where the scheduling protocol calculates the sleep/wakeup schedule for the whole network.
- The third phase is the steady phase, *i.e*, the data routing phase, where all the nodes send data to the AP based on the schedules calculated in the previous phase.

3.1 The Initiation Phase

This phase is itself divided into two phases. The first phase is the parent selection phase and the second phase is the child count phase.

3.1.1 Parent Selection Phase

The motes boot up at random times and the AP, waiting for a fixed time to ensure all motes are booted up, initiates the spanning tree formation by broadcasting a packet containing its *id*. A node on hearing this packet sets its parent to be the sender of the packet, and re-broadcasts, incrementing the hop-count by one and with itself as the sender. On hearing this packet again from a different sender, a node checks from the hop-count if it results in a shorter path to the AP. If so, it updates its parent and rebroadcasts after updating the hop-count and sender fields.

Alongwith its *id* and the hop-count, each node also includes its local time in the packet it broadcasts. The child updates its own time, when it sets its parent. This is only necessary for the application to make sense of the time stamps sent by the nodes along with the sensed data. The scheduling protocol doesn't need this as nodes are scheduled in their local time only.

Only the clock drifts over time effect the computed schedules and that is handled by periodically refreshing the topology and thus computing new schedules altogether. Periodic refreshing of topology is anyway required to take care of nodes dying out or new nodes joining.

In this phase, no Acks are included and once a node sets its parent, it broadcasts for a fixed number of times, backing-off for random time in-between two transmissions. The number of times each node broadcasts and the length of the phase are decided based on the size of the network, and are common to all nodes. The transmissions are stopped and random-backoffs terminated after the completion of this phase.

3.1.2 Child Count Phase

Here, the parents get to know of their children. The leaf nodes initiate by notifying their parents of themselves. Here, Acks have been included and the length of the phase is not based on the size of the network. The phase has to be long enough to handle collisions as child keeps on resending till it receives the Ack from the parent. The child waits between two transmissions based on a random timer which is terminated and transmission cancelled, once it receives the Ack.

3.2 The Schedule Formation Phase

Here the actual scheduling takes place. The following terms are important.

- **Cycle**: This is one power schedule cycle and is a multiple of one time slot. Its length is decided based on the number of nodes in the network. One cycle must be long enough for all the nodes to send their data to the AP.
- **Time Slot**: This is the time in which the parent and the child can achieve a **Reply-Ack/Neg-Ack** exchange.
- **Slot Count**: This is the the count for number of slots a child has to demand from its parent.
- **Child List**: This is the list that the parent maintains, containing the *id*s of all its children that have sent the **Request**.
- **Request**: This is the request from a child to its parent for number of slots equal to its **Slot Count**.
- **Reply**: This is the reply from the parent to the child, assigning it a time to transmit.
- **Ack**: This is the acknowledgement in which it communicates to the parent accepting the time assigned to transmit.

- **Neg-Ack**: This is the the negative-acknowledgement in which it communicates to the parent its inability to accept the time assigned to transmit.

3.2.1 The Scheduling Algorithm

Figures 3.1 and 3.2 highlight the working of the scheduling algorithm.

	Request From Child	Reply From Parent	Ack From Child	Neg-Ack From Child
Child	• Starts a timer to fire after random time.	• Turns Off the timer if it is On. • **If** at time at which it received the **Reply**, it has enough time for transmissions , it marks it for transmissions in steady phase and sends **Ack**. **Else** sends **Neg-Ack**.	-	-
Parent	• **If** child marked in the child list, ignores **Request**. **Else** marks the child and sends **Reply** at the first available time slot in the same cycle, **if** it is yet to come, **else** in the next cycle. • Monitors for **Ack/Neg-Ack** from the same child within that time slot.	-	• Increments its own required slot count. • Updates first available time slot. • **If** all child nodes scheduled and **if** not AP, sends **Request** to its own parent.	• Updates the first available time slot to be the next time slot and sends **Reply** in that slot.

Figure 3.1: State Transition Table For Received and Sent Packets

	Request From Child Lost	Reply From Parent Lost	Ack/Neg-Ack from Child Lost
Child	The timer fires. Again sends request to the parent.	The timer fires if it is On. Again sends request to the parent in case it was On.	-
Parent	-	Receives no packet from the same child in the monitored time slot. Sends reply again in the same time slot in the next cycle.	Receives no packet from the same child in the monitored time slot. Sends reply again in the same time slot in the next cycle.

Figure 3.2: State Transition Table For Lost Packets

Initiation

The whole scheduling is initiated by each node checking its child count. If it is zero, it sends a **Request** to its parent at a random time. Thus it is initiated by the leaf nodes, unlike Flexible Power Scheduling [17], where the initiation is top-down. In this algorithm, when a child and the parent interact for setting up of the schedule, only the time on the parent's side is slotted, unlike Flexible Power Scheduling [17]. And that too after the first available **Time Slot** for reception, which is initially at the start of the **Cycle**. In a **Cycle**, time prior to the first available **Time Slot** is already scheduled for reception.

Request From Child

In this algorithm, the time of sending the **Request** by the child is of no relevance to the final sleep/wakeup schedule. It is the time at which the parent sends the **Reply** that decides when the child has to transmit. Each time after sending the **Request** to the parent, the child starts a timer that fires randomly between one **Cycle** time and two **cycle** times from the time it was started at. This is to handle loss of packets. There has sure been a loss if the child doesn't receive the **Reply** within one **Cycle**.

Multiple requests from the child are needed, because the child's own **Request** or the parent's **Reply** could be lost. In case the parent's **Reply** is lost, the child sending the **Request** again won't make any difference beacause the parent has already enqueued it in the **Child List**. This futile activity comes from the child's ignorance of whether its **Request** has reached the parent or not.

Reply From Parent

When the child receives **Reply** from the parent, its timer could be ON, because it could be the first time it is receiving the **Reply** from the parent. In that case, the timer is turned OFF and the child doesn't need to send the **Request** again because it knows that its **Request** has been enqueued by the parent. The timer could well be OFF, because of two cases:

- The child had earlier received the **Reply**, but it has been sent again by the parent because the corresponding **Ack** or **Neg-Ack** from the child got lost in the previous **Cycle**.
- The **Neg-Ack** from the child was received by the parent in the last **Time Slot** in the current **Cycle** itself, in which case the parent updated its first available **Time Slot** by shifting it one slot ahead and sent the **Request** to the child again.

Neg-Ack From The Child

The child sends **Neg-Ack** to the parent, when the time at which it received the **Reply** from the parent is not favourable for it to transmit during the steady phase. It could be due to two reasons:

- It conflicts with its reception time from its own children. Remember that it sent the first **Request** to its parent only after committing reception times to its own children.
- The time left to the end of the **Cycle** from the time at which it received the **Reply**, is lesser than what the child needs to transmit **Slot Count** number of packets, *i.e.*, transmissions starting there would cross the **Cycle** boundary on the child's side.

In the case where the parent receives a **Neg-Ack** from the child, a **Time Slot** is wasted. Here arbitrations among different enqueued children can be employed. It could be done that the parent tries to schedule a different child among the ones already enqueued in the **Child List** in the same **Time Slot** in the next cycle and doesn't update its first available **Time Slot**. For sake of simplicity we don't take that approach.

Ack From The Child

In the case where the child has no issues with the transmission time assigned by the parent, it sends **Ack** to the parent. On reception of this **Ack**, the parent, that was monitoring this **Time Slot** for **Ack** or **Neg-Ack** by the child, updates its first available **Time Slot** and shifts it forward by child's **Slot Count** number of **Time Slots**. It updates its reception time based on this. It also updates its own **Slot Count** by incrementing it with the child's **Slot Count**. The child finalises its sleep/wakeup schedule. In the steay phase, it will sleep from the start of the **Cycle** to the start of the reception time, from the end of the reception time to the start of the transmission time, and from the end of the transmission time to the end of the **Cycle**. Based on the status of the **Child List** the parent takes the following decisions:

- Incase this was the last child enqueued in the **Child List**, and the size of the list is equal to the child count, it sends **Request** to its own parent.
- Incase this was the last child enqueued in the **Child List**, and the size of the list is not equal to the child count, it waits for **Request** from its children, as it knows some of them are yet to successfully send their first request.
- Incase this was not the last child enqueued in the **Child List**, it moves on to catering the next child.

3.2.2 Example

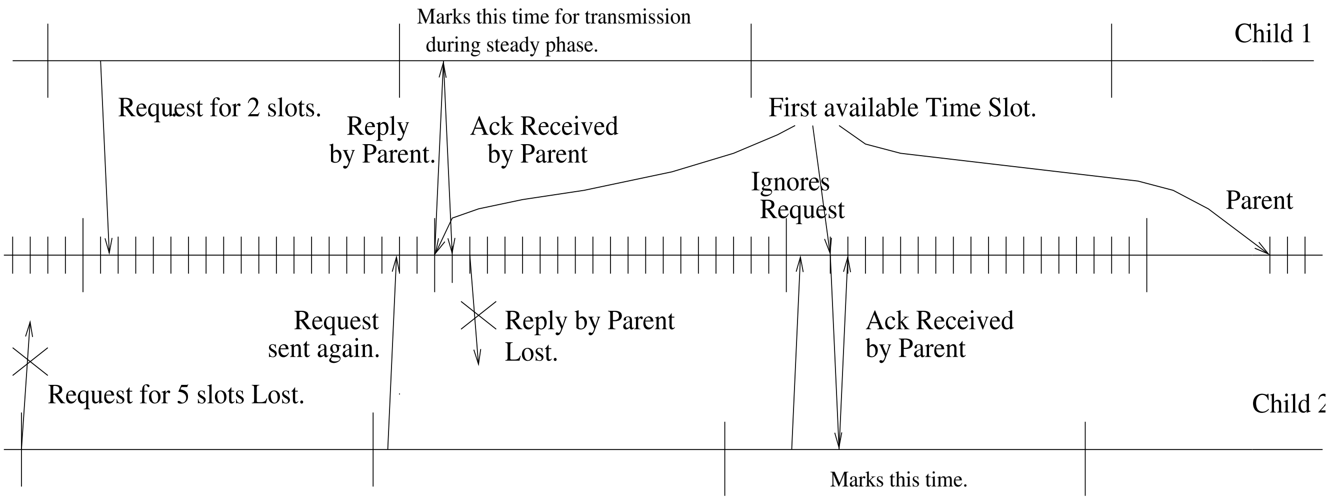

Figure 3.3: A Parent Scheduling its two children

Figure 3.3, shows an example of the scheduling algorithm at work. The **Cycle** here consists of 20 **Time Slots**.

Child 2 sends the first **Request** to the parent, which gets lost. It would send the **Request** again when the timer started by it would fire. In the meanwhile, Child 1 sends the **Request** for 2 slots and it is successfully received by the parent. The parent enqueues it in the **Child List** and would send the **Reply** in the first available **Time Slot** in the next **Cycle**, because the first available **Time Slot** in the current **Cycle**, which is at the beginning of the **Cycle**, is already over. The timer with Child 2 fires and it sends again the **Request** for 5 time slots, which is successfully received by the parent this time. The parent enqueues it also. Now in the first available **Time Slot** in the next **Cycle**, the parent sends the **Reply** to Child 1. Child 1 turns OFF its timer and accepts the time assigned for transmission and sends back the **Ack**. As the parent gets back the **Ack** within the **Time Slot** it was monitoring, it doesn't schedule to send the **Reply** to Child 1 again. It updates the first available **Time Slot** by advancing it ahead by 2 slots.

Now Child 1 is successfully scheduled. Parent sends back the **Reply** in the first available **Time Slot** to Child 2, which gets lost. The parent doesn't receive anything in the monitored **Time Slot** and schedules to send the **Reply** again in the same **Time Slot**, in the next **Cycle**. The timer with Child 2 fires again and it sends the **Request** again. This is ignored by the parent as Child 2 is already enqueued. On trying again in the next **Cycle**, the **Reply** by the parent is received by Child 2 and it turns OFF the timer. It sends back the **Ack** and the parent further advances the first available **Time Slot** by 5 slots.

3.2.3 DeadLock Handling

In the case that parent doesn't receive any **Ack** or **Neg-Ack** within the monitored **Time Slot**, it tries again by sending the **Reply** at the start of the same **Time Slot** in the next **Cycle**. This could be due to the loss of the parent's own **Reply** or the child's **Ack** or **Neg-Ack**. Let's call the parent in consideration as node A, and its child as node B. Now another parent, say node C, could be trying to schedule one of its child, say node D, in some **Time Slot** and these slots overlap with reference to the operation of the network irrespective of where they are placed in their respective **Cycle**. It could be that:

- A parent in one of the parent-child pairs is in transmission range of the other parent.
- A child in one of the pairs is in transmission range of the other child.
- A parent in one of the pairs is in transmission range of the child of the other pair, say node C is in transmission range of node B.

In the first case, collision will be there as **Reply** of both the parents will collide. In the second case, the **Ack** or **Neg-Ack** of one child will collide with that of the other child. Third case is a *hidden node problem*, and node C would not be able to receive the **Reply** from node D due to interference from node B, as both B and D would be transmitting their **Ack** or **Neg-Ack** at the same time. Figure 3.4 elaborates on this problem.

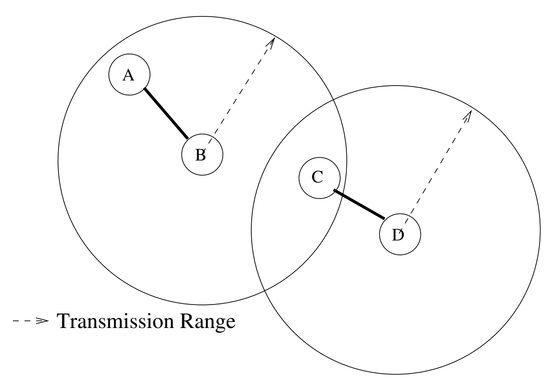

Figure 3.4: The Hidden Node Problem in Scheduling Phase

This collision deadlock will continue in every **Cycle**, as each parent would send the **Reply** again in the same **Time Slot** in the next **Cycle** and collisions will again follow. To come out of this deadlock, each parent can maintain a counter of how many times the **Reply** has been sent to the same child in the same **Time Slot** and based on that count and a random toss, the parent can update the next available **Time Slot** by one slot and send the **Reply** in that slot.

3.2.4 Phase Change Notification

The length of this phase is variable, based on how the scheduling actually proceeds, and is a multiple of the **Cycle**. After the AP also successfully schedules all its children, this phase is compelete. The AP then broadcasts, notifying the whole network that the scheduling phase is over. This broadcast is similar to the broadcast done in the topology learning phase, except that no change to the packet is required while re-broadcasting. This broadcast phase time is again decided based on the size of the network as in topology learning phase and after this time the data routing phase starts.

3.3 Data Routing Phase

In this phase, the nodes follow their sleep/wakeup schedules and the data routing is done to the AP.

The mechanism for adapting to changes in the topology is not event-driven as in [4]. It is achieved by re-initiation done by the AP. This re-initiation, important for handling clock drifts also as discussed earlier, is done based on a timer that is common to all the nodes in the network. Thus the whole topology is refreshed and all three phases repeated based on this timer.

4 Simulation Results and Analysis

In this chapter we describe the simulations done on the system and we analyse and explain the results. But prior to that we discuss the platform these simulations have been performed at.

4.1 TOSSIM

We simulated our protocol using TOSSIM[1] and TinyOS 1.1[9]. TOSSIM[1] is a discrete event simulator for TinyOS[9] and is based on its event-driven execution model. It is a *software-in-the-loop* simulator and incorporates the actual node software in to the simulation and as such the TOSSIM[1] simulation compiles directly from the TinyOS[9] code used to implement the system. It uses powerful abstractions to model a sensor network. A network is a directed graph, where each vertex is a sensor node and each directed edge has a bit-error rate and by setting these, wireless channel characteristics such as lossy and asymmetric links and hidden node problems can easily be modeled. The settings of the network have been mentioned in the results.

4.2 PowerTOSSIM

In simulation based studies, the estimation of the power consumption of the network and the motes individually has previously been done by ad hoc methods, as TOSSIM [1] in itself provides no support for power consumption estimation. In [18], the network lifetime results are based on the use of power consumption values similar to the ones available for the Berkeley motes. Here they implement a new simulator that idealizes the physical and MAC layers and no collisions occur and transmission time for each packet has been assumed to be a constant. In case of [3], where the simulation platform is TOSSIM[1], the estimate for the average lifetime is done based on network connectivity, similar to the method adopted in [19], where power dissipation of a node depends on the number of its children and the grouping techniques adopted. In some other systems, for power calculations, changes are made to the GenericComm component of the mica radiostack [6], to count transmit, receive, idle listening, and radio sleep time. These time values are then use alongwith the corresponding current draw values as in [18].

For power consumption calculations, we use PowerTOSSIM [2], which is a power modeling extension to TOSSIM [1]. It is based on the power model of mica2 mote. It adds

power state tracking module to the TinyOS [9] architecture and makes modifications to other modules to report transitions. The power profiling with it is much more accurate as it also incorporates the energy consumed by sensors and leds and also estimates CPU energy usage recording the runtime basic block execution counts and mapping these blocks to the cycles used on Atmel AVR microcontroller instructions. This way it also accounts for the energy consumed in computational overheads incurred while trying to save the radio power by scheduling. The output format of PowerTOSSIM[2] for power consumption of a mote is shown below in a trace taken from one of our results during power profiling. It generates results for all the motes in the network in the similar format. All values are in *millijoules*.
Mote 0, cpu total: 1217.637555
Mote 0, radio total: 2008.234210
Mote 0, adc total: 0.000000
Mote 0, leds total: 0.000000
Mote 0, sensor total: 203.430972
Mote 0, eeprom total: 0.000000
Mote 0, cpu_cycle total: 0.000000
Mote 0, Total energy: 3429.302737

4.3 Simulation Results

We have done our simulations over various topologies, but for the results here we have used the topology as shown in Figure 4.1, consisting of 20 nodes. The nodes with an edge between them are in the transmission range of each other. All links are symmetric and have BER(Bit Error Rate) of 0. The dark edges show the parent-child relationships set-up by the Min-Hop distance based routing algorithm. The Table 4.1 summarizes the important time values used in the simulation. Re-Initiation timer is dependent on how

Parent Selection Phase	10000 ms
Child Count Phase	1000 ms
Cycle Time	5000 ms
Time Slot Length	100 ms
Phase Change Notification	10000 ms
Re-Initiation	1000000 ms

Table 4.1: Various time values used in simulation

much clock skew can be accumulated over time. For mica motes the skew is of the order of 1 ms in 50,000 ms. Assuming the parent only transmits in the middle of the time slot and it takes 10 ms for the exchange to be over, we have a 45 ms cushion for clock skew. As such, even a 30 minute Re-Initiation timer would be safe. The Re-Initiation timer

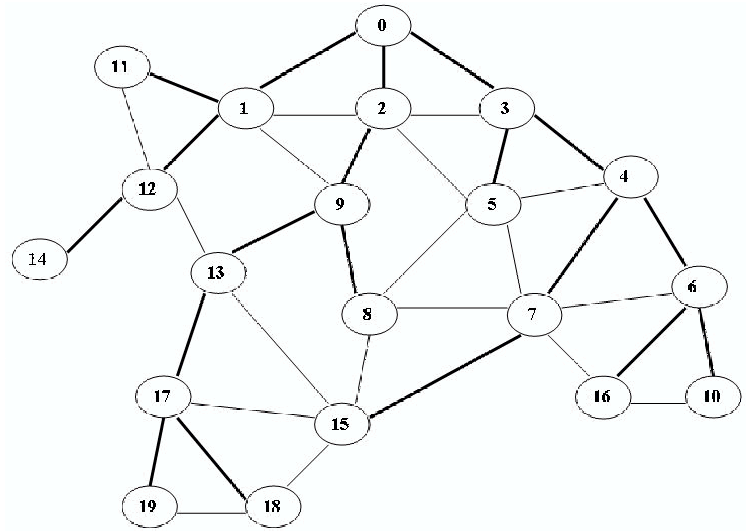

Figure 4.1: The Topolgy used for simulations

could be larger if the time slot is larger. Apart from the time values in Table 4.1, the sense time for a node has been kept 13 ms and is at the beginning of the cycle. Each node turns on the radio for atleast one time slot, *i.e.*, 10 ms, even if it is a leaf node. And the transmission time can only start after one time slot from where the reception time ends.

In Figure 4.2 and Figure 4.3 we see the radio power consumed by the nodes over a particular path in the spanning tree during the steady phase, *i.e*, the data routing phase. Zero values correspond to radio OFF, intermediate level corresponds to the receptions, and the peaks correspond to the transmissions. The receive and transmit times have been kept larger than required to allow for retransmissions in a lossy topology.

Figure 4.2 shows the results of radio energy consumption over time for the path consisting of nodes 19, 17, 13, 9, 2 and 0 , *i.e*, the AP. Figure 4.2 shows the results for the a different path consisting of nodes 16, 6, 4, 3, and the AP.

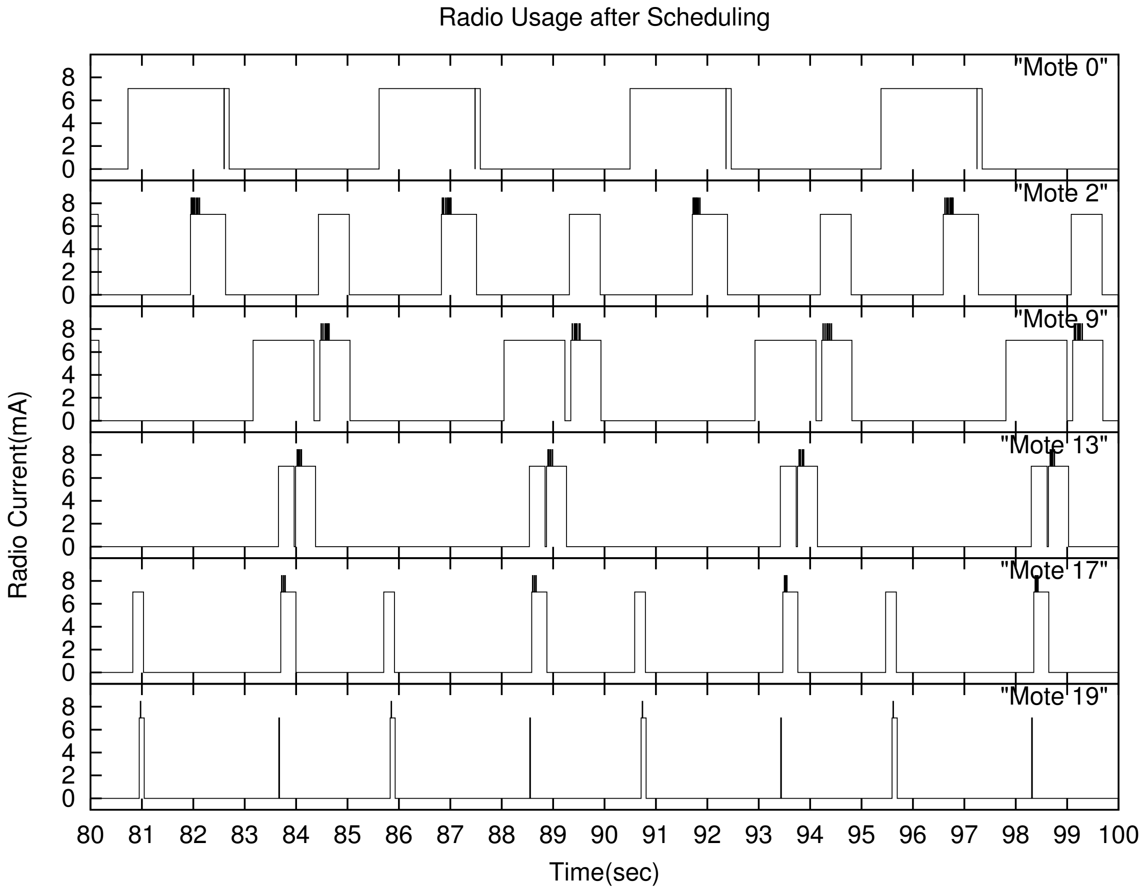

Figure 4.2: The Radio Energy Consumption With Time

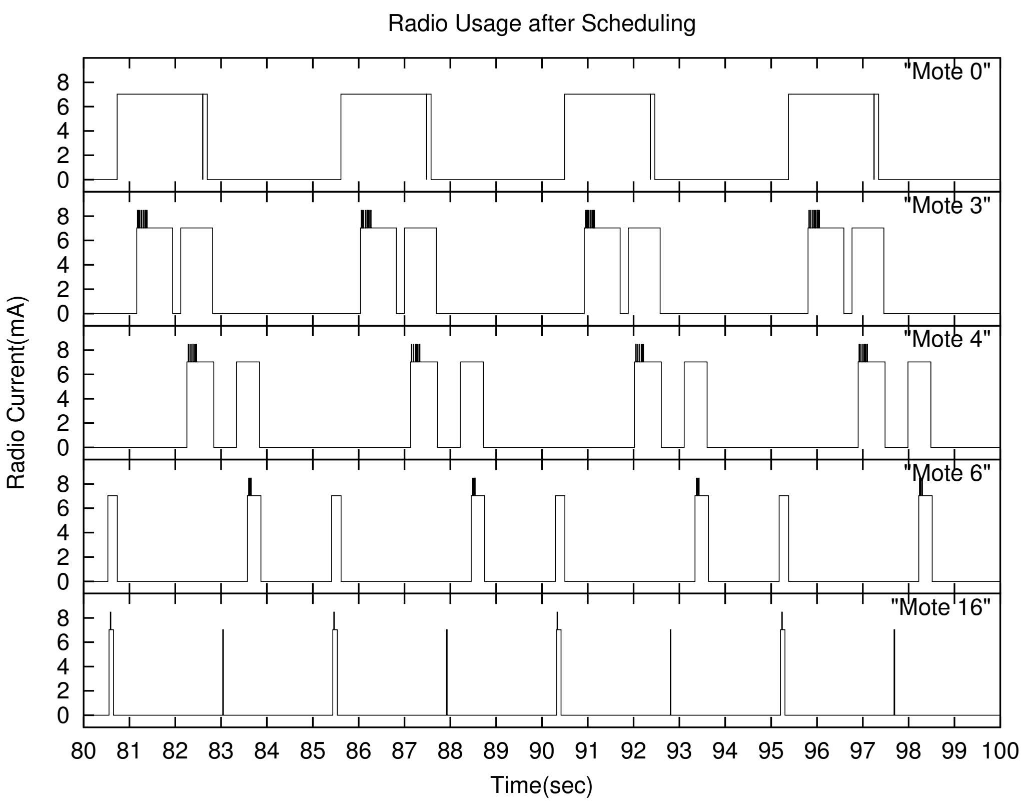

Figure 4.3: The Radio Energy Consumption With Time

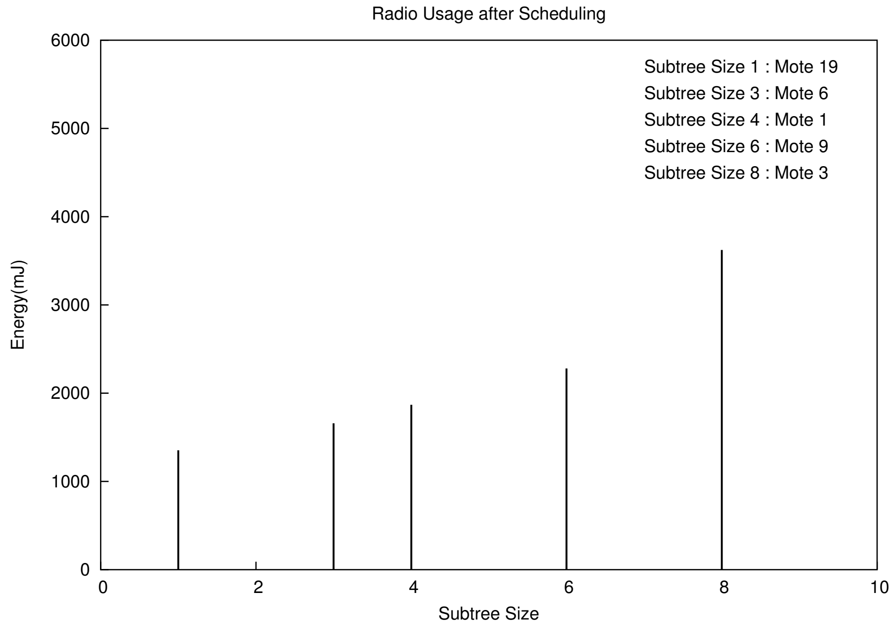

Figure 4.4: Comparison of Power Savings among nodes

In Figure 4.4, we compare the energy savings over the nodes. The results show that leaf nodes are able to save maximum amount of energy as they sleep most of the times, and with nodes nearer to the AP, as the wakeup times are more, the energy savings are lesser. In Figure 4.5, we make a comparison of the contention based scheme and our

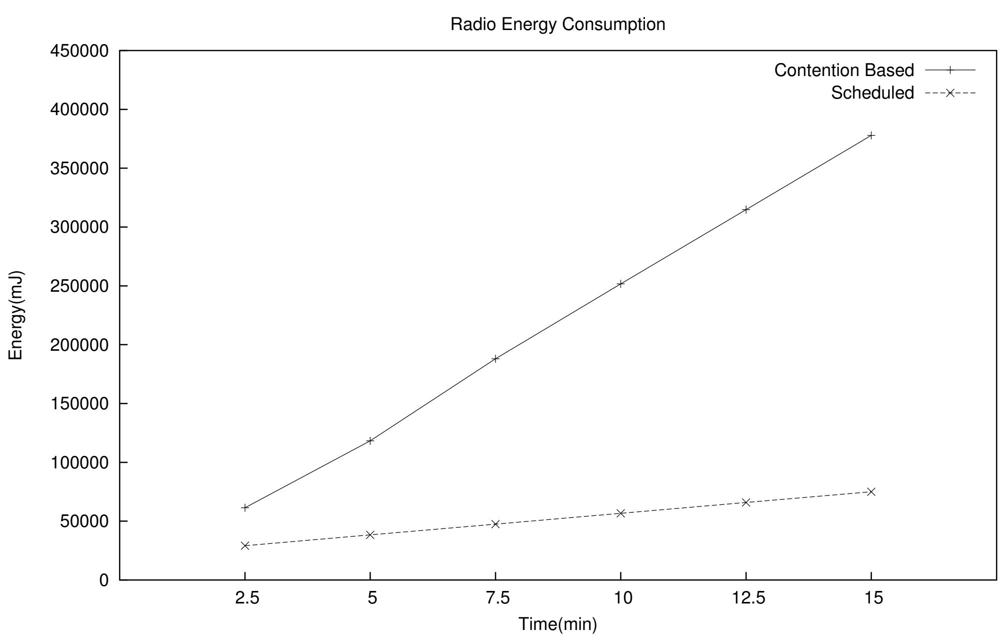

Figure 4.5: Radio Power Consumed in the two schemes

algorithm with respect to the consumption of radio power. The results show that over time, our algorithm performs far better than the contention based scheme with an order decrease in power consumption achieved within 15 minutes of operation of the network. Figure 4.6 makes a comparison of the toal power consumed over time. With time, the radio power consumed starts dominating and the gains made there reflect in total power consumption also.

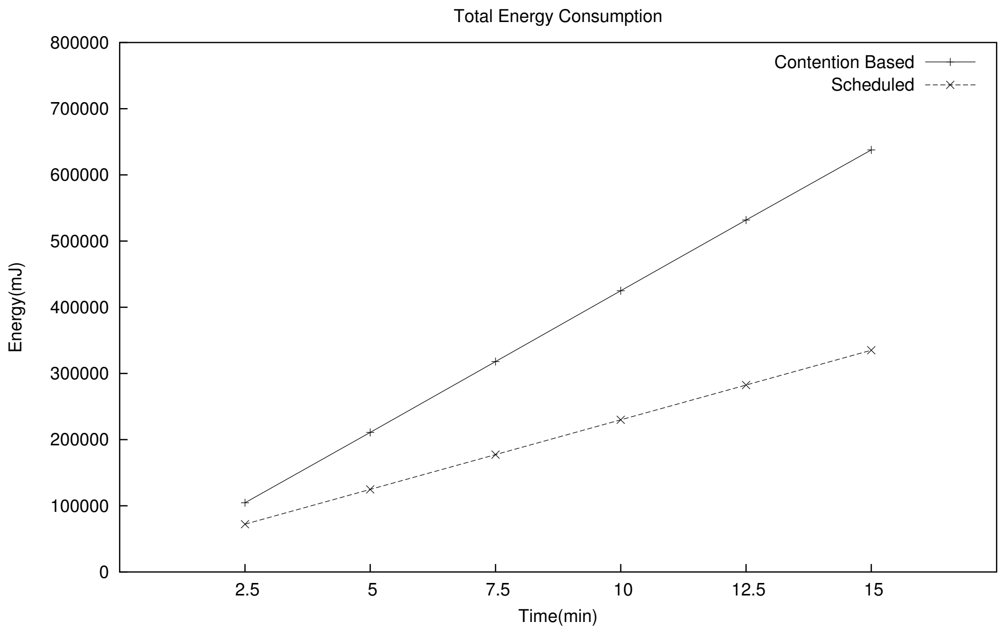

Figure 4.6: Total Power Consumed in the two schemes

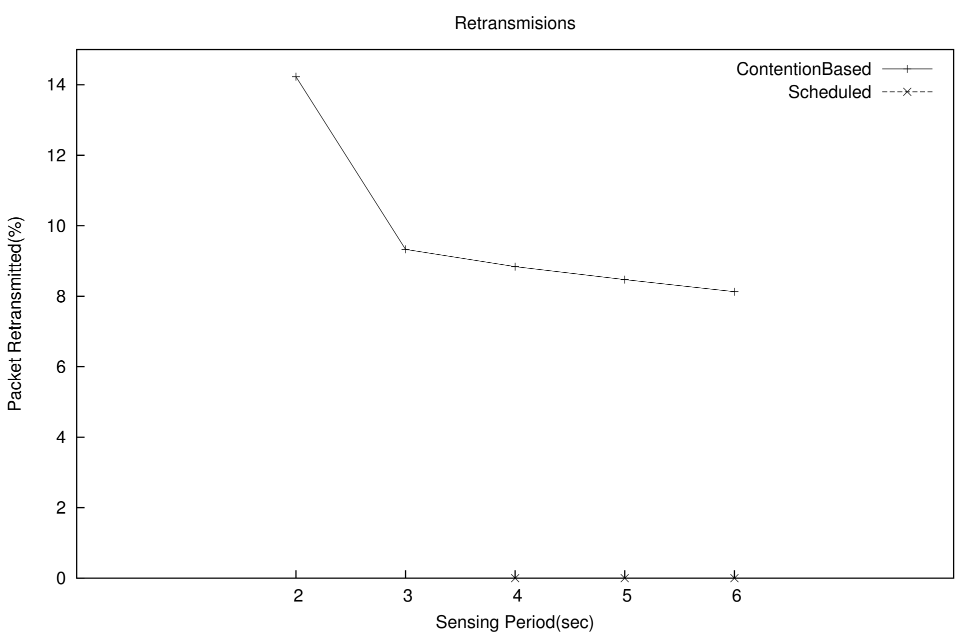

Figure 4.7: Percentage Retransmissions Done

Figure 4.7 shows the packet retransmissions done in the contention scheme. In comparison, there are no retransmissions in our implementation at all. As the sensing rate of the nodes decreases, *i.e*, the length of one cycle is made smaller, the retransmissions in the contention scheme increase. The scheduling ensures no retransmissions in our case but the cycle length has to be enough for the scheduling algorithm to work. Our algorithm, in case of the topology taken, can only schedule given the cycle is greater than 4 seconds. The results here correspond to 300 seconds of operation of the network.

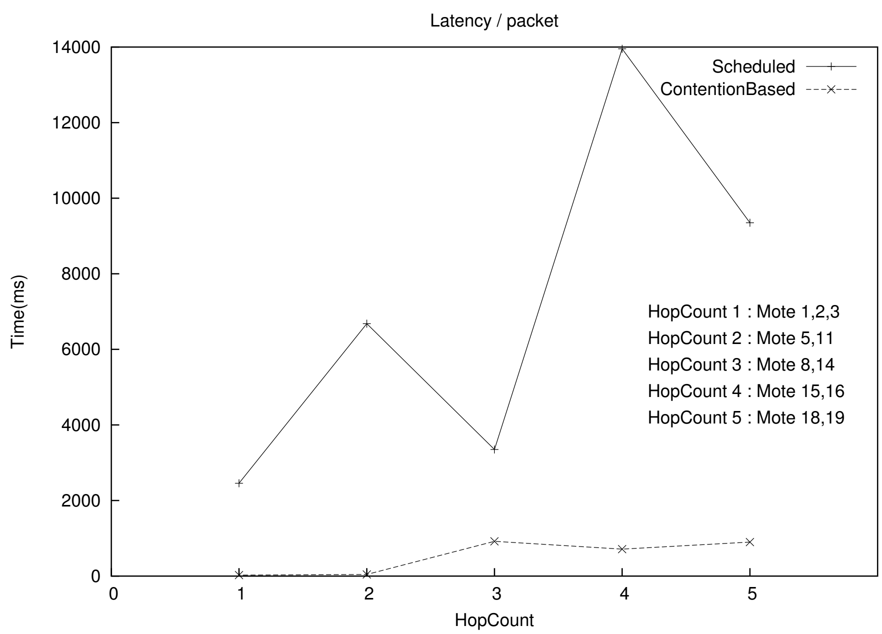

Figure 4.8: Latency Comparison in the two schemes

Figure 4.8 shows the latency in reporting the data to the AP in the 2 schemes. The behaviour of our algorithm is easily explainable from the topology and Figures 4.2 and 4.3. This is due to the store and forward nature of our algorithm. The upper limit on the latency of data of a node in our algorithm is its hop-count times the **Cycle** length. The simulation time for deriving this result has been 300 seconds.

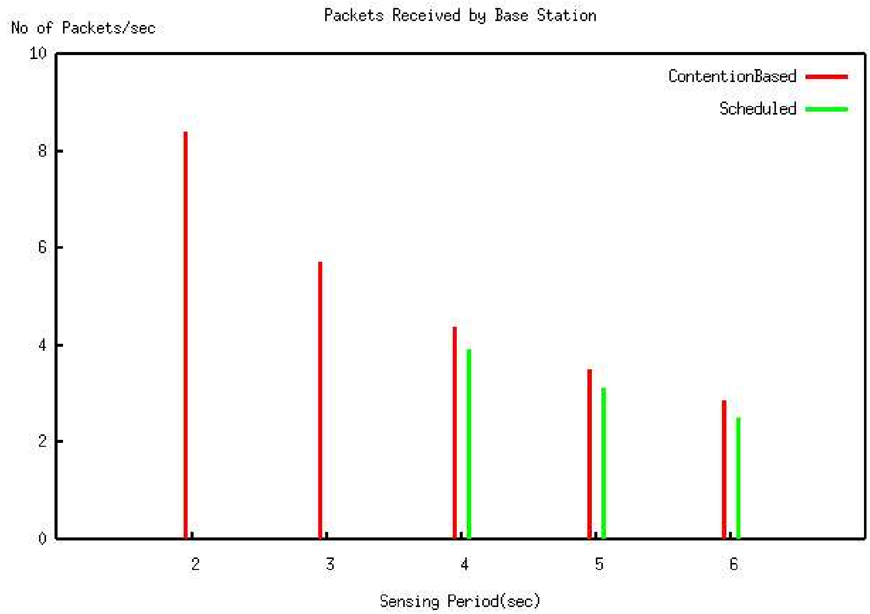

Figure 4.9: Throughput Comparison in the two schemes

In Figure 4.9, the throughput is reported as number of packets received by the AP per second, against the sensing rate. The througput values in our case are lesser as the latency is higher. The difference corresponds to the latency of the first cycle only and fades away over time. Again this result is derived from 300 seconds of operation of the network.

5 Implementation

5.1 TinyOS and nesC

We have implemented the system in nesC version 1.1 [7]. nesC [7] [8] is an extension of C and was designed to support and evolve TinyOS's programming model and to reimplement TinyOS in the language itself. TinyOS [9] is an event driven operating system designed specifically for mica mote platform and provides minimal device and networking abstractions. It has a component architecture and it provides a library as a set of reusable system software components which are organised in to layers, with lower layers "closer" to hardware and higher layers "closer" to the application. Thus a TinyOS [9] application is implemented by wiring the reusable components and also the newer ones implemented , together, as specified in a top level configuration file. As different OS services have been decomposed in to separate components in the component library, only the necessary ones are compiled with the application, keeping the footprint of the application code really small. This is highly desirable keeping in view the memory constraints of the mica motes, mentioned in Table 2.1. Again, to ensure the small size of the footprint, TinyOS has no file system, its supports only static memory allocation, and has simple FIFO based task model. All these constraints make coding up the system in nesC a real challenge.

5.2 Implementation Details

The building blocks of the implementation are TinyOS components. These can be of two types, configurations and modules, and are represented as elliptical shapes in figures in this section. Interfaces among the components are represented as solid arrows. Component at solid arrow's tail uses the interface and the one at the head provides the interface. Hashed arrow also shows an interface, but here the interface provided by the tail component is 'equivalent to' the implementation in the head component. The main configuration file as shown in Figure 5.1, is Surge. It connects various 'used interfaces' of other configurations and modules to required 'provided interfaces'. Its 'interface StdControl' is connected to different components to initialize and start them. 'Module BcastM' and 'module ProcessMsg' together create the routing tree based on Min-Hop distance. A Timer is started in both modules, which on firing recreates the routing tree and reinitiates various parameters. This parameter reinitiation is done using 'event startReconf()' of 'interface reconf'. 'Module Processmsg' after PARENT_SELECTION_TIME signals

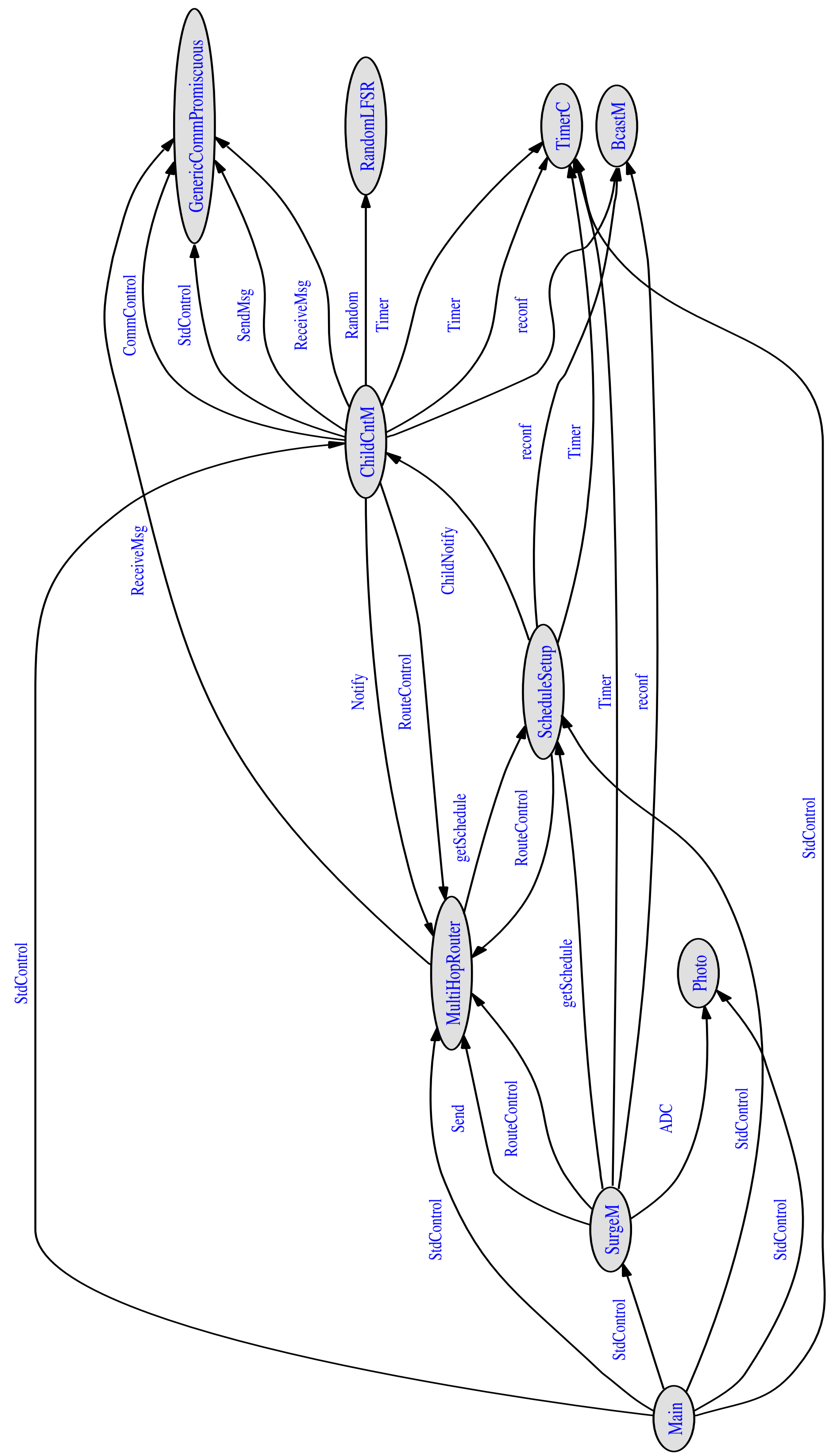

Figure 5.1: Implementation details of the application.

'event IsParentSet()' of 'interface Notify'.

'Module ChildCntM' captures this event and starts child count phase, in which each parent know about number of its immediate children. After CHILD_COUNT_TIME 'Module ProcessMsg' signals 'event IsChildCnt()' of 'interface ChildNotify', which is captured by 'configuration ScheduleSetup'.

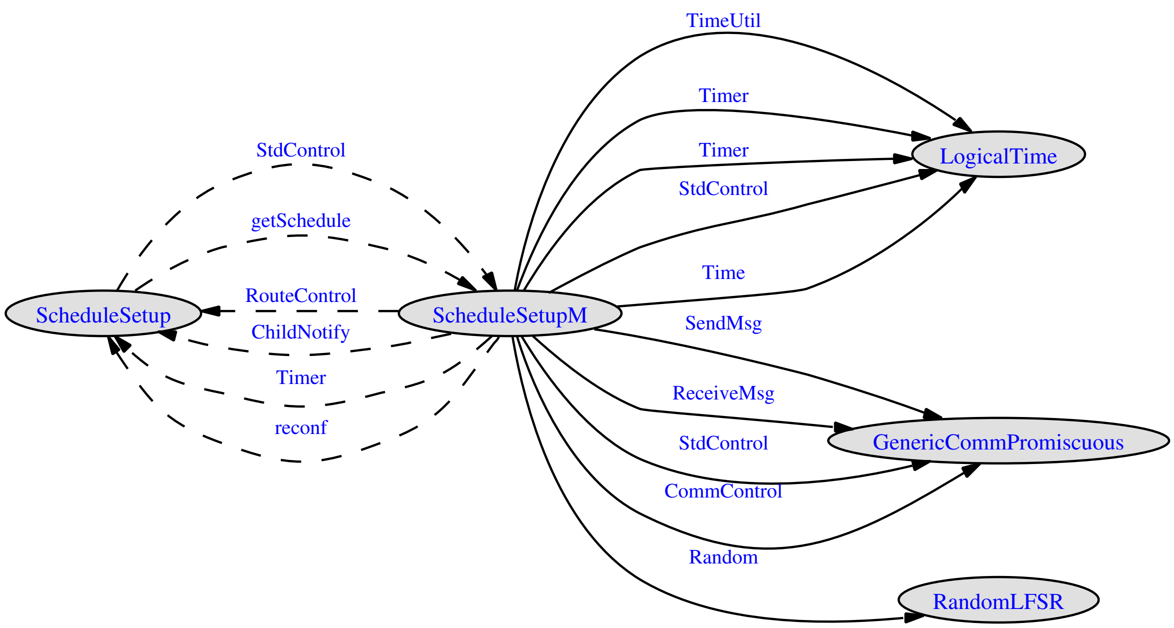

Figure 5.2: Configuration ScheduleSetup implementation

'Configuration ScheduleSetup' in Figure 5.2 implements its 'interface ChildNotify' handler in 'module ScheduleSetupM'. 'Module ScheduleSetupM' computes schedule using 'interface RouteControl' provided by 'configuration MultiHopRouter'. After computing the schedule, it synchronizes its sensing timer cycle with the sensing timer cycle of 'module SurgeM' using 'event syncTime()' of 'interface getSchedule'. When the whole network is scheduled, AP broadcasts phase change packet, which is flooded in the network by re-broadcasts from each node. Each node after PHASE_CHANGE_TIME, after receiving first phase change packet, signals 'event scheduleDone()' of 'interface getSchedule'. 'Module SurgeM' captures this event and starts taking sensor reading.

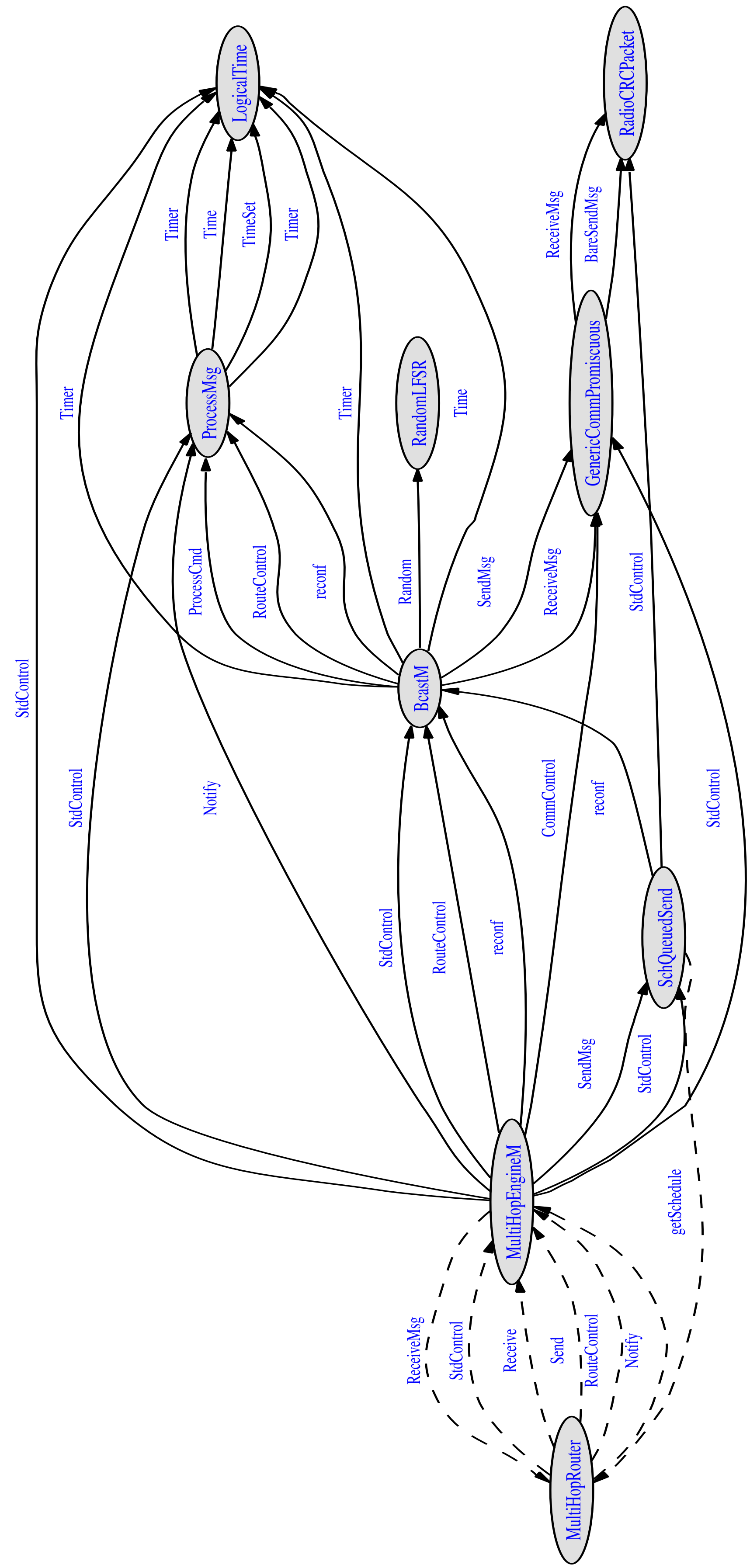

Figure 5.3: Configuration MultiHopRouter.nc implementation

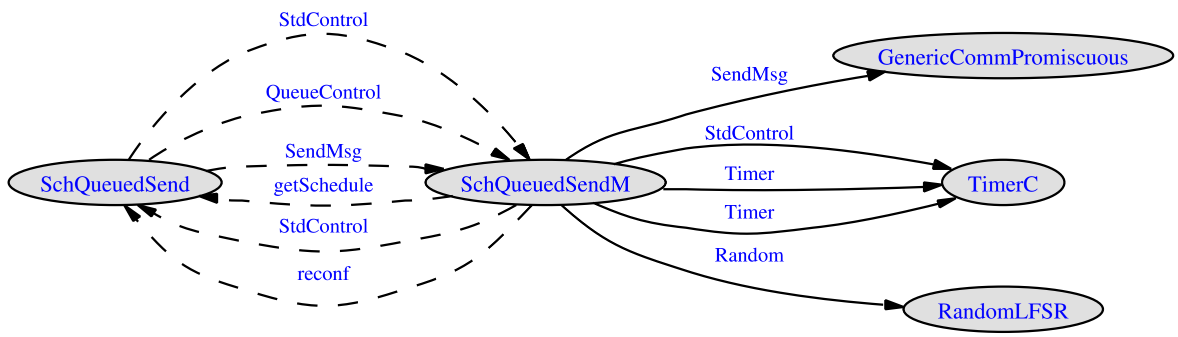

Figure 5.4: Configuration SchQueuedSend implementation

The sensor reading then is sent to its parent using 'interface Send' implemented by 'configuration MultiHopRouter' as shown in Figure 5.3. Packets are sent by 'module MultiHopEngineM' and 'configuration SchQueuedSend', Figure 5.4, according to schedule provided by 'configuration ScheduleSetup'. Radio power control is used in 'module SchQueuedSendM' and implemented by 'configuration RadioCRCPacket'. Transmission and reception of packets is implemented internally by 'configuration GenericCommPromiscuous' and 'module AMPromiscuous'. Radio stack used is CC1000Radio stack provided for simulation.

6 Conclusions and Future Work

In this thesis, we presented a distributed scheduling algorithm that efficiently computes the sleep/wakeup cycles for each node in the wireless sensor network.

The schedule is calculated in a separate schedule formation phase and each node follows its schedule to send the data to the AP over multi-hop. No control packets are required during the steady-phase, which is the phase the network is in for the longest time.

Each node sets up its schedule according to its local time only to avoid overhead of time-synchronisation. The periodic re-inititation of all the three phases effectively handles the topology changes and the clock drifts.

The simulations over a random and elaborate topology of 20 nodes show that within 15 simulation minutes of running the application on TOSSIM, our algorithm achieves almost an order reduction in the radio power consumed as compared to the contention based scheme for media access.

6.1 Scope Of Improvement

In our algorithm, each parent is assumed to have enough slots to meet the requirements of all its children, but it might be constrained from the buffer space it has as every node has to store together packets equal to the size of the subtree rooted at itself, before relaying them off to the parent.

6.1.1 Proposed Solution

Currently each node requests its parent only when all its children have been scheduled. But if the request from one of its children exceeds its buffer space left, the parent can partly schedule the child for number of slots equal to the buffer space left in terms of packet size. In that case, it sets a flag in the request to its own parent. And that parent node again sets this flag while requesting its own parent. This way each parent can keep track of the route over which the scheduling is yet to be completed. Once the partial scheduling is done with the AP also, it reinitiates the scheduling along the required routes and the parents start with their child list to schedule the partly scheduled and yet to be scheduled nodes.

Bibliography

[1] Phil Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim:Accurate and Scalable Simulation of entire TinyOS Applications.In *First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003.

[2] Victor Shnayder, Mark Hempstead, Bor-rong Chen, and Matt Welsh, Harvard University. PowerTOSSIM:Efficient Power Simulation for TinyOS Applications.*SenSys 2004*, November 2004.

[3] Sinem Coleri, Anuj Puri, Pravin Varaiya. Power Efficient System for Sensor Networks. In *Proceedings of the Eighth IEEE International Symposium on Computers and Communication (ISCC 2003)*, July 2003.

[4] Khaled A. Arisha, Moustafa A. Youssef, Mohamed F. Younis. Energy-Aware TDMA-Based MAC for Sensor Networks.

[5] Tian He, Bruce Krogh, Sudha Krishnamurthy, John A. Stankovic, Tarek Abdelzaher, Ligian Luo, Radu Storelu, Ting Yan,Lin Gu, Jonathan Hui. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services (MobiSys 2004)*, June 2004.

[6] Nelson Lee, Philip Levis, Jason Hill. Mica High Speed Radio Stack. September 2003.

[7] David Gay, Philip Levis, David Culler, Eric Brewer. nesC 1.1 Langage Reference Manual. May 2003.

[8] The nesC Langugage: A Holistic Approach to Networked Embedded Systems. David Gay, Philip Levis, David Culler, Eric Brewer, Matt Welsh, Robert von Behren. http://nescc.sourceforge.net

[9] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networks sensors, November 2000.

[10] Wei Ye, John Heidemann, and Deborah Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks.In *Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM 2002)*, June 2002.

[11] V. Rajendran, K. Obraczka, and J.J Gracia-Luna-Aceves. Energy-efficient, collision free medium access control for wireless sensor networks.In *ACM SenSys 2003*, November 2003.

[12] Simon Coleri. PEDAMACS:power efficient and delay aware medium access protocol for sensor networks,2002.

[13] S. Singh and C. Raghvendra. Pamas: Power aware multi-access protocol with signalling for ad hoc networks, 1999.

[14] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks,2003. UCLA.

[15] Feng Zhao, Leonidas J. Guibas. Wireless Sensor Networks: An Information Processing Approach. Morgan Kauffman Publishers.

[16] TinyOS tutorial. URL:http://www.tinyos.net/tinyos-1.x/doc/tutorial.

[17] Bar bara A. Holt, Lance R. Doherty, and Eric A. Brewer. Flexible Power Sceduling for Sensor Networks. In *Proceedings of Third International Symposium on Information Processing In Sensor Networks (IPSN 2004)*, April 2004.

[18] Mihail L. Sichitiu. Cross-Layer Scheduling for Power Efficiency in Wireless Sensor Networks. *IEEE INFOCOM 2004*, March 2004.

[19] Sinem Coleri, Mustafa Ergen, T. John Koo. Lifetime Analysis of a Sensor Network with Hybrid Automata Modelling.In *Proceedings of First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, September 2002.

[20] Max-Min Fair Collision-Free Scheduling for Wireless Sensor Networks. Avinash Sridharan, and Bhaskar Krishnamachari. Department of Electrical Engineering, University of Southern California.

[21] A Coverage-Preserving Node Scheduling Scheme for Large Wireless Sensor Networks. Di Tian, and Nicholas D. Georgonas, University of Ottawa.

[22] Taming the Underlying Challenges of Reliable Mulithop Routing in Sensor Networks. Alec Woo, Terence Tong, David Culler.In *Proceedings of SenSys 2003*, November 2003.

[23] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronisation using refernce broadcasts. UCLA, Feb 2002.