CLUSTER-BASED ENERGY-EFFICIENT MAC PROTOCOL FOR WIRELESS SENSOR NETWORKS

A Thesis by

Nikhil Marrapu

Bachelor of Technology, Jawaharlal Nehru Technological University, 2008

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

December 2010

CLUSTER-BASED ENERGY-EFFICIENT MAC PROTOCOL FOR WIRELESS SENSOR NETWORKS

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

_____

Neeraj Jaggi, Committee Chair

_____

Vinod Namboodiri, Committee Member

_____

Hamid M. Lankarani, Committee Member

DEDICATION

To my friends at Wichita State University.

ACKNOWLEDGEMENTS

It gives me great pleasure to write a few lines about people who helped and motivated me to complete my thesis. First of all, I would like to thank my professor and adviser, Dr. Neeraj. I have learned many things from Dr. Neeraj through his suggestions and witty comments. I am greatly indebted to him for everything he has taught me. It was really a good experience working under his guidance.

My family plays a major role in my achievement, and without them, I would not have been what I am today. I am fortunate to be a part of such a great family. My parents and siblings have always supported me in whatever I believed and executed.

My friends Vamshi Krishna, Sreenivas, and Umesh need special mention here. They have helped me at various stages of my thesis and have been good companions every second of the time I spent at Wichita State University. My research group was a spirited driving force in helping me achieve many things during the course of my thesis. Interacting with them made me grow technically in several directions. I would also like to thank my friends Abhinav, Anudeep, Avinash, Lakshman, Nishant, Vummadi, and Vinay, who made Wichita State University feel like home in every aspect.

I sincerely thank Dr. Vinod and Dr. Lankrani for reviewing my thesis and providing valuable suggestions.

ABSTRACT

The Wireless Sensor Networks (WSNs) employs thousands of small sensors that communicate between themselves in a distributed manner using Medium Access Control (MAC) protocols. The energy required for wireless sensors is obtained from non-rechargeable energy sources. Due to their small size, wireless sensors are highly constrained in terms of battery energy. Hence, energy efficiency is considered a key factor in the design of a WSN. MAC protocols play an important role in the successful operation of WSNs. Energy efficiency can be achieved by introducing some significant changes that effect the energy consumption at the MAC layer. Existing protocols achieve energy savings, trading off either latency or throughput. The Sensor Medium Access Control (S-MAC) is one such protocol that identifies a few sources of energy wastage and proposes an adaptive sleep-and-listen scheme to minimize energy wastage.

This thesis studies the S-MAC protocol and proposes a new one Cluster-Based Energy-Efficient Medium Access Control (CBMAC), which attempts to increase energy savings by introducing changes to the existing S-MAC protocol. CBMAC introduces a clustering mechanism, in which the nodes form clusters and elect a cluster head. The cluster head takes care of data transfer and synchronization issues, while the cluster nodes are allowed to spend maximum time in the sleep state. The role of the cluster head is shared among several nodes over time in order to achieve uniform energy utilization in the network. The energy savings achieved with the CBMAC protocol are primarily due to the increased sleep-time fraction for cluster nodes. The clustering mechanism also reduces control overhead, which is prevalent in the S-MAC protocol due to the periodic control packet exchanges. CBMAC was studied under various topologies, and the results show significant energy savings over the S-MAC protocol, particularly in low-data-traffic scenarios.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES (continued)

Figure **Page**

LIST OF ABBREVIATIONS / NOMENCLATURE

ACK	Acknowledgement
BMAC	Berkeley Medium Access Control
CBMAC	Cluster-Based Medium Access Control
CDMA	Code Division Multiple Access
CH	Cluster Head
CMAC	Convergent Medium Access Control
CTS	Clear To Send
CW	Contention Window
DATA	Data Packet
DIFS	Distributed Inter-Frame Space
DCF	Distributed Coordination Function
DSR	Dynamic Source Routing
EIFS	Extended Inter-Frame Space
IAT	Inter-Arrival Times
IEEE	the Institute of Electrical and Electronics Engineers
LAN	Local Area Netwrok
MAC	Medium Access Control
MACAW	Multiple Access with Collision Avoidance for Wireless
NAV	Network Allocation Vector
NOAH	NO-AdHoc
NS-2	Network Simulator-2

LIST OF ABBREVIATIONS / NOMENCLATURE (continued)

PAMAS	Power Aware Multi-Access protocol
PMAC	Pattern Medium Access Control
RTS	Request To Send
SCPMAC	Scheduled Channel Polling Medium Access Control
SIFS	Short Inter-Frame Space
SYNC	Synchronization
S-MAC	Sensor Medium Access Control
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TMAC	Timeout Medium Access Control
WSN	Wireless Sensor Networks

CHAPTER 1

INTRODUCTION

1.1 Wireless Sensor Networks

Wireless Sensor Networks (WSNs) have attained significant importance in the recent past. A typical WSN can have tens to thousands of distributed sensors that organize themselves into a multi-hop network. Sensors are miniature devices that can connect wirelessly, and coordinate to perform a common task. Their small form factor and their ability to support wide range of applications has rendered sensor networks as one of the emerging wireless networking technology and has enabled their widespread deployment.

Sensor devices are small and are approximately 30-35 mm in diameter. A typical sensor device consists of batteries, radio antenna, sensor, and a board to accommodate all of these features. Figure 1.1 depicts a sensor and a sample topology. Given the miniature size of sensors, the vast range of applications they support is worth mentioning. Some of the potential applications include environmental sensing such as temperature sensing and rainfall detection in forests, and intruder detection in homes and other locations.

Sensor devices are inexpensive and can be deployed in large numbers. They are deployed in remote locations and places where there is less human intervention. Hence, sensors are inaccessible to replace the batteries, once they run out of energy. Moreover, the operational lifetime and computational behavior of sensor are greatly affected due to the limited energy and memory resources.

The characteristics of sensors can be summarized as follows:

- Very small in size.
- Limited memory and energy resource.
- Mainly used for sensing activities.

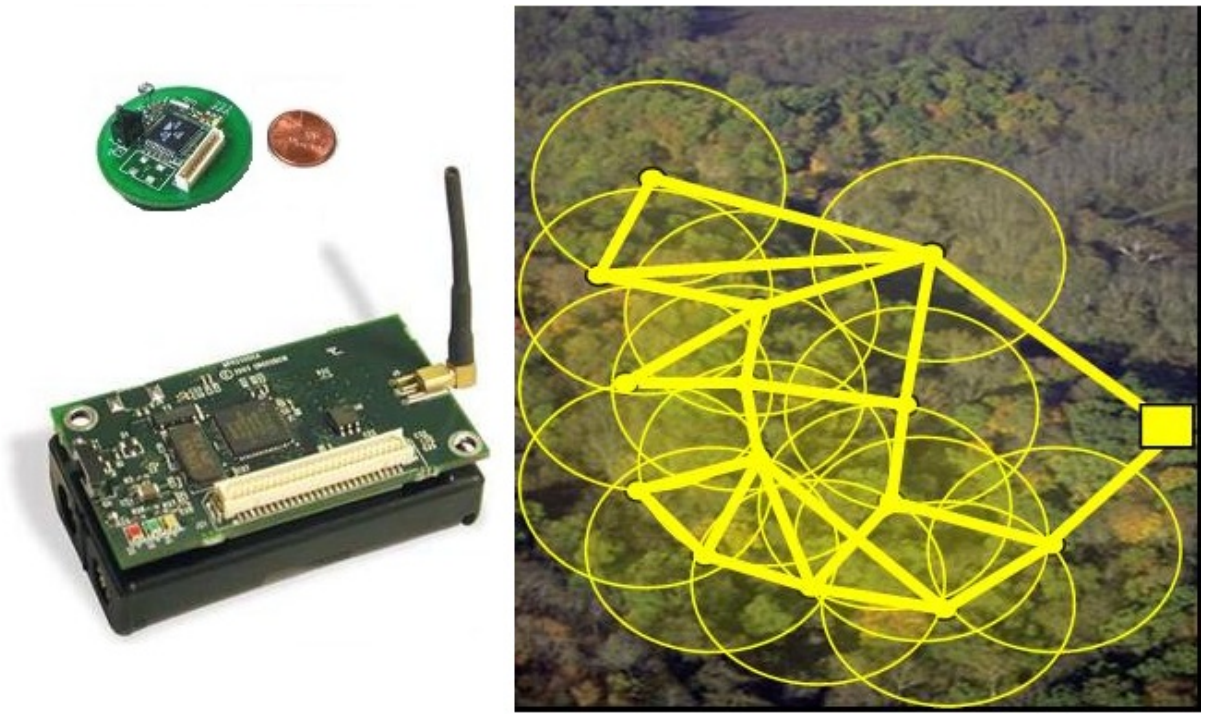

Figure 1.1 – Wireless senor network and sensors

The above characteristics crucially effect the sensor lifetime[1]. The sensor lifetime and the energy utilization are inversely related. As the energy utilization increases the sensor lifetime decreases. Generally sensors are distributed in large numbers and the connectivity between them is lost, when an intermediate sensor(s) is dead (run out of energy). Therefore energy conservation in sensor operations is a major concern.

1.2 MAC Protocols in WSN

Medium Access Control (MAC) protocols play an important role towards the the successful operation of the WSN. The primary task of a MAC protocol is to avoid collisions between the interfering nodes. Protocols like CDMA, TDMA and IEEE 802.11 are some of the well-known protocols for WSN. Several other protocols are proposed for WSN, giving importance to attributes like energy, fairness, throughput, latency and bandwidth. Among these, energy is the primary factor considered in this thesis because energy conservation helps towards achieving longer lifetime.

[1] Sensor lifetime is defined as the time difference between the time a sensor is switched on and the time it runs out of battery energy

All other parameters are assumed secondary.

Protocols like S-MAC (Sensor Medium Access Control) [1], SCPMAC (Scheduled Channel Polling Medium Access Control) [2], CMAC (Convergent Medium Access Control)[3], B-MAC (Berkeley Medium Access Control) [4], PMAC (Pattern Medium Access Control) [5] etc., have been proposed to address the energy issues in WSN. Depending on the sensor activity, three states i.e., idle, sleep and listen states are defined for sensors.

- Idle state: In idle state, sensors have no data to send or receive and they still listen to the channel.
- Listen state: In listen state, sensors may have data to send or receive and they listen to the channel. The sensor is capable of sending or receiving data in listen state. A sensor is said to be in send state, if it is sending data and is in receive state, if it is receiving data.
- Sleep state: In sleep state, sensors have no data to send or receive and they turn off their radio (do not listen to channel).

Architectures like Piconet [1] are proposed, which put sensors in periodic sleep and listen to conserve energy. When sensors are in sleep mode, they switch off their antennas and conserve energy. Maximum energy is conserved when sensors are set to spend maximum time in sleep state. However, there should be a balance between time spent in sleep state and listen state, because if a sensor spends large amount of time in sleep state, the communication with neighboring sensors is lost and the packet delays increase drastically. Delay of a packet is defined as the difference between the packet sent time and the time acknowledgement for that packet is received.

The power save mode in IEEE 802.11 DCF [6] has proposed the periodic sleep and listen mechanism in wireless sensors. But this mechanism does not synchronize the sleep times between sensors. Even when sensors are programmed with periodic sleep-listen, and are started at the same instance, over time sensors get desynchronized due to clock drift (phenomena where a sensor's clock does not run at the exact speed compared to another sensor's clock). This explains the importance of synchronization in WSN that utilize periodic sleep-listen mechanism.

S-MAC protocol has identified sources of energy wastage in WSN and has proposed an adaptive approach to counter those sources. The sources of energy wastage are identified as follows

- Collision: When a transmitted packet is lost or corrupted, the packet needs to be transmitted again. This leads to wastage of energy spent in transmitting the packet. This increases latency too.
- Over hearing: Over hearing happens when a sensor receives data that is not destined to it.
- Control packet overhead: Control packets are packets other than data packets. Sending and receiving these packets consumes energy.
- Idle listening: Listening to the channel when there are no packets to send, no packets to receive or packets that are sent are lost, is termed as idle listening.

To overcome the above sources of energy wastage S-MAC has proposed mechanisms like periodic sleep and listen, coordinated sleeping, adaptive listening and collision avoidance. S-MAC reduces the listen time in sensors by putting them to sleep state. The basic idea of periodic sleep and listen is that each sensor sleeps for a certain amount of time and wakes up to check if any other sensor wants to communicate with it. While entering the sleep state, the sensor sets its radio antenna to off state and sets a timer to awaken it. The timer value is sent to other sensors through SYNC packets. SYNC packets are exchanged between sensors at regular intervals to maintain schedule synchronization. A disadvantage of this scheme is periodic exchange of SYNC packets between all the nodes, which causes overhead. Also, latency is increased due to periodic sleep.

The listen time in S-MAC is fixed depending on the physical and MAC layer parameters. The sleep time in S-MAC can be varied by changing the duty cycle value. Duty cycle is defined as-

$$Dutycycle = \frac{listentime}{listentime + Sleeptime} \times 100 \tag{1.1}$$

The fixed listen time is a drawback in S-MAC, since even in the absence of traffic in the network, nodes operating on S-MAC are required to listen to channel for the fixed listen time.

1.3 Contributions of This Thesis.

This thesis studies S-MAC protocol and proposes significant modifications to the S-MAC protocol resulting in a new Cluster-Based Energy-Efficient MAC protocol (CBMAC), and demonstrates the advantages of the proposed approach. The fixed listen time in S-MAC is reduced to a smaller value in CBMAC and this reduces idle-listening to some extent. When the nodes have data to send or receive, the nodes operating on CBMAC extend their listen time. Actively listening to the channel when there is no traffic in the channel is called idle listening. The proposed cluster-based protocol saves considerable amount of energy by suppressing idle listening.

In CBMAC, a Cluster Head (CH) is elected for every set of one-hop nodes, collectively referred to as a cluster. The duty of CH is to perform data transfer between nodes and maintain schedule synchronization between them. As the CH handles more duties than the regular nodes, CH is provided with higher listen time than the other nodes in the cluster. In order for two clusters to communicate with each other, the concept of border node is introduced. The purpose of border node is to accept data from one CH and deliver it to another CH, in case of inter-cluster communication. As the CH listen time is more than that of the other nodes, there is a possibility for CH to run out of energy very soon. For this reason, the CH is elected periodically, and each time a different node could assume the responsibilities of a CH, leading to a more uniform energy utilization of the nodes in the cluster. Nodes operating on CBMAC protocol achieve more energy savings than nodes operating on S-MAC. The CBMAC protocol achieves energy savings with minimal tradeoffs in throughput and latency, compared to S-MAC protocol.

The rest of the thesis is organized as follows. Chapter 2 discusses the related work with emphasis on existing energy saving MAC protocols for WSN. Chapter 3 describes S-MAC protocol and its features in detail. Chapter 4 introduces CBMAC, the protocol design and features and discusses the choice of routing protocol used in the experiments performed in this thesis. The CBMAC performance in intra-cluster and inter-cluster topology is discussed in Chapter 5. Chapter 6 concludes the thesis with a discussion of the proposed protocol's future work. CBMAC has been implemented in NS-2 by making appropriate changes to S-MAC code.

CHAPTER 2

RELATED WORK

This chapter presents the related work on energy efficient MAC protocols in WSN. This chapter is divided in two sections. The first section discusses the MAC protocols that have been proposed to conserve energy and the second section presents the advantages, tradeoffs and drawbacks in S-MAC protocol.

2.1 Energy Efficient MAC Protocols

As discussed in the Chapter 1, energy conservation is considered a primary factor in WSN. The sensor lifetime is greatly effected by the energy consumption in the sensors, and hence efficiently utilizing energy at each sensor node in the network prolongs the network lifetime. Design of energy efficient MAC protocols for sensor networks is an active research area. MAC protocols achieve energy efficiency either by trading off throughput or latency or fairness. Some of the well known energy efficient MAC protocols are discussed in this section.

IEEE 802.11 distributed coordination function (DCF) [6] is a good example of contention-based protocol, and is mainly built on the research protocol MACAW [9]. IEEE 802.11 is widely known for its robustness to hidden terminal problem and is used in ad hoc wireless networks. However, recent studies [10] has shown that the energy consumption using this protocol is very high when nodes are in idle mode. Nodes are in idle mode, when they are not part of an active communication process. Singh and Raghavendran proposed PAMAS [7], to improve energy savings by avoiding the overhearing among neighbor nodes. Overhearing is a part of idle mode, and is considered as a source of energy waste. In PAMAS RTS-CTS exchanges are performed over a signaling channel and data transmissions are done over a separate data channel. Nodes listen to the signaling channel to determine when it is optimal to power down transceivers (antenna) to improve energy savings. Stemm and Katz [10] have measured the ratio of energy consumed, when a sensor is in idle, receive and send state and the ratio is 1:1.05:1.4. The Digitan wireless LAN module specification specifies the same ratio to be 1:2:2.5 [8]. From the energy consumption ratio

it is evident that the energy spent by a sensor in idle state is almost same as the energy consumed in the receive state. The sensors operate for a prolonged time, and the time spent in idle state also increases accordingly. This results in energy waste due to idle listening [1] (Idle listening is explained in Section 1.2). Ye et al. [1] proposed S-MAC protocol which has introduced low duty-cycle operation, where sensors spend less time in listen state and more in sleep state to achieve energy savings. Ye et al. [2] have also proposed a ultra low duty cycle operating protocol SCP-MAC, which has low power listening mechanism to reduce the energy consumption. This ultra low duty cycle protocol is designed to perform efficiently under variable and burst traffic. Liu et al. [3] proposed CMAC protocol to avoid synchronization overhead and provide low latency in the network. To conserve energy, CMAC protocol puts the nodes to sleep, and wakes them up when there is data, by sending a prolonged RTS. This is termed as aggressive RTS mechanism. The sleep schedules are left unsynchronized in CMAC. BMAC protocol proposed by Polastre et al. [4] also propose unsynchronized sleep schedules, and this protocol sends a long preamble message to wakeup the receiver before the actual data is sent. The preamble message is sent for a period of time, which is greater than the sleep time of the receiver.

Dam and Langendoen [13] proposed TMAC protocol to introduce an adaptive duty-cycle in stationary network to handle the various traffic loads in the network. TMAC protocol achieves these advantages at the cost of decreased throughput. Li et al. [12] proposed a global schedule based algorithm (a single schedule that is been followed throughout the network) for the sensors to avoid the synchronization issues between sensors. PMAC or Pattern MAC is another MAC protocol proposed by Zheng et al. [5] to adjust the duty cycles of a sensor according to the traffic pattern at the neighbor sensors. PMAC achieves throughput under high loads by adaptively changing the duty cycle in the network.

Yao et al. [11] proposed a cluster based adaptive low power MAC protocol for WSN, which is based on clustering topology, and the MAC protocol attempts to adjust the duty cycle according to traffic information collected by cluster head node. The protocol fails to explain the means of achieving the clustering mechanism and exploits cross-layer routing information to reduce the

active period of sensor nodes.

2.2 Open Issues in S-MAC Protocol

The choice of sleep time is a primary challenge for an energy efficient MAC protocol, and the sleep time chosen should not adversely effect the communication in the network. S-MAC is one such energy efficient MAC protocol, which addresses the energy efficiency by introducing novel mechanisms such as:

- Periodic sleep and listen: This mechanism lets the sensor switch off its antenna to enter sleep state and switch it on at a later time to communicate with other nodes. This is repeated throughout the lifetime of the sensor to extend the sensor lifetime.
- Coordinated sleeping: The nodes following periodic sleep and listen coordinate their sleep time with neighboring nodes to enter listen state at the same time. By doing this, the nodes are able to exchange data with minimum delay.
- Overhearing avoidance: This mechanism stops the neighboring nodes interfering from the ongoing communication, by putting them in sleep state until the data transfer is completed. Overhearing avoidance is explained with an example in Section 3.2.9.
- Collision avoidance: S-MAC utilizes virtual and physical carrier sense introduced by IEEE 802.11 to declare whether the channel is idle or not.

In S-MAC protocol, listen time is fixed and the sleep time can be varied by altering the duty cycle value (ratio between listen time and a full listen/sleep interval). The sensor nodes operating on S-MAC freely choose a sleep time upon startup and later on synchronize their sleep time with neighbor nodes through control packet exchange. The energy savings in S-MAC are obtained by periodically switching the nodes to sleep state in which senors switch off their antenna. The sleep time in S-MAC is greater than the listen time. Even though sensor nodes do not have data to send, they are forced to spend time in listen state, which results in idle listening. As described in previous section, idle listening is one of the major causes of energy waste.

The nodes operating S-MAC exchange control packets called SYNC packets with neighboring nodes to synchronize their sleep time with them. The SYNC packet carries the timer value, at which the sensor switches to sleep state. When a sensor receives the SYNC packet from its neighbor, it adjusts its sleep time according to the timer values from the SYNC packet. This SYNC packet exchange takes place between all the nodes in the network. Under high traffic conditions, the SYNC packet exchange can create overhead in the network. Thus, a few aspects in S-MAC which could be improved further include

- Idle listening: Nodes operating on S-MAC have to spend a fixed amount of time in listen state leading to idle listening. This fixed listen time can suppressed to a smaller value and adaptively varied when there is data to send at a node. In CBMAC, the small listen time is used to get synchronized with the cluster head and the listen time is increased when the node has data, otherwise the node switches back to sleep state.

- Synchronization overhead: Nodes operating on S-MAC exchange control packets to get synchronized with the neighboring nodes. The control packets are called SYNC packets and to transmit them a significant amount of energy is consumed. CBMAC attempts to decrease the overhead by synchronizing the nodes with cluster head. The interval between SYNC exchange is increased and this leads to low overhead in the network.

Increased energy savings helps the sensors to increase their lifetime and hence the network remains operational for a long time.

This thesis studies S-MAC and proposes a Cluster-based MAC protocol (CBMAC) to address the open issues in S-MAC. The proposed protocol also introduces counter measures to overcome the specified issues in S-MAC. The CBMAC simulations under various scenarios show significant improvements over S-MAC in terms of energy with little or no change in throughput and latency. CBMAC protocol overcomes the idle listening in S-MAC by suppressing the listen time in sensors and providing more sleep time to them. CBMAC also proposes a centralized approach called clustering mechanism to decrease the control packet overhead in S-MAC.

CHAPTER 3

SENSOR MEDIUM ACCESS CONTROL PROTOCOL

3.1 Introduction

S-MAC stands for Sensor Medium Access Control [1] and is designed to achieve energy efficiency in Wireless Sensor Networks. S-MAC is proposed by Information Sciences Institute at University of Southern California. The energy efficiency in S-MAC is achieved by introducing low duty cycle operation, and the energy savings achieved is larger compared to the Power Save (PS) mode in IEEE 802.11 DCF. In IEEE 802.11 DCF mode, nodes switch between sleep and wake-up states, but their ability to synchronize the sleep time with neighbor nodes is limited to a single-hop. Due to network partitions and clock value drifts, sensor nodes operating on IEEE 802.11cannot perform well in the multi-hop scenario. S-MAC provides support for energy savings and schedule synchronization on multi-hop networks with some latency tradeoff, and also addresses how to efficiently utilize energy.

S-MAC attempts to address the following aspects in Sensor Networks:

- Energy Efficiency: S-MAC identified the sources of energy waste as idle listening, packet collision, control packet overhead and overhearing between nodes. How S-MAC handles these energy waste is explained in the following sections. As viewed from hardware layer, radio communication consumes most of the energy. To encounter this problem, sensor nodes operating on S-MAC switch off the radio for a part of the time, which decreases the overall energy consumption.

- Synchronization: The synchronization between sensor nodes operating on S-MAC is maintained by exchanging control packets called SYNC packets at regular time intervals.

- Scalability: The S-MAC protocol is adaptive to changes in the network. Some nodes join the network over time and some node die and some other move to new locations. The periodic neighbor discovery in S-MAC gracefully accommodates all the changes in the network.

Other properties like latency and per-node fairness are considered less important in S-MAC. The S-MAC features that have been implemented in NS-2 are described in Section 3.4.

3.2 S-MAC Features

The periodic sleep and listen is the most important feature in S-MAC (Section 3.2.1). In order for nodes operating on S-MAC to follow the same sleep-wakeup pattern, synchronization between nodes is needed. S-MAC defines a complete synchronization mechanism (Section 3.2.2), including periodic SYNC packets broadcast, schedule table (Section 3.2.5), neighbor list maintenance (Section 3.2.6) and periodic neighbor discovery (Section 3.2.8) to maximize energy saving and put synchronization in place.

Overhearing is another source of energy waste, especially when the node density is high and the traffic load is heavy in the network. Section 3.2.9 details how S-MAC decreases the overhearing between nodes.

In shared-medium networks, one of the major concerns is to avoid collision between nodes. In this aspect, S-MAC is quite similar with the IEEE 802.11 DCF protocol standard. The features that S-MAC has adopted include physical and virtual carrier sense, RTS/CTS/DATA/ACK sequence for hidden terminal problem. The mechanisms used for collision avoidance are discussed in Section 3.3.

3.2.1 Periodic Listen and Sleep

The periodic sleep accounts for a major share in the energy savings achieved in S-MAC. The basic idea is to let each node follow a periodic sleep and listen pattern, as shown in Figure 3.1. In listen period, the node switches its radio ON to communicate with other nodes and when listen period expires, the node switches to sleep state by turning off the radio. In this way, the time spent on idle listening is reduced to some extent, especially under low traffic.

During listen period, the node starts sending or receiving packets, if it has any. S-MAC provides a controllable parameter duty cycle, whose value is the ratio of the listen time to the frame time

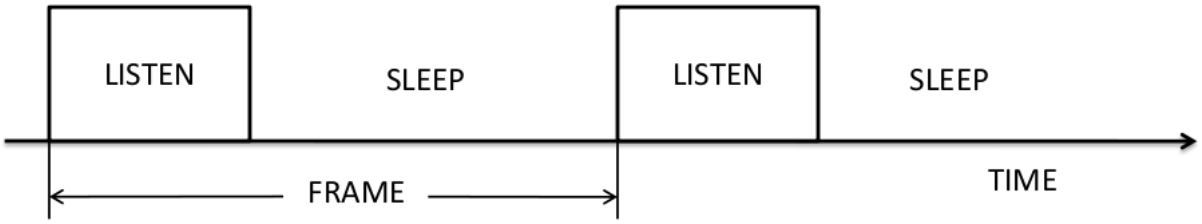

Figure 3.1 – Periodic sleep and listen

(sum of listen and sleep times). The listen period is normally fixed according to physical and MAC layer parameters. The user can adjust the duty cycle value from 1% to 100% to control the length of sleep period. Normally, the frame time is dependent on the duty cycle. As the duty cycle is varied, the frame time varies and hence the sleep time, but the listen time remains constant at all times. Also the nodes adaptively vary their duty cycle depending on overhearing avoidance in S-MAC. Overhearing avoidance is described in Section 3.2.9.

The listen period is further divided into two parts. The first one called SYNC period is designed for SYNC packets, which are broadcast packets and are used to establish synchronization between sensor nodes operating S-MAC . We will discuss all synchronization issues in Section 3.2.2. The second one called DATA period is designed for transmitting DATA packets. The following figure shows the SYNC and DATA times in S-MAC.

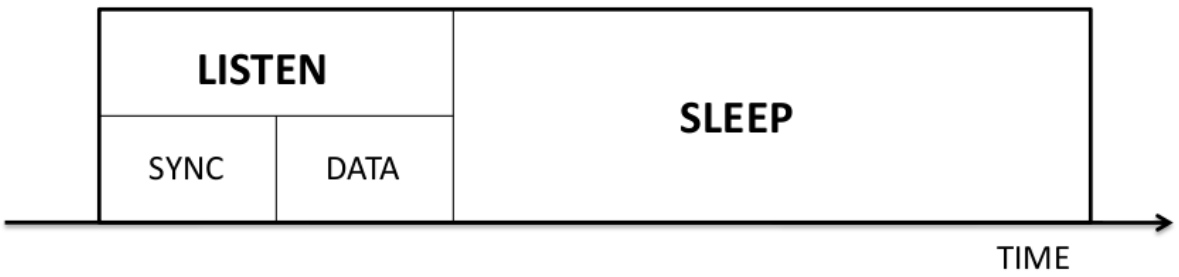

Figure 3.2 – Listen time in S-MAC

The Sync time and Data time are calculated using the following expressions:

Sync time = DIFS + Slot time $\times$ SYNC CW_{size} + Duration of SYNC pkt (secs) + Guard time

Data time = DIFS + Slot time $\times$ DATA CW_{size} + Duration of Control pkt (secs) + Guard time

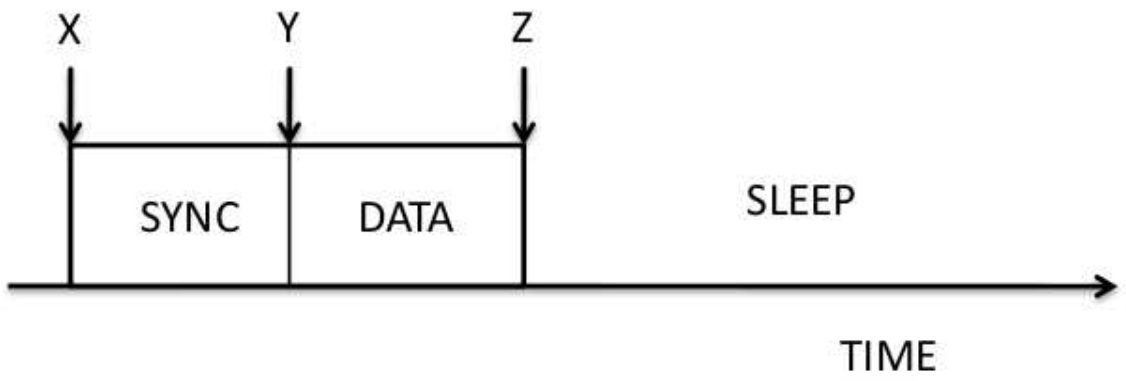

Figure 3.3 – Expiration points in between states

The sleep and listen times followed by a sensor node operating S-MAC are collectively referred to as its *schedule*. Each node operating on S-MAC maintains atleast one schedule to follow. Each schedule is controlled by a schedule timer, which reschedules the next period when current period expires. Figure 3.3 explains how the states are switched in a schedule. Each frame has three expiration points, which are called checking points. At any of these checking points, S-MAC will decide what to do in the next period. For example, at checking point Y, S-MAC checks if it has a data packet in the buffer to send. If yes, it will start carrier sensing. Otherwise, S-MAC will never try to send any data packet in this frame. Even if S-MAC receives a new data packet from its upper layer right in the middle of the DATA period, it will buffer it and wait until the next checking point Y comes, rather than start contending for the medium right away.

When a node operating under S-MAC is following a sleep and listen schedule, it doesn't mean that the node must keep listening in the listen period or must go to sleep when the scheduled sleep period arrives. The actions of S-MAC at a particular time instance depend not only on the current scheduled period, but also on some other factors like current MAC state, radio state, channel state, neighbors state, etc.

For example, at checking point Z, S-MAC will go to sleep by turning off its radio, only when all the following listed conditions are satisfied. If the node:

- Has more than one schedule in its schedule table (Section 3.2.5).
- Is idle and it is neither a sender nor a receiver of an ongoing data transmission. The node cannot go to sleep, if it has sent a CTS packet to another node in last DATA period and is now waiting for the data packet.
- Is not in a neighbor discovery period (Section 3.2.8).

In this section, the basic idea of sleep-listen mechanism in S-MAC is primarily discussed. Other features of S-MAC are explained in the following sections.

3.2.2 Synchronization

The nodes operating under S-MAC follow periodic sleep and listen mechanism, because of which there is a possibility that when one node is in sleep state, neighboring node may be in listen state. To handle this issue, nodes operating under S-MAC follow a synchronized periodic sleep and listen, so that all the neighboring nodes sleep at same time and wake up at the same time. The synchronization between nodes is achieved by SYNC packet exchange. This section outlines the synchronization issues and how S-MAC handles synchronization.

As mentioned earlier, each node operating under S-MAC needs to exchange its schedule by periodically broadcasting a SYNC packet to its neighbors. The period of sending SYNC packet is called synchronization period. The default value of synchronization period in NS-2 [14] is 10 frames (one frame = one sleep and listen cycle). Sending or receiving SYNC packets takes place in SYNC period. The definition of SYNC packet frame in NS-2 is shown in Table 3.1.

Table 3.1 – S-MAC SYNC Packet Format

Field	Description
type	Flag indicating this is a SYNC packet
length	Fixed size with 9 bytes
srcAddr	ID of the sender
syncNode	ID of the sender's synchronization node
sleepTime	Sender's next sleep time from now
crc	Cyclic Redundancy Check

From the table, it can be observed that *sleepTime* is the most important information contained in the SYNC packet. The *sleepTime* specifies the time when the sender of that SYNC packet is going to sleep state. To avoid synchronization errors due to clock drift on each node, the sleepTime uses a relative value rather than an absolute time. Also, when a SYNC packet is received, the value of *sleepTime*(s) are updated by replacing the *sleepTime* of the node with that of the received *sleepTime* in SYNC packet.

3.2.3 Schedule

Every node operating under S-MAC follows a sleep and listen pattern, which is called a schedule. A node operating S-MAC has two types of schedules, primary and secondary schedule. The schedule that the node chooses for the first time is called primary schedule and the schedule that the node receives from its neighbor after choosing primary schedule is called secondary schedule. Section 3.2.4 describes how a schedule is selected.

3.2.4 Schedule Selection

When a new node joins the network, it first listens to the channel for a fixed amount of time. This period is called initial listening period. Two cases are defined depending on whether the node receives the SYNC packet or not in the initial listening time.

- If a SYNC packet is NOT received during the initial listening, the node chooses a schedule by itself and sets a schedule timer for it by the end of the initial listening period. The schedule is added to the first entry of its schedule table and it becomes the primary schedule. To announce this new schedule, the flag *txSync* of this schedule is set, which indicates that the node will try to broadcast a SYNC packet in the next SYNC period. The field definition of schedule table is explained in section 3.2.5.

- If a SYNC packet is received before the initial listening ends, then the node's first SYNC packet is said to be received. The node follows the schedule that comes with the SYNC

packet received, instead of choosing a schedule by itself. The value of sleepTime in the SYNC packet determines the starting point of the first cycle. Meanwhile the schedule is added to the first entry of the schedule table, and the sender of the SYNC packet is added to the neighbor list (Section 3.2.6). The node also prepares to announce this schedule by setting the flag *txSnyc* in this schedule.

3.2.5 Schedule Table

Each node operating on S-MAC maintains a schedule table, which stores its own schedule and the schedules of its neighbors. Both the primary and secondary schedules are stored in the schedule table. Every node may not have secondary schedule, but every node has a primary schedule. For example, if all the nodes in the network follow the same schedule, then there is only one schedule in the network and that schedule becomes the primary schedule for every node. The maximum number of schedules a node can follow is modified by a user-adjustable parameter called S-MAC_MAX_NUM_SCHEDULES. The default value of this S-MAC parameter is 4, implying a node can at most follow 4 different types of schedules. Following table shows how each field is defined in a schedule table.

Table 3.2 – S-MAC Schedule Table Format

Field	Description
txSync	If set, need to send a SYNC
txData	If set, need to send DATA
numPeriods	Counter for sending SYNC periodically
numNodes	Number of nodes following this schedule
syncNode	ID of node who initialized the schedule
chkSched	Flag indicating need to check numNodes

Why is a Schedule table necessary? The authors in [1] intuitively hope that only one schedule exists in the entire network, because sleep durations and energy savings are maximized when all nodes are on the same schedule. But in practice this mechanism takes effect only locally. The existence of multiple schedules in a network is quite common, especially in a large multi-hop network where nodes do not start initializing at the same time and join in a random way. The

nodes that share the same schedule can be collectively called a virtual cluster and nodes with neighbors in two or more clusters as border nodes. Obviously, if the border node wants to talk to its neighbors at their scheduled listen time, it has to know when its neighbors wake up and go to sleep. Hence multiple schedules for a single node are common and in order to accommodate these schedules, a schedule table is necessary.

A schedule is actually a timer. Both primary schedule and other secondary schedules have their own timers. These timers run independently and their working is quite similar. A simple example is considered to illustrate how a border node works under multiple schedules. Let us assume that node B has two neighbor node A and C. As shown in Figure 3.4, B has two schedules in its schedule table. One is its own schedule (primary schedule), which is the same as A's, and the other is heard from C. B will allocate a schedule timer for either of the two schedules. The two schedule timers run independently.

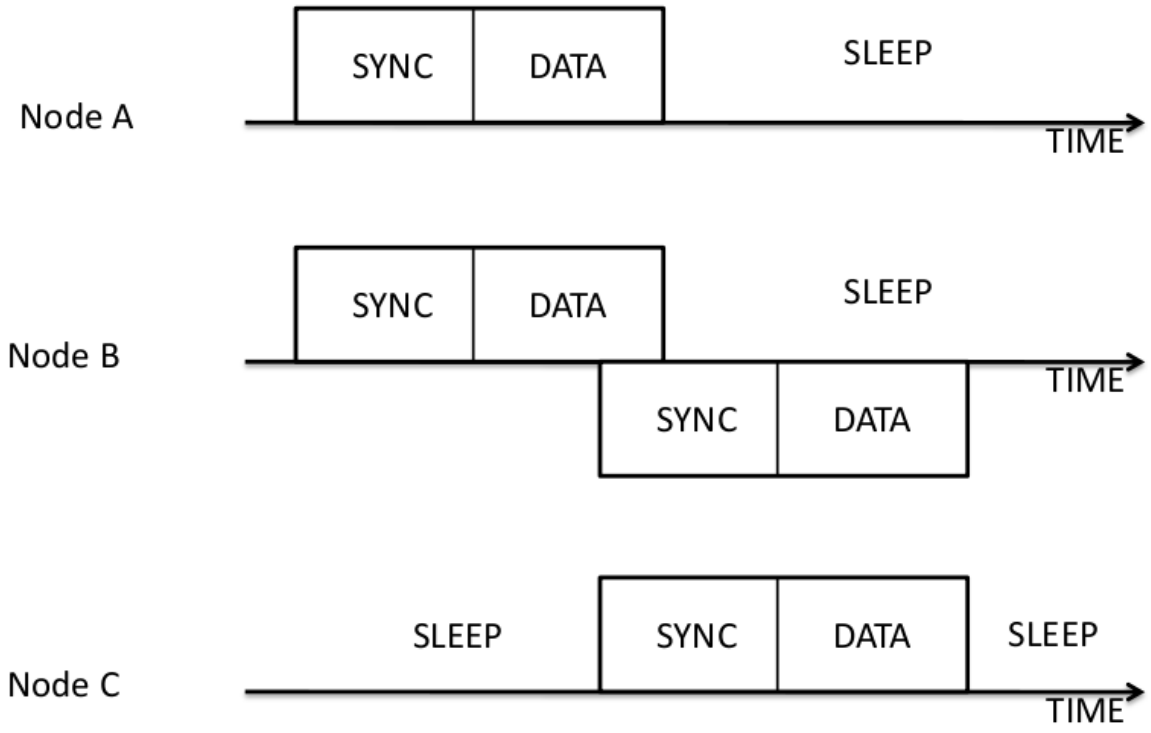

Figure 3.4 – Sleep-and-listen format for border node

B has to broadcast its SYNC packet twice to ensure that both A and C can receive that SYNC

packet in their respective SYNC periods. Also in a SYNC packet, a border node conveys to its neighbors, the time from now to its next sleep time according to its primary schedule, but not the secondary schedule. The border node follows the same procedure to send DATA broadcasts if any. When B has a data packet to broadcast, it has to broadcast it twice. One takes place on DATA period of A and the other on DATA period of C.

If B has to unicast data packets, it acts as follows. Suppose B receives a data packet destined to C from the upper layer, it first searches in its schedule table to find out the schedule that C is following, and it sets the *txData* flag on that schedule to 1. When the next DATA period of C arrives, the corresponding schedule timer on B will inform B that its data buffer has a data packet destined to C and now it is time to send it out.

From the above discussion, it is evident that a border node has to follow several schedules to get synchronized with its neighbors that are on different schedules. It is evident that a border node will consume more energy than non-border nodes.

3.2.6 Neighbor List

Another important component in S-MAC is neighbor list. Each node operating on S-MAC sets up a neighbor list table to keep track of all its known neighbor's information. The number of entries that can be made to the list is also limited by a user-adjustable S-MAC parameter, which defines the maximum number of neighbors for each node. Like schedule table, neighbor list is also established by exchanging SYNC packets between neighboring nodes. Table 3.3 describes the fields of neighbor list.

Table 3.3 – Field Definition of Neighbor List

Field	Description
nodeId	ID of the Node
schedId	Schedule that this node is following in the Schedule table
active	Flag is set if the node was active recently
state	Flag indicating whether the node has changed its schedule

When S-MAC processes a unicast data request received from the upper layer, it will first check its neighbor list to see if the destination node is on the list. If it finds the node the flag *txData* of the destination node's schedule is set to 1, otherwise it will reject the request. When the next DATA period on this schedule arrives, the node will try to send out this packet.

3.2.7 Updating and Maintaining Schedules

The nodes operating under S-MAC periodically exchange SYNC packets to maintain synchronization. In this section, how nodes operating under S-MAC handle the received SYNC packets after choosing primary schedule is described.

The algorithm for handling received SYNC packets is described below. The class implementing this algorithm is S-MAC::handleSYNC(Packet *p) in S-MAC.cc file of NS-2.

When a node receives a SYNC packet after choosing its first schedule, it takes action depending on the following algorithm.

- If the SYNC packet received is node's first SYNC packet.

  This case happens when the node sets the schedule by itself and after the initial listening period is over. The packet received will be the first SYNC from one of its neighboring nodes. The node would discard its current schedule and follow the schedule as its primary schedule and adds the node to its neighbor list. The scheduler timer is reassigned according to the *sleepTime* in the SYNC packet.

- If the SYNC packet is not the first one after node chooses its schedule.

  There are five possibilities to consider in this condition. For simplification, let S represent the sender of the SYNC packet and R is the node receiving the SYNC packet. The algorithm is presented in Table 3.4.

Table 3.4 – Algorithm for New SYNC Packet

Condition	Action Taken
S is a known neighbor and is in the neighbor list of R. S has not changed its schedule since the last time it has sent a SYNC to R.	Update the sleepTime of S in the schedule table. Once the latest sleep time is updated, any drift in clock values is eliminated.
S is a known neighbor and is in the neighbor list of R. S has changed its schedule since the last time it has sent a SYNC to R. By checking the schedule of S in the neighbor list, R comes to know about the schedule change of S.	Process the schedule that S has switched from. Decrement the number of nodes on that schedule by 1. If the new value is 0, delete the schedule from table. If the *txData* bit is set, defer the previous process until DATA is transmitted. Process the schedule that S has switched to. Add the new schedule of S to the schedule table, if the schedule table is not full. If table is full S is deleted from the Neighbor list. If the deleted schedule is R's primary schedule, then the next schedule to it will be R's new primary schedule. If *txData* bit is set defer this process until DATA sent.
S is a known neighbor and is in the neighbor list of R. S has changed its schedule since the last time it has sent a SYNC to R and the new schedule is in my schedule table.	Process the schedule that S has switched from. Decrement the number of nodes on that schedule by 1. If the new value is 0, delete the schedule from table. If the *txData* bit is set, defer the previous process until DATA is transmitted. Process the schedule that S has switched to by updating the timer with the sleepTime in the SYNC packet and increment the numNodes by 1.
S is a new node and its schedule is also new.	If schedule table or neighbor list are not full, add them.
S is new node but its schedule is in the schedule table.	Update the schedule table by changing the sleepTime and add S to the neighbor list if it is not full. The numNodes field in schedule table corresponding to the schedule is incremented by 1.

3.2.8 Neighbor Discovery

In S-MAC, neighboring nodes discover each other through SYNC packet exchange. However, sometimes two neighboring nodes may miss each other forever, for instance, when two nodes follow different schedules, whose SYNC periods do not overlap. S-MAC introduces periodic neighbor discovery to address this issue.

The idea of neighbor discovery in S-MAC is to make each node periodically execute neighbor discovery for whole synchronization time. During neighbor discovery time, nodes operating under S-MAC never enter sleep state, even when the listen timer expires, so the node can listen for more

time and the chances to hear new SYNC packets from new neighbors is increased. For a border node, neighbor discovery is only executed on its primary schedule. The reason is that on secondary schedule, the node will not try to go to sleep when scheduled sleep period arrives.

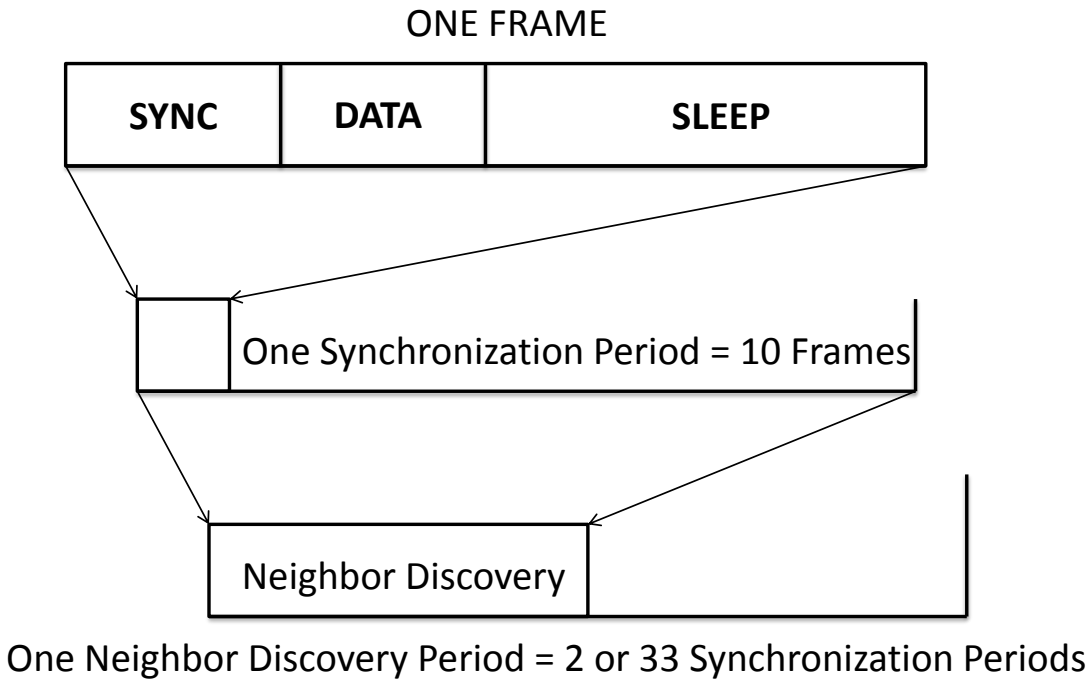

Figure 3.5 – Hierarchy of periods in S-MAC

Figure 3.5 shows how the neighbor discovery period is defined and its relationship with other periods defined in S-MAC. The neighbor discovery period may vary, depending on the current number of known neighbors of a node. In NS-2, the period of executing neighbor discovery is 2 synchronization periods if the node has no neighbor. Otherwise, the period is much longer and reaches 33 synchronization periods. When a node has neighbors, it need more time to discover potential neighbors or get synchronized with the existing neighbors. Hence the neighbor discovery period is set to a higher value (33).

3.2.9 Overhearing Avoidance

For contention-based protocols like IEEE-802.11, overhearing is one of the major sources of energy waste. A node is said to overhear, when it receives packets that are destined to other nodes. To achieve better performance in a shared-medium network, carrier sense, especially virtual carrier sense should be performed more efficiently. The best way to achieve it is to let each node keep listening to all its neighbors transmissions. But this method will lead to large amounts of energy consumptions, especially when node density is high and the traffic load is heavy.

The primary goal of S-MAC is to conserve energy. To avoid overhearing, S-MAC puts the interfering nodes in sleep after they receive an RTS or a CTS packet that is not destined for them. In this way, nodes that interfere with other's transmission will not hear DATA packets. As DATA packets take more amount of time than the control packets, the interfering nodes can be put in to sleep for more time. Consider the following example to illustrate overhearing avoidance.

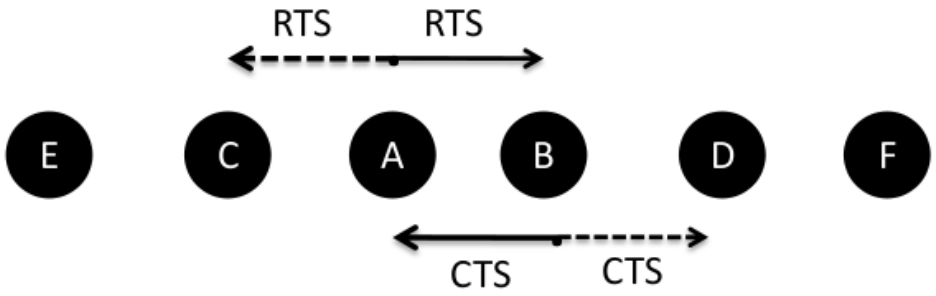

Figure 3.6 – C and D overhear transmission between A and B

Figure 3.6 shows a five-hop linear network. Each node can only hear its immediate neighbors. Assume that all the nodes share the same schedule. Suppose A is communicating with B. D is supposed to go to sleep, because D's transmission interferes with B's reception. C is two hops away from B, so C's transmission will not interfere with B's reception. But if C talks to E while A is sending data to B, C will not receive any packet from E because a collision occurs at C. C's transmission is a waste of energy and it should go to sleep. Both E and F are at least two hops

away from the nodes that are transmitting, and they will never interfere. Therefore, E and F need not go to sleep. To summarize, the immediate neighbors of the sender and the receiver should go to sleep.

Consider the case of C and D. C, who is A's immediate neighbor can hear the RTS packet that A sends to B. However, C does not go to sleep after receiving the RTS, because at this moment the communication has not been really set up and may be cancelled for some reason, e.g. collision of RTS packets. Therefore, it has to keep listening for a CTS timeout period to see if it can hear the following CTS packet. If C receives the CTS packet, it will go to sleep right away. However, in this example, C cannot hear the CTS that B sends back to A. Therefore, after the CTS timeout period, C will keep listening for another CTS timeout period and try to hear the DATA packet. As soon as C hears the DATA packet, it knows from the duration field in the DATA packet, how long the current transmission will last and goes to sleep until then. If no DATA packet is heard in this period, C will go to sleep only when C's primary schedule timer enters sleep period. For D, who is B's immediate neighbor, it can only hear the CTS packets sent by B. D will go to sleep right after it gets the CTS and updates its NAV with the duration value in the CTS, and tries to access the channel, if it has DATA to send, after the NAV value is zero.

3.3 Carrier Sense

Like IEEE 802.11 DCF, S-MAC performs both physical and virtual sensing. The medium is determined free only when both indicate that it is free.

Physical carrier sense is performed at Physical layer by checking the current radio state. Every time when the radio starts receiving or transmitting, the PHY layer will inform the MAC. It also happens when the receiving or transmitting is over. The medium will be determined as busy when radio is in receiving or transmitting state.

Virtual carrier sense is performed at the MAC layer. We know that in IEEE 802.11, each station maintains a Network Allocation Vector (NAV), which is actually a counter. It will count down to zero at a uniform rate after it is assigned with a certain value. NAV mechanism requests that each

packet sent by MAC contains a duration field, which indicates how long the current transmission will last. When a node receives a packet destined to another node, the duration in the packet will let the node know how long it has to remain silent. The NAV will be updated with the duration value, only if the duration value is larger than the current NAV value. Virtual carrier sense indicates that the medium is idle, when the NAV value is zero. When non-zero, the medium is busy.

S-MAC obtains a transmission chance through contending for the medium in a contention window. Actually there are two contention windows defined in one frame. One is for sending SYNC packets in the SYNC period and the other is for sending DATA packets in the DATA period. Both contention windows have fixed size (fixed number of slots) e.g. 31 slots for SYNC and 63 for DATA. Consider the following simple example to see how carrier sense is performed before sending a SYNC packet in the SYNC period. The following steps are also applicable to send a DATA packet in the DATA period.

When the schedule timer expires at the end of the sleep period and transiting into a new SYNC period, a node checks the conditions for sending a SYNC packet. If the *syncFlag* on this schedule is set and the NAV value is zero (virtual carrier sense indicates free medium), the node will be ready to broadcast a SYNC packet. If the radio is in sleep state, the node will wake itself. Then the node starts physical carrier sense. The duration for carrier sense is chosen uniformly within the contention window and consists of a DIFS. One of two following possibilities may happen during this period.

- If nothing is heard throughout the whole carrier sense period, the node will assume that the medium is free and start transmitting the packet.
- Once the medium is sensed busy during the carrier sense period, the node will stop sensing and defer sending the packet. When the same period in the next frame arrives, the node will retry sending. While retrying, it will follow the same procedure again.

S-MAC always chooses the number of slots for carrier sense within the fixed contention window size, uniformly and independently. It does not perform binary exponential back off (BEB)

introduced in IEEE 802.11.

Collision Avoidance: The RTS/CTS mechanism defined in the DCF [6] can efficiently reduce the collisions and solve the hidden terminal problem [16]. While sending the DATA packet, sender is required to exchange RTS and CTS packets with the receiver to reserve the channel. After successful RTS/CTS bond, DATA packet is sent and ACK is expected upon packet success. Broadcast packets like SYNC and other control packets are sent without using RTS/CTS.

When two nodes are planning on data exchange, the reservation information is recorded in the duration field of RTS/CTS/DATA/ACK packets. RTS/CTS packets will reserve the medium for whole transmission process. All immediate neighbors of both the sender and the receiver will learn the coming transmission from the RTS or CTS packets and will keep silent during the reserved period. Although the combination of carrier sense and RTS/CTS mechanisms can largely reduce the probability and durations of collisions, collisions cannot be completely avoided. If two neighboring nodes finish carrier sense and start sending at almost the same time, collision still occurs.

3.4 S-MAC Features Available in NS-2

S-MAC is primarily developed considering the Mote platform. Motes run on TinyOS, an event driven operating system developed for Wireless Sensors. But as a hardware implementation, which needs real hardware, it is not suitable to study S-MAC comprehensibly. S-MAC has been also implemented in NS-2, the network simulator.

NS-2 (Network Simulator version 2) [14] is an object-oriented, discrete-event-driven network simulator targeted at networking research, which has been extensively used by the networking research community. NS-2 is a powerful network simulator. It provides substantial support for simulation of TCP, routing, multicast protocols over wired and wireless (local and satellite) networks. Users can define arbitrary network topologies composed of nodes, routers, links and shared medium. A rich set of protocol objects can then be attached to nodes, usually as agents. The sim-

ulator suite also includes a graphical visualizer called network animator (NAM) to assist the users in providing more insights about the simulation by visualising packet trace data.

NS-2 is written in C++, with an Otcl interpreter as a frontend. Otcl is an object-oriented variant of the well-known scripting language tcl. C++ is utilized for per-packet processing and forms the core of the simulator. Otcl is used for simulation scenario generation, periodic or triggered action generation and for manipulating existing C++ objects.

NS-2 is open-source software and free to use for all users. Its open source nature allows any user to modify parameters at different layers, create their own applications, and develop new protocols. The freeware nature of NS-2 makes it very attractive to students and network researchers and is ideal for research purposes. Most current version of NS-2 is NS-2.34 and more information about NS-2 download/update can be found on the NS-2 homepage [14].

In this thesis, S-MAC is studied under NS-2 on a Ubuntu Linux system. After installing NS-2, the source files for S-MAC (C++ files) can be found under mac folder of NS-2.34. The complete path is as shown below.

```
Ns-allinone-2.34/ns-2.34/mac/S-MAC.cc(OR)S-MAC.h
```

Most of the features proposed in the S-MAC [1] have been implemented in this version. Some portion of the code is called JOUNAL_PAPER code, because a macro named JOURNAL_PAPER controls compiling this portion of code. By default, the definition statements for this macro are disabled in the S-MAC header file and JOURNAL_PAPER code will not be compiled when NS-2.34 is run for the very first time. The features that JOURNAL_PAPER include are listed in Table 3.5. In this thesis, S-MAC is evaluated without including the JOURNAL_PAPER macro, as the macro has minor issues while compiling and is not completely updated.

3.5 S-MAC Parameters

All parameters for S-MAC can be found and modified in the header file S-MAC.h, including user adjustable S-MAC parameters, internal S-MAC parameters, and physical layer parameters. Some other parameters can be assigned in the TCL script at the start of the simulation run.

Table 3.5 – Features of S-MAC

Basic features	Features in JOURNAL_PAPER code (Not considered for this thesis)
Listen and sleep periodically according to schedules	Adaptive listening
Both physical and virtual carrier sense	Message passing
Overhearing avoidance	
RTS/CTS mechanism for unicast data packets	Neighbor list updating
Synchronization algorithm (including schedule choosing, schedule updating and maintaining)	Global schedule algorithm
Neighbor discovery	

Table 3.6 – Parameters in S-MAC

Name	Comment
syncFlag_	If set to 1, S-MAC runs with periodic sleep. If it is set to 0, S-MAC runs without periodic sleep.
dutyCycle_	The value of duty cycle in percent. It controls the length of sleep. If not set, NS-2 uses the default value 10% from the ns-default.tcl file. This parameter is active only when syncFlag_ is set to 1.
selfConfigFlag_	If it is set to 1, all nodes operating on S-MAC follow the schedule initialization algorithm. If it is set to 0, the schedule start time (first listen period start time) for each node is user-configurable.

For example, the sleep time and cycle time (frame time) depend on the value of duty cycle that have been set in the TCL script. The parameters that can be altered in the TCL script are listed in Table 3.6. We set *syncFlag_* to 1, *dutyCycle_* to 20% and *selfConfigFlag_* to 1 in TCL Script.

The features of S-MAC that are implemented for this thesis study are:

- Periodic listen and sleep.
- Physical and virtual carrier sense.
- Overhearing avoidance.
- RTS/CTS mechanism for unicast DATA packets.

- Synchronization algorithm (including schedule choosing, schedule updating and maintaining).
- Neighbor Discovery.

CHAPTER 4

CLUSTER-BASED ENERGY-EFFICIENT MAC PROTOCOL

4.1 Introduction

The primary objective of an energy efficient MAC protocol is to reduce energy wastage in the network. Previous studies about wireless networks indicate that idle listening consumes 50-100% of the energy spent by sensors in listen mode. One way to overcome idle listening is to switch ON nodes when there is some data generated/received and keep them off for the rest of the time. But due to the randomness in the data arrival, the nodes cannot adaptively switch between sleep and listen states. So the listen time should be minimized to an extent that maximum energy can be saved and the wake up time should change adaptively depending upon the application that is running. Delay is another prominent feature while dealing with energy. Delay can occur due to various reasons, such as lost transmission opportunity caused by sleep, node fail to transmit the packets in a cycle due to collision and back off procedure. These should be minimized to decrease delay in the network.

In order to increase the energy savings, a new protocol called Cluster-Based Energy-Efficient MAC protocol (CBMAC) is proposed. The CBMAC protocol is introduced to overcome the energy wastage in S-MAC protocol. The S-MAC protocol has introduced adaptive sleep mechanism to overcome the idle listening in sensors and achieve energy savings. Although energy is saved because of this fixed sleep-wake up pattern, some amount of energy is still wasted, when the nodes spend time in listen state, without any data to send or receive. The proposed cluster-based protocol saves considerable amount of energy, by suppressing the listen time in nodes. CBMAC protocol adapts features in S-MAC and proposes additional features to improve energy savings.

The S-MAC features that are improved in CBMAC are

- Periodic sleep and listen: The listen time is further decreased in CBMAC to reduce the time spent in listen state and hence reduce the energy consumption.

- Synchronization algorithm: The synchronization in CBMAC is different from S-MAC. The nodes operating CBMAC do not exchange SYNC packets with each other, instead they exchange SYNC packets only with cluster head. Also, the periodic time interval between two SYNC exchanges in CBMAC is increased to a larger value to decrease the overhead in the network.

- S-MAC features like physical and virtual carrier sense, overhearing avoidance and RTS/CTS mechanism remain same in CBMAC.

4.2 Protocol Design

The CBMAC protocol follows a centralized mechanism, where a cluster head controls the remaining nodes in the cluster. Whereas in S-MAC, nodes communicate in ad-hoc[2] fashion and exchange data. It is easy to synchronize nodes that follow centralized architecture as the control packets required for synchronization are sent by the base station or central node (cluster head in this case). Whereas ad-hoc nodes need to exchange control packets between themselves, where a lot of processing and overhead is involved.

Figure 4.1 shows a cluster with cluster head, border node and the cluster nodes. Figure 4.2 shows the inter-cluster scenario, where border node connects two clusters. All the nodes that are one hop away from each other form a cluster. Every cluster has a separate cluster head elected according to election process described in Section 4.5.

The energy savings in CBMAC are due to the increased sleep time at the cluster nodes of a cluster. Initially the listen time is same for all the nodes and cluster head in the network. But when a node needs to send data, it sends it to cluster head and switches to sleep state, and the cluster head increases its listen time to forward the data to the destination (to border node in case of inter-cluster scenario). Therefore, the overall listen time is decreased at cluster nodes and increased at cluster head, so as to enable cluster head to handle the data packets. The nodes save considerable amount

[2]Operating in ad-hoc mode allows all wireless devices within range of each other to discover and communicate in peer-to-peer fashion without involving central access points.

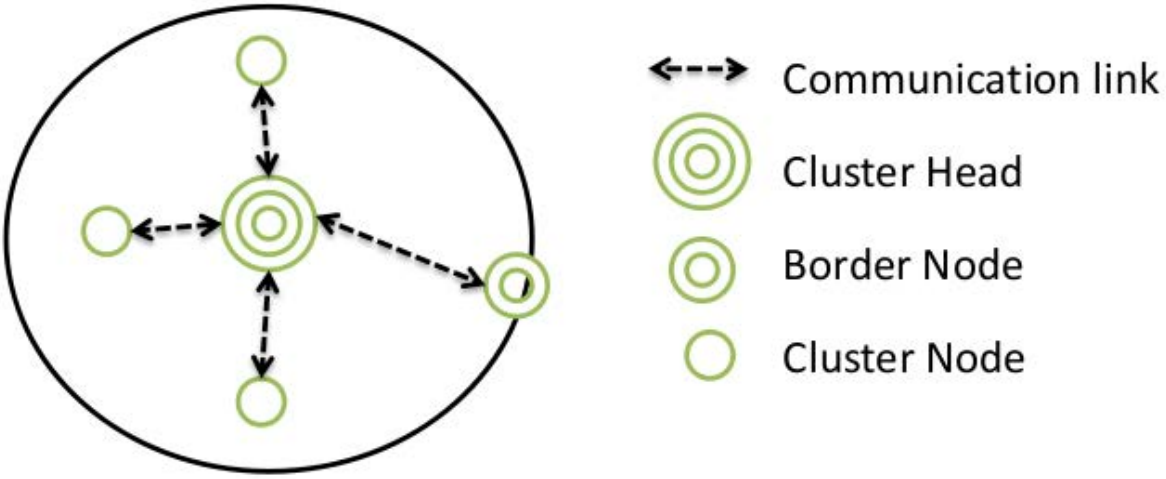

Figure 4.1 – Sample cluster with cluster head, border, and cluster nodes

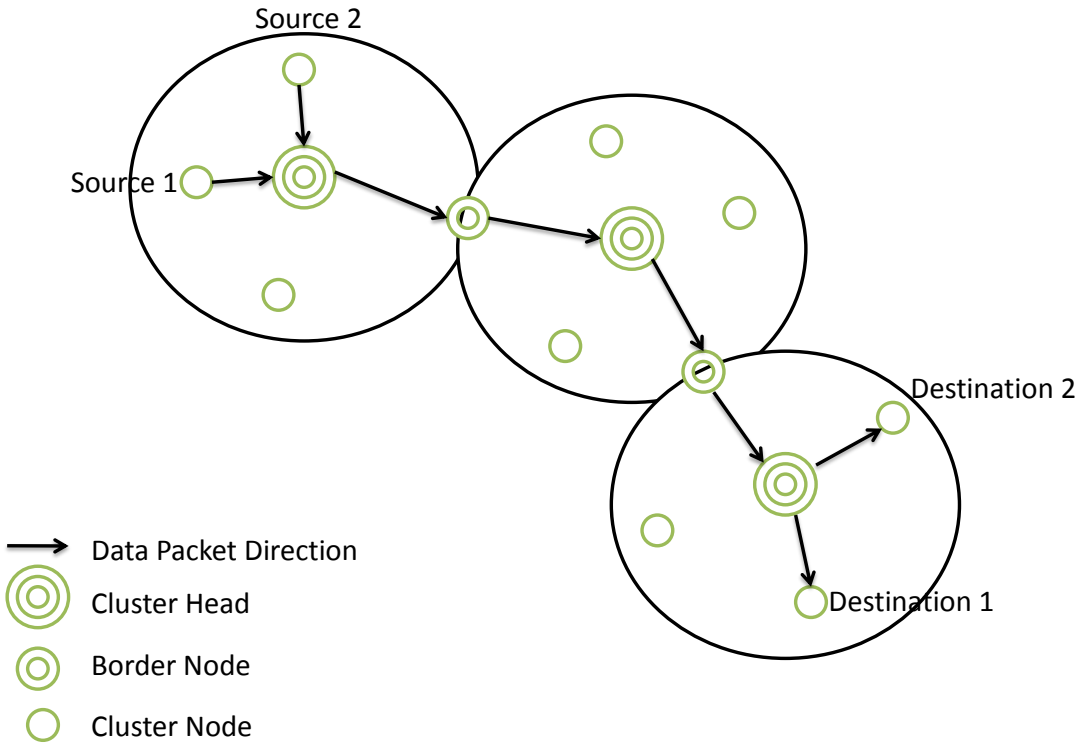

Figure 4.2 – Sample multi-hop scenario in CBMAC

of energy as they spend more time in sleep state, besides accepting latency as a tradeoff. Overall energy savings in CBMAC are due to aggregation of the energy savings at all the cluster nodes in the network. The cluster head synchronizes its members to enter sleep state at the same time. The schedule synchronization procedure in CBMAC is quite different from S-MAC. The schedule synchronization in CBMAC is achieved by exchanging SYNC packets between nodes for the first time and later on after cluster head is elected, only cluster head sends SYNC packets, while rest of the nodes get synchronized to the cluster head. Figure 4.3 compares the sleep and listen times in S-MAC and CBMAC.

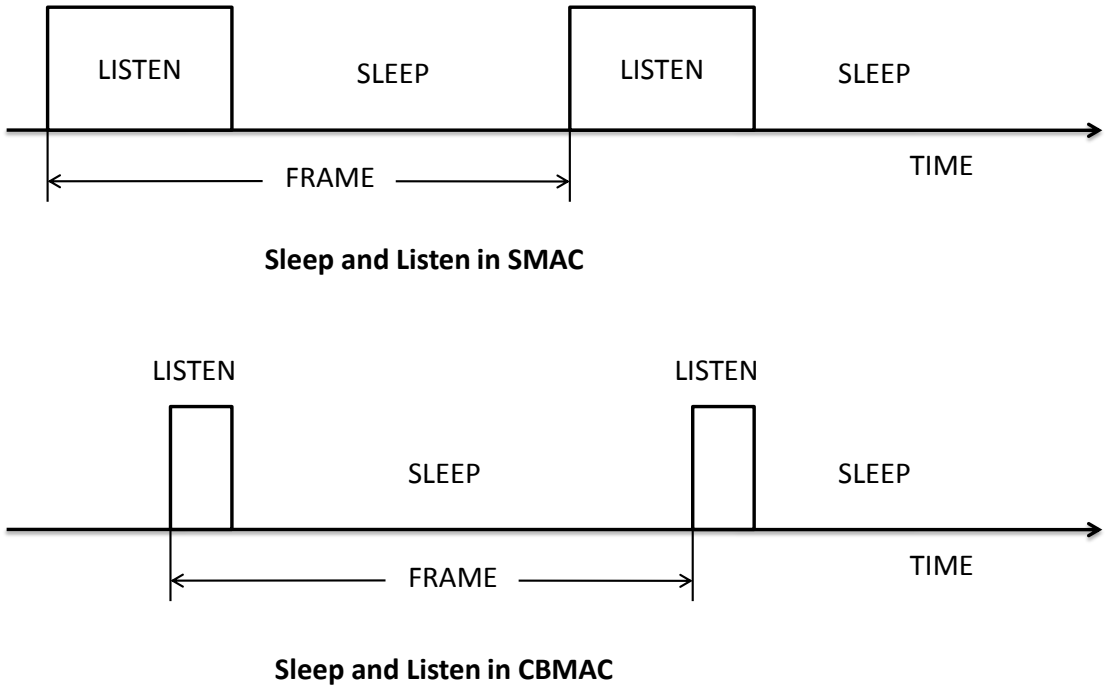

Figure 4.3 – Sleep and listen times in S-MAC and CBMAC

The frame time as shown in Figure 4.3 is same for nodes operating under S-MAC and CB-MAC, but in nodes operating under CBMAC, the listen time is decreased and the amount of time decreased is added to sleep time. The listen time in CBMAC is divided into SYNC and DATA

times, similar to S-MAC as shown in Figure 3.2. The SYNC packets are exchanged in SYNC period and when a node has data to send, it uses the DATA time to send data. When a node has data to send, and the DATA time period is not sufficient, the node extends the listen time required to send all the data it has.

When a cluster node receives SYNC packet from more than one cluster head, it follows the schedule in all those SYNC packets and that cluster node is said to be a border node. It follows the schedule of all the cluster heads, from which it receives SYNC packets and bridges two cluster heads to exchange DATA packets. Border nodes are the backbone for inter-cluster communication. Hence border node life time is crucial in cluster-based communication scenarios. When a border node runs out of energy, the communication between two clusters it is connecting is lost. A new border node has to be assigned in such a case. A border node will have same sleep and listen times as the cluster node and it extends its listen time to forward the data to next cluster head if needed. The unicast packets are sent by following RTS/CTS/DATA/ACK, where as broadcast packets are sent directly.

The cluster node which has maximum number of neighbors becomes the cluster head. The cluster head election process is explained in Section 4.5. The cluster head position is temporary and is switched between different nodes for uniform energy utilization. As the cluster head handles majority of traffic in the network, there is a delay accumulated in high traffic scenarios.

4.3 Choice of Duty Cycle for S-MAC

To study S-MAC, the duty cycle of 20% is chosen and the choice of 20% is explained as follows. Duty cycle is defined as the ratio of listen time to the frame time. Duty cycle determines the length of the cycle time, cycle time in turn decides the length of sleep time. The interdependencies can be understood from equations (4.2), (4.3), and (4.4).

With S-MAC as the MAC protocol, a 10 hop linear network is considered for simulation, where node 0 is sending traffic to node 10. The traffic rate is set as 1 packet/10 sec and the packet size is 80 bytes. The simulation time is 1000 sec and the traffic is sent for 1000 sec. A graph is plotted

Figure 4.4 – Ten-hop linear network

with different duty cycles on x-axis and energy consumed in Joules, latency in seconds on Y-axis. The plot is shown in Figure 4.5.

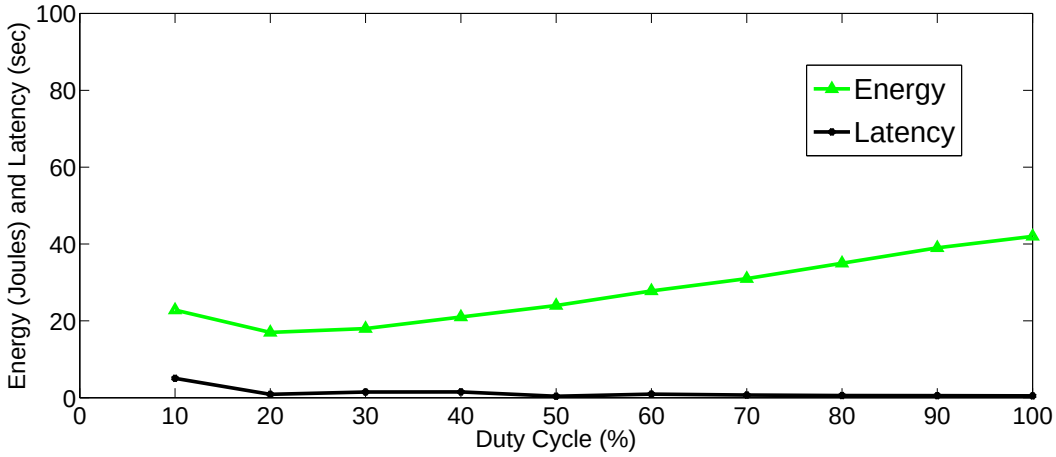

Figure 4.5 – Comparison of energy and latency for different duty cycles in S-MAC

The above graph is simulated by varying the duty cycle from 10% to 100% under average load (10 packets/sec). The energy consumed is read from the energy model available in NS-2. The energy model in NS-2 provides user to define the initial energy of a node. Throughout the thesis, initial energy values assigned to all nodes is 1000 Joules. At the end of simulation, the energy remaining in each node is obtained from the trace[3] file. The energy consumed can be calculated from the difference between initial energy and energy remaining. The simulation time is 10000 sec.

A close look at the plot indicates that the energy value and latency value pair for 20% duty cycle is minimum when compared with other pairs. This explains the reason for choosing 20% duty cycle to study S-MAC protocol.

[3]All the desired parameters can be set to ON to view them on trace file at the end of the simulation in NS-2

4.4 Features in CBMAC

4.4.1 Synchronization

The nodes operating under CBMAC upon startup, listen to the channel for a initial listening time period. If a SYNC packet is received for the very first time, the node checks the *CH_flag* value. If the value is 1, it follows the schedule from that SYNC packet, otherwise the $numNeighb$ field is updated and the packet is dropped. The $numNeighb$ field is only updated for SYNC packets received from nodes that are not present in the neighbor list of a node. If the node does not receive any SYNC packet with *CH_flag* set to 1, in the initial listening time, it would choose its own schedule and broadcast it in the next SYNC cycle. Later on when it receives SYNC packet with *CH_flag* set to 1, the node will drop the schedule it is following and follows the cluster head's schedule. The nodes broadcast SYNC packets until they receive a SYNC with *CH_flag* value set to 1.

The SYNC packet format of nodes operating CBMAC is as shown in Table 4.1.

Table 4.1 – Field Definition of CBMAC SYNC Packet

Field	Description
type	Flag indicating this is a SYNC packet
length	Fixed size with 9 bytes
srcAddr	ID of the sender
syncNode	ID of the senders synchronization node
sleepTime	Sender's next sleep time from now
crc	Cyclic Redundancy Check
CH_flag	If the flag is set to 1, then the SYNC packet is sent from Cluster head.
energy_sensor	Energy remaining at the sender node

The *type* field is checked to identify the received packet as SYNC packet. The size of the packet is specified by *length* and the default value is set to 9 bytes. The *srcAddr* field is filled with the sender address, ID of the node. The *syncNode* address specifies the ID of the node whose schedule is being followed. $sleepTime$ specifies the timer value, when the node goes to sleep.

The *CH_flag* and *energy_sensor* carry very important information. The *CH_flag* lets the node

to check if the SYNC packet received is from cluster head. If the node receives two SYNC packets from different nodes and both packets have *CH_flag* set to 1, then the node gets synchronized to both cluster heads and becomes a border node. In such a case the node will follow both the schedules. *energy_sensor* specifies remaining energy at the sender node. This information along with *numNeighb* (field specifying number of neighbors a node has) is used to elect the cluster head.

The nodes operating under CBMAC maintain the same schedule tables as nodes operating under S-MAC do. They maintain a primary schedule and secondary schedules, where the primary schedule is the one that is being followed in the cluster, which is set by the cluster head. The border nodes have secondary schedule and they carry the second cluster head's schedule in that space. The default value for maximum number of schedules that can be followed by a node is set to 4 and is user-adjustable.

4.4.2 Choice of Duty Cycle for CBMAC

As discussed earlier when there is data to send, the sleep and listen times might be different for the cluster nodes and cluster head in CBMAC protocol. The function handling the timer values is located at *S-MAC::S-MAC( )* in NS-2. The sleep, listen and cycle times are calculated using the following expressions:

$$Listentime = synctime + datatime \tag{4.2}$$

$$Cycletime = \frac{listentime}{Dutycycle\%} \times 100 \tag{4.3}$$

$$Sleeptime = Cycletime - Listentime \tag{4.4}$$

In CBMAC the default listen time for all the nodes is assigned, such that the ratio of listen time in S-MAC to listen time in CBMAC is 1.6. The ratio is varied from 1.1 and incremented in steps of 0.1 until 1.6. The duty cycle also changes effectively. The values for different factors is calculated and the results are tabulated in Table 4.2.

With 0.6 as the factor, 53.7 sec of additional sleep time per frame is provided to every node in CBMAC. 53.7 sec for each frame would aggregate to a very large value when the entire simulation

Table 4.2 – Sleep Time in S-MAC and CBMAC

Factor	CBMAC Duty cycle (%)	S-MAC Listen time (sec)	CBMAC Listen time (sec)	Additional sleep time in CBMAC (sec)	CBMAC Sleep time (sec)	CBMAC/S-MAC Cycle time (sec)
0.1	18.14	143.2	130.1	13	586.9	717
0.2	16.63	143.2	119.3	23.9	597.7	717
0.3	15.35	143.2	110.1	33.05	606.9	717
0.4	14.26	143.2	102.3	40.9	614.7	717
0.5	13.3	143.2	95.4	47.8	621.6	717
0.6	12.48	143.2	89.5	53.7	627.50	717

time is considered. The choice of 0.6 is based on the following simulation. A multi-hop network

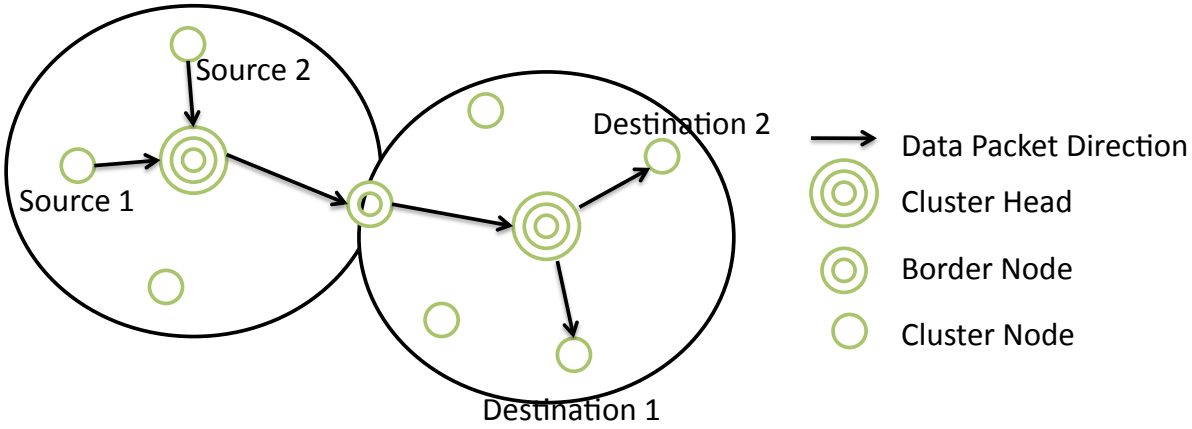

Figure 4.6 – Multi-hop network

with 9 nodes form two clusters, with four nodes in each cluster and a border node as shown in Figure Referencesfig:Fig10.5. The entire simulation is run with packet inter-arrival time as 10 sec for a simulation time of 1000 sec. The traffic is sent from two different nodes of one cluster to two other different nodes of another cluster. The data packet size is 80 bytes and they are sent for 200 sec and the number of packets received at the sink are observed. Beyond the factor 0.6, the sleep time in nodes is very large and the nodes are not able to deliver packets to cluster head, and cluster head is not able to forward them if it has received any from the nodes. As a result, packets get dropped at the sender node or intermediate nodes. So the value of 0.6 is chosen to maximize

the sleep time of nodes and the corresponding duty cycle is 12.48%. The traffic rate is varied by adjusting packet IAT from 1 sec to 50 sec, the factor 0.6 is observed to be ideal in all the cases.

The initial duty cycle for all nodes operating on CBMAC is 12.48%. For duty cycle of 12.48%, the listen time is 89.5 sec, sleep time is 627.5 sec and cycle time is 717 sec. The listen time consists of sync time and data time, and therefore 89.5 sec of listen time is equal to sum of 34.5 sec of sync time and 55 sec of data time. When a node enters into listen state, it recieves the SYNC packet from cluster head and gets synchronized to the cluster head. When it has data to send, it uses the data period to contend for the channel and forward the data packet. If it does not have any data to send it simply switches to sleep state after getting synchronized with the cluster head. Two scenarios can be considered to explain the adaptive duty cycle in CBMAC. Consider scenario one, where there is no data to forward. The sender node would only wakeup to get synchronized with the cluster head and the duty cycle will drop to as low as 4.8%. This is because the node only spend 34.5 sec of listen time. In scenario 2, consider there is infinite amount of data to forward. The node contends for the channel access and if it wins the contention, it will forward the data to the cluster head. After sending one packet it again contends for the channel to send the second packet. There are two possible cases, first case is that the node always wins the contention of the channel (this could happen when no other nodes are contending for the channel access). In such a case the node always spend time in listen state to send all the data it has and hence the duty cycle is 100%. In second case, when there are other nodes contending for the channel access, when one node wins the channel access, the other nodes will freeze their NAV and switch back to sleep state, until the channel gets free. The amount of time to sleep is known from the control packets over heard. Hence the duty cycle may vary from 12.48% to 100%, when there is infinite amount of data.

4.5 Cluster Head in CBMAC

Cluster Head Election: When the node receives a new SYNC packet, the variable *numNeighb_* is updated if the SYNC packet is received from the node that does not exist in the neighbor list. The neighbor list in CBMAC is similar to that of S-MAC and the number of neighbors in the

neighbor list can be accessed by *numNeighb_* variable. Also if the *CH_flag* value is one (implying that the SYNC is sent from cluster head) in that SYNC packet, the node follows that schedule. After processing the received SYNC and before broadcasting the next SYNC, cluster head election algorithm is run. If the node becomes the cluster head, it sets the *CH_flag* to one in its SYNC packet and broadcasts it in the next SYNC period. This election process takes place in the function *S-MAC::handleSYNC*.

For simplicity, all the energy values for different nodes are stored in an array. The number of neighbors for every node is updated in $number\_neighb$ array. Based upon the energy and number of neighbor's values and according to the following algorithm, the cluster head is elected. The code for cluster head election is included in the Appendix.

Let *E[ ]* be the array that stores all the energy values of all the nodes and *N[ ]* be the array that stores the number of neighbors for all the nodes. When the following algorithm is run for scenarios where there are more than one cluster, there is a possibility for only one cluster head to be elected. Therefore, the nodes are set static and the algorithm is run separately for every cluster. This ensures that every cluster has an elected cluster head.

- The node that has the highest value of *N* set its *CH_flag* to one and broadcasts its SYNC.
- If two nodes have same *N* value, their corresponding *E* values are compared.
- The node with the highest *E* value sets its *CH_flag* to one and broadcasts its SYNC.

The above steps are run separately for every cluster in case of Inter-cluster scenario. The cluster head election algorithm is written for stationary nodes and therefore the case where cluster head moves away from its cluster is not considered.

Once the cluster head is decided, the *CH_flag* for the corresponding node is set to one and a SYNC packet is broadcasted. When the nodes receive the SYNC packet they check for the *CH_flag* field, and follow the schedule of the cluster head. The broadcasting of SYNC packet is handled by the function *S-MAC::sendSYNC*. Before the cluster head election takes place, every node in the cluster enters the function *S-MAC::sendSYNC*, but once the election is done, only cluster head

will enter the function *S-MAC::sendSYNC* to broadcast its SYNC. Initially all the nodes keep broadcasting their SYNC packets, until they receive a SYNC from cluster head.

Minimal energy is consumed for the cluster head election and SYNC exchanges because of the following conditions.

- After the cluster head election happens, the SYNC packets are only sent by cluster head.
- The cluster head election does not take place very often. The default value is adjusted to be 100 Synchronization periods. (one Synchronization period = ten frames).
- The cluster nodes do not broadcast SYNC packets. They receive SYNC packets from cluster head and then SYNC their schedules with cluster head.

Maintenance of Cluster Head: After the cluster head is elected, all the data packets are forwarded to cluster head and the cluster head forwards them to destination or border node. The cluster head will increase its listen time to accomplish this. Therefore, it is evident that cluster head consumes more energy on an average because of its increased listen time compared to other nodes in the cluster. So in order to have uniform energy utilization in the network, another algorithm is proposed, which is as shown below. Let $Eavg$ be the average energy of all the nodes in a cluster. The algorithm in Table 4.3 should be checked for every 1000 Synchronization periods and the corresponding action is to be taken.

Table 4.3 – Cluster Head Maintenance Algorithm

Condition	Action
If cluster head energy is greater than 70% of $Eavg$	The node continues to be cluster head
If cluster head energy is less than 70% of $Eavg$	The cluster head retires and the node with next highest number of neighbors becomes the cluster head.

The factor of 70% is a safe assumption and it is user configurable. The value can be changed depending on the traffic in the network. To explain the cluster head maintenace algorithm in detail, a single-hop scenatio as shown in Figure 4.7 is considered to differentiate the energy consumption

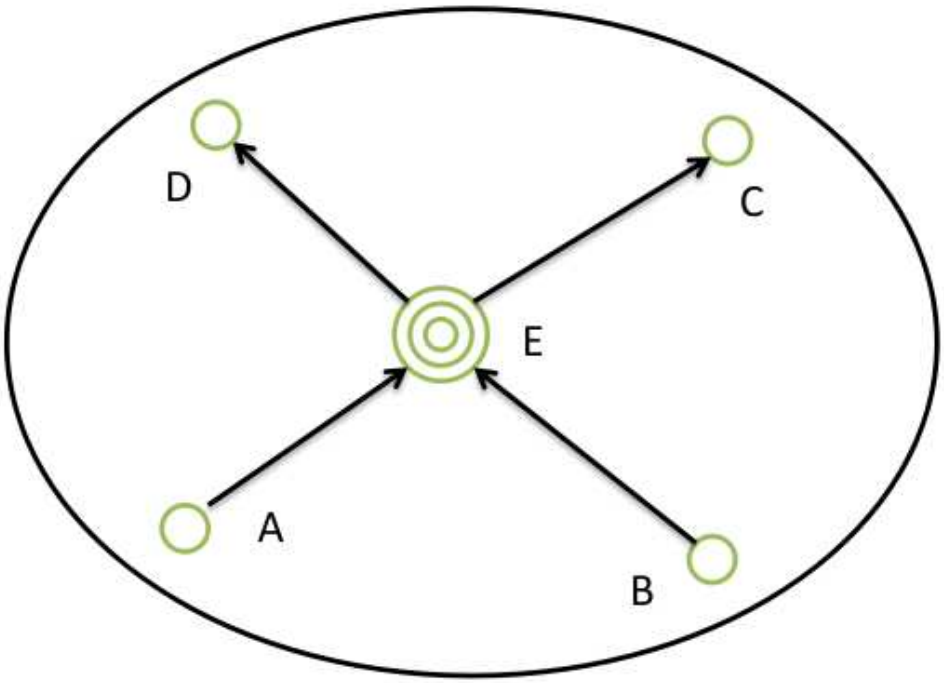

Figure 4.7 – Single hop scenario to calculate energy values of cluster head and cluster nodes

at cluster head and cluster node. The simulation time is set to 10000 sec and traffic is sent for 2000 sec. The packet size is set to 80 bytes. The packet IAT is varied from 1 sec to 50 sec, where A and B are sending data packets to C and D, respectively at the rate of 640 bits/sec.

The energy values of cluster head are captured along with other nodes in the cluster and tabulated in Table 4.4. The initial energy for every node is 1000 Joules.

Table 4.4 – Energy Consumed by Cluster Head and Nodes in CBMAC

Energy consumed by the Nodes for different Inter-Arrival Times (IAT)											
IAT	1	5	10	15	20	25	30	35	40	45	50
A	1000	241	139	118	109	109	98	95	93	93	89
CH	891	225	134	144	131	131	124	120	121	116	118
B	1000	259	145	122	110	106	99	96	92	94	90
C	918	222	135	115	106	106	97	94	90	91	89
D	940	224	144	116	106	106	96	93	90	90	88

The energy values shown are the energy remaining at each node after the last data packet is received at the destination. In the Table 4.4 it is observed that the energy consumed by the cluster

head is more than that of the other nodes for IAT $\geq$ 15 sec (under low load). The reasons for cluster head's low energy consumption for IAT $<$ 15 and high energy consumption for IAT $\geq$ 15 are as follows:

- Due to contention and collision. As all the nodes try to foward data to a common cluster head, the contention for channel access increases and collisions also increases rapidly. Due to this the sender nodes A and C try to gain channel access to transmit data to cluster head very often. For IAT $<$ 15 sec, there is high data generated at nodes A and C, which leads to increased energy consumption at nodes A and C compared to cluster head.

- In CBMAC, the duty cycle value is intitially very low and when there is data, the apaptive duty cycle increases listen time to forward the data. As every node that has data forwards it to cluster head, the cluster head extend its listen time to forward all that data. Hence the energy consumption at cluster head for IAT $\geq$ 15 is higher than the consumption at nodes.

For the cluster head maintenance algorithm to run, the following possibilities should be considered. When the cluster head's energy is less than 70% of the *Eavg* :

- Another node in the network with same number of neighbors and energy $\geq$ *Eavg* can be elected as cluster head.

- When mobility is enabled for the nodes in the network, a mobile node may join the cluster and acquire same number of neighbors as the cluster head. Then the new node can be elected as the new cluster head.

- When the above two cases fail, the cluster should be divided into two clusters and the election process should be run again individually for both the clusters.

The cluster head maintenance feature is not implemented and is left open for future work because the simulations assume stationary nodes, hence cluster head does not move away throughout the simulation nor any new nodes join the cluster. The reasons for not implementing the cluster division:

- Under moderate and high traffic, cluster head will not run out of energy before cluster nodes.
- In all other cases, the cluster head's and cluster node's energy decreases simultaneously.

The CBMAC and S-MAC are simulated in two different scenarios

1. Intra-cluster scenario (Single hop)
2. Inter-cluster scenario (Multi hop)

The results are compared considering the following important network performance criteria: energy consumption, latency and throughput parameters. The results are discussed in detail in Chapter 5.

4.6 Choice of Routing Protocol

Routing agent is important in simulating a wireless network, as it specifies the routing information such as next hop address for the unicast packets. The nodes cannot deliver packets without the routing information. NS-2 supports four routing protocols for wireless simulations, which are DSDV (Destination-Sequenced Distance-Vector), DSR (Dynamic Source Routing), TORA (Temporally Ordered Routing Algorithm), and AODV (Ad hoc On-demand Distance Vector). S-MAC is implemented using DSR protocol.

A large delay is possible when S-MAC is run on multi-hop network due to the sleep time at each node. Generally, the routing protocols are designed to provide routing support for IEEE 802.11 type of network with high bandwidth and low latency. Due to sleep at each node operating on S-MAC, the protocols timeout very fast in finding paths, and are not able to establish a routing path. S-MAC runs successfully with DSR by modifying some user adjustable parameters in DSR protocol.

The routing is established by exchanging routing packets between the nodes in a network. A considerable amount of energy is consumed in exchanging routing packets. To study the energy performance of a protocol at MAC layer, energy consumed at MAC layer is of more interest than

the energy consumed at network layer. To address this issue, a third-party routing agent is being used to study S-MAC. The routing agent used is NO Ad-Hoc routing agent (NOAH), which is a wireless routing agent that supports static multi-hop routing. NOAH is quite suitable for simulation scenarios where routing overhead is not desired. More information on usage/installation of NOAH can be found at the NOAH webpage [15].

To explain the energy consumption when a routing protocol is used, S-MAC is simulated with DSR and NOAH routing agent separately, and the energy consumed in both the cases are plotted in Figure 6.5. The scenario consisits of 5 nodes as shown in 4.7, where A and B are sending data to C and D, respectively. The simulation is run with cbr[4] traffic set for 1000sec. The packet size is 80 bytes and the packet inter-arrival times ranging from IAT = 1 to IAT = 10 sec, one pkt/sec is higher load than one pkt per 10 sec.

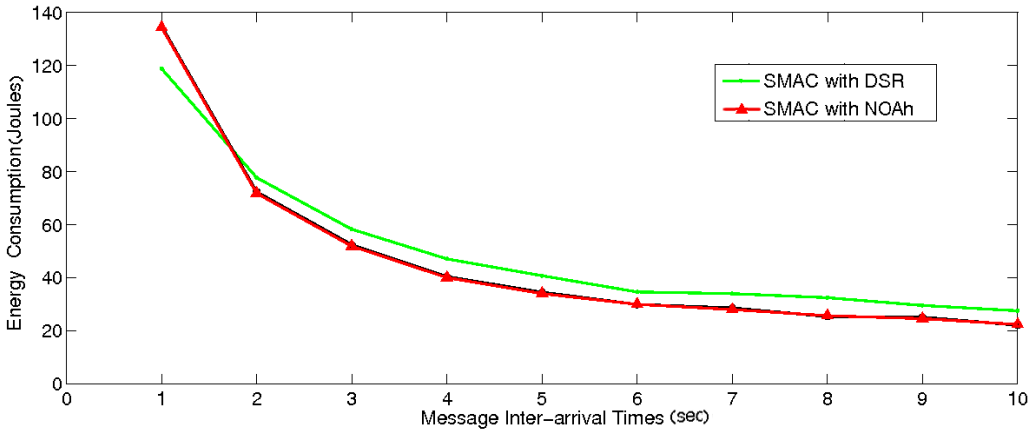

Figure 4.8 – Energy consumption under DSR agent vs NOAH agent

The scenario at one packet per second is set to send 1000 packets in a multi-hop scenario. While running S-MAC with DSR agent, all 1000 packets are not received at the sink, which explains the low energy consumption in S-MAC running with DSR. From the above energy consumption graph it is deduced that the dynamic routing protocol like DSR would consume significant amount of energy when transmitting packets. Whereas NOAH uses static routes and it does not send any routing related packets. Hence performance of S-MAC is studied using NOAH routing agent.

[4]constant bit rate

CHAPTER 5

5 PERFORMANCE EVALUATION OF CBMAC PROTOCOL

6.1 Introduction

To evaluate the performance of CBMAC against S-MAC, both protocols are simulated in single-hop and multi-hop scenarios. The single-hop scenario does not involve border node as the sender and receiver are located in the same cluster. Whereas in multi-hop scenario, border node(s) come in to picture as the border nodes forward the data from one cluster head to another. NOAH is set as the routing agent and cbr traffic is set between the nodes. The scenario Intra-cluster provides the single-hop scenario and the Inter-cluster scenario generates the multi-hop scenario. Both these scenarios are explained in Sections 5.2 and 5.3, respectively. The simulations are run with traffic load varying from very high to low. Three network parameters are used to compare S-MAC and CBMAC. They are

- Energy consumption: The energy consumed to receive the entire data at the destination. The energy value of every node is read at the instance when the ACK packet is received for the last data packet. The mean of the energy values is plotted for various IAT values.
- LAN Throughput: Overall LAN throughput is calculated in bits per second by sending fixed amount of data from source to destination. LAN throughput is rate of successful message delivery over a communication channel.
- Latency: The latency calculated is average delay of all the packets successfully delivered from the source node to the destination node.

The simulation time is set to 10000 sec for all the simulations involving energy, throughput and latency calculations. The default buffer size for every node is 1000, which implies that a node can store up to 1000 packets in its buffer before delivering them to next hop. The performance is studied by plotting each parameter separately for both the protocols. The simulations are performed

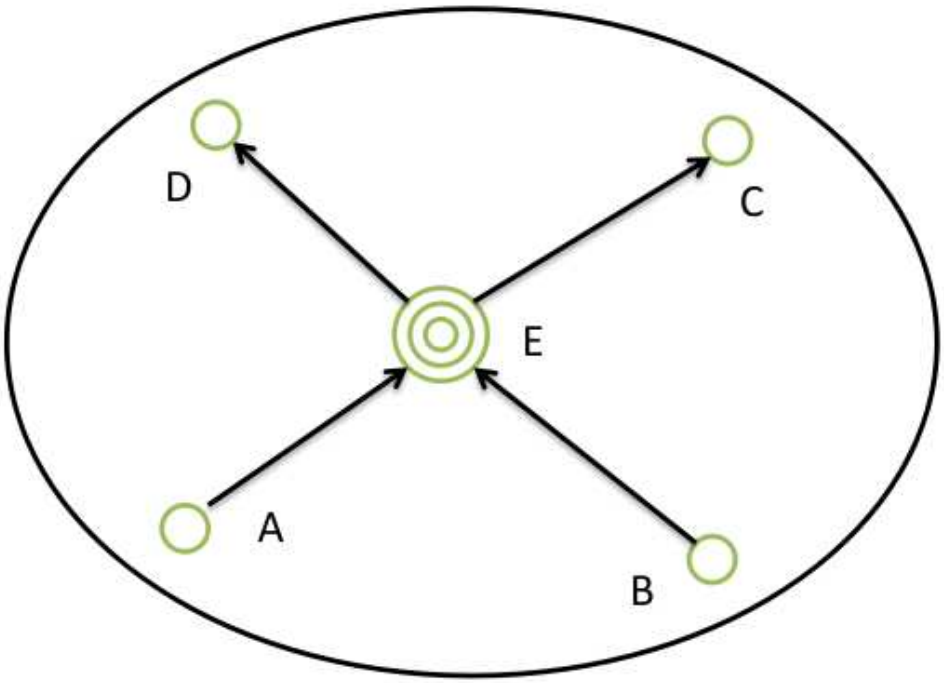

Figure 6.1 – Intra-cluster scenario

in NS-2, a discrete-event-driven simulator. The tcl script used to run the simulations is attached in the Appendix.

6.2 Intra-Cluster Scenario

To evaluate the performance of S-MAC and CBMAC in a single cluster scenario, a cluster with 4 nodes is considered and cbr traffic is set to flow across the cluster head. NOAH static routing is applied and cbr traffic rate is varied from one packet per second to one packet per 50 seconds. The packet size is 80 bytes and is constant for all the network parameters evaluation.

The intra-cluster scenario is as shown in Figure 6.1, where nodes A and B are sending traffic to nodes C and D, respectively.

6.2.1 Energy Consumption

The energy consumption plotted is the aggregate energy consumed on all the nodes in the network. The simulation is run for 10000 sec, by varying the cbr packet IAT at the source node

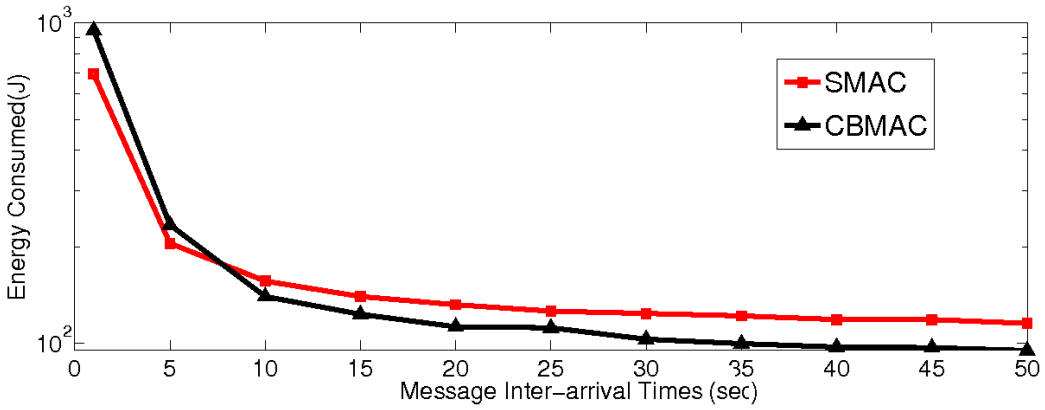

Figure 6.2 – Aggregate energy consumption for S-MAC and CBMAC for various inter-arrival times

from one sec to 50 sec, in steps of 5, where one sec means all the cbr packets generated are queued with one sec interval at the source node, which is the high load scenario. Since NS-2 does not support zero inter-arrival time for traffic sources, IAT of one sec is used as the heavy traffic scenario. The 50 sec IAT is considered to simulate low load scenario. The cbr packets are set to generate at the source node for 1000 sec, between 2000 sec and 3000 sec. The average energy consumed on each node is calculated at the instance where last CBR packet is recieved at the reciever. The duty cycle is 20% and the initial node energies are 1000 Joules each. The average energy consumed by nodes operating on S-MAC and CBMAC vs Inter-arrival times are plotted in Figure 6.2.

For IAT of one to five, there is high traffic flow in the network. The buffer on every node is set to 1000 and under high traffic condition the buffer is never empty. So the nodes have to perform carrier sense or wait for other's CTS most of the time, which happens as long as the node has data to send. This explains the high energy consumption for S-MAC and CBMAC under high load. In order to access the cluster head and forward the packets, the nodes operating CBMAC contend to gain access to the cluster head actively. As the traffic rate is set in such a way that both the senders have data to send at the same time, this could lead to collisions at the cluster head. The two major reasons for high energy consumption at nodes operating under CBMAC, when traffic load is high are:

- First, all the nodes try to foward data to a common cluster head, the contention for channel access increases and collisions also increases rapidly. Due to this the sender nodes A and C try to gain channel access to transmit data to cluster head very often. For IAT $<$ 8 sec, there is high data generated at nodes A and C, which leads to increased energy consumption.

- When there is data to send, the cluster nodes and the cluster head increase their time spent in listen state and try to forward the data.

These two factors degrade CBMAC performance when the inter-arrival time is less than eight sec (one packet per eight sec). The amount of energy saved in CBMAC is tablulated as shown in Table 6.1. The first two values of energy savings are negative because S-MAC has more savings under high-load scenario.

Table 6.1 – Energy Consumed by Nodes Operating CBMAC and S-MAC in Intra-Custer Scenario

Average energy consumed by the nodes for different Inter-Arrival Times (IAT)											
IAT	1	5	10	15	20	25	30	35	40	45	50
Energy consumed by S-MAC	526	128	91	73	74	67	67	63	61	62	64
Energy consumed by CBMAC	610	160	86	65	59	45	42	36	31	28	26
Amount of energy saved	-84	-40	5	8	15	22	25	27	30	34	38

From IATs 10 to 50 the CBMAC outperforms energy savings of S-MAC. This can be accounted to the extra wakeup time in nodes operating on S-MAC when there are no packets to send. Energy savings at the nodes operating under CBMAC are the result of the extended sleep time they achieve by decreasing the duty cycle to a smaller value as explained earlier.

From this it is evident that CBMAC energy performance is better than S-MAC when the inter-arrival time is greater than 8 secs.

6.2.2 Throughput

This section describes the throughput achieved in S-MAC and CBMAC for various inter-arrival times. The throughput calculations are performed by sending cbr packets of size 80 bytes each for a period of 1000 seconds. The number of packets received is observed and throughput is calculated

according to the following equation.

$$Throughput(bits/sec) = \frac{No.of packets received * Packet size * 8}{t_2 - t_1} \tag{6.1}$$

- Throughput calculated is in bits per second.
- No. of packets received is observed at the sink.
- Size of each CBR packet is 80 bytes or 640 bits.
- t_2 is the time when the ACK for last packet is received.
- t_1 is the time when the very first CBR packet is sent.

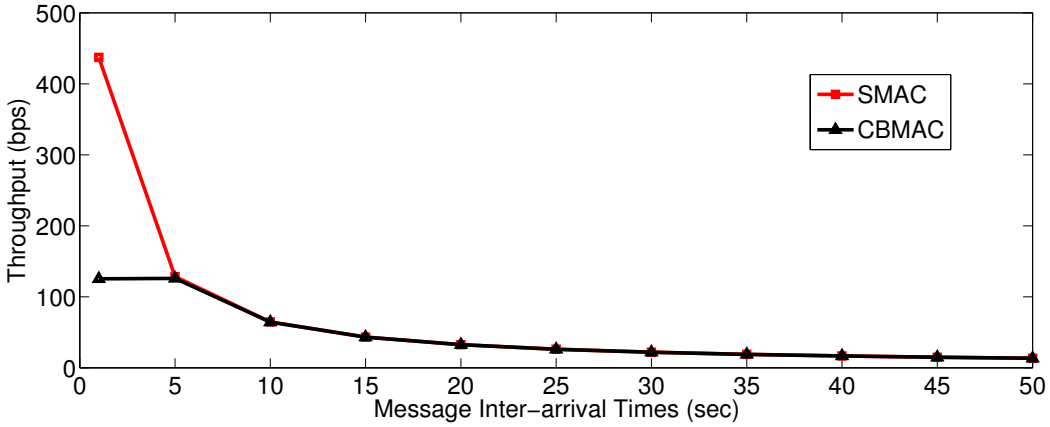

Figure 6.3 – Throughput (bits/sec) in S-MAC and CBMAC for different inter-arrival times (sec)

The overall throughput performance of CBMAC is similar to that of S-MAC for inter-arrival times greater than 5 seconds as shown in the Figure 6.3. The low throughput in CBMAC is due to the high traffic flow in the network. From the above equation, with increase in t_2 value, the denominator value increases and hence the throughput decreases. This implies that the packet delivery time should be minimal to achieve high throughput value. When all the packets are sent with no delay, then the protocol can achieve ideal throughput. In this case, due to increased sleep time in CBMAC, delay gets accumulated on each packet and hence the time t_2 increases. This

results in low throughput value. However the CBMAC performs as well as S-MAC for inter-arrival times greater than 5 until 50, because the packets arriving at or after 5 seconds provide sufficient time for previous packets to reach the sink. The delay analysis is performed in the next section.

6.2.3 Latency

The objective is to compute mean latency of all data packets at the various traffic loads across the network. The latency on a packet can be defined as the difference between the packet received time and the packet sent time. The delay in the network can be due to various reasons like the destination node might be in sleep state, transmission delay due to collisions, packet processing time, dropped packets etc.

The delay in S-MAC and CBMAC is computed in a single-hop scenario by setting the traffic similar to that used in throughput calculation. The plot for average packet latency vs inter-arrival times is shown in Figure 6.4.

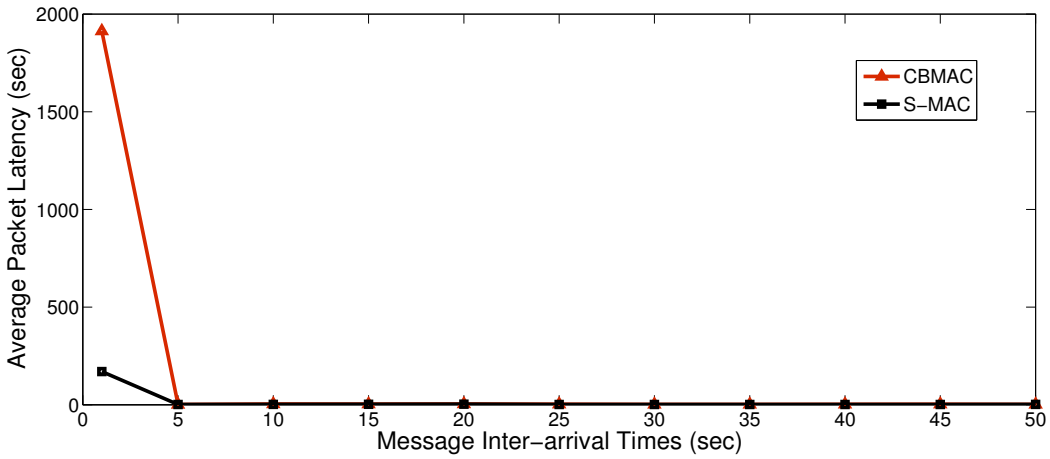

Figure 6.4 – Latency in S-MAC and CBMAC for different inter-arrival times

The latency performance in CBMAC is worse under high traffic because of the sleep time in CBMAC. Latency is major tradeoff for energy. When nodes are set to listen the medium all the time, the packets are processed with minimum delay. In CBMAC, the nodes spend large amount

of time in sleep state. The additional sleep time in CBMAC adds significant delay to each packet. Figure 6.4 depicts that under high load scenario, the data to send is high and hence the delay increases. Whereas in low load scenarios, the delay is minimal as the data to send is also minimal. However for inter-arrival time greater than 5 seconds, the CBMAC shows steady performance as the S-MAC. In a real time WSN, the speed of the physical object sensed is less than the network speed, and therefore delays in order of few seconds are permissible. Also CBMAC performs identical to S-MAC after 5 sec IAT, which indicates that CBMAC is preferable for data rate 0 - 128 bits/sec.

As the objective of this thesis is to achieve energy savings, the delay shown in the plot for inter-arrival times less than 5 sec is considered as a tradeoff for energy savings . And the CBMAC shows similar delay times as S-MAC for inter-arrival times greater then 5 seconds, which makes CBMAC suitable under low traffic load scenarios.

6.2.4 Extended Sleep Time in CBMAC (Intra-Cluster Scenario

The extended sleep time in CBMAC is measured and plotted as shown in Figure 6.5. The scenario consists of 5 nodes arranged in single-hop as shown in Figure 6.1. The IAT between packets is set to 15 sec and data is sent from nodes A and B to nodes C and D, respectively. The packet size is 80 bytes and the traffic is sent for 1000 sec. The simulation time is 2,500 sec and the amount of time each node spends in sleep state is captured and plotted. The node E is cluster head and it has less sleep time as it recieves data from sender nodes A and B and deliver the data to node C and D, respectively. Hence the sleep time in nodes A, B, C, and D is more when compared with sleep time in node E. As shown in the figure 6.5, the nodes operating on CBMAC spend more time in sleep state than nodes operating S-MAC. This accounts for the energy savings in CBMAC protocol.

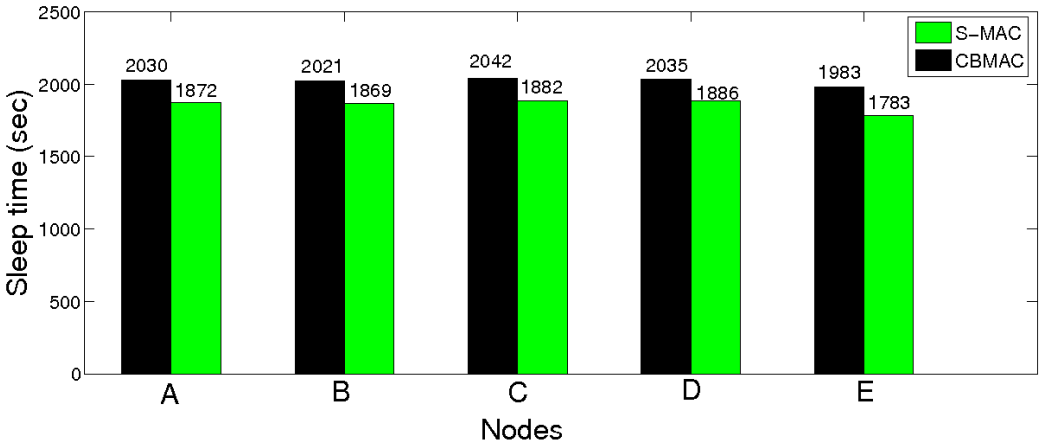

Figure 6.5 – Extended sleep time in CBMAC compared with S-MAC in intra-cluster scenario

6.3 Inter-Cluster Scenario

In this section performance of CBMAC in inter-cluster scenario is studied with respect to energy, throughput and latency. The inter-cluster scenario is different from Intra-cluster scenario. Some of the differences are

- The border node is introduced in inter-cluster.
- As the data packets are set to traverse through many hops, the study of energy and latency becomes interesting
- Cluster head coordinates with border node to forward the data.
- The energy consumption on cluster head and border nodes is intense and this can be observed.
- Overall performance of the S-MAC and CBMAC can be analyzed.

To study CBMAC and S-MAC in inter-cluster scenario, the following scenario is considered.

The topology is set such that the source and destination are 6 hops away from each other and the communication between them involves 3 cluster heads and 2 border nodes. The CBR traffic is

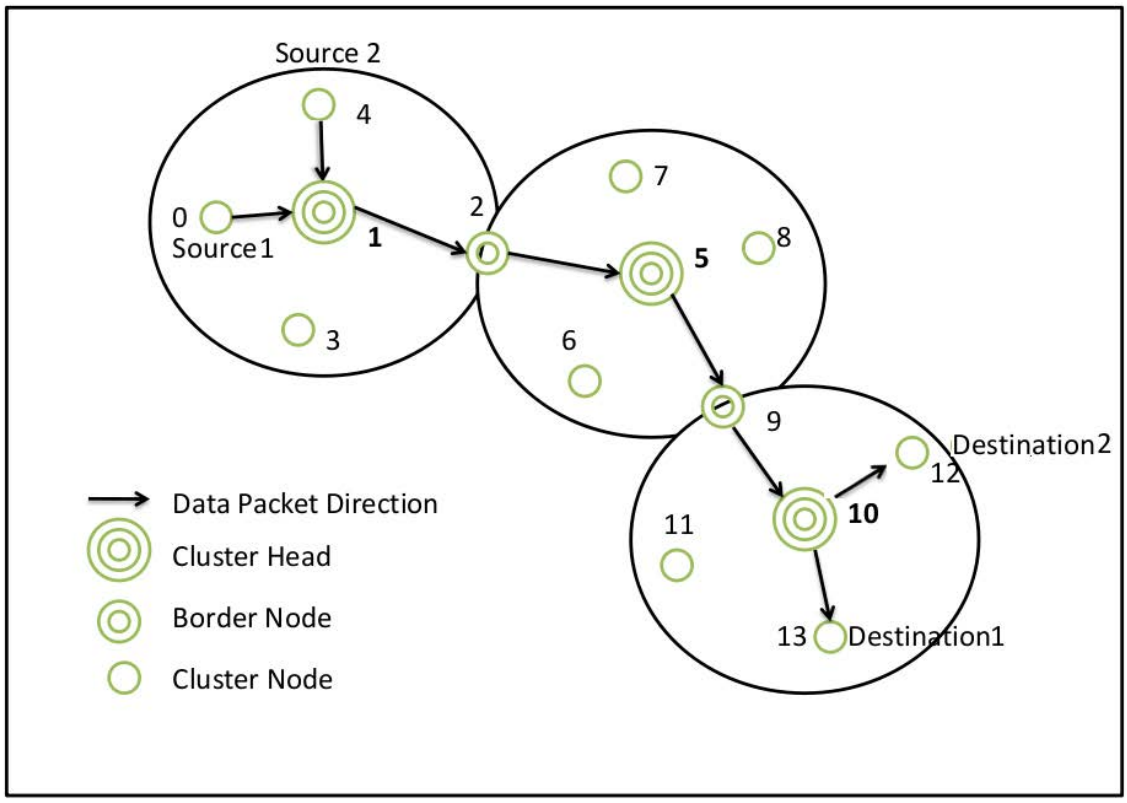

Figure 6.6 – Inter-cluster scenario

set for 1000 seconds and the IAT is varied from one to 50, one being the high load scenario and 50 being low load. Two nodes 0 and 4 send traffic to nodes 13 and 12, respectively. Each data packet is 80 bytes long, and as already said the interval between two data packets is varied from one to 50. The data rate is therefore 1280 bits/sec for IAT = one sec and 25.6 bits/sec for IAT = 50 sec. The IAT = one sec serves as the high load scenario and IAT = 50 sec is the low load scenario. The performance of CBMAC and S-MAC is observed and plotted for energy, throughput and latency. The duty cycle in S-MAC is set to 20%, where as in CBMAC the duty cycle adaptively varies from 12.48% to 100% depending on the following conditions.

- When there are no other nodes contending for the channel, then the node which has data will spend time in listen state, contending for the channel.

- When there are other nodes contending for the channel, the nodes which lost the contention will switch back to sleep state until the channel is free again. The amount of time to sleep is derived from the control packets.

NOAH routing protocol is set and static routes are setup. In order to simulate S-MAC with NOAH as close as S-MAC with a dynamic routing agent, the node positions are set in such a way that routing with a dynamic routing agent and NOAH follow similar pattern. The cluster heads 1, 5 and 10 are observed to have sleep time of 573.8 seconds, whereas the cluster nodes have sleep time of 627.5 for the cycle time of 717 seconds. The cluster head, border node and cluster node have same sleep time, but cluster head and border node extend their listen time when they have more DATA to handle. The performance of CBMAC against S-MAC is shown in the following sections.

6.3.1 Energy Consumption

Figure 6.7 shows the energy consumption of S-MAC and CBMAC for different inter-arrival times. The energy consumed for CBMAC and S-MAC to send DATA packets from node 0 to node 13 and node 4 to node 12 is plotted for various inter-arrival times in the following figure. CBMAC shows energy savings for inter-arrival times greater than 9, and the reasons for poor performance of CBMAC under high-load scenarios are discussed below.

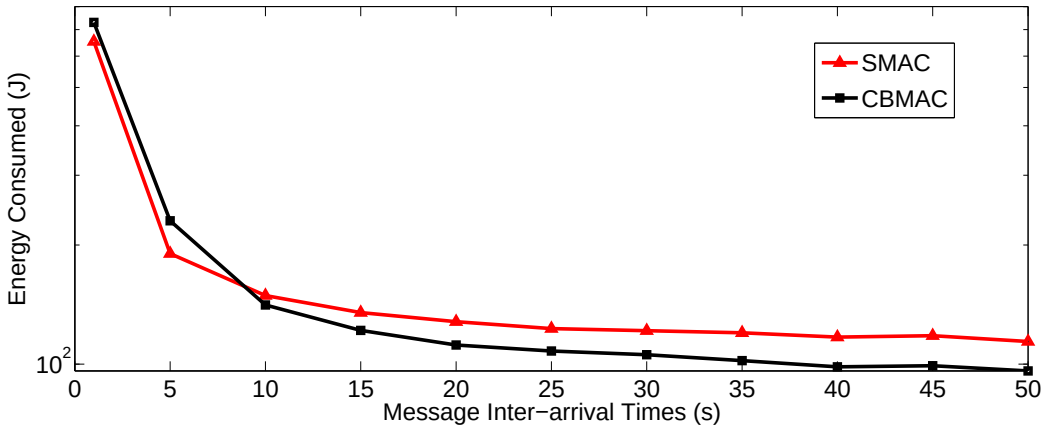

Figure 6.7 – Energy consumption in S-MAC and CBMAC for different inter-arrival times

For IAT 0 to 9 sec the CBMAC protocol's energy consumption is more than S-MAC protocol, and CBMAC shows consistent energy savings after 10 seconds. The reason for the poor performance of CBMAC: as the DATA packets are forwarded through cluster head and cluster head forwards them to border node and so on, the ACK packets also travel in the same path. The node 0 extends its listen time to receive ACK packets from the destination. This wait time increases for node 0 under high load. This accounts for poor performance of CBMAC under high load scenarios. For inter-arrival times 10 and above, the traffic is very low and the data packets are forwarded without extending the listen time to a higher value. This will let the nodes spend more time in sleep state and hence energy savings are maximized in CBMAC compared to S-MAC after IAT of 10 sec. In case of IAT less than 10, there is high traffic in the network and the cluster head and border node extend their listen time to a higher value to forward the data to the destination. For IAT = one to ten sec, the average energy consumption in CBMAC is increased as cluster head and border node spend more time in listen state and consume more energy to transfer the DATA packets. Also the path followed by DATA packets from node 0 and from node 4 is similar and this introduces contention in the network. The amount of energy saved in nodes operating CBMAC is as shown in Table 6.2. S-MAC has more energy savings for IAT 1 and 5, due to this the energy savings in CBMAC are negative.

Table 6.2 – Energy Consumed by Nodes Operating CBMAC and S-MAC in Inter-Cluster Scenario

Average energy consumed by the nodes for different Inter Arrival Times (IAT)											
IAT	1	5	10	15	20	25	30	35	40	45	50
Energy consumed by S-MAC	653	190	149	135	128	126	125	127	124.4	124	126
Energy consumed by CBMAC	730	230	141	122	112	108	106	102	98.4	95	92.2
Amount of energy saved	-77	-40	8	13	16	18	19	25	26	29	33.8

In order to improve the energy savings in CBMAC for inter-arrival times less than 10 seconds, message passing could be introduced. Message passing is one of the features of S-MAC, which transmits long messages in a burst of small individual packets with single RTS and CTS. This will reduce the control overhead in CBMAC under high load scenario to some extent which can add to energy savings.

6.3.2 Throughput

Throughput for S-MAC and CBMAC is calculated according to the Equation 5.1. The throughput values are plotted against message inter-arrival times varying from 1 to 50 sec in Figure 6.8.

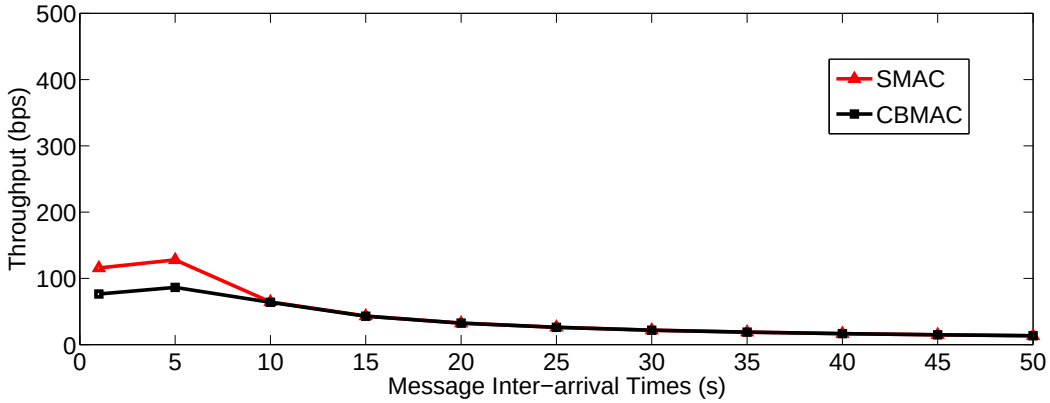

Figure 6.8 – Throughput in S-MAC and CBMAC for different inter-arrival times

CBMAC displays throughput drop for message inter-arrival times of 10 sec and lower. As the distance between sender and receiver increases the difference between packet receive time and packet sent time also increases. This is due to the extended sleep time in cluster nodes and collisions at the cluster head. The last ACK packet received time is of higher order and hence the throughput drop. Throughput for inter-arrival times greater than 10 in CBMAC is same as S-MAC's throughput. As discussed in previous section, decreasing the control overhead can lead to increased throughput in CBMAC under high load scenarios.

6.3.3 Latency

Latency in energy-efficient protocols is given less importance as delay is greatly effected with periodic sleep in the nodes. The CBMAC protocol is designed to suppress the energy consumption and latency is one of the tradeoffs to achieve that. But, still the latency of CBMAC tends to be high only under high load scenarios. The overall latency in CBMAC is the result of the delay accumulated on each node due to the extended sleep introduced to achieve the energy savings. The latency for S-MAC and CBMAC follows the same pattern for inter-arrival times greater than 10

sec.

The latency in S-MAC and CBMAC for various inter-arrival times is plotted in the Figure 6.9.

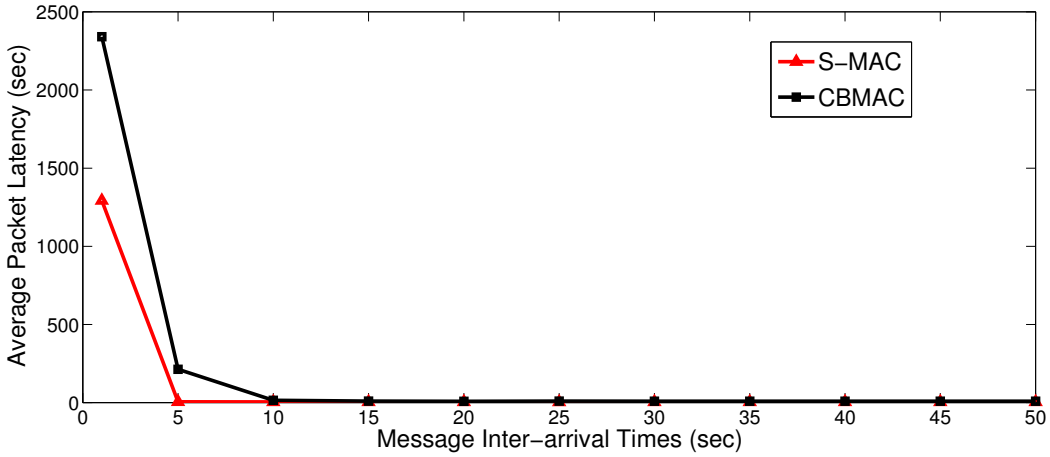

Figure 6.9 – Latency in S-MAC and CBMAC for different inter-arrival times

To improve performance of CBMAC in terms of latency, the sleep time in nodes should be decreased. This again depends on the type of application deployed. Most of the sensor networks do not experience high traffic and therefore latency is not given high priority in most of the cases. In cases where latency is important and data rate is high, we can tune CBMAC such that energy savings are traded to decrease latency. This can be achieved by increasing the duty cycle, for instance.

6.4 Extended Sleep Time in CBMAC (Inter-Cluster Scenario

The extended sleep time in CBMAC is measured and plotted as shown in Figure 6.10. The scenario consists of 14 nodes arranged in single-hop as shown in Figure 6.1. The IAT between packets is set to 15 sec and data is sent from nodes 0 and 4 to nodes 13 and 12, respectively. The packet size is 80 bytes and the traffic is sent for 1000 sec. The simulation time is 2,500 sec and the amount of time each node spends in sleep state is captured and plotted. The nodes 1, 5, and 10 are cluster heads and they consume more time in listen state as they handle the data most of the time. The nodes 2 and 9 act as border nodes and they spend a little more time in listen state to forward data between two cluster heads.

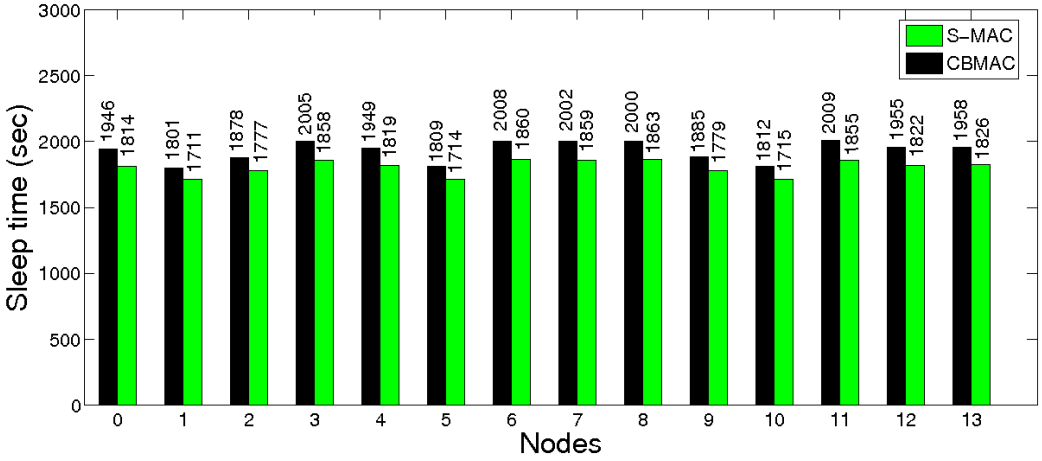

Figure 6.10 – Extended sleep time in CBMAC compared with S-MAC in inter-cluster scenario

6.5 Summary

The performance of S-MAC and CBMAC is compared under various traffic load scenarios. From the above results, overall performance of CBMAC exceeds S-MAC in energy savings and remains same in throughput and latency performance under light traffic loads. Under heavy traffic scenarios, either CBMAC should be used with higher duty cycle or S-MAC should be preferred.

CHAPTER 6

CONCLUSION

7.1 Conclusion

This thesis extensively evaluates the S-MAC protocol and proposes a new protocol named CB-MAC, which achieves energy savings over S-MAC. The factors affecting the energy utilization are identified and a new clustering mechanism is introduced through CBMAC. This mechanism significantly reduces synchronization overhead and redefines the communication procedure between nodes. The key factors that account for the energy savings are

- Reduced listen time in cluster nodes,
- Cluster head handling the data forwarding and
- Low control packet overhead.

From the results obtained, it is concluded that CBMAC achieves more energy savings than S-MAC under light load scenarios. And the reasons for low performance in CBMAC under very high traffic load scenarios are discussed. CBMAC with higher duty cycle value is suggested for high load scenarios to efficiently handle the high traffic in the network with minimal latency. CBMAC with low duty cycle provide excellent support for lighter traffic loads with latency as low as S-MAC and achieves similar throughput as S-MAC.

7.2 Future Work

The performance of CBMAC can be studied extensively under high load scenarios and depending on the traffic pattern, the duty cycle could be varied adaptively to minimize the delay. An optimal duty cycle value could be derived for various traffic scenarios to achieve a tradeoff between energy savings and latency.

The routing protocol used to study CBMAC uses static routes and does not support routing for mobile nodes. A dynamic routing protocol could be developed to support mobility in nodes and

provide robustness to CBMAC protocol. CBMAC could then be studied for various parameters of interest.

The cluster head maintenance can be further developed by studying the energy consumption rate at cluster head and cluster nodes and an algorithm can be devised for cluster head election. Also the cluster size could be extended by hops, which decreases the number of cluster heads in the network. Its performance should be studied under various traffic loads.

REFERENCES

REFERENCES

[1] W. Ye, J. Heidemann and D. Estrin, "Medium Access Control With Coordinated Adaptive Sleeping for Wireless Sensor Networks," in *IEEE/ACM Transactions on Networking*, vol 12. No. 3, June 2004.

[2] Wei Ye, Fabio Silva, and John Heidemann, "Ultra-Low Duty Cycle MAC with Scheduled Channel Polling," in *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pp 321 -334, November, 2006.

[3] S. Liu , K. Fan and P. Sinha, "CMAC: An Energy Efficient MAC Layer Protocol Using Convergent Packet Forwarding For Wireless Sensor networks," in *Proc. SECON* , pp 11-20, 2007.

[4] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," in *Proceeding of Sen-Sys*, pp. 95-107, November, 2004.

[5] T. Zheng, S. Radhakrishnan and V. Sarangan , "PMAC: An Adaptive Energy-Efficient MAC Procotol for Wireless Sensor Networks," in *Proceeding of the 19th IPDPS*, pp 237a, 2005.

[6] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specification," in *IEEE Std. 802.11*, 1999 edition.

[7] S. Singh and C. S. Raghavendran, "PAMAS: Power aware multi-access protocol with signaling for ad hoc networks," in *ACM Comput. Commun. Rec.,* vol.28, no. 3, pp 5-26, July 1998.

[8] O. Kasten, "Energy consumption. Eldgenossische Technische Hochschule Zurichin," in http://www.inf.ethz.ch/kasten/research/bathtub/energy_consumption.html, retrieved on 08/01/2010.

[9] V. Bharghavan, A. Demers. S. Shenker, and L. Zhang, MACAW A Media Access Protocol for Wireless LAN," in *Proceedings of ACM SIGCOMM*, pp 212-25, August 1994.

[10] M. Stemm, and R. H. Katz, "Measuring and reducing energy consumption of network interfaces in hand-held devices," in *IEICE Trans. Commun.*, vol. E80-B, no. 8, pp 1125-1131, August 1997.

[11] Hao Liu, Chen Hu, Longxin Shi, and Guoliang Yao, "A Cluster Based Adaptive Low Power MAC Protocol for Wireless Sensor Networks," in *Wireless Communications, Networking and Mobile Computing, WiCOM '08. 4th International Conference*, pp. 1-4, October 2008.

[12] Yuan Li, Wei Ye, and John Heidemann, "Energy and Latency Control in Low Duty Cycle MAC Protocols," in *Proceedings of IEEE Wireless Communications and Networking Conference*, pp 228-235, September, 2006.

[13] T. V. Dam and K. Langendoen, "An Adaptive Energy-Efficient MAC Procotol for Wireless Sensor Networks," in *Proceeding of Sen-Sys*, pp. 171-180, November, 2003.

[14] NS-2, "The Network Simulator - ns-2 homepage," in http://www.isi.edu/nsnam/ns/, retrieved on 01/28/2010.

[15] NO Ad-Hoc Routing Agent (NOAH), in http://icapeople.epfl.ch/widmer/uwb/ns-2/noah/, retrieved on 06/06/2010.

[16] IEEE standard for Wireless LAN-Medium Access Control and Physical Layer Specification, P802.11, November 1997.

APPENDIX

APPENDIX

A. SYNC Packet Format

```
struct S-MAC_sync_frame {
  int type;
  int length;
  int srcAddr;
  int syncNode;
  double sleepTime;  //  next sleep time from now
//Added for clustering
  int CH_flag;
  double energy_sensor;
```

B. Code for Cluster Head Election

```
void S-MAC::handleSYNC(Packet *p)
{
if ( selfConfigFlag_ == 1) {
  if(numSched_ == 0) {  // I do not have any schedule
mhGene_.cancel();
setMySched(p); // Setting its schedule to that of the received one
return;
}
if (numNeighb_ == 0) {  // My neighbor list is empty so this is
// my first sync pkt
        setMySched(p);
return;
}
}
```

```
// void S-MAC::handleSYNC(Packet *p) .. (contd).
state_ = IDLE;
struct S-MAC_sync_frame *sf = (struct S-MAC_sync_frame *)p->
access(hdr_mac::offset_);
int i, j;
int foundNeighb = 0;
int schedId = S-MAC_MAX_NUM_SCHEDULES;

for(i = 0; i < numNeighb_; i++) {
if (neighbList_[i].nodeId == sf->srcAddr) { // check if sender is on
foundNeighb = 1; //  my neighbor list
schedId = neighbList_[i].schedId;   // a known neighbor
if (sf->CH_flag ==1){ // Check if cluster head has sent the SYNC
mhCounter_[schedId]->sched(sf->sleepTime);}
break;
}
if (neighbList_[i].nodeId == sf->syncNode){ // found its synchronizer,
// remember it schedule id
schedId = neighbList_[i].schedId;
   }
if (!foundNeighb) {  // unknown node, add it onto neighbor list
neighbList_[numNeighb_].nodeId = sf->srcAddr;
if (schedId < S-MAC_MAX_NUM_SCHEDULES) {
  neighbList_[numNeighb_].schedId = schedId; // found its synchronizer
} else if (sf->syncNode == index_) { // this node follows my schedule
neighbList_[numNeighb_].schedId = 0;
} else {  // its synchronizer is unknown
```

```
int foundSched = 0; // check if its schedule equals to an existing one
for (j = 0; j < numSched_; j++) {
double t = mhCounter_[j]->timeToSleep();
double st = sf->sleepTime;
if (t == st || (t + CLKTICK2SEC(1)) == st ||
t == (st + CLKTICK2SEC(1))) {
neighbList_[numNeighb_].schedId = j;
foundSched = 1;
break;
}
}
if (!foundSched) {  // this is unknown schedule
schedTab_[numSched_].txSync = 1;
schedTab_[numSched_].txData = 0;
schedTab_[numSched_].numPeriods = 0;
neighbList_[numNeighb_].schedId = numSched_;
mhCounter_[numSched_]->sched(sf->sleepTime);
numSched_++;
 }
}
 numNeighb_++;   // increment number of neighbors
  }
number_neighb[index_]=numNeighb_;
a[index_] = netif_->node()->energy_model()->energy();
// since energy instances are overwritten everytime this loop is
// entered, we store all the energy values of nodes in an array
// a[20] declared in the top
```

```
cluster_head = 0;
if(var == 0)  // To make the index_ to be CH only once
{
 cluster_head = index_;
 var++;
}

  for (i=0; i<5 ; i++)
{
 for (j=0; j<5; j++)
  {
    if (i != j)
    {
if (number_neighb[i] >= number_neighb[j])  // Check which one has more
    // neighbors
{
if (number_neighb[i] == number_neighb[j])  // If both have same
   // no. of neighbors
{
 if (number_neighb[cluster_head] <= number_neighb[i])
  {
     if (a[i] < a[j]) // Compare energy values

        {
    cluster_head = j;
        }
```

```
   }
}
if (number_neighb[cluster_head] < number_neighb[i])
{cluster_head=i;}
 }
}
       }
             }
    if (index_ == cluster_head)
    {
      cluster_ = 1;
{
 sendSYNC(cluster_);
}
      clusterhead[index_]=1;  //updating CH info

     }
else
{
  cluster_ = 0;
  clusterhead[index_]=0;

if(var2 == 0) // To send the SYNC only once
{
 var2++;
 sendSYNC(cluster_);
}
```

```
}
}
```

C. Code for SYNC Packet Formation and Transmission

```
bool S-MAC::sendSYNC(int c)
{
  // construct and send SYNC pkt
  Packet *p = Packet::alloc();
  struct S-MAC_sync_frame *cf = (struct S-MAC_sync_frame *)p->
  access(hdr_mac::offset_);
  struct hdr_cmn *ch = HDR_CMN(p);

  ch->uid() = 0;
  ch->ptype() = PT_S-MAC;
  ch->size() = SIZEOF_S-MAC_SYNCPKT;
  ch->iface() = UNKN_IFACE.value();
  ch->direction() = hdr_cmn::DOWN;
  ch->error() = 0; /* pkt not corrupt to start with */
  cf->length = SIZEOF_S-MAC_SYNCPKT;
  cf->type = SYNC_PKT;
  cf->srcAddr = index_;
// Added for clustering
  cf->energy_sensor = netif_->node()->energy_model()->energy();
  cf->CH_flag = c;
//***
 cf->syncNode = mySyncNode_;
// shld change SYNCPKTTIME to match with the configures durSyncPkt_
```

```
  cf->sleepTime = mhCounter_[0]->timeToSleep() -
  CLKTICK2SEC(SYNCPKTTIME);
  if (cf->sleepTime < 0)
  cf->sleepTime += CLKTICK2SEC(cycleTime_);

  // send SYNC
  if (chkRadio()) {
    transmit(p);
    return 1;
  } else
    return 0;
}
```

D. TCL Script Used for Simulations

```
# ======================================================================
# Define options
# ======================================================================
set opt(chan)           Channel/WirelessChannel
set opt(prop)           Propagation/TwoRayGround
set opt(netif)          Phy/WirelessPhy
set opt(mac)            Mac/S-MAC
#set opt(mac)            Mac/802_11
set opt(ifq)            Queue/DropTail/PriQueue
set opt(ll)             LL
set opt(ant)            Antenna/OmniAntenna
set opt(x)              2000      ;# X dimension of the topography
set opt(y)              2000      ;# Y dimension of the topography
set opt(ifqlen)         1000      ;# max packet in ifq
```

```
set opt(seed)           0.0
set opt(tr)             test.tr    ;# trace file
set opt(adhocRouting)  NOAH ;# routing protocol
set opt(filters)        GradientFilter
set opt(nn)             5      ;# how many nodes are simulated
set opt(stop)           10000  ;# simulation time
set opt(syncFlag)       1  ;# is set to 1 when S-MAC uses
 #sleep-wakeup cycle
set opt(selfConfigFlag) 0   ;# is set to 0 when S-MAC uses
# user configurable schedule start time
set val(initialenergy)  1000
set val(energymodel)    EnergyModel;
set val(radiomodel)     RadioModel    ;
set val(receivepower)   0.281838      ;# Receiving Power
set val(transmitpower)  0.281838      ;# Transmitting Power
set val(idlepower)      0.281838      ;# Idle Power
set val(sleeppower)     2.81838e-4    ;# sleep power
set val(transitionpower) 0.0563676   ;# transition power
set val(transitiontime)  0.005        ;# transition time


# ======================================================================
# Other default settings
# ======================================================================

LL set mindelay_               50us
LL set delay_                  25us
```

```
LL set bandwidth_               0         ;

Agent/Null set sport_           0
Agent/Null set dport_           0

Agent/CBR set sport_            0
Agent/CBR set dport_            0

# unity gain, omni-directional antennas
# set up the antennas to be centered in the node and 1.5 meters above it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
Phy/WirelessPhy set Rb_ 2*1e6
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0

# S-MAC settings
```

```
Mac/S-MAC set syncFlag_ 1        ; # sleep wakeup cycles
#Mac/S-MAC set S-MAC_DUTY_CYCLE 10

# ======================================================================
# Main Program
# ======================================================================
Class TestSuite
Class Test/S-MAC-multi-hop -superclass TestSuite

proc usage {} {
    global argv0
    puts stderr "usage: ns $argv0 <tests>"
    puts "Valid tests: S-MAC-multi-hop"
}

Test/S-MAC-multi-hop instproc init {} {

    global opt
# create simulator instance
    $self instvar ns_
    set ns_ [new Simulator]

# set wireless topography object
    $self instvar wtopo
    set wtopo [new Topography]

# create trace object for ns
```

```
    $self instvar tracefd
    set tracefd [open $opt(tr) w]

    $ns_ trace-all $tracefd

# define topology
    $wtopo load_flatgrid $opt(x) $opt(y)

#
# Create God
#
    $self instvar god_
    set god_ [create-god $opt(nn)]

#
# define how node should be created
#

# node setting

$ns_ node-config -adhocRouting $opt(adhocRouting) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
```

```
                -phyType $opt(netif) \
                -channelType $opt(chan) \
                -topoInstance $wtopo \
                -diffusionFilter $opt(filters) \
                -agentTrace ON \
                -routerTrace OFF \
                -macTrace OFF \
                -energyModel EnergyModel \
                -idlePower 0.21838 \
                -rxPower 0.281838 \
                -txPower 0.281838 \
                -sleepPower 2.81838e-4 \
                -transitionPower 0.0563676 \
                -transitionTime 0.005\
                -initialEnergy 1000
#
#  Create the specified number of nodes [$opt(nn)] and "attach" them
#  to the channel.

    for {set i 0} {$i < $opt(nn) } {incr i} {
        set node_($i) [$ns_ node $i]
        $node_($i) random-motion 0          ;# disable random motion
    }

#
#  Create the topology
#
```

```
$node_(0) set Z_ 0.0
$node_(0) set Y_ 100.0
$node_(0) set X_ 100.0
$node_(1) set Z_ 0.0
$node_(1) set Y_ 200.0
$node_(1) set X_ 200.0
$node_(2) set Z_ 0.0
$node_(2) set Y_ 300.0
$node_(2) set X_ 300.0
$node_(3) set Z_ 0.0
$node_(3) set Y_ 100.0
$node_(3) set X_ 300.0
$node_(4) set Z_ 0.0
$node_(4) set Y_ 300.0
$node_(4) set X_ 100.0
#$node_(5) set Z_ 0.0
#$node_(5) set Y_ 500.0
#$node_(5) set X_ 500.0
#$node_(6) set Z_ 0.0
#$node_(6) set Y_ 600.0
#$node_(6) set X_ 600.0
#$node_(7) set Z_ 0.0
#$node_(7) set Y_ 400.0
#$node_(7) set X_ 600.0
#$node_(8) set Z_ 0.0
#$node_(8) set Y_ 1600.0
```

```
#$node_(8) set X_ 00.0
#$node_(9) set Z_ 0.0
#$node_(9) set Y_ 1800.0
#$node_(9) set X_ 100.0


# ======================================================================
# Set nodes in a circular topology
# ======================================================================
# Nodes Setup in a circle
#set pi 3.1416
#set r 5
#set a [expr {2*$pi*0.111111111}]

#for {set i 1} {$i < $opt(nn) } {incr i} {

#$node_($i) set X_ [ expr { 50 + ($r * $i )}]
#$node_($i) set Y_ 50
#$node_($i) set Z_ 0.0

#$ns_ initial_node_pos $node_($i) 2
#}
#$ns_ initial_node_pos $node_(0) 2


# setup static routing for line of nodes
#define static routing
```

```
set cmd "[$node_(0) set ragent_] routing 5 0 0 1 1 2 1 3 1 4 1"
eval $cmd
set cmd "[$node_(1) set ragent_] routing 5 0 0 1 1 2 2 3 3 4 4"
eval $cmd
set cmd "[$node_(2) set ragent_] routing 5 0 1 1 1 2 2 3 3 4 4"
eval $cmd
set cmd "[$node_(3) set ragent_] routing 5 0 0 1 1 2 2 3 3 4 1"
eval $cmd
set cmd "[$node_(4) set ragent_] routing 5 0 0 1 1 2 2 3 1 4 4"
eval $cmd

#for Multi-hop scenario
#set cmd "[$node_(5) set ragent_] routing
10 0 4 1 4 2 4 3 4 4 4 5 5 6 6 7 6 8 6 9 6"
#eval $cmd
#set cmd "[$node_(6) set ragent_] routing
10 0 5 1 5 2 5 3 5 4 5 5 5 6 6 7 7 8 7 9 7"
#eval $cmd
#set cmd "[$node_(7) set ragent_] routing
10 0 6 1 6 2 6 3 6 4 6 5 6 6 6 7 7 8 8 9 8"
#eval $cmd
#set cmd "[$node_(8) set ragent_] routing
10 0 7 1 7 2 7 3 7 4 7 5 7 6 7 7 7 8 8 9 9"
#eval $cmd
#set cmd "[$node_(9) set ragent_] routing
10 0 8 1 8 2 8 3 8 4 8 5 8 6 8 7 8 8 8 9 9"
#eval $cmd
```

```
#
# Define traffic model
#
puts "Loading connection pattern..."
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(0) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 80
 #Interval is measured in seconds
$cbr_(0) set interval_ 50
$cbr_(0) set maxpkts_ 1000
$cbr_(0) set fid_ 0
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 1000.0 "$cbr_(0) start"
$ns_ at 2000.0 "$cbr_(0) stop"

#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).000000001 "$node_($i) reset";
}
```

```
$ns_ at  $opt(stop).000000001 "puts \"NS EXITING...\" ; $ns_ halt"
}

Test/S-MAC-multi-hop instproc run {} {
    $self instvar ns_
    puts "Starting Simulation..."
    $ns_ run
}

proc runtest {arg} {

    set b [llength $arg]
    if {$b == 1} {
        set test $arg
    } else {
        usage
    }
    set test "S-MAC-multi-hop"
    set t [new Test/$test]
    $t run
}

runtest $argv
```