Versatile Medium Access Control (VMAC) Protocol for Mobile Sensor Networks

by

Vincent Ngo

B.Eng., Ryerson University, 2007

A Thesis

Presented to the School of Graduate Studies at

Ryerson University

in partial fulfillment of the requirements for the degree of

Master of Applied Science

in the Program of Electrical and Computer Engineering

Toronto, Ontario, Canada

Author's Declaration

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Abstract

Versatile Medium Access Control (VMAC) Protocol for Mobile Sensor Networks

© Vincent Ngo 2010

Master of Applied Science
Electrical and Computer Engineering
Ryerson University

In this thesis, the problem of mobility handling in wireless sensor network is introduced with an appreciation for the applications that may be possible once the problem is resolved. Mobility handling is solved with a simple priority backoff technique inspired by novel MAC protocols. To incorporate this technique for stationary and mobile sensor nodes, a hybrid MAC protocol called VMAC is designed with a fixed frame length. VMAC combines the advantages of scheduled-based MAC for energy savings and contention-based MAC for short transmission delays. To exploit network bandwidth, channel reuse is encouraged and is readily integrated into the protocol. To evaluate VMAC and its performance when compared to other MAC protocols, an implementation inside NS-2 is conducted with simulations of various topologies. These topologies vary in hop-count from source to destination and also contention levels. Simulation results show that VMAC with certain frame lengths are suited for selected topologies, but the frame length of one can always provide sufficient performance. The backoff technique is shown to be fair when nodes contend for medium access and it is even resourceful in speeding up hardware address resolution and routing.

Acknowledgement

There are many people I need to thank for their help, support, and encouragement. First, I would like to thank my M.A.Sc. supervisor, Dr. Alagan Anpalagan, for keeping me on track and guiding me throughout the course of this thesis. He is definitely a great supervisor and a very easygoing person that I can approach for help whenever I need, even though his schedule is overly cramped. I would like to thank the committee members for reviewing my thesis within a short notice and their constructive feedbacks which have improved the thesis overall. Next, I have to thank my research group in the WINCORE lab for being great friends, great colleagues, and great people.

Of course, I have to thank my parents and my sisters for their support through the rough times of graduate studies, and their effort to comprehend the stress involved. My parents have done a lot to allow me to focus my time on research and I can never thank them enough, especially my mom. Here is the lyric to a song from Beyond, a Hong Kong band, called Truly Love You which they dedicated to their mother and I would like to dedicate this to my parents:

Beyond - 真的爱你 (Truly Love You) :

*	無法可修飾的一對手	帶出溫暖永遠在背後
	縱使囉嚕始終關注	不動珍惜太內咎
	沉醉於音階她不讚賞	母親的愛卻永未退讓
	決心衝開心中掙扎	親恩終可報答
#	春風化雨暖透我的心	一生眷顧無言地送贈
+	是妳多麼溫馨的目光	教我堅毅望著前路
	叮嚀我跌倒不應放棄	沒法解釋怎可報盡親恩
	愛意寬大是無限	請準我說聲真的愛妳
重唱	*	
	仍記起溫馨的一對手	始終給我照顧未變樣
	理想今天終於等到	分享光輝盼做到
重唱	#, +, #, +, +	

Last but not least, I would like to thank my friends for their encouragement and the great times we had together and hopefully many more great memories to come.

Acronyms

ACK	Acknowledgement
AODV	Ad-Hoc On-Demand Distance Vector
ARP	Address Resolution Protocol
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CBR	Constant Bit Rate
CPU	Central Processing Unit
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear to Send
CW	Contention Window
DCF	Distributed Coordination Function
DSSS	Direct-Sequence Spread-Spectrum
EKF	Extended Kalman Filter
FL	Frame Length
FTP	File Transfer Protocol
G-MAC	Gateway MAC
GPS	Global Positioning System
IP	Internet Protocol
LEACH	Low-Energy Adaptive Clustering Hierarchy
MAC	Medium Access Control
MACA	Multiple Access with Collision Avoidance
MACAW	MACA for Wireless
MANET	Mobile Ad-Hoc Network
MEMS	Micro-Electro-Mechanical System
MMAC	Mobility-Adaptive, Collision-Free MAC
MS-MAC	Mobility-Aware Sensor MAC
MSN	Mobile Sensor Network
NAV	Network Allocation Vector
NS-2	Network Simulator version 2

OSI	Open System Interconnection
OTcl	Object Tool Command Language
PER	Packet Error Rate
RSS	Received Signal Strength
RTS	Request to Send
S-MAC	Sensor MAC
SNR	Signal-to-Noise Ratio
SYNC	Synchronization
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TRAMA	Traffic-Adaptive Medium Access
UDP	User Datagram Protocol
VMAC	Versatile MAC
WSN	Wireless Sensor Network
Z-MAC	Hybrid MAC

List of Figures

Table of Contents

1. Introduction

In the past decade, wireless communication has brought about many new applications. Recently, a lot of excitement has been built up around sensor communication, where sensors not only sense different phenomena but also forward their findings to a remote station. As applications require different features of sensor nodes, supporting a network of them becomes a great challenge.

1.1 Wireless Sensor Networks

A wireless sensor network (WSN) is a special case of ad hoc networks where nodes can be interactive with their surrounding environment in terms of sensing and actuating. Generally, a sensor network has sensor nodes and base stations, but the amount of sensor nodes is a lot more than base stations. Sensor nodes have sensing and wireless communication capabilities, and they are the main data generators or sources in the network. The data is created from the elements sensed and these elements include temperature, light, sound, and motion. Base stations are sinks that collect data from the nodes and they process them to test for occurrences of irregular events. To minimize cost and size, nodes are limited in sensing, computing, communicating, and power (small battery supply) while base stations are powerful computers with connection to a continuous power source.

Research on WSNs has grown rapidly in the past several years and new techniques have been developed for the physical, data link or medium access control (MAC), and network layers. This popularity is the result of technological advancements in micro-electro-mechanical systems (MEMSs) for sensing, microelectronics for computation and communication, and wireless networking techniques for efficient transmission [1]. A WSN consists of a large number of wireless sensor nodes scattered among an area of interest, and are networked together to collaboratively gather data from the environment (or object in the case of monitoring a target). Nodes can be uniformly or non-uniformly distributed depending on the application and Fig. 1.1 below shows an example with uniform distribution and one sink node. Collaborative gathering between adjacent nodes is needed since sensed data from one node is not accurate and must be compared with others, which improves fault tolerance of the system and allows data fusion [2]. This gives the intuition of moving sensors closer to the objects

being monitored to improve the accuracy of collected data. Each sensor routes the sensed data towards the base station by using other sensors as intermediate hops. This multi-hopping method is preferred since it helps minimize the transmission power required, but suffers from increased delay.

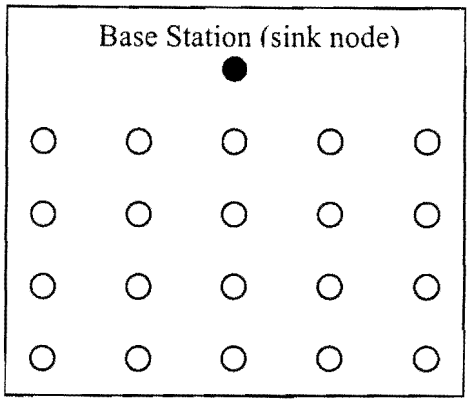

Fig. 1.1 Example of WSN with Uniformly Scattered Sensor Nodes (clear circles)

The primary goal of WSNs is to meet the application needs while maintaining low power consumption, in order to extend network lifetime. Protocols for minimizing power consumption have been developed for the network layer by performing data aggregation at intermediate hops [3], but these protocols assume the sensor nodes are stationary. Current research on WSNs mainly focuses on stationary sensor nodes or controlled mobility for hop-count reduction in data collection [4]. Mobile sensor nodes are needed in applications where sensors are deployed on randomly moving objects for monitoring purposes, such as ZebraNet [5] where zebra positions are tracked, or to maneuver around for discovery purposes. Other applications can involve humans as participants such as flu-virus tracking or air-quality monitoring [6]. Mobility handling in WSNs is an important issue and it should be addressed at different layers of the network, but the focus here is on the MAC layer since it can control and synchronize node radio states for energy efficiency [7]. In general, some challenges affecting the operation of WSNs are self-organization after deployment (i.e. synchronization and scalability), robustness towards topology changes (i.e. node joins and failures), and energy efficiency. These challenges also exist in mobile sensor networks (MSNs) where they are even more difficult to solve.

The radio interface of sensor nodes is designed with three modes or states: receiving (or idling), transmitting, and sleeping (i.e. radio turned off). Commonly, transmitting consumes almost double the amount of power required by receiving, and sleeping uses about a thousand times less power than receiving. Transmitting and receiving spends about tens of milliwatts and sleeping requires only tens of microwatts [8]. Thus, almost all energy-efficient MAC protocols put nodes to sleep during idle periods. Data delivery models are application specific and they can be categorized into periodic, event-driven, and query-driven [9]. In the case of periodic, data is generated once per period so the radio should only be awake at those times, and it can be asleep in between two data generations. In event-driven, data is created when a node senses a particular event, and to delivery it, a path towards the base station or sink is found on demand. Query-driven is similar to event-driven, but instead of sensing an event, a node receives a query and replies to it with the data type specified in the query. Both driven models are random models where data flows are created on demand. For all models, putting nodes to sleep increases the chance of intermediate nodes being asleep when needed and delay is largely dependent on the sleep period. Sleep synchronization in the periodic model is simpler to execute than in the other two and it is crucial in exploiting the sleep mode for energy conservation. The MAC protocols examined here approach this sleeping problem differently.

1.2 Thesis Motivation

Versatile medium access control (VMAC) is a protocol designed to be efficient in both stationary and mobile scenarios by utilizing sleep synchronization, therefore, making it versatile to different application needs. The design is influenced by the current state-of-the-art tactics in WSNs and builds on it by providing mobility handling. As mentioned earlier, considering mobile nodes in the network can broaden the range of applications of sensor nodes. Mobile nodes are conceived as having higher priority in communication than stationary nodes, because their random or controlled movement leave them minimal amount of connection time to other nodes, which may be in motion or not. Current mobility handling techniques can be too cumbersome due to the large amount of control overhead. In addition, some techniques go beyond the responsibilities of the MAC layer, which is shown to be unnecessary by the methodology of VMAC. For testing purposes, VMAC has been implemented in the Network Simulator version 2 (NS-2). To be efficient in the stationary instance, VMAC performance

approaches the MAC protocols of IEEE 802.11 distributed coordination function (DCF) [10] and time division multiple access (TDMA). For the mobile case, VMAC insist on maintaining a performance level close to the stationary case by using a simple yet effective method to handle mobility.

1.3 Contribution and Organization

The main work and contribution of the thesis is summarized below:

- proposed VMAC for stationary and mobile sensor networks through efficient mobility handling with a technique that is resourceful even in the stationary case
- verified the performance of VMAC through implementation in NS-2 and NS-2 simulations with detailed wireless configurations
- showed that various frame lengths of VMAC are effective under certain contention levels and the frame length of 1 merges the advantages of TDMA and 802.11

The rest of the thesis is organised as follows. In Chapter 2, the technical background of wireless networking is briefly discussed and the NS-2 simulator is introduced with focus on wireless and mobile communication. Chapter 3 goes through an extensive list of novel MAC protocols that have inspired VMAC. Chapter 4 is the creation of VMAC through ideas sprung out by analyzing the readings conducted. Chapter 5 provides an elaborate evaluation of VMAC and relates the performances to VMAC methodology. Chapter 6 is the conclusion and possible future work for more in-depth testing of VMAC and foreseeable improvements.

2. Technical Background

To understand the importance of the MAC layer and its functions in sensor networks, the background information is provided here. This chapter goes through the general network layers that comprise the communication system used in contemporary wireless networking, but adds the modifications required by their counterparts in sensor networks. The NS-2 simulator is also explained to comprehend the correctness of its use in testing the performance of VMAC and other protocols. Wireless channel models in the NS-2 distribution are introduced with detailed explanations on received power.

2.1 Network Layers

The open system interconnection (OSI) model divides the responsibilities of a communication network into seven separate layers, and thus, simplifies the implementation of each layer. With responsibilities decomposed, each layer must support the layers above it in order for the system to be functionally correct. Support from lower layers is assumed when each layer is constructed, so the layers depend on each other and work together to form a communication network. Fig. 2.1 shows the OSI model with only five layers since the other two layers, presentation and session, can be combined together with the application layer. Applications, such as web browsing and the file transfer protocol (FTP), are designed with the assumption that communication between two ends is sustained by other layers below. The applications depend on the transport layer to segment and transport the large amount of data from the source to the destination. The user datagram protocol (UDP) [11] does segmentation for applications and uses the Internet protocol (IP) [12] to deliver the segment. UDP is efficient in delivering segments with small processing delay, which is important in mission critical applications in sensor networks. Delay is minimal since acknowledgements (ACKs) are not realized, and reliability and in-order delivery of segments are not guaranteed. These services are provided by the transmission control protocol (TCP), along with flow control to prevent receiver buffer overflow and congestion control to prevent network congestion [13].

TCP provides many services, but they are unnecessary in sensor networks. Sending acknowledgements back to the source may interrupt other data flows to the sink, and the same

data may be sent by multiple sources (due to close spacing of sensors) so missing it is highly unlikely. In addition, the MAC layer may perform ACK making it repetitive and waste of bandwidth if TCP does it as well. Also, reliability and in-order delivery of segments are not needed when data is typically time-stamped and only the most recent ones matter. TCP also depends on IP to deliver the segments, which are called packets at the network layer. IP supplies the upper layers with a best-effort delivery service, because it tries to route the packet to the destination over a number of hops (intermediate routers or nodes) and the packet can be dropped at any hop due to congestion or some admission policy.

In networks where the medium (wired or wireless) is shared, a MAC protocol is used to schedule the access time of each user. IP passes the packet to the MAC layer, where it is called a frame and is stored in a buffer until it gets scheduled to access the medium. Upon accessing, the physical layer modulates the bits of the frame using a technique that is efficient for the medium in terms of energy and bandwidth, and it does this before the frame is sent out to the next hop towards the destination. Additionally, transmit power can be adjusted to reduce probability of errors or to save energy, depending on the channel condition.

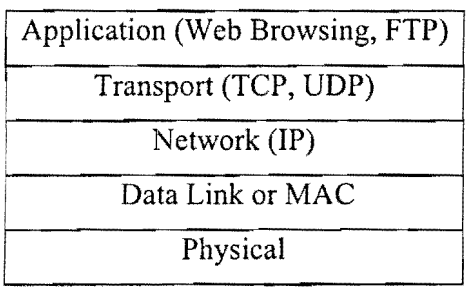

Fig. 2.1 Simplified OSI Model

In traditional networks, the TCP/IP network stack is sufficient but in resource-constrained sensor networks a routing architecture based on unique IP addresses is not practical. Even routing in ad-hoc networks is still address-centric, where specific node identities are the main concerns rather than what data each node carries. The objective of sensor networks is the timely delivery of sensed data to the destination, so it can be categorized as being data-centric [2]. Thus, the source that generated the data is not important and similar data can be aggregated by routers along the route to the destination. The node itself only becomes

vital when the application requires information from nodes in a certain location, in which case this becomes location-based routing [7]. Design of the routing protocol can be crossed over with the MAC protocol to optimize the overall performance. The MAC protocol used in ad-hoc networks is typically IEEE 802.11, and this protocol has several features that may not be favoured in sensor networks. These include sending acknowledgements of received packets and non-periodic (non-scheduled) data transmission which prevents sleep synchronization.

2.2 Network Simulator 2 (NS-2)

NS-2 [14] is an object-oriented, discrete-event simulator for testing protocols by manipulating and recording the properties of packets according to the network configuration (the protocols and settings involved). The status of the packets, whether sent, received, or dropped, are traced and outputted to a trace file for analysis conducted later. The network could be wired or wireless (local or satellite) or both (wired-cum-wireless), and there is even support for mobility in ad-hoc networks. Nodes inside the defined network topology can establish connections and disconnect with each other at any moment. The network settings cover all layers of networking and there are multiple choices for each layer, which are still growing as research continues to bring about new protocols and architectures. Although NS-2 is a powerful tool, development work such as implementing VMAC is difficult and reusing code in real-life systems is impossible. However, the currently under development and expansion NS-3 covers these issues and much more, but since it is still in its beta stage it is not used here.

NS-2 is written in C++ and simulations are written in OTcl (Object Tool Command Language) scripts that act as an interpreter frontend for the simulator by calling NS-2 functions. The point of using two languages is to have a fast simulation run-time and allow quick variations to the network configuration since these objectives are important to research. C++ executes rapidly but requires a lot of time to alter settings. OTcl is slow in execution but can be easily and quickly changed for different tests. Thus, the combination of the two is perfect and also provides a separation between two different purposes. For simulations with the current functions provided, it is best to program with OTcl. For developing new protocols, C++ must be utilized for writing new objects and functions that can be called upon by OTcl scripts. The C++ and OTcl codes written for VMAC are provided in the Appendix. In the following

sections, the mobile networking features and protocols in NS-2 will be discussed in detail. The information is mainly from Chapter 16 and 18 of the NS manual [15].

2.2.1 Mobile Networking: Features

For mobility support, a mobile node is constructed as child of a normal node in C++ and it has additional functionalities for movement, and transmission/reception on a wireless channel that can be configured according to different simulation environments. However, the channel models implemented only considers path loss and does not take into account the signal-to-noise ratio (SNR) or Doppler shift that are significant in mobile scenarios. These parameters determine the packet error rate (PER), or more specifically, the bit error rate (BER). Thus, Raviraj et al. [16] have developed and implemented a physical layer model for wireless channels in NS-2 that includes the effects of SNR and Doppler shift. Also, with physical models that affect bit error rate, channel coding and modulation can be included and tested for their effectiveness. Currently, binary phase shift keying (BPSK) is the only modulation implemented in NS-2.

Energy consumption is an important aspect in protocol design so the energy model in NS-2 logs the current energy level of mobile hosts especially after a transmission or reception. Each node is initiated with an energy level value and energy usage is dependent on the power setting for transmission and reception of packets. However, energy consumed during central processing unit (CPU) usage for computations is not considered and this is important to accurately calculate the energy required by complex functions, such as mobility prediction for mobile ad-hoc networks (MANets) and WSNs. If the energy model can take into account the energy savings of putting the CPU to sleep, then protocols can utilize this in their simulations and comparisons between their results become more practical. The following is a general overview of the options and tools available for mobile nodes:

Link Layer (LL): The LL is the same as the one used for normal nodes, but it is attached with an address resolution protocol (ARP) module that resolves hardware addresses when given a destination IP address. This is required since links for wireless networks are not defined, so not enough information is known about the receiving node for actual transmission.

Address Resolution Protocol (ARP): The ARP is implemented in BSD (Berkeley software distribution) style and its responsibility is just to resolve the hardware address of the destination when queried by the LL.

Interface Queue: Several network interface queues can be chosen for different purposes such as fair queuing and first-in first-out (FIFO). One queuing scheme that is important for smooth routing protocol execution in MANets is the priority queue (PriQueue). The PriQueue places routing protocol packets at the head of the line, which shortens the delay for resolving a route and minimizes the packet loss due to disconnection. However, this has been tested to be problematic in NS-2 since ARP needs to resolve the hardware addresses needed by the route first.

MAC Layer: Traditionally, the protocol for wireless MAC is IEEE 802.11 DCF. Now, in the current release NS-2.33, several other 802.11 implementations are available.

Network Interface: The network interface is a hardware interface used to determine access to the wireless channel. The transmitter interface receives all packets to be transmitted and stamps each one with information regarding the transmission power, wavelength, and etc. The receiver interface uses the propagation model to determine if the minimum power for reception (receiving threshold) or detection (carrier-sensing threshold) is met. The model is an approximation of the direct-sequence spread-spectrum (DSSS) radio interface from Lucent WaveLan.

Radio Propagation Model: The Friss free-space model ($1/r^2$) and the two-ray ground reflection ($1/r^4$) model for attenuation are used extensively in NS-2 for near distances and far distances, respectively.

Antenna: Currently, only omni-directional antennas are supported by NS-2. Other implementations of directional antennas exist outside of the NS-2 distribution and they require more testing.

2.2.2 Mobile Networking: MAC Protocols

Two types of MAC protocols for mobile networking are available in NS-2 and they are IEEE 802.11 DCF and TDMA. IEEE 802.11 DCF is a traditional contention-based MAC and its implementation includes both physical and virtual carrier sensing. Furthermore, unicast packets use the RTS/CTS/DATA/ACK pattern while broadcast packets are sent without any RTS/CTS exchange. Although these components are integrated, the 802.11 implementation in NS-2 is far beyond the actual one governed by the standards. Thus, in the current NS-2.33 release, 802.11 has been extended to incorporate a lot of these missing features. The TDMA MAC protocol is preamble-based (scheduled-based) and it is still in the preliminary stage, so contention during the preamble slot and time slot reuse in a multi-hop scenario are not supported. Instead, the protocol provides a global TDMA schedule by assigning a permanent time slot to each node. The preamble slot is used to mimic contention where source nodes announce their destination node IDs, but no actual information exchange through packet transmission takes place. All nodes listen during this preamble and they record the time slots of when they are suppose to receive the packets. Nodes that are not receiving or transmitting in certain time slots can go to sleep, but must be awake when they are suppose to send or receive packets. Collisions are non-existent since nodes only transmit on their own permanently reserved time slot.

2.2.3 Mobile Networking: Routing Protocols

There are four different ad-hoc routing protocols implemented in NS-2 for mobile networking. The first one is based on proactive routing where periodic route discovery is performed to store routes before they are needed. The other three are based on reactive routing where on-demand route discovery is used when the route is required. They are briefly summarized below:

Destination-Sequenced Distance Vector (DSDV): This protocol is table-driven, so all exchanged routing information are kept even when the routes are never used. Routing updates are done periodically unless a change in the routing table advertised by a node triggers the update. There is traffic overhead even when the network topology has not changed. When a route to the destination is unknown, the packet is cached in a buffer and query for the route is sent out. Once the buffer is full, incoming packets are dropped. The destination replies to the query and sets a sequence number for the route.

Dynamic Source Routing (DSR): The protocol checks the header of every data packet for route information inserted by the source. Forwarding is based on the next hop indicated by the source, but if that information is not there then intermediate nodes can provide the source route if it is known or send a query for it. Routing queries are broadcasted and replies can be sent by intermediate nodes or the destination if the route is known. The complete route is kept in the header of the data packet as it traverses the network and this is substantial when the path is long.

Temporally Ordered Routing Algorithm (TORA): The protocol is based on a link reversal algorithm and it is similar to DSR. When a packet needs a route, a QUERY is broadcasted and it traverses the network until the destination or an intermediate node that knows the route is reached. The recipient of the query replies with a broadcasted UPDATE message that indicates its height (distance) from the destination. The distance information is incremented as it traverses the network and the source node will have several paths to the destination. These paths are directed links that begin at the source and end at the destination. Unreachable destinations have a defined maximum height and network partitions are announced using a broadcasted CLEAR message. TORA runs on top of the Internet MANet encapsulation protocol (IMEP), which provides reliable delivery of routing messages and notifies TORA of changes made to links.

Ad-hoc On-Demand Distance Vector (AODV): This protocol combines DSR and DSDV since it implements route discovery and route maintenance of DSR, and performs hop-by-hop routing with sequence numbers and beacons of DSDV. When a packet needs a route, a ROUTE REQUEST is sent and intermediate nodes forward the request until it reaches a node that knows the route to the destination. The request is replied with a ROUTE REPLY that contains the hop count, and nodes forwarding the reply increments the hop count and records the forward path to the destination. The next hop node is kept for each destination and not the entire route.

Another routing protocol worth stating is Directed Diffusion [3] and it is a protocol that does not run on top of IP, but instead replaces it. It is designed for WSNs and applicable for periodic data gathering. It differs from IP since sensed data is important to the application and not the individual node addresses. Thus, Directed Diffusion is data-centric and sinks send out interest messages to request for data in specific areas of the network. Sensors in those areas respond when the data generated from sensing (attribute-value pair) matches these interests. The data flows towards the sinks and intermediate nodes can aggregate similar attribute-value pairs to save energy.

2.2.4 Wireless Channel Models

There are three propagation models included in the NS-2 distribution and they are the Friis free-space model, two-ray ground reflection model, and shadowing model. These models are taken directly from Rappaport's textbook on wireless communication [17]. The models predict the received signal power of each packet. A receiving threshold is set during the physical layer configuration in OTcl simulation, and packets are dropped if their received power is below this threshold. The free-space model assumes ideal propagation so it only considers the line-of-sight path between the transmitter and receiver. According to the model, the following equation is the received signal power at a distance, d, from the transmitter:

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L} \tag{1}$$

where P_t is the transmitted signal power, G_t is the transmitter antenna gain, G_r is the receiver antenna gain, λ is the wavelength, and L is the system loss (usually assumed to be 1). The free-space model is only accurate for short distances and requires a different model for long distances. The two-ray ground reflection model considers both the line-of-sight and ground reflection paths so it is more accurate than the free-space model. The reflection model can only be used when the distance is beyond a certain threshold and is not as accurate as free-space for small distances. The received signal power according to this two-ray ground reflection model is as follows:

$$P_r(d) = \frac{P_t G_t G_r h_t^2 h_r^2}{d^4 L} \tag{2}$$

where h_t is the transmitter height and h_r is the receiver height. Equation (2) does not require the wavelength, because at far distances wavelength does not affect attenuation. The cross-over distance separates the free-space and ground reflection models, and helps to determine the better model for the current transmitter/receiver separation. This distance is the result of equation (1) being equal to equation (2) and it can be calculated with the following equation:

$$d_c = \frac{4\pi h_t h_r}{\lambda} \tag{3}$$

The above two models predict from a deterministic function the received signal power and they represent the transmission range as an ideal circle. The received power becomes a random variable at certain distances because multi-path propagation effects create this randomness. The equation that considers such probabilistic phenomena is the shadowing model and it is simplified and shown below:

$$\left[\frac{P_r(d)}{P_r(d_O)}\right]_{dB} = -10\beta \log\left(\frac{d}{d_O}\right) + X_{dB} \tag{4}$$

where d_O is a reference distance at which the received power is known, X_{dB} is a Gaussian random variable with zero mean and standard deviation, σ_{dB}. β and σ_{dB} are tabulated values and depend on the selected wireless environment (indoor or outdoor).

After an elaborate background of the network layers and the simulator involved, it is now critical to review novel MAC protocols for sensor networks. Chapter 3 is a literature survey that covers MAC protocols exploiting the nature of stationary sensor networks to save energy. More interestingly, protocols that attempted to handle mobility are built upon these techniques and their insights are identified.

3. State-of-the-art Literature Review

The review on preceding protocols has provided new ideas based on the successes and failures of each protocol. Some of these protocols have been referenced extensively and others have not gained enough recognition, but they are all important and are briefly introduced here.

A popular contention-based scheme for sensor networks is Sensor-MAC (S-MAC) [8]. S-MAC is an energy-efficient protocol that tries to save energy by periodically coordinating sensor nodes to sleep for most of the time, and wake up only for a short duty cycle to listen for transmissions (i.e. the nodes idle until they receive packets) or to transmit sensed data. The two states, listen and sleep, form a frame time which is usually 1 s (second). Nodes form a virtual cluster by synchronizing the start and end of their listen and sleep schedules using broadcasted synchronization (SYNC) packets. Border nodes in between two clusters can communicate with either cluster, as long as the border nodes have knowledge of both schedules. To prevent network partitions when nodes miss each other due to non-overlapping listen intervals, sensor nodes leave 10 s after every 2 min (minute) of periodic listening and sleeping for neighbour discovery or after a short period if the sensor node has no neighbours. The same method is used to synchronize with new sensor nodes joining the network or from other clusters (i.e. border nodes). This means a mobile node moving into a new cluster can wait up to 2 min to connect with nodes in that cluster.

To handle mobility while keeping the energy benefits of putting nodes to sleep, the mobility-aware sensor MAC (MS-MAC) protocol [18] adaptively changes the synchronization period to be 10 s after every 30 s of listening and sleeping when mobility is detected. MS-MAC is an extension to S-MAC made by other researchers [18] in hope of providing better mobility handling. Detection of mobility is based on the received signal level of SYNC packets from a mobile node. An active zone is created around the mobile node and border nodes also form active zones with nodes around them. All nodes inside the zones are within two hops from the mobile or the border nodes, and they perform synchronization for 10 seconds after every 30 seconds of listen and sleep cycles. As the mobile moves, the active zone moves along with it and nodes within two hops of it join the zone as the mobile node approaches. Distance (in hop

count) from the mobile and border node IDs are stored in SYNC packets so nodes can assume detection of mobility when they are within two hops.

Mobility increases frame errors due to low SNR or high Doppler shift. Low SNR and high Doppler shift increase BER, which in turn increases the number of frame errors. Thus, another approach to handle mobility is to use extended Kalman filter (EKF) [16] to make the frame size smaller when the channel condition is harsh, and larger when the channel has high SNR and low Doppler shift. This method is based on adaptively estimating the frame size to be used for transmission, and requires knowledge of previous frame size and channel conditions. On the other hand, mobility has been considered as an improvement to channel capacity, but instead of assuming random mobility or predicting it, mobility is controlled for data delivery to sinks [4]. Another prediction approach similar to S-MAC with EKF is the mobility-adaptive, collision-free MAC (MMAC) protocol [19] which adaptively changes the frame time according to the dynamic change in mobility. The protocol is scheduled-based or TDMA based, so there is no collision. Instead of using a fixed frame time like S-MAC or MS-MAC, MMAC shortens the frame time when mobility is high and lengthens it when mobility is low. The protocol is an extension to the traffic-adaptive medium access (TRAMA) protocol [20], which uses a hash table to provide distributed scheduling and utilizes a fixed frame time. Mobility is estimated using a probabilistic autoregressive model and it produces a prediction on the mobility of two-hop neighbours.

The following sections will be focused on the analysis of each approach and comparison of their effectiveness in handling mobility, whether the technique is changing the rate of synchronization, frame size, or frame time. Effectiveness will be based on the time required to connect with a mobile node and the scalability of the technique in handling many mobile nodes or even a network of entirely mobile nodes. Also, the complexity or protocol overhead of each approach will be compared, such as computation time and memory requirements. When appropriate, possible mobility handling techniques are proposed and the trends set for future MAC protocols will be discussed.

3.1 Contention-based MAC Protocols

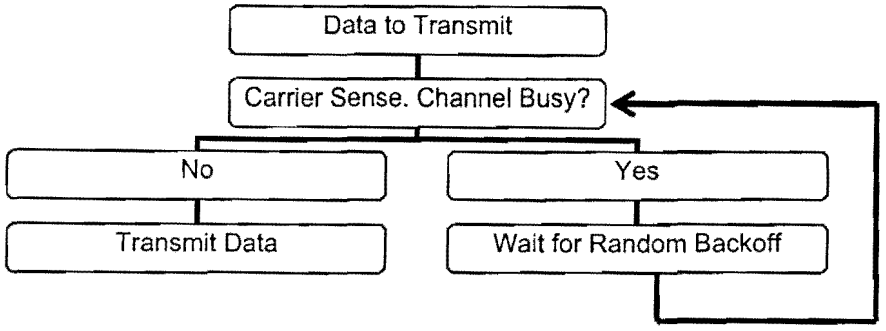

Fig. 3.1 Traditional Contention-based MAC Flowchart

In a single-channel wireless network, there are two predominant types of MAC protocols and they are contention-based and scheduled-based. In contention-based (see Fig. 3.1 above), nodes contend for the channel whenever they have data to transmit, so carrier sense is usually used to avoid collisions by ensuring the channel is free before transmitting. To support fairness and prevent collisions once the channel is free, a random backoff period is initiated when the channel is busy so contending nodes wait for a random amount of time before executing carrier sensing again. An example of such a MAC protocol is IEEE 802.11, which is one of the most commercially used multiple access schemes. IEEE 802.11 differs from Fig. 3.1 since it performs a RTS/CTS exchange before transmitting the data and it supports virtual carrier sensing using RTS/CTS control packets.

3.1.1 Sensor-MAC (S-MAC)

The goal of S-MAC [8] is to minimize energy usage for prolonging network lifetime of a stationary network. The four major sources of energy consumption are: collision that occurs in a contention-based MAC, overhearing other transmissions, control packet overhead, and idle listening. S-MAC is a contention-based protocol that is similar to the IEEE 802.11 DCF protocol, so it utilizes carrier sensing and RTS/CTS to prevent collisions. Carrier sensing is done both physically within a fixed contention window and virtually by inserting a network allocation vector (NAV) value in each RTS and CTS. The NAV value is the transmission time for one data packet with a corresponding ACK. The RTS/CTS exchange is first developed for minimizing the cost of hidden terminal problems, and it is based on the carrier sense multiple

access with collision avoidance (CSMA/CA) protocol. The *hidden terminal problem* occurs when there are two transmitters separated by two hops (such as Node 1 and 3 in Fig. 3.2a), and there is a node in between them that is the receiver for one of them (Node 2 is the receiver for Node 1 while Node 4 is the receiver for Node 3). Both RTS transmissions will overlap at the receiver (Node 2) and cause co-channel interference, but with RTS the cost of this loss is far less than if a data packet is transmitted directly. Multiple access with collision avoidance (MACA) further used RTS/CTS to prevent the exposed terminal problem and it also added virtual carrier sensing. The *exposed terminal problem* happens when two adjacent transmitters try to transmit to their corresponding receivers (such as Node 2 transmitting to Node 1 and Node 3 transmitting to Node 4 as shown in Fig. 3.2b) and their transmissions do not overlap at the receivers, but one (such as Node 3) is prevented to reuse the channel and transmit since the other (Node 2) is already transmitting. To solve the exposed terminal problem, if Node 3 has received a RTS from Node 2 but not a corresponding CTS, then it is two hops from the receiver and will not interfere so it can transmit as well.

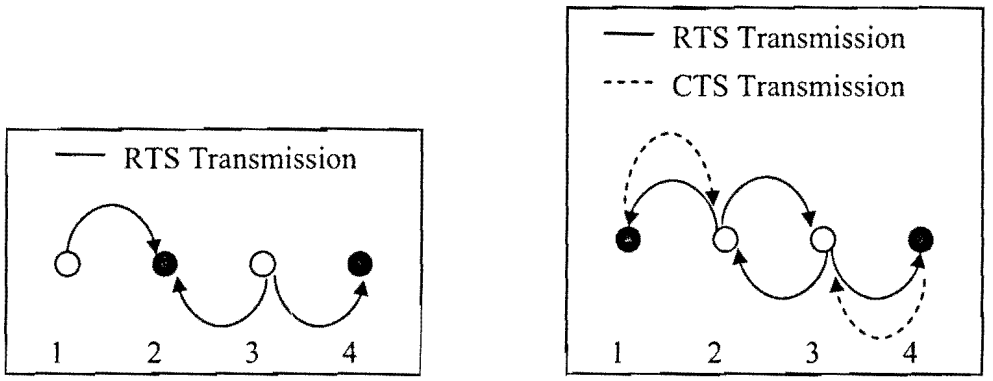

Fig. 3.2 a) Hidden Terminal Problem, and b) Exposed Terminal Problem

In S-MAC, unlike unicast packets, broadcast packets like SYNC do not utilize RTS/CTS. Even though DCF is one of the best in avoiding collisions in a contention-based environment, collisions still occur especially when the contention window is fixed. Fixing the window has the advantage of simplifying the computation involved and thus leads to energy savings. Also, the fixed window, adopted from the analysis done by MACA for wireless (MACAW) [21], promotes some amount of fairness for medium access since all nodes have an

equal opportunity unlike varying contention windows, where nodes with failed transmissions have to wait even longer.

Overhearing is receiving data packets that are destined for other nodes and this is an energy waste in support of virtual carrier-sensing, because nodes listen to all transmissions to find the transmission end times. Continuous listening to other transmissions is unnecessary when the reception of the RTS or CTS packet indicates the duration of the transmission in the NAV, so a node receiving either of them can go to sleep until the end of transmission. For now, assume the node wakes up at the next listen interval. When sleeping, the transceiver of the node is turned off and the CPU is kept on. The CPU would basically keep track of the sleep timer and wakes the node when the timer expires. A possible improvement to this can be assigning the task to a specific hardware and allow the CPU to sleep as well. Although interference occurs at the receiver and receiving a RTS but not a CTS means a node can still transmit (the MACA approach for avoiding the exposed node problem), receiving a RTS puts a node into sleep mode since the node may interfere the ACK packet destined for the sender or the sender's reception of CTS. When a node fails to capture the medium using physical carrier-sensing, it also sleeps until the next listen interval begins. The duty cycle (ratio between the listen period and the frame time) can be configured from 1% to 99%.

Minimizing control packet overhead is done by synchronizing the coordinated sleeping of nodes in a distributive and local manner. Nodes choose and maintain their schedules by following an algorithm and keeping schedules of neighbours in a table. When deployed, each node listens for the duration of the synchronization period of 10 s for SYNC packets. If none is found, then it chooses its schedule and begins to follow the periodic listen and sleep cycle. During the first listen interval, it announces its schedule by placing its address and next sleep time (a relative value rather than absolute) in a SYNC packet. A node receiving a SYNC packet before announcing its schedule will follow the received schedule, and announce the schedule at the next listen time. This way the first schedule in the network will propagate through the network. A node receiving a SYNC packet after announcing its schedule can do one of two things: if the node has no neighbours, it will follow the new schedule received or else, it will adopt both schedules and wake up on both listen intervals by becoming a border node.

Nodes following the same schedule form a virtual cluster and border nodes have two schedules so two different virtual clusters may communicate. Virtual clusters may overlap in physical space but cannot communicate unless their listen intervals overlap as well or border nodes link them. This method drains the batteries of border nodes faster than others since they sleep less in order to keep the network together. Also, border nodes may fail to adopt two schedules if the SYNC packet from another virtual cluster is corrupted, by collisions or interference, or delayed by the wait for medium access. Furthermore, border nodes may be asleep while nodes from other clusters try to synchronize with them. To prevent network partitions due to failure of border nodes in adopting two schedules, a neighbour discovery period, which is the synchronization period, is executed every 2 min when a node has at least one neighbour and every 30 s when it has no neighbours. Neighbour discovery puts nodes in listen mode and forces them to announce their schedules with SYNC packets, so energy consumption is higher and during this time data transmission is stopped. Synchronization also prevents clock drift, which is a small value when compared to the listen period.

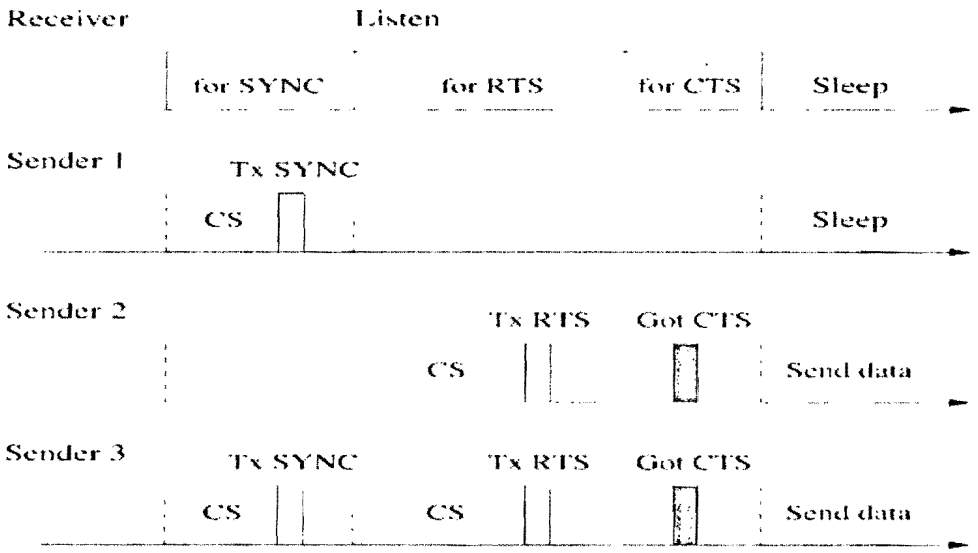

Fig. 3.3 S-MAC Possible Sender and Receiver Timing Relationships (source: [8])

Idle listening is reduced greatly by coordinating the sleep time of nodes and waking them up only when necessary. Fig. 3.3 shows the complete listen and sleep cycle with three possible timing relationships between sender and receiver. During the listen interval, if there is data to be sent, a sender can announce its schedule and try to access the medium using physical

Nodes following the same schedule form a virtual cluster and border nodes have two schedules so two different virtual clusters may communicate. Virtual clusters may overlap in physical space but cannot communicate unless their listen intervals overlap as well or border nodes link them. This method drains the batteries of border nodes faster than others since they sleep less in order to keep the network together. Also, border nodes may fail to adopt two schedules if the SYNC packet from another virtual cluster is corrupted, by collisions or interference, or delayed by the wait for medium access. Furthermore, border nodes may be asleep while nodes from other clusters try to synchronize with them. To prevent network partitions due to failure of border nodes in adopting two schedules, a neighbour discovery period, which is the synchronization period, is executed every 2 min when a node has at least one neighbour and every 30 s when it has no neighbours. Neighbour discovery puts nodes in listen mode and forces them to announce their schedules with SYNC packets, so energy consumption is higher and during this time data transmission is stopped. Synchronization also prevents clock drift, which is a small value when compared to the listen period.

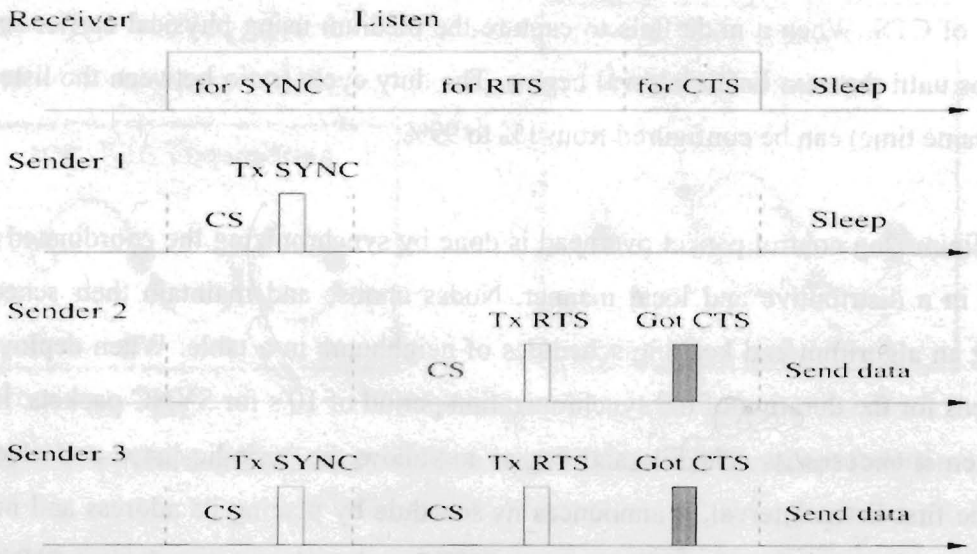

Fig. 3.3 S-MAC Possible Sender and Receiver Timing Relationships (source: [8])

Idle listening is reduced greatly by coordinating the sleep time of nodes and waking them up only when necessary. Fig. 3.3 shows the complete listen and sleep cycle with three possible timing relationships between sender and receiver. During the listen interval, if there is data to be sent, a sender can announce its schedule and try to access the medium using physical

carrier sensing and RTS/CTS exchange. If the medium is captured, then data transmission occurs during the sleep interval, or else, the node sleeps until the next listen interval. This poses the problem of a long delay to pass data across multiple hops, because each listen and sleep cycle can only move the data one hop closer to a sink (base station), or not at all if the medium is busy. To speed up the hopping, an adaptive listen period is incorporated where a node that hears a RTS or CTS stays awake for a short period at the end of the transmission. With this method, neighbours of current sender and receiver assume the possibility of being the next-hop after the transmission. For adaptive listen, the SYNC interval in the second listen interval is not included since nodes only synchronize during the first listen interval. If a node does not hear a packet for it after the adaptive listen period ends, then it sleeps until the next regular listen interval. The sleep interval should be long enough for several data transmissions to happen continuously, but most likely only two data transmissions can occur since nodes that are two hops away from the first receiver cannot hear its CTS and will be asleep when they are needed during adaptive listen. Thus, this method shortens the delay by half when compared with S-MAC without adaptive listen.

Looking at the aspect of mobility handling, S-MAC has not proposed anything specific in this category since it is designed for applications over stationary nodes. To handle mobility, a single mobile moving across virtual clusters must somehow discover that it is entering a new cluster and needs to proactively synchronize with the nodes inside. Assuming a node knows its speed (can be easily achieved using a global positioning system (GPS) receiver) and the topology consist of several clusters with each one in a specific area, so when the node speed is above a certain threshold it will try to synchronize with others since it might have moved out of the current cluster.

Proactive synchronization would be fast, that is within a frame time (e.g. 1 s), if the mobile continuously broadcast a PROSYNC packet during the frame time and nodes receiving the PROSYNC respond with their schedules using SYNC. If the schedules received are the same as the mobile, then it is still within the cluster, or else, it could have entered a new cluster or is located between two clusters. This method can quickly synchronize a single mobile, but with multiple mobiles the SYNC interval must be longer (maybe adaptively extended when

more and more mobiles are detected or fixed for a certain node density), or else the RTS/CTS interval will be interfered and delay will accumulate. The assumption is practical when nodes are deployed in separate areas at different times, but this method should only be effective when a certain percentage of nodes are mobile and others are stationary for the main purpose of sharing the schedule. If the whole network is one big virtual cluster, then synchronization would not be a problem but mobility may cause adaptive listen to fail.

The memory requirement of S-MAC is proportional to n, the number of neighbours, since a schedule table entry is kept for each neighbour. The schedule table is built locally without any central controller so synchronization is simplified. Neighbour discovery prevents network partitions and adaptive listen reduces packet delivery delay. The overall protocol is simple and energy efficient since it does not require any rigorous computations, but its ease fails to handle mobile nodes. A mobile moving across two virtual clusters can wait up to 2 min to obtain the new schedule and during this time it is disconnected from the network.

3.1.2 Mobility-aware Sensor MAC (MS-MAC)

MS-MAC [18] is one of the first protocol to handle mobility in WSNs and it extends the S-MAC protocol. The goal of the protocol is to be energy-efficient in both stationary and mobile cases, so nodes sleep more when they are stationary and sleep less when they are mobile. When stationary, the protocol is the same as S-MAC, but when mobility is detected, nodes go directly into synchronization or neighbour discovery. MS-MAC identifies that in S-MAC a node with no neighbour increases its frequency of synchronization, but the rest of the network stays the same. A fast moving mobile node will most likely lose all its current neighbours and this may lead to high data loss, because it is disconnected from the network. Thus, groups of nodes around the mobile perform neighbour discovery after every 30 s of listen and sleep cycles when a mobile is detected. The indication of mobility is based on the received signal strength (RSS) of the SYNC packet, which is not an accurate estimation but the results from simulations show that it can support one mobile. To assist the mobile in capturing a new schedule when it enters a virtual cluster, border nodes and nodes two hops away from them must always form an active zone (see Fig. 3.4). In this zone, neighbour discovery is 10 s for every 30 s of listen and sleep. For nodes to discover their distance from the mobile, the SYNC packet now carries a hop count

relative to the mobile. While the mobile is inside the new cluster, nodes two hops away from the mobile must also form an active zone. If the signal level of the received SYNC packet from a node is below a certain threshold for example, then it is declared as a mobile.

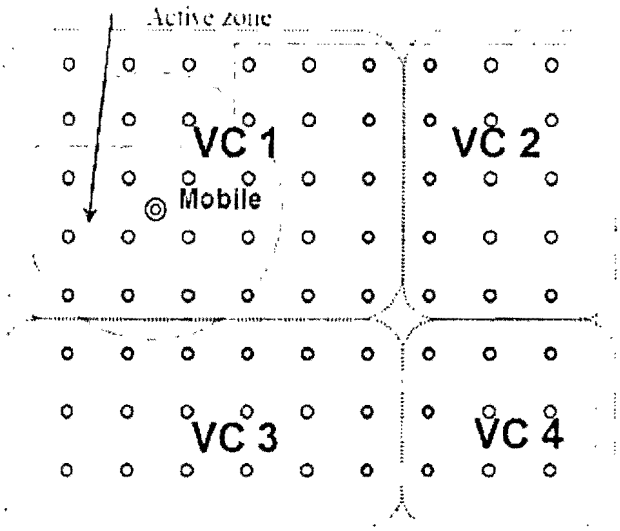

Fig. 3.4 MS-MAC Active Zones formed for Four Virtual Clusters (source: [18])

The MS-MAC simulation results in NS-2 are compared with S-MAC and they show that the protocols have similar average energy consumption when the mobile speed is 5 m/s (see Fig. 3.5) and delay when the mobile speed varies from 0 to 20 m/s (see Fig. 3.6). When considering the packet drop rate, the performances differ and, at some points between 0 and 20 m/s (see Fig. 3.7), MS-MAC has nearly half the drop rate of S-MAC. The simulations ran for only the case of a single mobile and under the scenario of multiple mobiles there will be many active zones being formed. In this case, energy consumption can be significant. Also, border nodes always stay active so they will most likely empty their battery supplies first. The advantage of the signal level being the indicator of mobility is that no extra energy is used. The additional use of the SYNC packet in passing distance information is simple and does not add any major complexity to the protocol. In terms of memory requirements, schedule table size is still proportional to n, the number of neighbours.

For handling mobility, this protocol assumes most nodes are stationary and some nodes are mobiles that traverse the network to collect data from stationary nodes. The function of

relative to the mobile. While the mobile is inside the new cluster, nodes two hops away from the mobile must also form an active zone. If the signal level of the received SYNC packet from a node is below a certain threshold for example, then it is declared as a mobile.

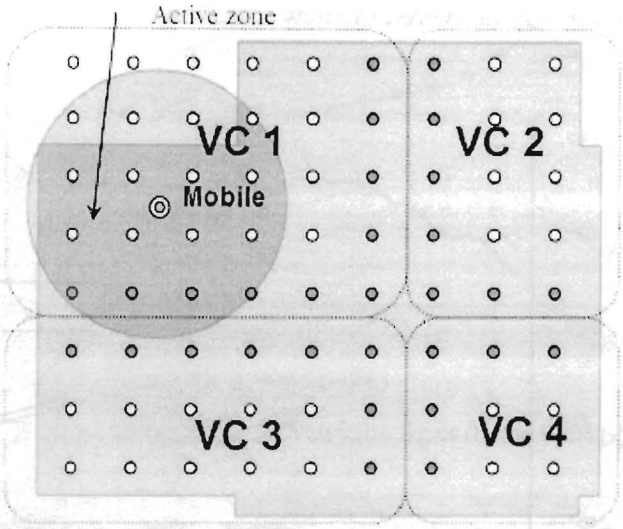

Fig. 3.4 MS-MAC Active Zones formed for Four Virtual Clusters (source: [18])

The MS-MAC simulation results in NS-2 are compared with S-MAC and they show that the protocols have similar average energy consumption when the mobile speed is 5 m/s (see Fig. 3.5) and delay when the mobile speed varies from 0 to 20 m/s (see Fig. 3.6). When considering the packet drop rate, the performances differ and, at some points between 0 and 20 m/s (see Fig. 3.7), MS-MAC has nearly half the drop rate of S-MAC. The simulations ran for only the case of a single mobile and under the scenario of multiple mobiles there will be many active zones being formed. In this case, energy consumption can be significant. Also, border nodes always stay active so they will most likely empty their battery supplies first. The advantage of the signal level being the indicator of mobility is that no extra energy is used. The additional use of the SYNC packet in passing distance information is simple and does not add any major complexity to the protocol. In terms of memory requirements, schedule table size is still proportional to n, the number of neighbours.

For handling mobility, this protocol assumes most nodes are stationary and some nodes are mobiles that traverse the network to collect data from stationary nodes. The function of

mobiles here is in data collection, which reduces hopping to sinks. Even though synchronization rate is increased, mobiles can still wait up to 30 s to capture a new schedule when entering a cluster. Thus, packet drop rate can be large when multiple mobiles exist and are disconnected from the network. Also, the zone around the mobile is needless since border nodes would have already delivered the new schedule.

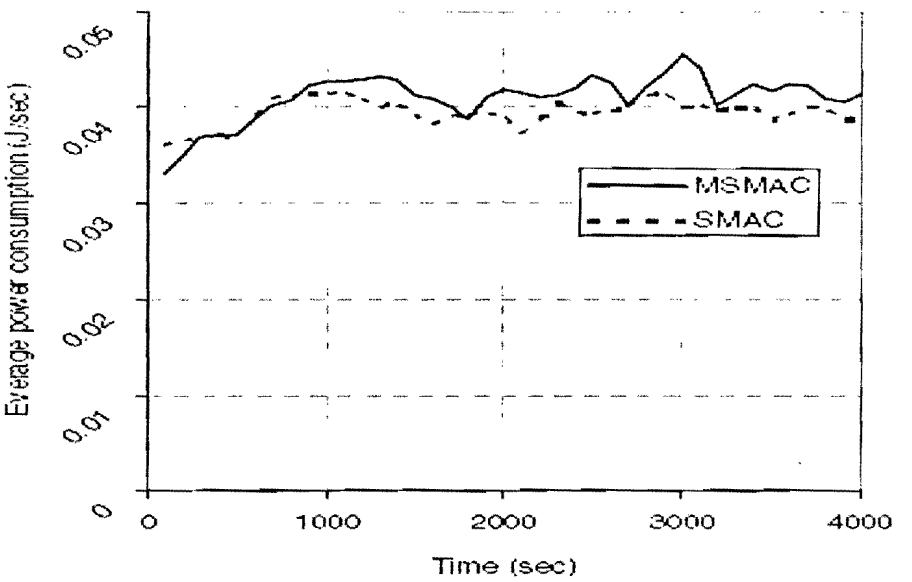

Fig. 3.5 MS-MAC Average Energy Consumption at 5 m/s (source: [18])

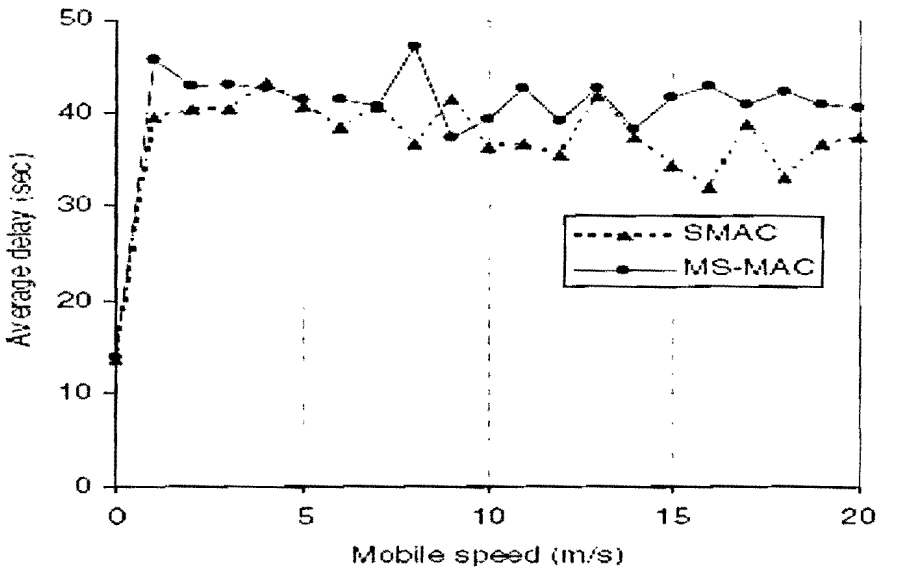

Fig. 3.6 MS-MAC Average End-to-End Delay at Various Speeds (source: [18])

mobiles here is in data collection, which reduces hopping to sinks. Even though synchronization rate is increased, mobiles can still wait up to 30 s to capture a new schedule when entering a cluster. Thus, packet drop rate can be large when multiple mobiles exist and are disconnected from the network. Also, the zone around the mobile is needless since border nodes would have already delivered the new schedule.

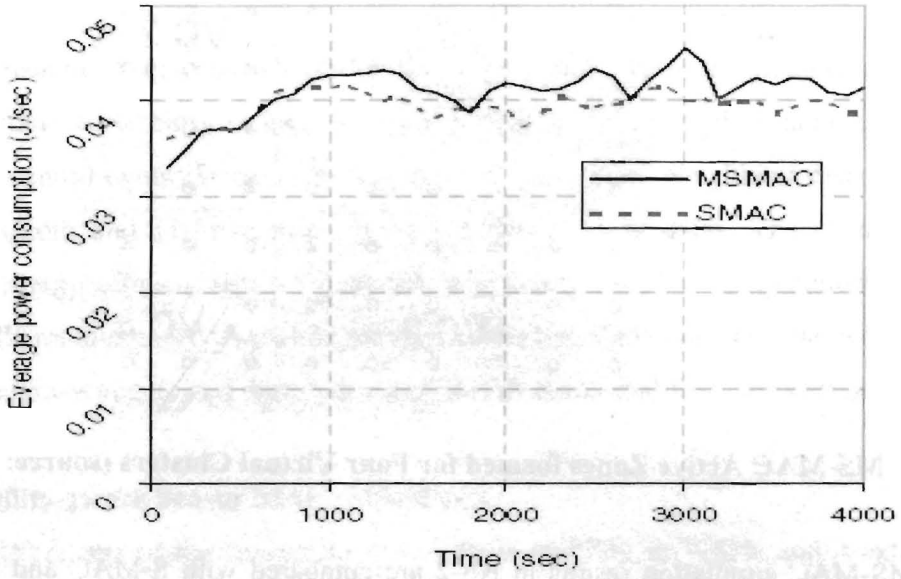

Fig. 3.5 MS-MAC Average Energy Consumption at 5 m/s (source: [18])

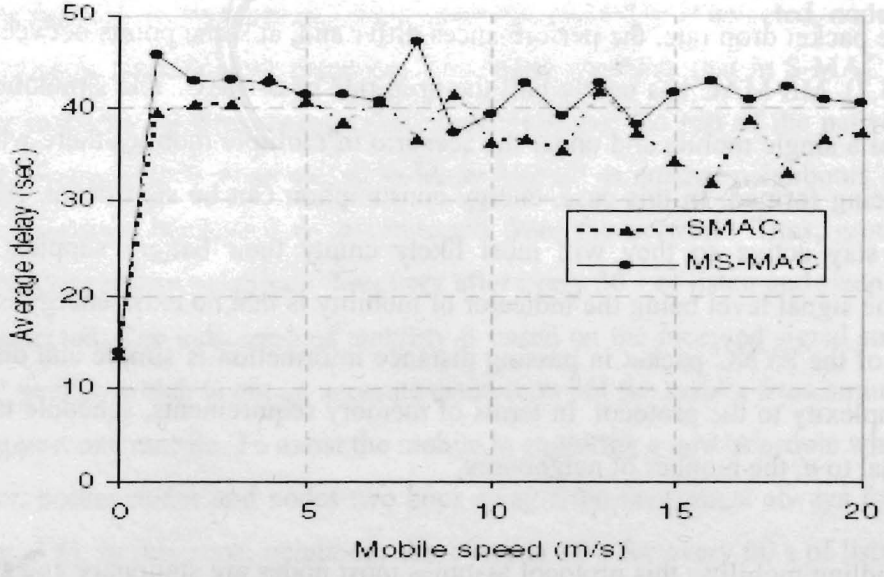

Fig. 3.6 MS-MAC Average End-to-End Delay at Various Speeds (source: [18])

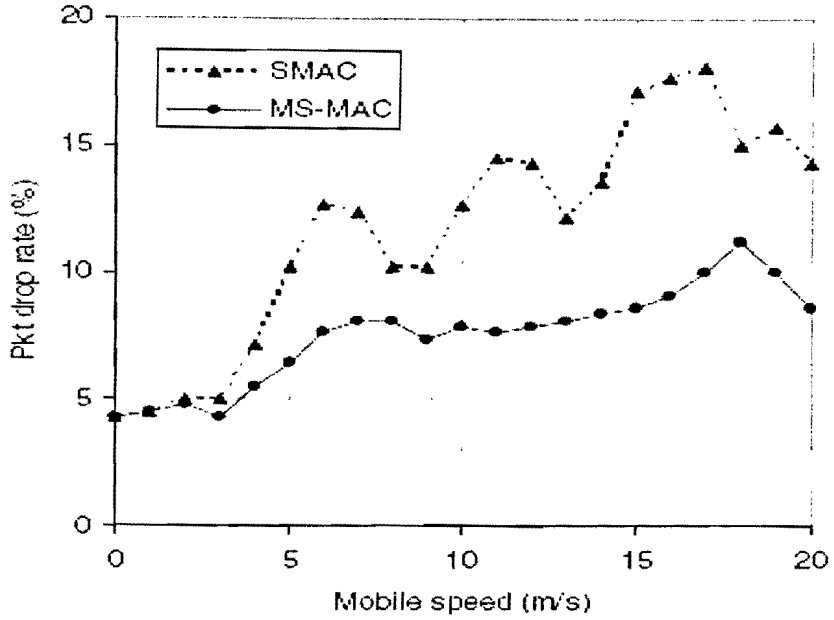

Fig. 3.7 MS-MAC Packet Drop Rate at Various Speeds (source: [18])

3.1.3 S-MAC with Extended Kalman Filter (EKF)

The EKF is used with the S-MAC protocol [16] to predict the frame size of every data transmission according to the previous frame size and channel condition, and the current channel condition (i.e. SNR and Doppler shift). The idea is to minimize the number of frame losses (retransmissions) caused by Doppler shifts when mobility exist. MS-MAC fails to consider frame losses due to bit errors, since NS-2 assumes correct frame reception when received power is above the receiver and carrier-sense thresholds. In other words, only path loss is considered in NS-2 and simulations conducted in NS-2 must incorporate bit errors (and therefore frame losses) due to non-ideal channel conditions. Thus, the NS-2 physical layer is replaced by an accurate Mica-2 (a sensor node developed by University of California, Berkeley) physical layer. The BERs for various SNRs and relative velocities (velocity between transmitter and receiver) are calculated from simulations in MATLAB using the transceiver parameters of Mica-2. These BERs are used as a lookup table in the NS-2 physical layer when simulations are conducted for the Mica-2 layer. The BER is used to calculate the packet error-free reception probability with Manchester coding, since this coding scheme is used by Mica-2 and the equation for this calculation is available. One minus the packet error-free reception probability is the PER, and if the PER is above a certain threshold then the frame is dropped.

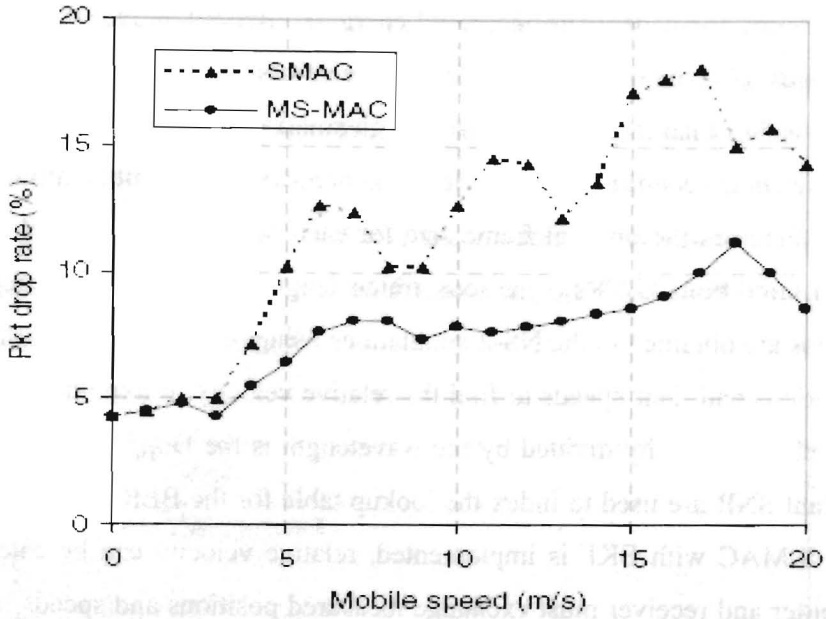

Fig. 3.7 MS-MAC Packet Drop Rate at Various Speeds (source: [18])

3.1.3 S-MAC with Extended Kalman Filter (EKF)

The EKF is used with the S-MAC protocol [16] to predict the frame size of every data transmission according to the previous frame size and channel condition, and the current channel condition (i.e. SNR and Doppler shift). The idea is to minimize the number of frame losses (retransmissions) caused by Doppler shifts when mobility exist. MS-MAC fails to consider frame losses due to bit errors, since NS-2 assumes correct frame reception when received power is above the receiver and carrier-sense thresholds. In other words, only path loss is considered in NS-2 and simulations conducted in NS-2 must incorporate bit errors (and therefore frame losses) due to non-ideal channel conditions. Thus, the NS-2 physical layer is replaced by an accurate Mica-2 (a sensor node developed by University of California, Berkeley) physical layer. The BERs for various SNRs and relative velocities (velocity between transmitter and receiver) are calculated from simulations in MATLAB using the transceiver parameters of Mica-2. These BERs are used as a lookup table in the NS-2 physical layer when simulations are conducted for the Mica-2 layer. The BER is used to calculate the packet error-free reception probability with Manchester coding, since this coding scheme is used by Mica-2 and the equation for this calculation is available. One minus the packet error-free reception probability is the PER, and if the PER is above a certain threshold then the frame is dropped.

Reducing frame losses leads to increased energy conservation and smaller frame sizes have two advantages: (1) cost of losing smaller frames is less than larger frames, and (2) PER in smaller frames is less than larger frames. Although small frames are beneficial, using small frames when the channel condition is good leads to bandwidth underutilization. The EKF is distributive and calculates the optimal frame size for each transmission using the current and previous channel conditions (BERs), previous frame length, and overall protocol overhead. Channel conditions are obtained in the NS-2 simulations using the knowledge of the transmitter and receiver positions and their speeds to find the relative velocity. Maximum Doppler shift is assumed so the relative velocity divided by the wavelength is the Doppler shift. This Doppler shift and a constant SNR are used to index the lookup table for the BER and, finally, calculate the PER. When S-MAC with EKF is implemented, relative velocity can be calculated using GPS, but transmitter and receiver must exchange measured positions and speeds. Also, instead of BER, the EKF equation must be modified to account for realistic channel gain, which can be obtained using methods such as training sequences and this requires sensor nodes to have a digital signal processor (DSP).

The simulations involve five nodes and one of them is a base station node that can communicate with others directly (multi-hopping is not simulated). In all of the simulations, the Mica-2 physical layer is used so in the case of low SNR or RSS (see Fig. 3.8), normal S-MAC consumes a lot of power since it does not adapt to channel condition by using a smaller frame size, which leads to many frame losses. However, the lowest frame length is not mentioned and this is important since S-MAC uses the RTS/CTS exchange so its control packet can easily exceed the data portion. For high mobility (see Fig. 3.9 and 3.10), S-MAC with EKF conserves energy by accurately setting the frame length and achieves about 24% in energy savings when compared to regular S-MAC at average node speed of 75 mph. Although S-MAC with EKF is a great solution to handle high-speed mobile scenarios, the EKF is not a protocol and its algorithm will be better if position and speed information exchange is explained in detail. Such information can be carried in the header of the RTS/CTS exchange when GPS is incorporated, which simplifies the protocol overhead.

The memory requirements are more than S-MAC and MS-MAC since channel conditions, frame size, and protocol overhead are needed. Computation time of the optimal frame size is not mentioned in the paper and this can cause an extended delay, but the equation is straight forward and should be implemented by a application specific integrated circuit (ASIC). Most importantly, only a single cluster is considered so synchronization across clusters is not analyzed at all. Thus, S-MAC with EKF is similar to S-MAC and should require up to 2 min to synchronize with a new cluster.

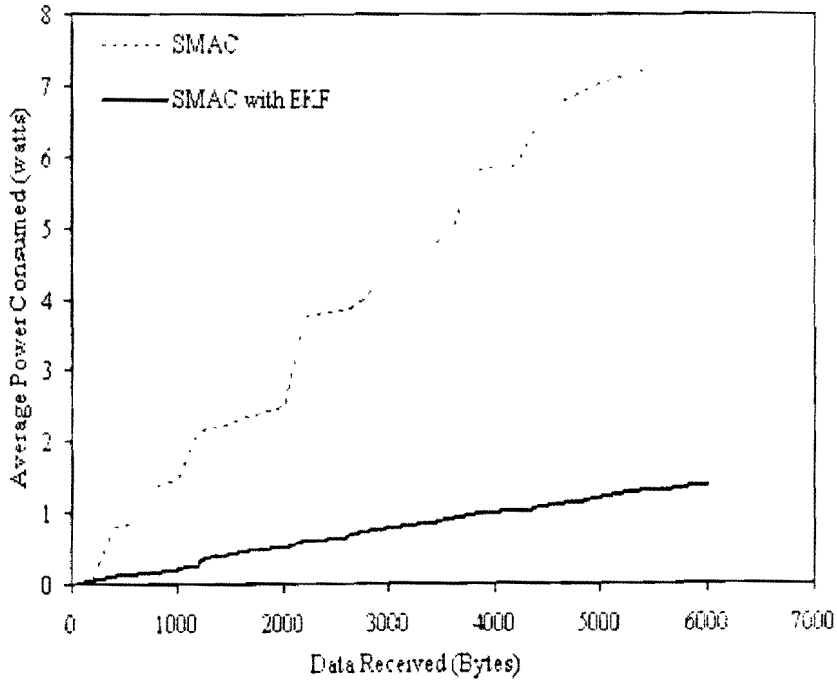

Fig. 3.8 SMAC with EKF Average Power Consumed at Low RSS (source: [16])

The memory requirements are more than S-MAC and MS-MAC since channel conditions, frame size, and protocol overhead are needed. Computation time of the optimal frame size is not mentioned in the paper and this can cause an extended delay, but the equation is straight forward and should be implemented by a application specific integrated circuit (ASIC). Most importantly, only a single cluster is considered so synchronization across clusters is not analyzed at all. Thus, S-MAC with EKF is similar to S-MAC and should require up to 2 min to synchronize with a new cluster.

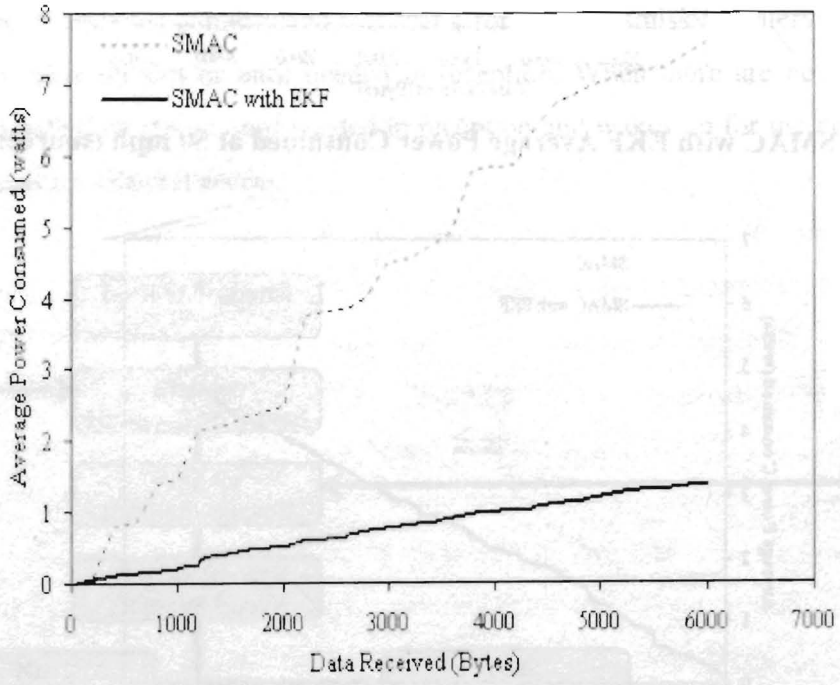

Fig. 3.8 SMAC with EKF Average Power Consumed at Low RSS (source: [16])

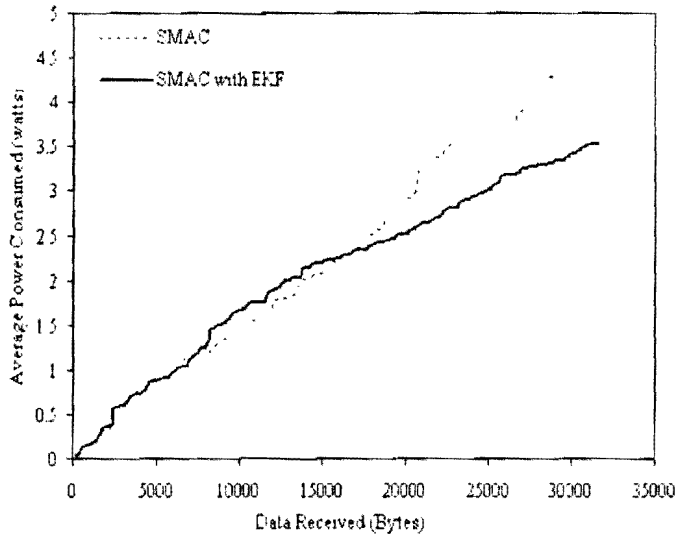

Fig. 3.9 SMAC with EKF Average Power Consumed at 50 mph (source: [16])

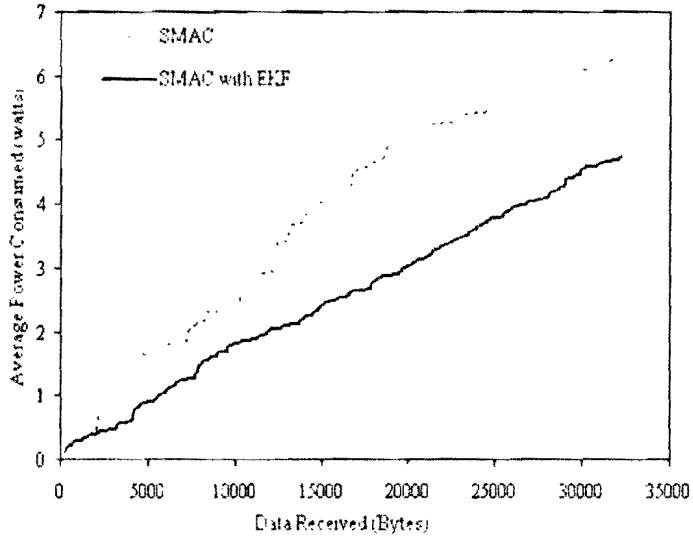

Fig. 3.10 SMAC with EKF Average Power Consumed at 75 mph (source: [16])

3.2 Scheduled-based MAC Protocols

In scheduled-based, time is slotted for each transmission and transmitters reserve slots if they have a packet to send while receivers are notified for their participation in the corresponding time slot (see Fig. 3.11). Reservation and transmission intervals form a frame and this frame time/length can be fixed or varied. In the case of central scheduling, a central scheduler collects all time slot allocation requests and distributes the final schedule back to the transmitters and

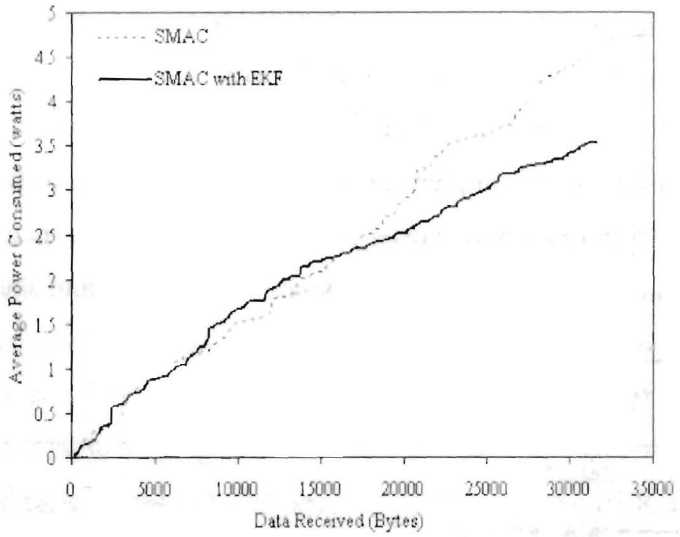

Fig. 3.9 SMAC with EKF Average Power Consumed at 50 mph (source: [16])

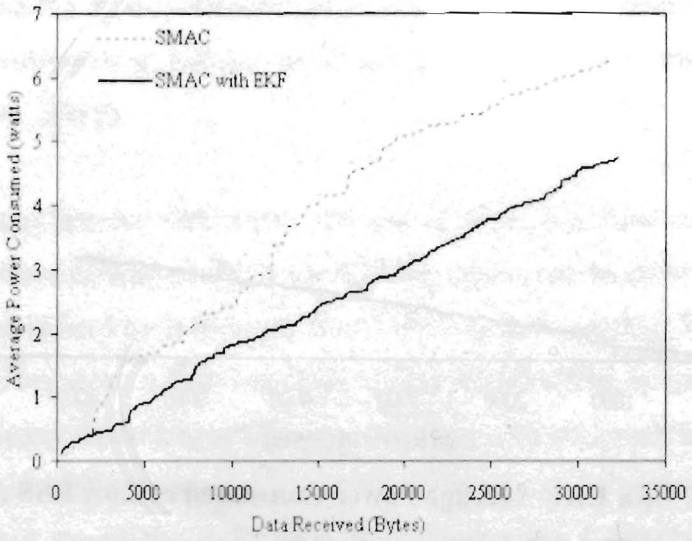

Fig. 3.10 SMAC with EKF Average Power Consumed at 75 mph (source: [16])

3.2 Scheduled-based MAC Protocols

In scheduled-based, time is slotted for each transmission and transmitters reserve slots if they have a packet to send while receivers are notified for their participation in the corresponding time slot (see Fig. 3.11). Reservation and transmission intervals form a frame and this frame time/length can be fixed or varied. In the case of central scheduling, a central scheduler collects all time slot allocation requests and distributes the final schedule back to the transmitters and

receivers. Nodes that are not participating in the transmissions can go to sleep until the next round of schedule distribution. During the collection interval, contention-based (random-access) MAC is typically used when nodes are not assigned fixed time slots for sending requests, and this is beneficial in maximizing the use of available time slots. In the case of distributed scheduling, nodes schedule themselves based on the reservations they received within their two-hop neighbourhoods. This is because carrier-sensing range is twice the communication range so only one node transmits within a two-hop radius. VMAC is based on distributed scheduling and its method is discussed in Chapter 4. Once a node successfully reserves a slot or receives a notification, it waits for the transmission interval to begin and sleeps until its reserved slot or until needed in reception. When there are no more available slots, the node with data sleeps until needed in reception and wakes up for the next reservation period to contend for channel access.

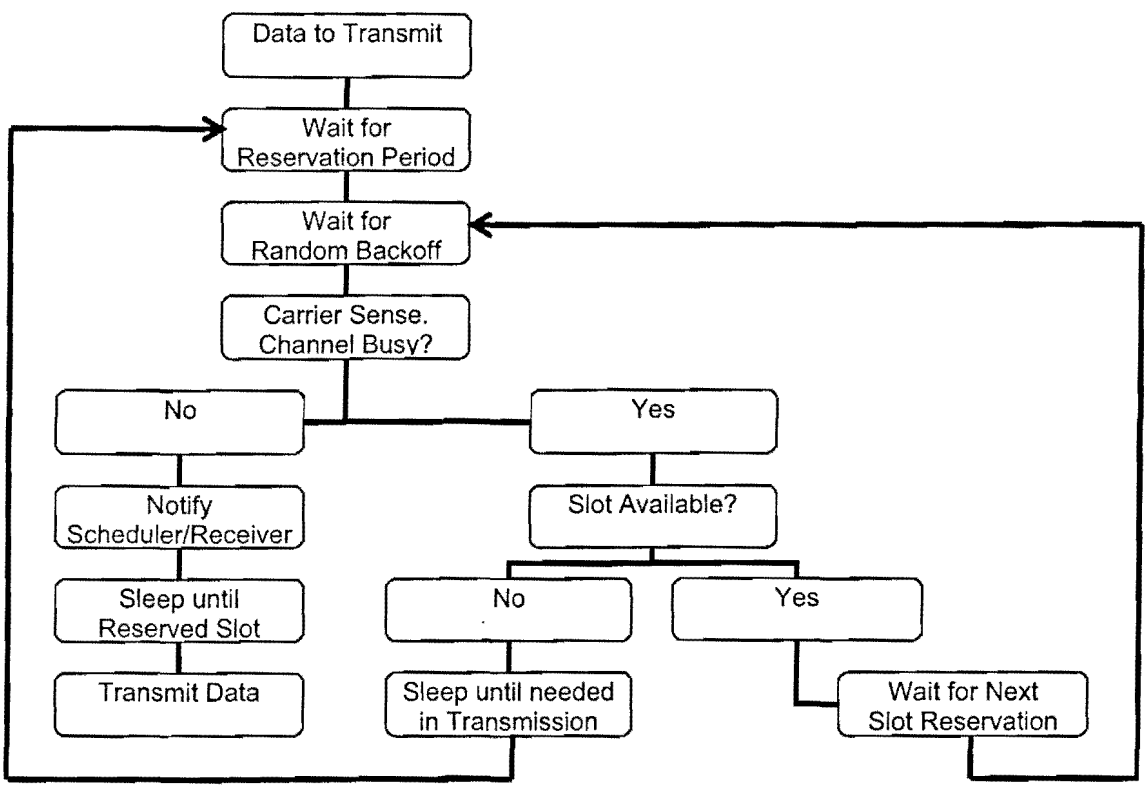

Fig. 3.11 Scheduled-based MAC Flowchart

receivers. Nodes that are not participating in the transmissions can go to sleep until the next round of schedule distribution. During the collection interval, contention-based (random-access) MAC is typically used when nodes are not assigned fixed time slots for sending requests, and this is beneficial in maximizing the use of available time slots. In the case of distributed scheduling, nodes schedule themselves based on the reservations they received within their two-hop neighbourhoods. This is because carrier-sensing range is twice the communication range so only one node transmits within a two-hop radius. VMAC is based on distributed scheduling and its method is discussed in Chapter 4. Once a node successfully reserves a slot or receives a notification, it waits for the transmission interval to begin and sleeps until its reserved slot or until needed in reception. When there are no more available slots, the node with data sleeps until needed in reception and wakes up for the next reservation period to contend for channel access.

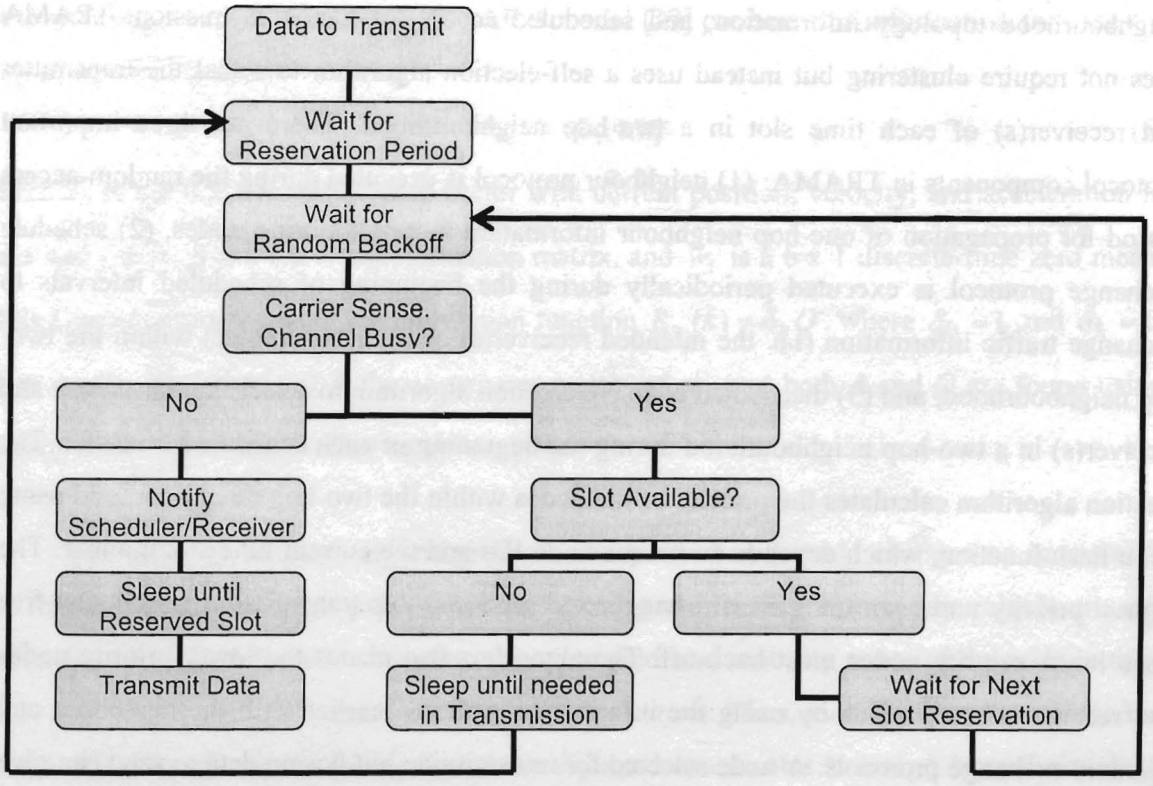

Fig. 3.11 Scheduled-based MAC Flowchart

For large wireless networks that use central scheduling, clustering is usually utilized for a simpler and less-hopping request collection procedure. Cluster heads are the schedulers and nodes within a cluster send their request to the cluster head. The cluster head schedules the transmissions based on the requests and distributes the time slot assignment to the cluster members. Packets that are destined for the sink must either hop through the cluster heads which form the second tier of nodes or transmitted directly to the sink using long-range radio. Generally, techniques based on clustering do not explain communication between cluster heads in detail, and cluster formation and interference between clusters are vague.

3.2.1 Mobility-adaptive, collision-free MAC (MMAC)

MMAC [19] is a mobility-adaptive, collision-free protocol since scheduled access during data transmission guarantees this absence of collisions. MMAC follows the design of TRAMA [20] where a frame time is separated into two parts: random access for signalling of two-hop neighbourhood topology information, and scheduled access for data transmission. TRAMA does not require clustering but instead uses a self-election algorithm to select the transmitter and receiver(s) of each time slot in a two-hop neighbourhood. There are three important protocol components in TRAMA: (1) neighbour protocol is executed during the random-access period for propagation of one-hop neighbour information to neighbouring nodes, (2) schedule exchange protocol is executed periodically during the beginning of scheduled intervals to exchange traffic information (i.e. the intended receiver(s) of each transmitter) within the two-hop neighbourhood, and (3) distributed adaptive election algorithm to select the transmitter and receiver(s) in a two-hop neighbourhood during the beginning of each scheduled-time slot. The election algorithm calculates the priority of the nodes within the two-hop neighbourhood using a fair hash function, which depends on unique node IDs and the current time slot number. The highest priority node can transmit if it has data to send and this transmission is collision-free since lower priority nodes must back-off. The algorithm also places the lower priority nodes into receive or sleep mode by using the information gathered earlier with the neighbour and schedule exchange protocols. A node selected for transmission but has no data to send can give up its time slot by announcing this to others.

For mobility handling, MMAC uses a dynamic (adaptive) frame time based on the predicted mobility of two-hop neighbours. Mobility is separated into two types: (1) weak mobility which is topology changes caused by node joins and failures, and (2) strong mobility which consists of concurrent node joins and failures, and physical mobility. Fixed frame time, such as that used in TRAMA, in mobile environments is described to have three disadvantages: (1) mobile nodes must wait for a long period for connectivity in a new neighbourhood, (2) packet collisions increase dramatically in a contention-based MAC scheme, and (3) two-hop neighbourhood information is inaccurate during this mobility period in a scheduled-based MAC scheme. To overcome these drawbacks, the dynamic frame time is set inversely proportional to the level of predicted two-hop neighbourhood mobility. An autoregressive (AR) model [19] is used to predict the mobility of the two-hop neighbours, but it requires current node position, velocity, and acceleration at time t, which can be assessed when each node is integrated with a GPS receiver. Mobility estimation requires this information in their corresponding x and y coordinates. The AR-1 model [19] predicts this information for time $t + 1$ by using this equation:

$$s_{t+1} = As_t + w_t \tag{5}$$

where s_t is the 6 x 1 mobility state vector with current position, velocity, and acceleration in the x and y axis, A is a 6 x 6 transformation matrix, and w_t is a 6 x 1 discrete-time zero mean, white Gaussian process with autocorrelation function $R_w(k) = \delta_k Q$, where $\delta_0 = 1$ and $\delta_k = 0$ when $k \neq 0$. The matrix Q is the covariance matrix of w_t and both A and Q are found using training data in the Yule-Walker equations [19].

The mobility-adaptive algorithm is distributive and each node changes its next frame time based on the number of expected nodes entering and leaving its two-hop neighbourhood. First, each node uses the AR-1 model to calculate predicted mobility states of the node in time $t, t + 1, t + 2, \ldots$, up to $t + max$, where $max = frame\ time$. Thus, mobility states are predicted for the entire frame time beforehand, and the expected coordinates (x and y) from the average of these states is used in calculating the expected number of nodes entering and leaving. Nodes entering during the next frame time are not added to the two-hop neighbour list, while those that are leaving are removed from the list. The exact times when nodes leave and enter are

uncertain, so it is best to exclude them from the list during the next frame time, and packets to those mobile nodes are dropped. If the number of entering and leaving nodes is above a maximum threshold value, then the new frame time is the original frame time minus by a percentage of the original, where the percentage can vary depending on, for example, the number of nodes in the network. On the other hand, if the number falls below a minimum threshold value, then the original frame time is increased by a certain percentage. Both the scheduled access and random access slots are increased or decreased proportionally when the frame time changes.

There are several problems associated with the mobility-adaptive algorithm since each node requires the future mobility states of the whole network (current and potential two-hop neighbours), and individually calculated frame times can be different leading to synchronization errors. To solve these two issues, clustering is introduced and the cluster head selection/rotation mechanism is a variation of the low-energy adaptive clustering hierarchy (LEACH) protocol [22]. At the end of each round, where a round is several frames, all nodes send their expected coordinates in the next frame to the cluster head. After collecting all this information, the cluster head broadcasts the expected coordinates to all nodes in the cluster using a BROADCAST message. Each member node calculates the new frame time individually and sends it to the cluster head. All member frame times are collected by their cluster heads and the average frame times are calculated and sent to a single second level head. The period when all of this occurs is called the global synchronization period (GSP). The second level head calculates the average of the average frame times and disseminates it to the entire network. During the frames when GSP does not occur, mobility is adapted by varying the number of scheduled access and random access slots according to the BROADCAST message sent by the cluster head, while keeping the frame time constant. More mobility means less scheduled access and more random access and vice versa, so nodes have more chances to synchronize.

The simulation results of MMAC from NS-2 are compared with TRAMA, S-MAC, and regular carrier sense multiple access (CSMA). The simulation scenario is a 500 m (meter) x 500 m plane with mobile nodes having a transmission range of 100 m and sinks being in one corner only. Average node speed is varied from 0.1 m/s to 1 m/s. From the simulation results

(see Fig. 3.12), MMAC has better average delay performance than TRAMA, but since MMAC is scheduled-based, its delay is above all contention-based protocols. Percentage received (packet success rate) is the highest for MMAC (see Fig. 3.13) since it is the only protocol that handles mobility, and number of received packets lower slightly as compared with others when mobility increases. With TDMA and mobility handling, MMAC suffers less collisions and packet drops making it the most energy-efficient (see Fig. 3.14) among these protocols.

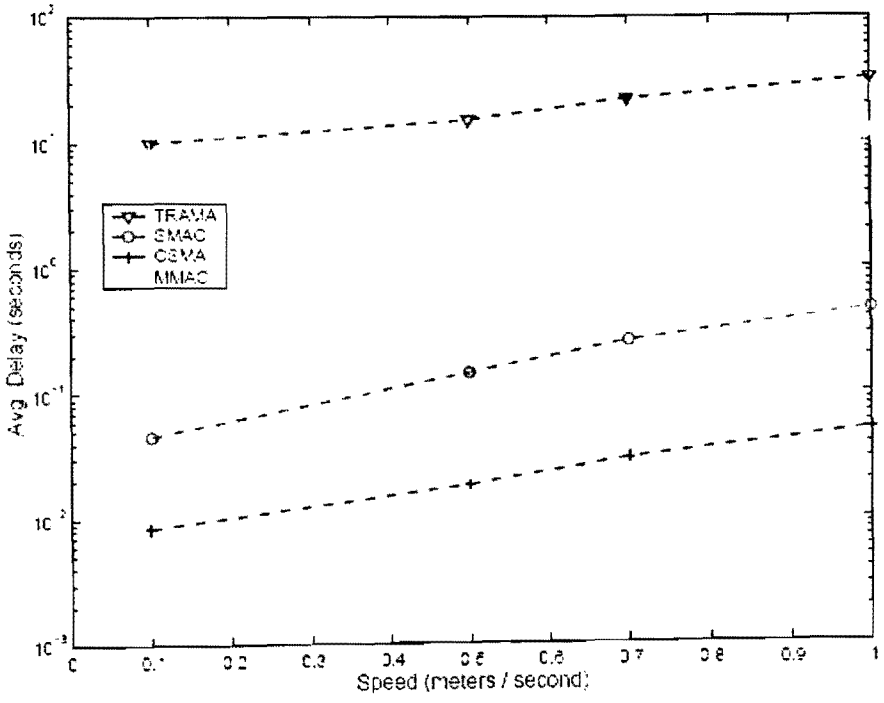

Fig. 3.12 MMAC Average End-to-End Delay at Various Speeds (source: [19])

(see Fig. 3.12), MMAC has better average delay performance than TRAMA, but since MMAC is scheduled-based, its delay is above all contention-based protocols. Percentage received (packet success rate) is the highest for MMAC (see Fig. 3.13) since it is the only protocol that handles mobility, and number of received packets lower slightly as compared with others when mobility increases. With TDMA and mobility handling, MMAC suffers less collisions and packet drops making it the most energy-efficient (see Fig. 3.14) among these protocols.

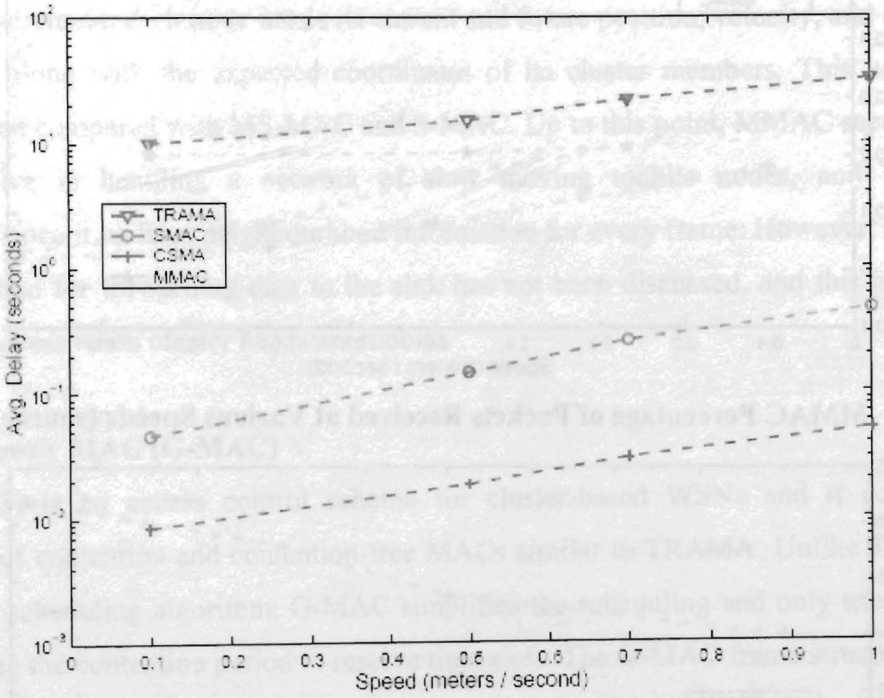

Fig. 3.12 MMAC Average End-to-End Delay at Various Speeds (source: [19])

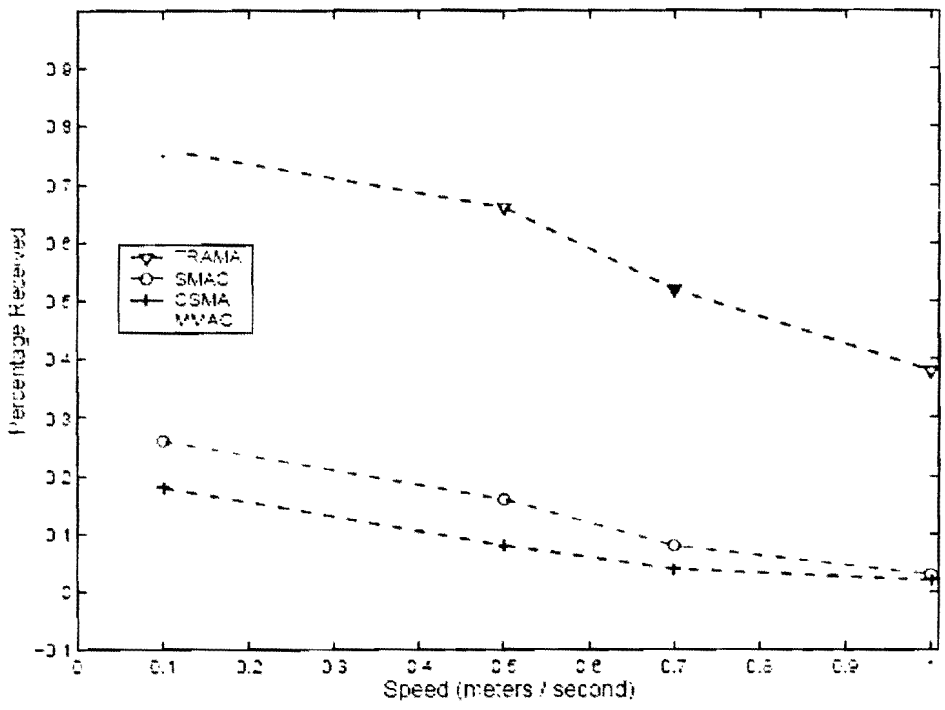

Fig. 3.13 MMAC Percentage of Packets Received at Various Speeds (source: [19])

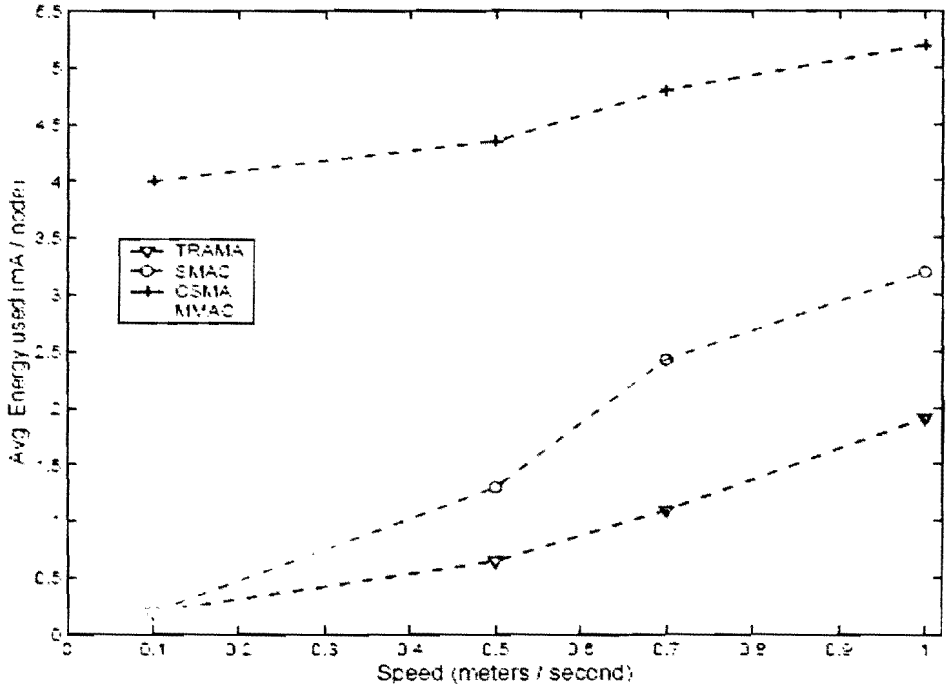

Fig. 3.14 MMAC Average Energy Used at Various Speeds (source: [19])

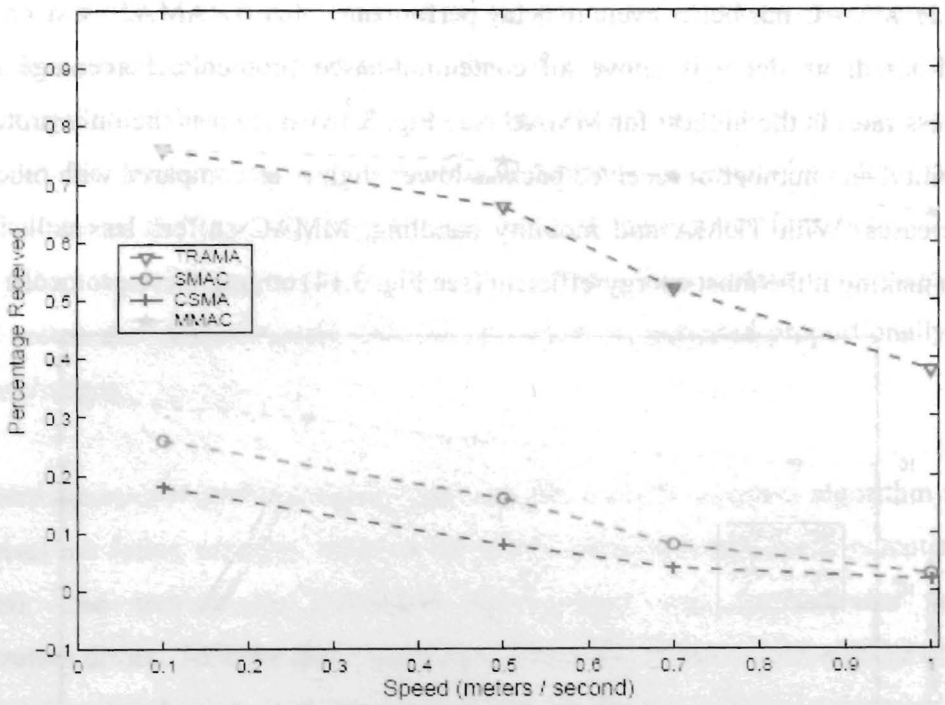

Fig. 3.13 MMAC Percentage of Packets Received at Various Speeds (source: [19])

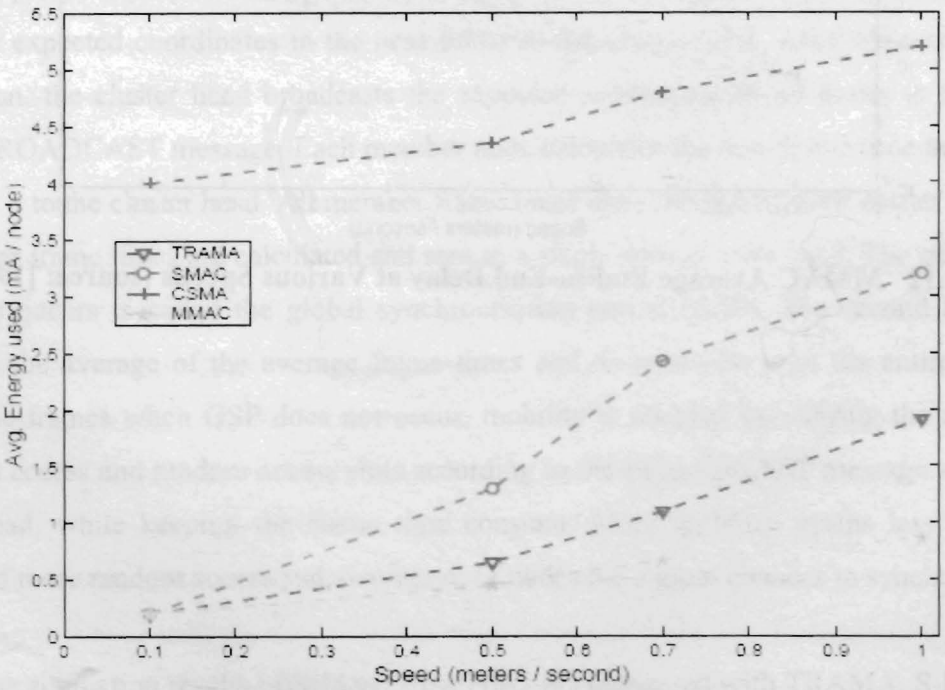

Fig. 3.14 MMAC Average Energy Used at Various Speeds (source: [19])

On top of the protocol overhead of TRAMA, MMAC adds its own complex procedures and calculations for mobility prediction, which can dramatically affect performance on a real sensor node that has minimum computing power. The problems with simulations in NS-2 are exclusions of modeling sensor node computation time and power consumed by the GPS receiver. Thus, the actual delay associated with MMAC can be extremely high when implemented and energy consumption with the use of GPS can be significant. In terms of memory requirement, each node needs its current and future position, velocity, and acceleration information along with the expected coordinates of its cluster members. This uses a lot of memory when compared with MS-MAC and S-MAC. Up to this point, MMAC seems to be the most effective in handling a network of slow moving mobile nodes, and TRAMA is comparable since it updates neighbourhood information for every frame. However, inter-cluster communication for forwarding data to the sink has not been discussed, and this is crucial for reception success when cluster heads are mobile.

3.2.2 Gateway MAC (G-MAC)

G-MAC [23] is an access control scheme for cluster-based WSNs and it combines the advantages of contention and contention-free MACs similar to TRAMA. Unlike TRAMA and its complex scheduling algorithm, G-MAC simplifies the scheduling and only transmitters are awake during the contention period to reserve time slots. The G-MAC frame structure is shown in Fig. 3.15. There are three sections: contention section for nodes to send transmission requests (inter-network or intra-network) to the gateway (cluster head), gateway traffic indication message (GTIM) section for the cluster head to broadcast the schedules, and contention-free section for scheduled transmissions within the cluster.

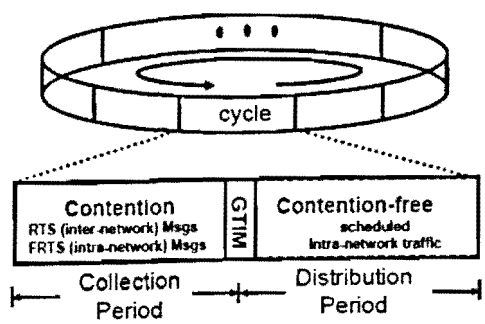

Fig. 3.15 G-MAC Frame Structure (source: [23])

On top of the protocol overhead of TRAMA, MMAC adds its own complex procedures and calculations for mobility prediction, which can dramatically affect performance on a real sensor node that has minimum computing power. The problems with simulations in NS-2 are exclusions of modeling sensor node computation time and power consumed by the GPS receiver. Thus, the actual delay associated with MMAC can be extremely high when implemented and energy consumption with the use of GPS can be significant. In terms of memory requirement, each node needs its current and future position, velocity, and acceleration information along with the expected coordinates of its cluster members. This uses a lot of memory when compared with MS-MAC and S-MAC. Up to this point, MMAC seems to be the most effective in handling a network of slow moving mobile nodes, and TRAMA is comparable since it updates neighbourhood information for every frame. However, inter-cluster communication for forwarding data to the sink has not been discussed, and this is crucial for reception success when cluster heads are mobile.

3.2.2 Gateway MAC (G-MAC)

G-MAC [23] is an access control scheme for cluster-based WSNs and it combines the advantages of contention and contention-free MACs similar to TRAMA. Unlike TRAMA and its complex scheduling algorithm, G-MAC simplifies the scheduling and only transmitters are awake during the contention period to reserve time slots. The G-MAC frame structure is shown in Fig. 3.15. There are three sections: contention section for nodes to send transmission requests (inter-network or intra-network) to the gateway (cluster head), gateway traffic indication message (GTIM) section for the cluster head to broadcast the schedules, and contention-free section for scheduled transmissions within the cluster.

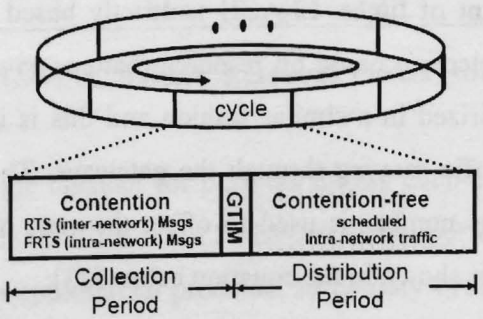

Fig. 3.15 G-MAC Frame Structure (source: [23])

The contention section is the collection period since both inter-network (non-local) and intra-network (local) traffic requests are collected in this interval. Inter-network traffic carries messages destined for nodes in another cluster, and intra-network traffic contains messages to be exchanged between nodes within a cluster for data fusion. The two types of traffic are differentiated using RTS as the request message for non-local traffic and FRTS (future RTS) for local traffic. Fair scheduling is supported by the random exponential backoff that each sender must wait for before sending the RTS or FRTS, and of course, carrier-sensing is performed before each transmission. The probability of request collision depends on the size of the contention window and the number of senders. For inter-network messages, the sender and gateway message sequence is RTS/CTS/DATA/ACK and this happens immediately during the collection period, since the gateway is the receiver. CTS is only used for inter-network messages and this reduces protocol overhead. After all transactions are completed, the gateway uses time slots at the end of the distribution period to forward all inter-network messages out of the cluster, and then goes into sleep mode. During the distribution period, all nodes are awake to receive GTIM messages which carry the current time, the next collection period, the next distribution period, and traffic exchange slots described by source, destination(s), and relative time offset. Nodes sleep during the contention-free interval and wake up only on the time slots they are scheduled to transmit or receive. After each exchange slot, both transmitter and receiver(s) return to sleep mode. If a node is not scheduled, it remains asleep until the distribution period ends.

In addition to defining a communication scheme, G-MAC proposes a resource adaptive voluntary election (RAVE) method for cluster head selection. RAVE considers current node energy and memory unlike LEACH [22] where the goal is to ensure every node serves as the cluster head the same amount of times. LEACH is strictly based on probability calculation while RAVE performs self-election based on resource (battery power) levels (see Table 3.1). Memory level can be categorized in a similar fashion and this is important since memory is limited for inter-network traffic moving through the gateways. The resource level (RL) is a number from 0 to 3 and this number is used to offset the start of the self-election random backoff by multiples of 128 as shown in the equation below [23]:

$$ElectionBackoff = Random(2^7) + (RL * 128) \quad (6)$$

where *ElectionBackoff* is the wait time before a self-election message is broadcasted, and *Random*(2^7) is a number from 0 to 127. This election procedure happens when: (1) the gateway transitions to a lower power level and initiates election by setting an election flag bit in the GTIM, (2) the gateway is at a critical memory level and initiates election, and (3) the default gateway changeover frequency is reached. The new gateway is the first node that transmits a self-election message during the beginning of the GTIM period. The departing gateway acknowledges the new gateway and continues distributing the schedule, and then finally changes to regular node status. In the case of gateway failure, three consecutively missed GTIMs will trigger an election and this method of self-election is used during self-configuration of the cluster.

G-MAC, S-MAC [8], Timeout MAC (T-MAC) [24], Berkeley MAC (B-MAC) [25], and IEEE 802.11 MAC are modeled and simulated with 40 nodes in one cluster and bandwidth of 62.6 kbps. Power consumption is based on the CC2420 transceiver from Chipcon Corporation, which requires 19.7 mA for receiving, 17.4 mA for transmitting, and 0.02 mA for sleeping. Also, the frame time for all MAC protocols is 500 ms. For G-MAC, the size of the GTIM is 33 bytes + (three bytes * number of packets/frame) and its average duty cycle is 0.95%. The S-MAC has a fixed duty cycle of 10%. The T-MAC has adaptive sleep timeout set at 10.2 ms and contention period fixed at 5 ms for every packet, so its duty cycle is 2.1%.

Battery Pwr Level	Power Level Nomenclature	Voltage Range (volts)
00	High	2.6 < Pwr ≤ (3.0-3.6)
01	Med	2.4 < Pwr ≤ 2.6
10	Low	2.1 < Pwr ≤ 2.4
11	Min	Pwr ≤ 2.1

Table 3.1 G-MAC Battery Resource Levels (source: [23])

The B-MAC senses the channel for 0.35 ms during each 14 ms check interval, so its duty cycle is 2.5%. The modeled IEEE 802.11 MAC does not perform power saving, which is typically not implemented in commercial products, so its duty cycle is 100%. Less duty cycle results in more energy conservation and this is shown in Table 3.2.

MAC Protocol	Network Lifetime (days)		
	Empty Network (no traffic)	Unicast Traffic	Broadcast Traffic
802.11	6	6	6
S-MAC	63	88	63
B-MAC	244	87	87
T-MAC	295	130	108
G-MAC	480	455	203

Table 3.2 G-MAC Network Lifetime at Various Traffic Patterns (source: [23])

The unicast and broadcast traffics are modeled by four 32-byte messages per second. Since only transmitters are awake during the contention period, G-MAC is more energy-efficient than the other protocols. Also, G-MAC does not require a densely populated cluster to save energy by distributing the gateway cost to all cluster members, and this is shown in Fig. 3.16 by the slow variation on network lifetime between 25 nodes and 100 nodes. On the other hand, although the scheduling algorithm is said to be simple, the actual scheduling procedure is not explained (which could be first-come-first-serve). Also, global synchronization and communication (inter-network message exchange) between clusters is not considered at all. For G-MAC, each cluster head should leave a fixed number of time slots for inter-network messages.

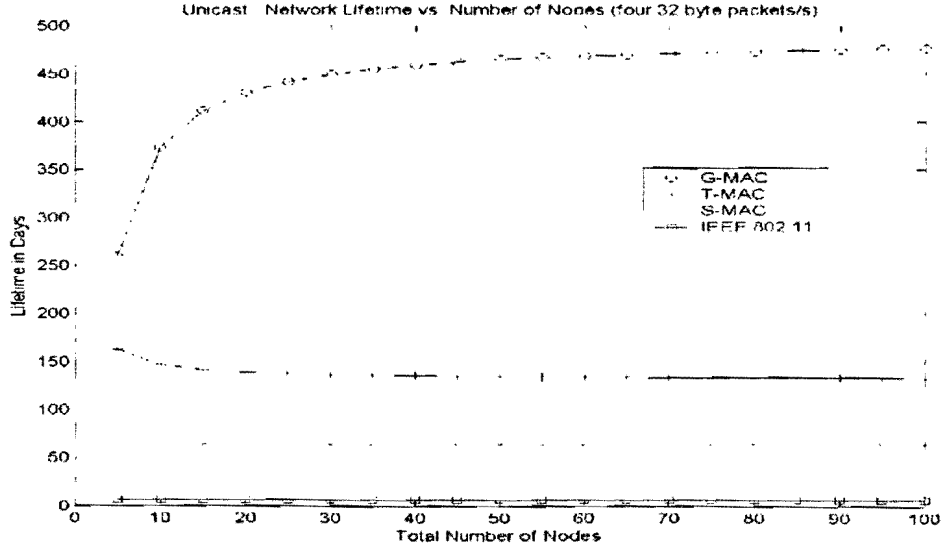

Fig. 3.16 G-MAC Network Lifetime at Various Numbers of Nodes (source: [23])

MAC Protocol	Network Lifetime (days)		
	Empty Network (no traffic)	Unicast Traffic	Broadcast Traffic
802.11	6	6	6
S-MAC	63	88	63
B-MAC	244	87	87
T-MAC	295	130	108
G-MAC	480	455	203

Table 3.2 G-MAC Network Lifetime at Various Traffic Patterns (source: [23])

The unicast and broadcast traffics are modeled by four 32-byte messages per second. Since only transmitters are awake during the contention period, G-MAC is more energy-efficient than the other protocols. Also, G-MAC does not require a densely populated cluster to save energy by distributing the gateway cost to all cluster members, and this is shown in Fig. 3.16 by the slow variation on network lifetime between 25 nodes and 100 nodes. On the other hand, although the scheduling algorithm is said to be simple, the actual scheduling procedure is not explained (which could be first-come-first-serve). Also, global synchronization and communication (inter-network message exchange) between clusters is not considered at all. For G-MAC, each cluster head should leave a fixed number of time slots for inter-network messages.

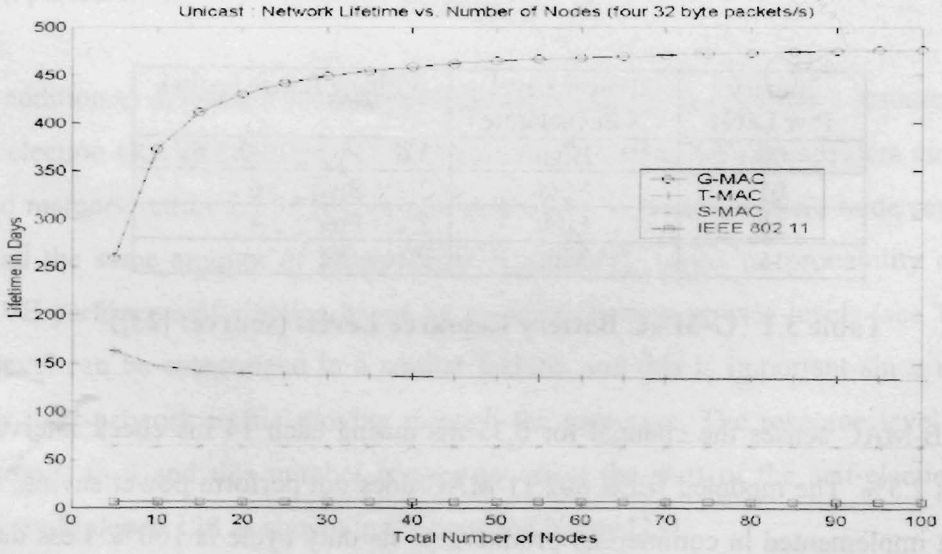

Fig. 3.16 G-MAC Network Lifetime at Various Numbers of Nodes (source: [23])

3.2.3 Hybrid MAC (Z-MAC)

Z-MAC [26] also combines contention and contention-free methods, but time slots are assigned at deployment and will only be reassigned when significant changes in topology (physical relocation of nodes) occur. Cluster formation is not required for scheduling since nodes run a distributed scheduling algorithm known as DRAND or distributed RAND. The algorithm takes in the two-hop neighbourhood information and uses local message passing within the neighbourhood to generate a time slot assignment, where no two nodes are given the same slot to prevent the hidden terminal problem. Protocols, such as S-MAC, uses RTS/CTS to reduce the hidden terminal problem, but research results show that it can use 40-75% of channel capacity since sensor network packets are usually very small (e.g. 40 bytes). Although each time slot has an owner, non-owners can transmit on any time slot but higher priority is given to owners and this is guaranteed by the difference in contention window size. As with other protocols, carrier sensing with backoff is used as a technique for overcoming synchronization errors. For low contention level (LCL) the protocol works as CSMA and for high contention level (HCL) the protocol behaves like TDMA.

After deployment, each node runs a neighbour discovery protocol which uses broadcasted ping messages to one-hop neighbours to collect their one-hop neighbour list. Each ping message carries the current list of one-hop neighbours belonging to the sender and overtime this protocol will gather the two-hop neighbour list needed by each node. In the implementation, each node sends a ping randomly per second for 30 s. DRAND is capable of generating slot numbers (starting from 0) that does not exceed the size of the two-hop neighbourhood. The complexity of DRAND is upper bounded by $O(\delta)$, where δ is the size of the neighbourhood, so its energy consumption is proportional δ. Even when a small number of nodes join after DRAND has assigned the time slots, additional slots can be assigned without changing the previously assigned slots. Details of DRAND can be found in [27]. With the assigned slots, time framing is required for proper transmissions on those slot periods. Global time framing is not efficient since the maximum slot number needs to be propagated throughout the network, and it becomes costly when topology changes (node joins and failures). Thus, local time framing occurs within each two-hop neighbourhood according to the following time frame rule [26]:

Let a node i be assigned a slot s_i and the maximum slot number of its two-hop neighbourhood is F_i. Set i's time frame to 2^a where a is a positive integer satisfying the condition $2^{a-1} \le F_i < 2^a - 1$. This means i's slots are $l \cdot 2^a + s_i$, for all $l = 0,1,2,3....$

With this rule, only one node in a two-hop neighbourhood will own a time slot in every frame. By using the local maximum slot number, channel usage is higher and delay is lower. On the downside, empty slots will always be created by the time frame rule and only global time framing can remove them. For networks with many sparse areas and a few dense areas, the local time framing is more efficient, but even if the network is sparse, empty slots can still be contended since CSMA is used during every slot. After running DRAND for slot assignment and using the time frame rule, the frame size and slot number of each node is forwarded within the two-hop neighbourhood. Every node then synchronizes to slot 0. Local and global synchronizations will be discussed later.

The two contention levels, LCL and HCL, are used by the transmission control of Z-MAC for channel utilization and fairness purposes. If a node receives an explicit congestion notification (ECN) message from a two-hop neighbour, then the node changes its state from LCL to HCL. It stays in HCL until the t_{ECN} period expires and receiving another ECN will refresh this timer. After the t_{ECN} period expires, the node goes back to LCL. All nodes in LCL can contend for transmission in any slot, but in HCL, only slot owners and one-hop neighbours of those owners can contend. If a slot is empty or if the owner has no data to send, then the slot is open for competition to non-owners. For every slot, the owner with data to transmit must perform a random backoff within a fixed time period T_o and carrier sense the channel before transmitting. A busy channel will force the node to wait for a clear channel and perform random backoff again. Empty slots are contended fairly by LCL and HCL nodes and this is ensured by having them wait for T_o, and then random backoff within the contention window T_o to T_{no}. The values of T_o and T_{no} are chosen based on stochastic analysis for maximizing throughput. They can also be chosen intuitively by noting that if a synchronization error is no more than one slot size, then there are either two or three owners and a contention window of

eight slots for T_o can optimally overcome this issue. T_{no} is set as 32 slots since this is the common initial size of the contention window in IEEE 802.11.

ECN messages are sent by the owner of a slot to remove hidden terminals (two-hop neighbours) from contending for the channel when contention level is high. Having the carrier-sensing range nearly twice the communication range can help to measure the two-hop contention level. Under this case, a node's carrier-sensing range can sense if any of its two-hop neighbours is transmitting, and if one of them is then it will back off. The number of backoffs can be used as an indication of the current two-hop contention level. When there is high contention, a one-hop ECN is unicasted to the destination and if there are multiple destinations, then a one-hop ECN is broadcasted with the destinations included in it. A node that receives the one-hop ECN will check if it is the destination of the ECN message, and if it is then it will broadcast this ECN to its one-hop neighbours (this ECN now becomes the two-hop ECN). If it is not the destination, then it will discard the message. To prevent ECN implosion, if a node during random backoff receives an ECN with the same destination as its ECN, then it cancels the ECN transmission. Forwarding nodes will also cancel their transmissions if they previously forwarded an ECN with the same destination and within the t_{ECN} period. After every t_{ECN} period, if a node still experiences high contention, it will send an ECN message again. In all cases, receivers will maintain a short listening duty cycle similar to B-MAC [25], which has a short check period for the transmitter to send a preamble to the receiver. Preambles are as large as the check periods and they are sent before any data transmissions. This means a time slot should be longer than the sum of the check period, T_o, T_{no}, carrier-sensing period, and one packet propagation time.

Under low contention, Z-MAC works like CSMA even without synchronization and only under high contention is synchronization needed since the protocol behaves like TDMA. Nodes transmitting at higher data rates will transmit more frequent synchronization messages that carry the current clock value since neighbouring senders require it. Receivers will passively synchronize with their senders using a weighted moving average of its current clock and the received clock value. One synchronization packet is sent for every 100 data packets. Since only senders transmit synchronization packets, low traffic areas will have high clock

drifts apart from the high traffic areas. Thus, the averaging weight is determined by a trust factor β_t of the sending node. The minimum synchronization interval required to achieve the maximum clock error or less is $I_{sync} = \epsilon_{clock} / r_{drift}$, where ϵ_{clock} is the maximum acceptable clock error and r_{drift} is the maximum clock drift rate. The trust factor $\beta_t = \min\{\alpha, S \times I_{sync} \times \alpha_{sync}\}$, where α is the current moving average weight, S is the average rate of receiving and sending synchronization messages, and α_{sync} is the maximum weight of the new clock value received. The weighted moving average of the new clock value received is $C_{avg} = (1 - \beta_t) C_{avg} + \beta_t \cdot C_{new}$.

Experiments include NS-2 simulations and Mica-2/TinyOS implementations for Z-MAC and B-MAC [25]. B-MAC has been shown to have better performance than both S-MAC [8] and T-MAC [24]. Other NS-2 simulations are done for PTDMA and Sift, but B-MAC is a better comparison to examine since Z-MAC is built on top of B-MAC. There are three separate benchmarks: one-hop, two-hop, and multi-hop. For better comparisons between the two protocols and a realistic setting, the multi-hop results for Mica-2 experiments are shown and discussed. The multi-hop benchmark includes 42 Mica-2 nodes with the maximum two-hop neighbourhood size at 27 and the maximum local frame size at 32. One of the 42 nodes is a sink while the rest are sources, and data rate is varied to analyze the protocol behaviour in both low and high contention scenarios. Fig. 3.17 shows the throughput of both protocols at various data rates (packets per second where each packet is 36 bytes). Up to four packets per second, B-MAC performs slightly better than Z-MAC, but above that data rate, high contention occurs and Z-MAC continues to deliver higher throughput while B-MAC suffers from added contention (possibly added collisions). Fairness is also important so it is studied by measuring how packet delivery to the sink is uniformly distributed among the sources, where uniform distribution will result in a fairness index of one. From the results shown in Fig. 3.18, increase in data rate has significant impact on fairness, but HCL mode with ECN messages help alleviate this since slot owners (every node) have higher chances of transmitting on their slots. Finally, the throughput per energy in Fig. 3.19 again reveals the advantages that Z-MAC brings

when contention is high. Since Z-MAC does not update neighbourhood information per frame time, mobility will cause its performance to degrade significantly.

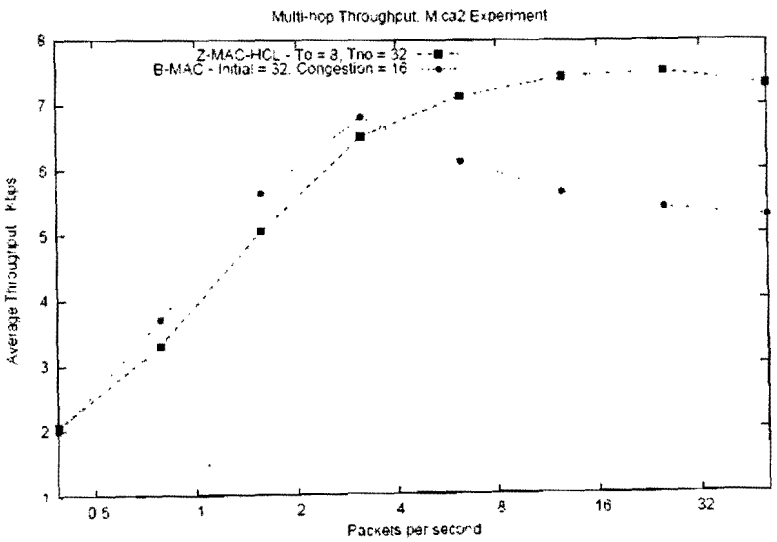

Fig. 3.17 Z-MAC Multi-hop Throughput at Various Data Rates (source: [26])

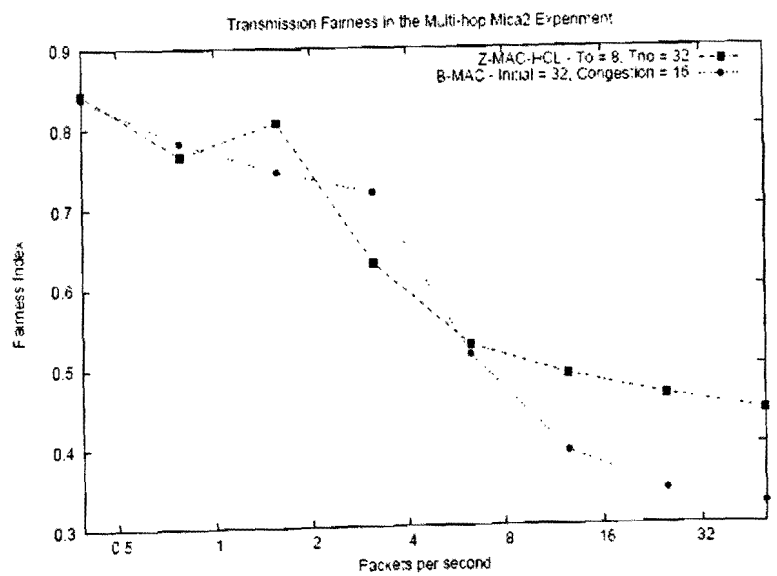

Fig. 3.18 Z-MAC Multi-hop Fairness Index at Various Data Rates (source: [26])

when contention is high. Since Z-MAC does not update neighbourhood information per frame time, mobility will cause its performance to degrade significantly.

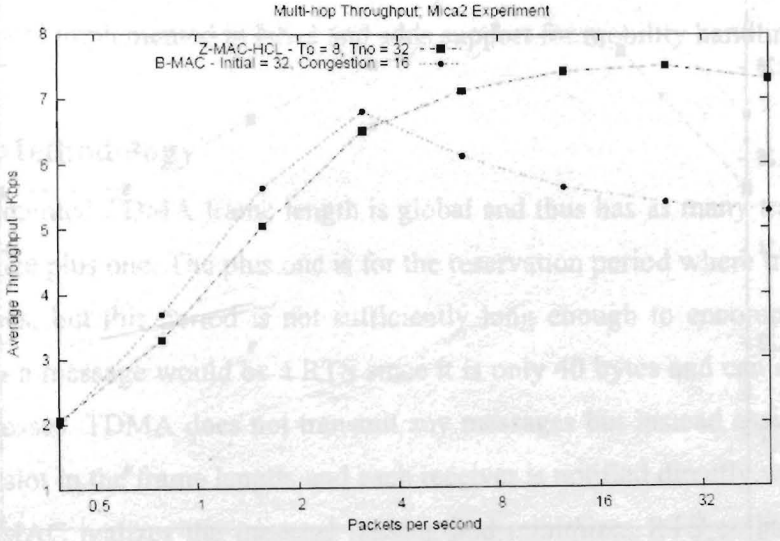

Fig. 3.17 Z-MAC Multi-hop Throughput at Various Data Rates (source: [26])

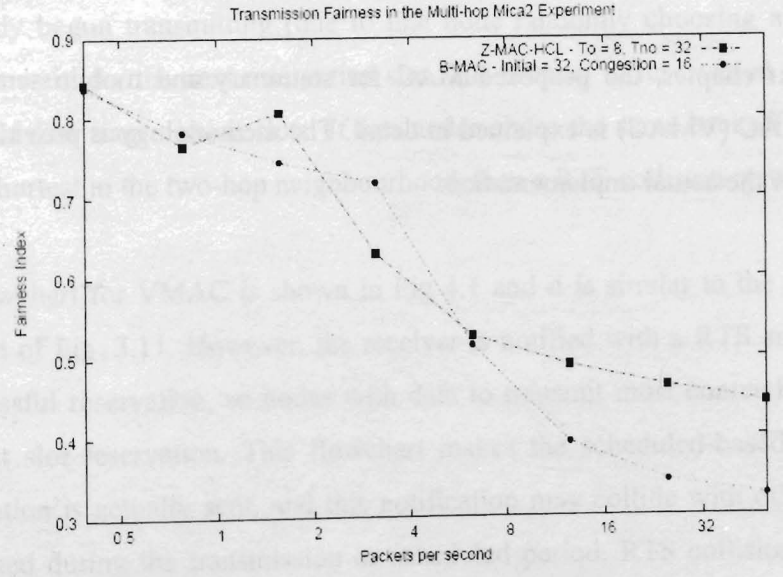

Fig. 3.18 Z-MAC Multi-hop Fairness Index at Various Data Rates (source: [26])

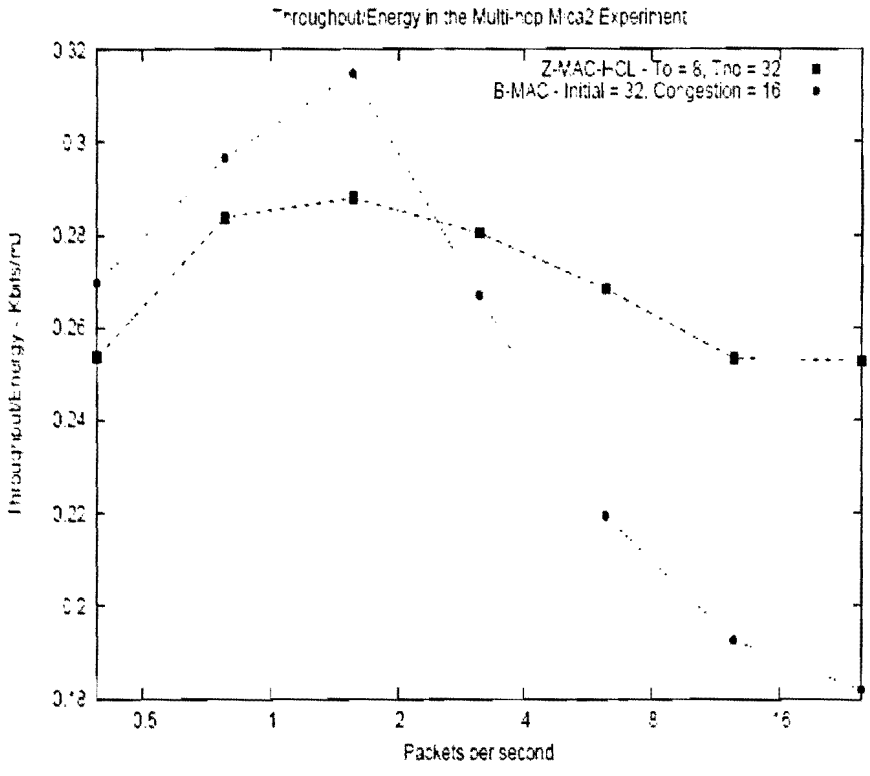

Fig. 3.19 Z-MAC Multi-hop Throughput/Energy at Various Data Rates (source: [26])

In the next chapter, the proposed MAC for stationary and mobile sensor networks, called versatile MAC (VMAC) is explained in detail. The methodology is provided along with the pseudo-code of the actual implementation.

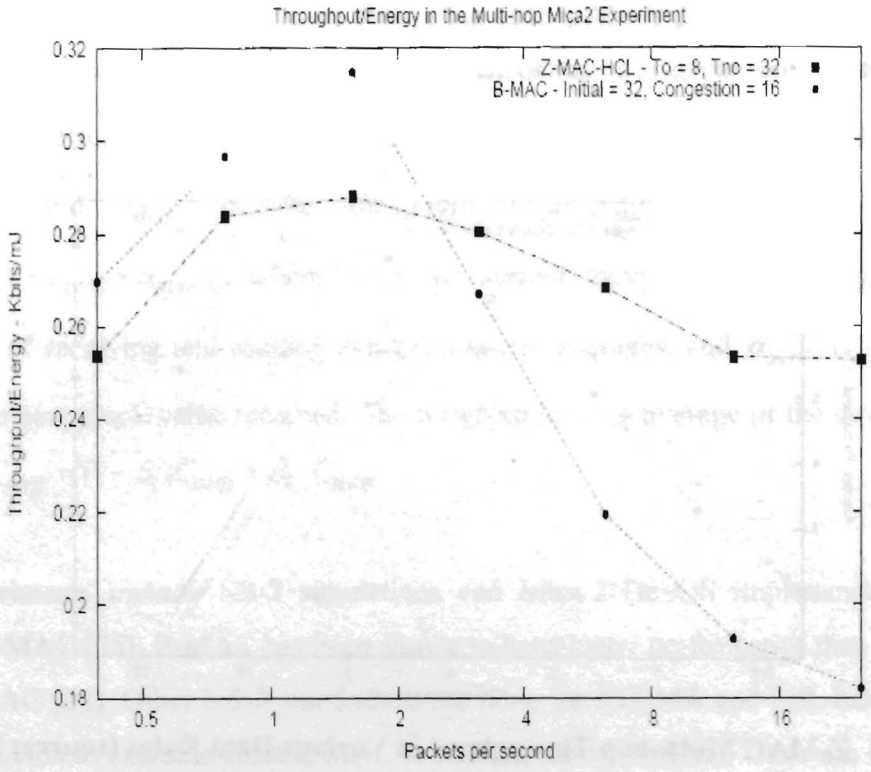

Fig. 3.19 Z-MAC Multi-hop Throughput/Energy at Various Data Rates (source: [26])

In the next chapter, the proposed MAC for stationary and mobile sensor networks, called versatile MAC (VMAC) is explained in detail. The methodology is provided along with the pseudo-code of the actual implementation.

4. Versatile MAC (VMAC) Protocol

Although all the protocols mentioned earlier have their own flaws when it comes to mobility handling, the study of them has lead to the development of VMAC. VMAC is an extension of the TDMA protocol implemented in NS-2 and adds support for mobility handling.

4.1 VMAC Methodology

The NS-2 implemented TDMA frame length is global and thus has as many transmission slots as the network size plus one. The plus one is for the reservation period where transmitters share receiver addresses, but this period is not sufficiently long enough to encompass all message exchanges. Such a message would be a RTS since it is only 40 bytes and can carry source and destination addresses. TDMA does not transmit any messages but instead assigns each node a permanent time slot in the frame length, and each receiver is notified directly which time slot to wake up on. VMAC realizes the message passing and minimizes RTS collisions by using a suitable contention window (CW) size as backoff. After waiting for a random number of backoff slot times within the range of the CW size, a node tries to transmit a RTS. If another node has already begun transmitting (due to that node randomly choosing a shorter backoff number), then it waits for the next reservation slot. The number of reservation slots is the same as the number of data transmission slots. If two nodes chose the same backoff number and the backoff is the shortest in the two-hop neighbourhood, then a RTS collision results.

The flowchart for VMAC is shown in Fig 4.1 and it is similar to the scheduled-based MAC flowchart of Fig. 3.11. However, the receiver is notified with a RTS and RTS collision means unsuccessful reservation, so nodes with data to transmit must contend for another slot during the next slot reservation. This flowchart makes the scheduled-based MAC practical since a notification is actually sent, and this notification may collide with others leading to a slot being unused during the transmission or scheduled period. RTS collisions happen at the receiver and this is explained earlier as the hidden terminal problem, but since the transmitter does not have to wait for a CTS before assuming success of the RTS, both transmitters wake up during the unreserved slot to transmit while the receiver is still sleeping. This is inefficient but a CTS is also 40 bytes in size and can be significant if the data packet size is only 40 bytes as

well. No matter if the RTS is received or collided, both receiver and transmitter keep track of it and this allows the current reserved slot to be synchronized across the network.

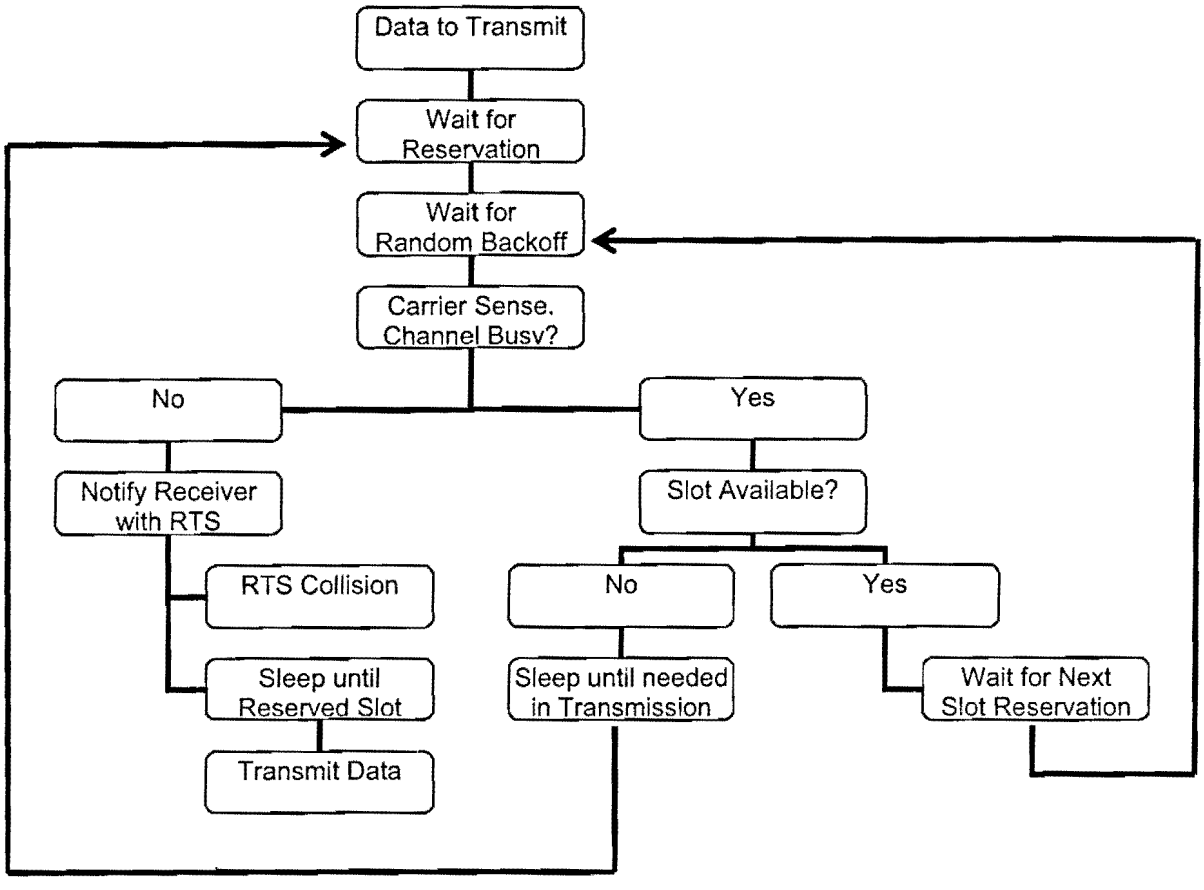

Fig. 4.1 VMAC Flowchart

The CW is similar to IEEE 802.11 DCF but is fixed instead of doubling after every failed attempt (failed either because of RTS collision or another node choosing a shorter backoff time and winning the transmission slot). Having a fixed CW allows a fair distribution of medium access time for each node since failed attempts do not make winning even more difficult. Also a fixed CW is deterministic and global time synchronization is possible when the maximum backoff time, RTS transmission time, and data transmission time are all known. The reservation period becomes a practical random-access, slot-reservation period (with reservation slots equal to transmission slots) since now it is long enough for backoff and RTS transmission for each and every transmission slot. Therefore, a reservation slot is split into a CW period for random backoff and a RTS transmission period as depicted in Fig. 4.2.

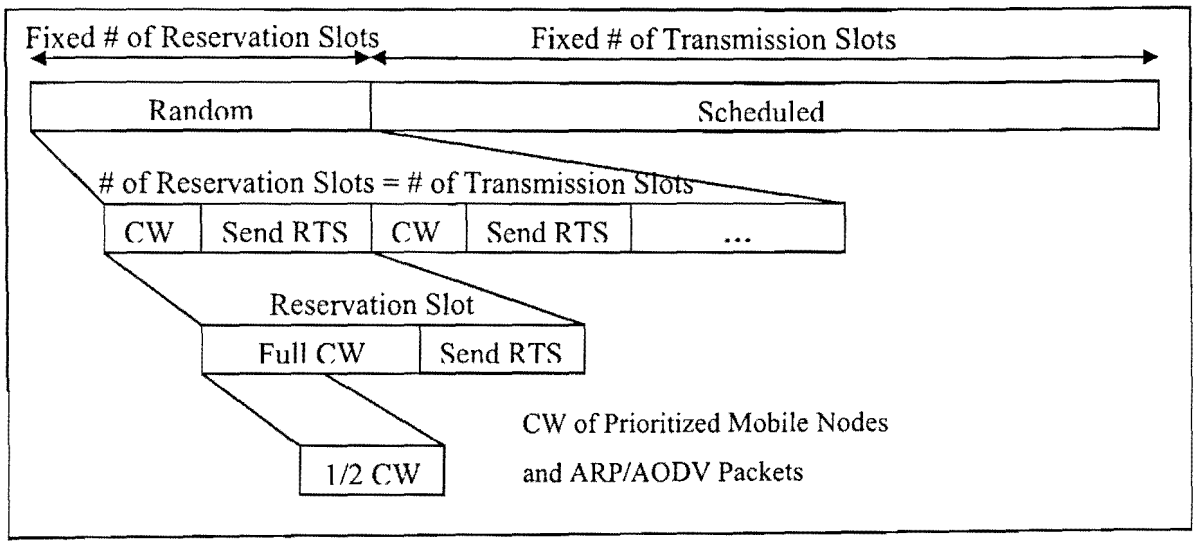

Fig. 4.2 VMAC Frame Structure

As shown on Fig. 4.2, VMAC is a scheduled-based MAC protocol that does distributed scheduling with a first-come first-serve basis. VMAC can be configured for periodic, event-driven, and query-driven data delivery models. Also, VMAC can exploit sleep synchronization which is more difficult in contention-based. This minimizes energy consumption of the radio by turning it on only when necessary. There are two parts to the frame structure: (1) a random period for contention-based reservation of a fixed number of transmission time slots, and (2) a scheduled period for collision-free transmission of data on reserved slots. All nodes with data to send compete for a time slot in the two-hop neighbourhood during the random period. The CW is a fixed number of possible wait times a node randomly picks before it performs carrier sensing (for sensing transmissions already taking place) and transmits a RTS to reserve a time slot.

The CW of mobile nodes (and also ARP/AODV packets since data cannot be transmitted without a hardware address or route) is half of the CW of stationary nodes, therefore giving mobiles priority in sending data. Stationary nodes have a wait time greater than the CW of the mobiles so stationary nodes can never compete with mobiles. During the random period, all nodes are awake and if they are the receiver of a time slot they will receive a RTS from the transmitter for that particular slot. Even if they are not the receiver, they are still listening to the RTS to keep track of the current reserved slot, so that when they do receive a

RTS they will know what time slot to wake up on during the scheduled period. After the random period, a node sleeps if it is neither sending nor receiving and wakes up in the beginning of a slot if it is the transmitter or receiver. After each transmission or reception, the node is placed in sleep mode again until it is needed in transmission or the start of the next random period. Time synchronization can be provided by simple techniques that can be added to the protocol, but currently it is assumed to be time-error free.

4.2 VMAC in Pseudo-code Format

The following is the pseudo-code of the actual process conducted in C++. At the beginning of the reservation period, all nodes are woken up and they call the function `reserveSlot()`. (Line 1) First, the function checks the node to see if it has a packet to transmit, and it enters a contention process if it does. (Line 2) The contention process is only continued if a slot is still available, i.e. the last reserved slot is not equal to the frame length, but since it is the start of the period all slots should be available. (Line 3) A random positive integer is returned by the `backoffCalculator()`, and the range of random backoff numbers for prioritized packets is between 0 to 60 while regular packets range from 61 to 121. If priority is turned on, then priority is given to a node if its speed is greater than zero (in motion) or the packet to be sent belongs to ARP or AODV. To provide numbers within the range of 0 to 60 in a uniform distribution, randomly generated numbers are divided by the prime number 61 and its remainder becomes the random backoff. Regular packets are backed off the same way, but are offset by 61, therefore separating the packets into two ranges. Using a prime number is necessary to minimize RTS collisions since each node is equally likely to receive anyone of the numbers in the range specified.

(Line 4) Next, the random backoff is checked and if it is zero then (Line 5) the time left in the reservation slot is recorded as the reservation slot time, and (Line 6) the node goes into the backoff handler without waiting. (Line 8) If the random backoff is nonzero, (Line 9) random is decremented by one since the node is only waiting for one backoff slot time, and then it enters the backoff handler. (Line 10) Time left in the reservation slot is recorded after subtracting one backoff slot time from the reservation slot time. (Line 11) The node waits for one backoff slot time and the backoff handler is called.

The `backoffHandler()` handles the event when the backoff timer expires and will repeat the contention process until the node wins out or all slots are reserved. (Line 16) As a precaution, the existence of a packet is checked again and the reserved time cannot be expired. (Line 17) If a slot is still available, the contention process continues or else it is stopped until the next random period. (Line 18) The time left in the reservation slot is checked and if it is nonzero, then the node is still in contention with its given random backoff. (Lines 33 to 42) If it is zero, then the node is in a new reservation slot and must repeat the previous procedure taken in Lines 3 to 11. (Line 19) If the node is still in the same reservation slot, the channel status is checked. (Lines 20 to 22) If the channel is idle and the random backoff has reached zero, then a RTS is transmitted with the receiver set the same as the receiver of the packet, and the contention process stops. (Lines 28 to 30) If the channel is busy due to another node sending a RTS, then that time slot has been reserved and the node waits until the start of the next random period. (Lines 24 to 26) If the random backoff is nonzero, then the backoff is decremented and the time left is updated, and the node waits for another backoff slot time.

Sending a RTS during the contention process does not guarantee the slot is reserved, because a RTS collision could have occurred. If a RTS collision occurred, the transmitter wakes up during the unreserved slot and will try to send a packet even though the receiver is still sleeping. As explained earlier, this is a trade-off that must be taken since the control overhead of RTS/CTS exchange is overwhelming for small data packets. If no RTS collision occurred, then the transmitted data packet will be received successfully without collision since both nodes will be awake at that specific slot. Each node may only send one RTS per frame whether it is collided or not, and it may only reserve one slot per frame.

```
reserveSlot() // Called upon at the start of reservation period
1  if (packet pending) {
2     if (last_reserved_slot_ != frame_length_) {
3       random = backoffCalculator();
4       if (random == 0) {
5         restimeleft = RESERVE_SLOT_TIME;
6         backoffTimer.start(0);
7       }
```

```
8       else {
9         random--;
10        restimeleft = RESERVE_SLOT_TIME - Backoff_SlotTime
11        backoffTimer.start(Backoff_SlotTime);
12      }
13    }
14
15  }

backoffHandler() // Handles event when Backoff Timer expires
16  if (packet pending && reserve time not expired) {
17    if (last_reserved_slot != frame_length_) {
18      if (restimeleft != 0) {
19        if (channel idle) {
20          if (random == 0) {
21            sendRTS();
22            return;
23          }
24          random--;
25          restimeleft = restimeleft - Backoff_SlotTime;
26          backoffTimer.start(Backoff_SlotTime);
27        }
28        else {
29          backoffTimer.start(restimeleft);
30          restimeleft = 0;
31        }
32      }
33      else {
34        random = backoffCalculator();
35        if (random == 0) {
36          restimeleft = RESERVE_SLOT_TIME;
37          backoffTimer.start(0);
38        }
39        else {
40          random--;
```

```
41            restimeleft = RESERVE_SLOT_TIME - Backoff_SlotTime
42            backoffTimer.start(Backoff_SlotTime);
43          }
44        }
45      }
46  }
```

4.3 Mobility Handling in VMAC

Sensor nodes become mobile either randomly or under control. Random movement is expected when nodes are placed on mobile targets for monitoring purposes. Controlled movement is initiated either for discovery purposes when nodes are placed on remote-controlled vehicles or for data collection when sinks maneuver around to gather data from sources. Randomized mobility can be detected or predicted, and exploiting detection is the focus of this thesis. Controlled mobility sometimes require added infrastructure, such as tracks or cableways, for tractability of paths taken by sinks and it is out of the scope of the materials covered here. Out of all the examined approaches, handling randomized mobility can be broken down into three schemes: (1) using explicit detection of mobility such as with RSS or GPS, (2) predicting mobility with the help of GPS, and (3) adaptive rate of two-hop neighbourhood update. RSS is inaccurate in estimating mobility when more than one node is mobile. Thus, GPS is the common factor of the first two schemes when there are several or even a network of mobile nodes. Updating the two-hop neighbourhood provides an indication of overall node movements, but performing neighbour discovery is the task of the routing protocol so it is inefficient if the MAC protocol repeats it. For efficiency, a cross-layer technique between the MAC and routing protocol should be considered. However, neighbour discovery requires a lot of message passing; and when used to dynamically change frame time of the neighbourhood, global synchronization becomes a major issue.

The methods from the literature survey have their own pros and cons and it is briefly summarized here. S-MAC is energy efficient, but suffers performance degradation when nodes are mobile since the protocol is designed for stationary nodes. MS-MAC adds mobility handling to S-MAC, but can only detect a single mobile node since RSS is not sufficient for

detection. S-MAC with proactive synchronization requires GPS and it is theoretically believed to be able to accommodate several mobiles. S-MAC with EKF also requires GPS in real implementations and can significantly reduce frame losses during high-speed mobile scenarios, but it is only tested with single hopping and within a single cluster. MMAC also requires GPS and it is a complex protocol, but it supports a network of slowly moving mobiles. G-MAC requires clustering and does not constantly gather the neighbour list. TRAMA and Z-MAC does not require clustering, but TRAMA should be more effective in mobile scenarios since it updates neighbourhood information during every frame. Mobility handling is still a new topic in the research community of WSNs. Future protocols should combine a straightforward mobility detection scheme such as GPS with contention-based or scheduled-based MAC. Prediction requires too much control overhead and may greatly extend end-to-end delay.

Detecting mobility in VMAC requires the use of GPS, but each node only needs to know its own speed and not the speed of others in the two-hop neighbourhood. Also, this speed is only required when it has data to send. VMAC does not require clustering nor constant neighbourhood update. Mobility is viewed as a communication problem between a transmitter and a receiver, and their relationship can be classified into four cases:

1. Stationary transmitter and receiver
2. Stationary transmitter and mobile receiver
3. Mobile transmitter and stationary receiver
4. Mobile transmitter and receiver

Case 3 and 4 are given priority over the other two cases when mobiles have data to send, because high mobility leads to network partitions so mobiles must take advantage of their current connections. Each case also reflects on the application it supports. For instance, Case 1 represents the stationary scenario where nodes are placed in specific areas and are kept there throughout their lifetime. For this type of application, mobility handling is unnecessary and the protocol approaches the performance of TDMA. In Case 2, the receiver is mobile and it could represent the situation when sinks move around to collect data from stationary sensor nodes, which does not necessitate mobility support. For Case 3, the sensor node could be moving because it is attached to a moving target or it is placed on a vehicle for searching purposes. At any point, this mobile could lose connection to the network so it is best to give it priority in

sending its data. Case 4 is similar to Case 3 so it is treated equally. Providing mobile transmitters with high priority is simple and effective in handling the different cases.

The carrier-sensing threshold is set to be twice the receiver threshold so each node can sense twice the range it can transmit. With this setting, carrier-sensing can detect transmissions within the two-hop neighbourhood which helps prevent collisions of RTS during reservation. RTS/CTS exchange is not required since the sender of the RTS can detect if the RTS is corrupted by other transmissions at the receiver (by carrier-sensing the RTS) and retransmit it if collision occurs. Larger CW sizes can also help prevent RTS collisions but leads to longer random period and overall end-to-end delay for data transmissions. The size of CW and the number of transmission slots should be proportional to the average node density or number of nodes in the two-hop neighbourhood. The number of time slots can be more or less than the number of neighbours because nodes can be constantly moving in and out of the two-hop neighbourhood. Less transmission slots still lead to utilization of bandwidth since it achieves only a small amount less than having the transmission slots equal to the neighbourhood size. More transmission slots result in bandwidth underutilization and longer delay. Less transmission slots than number of neighbours is therefore more beneficial in high mobility environments.

In the next chapter, the performance of the proposed VMAC is evaluated using the NS-2 simulator. To implement VMAC in the simulator, two new C++ files (one source and one header) are written and added to the MAC directory of NS-2. These files are created by modifying the TDMA files already provided in the NS-2 distribution. All the features of VMAC mentioned earlier are covered by the code and its performance compared to other protocols is shown in the following chapter.

5. VMAC Performance Evaluation

5.1 Simulation Setup

To test various properties of VMAC and its comparison with other MAC protocols, it is simulated in three topologies that differ by the maximum number of hops required to reach the base station, and also the contention levels between source nodes. For all three topologies, the network configurations are the same except for the rate of data generation and protocol-specific parameters, such as VMAC frame length. The general network configuration used is tabulated below:

Channel Model	Friis free-space with TX/RX gain of 1.0
Antenna	Omni-directional
Carrier Frequency	914 MHz
Communication Range	40 m
Interference/Carrier-sensing Range	80 m
Data Rate (Bandwidth)	200 kbps
Interface Queue	Holds 50 packets with drop-tail (FIFO) queuing
Routing Protocol	AODV
Traffic Generator	Constant bit rate (CBR) with 100 Bytes of data
Rate of Traffic Generation	10 packets/s per node for one-hop, and 1 packet/s per node for two-hop and four-hop
Contention Window Slot Time	20 μs
Simulation Time	150 s
Initial Energy	200 J
Idle/Receive Power Consumption	1 W [1 J/s]
Transmit Power Consumption	2 W
Sleep Power Consumption	0.001 W
Transition Power Consumption	0.1 W (transition from sleep to active only)
Time Required for Transition	0.001 s
Number of Nodes	21 (20 source nodes and 1 sink node)

The three topologies are one-hop, two-hop, and four-hop. Each stands for the farthest a sensor node can be from the base station. For one-hop, all 20 source nodes are randomly placed within one hop of the sink and all are within each others' transmission range of 40 m. The topologies for two-hop and four-hop simulations are illustrated in Fig. 5.1 and 5.2, respectively, and Nodes 0 through 19 are source nodes while Node 20 is the sink node. For the one-hop simulation, packet generation rate is 10 packets/s per node and this rate produces a traffic of 10 packet/s/node * 20 node * 172 Byte/packet * 8 bit/Byte = 275.2 kbps, which is more than the amount the bandwidth (200 kbps) can handle. Each packet is 172 Bytes with 100 Bytes of data and 72 Bytes of control header, made up of 52 Bytes for VMAC and 20 Bytes for IP. The other protocols simulated with VMAC are TDMA and 802.11 with RTS/CTS exchange and their control header are 52 Bytes and 58 Bytes, respectively. TDMA represents the extreme of scheduled-based, while 802.11 represents the extreme of contention-based MAC. All control headers carry 28 bytes of preamble and frame check sequence (FCS), which are included to simulate synchronization between transmitter and receiver and error checking.

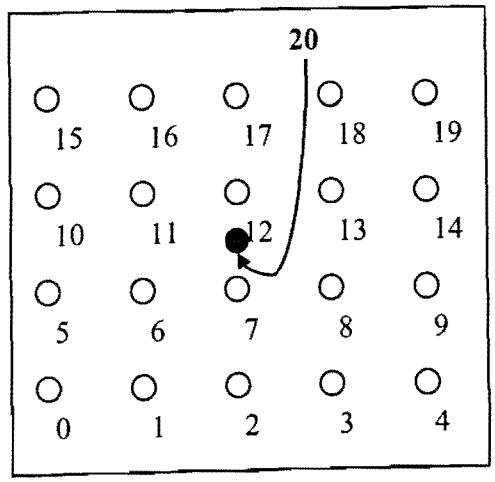

Fig. 5.1 Two-hop Topology with Centered Sink (shaded circle)

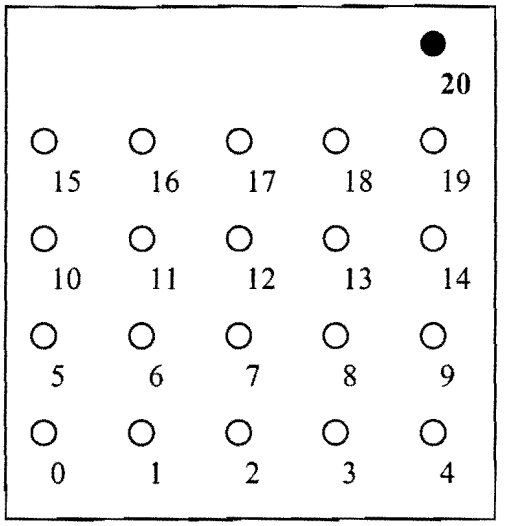

Fig. 5.2 Four-hop Topology with Cornered Sink (shaded circle)

The one-hop simulation creates a high contention level around the sink and places great pressure on the MAC protocol to resolve it. For the two-hop and four-hop simulations, packet generation rate is lowered to 1 packet/s per node but this is enough to cause congestion all over the network. The congestion is massive because nodes one-hop from the sink are now forwarders for other source nodes in the network, but these forwarding nodes also have their own data to send. To relief the congested network, VMAC utilizes channel reuse by allowing nodes three-hop from the transmitter to also transmit on the same time slot. This is possible because again the carrier-sensing range is twice the receiving range so if the receiver of a RTS senses another RTS transmission the slot reservation is still valid. An example is shown in Fig. 5.3, where two transmitters (clear circles) are contending for the same time slot and they transmit the RTS to their receivers (shaded circles). The dashed lines in the figure represent the sensing range, while the shaded lines represent the receiving or transmission range. The receivers can still receive a RTS properly while disregarding the sensed RTS. Since TDMA is a centralized scheduler with a global frame time, channel reuse is not possible. 802.11 with RTS/CTS exchange can also exploit channel reuse as well since it resolves the exposed node problem as discussed earlier in Chapter 3.

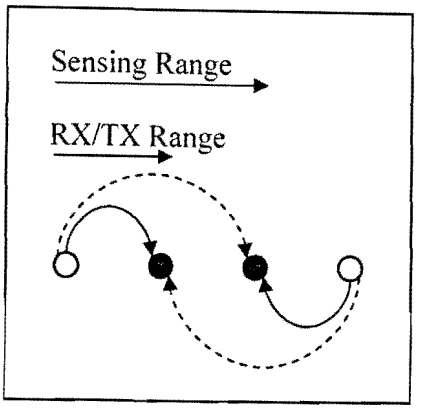

Fig. 5.3 Example of VMAC Channel Reuse

5.2 Simulation Results

The simulations are conducted with Windows Vista 64-bit Home Premium with Service Pack 2 running on the HP Pavilion dv6700 Notebook PC. The CPU is Intel Core 2 Duo with clock at 1.83 GHz and 4 GB of DDR2 memory. NS-2 is installed on and ran through Cygwin (a Unix-like environment) which runs on top of Vista. Data points are extracted from the logged trace files of each protocol and topology using OTcl scripts, and finally plotted with MATLAB.

5.2.1 Throughput

The throughput of the three simulated topologies is shown in Fig. 5.4 with various VMAC frame lengths (FLs). Throughput is defined here as a measure of the total amount of successful data transmission (data carried by constant bit rate (CBR) packets) after the topology is completely simulated. For the one-hop simulation, TDMA rises above 802.11 and VMAC since it utilizes a global frame time that is perfect in this topology where channel reuse is impossible; but TDMA does not simulate the exchange of receiver address as mentioned in the previous chapter. Therefore, a real implementation of TDMA will reduce its overall performance. VMAC has similar throughput to 802.11 since both have similar overhead such as RTS transmission and random backoff. The various FLs of VMAC differ by a small amount in throughput since reservation time is proportional to frame length, meaning the overhead is the same but longer frame lengths lead to less RTS collisions and therefore more throughput.

For the two-hop simulation, all protocols have similar throughput because again channel reuse cannot be utilized often, and TDMA suffers from having a global frame time when the average two-hop neighbourhood size is much less than 21. Also all protocols can take advantage of the sink being placed in the center of several forwarding nodes instead of being in the corner of the network. For the four-hop simulation, multi-hops are required to reach the sink node that is only reachable through two forwarding nodes. Channel reuse can be utilized here since forwarding near the sink is far from data generation in the opposite corner. Thus, VMAC with FL of 1 and 802.11 can maximize the throughput of this congested network, while others suffer from having large frame lengths. VMAC with a small frame length proves to be effective, in terms of throughput, throughout all three topologies.

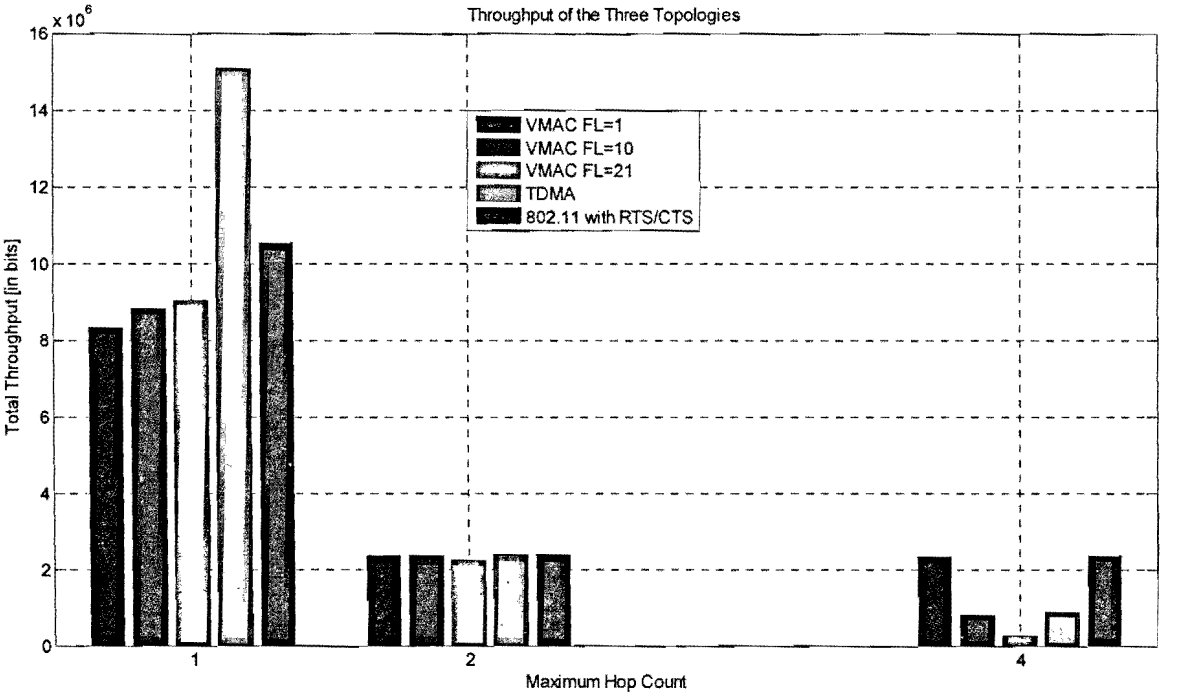

Fig. 5.4 Throughput of the Three Topologies

5.2.2 Utilization of Bandwidth

The utilization of bandwidth in the three topologies is shown in Fig. 5.5 with various VMAC FLs. Bandwidth utilization is a measure of the amount of bits transmitted and successfully received throughout the course of the simulation, and broadcasted packets such as ARP and AODV are counted as many times as they are received. Therefore, this is a measure of whether

For the two-hop simulation, all protocols have similar throughput because again channel reuse cannot be utilized often, and TDMA suffers from having a global frame time when the average two-hop neighbourhood size is much less than 21. Also all protocols can take advantage of the sink being placed in the center of several forwarding nodes instead of being in the corner of the network. For the four-hop simulation, multi-hops are required to reach the sink node that is only reachable through two forwarding nodes. Channel reuse can be utilized here since forwarding near the sink is far from data generation in the opposite corner. Thus, VMAC with FL of 1 and 802.11 can maximize the throughput of this congested network, while others suffer from having large frame lengths. VMAC with a small frame length proves to be effective, in terms of throughput, throughout all three topologies.

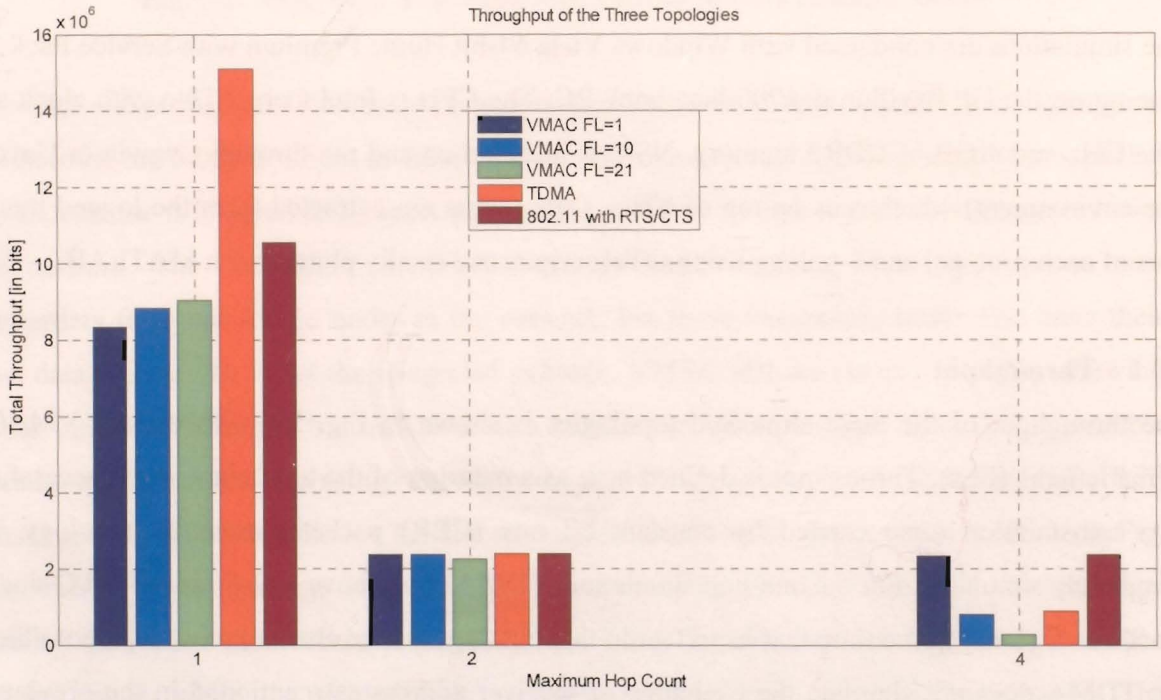

Fig. 5.4 Throughput of the Three Topologies

5.2.2 Utilization of Bandwidth

The utilization of bandwidth in the three topologies is shown in Fig. 5.5 with various VMAC FLs. Bandwidth utilization is a measure of the amount of bits transmitted and successfully received throughout the course of the simulation, and broadcasted packets such as ARP and AODV are counted as many times as they are received. Therefore, this is a measure of whether

the medium is exploited by the protocols. These bits could belong to data packets such as CBR or for hardware address and routing purposes such as ARP and AODV. Examining the bandwidth utilization and their throughput in the previous figure, it can be concluded that high bandwidth utilization can lead to the misconception of higher throughput. It also shows that high bandwidth utilization with low throughput is a result of constant ARP and AODV transmissions. Thus, a lot of important routing and address resolution packets must have been dropped in the forwarding process leading to more transmissions. Again, channel reuse can help lower this drop rate by lowering congestion.

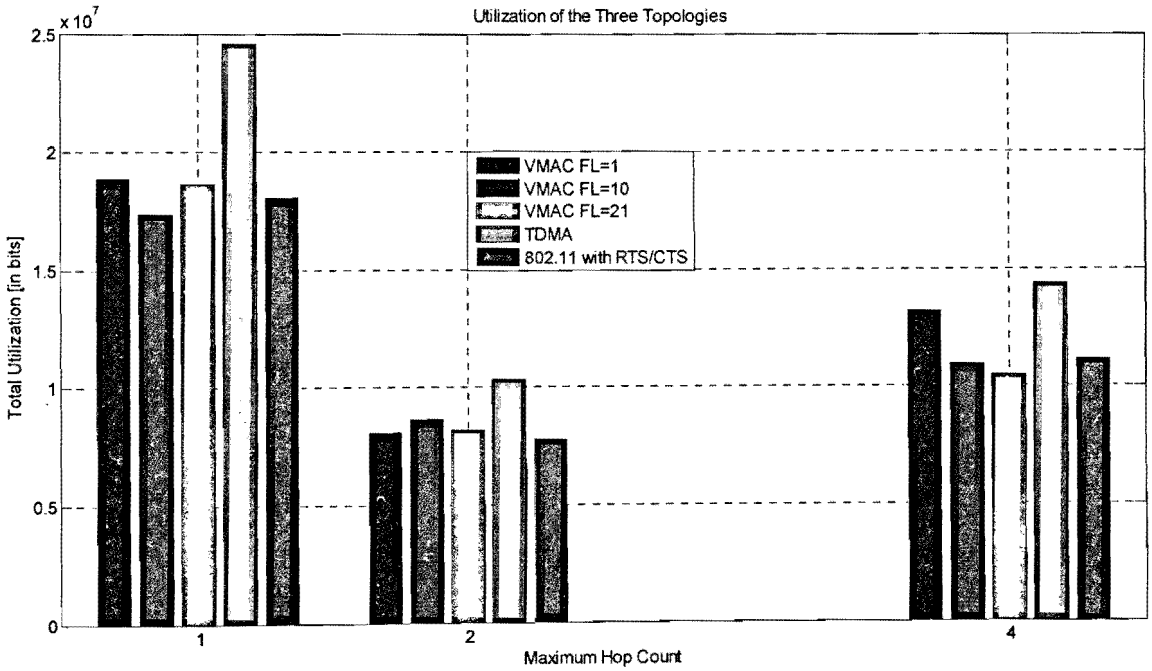

Fig. 5.5 Utilization of the Three Topologies

5.2.3 Source to Destination Delay

The average delay of packet delivery, from when the packet is first sent at the source to when the packet is received at the destination, is shown in Fig. 5.6. For the one-hop simulation, the scheduled-based protocols, VMAC and TDMA, have very minimal delay (near zero) since scheduled access prevents delay easily in one-hop scenarios. As the number of hops increase, scheduled access is no longer as fast as contention-based because unutilized long frame lengths cause delays to linger. Also inability to reuse the channel is a major disadvantage. Only VMAC

the medium is exploited by the protocols. These bits could belong to data packets such as CBR or for hardware address and routing purposes such as ARP and AODV. Examining the bandwidth utilization and their throughput in the previous figure, it can be concluded that high bandwidth utilization can lead to the misconception of higher throughput. It also shows that high bandwidth utilization with low throughput is a result of constant ARP and AODV transmissions. Thus, a lot of important routing and address resolution packets must have been dropped in the forwarding process leading to more transmissions. Again, channel reuse can help lower this drop rate by lowering congestion.

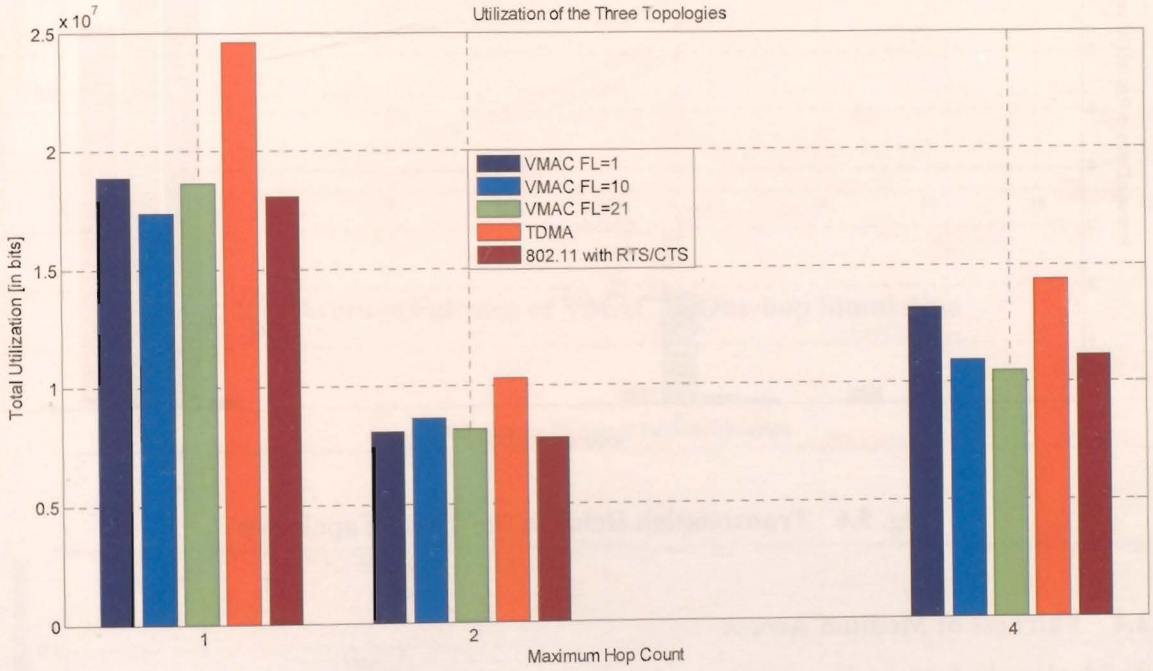

Fig. 5.5 Utilization of the Three Topologies

5.2.3 Source to Destination Delay

The average delay of packet delivery, from when the packet is first sent at the source to when the packet is received at the destination, is shown in Fig. 5.6. For the one-hop simulation, the scheduled-based protocols, VMAC and TDMA, have very minimal delay (near zero) since scheduled access prevents delay easily in one-hop scenarios. As the number of hops increase, scheduled access is no longer as fast as contention-based because unutilized long frame lengths cause delays to linger. Also inability to reuse the channel is a major disadvantage. Only VMAC

with FL of 1 behaves like the contention-based protocol of 802.11. VMAC with FL of 1 and 802.11 are very much alike since they both transmit RTS and utilize random backoff for only one data transmission. Furthermore, they are both capable of channel reuse which speeds up the rate of forwarding data.

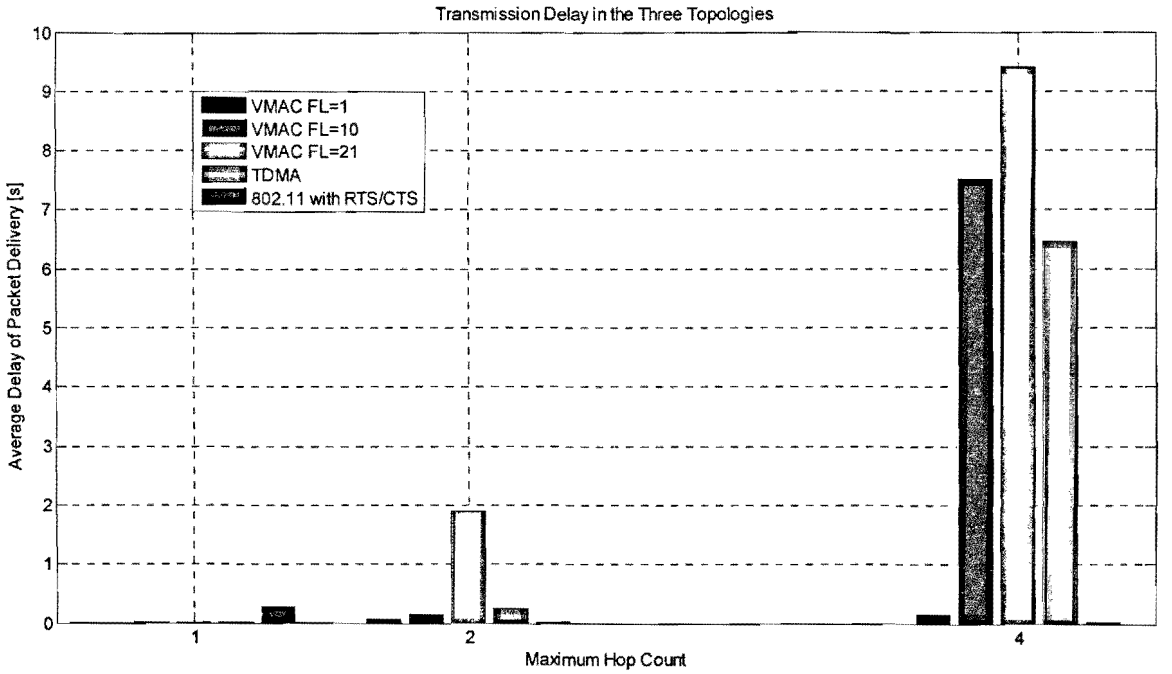

Fig. 5.6 Transmission Delay in the Three Topologies

5.2.4 Fairness of Medium Access

Individual node fairness is a measure of how successful a node is in accessing the medium when it makes an attempt, and the results from simulation of VMAC are shown in Fig. 5.7 to 5.9. For the one-hop (Fig. 5.7), a FL of 21 produces near unity fairness since the 21 nodes in the network have a really high chance of capturing a slot. Any FLs below 21 will have their fairness proportional to the FL divided by 21. A FL of 42 produces unity fairness since collisions may happen during a nodes attempt to capture a time slot, but having a FL way beyond the two-hop neighbourhood size makes those collisions negligible. For the two-hop and four-hop simulations, the average neighbourhood size is much less than 21, and fairness for FL of 1 is quite astonishing since it is able to reuse the channel.

with FL of 1 behaves like the contention-based protocol of 802.11. VMAC with FL of 1 and 802.11 are very much alike since they both transmit RTS and utilize random backoff for only one data transmission. Furthermore, they are both capable of channel reuse which speeds up the rate of forwarding data.

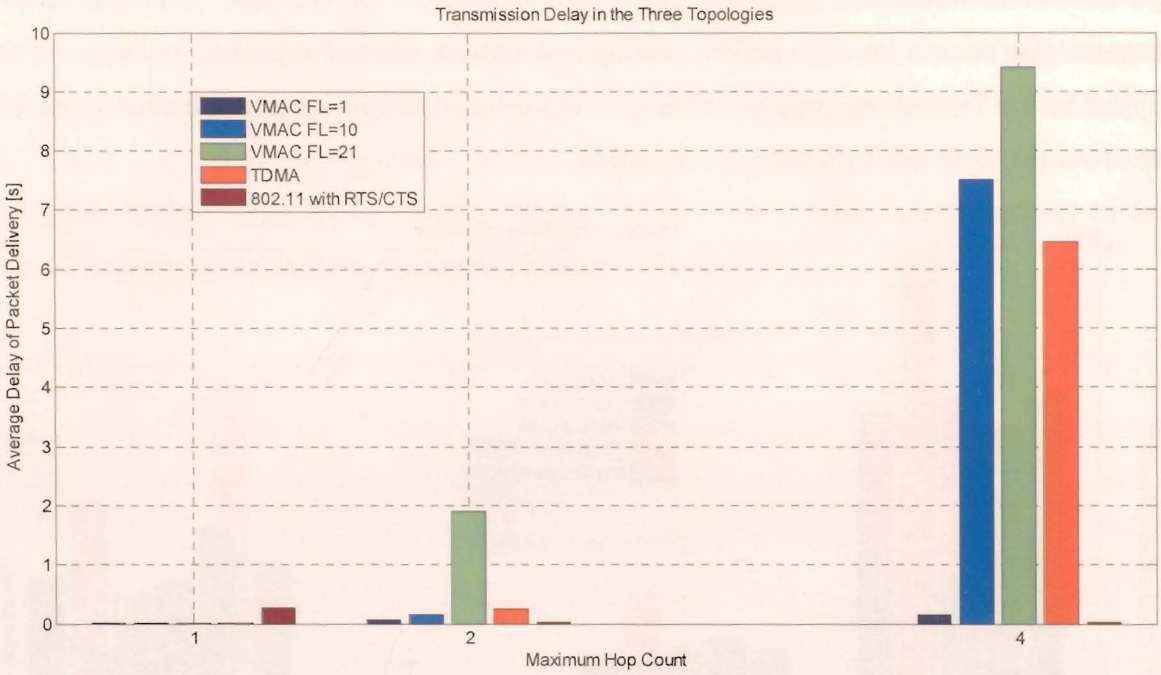

Fig. 5.6 Transmission Delay in the Three Topologies

5.2.4 Fairness of Medium Access

Individual node fairness is a measure of how successful a node is in accessing the medium when it makes an attempt, and the results from simulation of VMAC are shown in Fig. 5.7 to 5.9. For the one-hop (Fig. 5.7), a FL of 21 produces near unity fairness since the 21 nodes in the network have a really high chance of capturing a slot. Any FLs below 21 will have their fairness proportional to the FL divided by 21. A FL of 42 produces unity fairness since collisions may happen during a nodes attempt to capture a time slot, but having a FL way beyond the two-hop neighbourhood size makes those collisions negligible. For the two-hop and four-hop simulations, the average neighbourhood size is much less than 21, and fairness for FL of 1 is quite astonishing since it is able to reuse the channel.

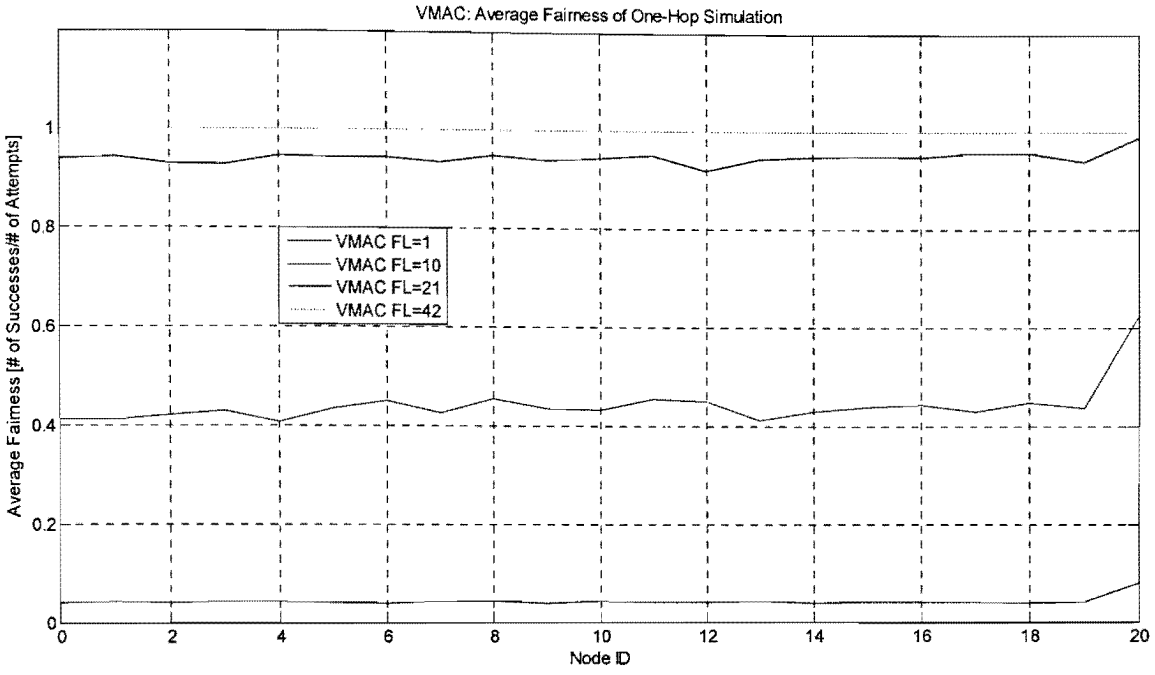

Fig. 5.7 Average Fairness of VMAC in One-hop Simulation

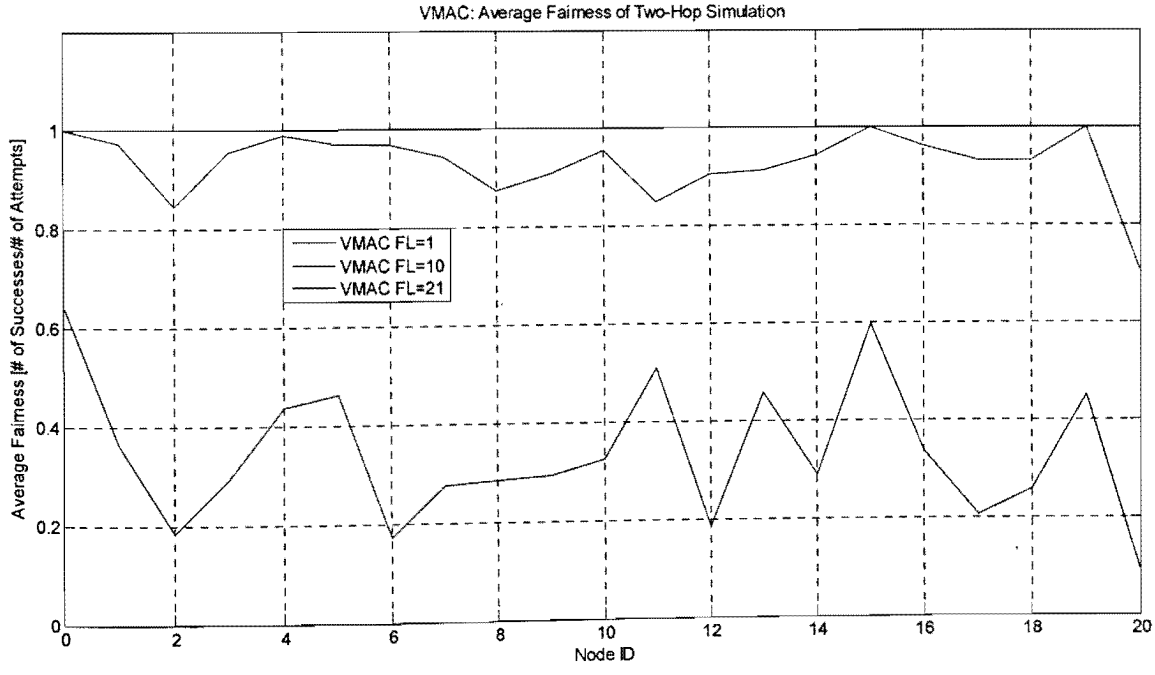

Fig. 5.8 Average Fairness of VMAC in Two-hop Simulation

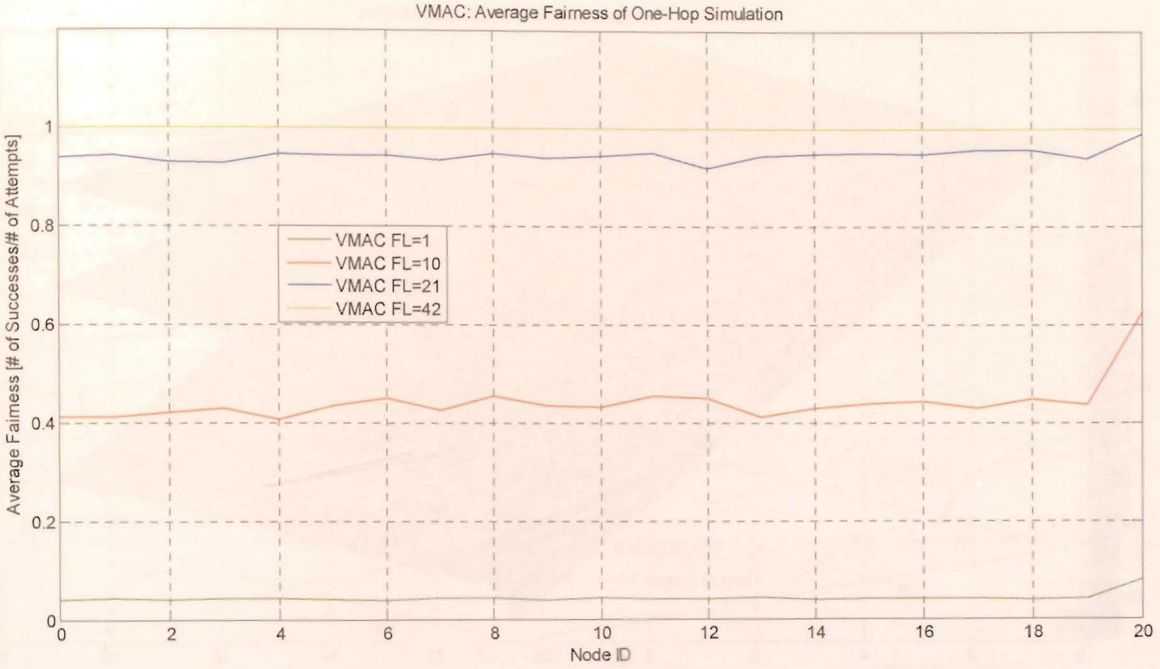

Fig. 5.7 Average Fairness of VMAC in One-hop Simulation

Fig. 5.8 Average Fairness of VMAC in Two-hop Simulation

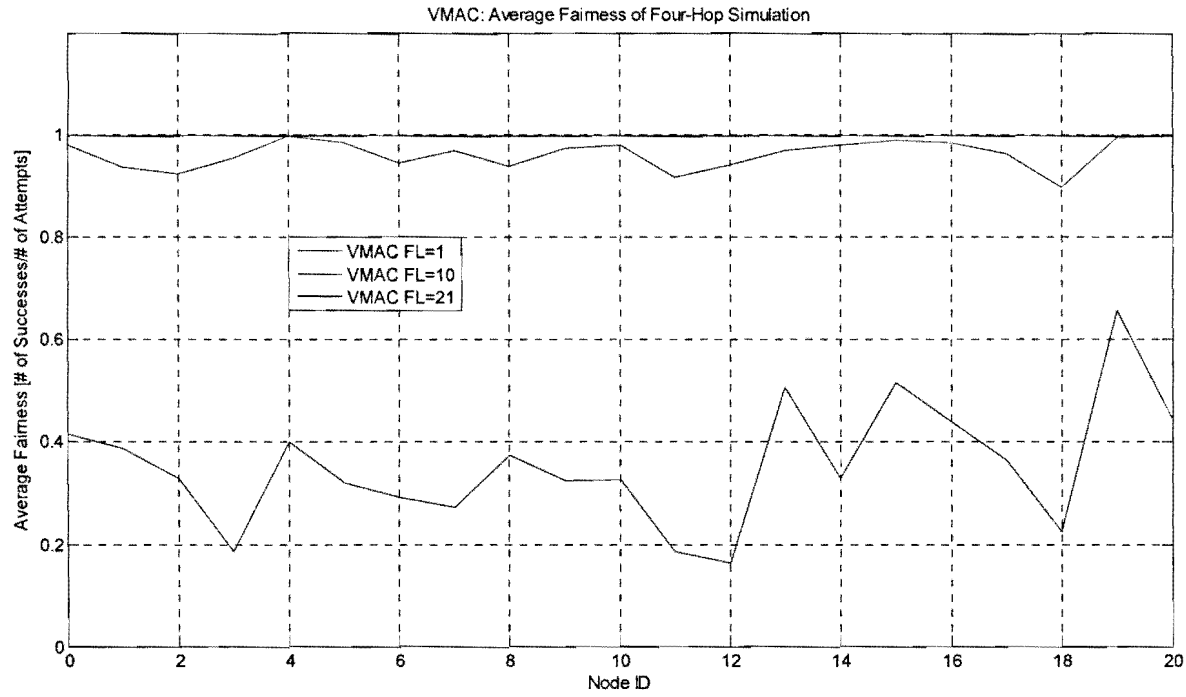

Fig. 5.9 Average Fairness of VMAC in Four-hop Simulation

5.2.5 Energy Consumption

Individual node energy consumption is traced throughout the course of the simulation by monitoring its current energy remaining, and the results from simulations are shown in Fig. 5.10 to 5.12. The energy consumption level for the one-hop simulation (Fig 5.10) is quite stable and the sink (Node 20) has the lowest energy remaining in all protocols and topologies since it is always on and participating in transmission. All three figures show that even though 802.11 performs well in all evaluations, its biggest drawback is not utilizing sleep in energy reduction. On the other hand, the two-hop and four-hop simulation results are wavy due to some nodes being forwarders and consuming more energy than others. Also, decrease in FL of VMAC leads to a small increase in energy consumption since smaller FLs transition more from sleep to active, which consumes a negligible amount of energy.

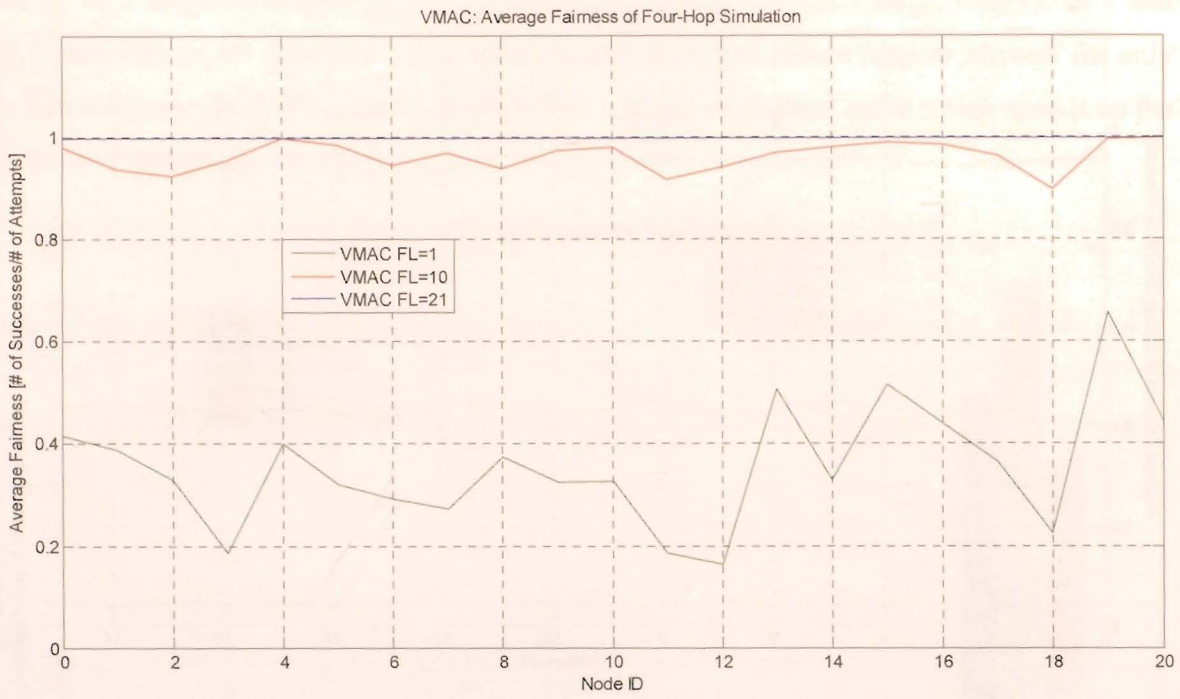

Fig. 5.9 Average Fairness of VMAC in Four-hop Simulation

5.2.5 Energy Consumption

Individual node energy consumption is traced throughout the course of the simulation by monitoring its current energy remaining, and the results from simulations are shown in Fig. 5.10 to 5.12. The energy consumption level for the one-hop simulation (Fig 5.10) is quite stable and the sink (Node 20) has the lowest energy remaining in all protocols and topologies since it is always on and participating in transmission. All three figures show that even though 802.11 performs well in all evaluations, its biggest drawback is not utilizing sleep in energy reduction. On the other hand, the two-hop and four-hop simulation results are wavy due to some nodes being forwarders and consuming more energy than others. Also, decrease in FL of VMAC leads to a small increase in energy consumption since smaller FLs transition more from sleep to active, which consumes a negligible amount of energy.

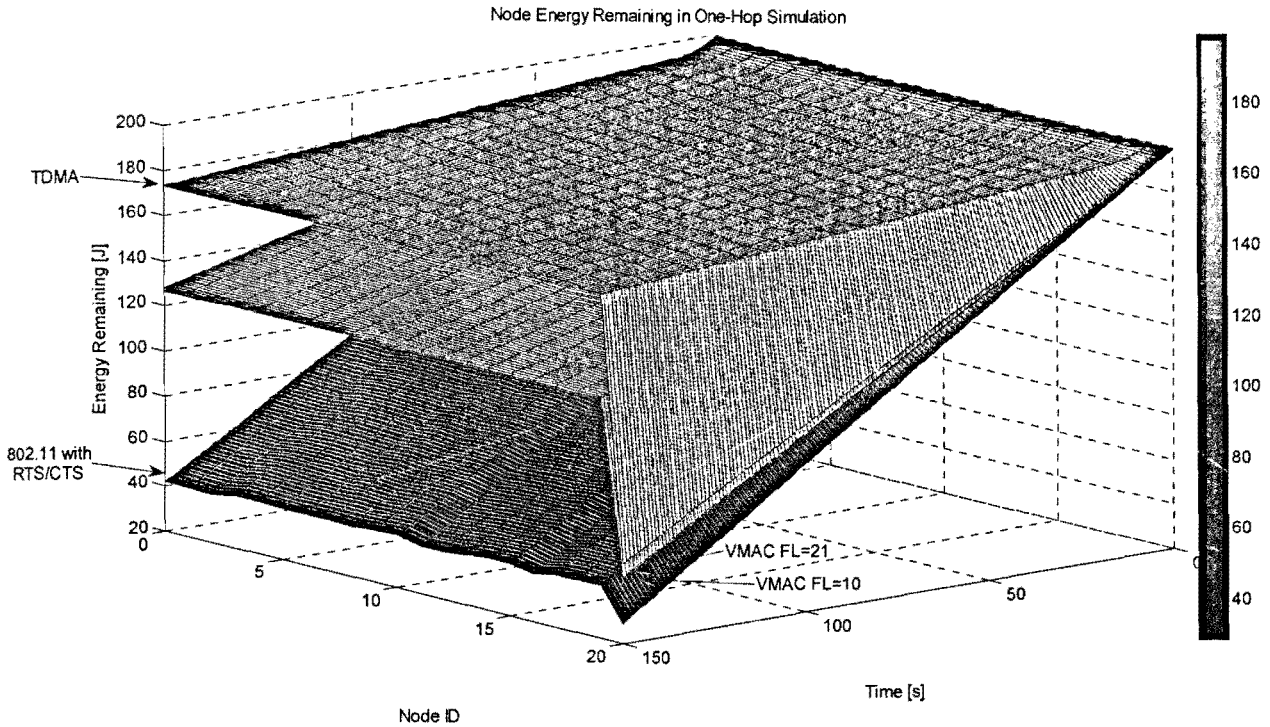

Fig. 5.10 Node Energy Remaining in One-hop Simulation

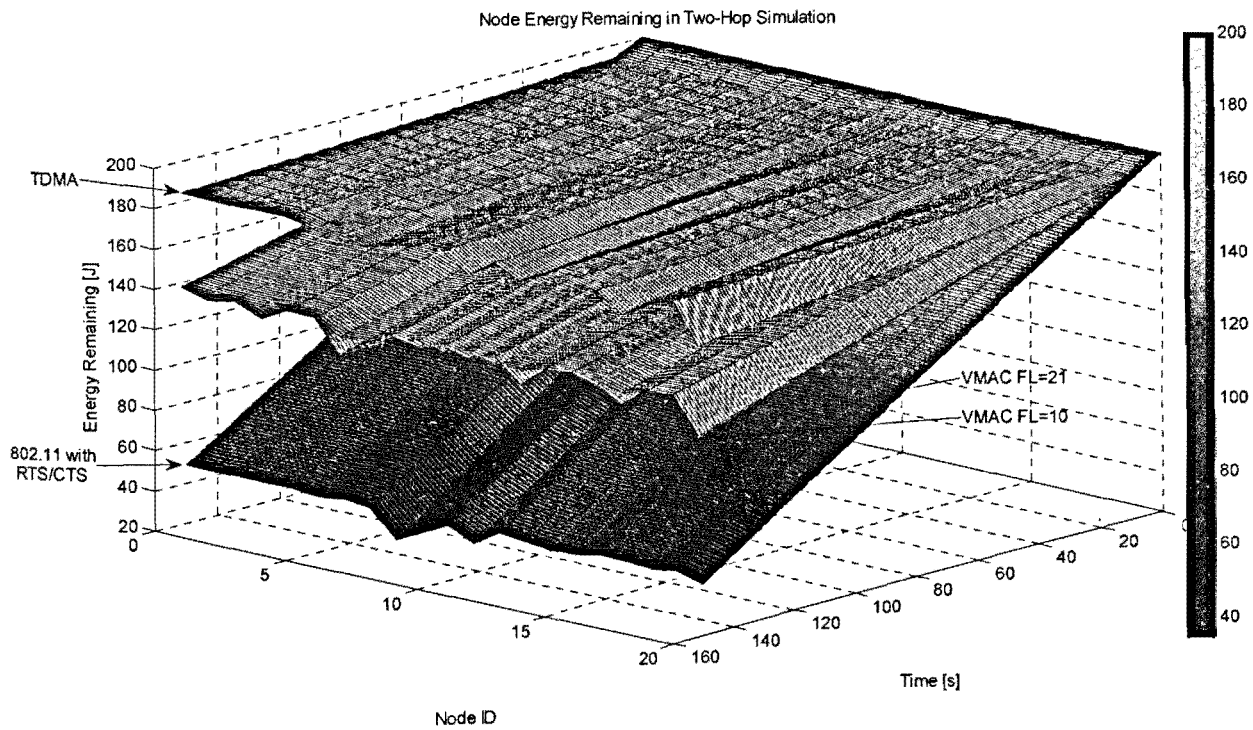

Fig. 5.11 Node Energy Remaining in Two-hop Simulation

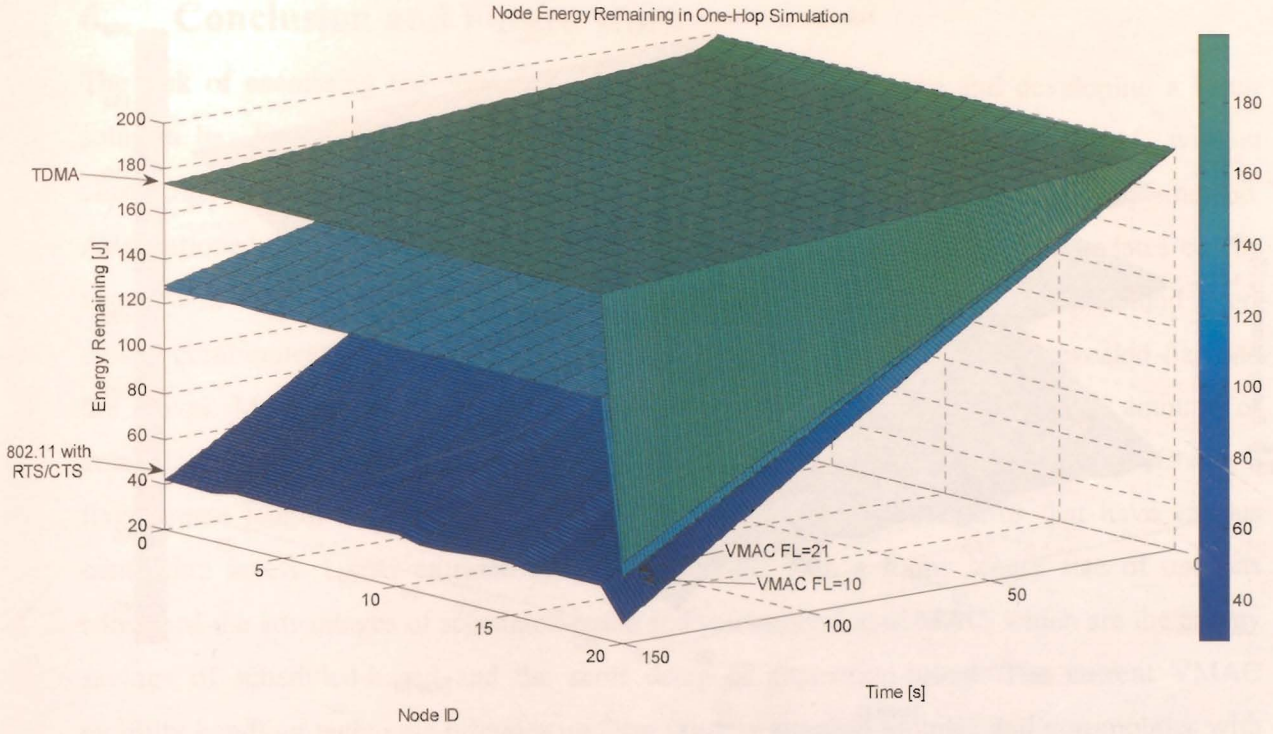

Fig. 5.10 Node Energy Remaining in One-hop Simulation

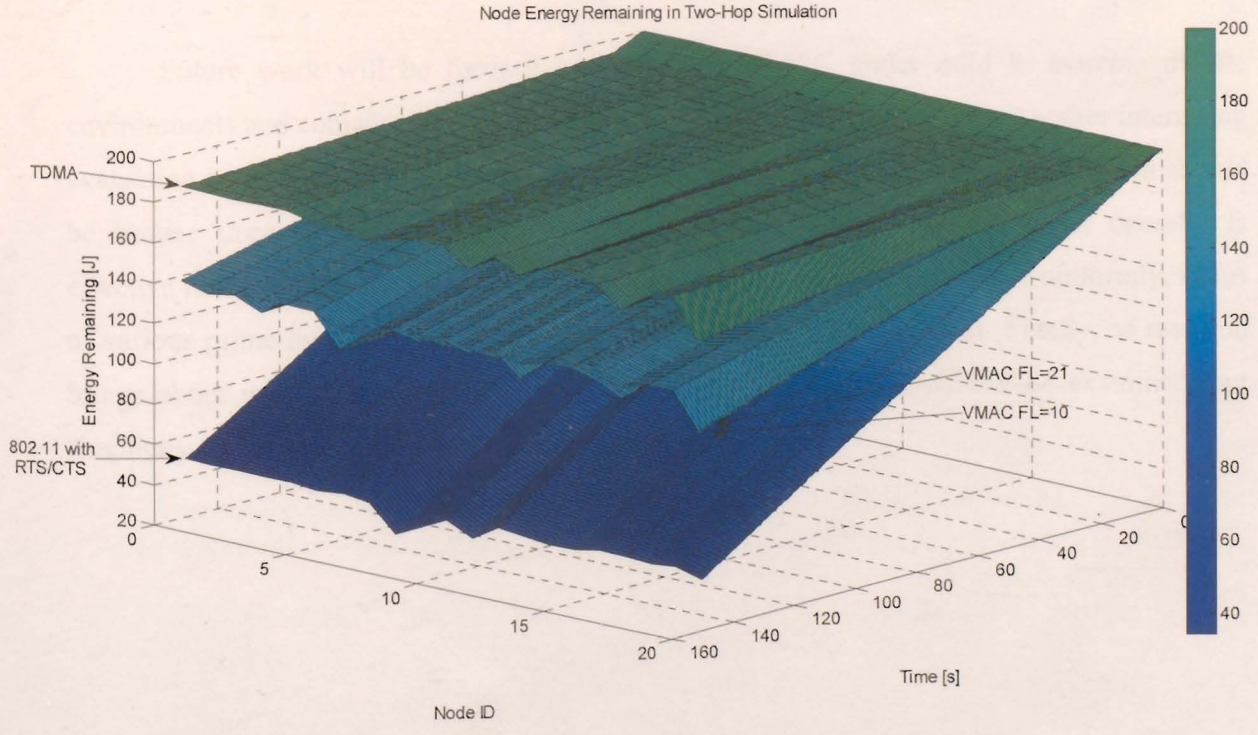

Fig. 5.11 Node Energy Remaining in Two-hop Simulation

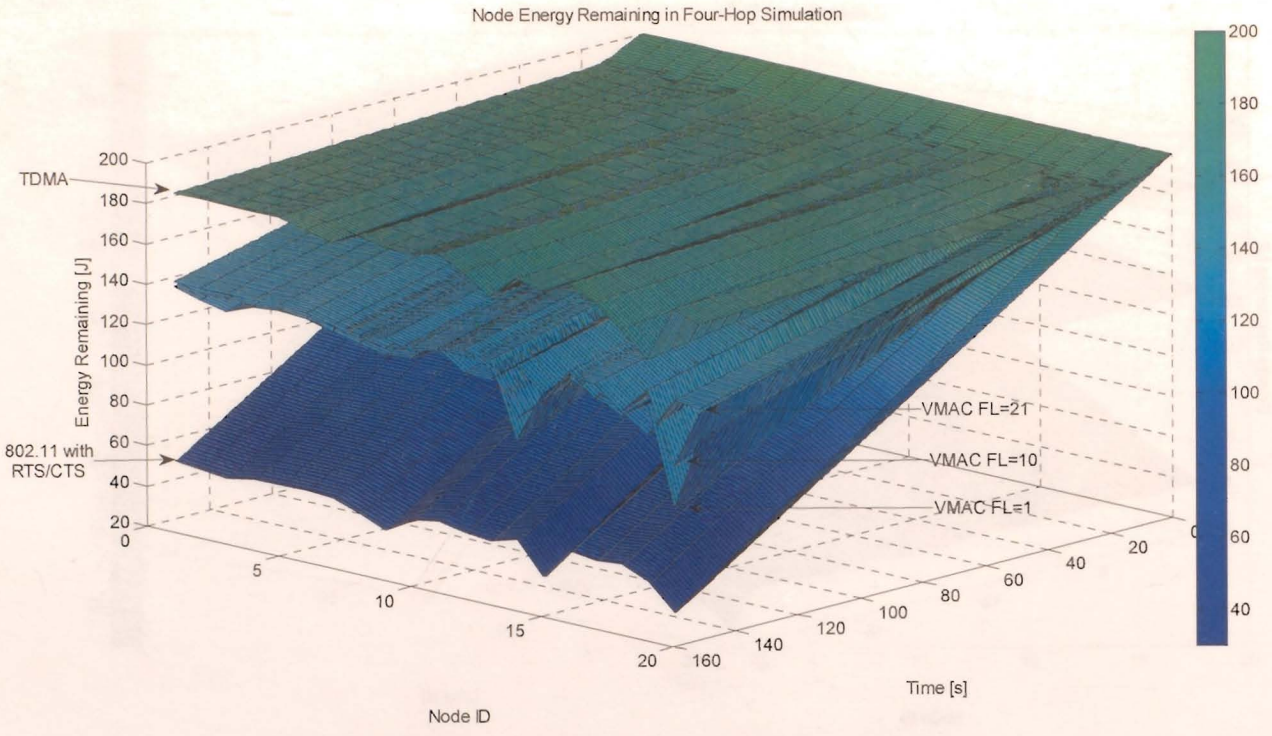

Fig. 5.12 Node Energy Remaining in Four-hop Simulation

6. Conclusion and Future Work

The task of examining the current state of mobile sensor networks and developing a better solution has been fulfilled with the design of VMAC. The simplicity of VMAC without clustering and constant neighbourhood update is appreciated especially when it is implemented. As mentioned in Chapter 3, protocols based on clustering and neighbourhood updates can be vague with their ideas on inter-clustering interference, message passing overhead, and inter-cluster communication. Reverting back to the original purpose of MAC has revealed flaws to the novel MAC protocols. Complex solutions generally require a greater amount of computation time and energy consumption, which sensor networks cannot risk. VMAC with a fixed frame length has been shown to be effective in different scenarios that have various contention levels. Under extreme congestion, VMAC with a frame length size of one has combined the advantages of scheduled-based and contention-based MAC, which are the energy savings of scheduled-based and the short delay of contention-based. The current VMAC mobility handling technique provides uniform fairness amongst mobiles and non-mobiles with just a remainder operator. The mobility handling technique is useful even in the stationary case because ARP and AODV can utilize it to quickly route packets.

Future work will be focused on evaluating VMAC under mild to extreme mobile environments and compare its performance to other hybrid MAC protocols. Another interesting evaluation will be based on individual throughput rather than overall throughput, which should be similar since the random backoff is fair. Also, even though the remainder operator is effective in having the backoff fall into any number in the contention window uniformly, ratios of various prime numbers to two-hop node density can be studied further. Finally, as research brings about new ideas, integration of these innovations into VMAC can be examined, but simplicity and efficiency are definitely the goals.

References

[1] R. Shorey, A. Ananda, M.C. Chan, and W.T. Ooi, *Mobile, Wireless, and Sensor Networks: Technology, Applications, and Future Directions*, John Wiley & Sons, Inc., Hoboken, New Jersey, 2006.

[2] J. A. Stankovic, T. F. Abdelzaher, C. Lu, L. Sha, and J. C. Hou, "Real-Time Communication and Coordination in Embedded Sensor Networks," *Proceedings of the IEEE*, vol. 91, no. 7, July 2003.

[3] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," *ACM Int. Conf. on Mobile Computing and Networking (Mobicom)*, pp. 56-67, August 2000.

[4] A. Kansal, M. Rahimi, D. Estrin, W.J. Kaiser, G.J. Pottie, and M. B. Srivastava, "Controlled Mobility for Sustainable Wireless Sensor Networks," *IEEE Conf. on Sensor and Ad Hoc Communications and Networks (SECON 2004)*, pp. 1-6, October 2004.

[5] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.S. Peh, and D. Rubenstein, "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet," *ACM SIGOPS Operating Systems Review*, vol. 36, no. 5, pp. 96-107, December 2002.

[6] H. Wu, Y. Wang, H. Dang, and F. Lin, "Analytic, Simulation, and Empirical Evaluation of Delay/Fault-Tolerant Mobile Sensor Networks," *IEEE Trans. on Wireless Communication*, vol. 6, no. 9, pp. 3287-3296, September 2007.

[7] W. Ye and J. Heidemann, "Medium Access Control in Wireless Sensor Networks," *USC Information Sciences Institute, Technical Report*, October 2003.

[8] W. Ye, J. Heidemann, and D. Estrin, "Medium Access Control With Coordinated Adaptive Sleeping for Wireless Sensor Networks," *IEEE/ACM Trans. on Networking*, vol. 12, no. 3, pp. 493-506, June 2004.

[9] K. Akkaya and M. Younis, "A Survey on Routing Protocols for Wireless Sensor Networks," in the *Elsevier Ad Hoc Networks Journal*, vol. 3, issue 3, pp. 325-349, May 2005.

[10] IEEE-SA Standards Board, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band," *IEEE*, New York, September 1999.

[11] J. Postel, "User Datagram Protocol," *RFC 768*, USC/Information Sciences Institute, August 1980.

[12] J. Postel (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," *RFC 791*, USC/Information Sciences Institute, September 1981.

[13] J. Postel, "Transmission Control Protocol," STD 7, *RFC 793*, September 1981.

[14] The Network Simulator Homepage [Online]. Available: http://www.isi.edu/nsnam/ns/

[15] The Network Simulator Manual [Online]. Available: http://www.isi.edu/nsnam/ns/ns-documentation.html

[16] P. Raviraj, H. Sharif, M. Hempel, S. Ci, H.H. Ali, and J. Youn, "A Mobility Based Link Layer Approach for Mobile Wireless Sensor Networks," *IEEE Int. Conf. on Electro Information Technology*, pp. 1-6, May 2005.

[17] T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed., Prentice Hall PTR, New Jersey, 2002.

[18] H. Pham and S. Jha, "Addressing Mobility in Wireless Sensor Media Access Protocol," *Intelligent Sensors, Sensor Networks and Information Processing Conf. (ISSNIP 2004)*, pp. 113-118, December 2004.

[19] M. Ali, T. Suleman, and Z.A. Uzmi, "MMAC: A Mobility-Adaptive, Collision-Free MAC Protocol for Wireless Sensor Networks," *IEEE International Performance, Computing, and Communications Conference (IPCCC 2005)*, pp. 401-407, April 2005.

[20] V. Rajendran, K. Obraczka, and J.J. Garcia-Luna-Aceves, "Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks," *ACM Int. Conf. on Embedded Networked Sensor Systems (SenSys 2003)*, pp. 181-192, November 2003.

[21] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A Media Access Protocol for Wireless LAN's," *ACM Special Interest Group on Data Communications Conf. (SIGCOMM 1994)*, vol. 24, issue 4, pp. 212-225, October 1994.

[22] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," *Annual Hawaii Int. Conf. on System Sciences*, vol. 2, pp. 1-10, January 2000.

[23] M. I. Brownfield, K. Mehrjoo, A. S. Fayez, and N. J. Davis IV, "Wireless Sensor Network Energy-Adaptive MAC Protocol," *IEEE Consumer Communications and Networking Conf. (CCNC 2006)*, vol. 2, pp. 778-782, January 2006.

[24] T. van Dam and K. Langendoen, "An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks," *ACM Int. Conf. on Embedded Networked Sensor Systems (SenSys 2003)*, pp. 171-180, November 2003.

[25] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," *ACM Int. Conf. on Embedded Networked Sensor Systems (SenSys 2004)*, pp. 95-107, November 2004.

[26] I. Rhee, A. Warrier, M. Aia, and J. Min, "Z-MAC: a Hybrid MAC for Wireless Sensor Networks," *ACM Int. Conf. on Embedded Networked Sensor Systems (SenSys 2005)*, pp. 90-101, November 2005.

[27] I. Rhee, A. Warrier, and L. Xu, "Randomized Dining Philosophers to TDMA Scheduling in Wireless Sensor Networks," *North Carolina State University, Computer Science Department, Technical Report*, 2004.

[28] W. Ye, J. Heidemann, and D. Estrin, "A Flexible and Reliable Radio Communication Stack on Motes," *USC Information Sciences Institute, Technical Report*, September 2002.

Appendix: OTcl and C++ Codes

VMAC OTcl One-hop Simulation File

OTcl simulation files of other hops and TDMA and 802.11 are similar to this file shown below. Differences are only minor changes to parameters and input and output file names.

```
# Energy model parameters# VMAC simulation of 21 nodes within one hop from each other

# Node parameters
set val(chan)           Channel/WirelessChannel  ; # channel type
set val(prop)           Propagation/TwoRayGround ; # signal propagation model
set val(ant)            Antenna/OmniAntenna      ; # antenna type
set val(ll)             LL                       ; # link layer type
set val(ifq)            Queue/DropTail/PriQueue  ; # priority queue
set val(ifqlen)         50                       ; # max number of packets in queue
set val(netif)          Phy/WirelessPhy          ; # network interface type
set val(mac)            Mac/Vmac                 ; # MAC type
set val(rp)             AODV                     ; # ad-hoc routing protocol
set val(nn)             21                       ; # number of nodes


set val(energymodel)   EnergyModel       ;
set val(radiomodel)       RadioModel     ;
set val(initialenergy) 200          ; # Initial energy in Joules (Joule = Power x Time)

# Write initial energy to file for reading later by Energy Tcl
set iEnergyFile [open "~/metricTcl/dirOutFiles/iEnergyOneVmac.out" "w+"]
puts $iEnergyFile "$val(initialenergy)"
close $iEnergyFile

# Priority is given to routing protocols (AODV, DSR, TORA, and MESSAGE) by default if priority queue is used.
# This can be modified in ~/queue/priqueue.cc
Queue/DropTail/PriQueue set Prefer_Routing_Protocols    0 ; # ARP should be more important than routing packets

# 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
# NOTE: Although TwoRayGround is assigned, Friis free space is used since
#  the frequency is 914e+6
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 9.1538e-11 ; # for 80m carrier sensing range
Phy/WirelessPhy set RXThresh_ 3.652e-10  ; # for 40m Xmission range using Friis free
```

```
Phy/WirelessPhy set bandwidth_ 0.2Mb       ; # in bits per second
# Phy/WirelessPhy set bandwidth_ 2e4       ; # in bits per second

Phy/WirelessPhy set Pt_ 8.5872e-4  ; # Transmit power in watts
Phy/WirelessPhy set freq_ 914e+6   ; # Carrier frequency
Phy/WirelessPhy set L_ 1.0

Mac set bandwidth_ 0.2Mb ; # Change 2Mb default in ns-mac.tcl (should be same as Phy bandwidth_)
# Mac set bandwidth_ 2e4

# Protocol specific variables
Mac/Vmac set slot_packet_len_ 173          ; # CBR 100B + IP 20B + VMAC 52B = 172B but 173B needed
Mac/Vmac set max_node_num_ $val(nn)
Mac/Vmac set schedule_length_ 1

# Create instance of simulator and setup trace files
set ns [new Simulator]
set namfd [open oneHopVMAC.nam w]
$ns namtrace-all-wireless $namfd 120 120
set tracefd [open oneHopVMAC.tr w]
$ns use-newtrace ; # For new trace format for wireless traces (see 16.1.7 in manual)
$ns trace-all $tracefd

# Create and configure topography of environment or boundaries
#  in metres wide (X) and metres long (Y)
set topo [new Topography]
$topo load_flatgrid 120 120

# Create god instance which is used by MAC objects in this case
create-god $val(nn)

# Configure nodes with parameters
$ns node-config  -adhocRouting $val(rp) \
        -llType $val(ll) \
        -macType $val(mac) \
        -ifqType $val(ifq) \
        -ifqLen $val(ifqlen) \
        -antType $val(ant) \
        -propType $val(prop) \
        -phyType $val(netif) \
        -topoInstance $topo \
        -channel [new $val(chan)] \
        -agentTrace OFF \
        -routerTrace OFF \
        -macTrace ON \
        -energyModel $val(energymodel) \
        -idlePower 1.0 \
        -rxPower 1.0 \
        -txPower 2.0 \
    -sleepPower 0.001 \
    -transitionPower 0.1 \
    -transitionTime 0.001 \
        -initialEnergy $val(initialenergy)
# Transition time may cause problems in synchronization of simulation
# Transition should only involve sleep to idle (active)
#  -> check this http://www.isi.edu/ilense/software/smac/ns2_energy.html
```

```
# Create an array of nodes with no random motion since nodes are stationary in this case
for {set i 0} {$i < $val(nn)} {incr i} {
        set node_($i) [$ns node]
        $node_($i) random-motion 0
        set mac_($i) [$node_($i) set mac_(0)]
}

source "./1-hop-topology.tcl"

set null_(0) [new Agent/Null]
$ns attach-agent $node_(20) $null_(0)

for {set i 0} {$i < 20} {incr i} {
 set udp_($i) [new Agent/UDP]
 $ns attach-agent $node_($i) $udp_($i)
 $ns connect $udp_($i) $null_(0)

 set cbr_($i) [new Application/Traffic/CBR]
 $cbr_($i) set packetSize_ 100              ; # Default UDP maximum segment size is 1000 bytes
 $cbr_($i) set interval_ 0.1
 $cbr_($i) set random_ 0
 $cbr_($i) attach-agent $udp_($i)

 # $ns at 0 "$cbr_($i) start"
 $ns at [expr $i * .05] "$cbr_($i) start"
 # $ns at 50 "$cbr_($i) set interval_ 0.1" ; # Interval calculated to approach max capacity
 $ns at 150 "$cbr_($i) stop"

}

for {set i 0} {$i < 21 } {incr i} {
 $ns at 151 "$node_($i) reset"
 $ns at 151 "$mac_($i) print-stat"
}

$ns at 151.0 "finish"

# Finish procedure to flush and close all traces
proc finish {} {
        global ns namfd tracefd
        $ns flush-trace
        close $namfd
        close $tracefd
        exit 0
}

puts "Simulation Begins..."
$ns run
```

VMAC C++ Header File

```
/*
```

```
* File:   vmac.h
* Author: Vincent Ngo
* Summary: Created with mods to mac-tdma.h to add declarations of variables and
* functions for priority slot reservation
*/

#ifndef _VMAC_H
#define  _VMAC_H

#include "marshall.h"
#include <delay.h>
#include <connector.h>
#include <packet.h>
#include <random.h>
#include <arp.h>
#include <ll.h>
#include <mac.h>

// Vince: For time(NULL) which returns current time in seconds used in seeding
// random number generator
#include <time.h>

// Vince: VMAC slot time based on DSSS_SlotTime
//#define VMAC_SlotTime 0.000010  // in seconds
#define VMAC_SlotTime   0.000020  // in seconds

// Vince: Variable added to turn priority scheduling ON/OFF (can be binded to OTcl)
#define PRIORITY        1

// Vince: Congestion window range for each type of packet (must be prime)
#define PRIORITY_CW     61
#define REGULAR_CW      61
#define TOTAL_CW        (PRIORITY_CW + REGULAR_CW)

// Vince: DATA transmission time when given length in bytes
#define DATA_Time(len)  (8 * (len) / bandwidth_)

// Vince: Reservation time (RTS packet with MAC header is only 40 bytes)
#define RESERVE_TIME \
(((TOTAL_CW-1) * VMAC_SlotTime * schedule_length_) + (DATA_Time(40) * schedule_length_))

// Vince: Length of time allowed for each reservation slot
#define RESERVE_SLOT_TIME   RESERVE_TIME/schedule_length_

// Vince: Same specs for PHY layer as 802.11
// IEEE 802.11 Spec, section 15.3.2 - default values for the DSSS PHY MIB
#define DSSS_CWMin                      31
#define DSSS_CWMax                      1023
#define DSSS_SlotTime                   0.000020        // 20us
#define  DSSS_CCATime                   0.000015        // 15us
#define DSSS_RxTxTurnaroundTime                 0.000005        // 5us
#define DSSS_SIFSTime                   0.000010        // 10us
#define DSSS_PreambleLength             144             // 144 bits (18 bytes)
#define DSSS_PLCPHeaderLength           48              // 48 bits  (6 bytes)

// Vince: struct hdr_mac_vmac header size is 26 bytes minus 2 bytes for body
```

```
// offset, Preamble is 18, PLCP is 6, FCS is 4 = Ethernet header length of 52 bytes
#define ETHER_HDR_LEN                                   \
   ((phymib_->PreambleLength >> 3) +       \
   (phymib_->PLCPHeaderLength >> 3) +   \
   offsetof(struct hdr_mac_vmac, dh_body[0] ) + \
   ETHER_FCS_LEN)

// Vince: Frame control is 2 bytes in size (same header structure as 802.11)
struct frame_control {
   u_char          fc_subtype              : 4;
   u_char          fc_type                 : 2;
   u_char          fc_protocol_version     : 2;

   u_char          fc_order                : 1;
   u_char          fc_wep                  : 1;
   u_char          fc_more_data  : 1;
   u_char          fc_pwr_mgt              : 1;
   u_char          fc_retry                : 1;
   u_char          fc_more_frag  : 1;
   u_char          fc_from_ds              : 1;
   u_char          fc_to_ds                : 1;
};

// Vince: VMAC header is 26 bytes
struct hdr_mac_vmac {
   struct frame_control   dh_fc;              // 2 bytes
   u_int16_t          dh_duration;          // 2 bytes
   u_char             dh_da[ETHER_ADDR_LEN];     // 6 bytes
   u_char             dh_sa[ETHER_ADDR_LEN];     // 6 bytes
   u_char             dh_bssid[ETHER_ADDR_LEN];  // 6 bytes
   u_int16_t          dh_scontrol;          // 2 bytes
   u_char             dh_body[1]; // XXX Non-ANSI // 2 bytes
};

// Vince: RTS frame structure is 20 bytes but 16 if FCS (frame checksum) is not used
struct rts_frame {
   struct frame_control   rf_fc;              // 2 bytes
   u_int16_t          rf_duration;          // 2 bytes
   u_char             rf_ra[ETHER_ADDR_LEN]; // 6 bytes
   u_char             rf_ta[ETHER_ADDR_LEN]; // 6 bytes
   u_char             rf_fcs[ETHER_FCS_LEN]; // 4 bytes
};

// Vince: Not used in any function except constructor
class PHY_MIB {
public:
   u_int32_t  CWMin;
   u_int32_t  CWMax;
   double          SlotTime;
   double          CCATime;
   double          RxTxTurnaroundTime;
   double          SIFSTime;
   u_int32_t       PreambleLength;
   u_int32_t       PLCPHeaderLength;

   inline u_int32_t getPLCPhdrLen() {
```

```
    return((DSSS_PreambleLength + DSSS_PLCPHeaderLength) >> 3);
  }

  inline u_int32_t getRTSlen() {
    return(getPLCPhdrLen() + sizeof(struct rts_frame));
  }
};

/*
 * Vince: Frame format parameters
 */
#define  MAC_ProtocolVersion    0x00
#define MAC_Type_Data           0x02
#define MAC_Subtype_Data        0x00

// Vince: Added for RTS reservation
#define MAC_Subtype_RTS                 0x0B
#define MAC_Type_Control        0x01

// Vince: For initializing all Tx/Rx time slots to start at unreserved status
#define NOTHING_TO_SEND       -2

// Vince: For initializing slot count to first round so reservation slot begins
#define FIRST_ROUND           -1

// Vince: Turn radio ON/OFF
#define ON              1
#define OFF             0

// Vince: Quoted from MAC-802.11
#define DATA_DURATION         5

// Vince: May be needed later for caching
class Host {
public:
  LIST_ENTRY(Host)   link;
  u_int32_t         index;
  u_int32_t         seqno;
};

// Vince: VMAC class declaration
class Vmac;

// Vince: VMAC timer definition
class VmacTimer : public Handler {
public:
  VmacTimer(Vmac* m, double s = 0) : mac(m) {
    busy_ = paused_ = 0; stime = rtime = 0.0; slottime_ = s;
  }

  virtual void handle(Event *e) = 0;

  virtual void start(Packet *p, double time);
  virtual void stop(Packet *p);
  virtual void pause(void) { assert(0); }
  virtual void resume(void) { assert(0); }
```

```
    inline int busy(void) { return busy_; }
    inline int paused(void) { return paused_; }
    inline double slottime(void) { return slottime_; }
    inline double expire(void) {
      return ((stime + rtime) - Scheduler::instance().clock());
    }

protected:
    Vmac        *mac;
    int         busy_;
    int         paused_;
    Event intr;
    double      stime; // Start time
    double      rtime; // Remaining time
    double      slottime_;
};

// Vince: Slot Timer to switch slot length accordingly
class SlotVmacTimer : public VmacTimer {
public:
    SlotVmacTimer(Vmac *m) : VmacTimer(m) {}
    void handle(Event *e);
};

// Vince: Receive Packet Timer to reset parameters after a packet is received
class RxPktVmacTimer : public VmacTimer {
public:
    RxPktVmacTimer(Vmac *m) : VmacTimer(m) {}
    void handle(Event *e);
};

// Vince: Transmit Packet Timer to reset parameters after a packet is transmitted
class TxPktVmacTimer : public VmacTimer {
public:
    TxPktVmacTimer(Vmac *m) : VmacTimer(m) {}
    void handle(Event *e);
};

// Vince: RTS Backoff Timer to sync reservation slots and provide proper backoff
class RTSBackoffTimer : public VmacTimer {
public:
    RTSBackoffTimer(Vmac *m) : VmacTimer(m) {}
    void handle(Event *e);
};

// Vince: RTS Transmit Timer to reset parameters after a RTS is transmitted
class RTSTxTimer : public VmacTimer {
public:
    RTSTxTimer(Vmac *m) : VmacTimer(m) {}
    void handle(Event *e);
};

// Vince: VMAC definition
class Vmac : public Mac {
    // Vince: Timer classes
```

```
    friend class SlotVmacTimer;
    friend class TxPktVmacTimer;
    friend class RxPktVmacTimer;
    friend class RTSBackoffTimer;
    friend class RTSTxTimer;

public:
    Vmac(PHY_MIB* p); // Constructor
    ~Vmac(); // Destructor (does not get called for some unknown reason)

    // Vince: Receive packet handler for packets going UP/DOWN
    void recv(Packet *p, Handler *h);

    // Vince: Called by ARP to set MAC header parameter. If called without
    // parameter (just header pointer), returns the parameter already stored
    inline int hdr_dst(char* hdr, int dst = -2);
    inline int hdr_src(char* hdr, int src = -2);
    inline int hdr_type(char* hdr, u_int16_t type = 0);

    // Vince: Timer handlers
    void slotHandler(Event *e);
    void recvHandler(Event *e);
    void sendHandler(Event *e);
    void backoffHandler(Event *e);
    void sendRTSHandler(Event *e);

protected:
    PHY_MIB *phymib_;

    // Vince: DATA slot length and max slot num (max node num) can be configured
    int     slot_packet_len_;
    int max_node_num_;

    // Vince: Variable added to control schedule length (or frame length)
    int schedule_length_;

    // Vince: Keeps track of residual time left until next synced reservation slot
    double restimeleft_;
    int random_;

    // Vince: Event added for RTS backoff timer
    Event rtsinit_;

private:
    // Vince: Command function to link OTcl commands to C++ functions
    int command(int argc, const char*const* argv);

    // Vince: Reserve slot function for transmitter nodes to contend
    void reserveSlot();

    // Vince: Switch radio ON/OFF
    void radioSwitch(int i);

    // Vince: Functions to take care of received packet that is going UP/DOWN
    void sendUp(Packet* p);
    void sendDown(Packet* p);
```

```
// Vince: Random backoff generator with priority
void backoffCalc();

// Vince: RTS creation/transmission functions and packet pointer
void createRTS(u_int32_t dst);
int sendRTS();
Packet *pktRTS_;

// Vince: RTS collision resolution functions
void recvCollision(Packet *p);
void discard(Packet *p, const char* why);

// Vince: Print statistics of fairnesss through Vmac::command
void printStat();

// Vince: Function called to pass data up after receive packet timer expires
void recvDATA(Packet *p);

// Vince: Sends buffered DATA packet
void send();

// Vince: Checks transmitter/receiver idleness
inline int is_idle(void);

// Vince: Debugging functions
void trace_pkt(Packet *p);
void dump(char* fname);

// Vince: Sets MAC log target
void mac_log(Packet *p) {
  logtarget_->recv(p, (Handler*) 0);
}

// Vince: DATA transmission time when given packet pointer
inline double TX_Time(Packet *p) {
  double t = DATA_Time((HDR_CMN(p))->size());

  //printf("<%d>, packet size: %d, tx-time: %f\n", index_, (HDR_CMN(p))->size(), t);
  if (t < 0.0) {
    drop(p, "XXX");
    printf("Error inside TX_Time function.\n");
    exit(1);
  }
  return t;
}

// Vince: Timer variable declarations
SlotVmacTimer mhSlot_;
TxPktVmacTimer mhTxPkt_;
RxPktVmacTimer mhRxPkt_;
RTSBackoffTimer mhBackoff_;
RTSTxTimer mhRTSTx_;

// Vince: Internal MAC transmitter/receiver states
MacState        rx_state_; // incoming state (IDLE, RECV, or COLL)
```

```
    MacState        tx_state_; // outgoing state (SEND, RTS, or COLL)

    // Vince: Radio active indicator
    int radio_active_;

    NsObject* logtarget_;

    // Vince: The time duration for each DATA transmission slot
    static double slot_time_;

    // Vince: Track number of active nodes as they are initialized
    static int active_node_;

    // Vince: The start time for whole TDMA scheduling (may be useless)
    static double start_time_;

    // Vince: Individually, each node keeps track of last reserved slot for VMAC
    //static int last_reserved_slot_;
    int last_reserved_slot_;

    int success_; // Individual success counter
    int attempt_; // Individual attempt counter
    int col_count_; // Keeps track of RTS collisions

    // Vince: Transmitter/receiver schedules
    //static int *tdma_transmitter_;
    int *tdma_transmitter_;
    //static int *tdma_receiver_;
    int *tdma_receiver_;

    // Vince: Keeps track of current transmission slot
    int slot_count_;

    // Vince: Output file for writing fairness to file
    static FILE *outFile_;

    // Vince: How many packets has been sent out?
    //static int tdma_ps_;
    //static int tdma_pr_;
};

double Vmac::slot_time_ = 0;
int Vmac::active_node_ = 0;
double Vmac::start_time_ = 0;

//int Vmac::last_reserved_slot_ = 0;
//int *Vmac::tdma_transmitter_ = NULL;
//int *Vmac::tdma_receiver_ = NULL;

FILE *Vmac::outFile_ = NULL;

//int Vmac::tdma_ps_ = 0;
//int Vmac::tdma_pr_ = 0;

#endif  /* _VMAC_H */
```

VMAC C++ Source File

```
/*
 * File:  vmac.cc
 * Author: Vincent Ngo
 * Summary: Created with mods to mac-tdma.cc to add priority slot reservation
 */

#include "delay.h"
#include "connector.h"
#include "../common/packet.h" // ../common/packet.h for HDR_MAC_VMAC(Packet *)
#include "random.h"

#include "arp.h"
#include "ll.h"
#include "mac.h"
#include "vmac.h"
#include "wireless-phy.h"
#include "cmu-trace.h"

#include <stddef.h>

// For speed() which returns speed of mobile node in double
#include "../common/mobilenode.h"

#define SET_RX_STATE(x)                         \
{                                               \
  rx_state_ = (x);                  \
}

#define SET_TX_STATE(x)                                 \
{                                               \
  tx_state_ = (x);                              \
}

/* Phy specs from 802.11 */
static PHY_MIB PMIB = {
  DSSS_CWMin, DSSS_CWMax, DSSS_SlotTime, DSSS_CCATime,
  DSSS_RxTxTurnaroundTime, DSSS_SIFSTime, DSSS_PreambleLength,
  DSSS_PLCPHeaderLength
};

/* ======================================================================
TCL Hooks for the simulator
====================================================================== */
static class VmacClass : public TclClass {
public:
  VmacClass() : TclClass("Mac/Vmac") {}
  TclObject* create(int, const char*const*) {
    return (new Vmac(&PMIB));
  }
} class_vmac;
```

```
// Vince: Vmac constructor
Vmac::Vmac(PHY_MIB* p) :
  Mac(), mhSlot_(this), mhTxPkt_(this), mhRxPkt_(this), mhBackoff_(this),
    mhRTSTx_(this){

  time_t seconds;

  // Vince: Seed to random number generator to randomize the outcome of every simulation
  seconds = time(NULL);
  Random::seed(seconds);

  /* Global variables setting. */
  // Setup the phy specs.
  phymib_ = p;

  // Vince: Get the simulation parameters from Tcl file
  bind("slot_packet_len_", &slot_packet_len_);
  bind("max_node_num_", &max_node_num_);
  bind("schedule_length_", &schedule_length_);

  slot_time_ = DATA_Time(slot_packet_len_);

  // Vince: Initialize the tdma transmitter and receiver data structures
  tdma_transmitter_ = new int[schedule_length_]; // Holds node IDs of TXs
  tdma_receiver_ = new int[schedule_length_]; // Holds node IDs of RXs

  // Vince: Initialize the fairness counters
  success_ = 0;
  attempt_ = 0;

  // Vince: Initialize the collision counter
  col_count_ = 0;

  // Vince: Keep track of the number of active nodes
  active_node_++;

  if (active_node_ > max_node_num_) {
    printf("Too many nodes taking part in the simulations, aborting...\n");
    exit(-1);
  }

  // Vince: Initialize channel/transceiver states
  tx_state_ = rx_state_ = MAC_IDLE;

  // Vince: Initialy, the radio is off, but can't use radioSwitch(OFF) here
  radio_active_ = 0;

  // Vince: Initialize slot count to beginning of first round
  slot_count_ = FIRST_ROUND;

  // Vince: Pass 0 remaining time so slot handler is called right away
  mhSlot_.start((Packet *) (& intr_), 0);

  if (index_ == 0) {
    outFile_ = fopen("fairVmac.out", "w"); // Vince: Open the file for writing
```

```
    }
}

Vmac::~Vmac() {
    printf("Destructor called\n");
    printf("<%d>, %f, destructor called\n", index_, NOW);
    fclose(outFile_);
}

// Vince: VmacTimer definition (all timers in Vmac follow this)
void VmacTimer::start(Packet *p, double time)
{
    //printf("<%d>, %f, start function called\n", index_, NOW);

    Scheduler &s = Scheduler::instance();
    assert(busy_ == 0); // Check if busy_ is zero or else abort

    busy_ = 1;
    paused_ = 0;
    stime = s.clock(); // Set slot start time to simulation time
    rtime = time; // Set slot remaining time
    assert(rtime >= 0.0); // Check if rtime is positive or else abort

    s.schedule(this, p, rtime);
}

void VmacTimer::stop(Packet *p)
{
    //printf("<%d>, %f, stop function called\n", index_, NOW);

    Scheduler &s = Scheduler::instance();
    assert(busy_);

    if (paused_ == 0)
        s.cancel((Event *)p);

    Packet::free(p);

    busy_ = 0;
    paused_ = 0;
    stime = 0.0;
    rtime = 0.0;
}

// Vince: Slot Timer to administrate start and end of scheduled slots
void SlotVmacTimer::handle(Event *e)
{
    //printf("<%d>, %f, SlotVmacTimer::handle function called\n", index_, NOW);

    busy_ = 0;
    paused_ = 0;
    stime = 0.0;
    rtime = 0.0;

    mac->slotHandler(e);
}
```

```
// Vince: Slot Timer Handler for node ON/OFF transitions and send/receive
void Vmac::slotHandler(Event *e)
{
  //if (index_ == 0)
  //   printf("<%d>, %f, slotHandler function called\n", index_, NOW);

  if ((slot_count_ == schedule_length_) || (slot_count_ == FIRST_ROUND)) {
    // Vince: Wait for reservation time to end, then begin scheduled period
    mhSlot_.start((Packet *)e, RESERVE_TIME);
    //printf("<%d>, %f, reservation time is %f.\n", index_, NOW, RESERVE_TIME);

    // Vince: Turn the radio ON for whole reservation time (random period)
    radioSwitch(ON);

    for(int i=0; i<schedule_length_; i++) {
      tdma_receiver_[i] = NOTHING_TO_SEND;
      tdma_transmitter_[i] = NOTHING_TO_SEND;
    }

    // Vince: Initializations needed for each frame
    slot_count_ = 0;
    last_reserved_slot_ = 0;

    reserveSlot(); // Reservation happens only if a packet is queued for TX
    return;
  }
  else {
    // Vince: Set timer to wait for transmission slot time
    mhSlot_.start((Packet *)e, slot_time_);

    // Vince: If it is the sending slot for me, then transmit
    if (tdma_transmitter_[slot_count_] == index_) {
      // Vince: Wake up to send packet
      radioSwitch(ON);

      send(); // Calling send without packet generates no packet buffered warning

      slot_count_++;
      return;
    }

    // Vince: If I am supposed to listen in this slot, then simulate
    // the reception period by staying awake
    if ((tdma_receiver_[slot_count_] == index_) || ((u_int32_t)tdma_receiver_[slot_count_] == MAC_BROADCAST)) {
      // Vince: Wake up to receive packet
      radioSwitch(ON);

      slot_count_++;
      return;
    }
  }

// Vince: If I don't send/receive in this slot, then sleep
radioSwitch(OFF);
slot_count_++;
```

```
}

// Vince: Turn the radio ON OFF
void Vmac::radioSwitch(int i)
{
    printf("<%d>, %f, radioSwitch function called\n", index_, NOW);

  radio_active_ = i;

  if (i == ON) {
    Phy *p;
    p = netif_;
    ((WirelessPhy *)p)->node_wakeup();
    // printf("<%d>, %f, node_wakeup function called\n", index_, NOW);
    return;
  }

  if (i == OFF) {
    Phy *p;
    p = netif_;
    ((WirelessPhy *)p)->node_sleep();
    //printf("<%d>, %f, node_sleep function called\n", index_, NOW);
    return;
  }
}

// Vince: Backoff Timer for contention collision resolution
void RTSBackoffTimer::handle(Event *e)
{
  printf("<%d>, %f, backoffTimer::handle function called\n", index_, NOW);

  busy_ = 0;
  paused_ = 0;
  stime = 0.0;
  rtime = 0.0;

  mac->backoffHandler(e);
}

// Vince: Backoff Timer Handler to send RTS after waiting for backoff
void Vmac::backoffHandler(Event *e)
{
  u_int32_t dst;
  struct hdr_mac_vmac* dh;

  //printf("<%d>, %f, backoffHandler function called\n", index_, NOW);

  // Vince: If pktTx_ is not 0 then a packet is buffered, and make sure this
  // is still the reservation slot and it has not expired
  if (pktTx_ && slot_count_ == 0 && mhSlot_.expire() > 0) {
    if (last_reserved_slot_ != schedule_length_) {
      // Vince: If residual time left for current reservation slot
      // did not expire, check channel
      if (restimeleft_ != 0) {
        // Vince: Check channel status and minus random by 1 if idle
        if (is_idle()) {
```

```
            // Vince: If random expires then transmit RTS
            if (random_ == 0) {
              //printf("<%d>, %f, random backoff expired\n", index_, NOW);
              dh = HDR_MAC_VMAC(pktTx_);
              dst = ETHER_ADDR(dh->dh_da); // DA should be set by IP layer

              // Vince: Check channel status and create RTS
              createRTS(dst);

              // Vince: Send RTS
              sendRTS();
              return;
            }

            random_--;
            restimeleft_ -= VMAC_SlotTime;
            mhBackoff_.start((Packet *)e, VMAC_SlotTime);
          }
          // Else, wait for next reservation slot since another node took it
          else {
            mhBackoff_.start((Packet *)e, restimeleft_);
            restimeleft_ = 0;
          }
        }
        // Vince: Else, generate new backoff to reserve a slot
        else {
          backoffCalc();
          if (random_ == 0) {
            restimeleft_ = RESERVE_SLOT_TIME;
            mhBackoff_.start((Packet *)e, 0);
          }
          else {
            random_--;
            restimeleft_ = RESERVE_SLOT_TIME - VMAC_SlotTime;
            mhBackoff_.start((Packet *)e, VMAC_SlotTime);
          }

          //printf("<%d>, %f, RETRY residual time left of %f\n", index_, NOW, restimeleft_);
        }
      }
    }
  }

// Vince: Reserve slot if a packet is buffered
void Vmac::reserveSlot()
{
  //printf("<%d>, %f, reserveSlot function called\n", index_, NOW);

  // Vince: If there is a packet buffered, contend for a slot
  if (pktTx_) {
    attempt_++;

    // Vince: Check to see if a slot is available before backoff
    if (last_reserved_slot_ != schedule_length_) {
      backoffCalc();
      // Vince: If random is zero, just enter backoff handler
```

```
        if (random_ == 0) {
          restimeleft_ = RESERVE_SLOT_TIME;
          mhBackoff_.start((Packet *) (& rtsinit_), 0);
        }
        // Else, decrement random and wait for one CW slot
        else {
          random_--;
          restimeleft_ = RESERVE_SLOT_TIME - VMAC_SlotTime;
          mhBackoff_.start((Packet *) (& rtsinit_), VMAC_SlotTime);
        }

        //printf("<%d>, %f, residual time left of %f\n", index_, NOW, restimeleft_);
      }
    }
}

// Vince: Backoff calculator with priority ON/OFF
void Vmac::backoffCalc()
{
    double speed;
    struct hdr_cmn *ch = HDR_CMN(pktTx_);

    // Vince: Implement priority scheduling if it is turned ON
    if (PRIORITY) {
      Phy *p;
      Node *n;
      MobileNode *m;
      p = netif_;
      n = p->node();
      m = (MobileNode *)n;
      speed = m->speed();

      if (ch->ptype() == PT_ARP || ch->ptype() == PT_AODV || speed > 0) {
        random_ = Random::random() % PRIORITY_CW;
        //printf("<%d>, %f, priority backoff of %d\n", index_, NOW, random_);
        return;
      }
    }

    // Vince: Offset by priority CW so regular packets have their own range
    random_ = (Random::random() % REGULAR_CW) + PRIORITY_CW;
    //printf("<%d>, %f, regular backoff of %d\n", index_, NOW, random_);
}

// Vince: Create RTS packet by allocating memory and setting headers
void Vmac::createRTS(u_int32_t dst)
{
    Packet *p = Packet::alloc();
    hdr_cmn* ch = HDR_CMN(p);
    struct rts_frame *rf = (struct rts_frame*)p->access(hdr_mac::offset_);

    //printf("<%d>, %f, createRTS function called\n", index_, NOW);

    assert(pktTx_);
    assert(pktRTS_ == 0);
```

```
    // Vince: Perform carrier sense
    if (!is_idle()) {
      printf("<%d>, %f, transmitting RTS, but the channel is not idle, RX %x, TX %x\n",
            index_, NOW, rx_state_, tx_state_);
      Packet::free(p);
      return;
    }

    ch->uid() = 0;
    ch->ptype() = PT_MAC;

    // Vince: RTS length is 40 bytes in total since RTS frame structure is 16
    // with no FCS, Preamble is 18, and PLCP is 6
    ch->size() = 40;

    ch->iface() = -2;
    ch->error() = 0;

    bzero(rf, MAC_HDR_LEN);

    rf->rf_fc.fc_protocol_version  = MAC_ProtocolVersion;
    rf->rf_fc.fc_type              = MAC_Type_Control;
    rf->rf_fc.fc_subtype           = MAC_Subtype_RTS;
    rf->rf_fc.fc_to_ds     = 0;
    rf->rf_fc.fc_from_ds   = 0;
    rf->rf_fc.fc_more_frag = 0;
    rf->rf_fc.fc_retry     = 0;
    rf->rf_fc.fc_pwr_mgt   = 0;
    rf->rf_fc.fc_more_data = 0;
    rf->rf_fc.fc_wep       = 0;
    rf->rf_fc.fc_order     = 0;

    // Vince: Set receiver address
    STORE4BYTE(&dst, rf->rf_ra);

    // Vince: Set transmitter address
    STORE4BYTE(&index_, rf->rf_ta);

    // Vince: Set transmission time with packet size
    ch->txtime() = DATA_Time(ch->size());

    pktRTS_ = p;
}

// Vince: Send RTS packet by passing packet downwards to physical layer
int Vmac::sendRTS()
{
    double stime;

    //printf("<%d>, %f, sendRTS function called\n", index_, NOW);

    if (pktRTS_ == 0)
      return -1;

    SET_TX_STATE(MAC_RTS);
```

```
  stime = TX_Time(pktRTS_);

  // Vince: Start RTS transmission timer so MAC Tx state is reset to idle after tx
  mhRTSTx_.start(pktRTS_->copy(), stime);

  // Vince: Packet is sent by this function by passing it downwards
  downtarget_->recv(pktRTS_, this);

  pktRTS_ = 0;

  return 0;
}

// Vince: Send RTS Timer for checking success/failure of RTS transmission
void RTSTxTimer::handle(Event *e)
{
  //printf("<%d>, %f, RTSTxTimer handle function called\n", index_, NOW);

  busy_ = 0;
  paused_ = 0;
  stime = 0.0;
  rtime = 0.0;

  mac->sendRTSHandler(e);
}

// Vince: Send RTS Handler to reset MAC Tx state and check RTS transmission
void Vmac::sendRTSHandler(Event *e)
{
  //printf("<%d>, %f, sendRTSHandler function called\n", index_, NOW);
  //printf("<%d>, %f, RTS packet sent\n", index_, NOW);

  // Vince: If no collision then set schedule for transmitter
  if (tx_state_ != MAC_COLL) {
    success_++;
    //printf("<%d>, %f, successes is %d\n", index_, NOW, success_);

    tdma_transmitter_[last_reserved_slot_] = index_; // index_ is node ID/MAC addr
    //printf("<%d>, %f, traffic exchange slot is %d\n", index_, NOW, last_reserved_slot_);
    last_reserved_slot_++;

    Packet::free((Packet *)e);
  }
  // Vince: Else, collision happened so retry by waiting for beginning of
  // next reservation slot (which is the restimeleft_ minus RTS Tx time). Set
  // restimeleft_ to zero so a new backoff is chosen
  else {
    pktRTS_ = (Packet *)e;
    mhBackoff_.start(pktRTS_, (restimeleft_-TX_Time(pktRTS_)));
    restimeleft_ = 0;
  }

  SET_TX_STATE(MAC_IDLE);
}

// Vince: Receive function which receives all packets going UP or DOWN
```

```
void Vmac::recv(Packet* p, Handler* h) {
  u_int32_t dst, src;
  struct hdr_cmn *ch = HDR_CMN(p);
  struct hdr_mac_vmac *dh = HDR_MAC_VMAC(p);

  //printf("<%d>, %f, recv function called\n", index_, NOW);

  dst = ETHER_ADDR(dh->dh_da); // Convert node ID into 4-byte address
  src = ETHER_ADDR(dh->dh_sa);

  // Vince: If incoming packet from PHY layer, then send up to LL layer if
  // sendUp finds no collision during reception and it is not RTS packet
  if (ch->direction() == hdr_cmn::UP) {
    // Vince: If radio inactive (sleeping), then free packet
    if (!radio_active_) {
      free(p);
      printf("<%d>, %f, I am asleep\n", index_, NOW);
      return;
    }

    sendUp(p);
    //printf("<%d> packet recved: %d\n", index_, tdma_pr_++);
    return;
  }

  // Vince: Else, packet is coming down from LL layer and needs to be sent
  // down to PHY layer according to MAC protocol
  callback_ = h; // callback for end-of-transmission (each node has one)
  sendDown(p);
  //printf("<%d> packet sent down: %d\n", index_, tdma_ps_++);
}

// Vince: If transitting and receiving or collision due to receiving
// from two sources, then packet is dropped after reception completes (recvHandler)
void Vmac::sendUp(Packet* p)
{
  struct hdr_cmn *ch = HDR_CMN(p);
  //printf("<%d>, %f, sendUp function called\n", index_, NOW);

  // Vince: If transmitting, then a collision occurred if the
  // packet is received, therefore set transmitter state
  if (tx_state_ != MAC_IDLE && ch->error() == 0) {
    //printf("<%d>, %f, can't receive while transmitting!\n", index_, NOW);
    SET_TX_STATE(MAC_COLL);

    col_count_++;
    //printf("<%d>, %f, transmitter collision (%d)\n", index_, NOW, col_count_);
  };

  // Vince: If not receiving, then receive packet and if transmitter collision happened
  // the packet will be dropped after it is received fully
  if (rx_state_ == MAC_IDLE) {
    pktRx_ = p;

    double rtime = TX_Time(p);
    assert(rtime >= 0);
```

```
        SET_RX_STATE(MAC_RECV);
        mhRxPkt_.start(p, rtime);
    }
    // Vince: Else, receiving from two sources so collision
    else {
        struct hdr_cmn *chNew = HDR_CMN(p);
        struct hdr_cmn *chOld = HDR_CMN(pktRx_);

        // Vince: If CS threshold passes and receiving threshold fails,
        // error flag should be set already. Therefore, check error flag of the
        // new and old packet and decide if collision is negligible
        if (chOld->error() == 0 && chNew->error() == 1)
            return; // New packet can be ignored because it is only CS
        else if (chOld->error() == 1 && chNew->error() == 0) {
            // Vince: Old packet is only CS and new packet is receivable
            // so drop old and receive new
            mhRxPkt_.stop(pktRx_);
            Packet::free(pktRx_);
            pktRx_ = p;
            mhRxPkt_.start(pktRx_, TX_Time(pktRx_));

            return;
        }

        // Vince: Receiver collision since both packets are receivable
        col_count_++;
        //printf("<%d>, %f, receiver collision (%d)\n", index_, NOW, col_count_);

        recvCollision(p);
    }
}

// Vince: Set receiver state to collision and drop shorter packet and receive
// longer packet, but eventually longer packet is dropped as well
void Vmac::recvCollision(Packet *p)
{
    SET_RX_STATE(MAC_COLL);

    assert(pktRx_);

    // Vince: Since collision, both packets will eventually be discarded.
    // Expire() returns residual amount of time left in timer
    if(TX_Time(p) > mhRxPkt_.expire()) {
        // Vince: Old packet is dropped since old packet is corrupted by new packet
        mhRxPkt_.stop(pktRx_);
        discard(pktRx_, DROP_MAC_COLLISION);

        pktRx_ = p;
        mhRxPkt_.start(pktRx_, TX_Time(pktRx_));
    }
    else
        discard(p, DROP_MAC_COLLISION);
}

void Vmac::discard(Packet *p, const char* why)
```

```
{
  hdr_cmn *ch = HDR_CMN(p);

  // Vince: If the received packet contains errors, a real MAC layer couldn't
  // necessarily read any data from it, so toss it now
  if (ch->error() != 0) {
    Packet::free(p);
    return;
  }

  // Vince: Packet from receiver collision is dropped with reason
  drop(p, why);
}

// Vince: Receive Packet Timer to check success/failure of reception and pass
// packet upwards to LL layer is successful
void RxPktVmacTimer::handle(Event *e)
{
  //printf("<%d>, %f, RxPktVmacTimer handle function called\n", index_, NOW);

  busy_ = 0;
  paused_ = 0;
  stime = 0.0;
  rtime = 0.0;

  mac->recvHandler(e);
}

// Vince: Receive Packet Handler to remove collided RTS or DATA packets, but
// increment last reserved slot if RTS collision. If RTS received successfully,
// then set receiver schedule or if DATA packet then pass upwards
void Vmac::recvHandler(Event *e)
{
  u_int32_t dst, src, size;
  struct hdr_cmn *ch = HDR_CMN(pktRx_);
  struct hdr_mac_vmac *dh = HDR_MAC_VMAC(pktRx_);
  struct rts_frame *rf = (struct rts_frame*)pktRx_->access(hdr_mac::offset_);

  assert(pktRx_);

  dst = ETHER_ADDR(dh->dh_da);
  src = ETHER_ADDR(dh->dh_sa);
  size = ch->size();

  // Vince: Check for RTS packet
  if (rf->rf_fc.fc_subtype == MAC_Subtype_RTS) {
    // Vince: If RTS collision, discard with drop reason
    if (rx_state_ == MAC_COLL || tx_state_ == MAC_COLL) {
      discard(pktRx_, DROP_MAC_COLLISION);
      //printf("<%d>, %f, exchange slot skipped is %d\n", index_, NOW, last_reserved_slot_);
    }
    // Vince: Else, set receiver schedule
    else {
      tdma_receiver_[last_reserved_slot_] = ETHER_ADDR(rf->rf_ra);
      Packet::free(pktRx_);
    }
```

```
      last_reserved_slot_++;
      goto RESET;
    }

    // Vince: If DATA packet not destinated for me, then free it
    if ((dst != MAC_BROADCAST) && (dst != (u_int32_t)index_)) {
      Packet::free(pktRx_);
      goto RESET;
    }

    // Vince: Turn the radio off only after receiving DATA packet
    radioSwitch(OFF);

    // Vince: Forward packet upwards
    recvDATA(pktRx_);

RESET:
    // Vince: Reset receiver state to idle
    SET_RX_STATE(MAC_IDLE);
}

// Vince: Strip off MAC header and pass DATA packet upwards to LL layer
void Vmac::recvDATA(Packet *p)
{
    struct hdr_cmn *ch = HDR_CMN(p);

    //printf("<%d>, %f, recvDATA function called\n", index_, NOW);

    ch->size() -= ETHER_HDR_LEN;
    ch->num_forwards() += 1; // Increment forward count

    uptarget_->recv(p, (Handler*) 0);
}

// Vince: Send packet down to PHY layer. Setup MAC header and packet will be
// sent when slot reservation is successful
void Vmac::sendDown(Packet* p)
{
    u_int32_t dst, src, size;

    struct hdr_cmn* ch = HDR_CMN(p);
    struct hdr_mac_vmac* dh = HDR_MAC_VMAC(p);

    //printf("<%d>, %f, sendDown function called\n", index_, NOW);

    // Vince: Check if another packet already exist and stop if it does
    assert(pktTx_ == 0);

    // Vince: Update the MAC header (same as 802.11)
    ch->size() += ETHER_HDR_LEN;

    dh->dh_fc.fc_protocol_version  = MAC_ProtocolVersion;
    dh->dh_fc.fc_type              = MAC_Type_Data;
    dh->dh_fc.fc_subtype           = MAC_Subtype_Data;
```

```
    // Vince: All control bits are not used
    dh->dh_fc.fc_to_ds      = 0;
    dh->dh_fc.fc_from_ds    = 0;
    dh->dh_fc.fc_more_frag  = 0;
    dh->dh_fc.fc_retry      = 0;
    dh->dh_fc.fc_pwr_mgt    = 0;
    dh->dh_fc.fc_more_data  = 0;
    dh->dh_fc.fc_wep        = 0;
    dh->dh_fc.fc_order      = 0;

    if ((u_int32_t)ETHER_ADDR(dh->dh_da) != MAC_BROADCAST)
      dh->dh_duration = DATA_DURATION;
    else
      dh->dh_duration = 0;

    dst = ETHER_ADDR(dh->dh_da); // Convert node ID into 4-byte address
    src = ETHER_ADDR(dh->dh_sa);
    size = ch->size();

    // Vince: Since TDMA, packet is sent only during the reserved slot
    pktTx_ = p;
}

// Vince: Actually send the DATA packet by passing it to PHY layer
void Vmac::send()
{
    u_int32_t dst, src, size;
    struct hdr_cmn* ch;
    struct hdr_mac_vmac* dh;
    double stime;

    //printf("<%d>, %f, send function called\n", index_, NOW);

    // Vince: Check if there is any packet buffered
    if (!pktTx_) {
      printf("<%d>, %f, no packet buffered.\n", index_, NOW);
      return;
    }

    // Vince: Perform carrier sense (should not have collision since TDMA)
    if(!is_idle()) {
      printf("<%d>, %f, transmitting, but the channel is not idle, RX %x, TX %x\n",
            index_, NOW, rx_state_, tx_state_);
      return;
    }

    ch = HDR_CMN(pktTx_);
    dh = HDR_MAC_VMAC(pktTx_);

    dst = ETHER_ADDR(dh->dh_da);
    src = ETHER_ADDR(dh->dh_sa);
    size = ch->size();

    stime = TX_Time(pktTx_);
    ch->txtime() = stime;
```

```
    // Vince: Set transmitter state to send
    SET_TX_STATE(MAC_SEND);

    // Vince: Start transmission timer
    mhTxPkt_.start(pktTx_->copy(), stime);

    // Vince: Transmit by passing packet down to PHY layer
    downtarget_->recv(pktTx_, this);

    pktTx_ = 0;
}

// Vince: Test if the channel is idle
int Vmac::is_idle()
{
    //printf("<%d>, %f, is_idle() function called\n", index_, NOW);

    if(rx_state_ != MAC_IDLE)
        return 0;
    if(tx_state_ != MAC_IDLE)
        return 0;
    return 1;
}

// Vince: Send DATA Timer to reset transmitter state and switch radio OFF
void TxPktVmacTimer::handle(Event *e)
{
    //printf("<%d>, %f, TxPktVmacTimer handle function called\n", index_, NOW);

    busy_ = 0;
    paused_ = 0;
    stime = 0.0;
    rtime = 0.0;

    mac->sendHandler(e);
}

// Vince: Send DATA Handler for freeing sent packet and unlocking the IFQ
void Vmac::sendHandler(Event *e)
{
    //printf("<%d>, %f, sendHandler function called\n", index_, NOW);
    //printf("<%d>, %f, DATA packet sent\n", index_, NOW);

    // Vince: Reset transmitter state and free sent packet
    SET_TX_STATE(MAC_IDLE);
    Packet::free((Packet *)e);

    // Vince: Turn radio off
    radioSwitch(OFF);

    // Vince: Unlock IFQ so other packets can be sent downwards
    if(callback_) {
        Handler *h = callback_;
        callback_ = 0;
        h->handle((Event*) 0);
    }
```

```
}

// Vince: Binded function to Tcl to print successes and attempts for each node
void Vmac::printStat()
{
  // Vince: Prevent division by zero
  if (attempt_ != 0) {
    printf("<%d>, %f, %d successes, %d attempts, %f, %d cols\n",
      index_, NOW, success_, attempt_, double(success_)/attempt_, col_count_);
    fprintf(outFile_, "%f ", double(success_)/attempt_);
  }
}


// Vince: Similar to 802.11, no cached node lookup. Create commands that are
// linked to Tcl object commands for access into C++ functions
int Vmac::command(int argc, const char*const* argv)
{
  //printf("<%d>, %f, command function called\n", index_, NOW);

  if (argc == 2) {
    if (strcmp(argv[1], "print-stat") == 0) {
      printStat();
      return TCL_OK;
    }
  }

  if (argc == 3) {
    if (strcmp(argv[1], "log-target") == 0) {
      logtarget_ = (NsObject*) TclObject::lookup(argv[2]);
      if(logtarget_ == 0)
        return TCL_ERROR;
      return TCL_OK;
    }
  }
  return Mac::command(argc, argv);
}

// Vince: Debugging routine trace packet
void Vmac::trace_pkt(Packet *p)
{
  //printf("<%d>, %f, trace_pkt function called\n", index_, NOW);

  struct hdr_cmn *ch = HDR_CMN(p);
  struct hdr_mac_vmac* dh = HDR_MAC_VMAC(p);
  u_int16_t *t = (u_int16_t*) &dh->dh_fc;

  fprintf(stderr, "\t[ %2x %2x %2x %2x ] %x %s %d\n",
    *t, dh->dh_duration,
    ETHER_ADDR(dh->dh_da), ETHER_ADDR(dh->dh_sa),
    index_, packet_info.name(ch->ptype()), ch->size());
}

// Vince: Debugging routine dump
void Vmac::dump(char *fname)
{
```

```
    //printf("<%d>, %f, dump function called\n", index_, NOW);

    fprintf(stderr, "\n%s --- (INDEX: %d, time: %2.9f)\n", fname,
        index_, Scheduler::instance().clock());

    fprintf(stderr, "\ttx_state_: %x, rx_state_: %x, idle: %d\n",
        tx_state_, rx_state_, is_idle());
    fprintf(stderr, "\tpktTx_: %lx, pktRx_: %lx, callback: %lx\n",
        (long) pktTx_, (long) pktRx_, (long) callback_);
}

// Vince: Called by ARP to set MAC header parameter. If called without
// parameter (just header pointer), returns the parameter already stored
int Vmac::hdr_dst(char* hdr, int dst )
{
    //printf("<%d>, %f, hdr_dst function called\n", index_, NOW);

    struct hdr_mac_vmac *dh = (struct hdr_mac_vmac*) hdr;
    if(dst > -2)
        STORE4BYTE(&dst, (dh->dh_da));
    return ETHER_ADDR(dh->dh_da);
}

int Vmac::hdr_src(char* hdr, int src )
{
    //printf("<%d>, %f, hdr_src function called\n", index_, NOW);

    struct hdr_mac_vmac *dh = (struct hdr_mac_vmac*) hdr;
    if(src > -2)
        STORE4BYTE(&src, (dh->dh_sa));
    return ETHER_ADDR(dh->dh_sa);
}

int Vmac::hdr_type(char* hdr, u_int16_t type)
{
    //printf("<%d>, %f, hdr_type function called\n", index_, NOW);

    struct hdr_mac_vmac *dh = (struct hdr_mac_vmac*) hdr;
    if(type)
        STORE2BYTE(&type,(dh->dh_body));
    return GET2BYTE(dh->dh_body);
}
```