Improving latency in Crankshaft - An energy-aware MAC protocol for Wireless Sensor Networks

by

Suvesh Pratapa

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

December 2009

APPROVED:

Professor Robert Kinicki, Thesis Advisor

Professor Carolina Ruiz, Thesis Reader

Professor Michael Gennert, Head of Department

Acknowledgements

I would like to thank Prof. Robert Kinicki for his guidance during the two years I have been a graduate student at WPI. I picked this topic after attending his talk titled 'A Glimpse at Three Wireless Networking Problems' back in the Fall of 2007, and I am very pleased with the amount of challenge this work has provided me. His encouragement during my research and his thorough knowledge of the domain helped me overcome many obstacles over the course of this research. I am grateful to him for his extremely meticulous and analytical critique during the last couple of months of this thesis. He was very understanding and made sure I was undeterred in seeing it through. I am fortunate to have had him as an academic advisor as well, as he helped me make some very thoughtful choices as a graduate student.

I would like to thank Mr. Gertjan Halkes from the University of Delft. As one of the original authors of Crankshaft, he was gracious enough to help me understand the intricacies of simulating the protocol. Moreover, he was the only source of help with issues regarding the OMNeT++/MiXiM simulator framework. He had a major role in helping me finish this thesis.

I would like to thank everyone in the Department of Computer Science at WPI for their support during these two years. I have learnt so much here as a graduate student and a teaching assistant. I saw Fuller Labs as a second home, and everyone in it as family. I definitely looked forward to being at work each day, and that made work much easier. I should mention Prof. Stanley Selkow, Prof. George Heineman, Prof. Mark Claypool, and the other professors in the department I have had a chance to work with. I am thankful to them not only for some outstanding courses, but also for their words of support at various junctures during these two years. They have all left a mark on my outlook for the future.

I was also fortunate to be in the company of extremely talented and highly well rounded graduate students in the department. I will forever cherish the friendships I have made here. Special thanks to Jeremy Denham: lab mate extraordinaire, partner in crime, kindred spirit, and most of all, a friend without par.

Finally, I am indebted to my parents for their constant encouragement, and their undying love. Looking back at all the years leading up to this moment, they have been, and forever will be, the pillars of support my life and my work will rest on. Thank you, *Amma*. Thank you, *Nana*.

Contents

List of figures

List of tables

Abstract

Due to the dramatic growth in the use of Wireless Sensor Network (WSN) applications - ranging from environment and habitat monitoring to tracking and surveillance, network research in WSN protocols has been very active in the last decade. With battery-powered sensors operating in unattended environments, energy conservation becomes the key technique for improving WSN lifetimes. WSN Medium Access Control (MAC) protocols address energy awareness and reduced duty cycles since the radio is the component that consumes most of the energy. This thesis investigates the performance of two recently published energy-aware MAC protocols, Crankshaft and SCP-MAC. Crankshaft has been shown to be one of the best protocols in terms of energy consumption in dense WSNs while SCP-MAC has a dedicated low duty cycle and low average latencies.

The focus in this investigation is to discover techniques for reducing the latency of Crankshaft. Using OMNeT++, an open source and component-based simulation framework, this study investigates possible modifications to Crankshaft to improve its latency. The potential improvements considered include modifications to Crankshaft's retransmission contention scheme (Sift), adjustments to its inherent settings, and investigating the impact of ACKs. Since OMNeT++ readily provided only a variant of SCP-MAC identified as SCP-MAC*, the simulations results presented involve comparing variants of both protocols (Crankshaft and SCP-MAC*). The performance of these protocols is also analyzed using distinct sensor node communication patterns.

It was determined that Crankshaft's latency depends on its ACK/Retransmission settings. Specifically, Crankshaft has the best latency with No ACKs, without much loss in energy consumption. But the latency can also be improved when ACKs are enabled by reducing the number of retries. Furthermore, the latency and delivery ratio are also directly governed by the WSN traffic pattern and the congestion in the network, as there was a noticeable improvement for both parameters in one-hop traffic, compared to multi-hop convergecast traffic to the sink. Finally, it was observed that Crankshaft's broadcast performance in flooding traffic can be improved by increasing the number of broadcast slots used, though this is detrimental to its performance in unicast traffic.

Chapter 1

Introduction

1.1 Wireless Sensor Networks

Over the past decade, the research area in wireless sensor networks (WSNs) has been driven by advances in digital circuitry leading to ever smaller systems [1]. In 1999, Pister et al., in their visionary paper "Smart Dust", proposed using a sensor, a micro-controller and a radio to form a sensor node that can be deployed wirelessly. A wireless sensor network can be defined as a network of multiple sensor nodes that can organize wirelessly and collectively report information about their environment [1]. The range of WSN applications includes environment, habitat and wildlife monitoring, precision agriculture, and ad-hoc deployments for disaster management and object tracking [2].

Langendoen et al., [1] propose a set of characteristics and constraints that define WSNs. The most important features that distinguish these networks are: 1) nodes must operate for a longer periods of time for economic reasons, 2) the network is usually unattended, 3) high error resilience is required, 4) data usually traverses common paths and is routed to a base station or a sink.

WSN performance is dictated by the resource limitations on the sensor nodes. Over the years, the CPUs have become faster and there has been a raw performance increase in terms of memory usage. Improvements in radio hardware means that higher data rates are possible. But the effective lifetime of a sensor node depends on the amount of work that can be done using the same set of batteries. Running a sensor node constantly will drain a pair of AA batteries in about four days

[1]. Energy usage also depends on the time the sensor spends in sleep state; taking into account the setup time that a radio spends in switching from sleep to send/receive state. Hence power management is a must. Moreover, the radio generally is the most power-consuming component of a sensor node (see section 2.1). Thus, large gains can be achieved at the data link layer by having MAC (Medium Access Control) protocols control the usage of the radio and prudently switch between awake and sleep states.

Taking into account all the various sources of energy usage in WSNs (see section 2.2), there have been numerous research efforts in implementing a variety of energy-aware MAC protocols [3]. These are varied in nature and diverse in inherent structure because each protocol proposed typically trades off other performance characteristics such as throughput and latency in favor of reduction in energy consumption.

1.2 Definitions

Some commonly used networking terms that will be used later in this thesis are defined below.

Delay is the measure of the time it takes for a source to send a packet to a destination. Often, delay is measured as round-trip time, which is easier to measure at the source itself. The terms delay and latency are used interchangeably later in this thesis, but technically, latency does not include any queuing delay along the path.

Delivery Ratio is computed as the ratio of the number of packets received by the destination to the number of packets actually sent by the source. This excludes duplicates. It is often a good measure of the reliability of the network.

Data Rate of a network is the overall amount of data that is being transmitted per unit time [1], averaged over all communication links. This factor usually increases for dense networks.

1.3 Goals of this thesis

This thesis investigates two of the leading MAC protocols in this domain, Crankshaft [4] and SCP-MAC [5]. Crankshaft was proven to be the best in terms of energy consumption in dense WSNs.

[1] as a standard, measured in bits per second (Kbps/Mbps)

But it suffers in terms of delivery ratio and latency at high data rates.

The goals of this thesis are three-fold.

- Investigate the design of Crankshaft to improve its latency. Provide the best conditions for Crankshaft that bring the latency down without adversely impacting its energy consumption.
- Provide a thorough analysis of Crankshaft with different settings of its various parameters. Since the Crankshaft protocol provides a lot of novel ideas for WSN MAC protocol design, this analysis will be helpful for future research.
- Provide a thorough comparison of Crankshaft and SCP-MAC under different traffic patterns, so as to understand the effect of the network topology and traffic on MAC protocol design.

Using OMNeT++, an open source and component-based simulation framework, this study investigates possible modifications to Crankshaft to improve its latency. The potential improvements considered include replacing Crankshaft's contention resolution scheme (Sift), improvements to its retransmission parameters and investigating the impact of ACKs. The network topology and the effects of multi-hop vs. single-hop traffic are also studied in this analysis. Since OMNeT++ readily provided only a variant of SCP-MAC identified as SCP-MAC*, the simulations results presented involve comparing variants of both protocols (Crankshaft and SCP-MAC*). The performance of these variants is analyzed using three different traffic patterns, commonly seen in many WSN applications.

The rest of this document is organized as follows. **Chapter 2** contains further background on WSN platforms, and an energy-aware MAC protocol is formally defined. This is followed by a background on the application scope and the simulator framework. **Chapter 3** contains a discussion on the classification of energy-aware MAC protocols, followed by an overview of various protocols relevant to this thesis. **Chapter 4** contains an argument to improve Crankshaft. The simulation setup and experimental methodology are discussed in **Chapter 5**. **Chapter 6** contains the results of the simulations. Conclusions and future work are presented in **chapter 7**.

Chapter 2

Background

This chapter provides a background on Wireless Sensor Network architecture. **Section 2.1** contains an introduction on sensor node platforms and their components. **Section 2.2** details WSN characteristics important to energy-aware MAC protocols. **Section 2.3** discusses the target application space, architectural choices for WSNs and germane communication patterns. The OMNeT++ simulation framework is described in **Section 2.4**.

2.1 Sensor node platforms

The energy limitations on sensor nodes and WSNs in general (as introduced in section 1.1) impose unique research directions in many different sub-domains. This includes hardware research in the field of MEMS (Microelectromechanical systems, technology for devices whose size ranges in the micrometers), software research in developing efficient operating systems for tiny sensors (TinyOS being the defacto standard), developing distributed routing algorithms, and other fields such as localization, data visualization and security [6]. Several standards are also under development, such as WirelessHART [7], an open standard, time synchronizing and self-organizing mesh architecture and ISA100 [8], which uses the 6LoWPAN (IPv6 over low power wireless personal area networks) standard for industrial applications.

One of the more important WSN standards is ZigBee (under IEEE 802.15.4), a wireless networking standard with packet based radios. Research [1] has shown that a wireless MAC protocol driving a simple packet-based radio leads to better application processing than using byte-based

radios, leading to more diversified applications. Thus, Zigbee standard radios are commonly used for many WSN applications. However, packet-based radios are not ideal for complex protocols that need byte-level hardware access.

As mentioned previously, there has been a huge improvement in hardware for sensor nodes. The earliest popular commercial sensor network platform was the Berkeley mote [9], developed and popularized by CrossBow technologies. Each mote is made up of a sensor board and a processing board consisting of a wireless radio transceiver and a micro-controller. All the motes were designed to run off standard AA or Li-ion watch batteries, such that under optimal conditions they could last more than a year off a single battery charge. The networking hardware used in different motes consisted of a number of low-power radios ranging from earlier models such as CC868 (running proprietary networking stacks) to later versions such as TR1000, CC1000 and the currently standard packet-based radios such as CC2420, which supports ZigBee. Other versions of the Berkeley motes were developed/improved over the years. These include Rene, Mica, Mica2, Mica2Dot, MICAz, MICA2dot, Telos and IMote2 [9]. A comparison of various platforms is provided in Table 2.1.

	René **1999**	**Mica-2** **2002**	**Tmote Sky** **2005**	**Imote2** **2007**
CPU	**ATMEL 8535** 8-bit, 4 MHz 36 μW sleep 60 mW active	**ATmega128L** 8-bit, 8 MHz 36 μW sleep 60 mW active	**TI MSP430** 16-bit, 8 MHz 15 μW sleep 5.4 mW active	**Intel PXA271** 32-bit, 13-416 MHz 390 μW sleep $\geq$ 31 mW active
Memory	512 B RAM 8 KB Flash	4 KB RAM 128 KB Flash	10 KB RAM 48 KB Flash	32 MB RAM 32 MB Flash
Radio	**RFM TR1000** 10 Kbps 2 μW sleep 12 mW receive 36 mW xmit 0.5 ms setup	**CC1000** 76 Kbps 100 μW sleep 36 mW receive 75 mW xmit 2 ms setup	**CC2420** 250 Kbps 60 μW sleep 63 mW receive 57 mW xmit 1 ms setup	

Table 2.1 A comparison between performance specifications of various WSN platforms [1]

2.2 Energy-aware MAC protocols

An energy aware wireless MAC protocol is one that attempts to minimize energy consumption in a WSN. This is explained as follows:

The *duty cycle* of a WSN, defined for a MAC protocol per node, is the ratio of time the sensor

node spends in the awake state, to the total time (awake + sleep) per cycle. Hence, energy-aware MAC protocols seek to minimize the duty cycle.

Effective lifetime of a sensor is the time it spends in actually sending/receiving/processing the data, plus the time it spends in sleep state. One must also consider the time taken in switching from awake to sleep states and vice-versa. The overall lifetime of a WSN depends on the effective lifetimes of all the sensors needed to keep the network running. The actual calculated value depends on the WSN application under consideration.

Energy consumption of a wireless sensor node is the overall power consumption of that node (CPU + Radio). The current trend is to minimize the CPU usage by using low-resource sensor motes. Thus the radio is the component that uses up the largest fraction of the total energy [1].

Minimizing energy consumption of a WSN means trying to get the most work done by the sensors that are awake, while having an energy aware MAC protocol intelligently control the sleep schedules. This is analogous to increasing the *overall lifetime of the WSN*. This can be at the cost of compromising other factors such as latency, but the actual tradeoff depends on the specific application that the WSN supports.

An energy-aware MAC protocol should take into account all the various sources of energy usage in WSNs.

- **Idle listening**: A sensor node is in the idle listening state when the radio is powered ON and the channel is idle. There is an energy waste because of actively listening for incoming packets during an idle channel. This happens because even though only a fraction of the bandwidth is needed for communication, sometimes the radio must be powered on longer than required to anticipate the next message.

- **Overhearing**: Overhearing can occur when a node receives messages intended for its neighbors, or due to hidden terminal [1] traffic. Unless this message can be used somewhere, it is a waste of energy as it leads to unnecessary overhead, especially in dense networks.

- **Collisions**: CSMA (Carrier Sense Multiple Access) is a MAC protocol where the sender

[1]The hidden terminal problem is defined as follows: During communication between two nodes A and B, a node C, which is in range of the receiver B, is not in range of the sender A, and has no way of knowing that that A is transmitting to B. C can thus initiate communication to B, leading to overhearing and collisions.

verifies the absence of other traffic [2] before sending. In wireless networks, direct collision detection is impossible as the radio operates on a half-duplex channel, so it cannot send and detect at the same time. Furthermore, the sender has no way of directly sensing whether a collision occurred at the receiver's end or in the medium.

Therefore CSMA/CA (CSMA with collision avoidance) is used for wireless LANs as collision detection cannot be implemented due to the nature of the channel. This scheme is enhanced in 802.11 DCF to include RTS/CTS. In this method, a Request to Send (RTS) packet is sent by the sender A, and a Clear to Send (CTS) packet sent by the intended receiver B. This can be used to alert all nodes within range of the sender, the receiver, or both that a transmission is reserved in the channel.

But in wireless sensor networks, RTS/CTS is considered prohibitive due to the small data payloads. Since the data itself is a small number of bytes long in most cases, RTS/CTS is considered to be expensive. Retransmissions on lost packets due to collisions can also be expensive due to wasted transmission energy. Since there is no easy way to overcome this problem, collisions are definitely a major source of energy waste for WSNs.

- **Protocol overhead**: This is another important issue. MAC headers, and control packets/frames are considered overhead, and this limits complex but efficient energy-aware MAC protocols.

2.3 Application space

The focus of this thesis is on the Medium Access Control (MAC) layer in WSNs. Therefore, the discussion is limited to WSN communication patterns and applications that directly govern the design of a MAC protocol.

2.3.1 WSN application space

WSN applications vary widely in many aspects. The following is a brief overview of the most common and useful categories according to [10], an excellent resource that investigates these appli-

[2] the physical layer signals the MAC layer

cations in a detailed manner.

- **Environmental Monitoring**: This use of WSNs in environmental applications is important in a world concerned with climate change, global warming and preserving natural resources. WSNs can contribute to emergency hazard response, natural disaster detection and energy-monitoring, among other uses. The most common deployments currently are in the area of meteorological, habitat and pollution monitoring [10].

- **Health Care**: WSN technology is being researched to potentially impact a number of health care applications such as patient monitoring, first response rescue or pandemic warning systems. A number of universal smart care systems are being developed within this area.

- **Security**: This domain includes a lot of challenging problems within tracking, localization, homeland security and military applications. This also constitutes other varied forms of applications involving audio or video surveillance networks for indoors and electro-magnetic and vibration detection systems for outdoors.

- **Commercial domain**: This includes a large number of industrial domains such as structural health monitoring, building monitoring, automotive monitoring and traffic control.

2.3.2 Parametric choices

A critique on sensor networks from an applications perspective is presented in [11]. The authors argue that once the application space is defined, the following standard parametric choices affect protocol design.

- **Rate of sensing**: This can differ, for example, between habitat monitoring, where data is not time sensitive, versus industrial monitoring, where data needs to be collected at high frequency.

- **Rate of sending data to the sink**: Data collection at the sink is useful in some cases where data analysis is required before planning any future sensing activities. For cases such as pollution monitoring, for example, the sink need not collect too much information.

- **Sensor data message size**: This too, depends on the criticality and nature of the application. Commonly seen data sizes for WSNs are in the range of several bytes (For this thesis, as an example, a data size of 25 bytes is used).

- **Data fidelity**: It is acceptable to lose a few samples of data in applications such as habitat monitoring, or in pollution monitoring where the same data is collected by various sensors. In other critical applications, such as applications in an emergency context, all samples may be required.

- **Node Density**: Protocol design for a network with tens of nodes (a *sparse* network) need not be as complicated as protocol design for hundreds or thousands of nodes (a *dense* network). On the other hand, protocols can compromise on issues such as data reliability and network lifetime when there are thousands of nodes. There are often redundancies in the data collected by these sensors.

- **Region of deployment**: This can vary from an uninterrupted medium indoors to an outdoor environment with many line-of-sight links. This can affect many decisions at the link layer.

- **Synchronization**: The issue of the type of time synchronization required is important based on the granularity of error the WSN application can allow. In some cases, no synchronization is necessary.

- **Other choices**: Cost constraints, security and data aggregation constitute the important choices to make when designing a protocol or standard based on the application space.

2.3.3 Architectural decisions

Apart from the above critical choices in designing WSN standards or protocols based on the application, other decisions arise based on the overall architecture of the network itself, and these decisions depend on the choices listed above. We argue that there are basically two basic forms of architectural design that can act as building blocks for more complex architectures.

- **Co-ordinated**: A significant design choice is whether the sensors can operate independently or if global co-ordination is required. For cases where all nodes need not form a single large

network in themselves, tight time synchronization is applied to **clusters** of sensor nodes, which organize among themselves in a hierarchical manner. Multiple independent gateways can be defined so as to have sets of clusters that organize information and have global synchronization at the top-most level. This makes scalability easy, and network lifetime can be re-defined as nodes collect data independently of each other.

- **Centralized**: A centralized approach can be used instead of a distributed one. The argument for a centralized design in some WSN applications can arise from the fact that the sink node has far greater CPU and memory capacity. More important to this thesis is the fact that the sink has access to *continuous* power.

A combination of both these approaches can be used as a **multi-tiered** approach; where the sink node acts as a single point of failure. In such an architecture, the rest of the nodes can organize among themselves in clusters [3].

Other design issues, such as using various forms of high gain or omni-directional antennae, and making use of multiple channels or multiple radios can also be considered based on the complexity of the application.

2.3.4 Communication patterns

Based on the design choices made, the following is a classification of communication patterns that are germane to this study of WSN MAC protocols.

- **Flooding**: In this communication pattern, the sink sends messages to its immediate neighbors; which are then propagated to their immediate neighbors, so that the messages reach all the nodes, with duplicates being filtered out. This is needed when there is a new code image or update/fix that is to be transmitted to all sensors in a broadcast fashion.

- **Convergecast**: All sensors report their findings to the sink along a spanning tree [4]. Messages are usually small (25 bytes or less) and travel along multiple hops. This is the most common communication pattern in monitoring applications.

[3] Clusters can select cluster heads, and cluster heads can talk to each other and relay traffic between the sink and cluster nodes.

[4] Informally, a spanning tree of a graph/network G is a selection of edges of G that form a tree spanning every vertex, without any cycles (or loops) being formed.

- **Local Gossip**: This is a rare communication pattern in which neighboring nodes share/gossip data among themselves. This can be used to filter out false reports or transmit KEEPALIVE messages. For the purpose of this thesis, the case of local *unicast* is considered, in which a node chooses its gossiping neighbor randomly from all of its one-hop neighbors.

These patterns will be defined in the results (Chapter 6) in more detail.

2.4 The OMNeT++ framework

OMNeT++ is an object-oriented (C++) modular discrete event network simulation framework [12]. It is one of the foremost simulators used for WSN research [13]. OMNeT++ in itself is not a simulator, but it provides the infrastructure and tools for that purpose. Modules can be used, re-used and assembled together in various ways to address a particular problem. This component based architecture allows for domain-specific functionality to be provided. For example, some of the applications are:

- Modeling and simulation of communication networks
- Modeling of various protocols and queueing systems
- Modeling of various distributed hardware systems
- Performance evaluation of various software systems and hardware architectures

The newer versions of OMNeT++ support full IDE integration through Eclipse and distributed/parallel distributed simulations. The latest version at the time of writing this document is 4.0. OMNeT++ supports command-line user interfaces for batch execution and extensive GUIs for demonstration and debugging purposes. A commercially available version called OMNEST that is fully supported is available online for a specific license fee; it is used widely in many companies and research labs all around the world.

2.4.1 OMNeT++ modeling concepts

In OMNeT++, a *network* is a group of simple *modules* that communicate by message passing. Any active module, written in C++ using the library, is a simple module, and these can be grouped

into any number of compound modules in any number of levels. Messages are passed around using *connections* and *gates* between the modules. Gates are the input and output interfaces which the connections link together.

A connection can have various parameters corresponding to the network connection properties such as propagation delay, capacity and bit error rate. Connections across hierarchical levels are not permitted. Modules are also defined by parameters to pass configuration data and these can define the topology itself. The parameter objects are used for random number generation within the framework.

The model structure for the entire network can be described using the NED language [14]. The simulation objects themselves are represented by C++ classes. These span modules, gates, connections, parameters, messages, container classes, and other statistical estimation and result-oriented classes.

2.4.2 OMNeT++ simulation concepts

OMNeT++ provides a simulation kernel, written in C++, to manage the code for the simulations and various user interfaces to facilitate debugging, GUI views, and batch executions. The simulation can be compiled once and can be run on other computers. One can also deploy it as a shared library (provided other essential libraries are present)

When a program is started, it reads all NED files containing the model topology, and a configuration file called **omnetpp.ini**. This file contains settings that control the execution of the simulation. The configuration file can prescribe several simulation runs at once.

The output of the simulation can be written into output vector files and output scalar files , with support for user-formatted output files as well. These files can be analyzed using the provided IDE, imported into any package like Matlab, or into OpenOffice Calc or Excel if they are spreadsheets.

2.4.3 MiXiM

MiXiM (**Mix**ed s**im**ulator) is a simulation framework, written using several OMNeT++ frameworks, to support mobile and wireless simulations [15]. It was written at the University of Delft, and it succeeds various successfully deployed frameworks like MacSimulator, Mobility framework

and Positif framework. The current stable version of MiXiM was only recently released and prior to this release two months ago, all development for this thesis happened on a previous test build.

MiXiM was released to provide a framework that accurately reflects various environment models, provide support for mobility and reception handling, and supply a rich protocol library that contains current implementations. Thus it is a very good choice for simulating Wireless Sensor Networks [16]. It supports most of the prevalent MAC protocols in WSNs, and is useful to develop complicated protocols that build on the existing ones. This is possible because of the inbuilt MAC layer models, called **BaseMACLayer**, providing the most basic MAC interfaces, and encapsulation/decapsulation of packets; and **EyesMACLayer**, to build WSN MAC protocols by supplying various support functions, including Low Power Listening and Sift-style carrier sense (See section 3.6)

Chapter 3

Related Work

This chapter formally discusses the classification of WSN MAC protocols. This is followed by a detailed summary of MAC protocols germane to this thesis.

3.1 MAC protocol classification

To minimize energy consumption, most of the energy aware MAC protocols either use a fixed duty cycle or adapt to changes over time and place [1]. They can be divided into the following approaches:

- *Random Access*: This is a low-level approach where the nodes do not organize by time, and contend for access using a winner-takes-all approach. The protocols in this class make use of increased preamble lengths in the MAC header to reduce idle listening. Since this does not specify when the nodes can access the channel, flexibility and dynamic changes can be accommodated easily [1]. Additionally, there is no synchronization required. The difficulty is that both collision avoidance and idle listening overhead prevention techniques should be applied to reduce energy usage.

- *Slotted Access*: Time is divided into slots and nodes are required to synchronize on a common time of reference. The possibility of collisions is much higher due to all transfers contending for an active part of the slot. Therefore effective contention resolution is required.

- *Frame-based Access*: In this TDMA-like approach, time is divided into frames containing fixed number of slots. They differ based on the strategy used to assign nodes to slots. Collisions are reduced and idle listening and overhearing are addressed (reduced but still possible).

- *Hybrid class*: Several combinations between random and frame-based access are possible (efficient but probably with more overhead). These schemes have a lot of flexibility in adapting to traffic and topology changes. The only drawback [1] is that the probability of collisions is high, limiting these protocols to low traffic applications.

3.2 Chronology of energy-aware WSN-MAC protocols

Figure 3.1 shows an overview of some prominent WSN MAC protocols over the last decade. The important thing to note is that most protocols have borrowed a lot of basic ideas and the optimizations have been propagated in achieving some of the state-of-the-art MAC protocols seen today. Most of the earlier protocols are mostly random access or slotted protocols. We limit our discussion to the protocols germane to the discussion of Crankshaft and SCP-MAC*. Crankshaft borrows from the contention resolution mechanism of Sift (discussed in Section 3.8) and SCP-MAC borrows from S-MAC (discussed in Section 3.5) Apart from these two basic protocols, we also discuss some of the more important ones in the history (including LPL - Section 3.3 and WiseMAC - Section 3.4). Though they might have not directly influenced the protocols we discuss, their fundamental philosophies of message transmission and adaptiveness are important to understand.

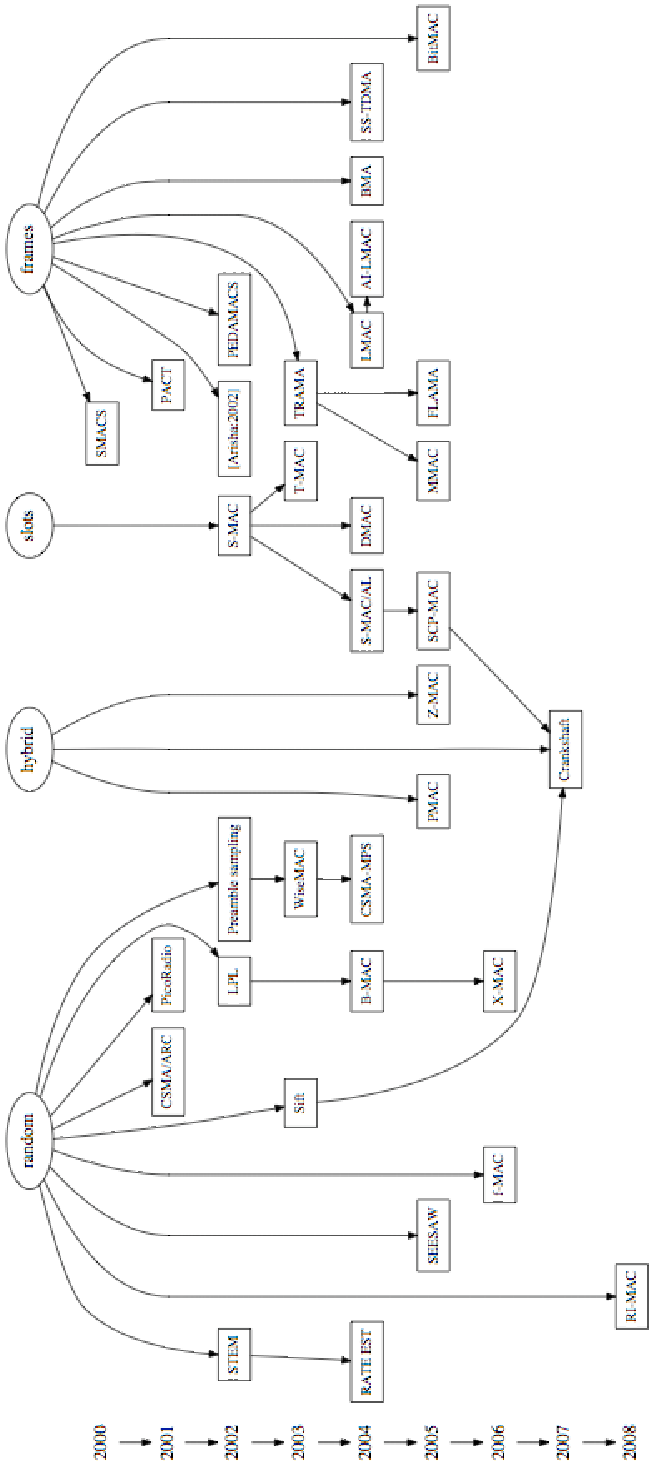

Fig. 3.1 An overview of energy-aware MAC protocols over the decade [3]

3.3 Low Power Listening (LPL)

To reduce the idle listening overhead in random access protocols, the strategy is to prepend each message with a preamble to alert receivers about the upcoming message transfer. The receiving node that senses a busy tone keeps its radio on until the end of the message. Hence, low power listening [17] requires sending a preamble longer than the receiver's poll cycle, (as shown in Figure 3.2), to guarantee that the sender wakes up the receiver, regardless of its phase in the poll cycle. On top of CSMA, this scheme is termed as **Low Power Listening (LPL)** and on top of ALOHA, it is termed as **Preamble Sampling**. This protocol leads to duty cycles of 10% and is good for low data rate applications. The disadvantages are that receiving or overhearing is expensive, and the time between the receiver waking up and receiving the start symbol is on average, half the length of the preamble (10 ms.) [1].

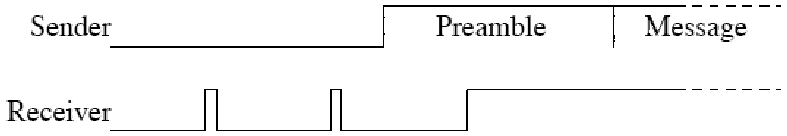

Fig. 3.2 Low Power Listening [1]

3.4 WiseMAC

El Hoiydi et al. [18] introduced **WiseMAC** in 2004. WiseMAC employs a refined preamble sampling method by storing the phase offset of the destination node. This allows the sender to send messages exactly before the intended receiver wakes up. This information can be piggybacked on ACKs. Since the sender can start transmitting the message just before the intended receiver wakes up, shorter preambles are now possible, as shown in Figure 3.3.

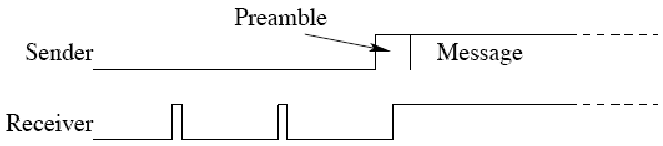

Fig. 3.3 WiseMAC [1]

The energy savings are very substantial. The drawback is that since the length of the preamble is dependent on the interval since the last message exchange, WiseMAC still has long preambles

for infrequent messages or broadcasts.

3.5 S-MAC

The first really successful energy aware MAC protocol was **S-MAC**, a novel slotted access protocol proposed by Ye et al. [19] in 2002. It is considered a novel protocol due to its scalability, distributed energy use and moderate resource requirements. SMAC has a fixed duty cycle [1] to reduce idle listening. Synchronization is achieved by having all the nodes regularly broadcast SYNC packets at the beginning of a slot. New nodes wanting to join the network wait for a SYNC packet and if none is received, they form a virtual cluster and broadcast their own SYNC packets. Multiple virtual clusters can be formed based on this synchronization information.

Collision handling is done by the RTS/CTS method. The designers get around the problem of overhead by including the length of the DATA packet in the RTS/CTS control packets. In SMAC, per-node energy consumption is actually inversely proportional to the traffic load: Energy usage due to idle listening increases for networks with low traffic. A disadvantage is that optimal fixed duty cycles have to be chosen before deployment, which may lead to overprovisioning [1].

3.6 T-MAC

T-MAC is another slotted access protocol by Langendoen et al. [20] proposed in 2002. This protocol uses an adaptive listening approach. Nodes listen only for a short duration at the beginning of a slot and go back to sleep if nothing is heard. If a node overhears a conversation, it schedules another listen period to determine if it can go to sleep again. It outperforms SMAC by a factor of 5 [1] in terms of energy consumption. But the disadvantage is that TMAC is very aggressive in switching the radio off. This leads to messages waiting in queue for the next slot, hence resulting in large amounts of latency over a given period of time. This effect is more visible under high load.

[1] the implementation has a fixed active period of 300 ms.

3.7 L-MAC

Van Hoesel et al. [21] used a distributed slot mechanism in LMAC, a **L**ightweight MAC protocol. This mechanism is based on two-hop neighborhood information. Each node chooses a fixed-length time slot, in which a header is always transmitted, optionally followed by a payload. In LMAC, the correct reception of the data is not acknowledged by the receiver to avoid expensive switching from receiving to transmitting: Reliability is delegated to the upper layers.

When new nodes begin to join the network, each header provides information about which slots are occupied by the one-hop neighbors of the sending node. By performing an OR on these *occupancy bitsets* of all headers in a frame, a new node can determine which slots are still available in its two-hop neighborhood. It randomly selects one of those as its own. In the unlikely event of two nodes selecting the same free slot, a collision occurs, resulting in garbled headers. On observing a mistake in the header checksum, a neighboring node will broadcast this information using the slot number involved in the collision, signaling the senders to try again.

The drawback in LMAC is that the number of slots in a frame have to be fixed before deployment, as the number of nodes in any two-hop neighborhood cannot exceed the number of slots. Choosing a large number of slots leads to over-provisioning and protocol overhead, and choosing a low number is a disadvantage in dense networks.

3.8 Sift

For effective contention resolution in random access protocols, Balakrishnan et al. [22] proposed Sift in 2006. The Sift protocol assumes spatially related contention in Wireless Sensor Networks, i.e., at any given moment only a subset R nodes out of N sensors have data to transmit after detecting an event.

Sift uses a fixed contention window (CW) with a truncated, increasing geometric distribution of choosing the contention slot for a node [22]. On initialization, every node uses a large estimate of node population, called the *belief* population, with a correspondingly small transmission probability, p_r. Each node also continuously checks the contention slots and reduces the node count estimate after every slot that goes free. Additionally, the node increases its transmission probabil-

ity geometrically after each slot. This is the core idea of Sift where the protocol continuously *sifts* the winning node towards the beginning of the contention window. This increasing slot-selection function is best illustrated using a balls-and-bins analogy as shown in Figure 3.4.

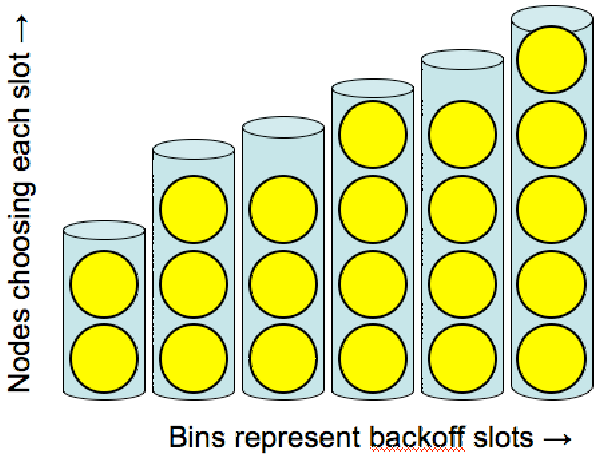

Fig. 3.4 Sifting process [22]

The probability of choosing slot r is

$$p_r = \frac{(1-\alpha)\alpha^{CW}}{1-\alpha^{CW}}\alpha^{-r}, r = 1...CW \tag{3.1}$$

where α is the sift parameter and CW is the size of the contention window, which is fixed.

$$\alpha = (N_1)^{\frac{-1}{CW-1}} \tag{3.2}$$

where the ideal N_1 (number of belief nodes) is 512. For a CW=32, this leads to the ideal α=0.82. The authors also back up these ideal values with performance tests in [22].

3.9 SCP-MAC

The Scheduled-Channel-Polling MAC protocol was introduced by Ye et al. [4] in 2004. It is one of the first in the dedicated low-latency and low duty cycle class of protocols. The main goal of SCP-MAC is to reduce duty cycles by a factor of almost 10 from the nominal 1-2% to 0.1%. In SCP-MAC, all senders wake up to contend for the channel, and all nodes wake up on the receiver probe to see if they are the potential receiver. If no message is sent, everyone goes to sleep.

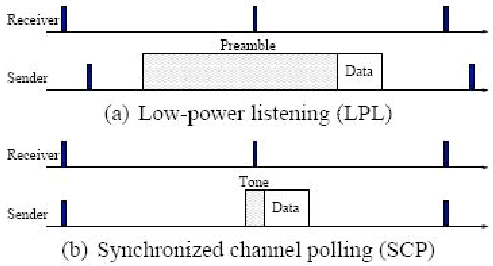

Fig. 3.5 LPL vs. SCP-MAC [4]

Figure 3.5 demonstrates the differences between synchronized polling (used in SCP-MAC) and LPL approaches. Thus, only a short wake-up time is required for setup in scheduled polling. SCP-MAC distributes schedules similar to SMAC: each node broadcasts a SYNC packet to its neighbors every synchronization period. In addition, SCP-MAC attempts to piggyback this information on any data packets.

SCP-MAC also addresses the problem of adapting to varying traffic loads. Other MAC protocols that see this type of traffic have to run at low duty cycles during idle periods, and suffer from long preambles and collisions during busy cycles [4]. Since SCP-MAC uses short preambles, this problem is partially solved. However, during times of heavy traffic, there is a problem of multi-hop latency that reduces throughput [4]. To handle this situation, SCP-MAC introduces the concept of *adaptive channel polling*. The idea is to detect bursty traffic [4] and provide additional polling slots to nodes on the path to allow fast streaming of all data packets coming in. This is illustrated in Figure 3.6.

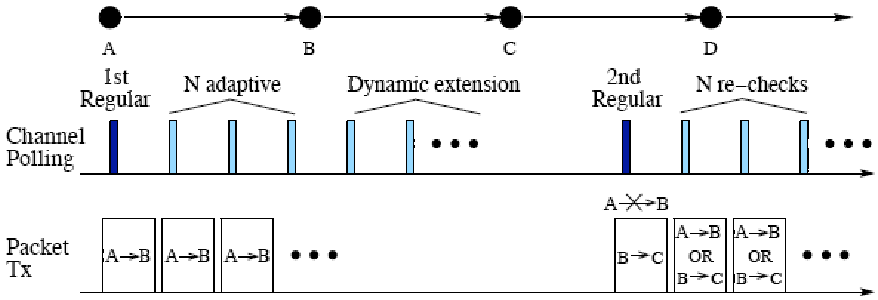

Fig. 3.6 Adaptive channel polling [4]

When B first receives a packet, it adds $\mathbf{n}$ [2] high-frequency polls in the same frame, following its regular poll. If A needs to send more packets, it uses these adaptive polling times. If B finds none of its additional polls useful, it goes back to its regular polling period. If any additional slots were

[2]The authors deduce an optimal value of n=3 [4]

successfully used, B extends the adaptive polling with n slots. This process repeats at each hop, and in bursty traffic, all nodes on the path are transferred to a *high-rate polling mode*. This is achieved due to the original sender giving up the transmission opportunity so that the subsequent nodes can induce polling periods to nodes along the path. In this way, data can be streamed quickly using all-adaptive slots.

The most interesting optimization in SCP-MAC is the two-phase contention period, seen in the following figure.

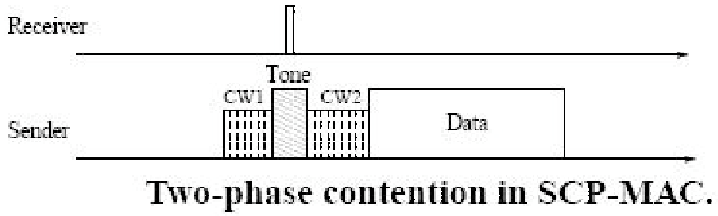

Fig. 3.7 Two phase contention [4]

A node that wants to send uses a random slot in its contention window using a uniform distribution. The winner then sends a preamble (as a busy tone) which continues till the end of the contention window CW1. However, when there is more than one sender colliding in the contention slot, then a second contention period CW2 is used to resolve this contention. The authors of SCP-MAC argue that this leads to better collision avoidance, as SCP tolerates collisions on tone transmissions [4]. Without any traffic, this leads to only one carrier sense per slot, which makes it the most efficient protocol in its class.

Furthermore, SCP-MAC tries to achieve overhearing avoidance by having the receivers examine the destination address of the packet as soon as it is received [3]. SCP-MAC also reduces preamble sizes which yields a considerable reduction in energy consumption. By having the SYNC packets piggybacked, SCP-MAC reduces its inherent overhead.

The authors implement SCP-MAC on top of a PHY layer, a basic CSMA layer supporting full RTS/CTS/DATA/ACK, along with retransmissions, and the LPL layer. Thus it allows for easy component reuse.

[3]Note that this is not possible in packet-based radios.

3.9.1 SCP-MAC*

SCP-MAC* was a variation of SCP-MAC implemented by Langendoen et al. [5] as part of their experimental setup in comparing Crankshaft to SCP-MAC. SCP-MAC* uses the Sift distribution for contention resolution instead of its standard two contention periods. This way the SCP-MAC protocol was easily implemented as a variation of the Crankshaft protocol, where each slot is a receive and broadcast slot at the same time. This was done because the MiXiM framework in which both the protocols were written made it easy to implement SCP-MAC this way. However, the authors do not use ACKs in SCP-MAC*. This results in messages not being kept waiting in queues waiting for retransmissions. This lack of acknowledgements complicates a fair comparison between SCP-MAC and Crankshaft.

3.10 Crankshaft

Crankshaft, one of the newest WSN MAC protocols, was proposed by Langendoen et al. [5] in 2007. It is a hybrid protocol specifically targeted at reducing the energy consumption in dense WSNs. Crankshaft does not maintain any per-neighbor state and uses receiver slot scheduling to bound overprovisioning.

The basic principle of Crankshaft is that nodes are only awake to receive messages at fixed offsets from the start. The receivers are scheduled in order by their MAC addresses (*MAC-Address* % *n*) This is not ideal in dense networks. The authors do mention Time Division Hashing as an alternative, but at a higher protocol complexity. This assignment method might result in two nodes being assigned the same slot, so nodes are allowed to act as senders in their own receive slot [5]. Time is divided into frames, and each frame is divided into slots (8 unicast + 2 broadcast). Each node listens to one unicast slot every frame. During the unicast slot, a node can send a message to the slotted receive node, provided it wins the contention. Crankshaft uses DATA/ACK for unicast messages [4]. During the broadcast slot, all nodes wake up to listen.

[4]RTS/CTS is not implemented

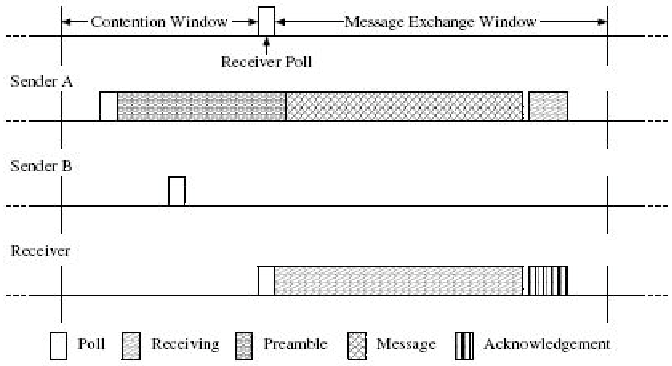

Fig. 3.8 Contention and data transfer in Crankshaft [5]

As shown in Figure 3.8, the period between the start of the slot and the point when the receiver turns on its radio is used to resolve contention. Crankshaft uses channel polling at the receivers similar to SCP-MAC. During the contention window, only the contending senders are awake, and after that only the winner is awake for a short time [5]. For contention resolution, Crankshaft uses the Sift distribution to choose the contention slot. Synchronization of nodes is achieved either through the sink node / using an algorithm such as GSA. Special provisions are made for the sink node, which listens to all unicast slots. Since the sink is the destination for most traffic, the authors propose that it be connected to a much larger battery or main power.

The header for Crankshaft packets consists of

Length	1 byte
To and From addresses	1 byte each
Message Type	1 byte
Synchronization information	3 bytes
CRC information	2 bytes

Table 3.1: Crankshaft header

Since Crankshaft also uses a DATA/ACK exchange for message transfers, the retry parameters must be set properly to ensure a tradeoff between message reliability and network latency. If the sender does not receive an acknowledgment, the protocol is set to retry each message three times in subsequent frames. To reduce contention when retrying, the node will only retry in the next frame with a probability of 70% [5]. It is important to note here that the authors do not deduce these numbers methodically, or provide any justification.

Chapter 4

Investigating Crankshaft Performance

This chapter contains an investigation of the original Crankshaft results (from [5]) to determine areas of improvement, and subsequently an argument to improve the protocol is presented.

4.1 Issues in performance evaluation

The authors of Crankshaft evaluate its performance using OMNeT++ simulations by comparing it with its predecessors: LPL, TMAC, LMAC and SCP-MAC*. They use the setup of of a real-world potato field experiment with 96 nodes on a field of approximately 90x50 meters. The base station is situated at the corner. The radio range (transmission radius) is 25 m and as a result, there is multi-hop traffic propagating to the sink. [1].

Note: In the following results, the delivery ratio and latency are calculated by the authors at the packet layer.

[1]We use the same setup for our simulations, and more details regarding the simulation setup are detailed in Chapter 5

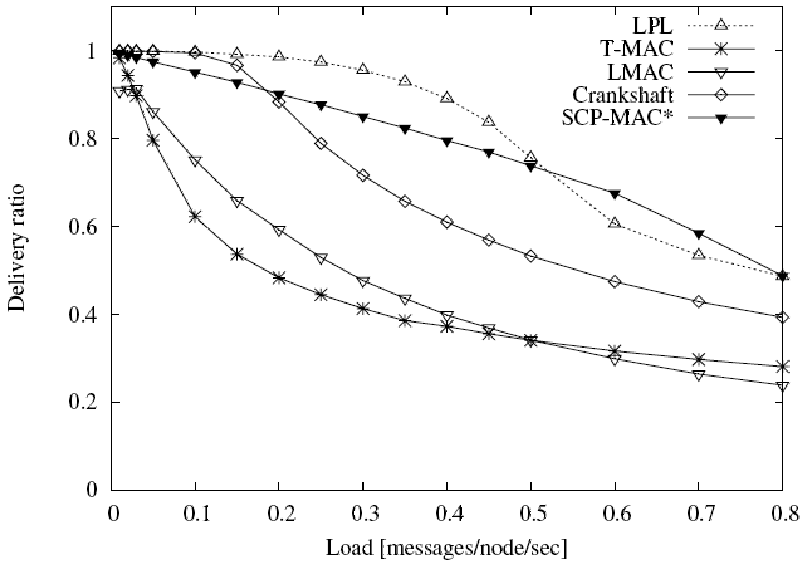

Fig. 4.1 a Delivery ratio

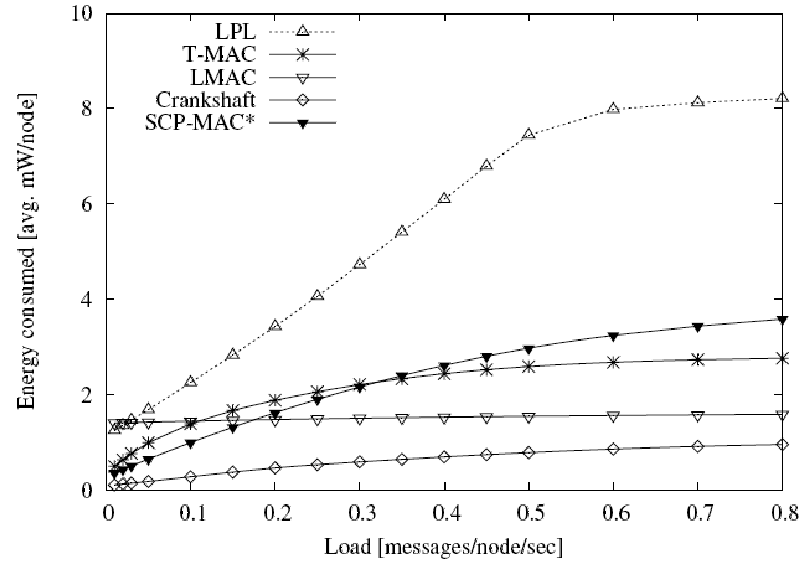

Fig. 4.1 b Energy consumption

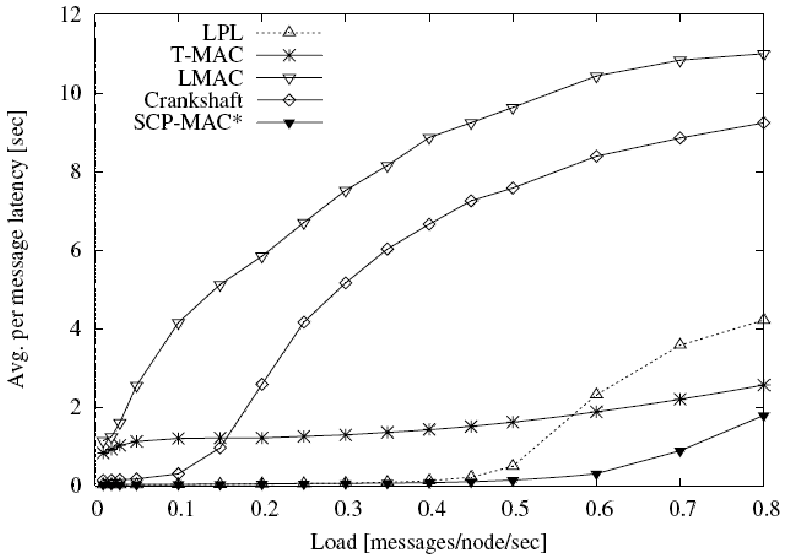

Fig. 4.1 c Latency

Fig. 4.1 Performance of Crankshaft and SCP-MAC* [5]

For convergecast traffic patterns, Crankshaft indeed does have the best energy consumption compared to the other energy-aware MAC protocols at all loads in Figure 4.1 b. This is attributed to the fact that only a set of nodes are awake in each slot and also that the contention is reduced at higher data rates [5]. It is interesting to note though, that the Crankshaft protocol does not do very well compared to LPL and SCP-MAC* in terms of delivery ratios (Figure 4.1 c), though it uses ACKs/Retransmissions. This is especially true at high data rates. This is because the protocol limits itself to one receive slot per frame.

It is also noticed that Crankshaft compromises on latency in favor of energy consumption. It is one of the worst performers in terms of latency. This is because messages are kept waiting in queues, and nodes have to wait a full frame when they lose contention [5]. Another reason for this could be the ACK/Retransmission setting, because of which senders that do not receive an ACK clog the next few frames as they are set to retry 3 times with a high probability. This gets especially worse at higher data rates.

Note: The original implementation of SCP-MAC was designed with a layered approach with MAC-independent PHY, CSMA and LPL layers below it. By changing the essential nature of the protocol using SCP-MAC* (See section 3.7.1), the Crankshaft authors raise questions as to why they proceed with a

modified version. They do mention that the Sift distribution used in SCP-MAC* is better [5], without any evidence. The main motivation seems to be reducing implementation complexity by developing SCP-MAC as a variant of Crankshaft.

An ETH-Zurich presentation of MAC protocols by the same authors [23] that compared Crankshaft and SCP-MAC (the original) on latency showed that SCP-MAC had a lower inherent latency factor than Crankshaft at nominal duty cycles (around 1%) . Thus there seems to be no reason to compare against a modified version of SCP-MAC, especially when it is one of the foremost protocols used in many real world and practical implementations.

This choice of parameters in SCP-MAC* is part of the major investigation of this thesis, and will be explored later in the results (Chapter 6).

4.2 An argument to improve Crankshaft

It was mentioned earlier that MAC protocols for Wireless Sensor Networks are application specific and usually give preference to a particular metric. Crankshaft is designed for dense networks, and the authors mention in [4] that one of their primary goals is monitoring applications. In monitoring applications, data needs to relay quickly to the sink in a convergecast pattern. This is especially true at high data rates where low message delivery ratios and high latencies can lead to a lack of reliable data delivery required on-the-fly. It becomes more important if the information being collected is time-sensitive. This importance of latency in WSNs is also addressed in [24].

Chapter 5

Simulation Setup

This chapter details the various implementation issues while setting up the simulations in MiXiM/ OMNeT++, followed by a section on the experimental setup for the simulations.

5.1 Development constraints

MiXiM was not yet formally released during the development timeframe of this thesis [1]. The only version available to developers to checkout was from the online SVN repository, available on the MiXiM webpage [15]. This checked out code from the online sourceforge repository contains the source code for most of the prominent WSN MAC protocols, including 802.11, CSMA, S-MAC, T-MAC, etc. This enables developing hybrid protocols like Crankshaft and SCP-MAC hierarchically, building on simple modules.

Unfortunately, MiXiM does not work under the latest OMNeT++ framework (4.0). The fact that MiXiM only works with OMNeT++ 3.3 was observed after nearly two months of investigation. This fact turned out to be a major bottleneck in the beginning adding to the already steep development curve [2].

[1] A stable version (1.0) was released in June 2009, available through [15]

[2] Even though the capabilities of MiXiM have been thoroughly tested and it has been proven to be a successful framework for the wireless sensor network domain [16], the working version used for this thesis was very poorly documented. Since this was a pre-release version, the code was also unfinished or poorly designed in some areas. While OMNeT++ itself is fairly well-supported, there were many development bottlenecks due to this lack of support working on the MiXiM framework. Since Crankshaft was originally implemented on this framework, there was no choice but to work through these issues by having to figure out most of the source code from scratch.

On contacting one of the authors of the Crankshaft paper, Mr. Gertjan Halkes, the information regarding directions to develop on the MiXiM framework were obtained. This information also included brief directions on how to configure the simulations for MiXiM.

5.1.1 Changing Sift

One of the original proposals was to change the methodology of Sift so that a node remembers which slot it chose before it retries. This is analogous to the countdown timer in 802.11. When the medium becomes idle and the node wants to compete again, instead of picking a new slot in the back-off window, it resumes from before. Specifically, the following change was to be investigated. If a node chooses a particular slot, say r, and does not receive an ACK when it wins, it retries with the rest of the nodes using Sift, but its notion of the contention window will be reduced to (r-1), i.e., it contends for an earlier slot compared to the other nodes.

If the number of retries is increased, the chance of a node picking an earlier slot is progressively increased so as to beat the others to successfully send its data. As before, the retrier does not clog the other senders, and is not kept waiting for a full frame in order to send, potentially solving the latency issue.

Changing Sift as specified turned out to be non-trivial as there was no straightforward way in the simulator to pass the previously allocated winner slot in future sift contentions. Maybe some cross-layered approach might have worked, though this might have increased the complexity. This was originally intended to be one of the important investigations, but due to development constraints had to be abandoned.

5.2 Configuration

Simulations under MiXiM depend on configuration files. In these files, one can configure various network parameters, topology details, simulation scenarios and protocol specifics. Some of the general settings germane to our simulations are:

- Simulation time: 200 sec
- Node count: 96 (including the base station)

- Sensor node type: EYES wireless sensor node
- Radio type: RFM1001 (byte-oriented)
- Radio range: 25 m
- Propagation model: Path loss propagation

See Appendix 1 for the format of the configuration file, including all settings.

A scenario file is used for the various scenarios to be simulated and analyzed. In this case, eight scenarios were used, corresponding to the eight different network loads that the authors used to analyze their results: scenario1 (0.1 messages/node/sec) - scenario8 (0.8 messages/node/sec). See Appendix 2 for the protocol configuration file which references a scenario.

The scenario files themselves contain information about the communication pattern to be used, the data size, the data rate, reply probability and other packet specific parameters such as time to live [3]. See Appendix 3 for a complete description of a scenario file.

5.2.1 Interpreting the simulator output

A simulation of a particular scenario in MiXiM is run in the following way:

```
./MiXiM -f generic.ini -f GMacF2.ini
```

This gives a command line output that shows the results for various parameters for each node and for the network overall.

There are a set of Perl scripts available in the code base to parse this command line output and pick out the necessary values, but they were found lacking to interpret all the results of protocols like Crankshaft. For example, there was no means to parse the delay for each message in the network, and hence no means to figure out the overall latency. Similarly, there was no means to specify which radio model to use to calculate the right energy values from the values given by the simulations. Furthermore, the names of the various results given out through the Perl scripts already provided were sometimes misleading, so initially the wrong values were calculated for certain parameters.

[3] Time to live (TTL) is a limit on the number of transmissions that a packet can experience in multi-hop traffic before it should be discarded.

To simplify the process, a couple of Perl scripts were written to produce a proper output, with the right notation and the right names. The delay parsing and the radio model were added into the codebase. The results were properly formatted to be fed into any spreadsheet or GNU-Plot style program [4]. See Appendix 4 for a sample of the formatted results fed into the simulator output through the Perl scripts.

5.3 Experimental Methodology

All development was done on CSOPT4 machine **(csopt4.wpi.edu)**.

The simulation environment used by the Crankshaft authors was replicated in order to be consistent when comparing the performance of the two protocols. The simulator contains a model of the EYES wireless sensor node, which includes an RFM TR1001 radio. An SNR-based radio model is used on top of a simple path-loss propagation model. A 32KHz crystal is used for timing, and the nodes are simulated as being powered by two 1.5V AA batteries supplying 3V.

Following are the various simulation parameters relevant to the results.

Table 5.1 Simulation parameters

General	
Packet payload	25 bytes
Effective radio data rate	61 kbps
Radio transmit energy	36 mW
Radio receive energy	11.4 mW
Radio sleep energy	0.0021 mW

[4]We propose to add these files to the MiXiM repository

Crankshaft and SCP-MAC*	
Sift nodes parameter	512
Number of slots	90
Contention window	9.15 ms
Poll length	0.3 ms
DATA frame size	64 bytes
ACK frame size	8 bytes

The setup is similar to the Crankshaft paper. To reiterate, it includes 96 nodes (95 nodes + 1 base station) on a field of approximately 90x50 meters. The base station is situated at a corner. The simulated nodes have a radio range of 25 meters. Average connectivity in the network is approximately 17.3 nodes. Each simulation lasts for 200 seconds and the results are the average of 20 runs with unique random seeds. This process was repeated for 8 different loads as mentioned in Section 5.2 (0.1 messages/node/sec - 0.8 messages/node/sec) [5]. It was determined that one cycle time for a message exchange cycle (MEC) is around 40 ms, providing for data+ACK exchange, and send stagger times in the simulator. This means that since Crankshaft uses 8 unicast slots, on an average, a node that has to send again has to wait around 0.3 seconds (MEC * 8). In SCP-MAC*, a sender can contend and send again in the next slot. Note that this leads to a vast difference in cycle times for both Crankshaft and SCP-MAC* (almost a factor of 8).

Routing is done using a static routing table, and hence there is no routing traffic exchanged during the simulation. Unless mentioned otherwise, the simulation results presented involve the convergecast traffic pattern. All nodes periodically send data to a sink node, which then processes or stores the data for further processing.

The performance metrics being evaluated were energy consumption, delivery ratio and latency, as defined previously (Refer to Section 1.2) - Note that these are similar to the calculations of the original Crankshaft authors.

- *Energy consumption*: The simulator records the amount of time spent by the MAC layer in the idle, collision, receive (including receive overhead and receive overhearing), transmit (including transmit overhead), and sleep states. Using this information, the energy consumed

[5]It is interesting to note that the most real world loads do not usually exceed 0.5 messages/node/sec. [25]

in each state can be calculated using specific radio (RFM1001) constants. The battery constants are used to deduce the total power in mW. Note that these energy measurements are in transmitting a wireless frame (at the MAC layer). The average energy consumption is calculated over all nodes, including the base station.

- *Delivery ratio*: For the convergecast traffic [6], we use the information about the total number of packets sent by all the nodes in the WSN and the number of packets received at the routing layer for the sink, and calculate the delivery ratio as the number of packets received at the sink divided by the total number of packets sent. This is calculated at the packet layer (layer 3). These numbers are averaged over all nodes, excluding the base station, as it is the sender in the convergecast traffic.

- *Latency*: For the convergecast traffic, latency is calculated as the time it takes for each node to send its packets to the sink, averaged over all nodes. This is also measured at the packet layer.

[6]We detail the calculation of these metrics for other communication patterns in their relevant sections

Chapter 6

Results

This chapter presents the results of our OMNeT++ simulations in accordance with the goal of the thesis, which is mainly to research ways to improve Crankshaft performance. **Section 6.1** contains the reproduction of the original Crankshaft results as a sanity check of the simulator. **Section 6.2** contains the results of investigating various areas of improvement in Crankshaft. This is to determine the parameters and the protocol settings that yield the best latency. **Section 6.3** contains a comparison of Crankshaft and SCP-MAC*. Finally, **Section 6.4** compares Crankshaft and SCP-MAC* under various types of traffic patterns. Specifically, *6.4.1* deals with one-hop traffic, *6.4.2* studies Crankshaft and SCP-MAC* with local gossip traffic pattern, and *6.4.3* discusses the results of the flooding traffic pattern.

6.1 Replication of the Crankshaft results

The first simulations run were an attempt to reproduce the Crankshaft and SCP-MAC* performance results that were reported by Langendoen et al. [5] and analyzed in Section 4.1. This involves a convergecast pattern with 95 nodes sending to the base station (sink). The specific simulation parameters are specified in section 5.3.

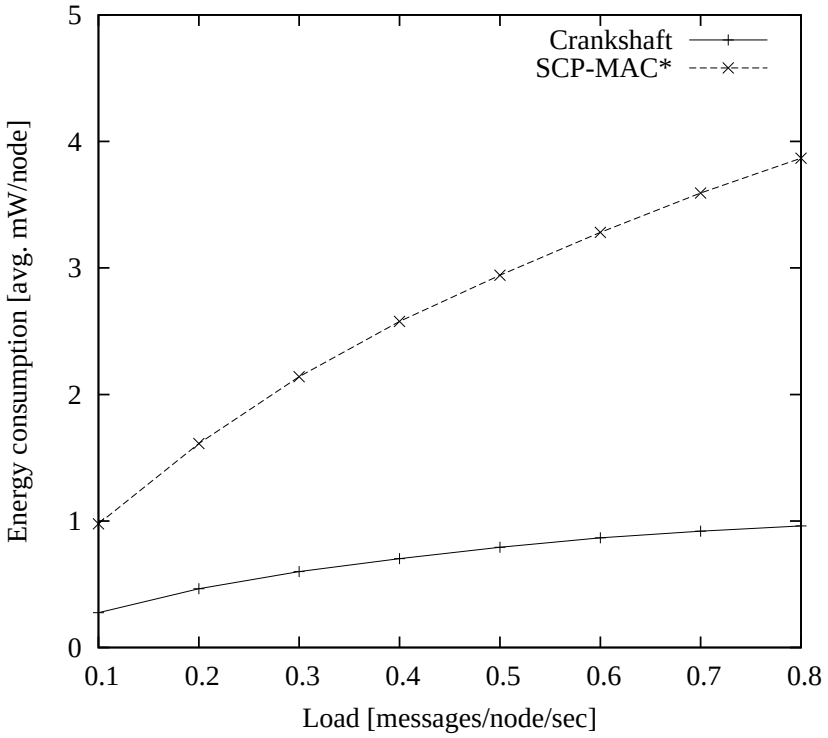

Fig. 6.1: Crankshaft vs. SCP-MAC* - Energy consumption

,

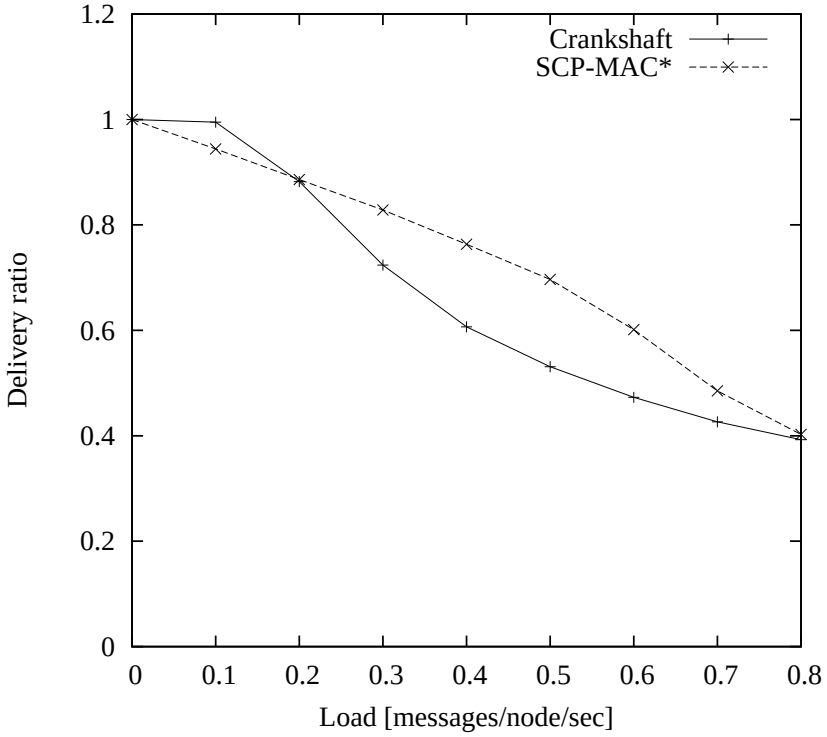

Fig. 6.2: Crankshaft vs. SCP-MAC* - Delivery ratio

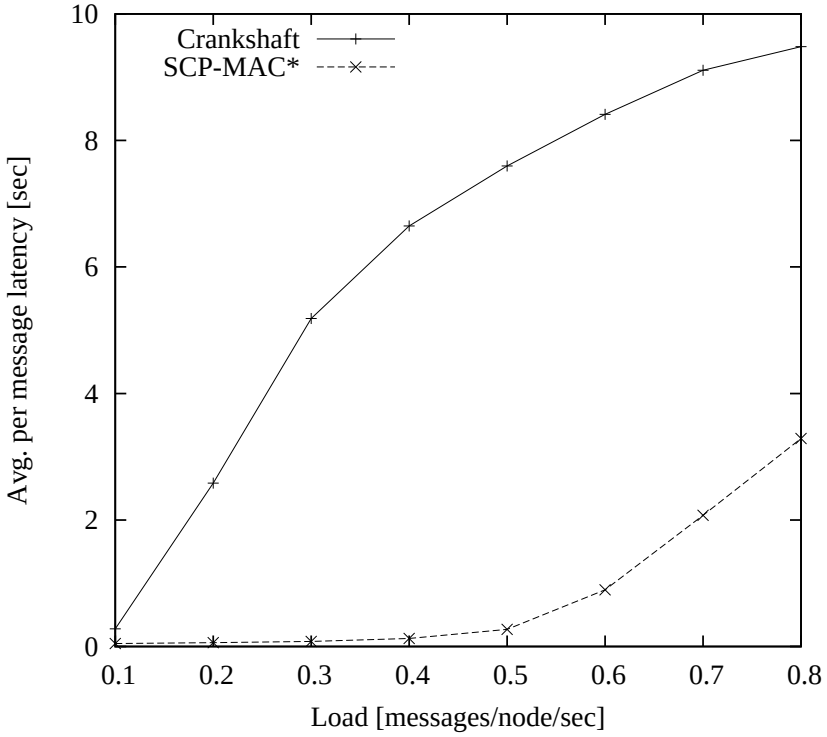

Fig. 6.3: Crankshaft vs. SCP-MAC* - Latency

The Crankshaft results presented in Figures 6.1 - 6.3 match well with the original results (See Section 4.1). The SCP-MAC* numbers do not agree at high loads. To analyze this discrepancy, Table 6.1 compares the three metrics as the original results (at U. of Delft) against in-house (WPI) measurements.

SCP-MAC* Energy [mW/node]		
Load	U. of Delft	WPI
0.1	1.0	0.977
0.2	1.6	1.612
0.3	2.1	2.14
0.4	2.5	2.577
0.5	2.9	2.942
0.6	3.2	3.28
0.7	3.5	3.592
0.8	3.8	3.868

SCP-MAC* Delivery ratio		
Load	U. of Delft	WPI
0.1	0.95	0.944
0.2	0.90	0.886
0.3	0.85	0.828
0.4	0.80	0.763
0.5	0.70	0.696
0.6	0.65	0.601
0.7	0.58	0.485
0.8	0.50	0.402

SCP-MAC* Latency [sec]		
Load	U. of Delft	WPI
0.1	Cannot approximate at this level	0.047
0.2		0.059
0.3		0.080
0.4		0.126
0.5		0.270
0.6	0.3	0.896
0.7	1.0	2.075
0.8	2.0	3.288

Table 6.1: SCP-MAC* - Comparison of in-house data vs. original data

(Varying loads 0.1 messages/node/sec - 0.8 messages/node/sec)

It is observed that the WPI simulations have lower delivery ratio and significantly higher latency at loads at or above 0.6 messages/node/sec. The cause for this was not discovered. Possible reasons for the discrepancy are some type of timing mismatch, or a difference in the implementation of the random number generators. Thus, at high loads, our SCP-MAC* results are not completely reliable, and further research is needed.

6.2 Investigating Crankshaft

This section considers the various parameters and settings within the Crankshaft protocol that can impact its latency. In the following sections, Crankshaft is analyzed under the convergecast communication pattern. Note that the spanning tree set up for the convergecast pattern is static, i.e., the same for a given topology under various runs. The default setting for the protocol includes using Sift, with 8 unicast and 2 broadcast slots. The default protocol also uses ACKs, with three retries and a 70% retry probability. The base station is located at the corner (except in the topology results where we move the base station). The base station also listens on all the unicast slots.

6.2.1 The impact of Sift

The effect of removing Sift and replacing it with a uniform (random) contention resolution mechanism was analyzed. A uniform contention resolution means that nodes choose their slot in the contention window randomly, with each node having an equal probability of choosing a given slot. The motivation for this was to understand if there is any advantage of a random contention resolution method for dense deployments. Note that the number of slots (90) nearly equals the number of sensor nodes (96).

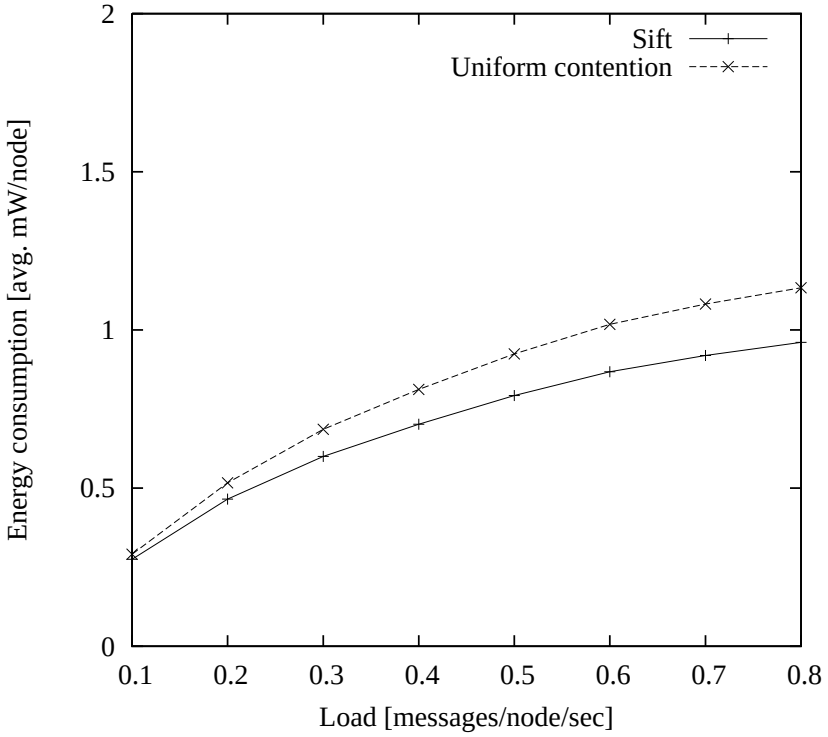

Fig. 6.4: Crankshaft: Uniform contention vs. Sift - Energy consumption

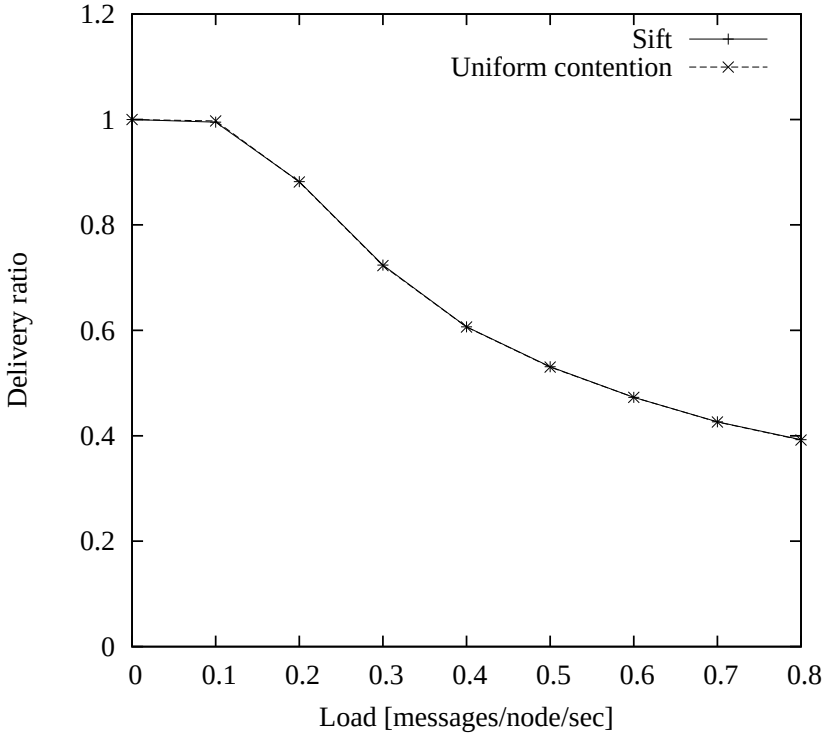

Fig. 6.5: Crankshaft: Uniform contention vs. Sift - Delivery ratio

Removing Sift had almost no effect on delivery ratio in Figure 6.5. This is attributed to the fact that there is sender queuing along the multi-hop routes. Crankshaft with Sift saves approximately 10% on energy above loads of 0.4 messages/node/sec (see Figure 6.4). Sift, by design, reduces the number of contention window collisions compared to the uniform contention method. But the argument here is that since the contention window size (90) is close to the total number of nodes, the number of these collisions is very small. The biggest contribution to energy reduction shown in Figure 6.6 is due to the idle listening component, which is lower for Sift.

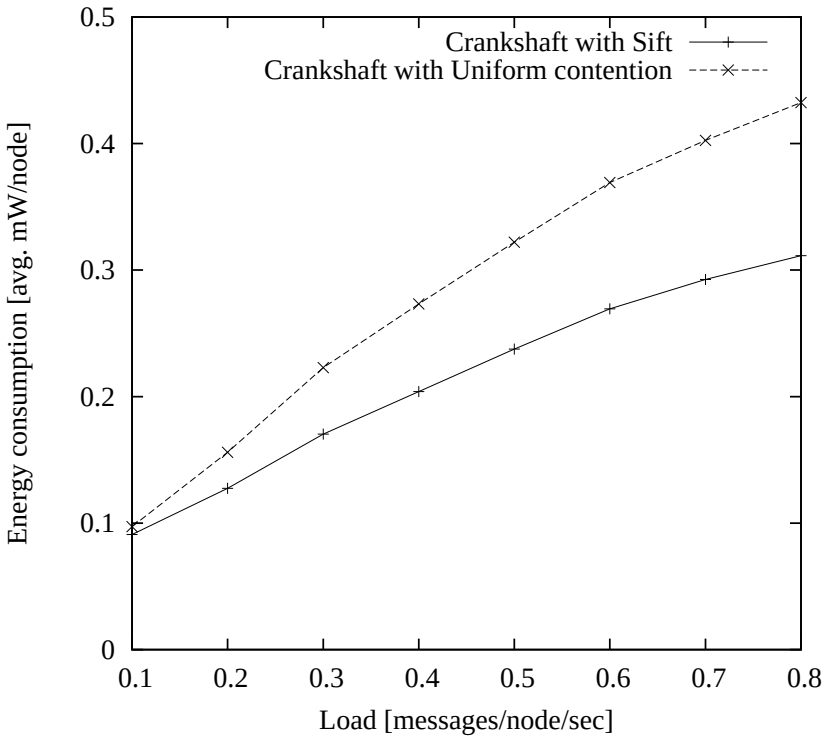

Fig. 6.6: Crankshaft: Uniform contention vs. Sift - Energy consumed in idle listening

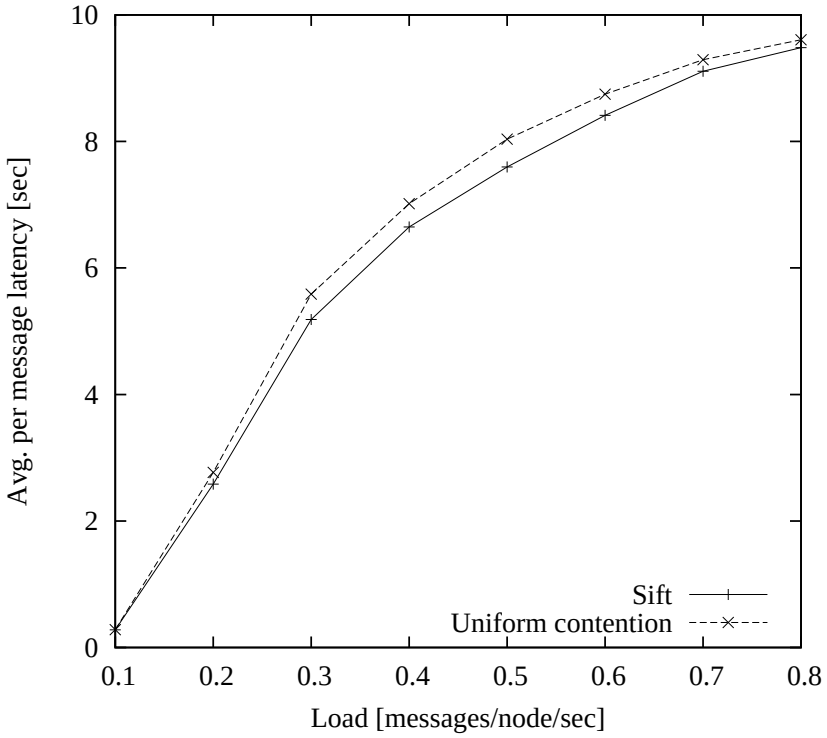

Fig. 6.7: Crankshaft: Uniform contention vs. Sift - Latency

The latency for Sift is slightly better (Figure 6.7), as it reduces the average amount of time between the start of sending and the wake up moment of the receiver. Clearly, Sift improves the performance of Crankshaft, and this impact should be higher at lower contention window sizes, or at least, at contention window sizes much lesser than the total number of nodes.

6.2.2 The impact of ACKs

The next set of simulations was compare the performance of Crankshaft with and without ACKs. It was mentioned before that retransmissions are considered expensive for WSNs, and most WSN MAC protocols usually do not use acknowledgements. Though they are designed to improve reliability, at meaningful and high loads in dense networks, they can lead to congestion in multi-hop traffic, increasing the latency. Since the major objective of this investigation was to find ways to reduce the latency of Crankshaft, simulations were run with ACKs disabled.

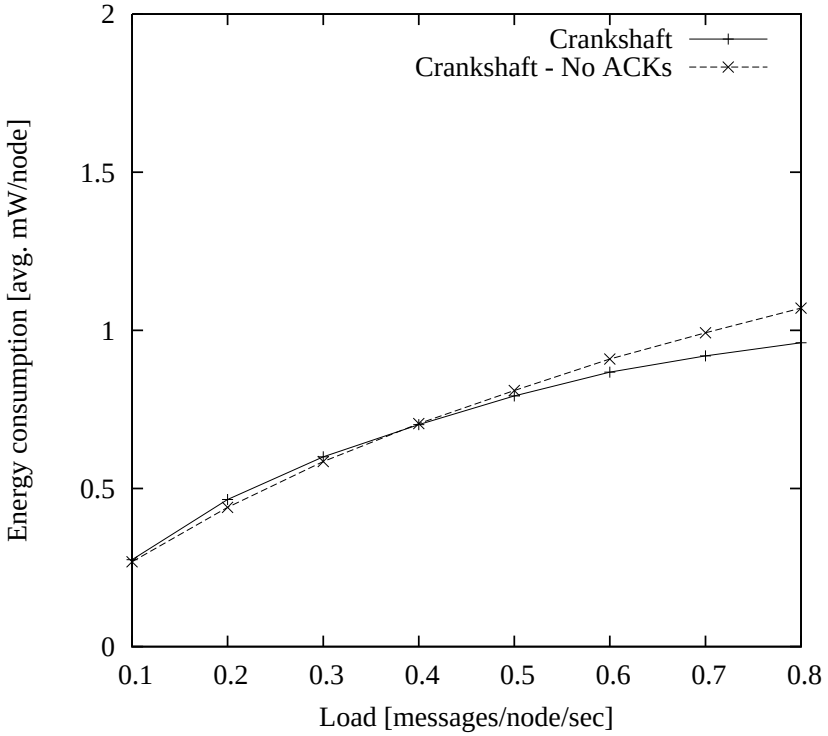

Fig. 6.8: Crankshaft with and without ACKs - Energy consumption

Crankshaft shows an increase in energy consumption without ACKs in Figure 6.8. This is attributed to the following factors. In the No ACKs case, there are fewer retries of frames and less overhearing overall. But idle listening is the dominating factor, as it is much higher without ACKs

(as seen in Figure 6.9). This is because the cycle time for Crankshaft takes ACKs into account. Moreover, the Message Exchange Cycle (MEC) without ACKs still includes the time and thus the receive wake-up energy for ACKs.

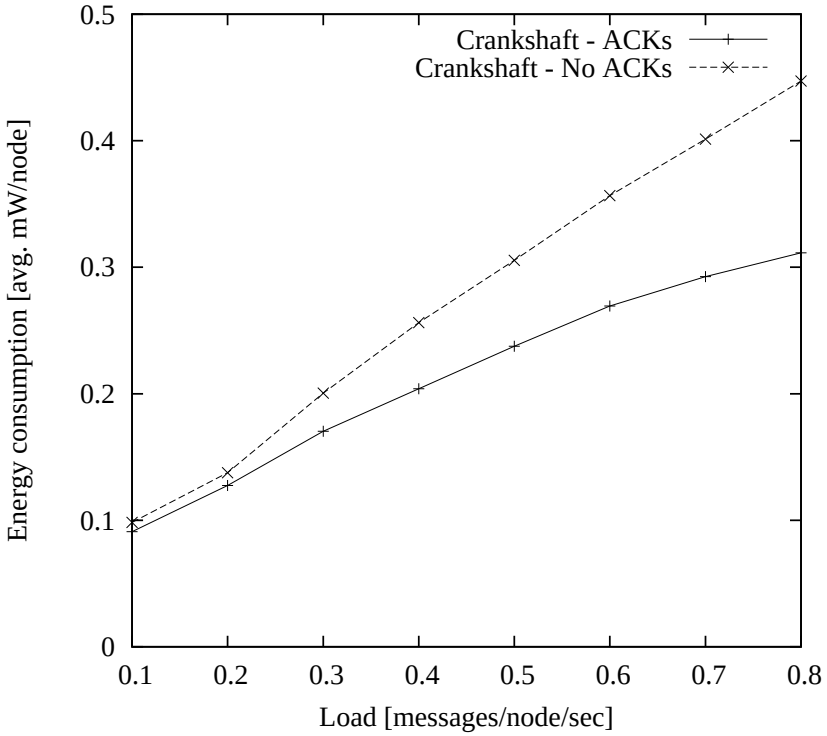

Fig. 6.9: Crankshaft with and without ACKs - Energy consumed in idle listening

Crankshaft does not show much of a performance difference in terms of delivery ratio (Figure 6.10). Though this is a little non-intuitive, the argument is that at high loads, the number of messages sent in the No ACKs case is much lower, and comparatively more successful messages are sent compared to the ACKs case [1]

[1]Note, also, that with ACKs, Crankshaft's retransmission parameters are three retries with 70% retry probability. This impacts the delivery ratio at high loads

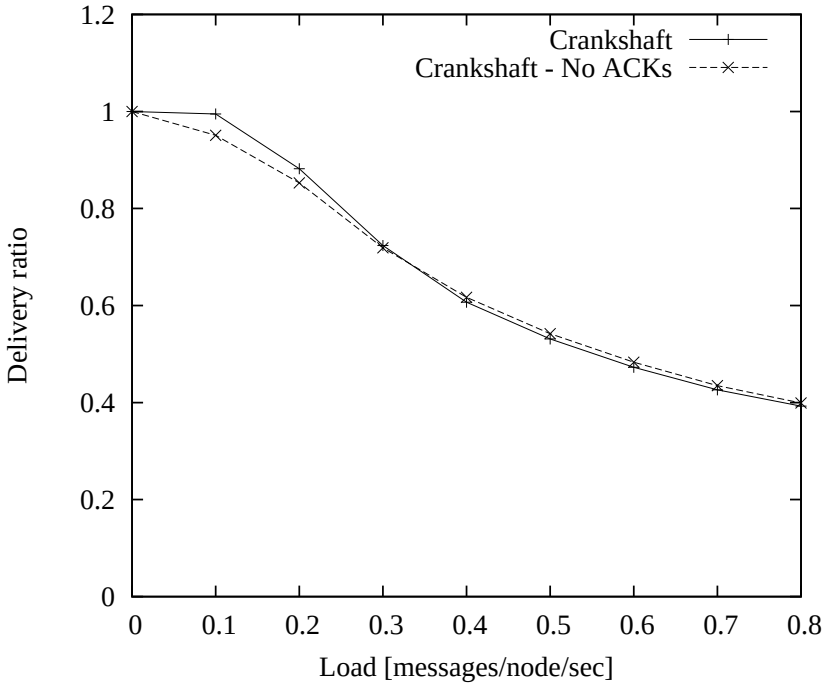

Fig. 6.10: Crankshaft with and without ACKs - Delivery ratio

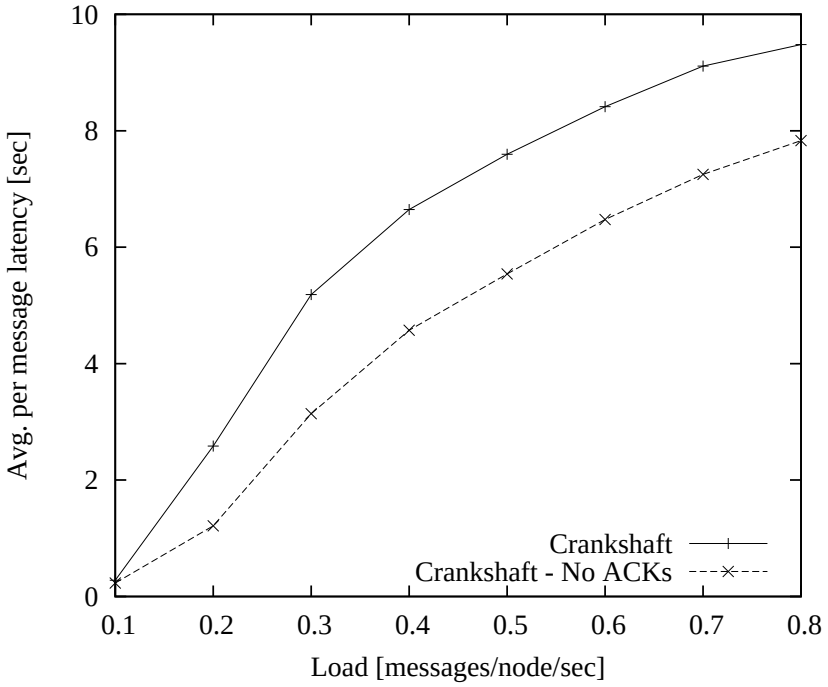

Fig. 6.11: Crankshaft with and without ACKs - Latency

The biggest improvement is seen in terms of latency (Figure 6.11). It is reduced by almost 28% at 0.5 messages/node/sec. This is due to the fact that there are no retries because of no ACKs being received. This is due to the fact that the traffic bottleneck near the sink is reduced because of no retries. This might be detrimental for networks requiring extreme reliability, but for dense monitoring networks, where there is usually redundancy in data due to the number of sensors operating in a particular environment, this can be a positive change.

One argument is that acknowledgements are useful when considering node locations within the topology. An example would be that a node sufficiently far away from the sink, outside its radio range and outside the general multi-hop vicinity can never know if its messages are being successfully sent.

6.2.3 Retry parameters

The next step was to investigate the authors' choice of the retry parameters in Crankshaft (three retries with a 70% retry probability) and study the effect of changing the number of retries in overall Crankshaft performance. Since retransmissions are expensive, changing the number of retries can potentially improve its latency. The number of retries was varied from 1 to 3 in Figures 6.12 to 6.14

The results are interesting because they show that the authors' choice is not justified in their simulations. Decreasing the number of retries brought down the latency of Crankshaft while only slightly affecting the energy consumption and delivery ratio. This result is furthermore evidence that a multi-hop network leads to congestion. When congestion occurs, reducing traffic (even if it is ACKs) by reducing retries can improve the performance.

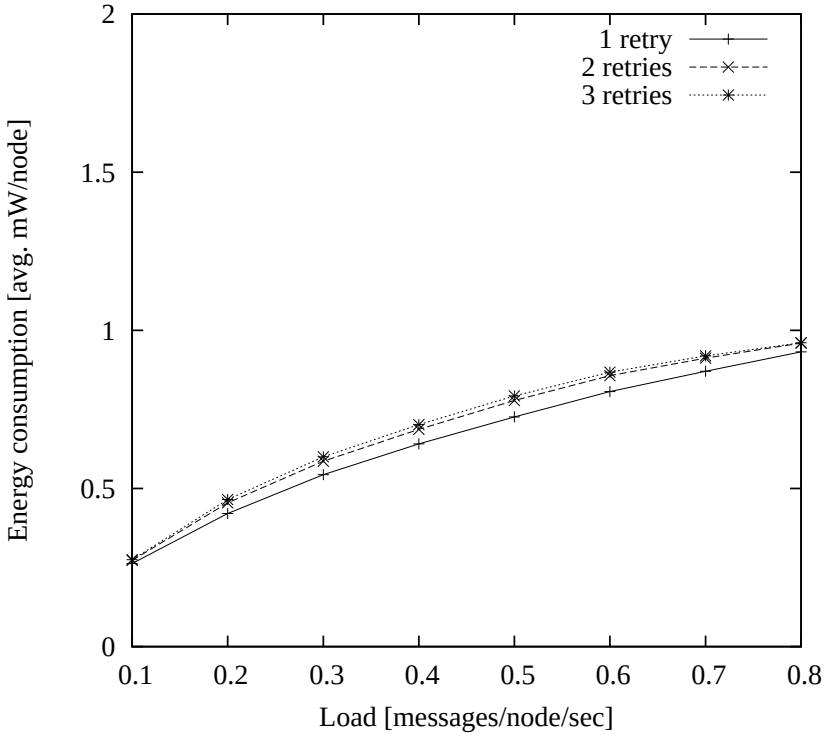

Fig. 6.12: Crankshaft: Varying number of retries - Energy consumption

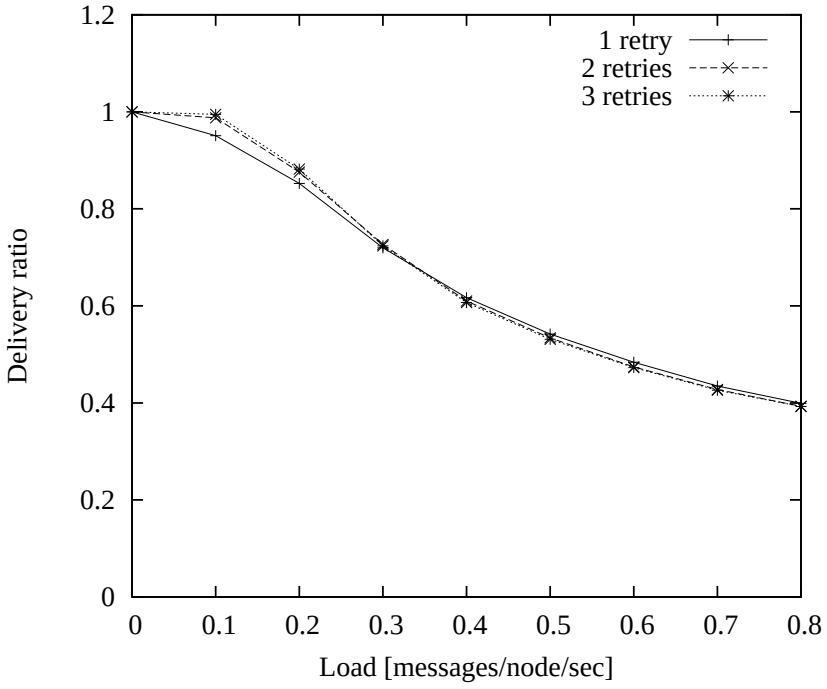

Fig. 6.13: Crankshaft: Varying number of retries - Delivery ratio

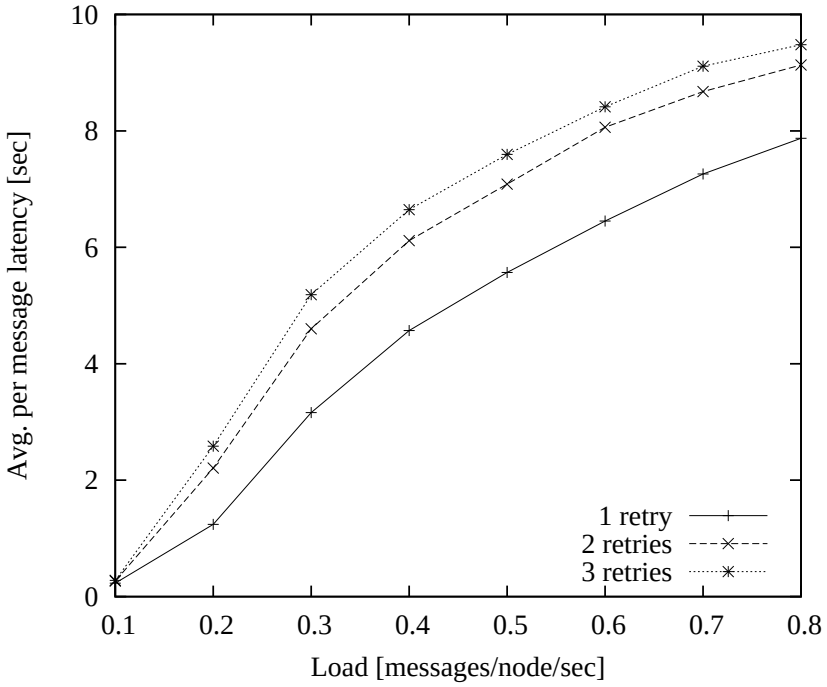

Fig. 6.14: Crankshaft: Varying number of retries - Latency

6.2.4 Changing the number of broadcast slots

Since broadcast slots waste energy in convergecast traffic, the next set of simulations were run to study the effect of decreasing the number of broadcast slots. Note, the number of unicast slots stays the same: 8 (The default setting, as mentioned before, is 8 unicast slots and 2 broadcast slots). The number of broadcast slots was varied from 0 to 2. The results shown in Figures 6.15 - 6.17 are convergecast simulations.

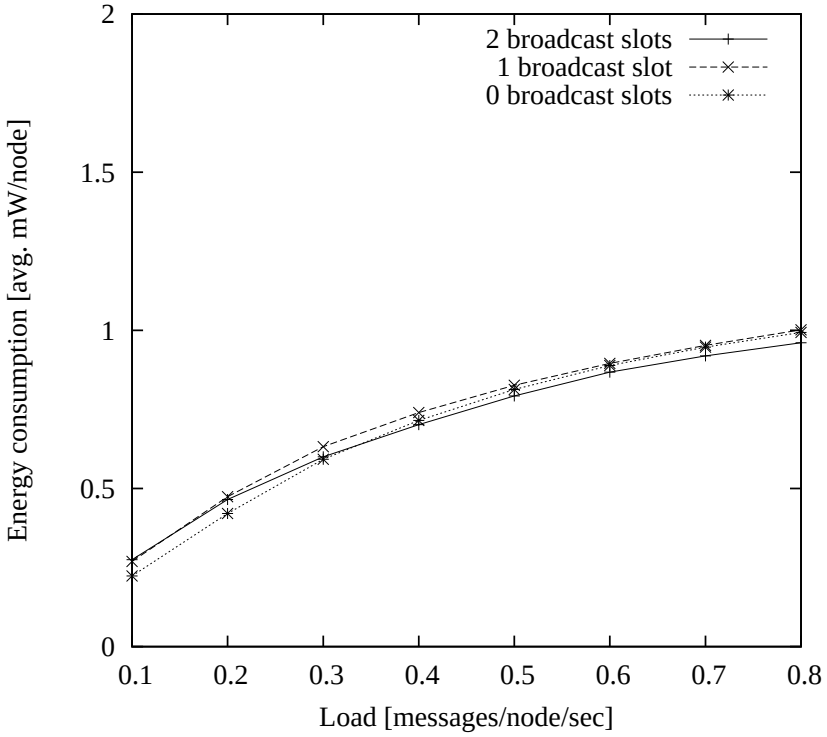

Fig. 6.15: Crankshaft: Changing number of broadcast slots - Energy consumption

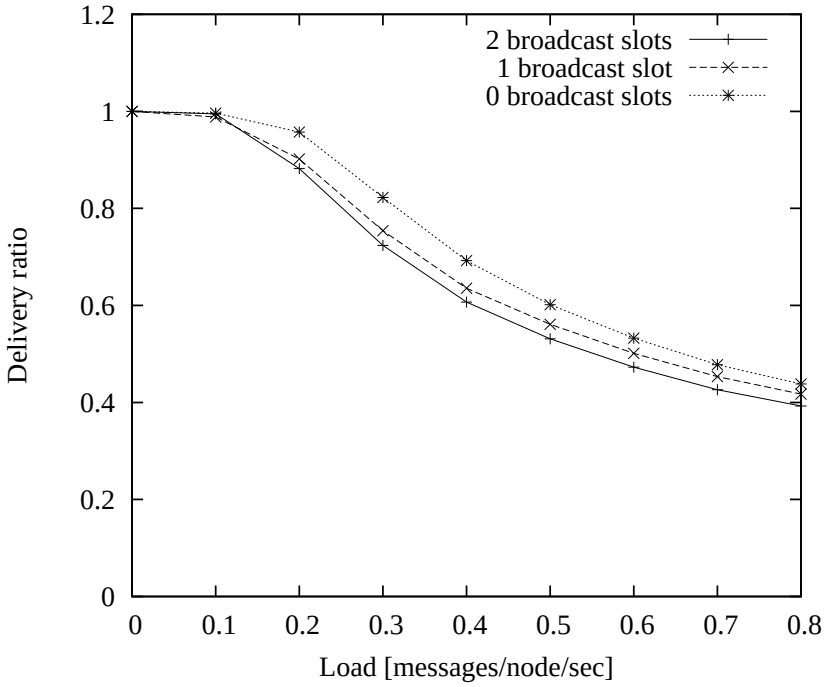

Fig. 6.16: Crankshaft: Changing number of broadcast slots - Delivery ratio

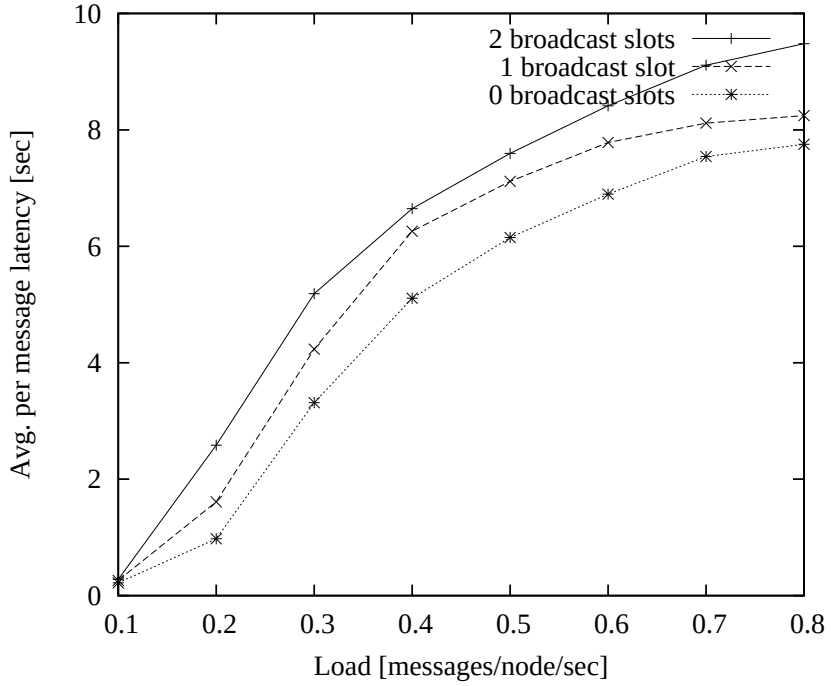

Fig. 6.17: Crankshaft: Changing number of broadcast slots - Latency

Crankshaft's latency is better when the number of broadcast slots was decreased, and this is expected because the overall cycle time for Crankshaft is reduced. The delivery ratio is also better as the per node bandwidth is improved due to more of the slots within the cycle being used effectively.

6.2.5 Two-phase contention

Since SCP-MAC (Note: Original SCP-MAC, not SCP-MAC*) is shown to be a low duty cycle and low latency protocol [4], the next set of simulations were designed to investigate whether the two-phase contention used in SCP-MAC can improve the latency of Crankshaft. This replaces Sift in Crankshaft. The following is the algorithm for two-phase contention as described in the SCP-MAC paper:

- Node performs carrier sense by randomly selecting a slot in the first contention window - CW1

- If the channel is idle, proceed and send a wakeup tone till the rest of the window.

```
else, abort transmission until next frame.
- If the node is successful in sending the wakeup tone, enter second contention by
randomly selecting a slot again - CW2
```

This was the corresponding pseudo-code in context of the code-base (the OMNeT++ simulator):

```
- Select a random slot by using the intuniform(...)  function.  Then set timeout to
simulate radio settling.  Finally, set timeout to the rest of the window.
- Check if the channel is idle by calling the getRssi() function.
If getRssi() <= 0.5 // Channel is idle
Transmit tone by using setRadioTransmit(), then call intuniform(...)  again to select
a random slot to complete the second phase of contention.  Set timeout to the rest
of the window.
else If getRssi() > 0.5 // Channel is busy
Wait till next frame, i.e., do nothing
```

Thus only the nodes that are successfully able to send a tone participate in the second phase of contention, the others end up waiting and the contention/data transmission occurs before the next frame.

Figures 6.18 - 6.19 show the results from three sets of simulations: Crankshaft (with Sift), Crankshaft with two-phase contention, and Crankshaft with two-phase contention and ACKs disabled.

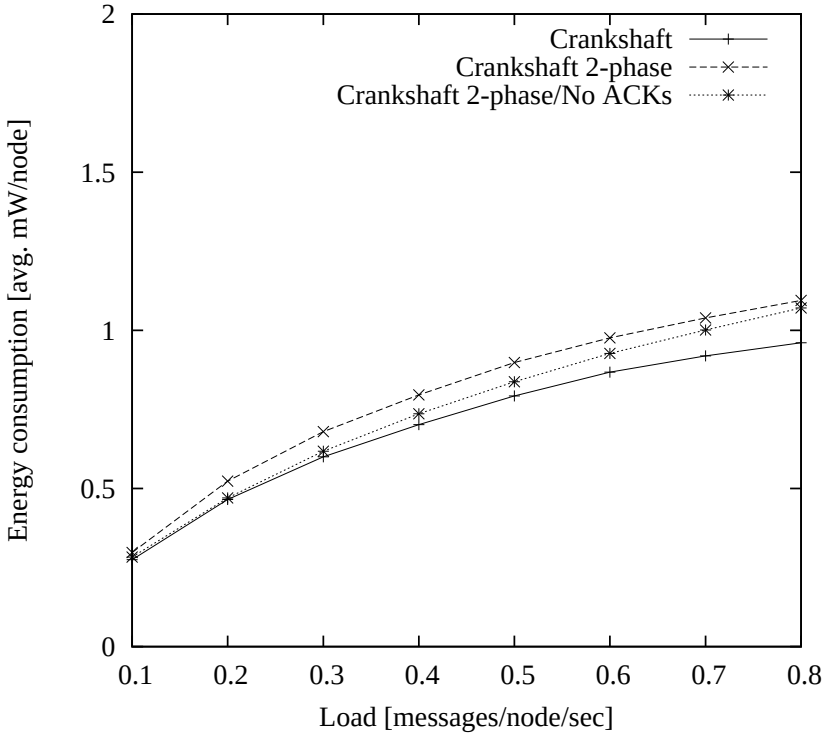

Fig. 6.18: Crankshaft: Effects of two phase contention - Energy consumption

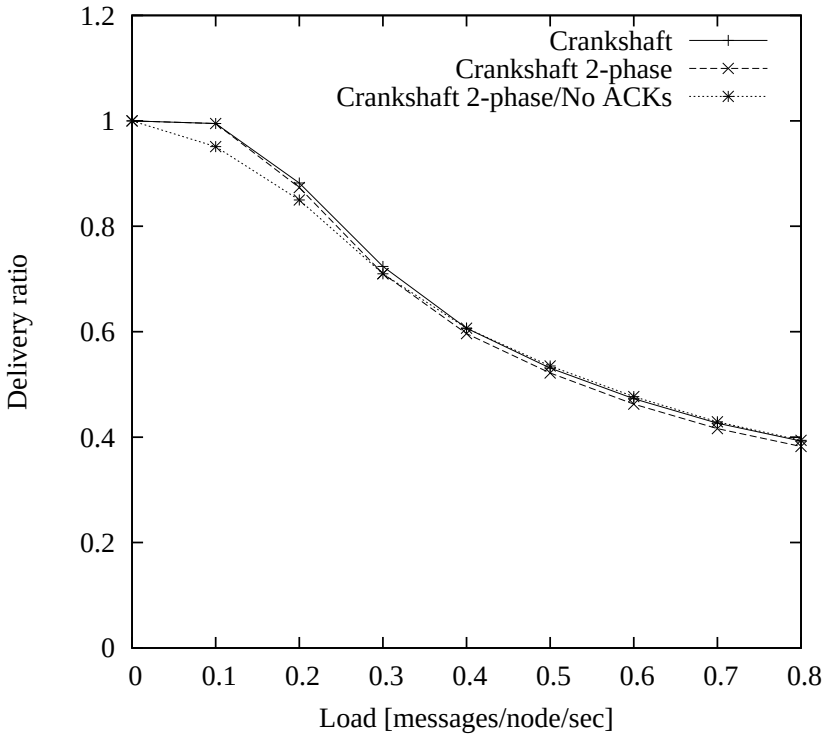

Fig. 6.19: Crankshaft: Effects of two phase contention - Delivery ratio

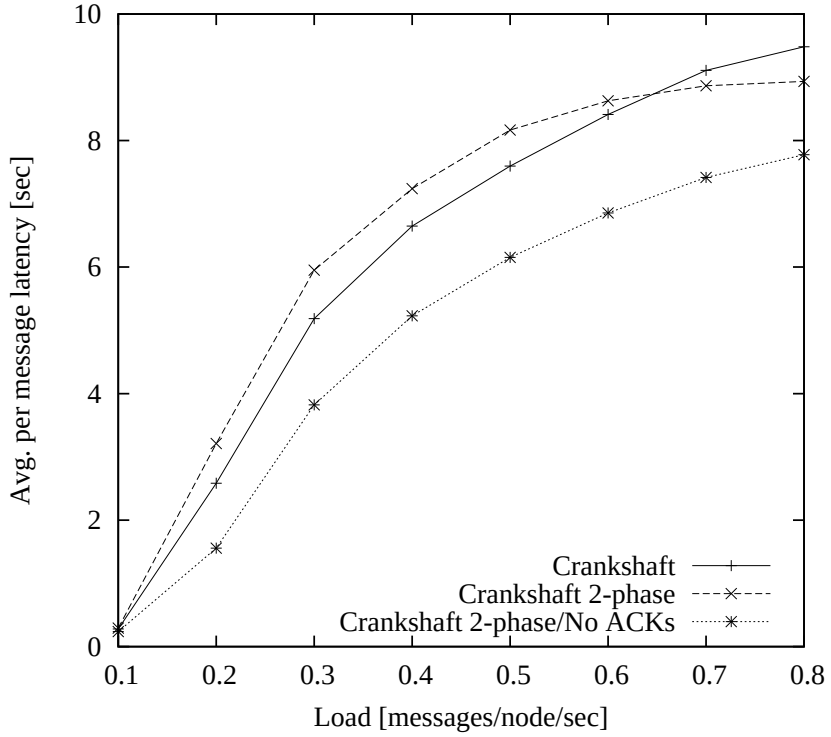

Fig. 6.20: Crankshaft: Effects of two phase contention - Latency

The delivery ratio results and the energy consumption show little difference. The most discouraging result is that the latency for Crankshaft with two-phase contention actually increases at low loads, though at very high loads the latency is lower. This is probably due to the fact that the amount of contention window collisions is reduced by the two-phase method at these loads, as it handles winner collisions in the contention window, unlike Sift.

The straightforward result from this observation is that Sift seems to be the better choice in dense networks for Crankshaft. Crankshaft with two-phase contention and No ACKs has the best latency, but it is not as good as Crankshaft with Sift and No ACKs, as shown in Figure 6.21

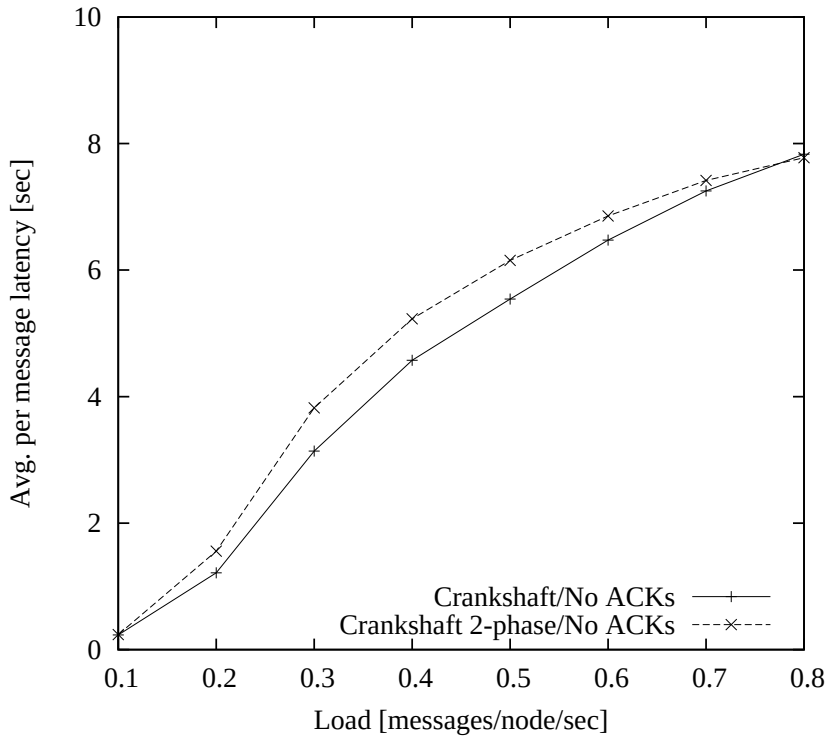

Fig. 6.21: Crankshaft: Comparing the Sift/No ACKs case to two phase contention/No ACKs - Latency

6.2.6 Changing the simulation topology

The next phase of this investigation was to study the effect of changing the topology on the Crankshaft performance. The only topology simulated thus far was the one specified in the original Crankshaft paper (dense network with its base station at the corner). In these simulations, we revert back to the default setting for the protocol (from the earlier sections). Crankshaft, as mentioned here, uses Sift, with 8 unicast and 2 broadcast slots. The protocol also has ACKs (with three retries and a 70% retry probability), and the convergecast traffic pattern is simulated.

Changing the sink location

The first investigation performed was to change the position of the base station relative to the other nodes. When the field is 90mx50m and the radio range is 25m, the base station location yields multi-hop traffic for a large percentage of the nodes. Moving the sink to a better location increases the one-hop neighbors of the sink, and should reduce the latency. The base station is moved from

the corner to the middle of the network (see Figure 6.24) and also to the sparser side (see Figure 6.26).

Figures 6.23, 6.35 and 6.27 show the distribution of the hop counts for the three different topologies simulated (These hop counts correspond to the 95 nodes sending to the base station). The density of close neighbors (one-hop, two-hop) is much higher with the sink in the middle; with the highest percentage of one-hop neighbors. When the sink is moved to the sparse location, a higher percentage of nodes use three or more hops to send to the sink.

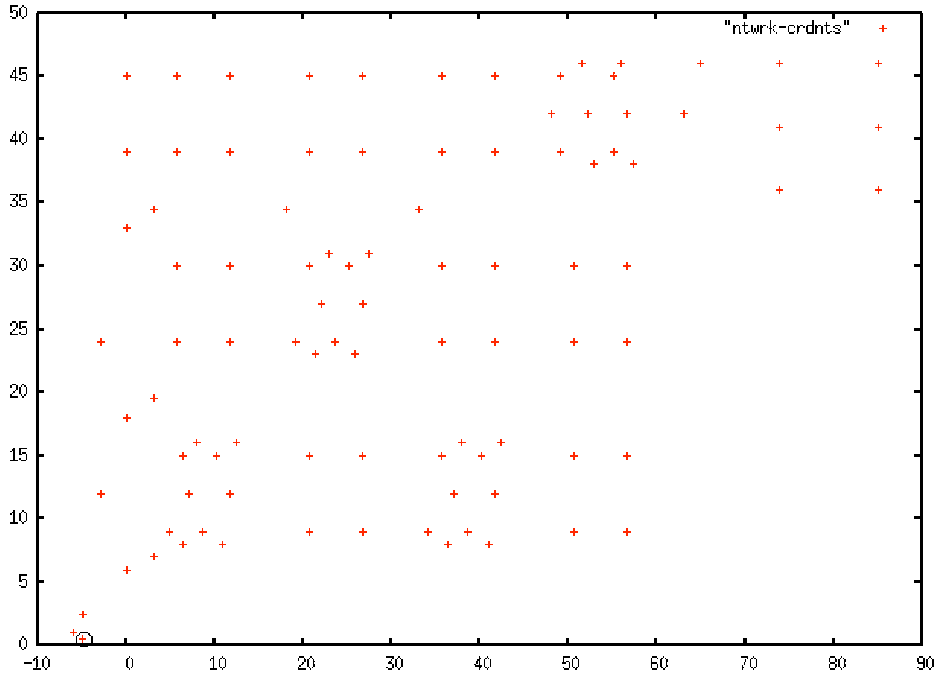

Fig. 6.22: Original topology (96 nodes) - Sink at the corner

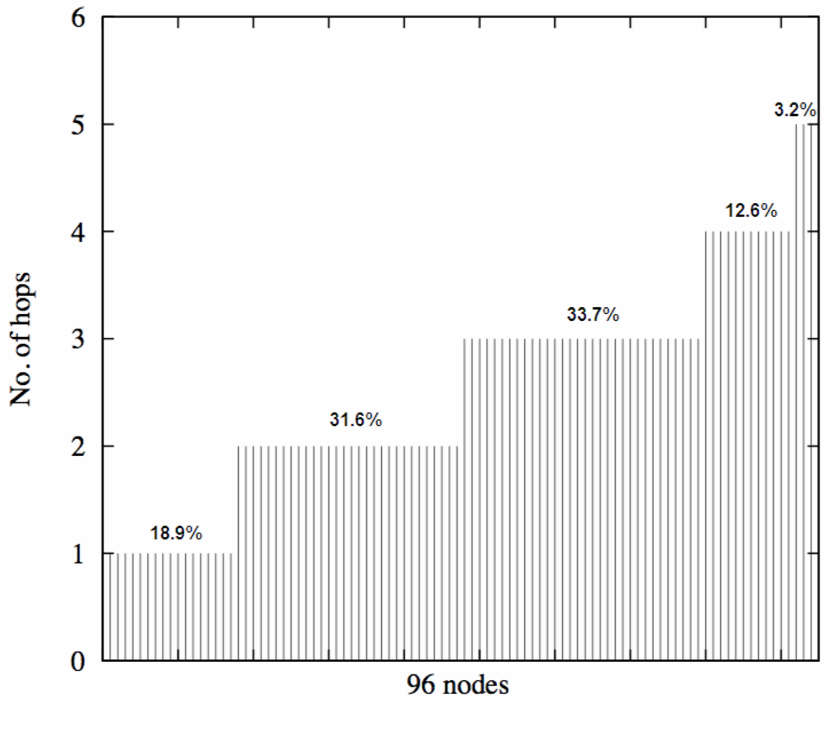

Fig. 6.23: Distribution of hop count - Sink at the corner

(Average number of hops: 2.5)

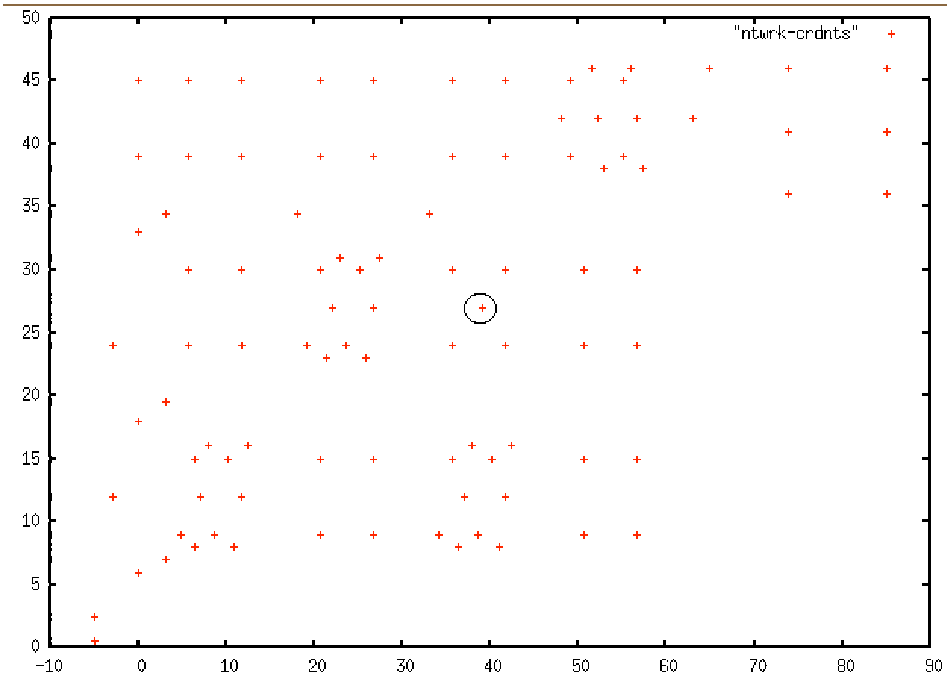

Fig. 6.24: Sink in the middle (96 nodes)

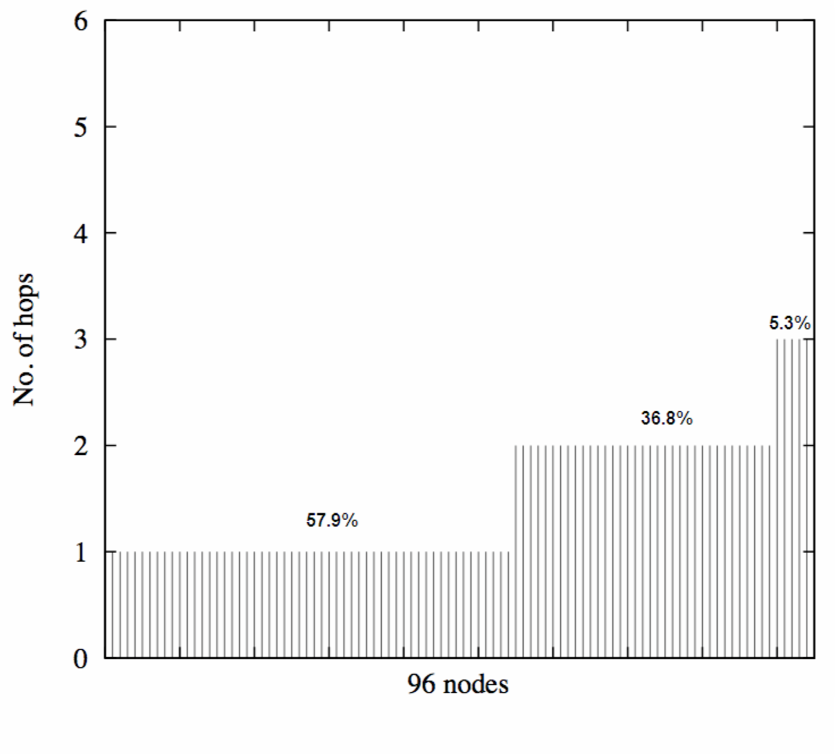

Fig. 6.25: Distribution of hop count - Sink in the middle

(Average number of hops: 1.47)

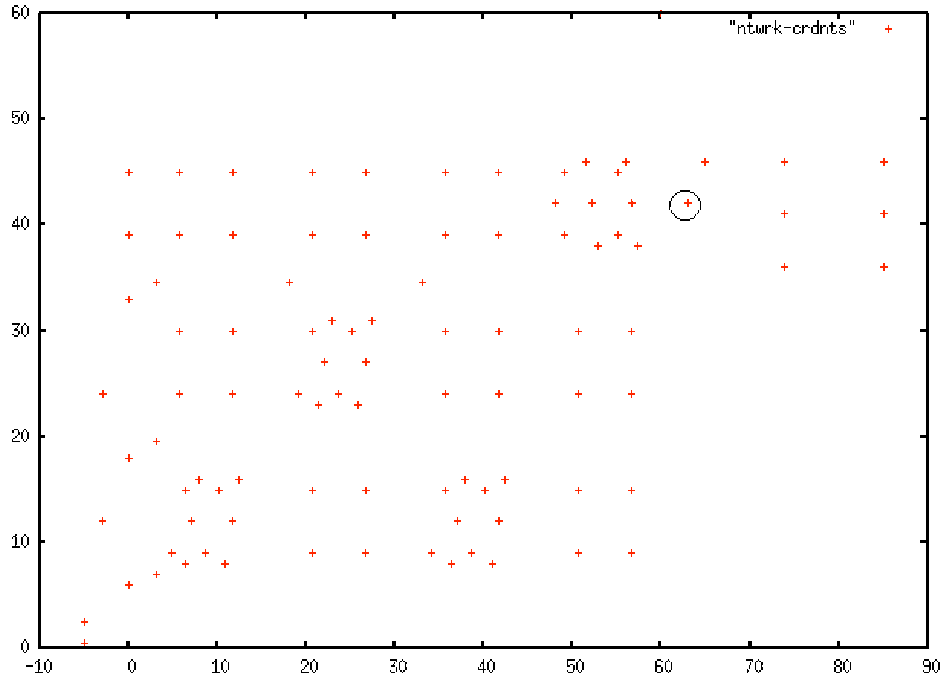

Fig. 6.26: Sink at the sparser side (96 nodes)

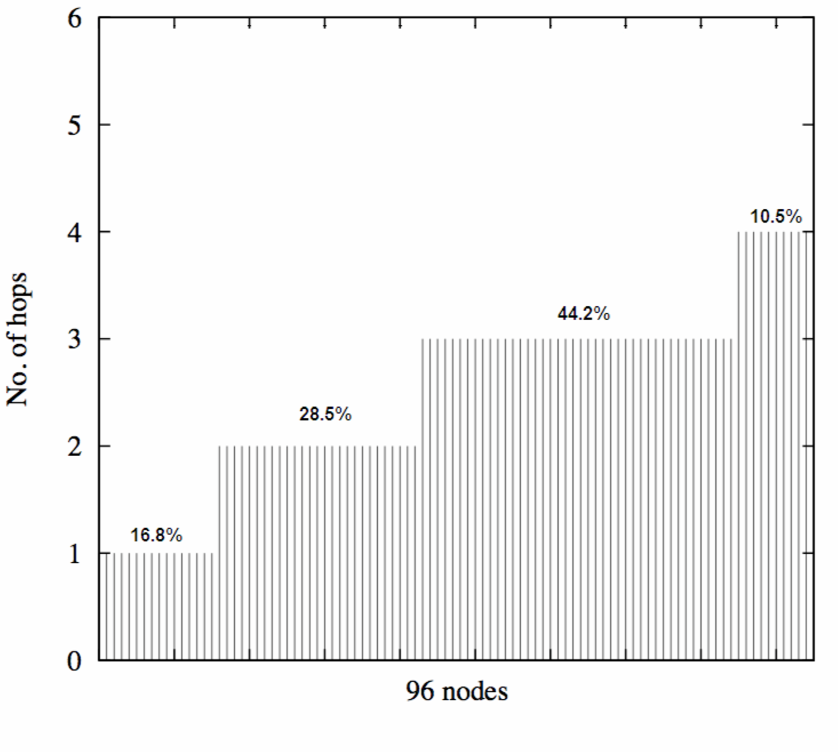

Fig. 6.27: Distribution of hop count - Sink at the sparser side (Average number of hops: 2.48)

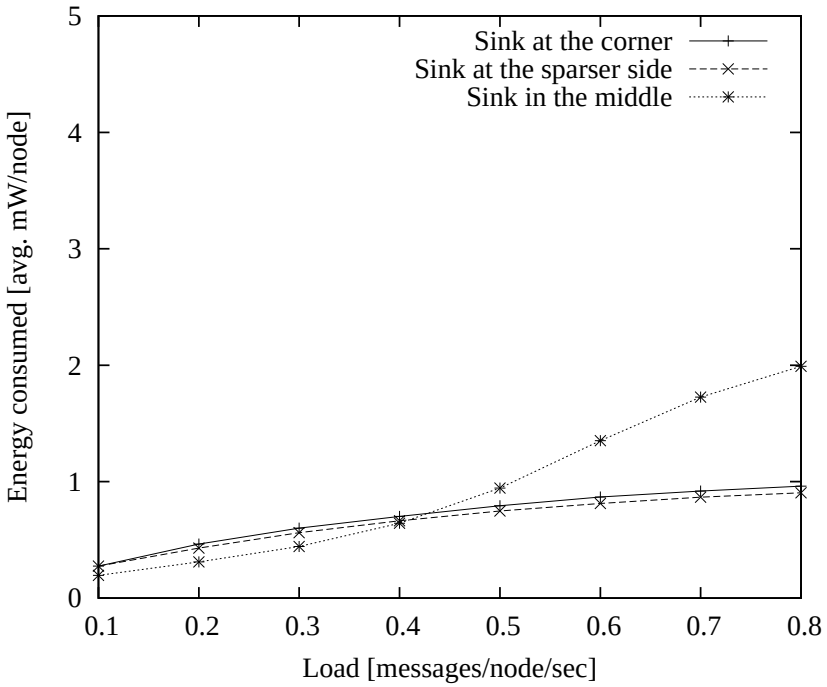

Fig. 6.28: Crankshaft: Changing sink position - Energy consumption

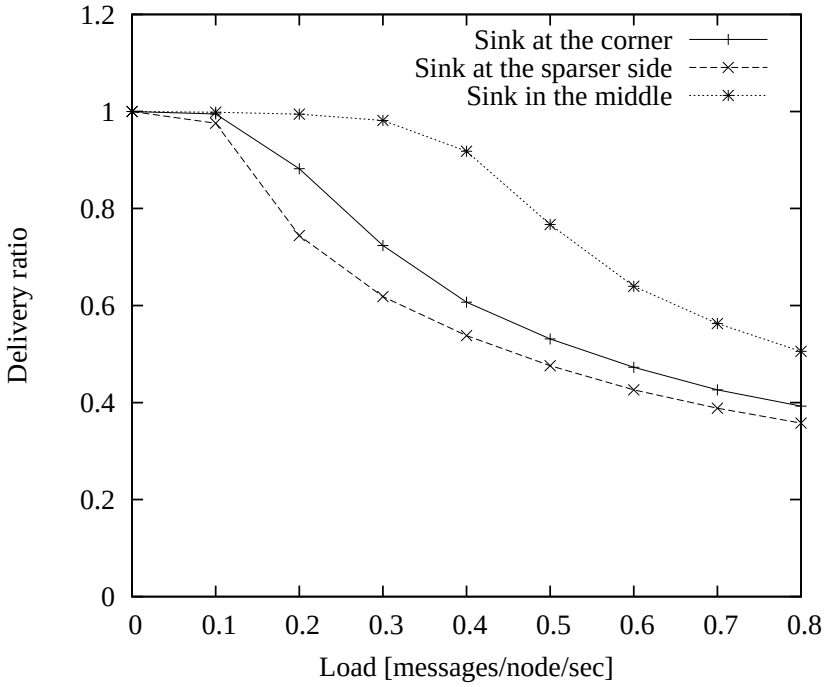

Fig. 6.29: Crankshaft: Changing sink position - Delivery ratio

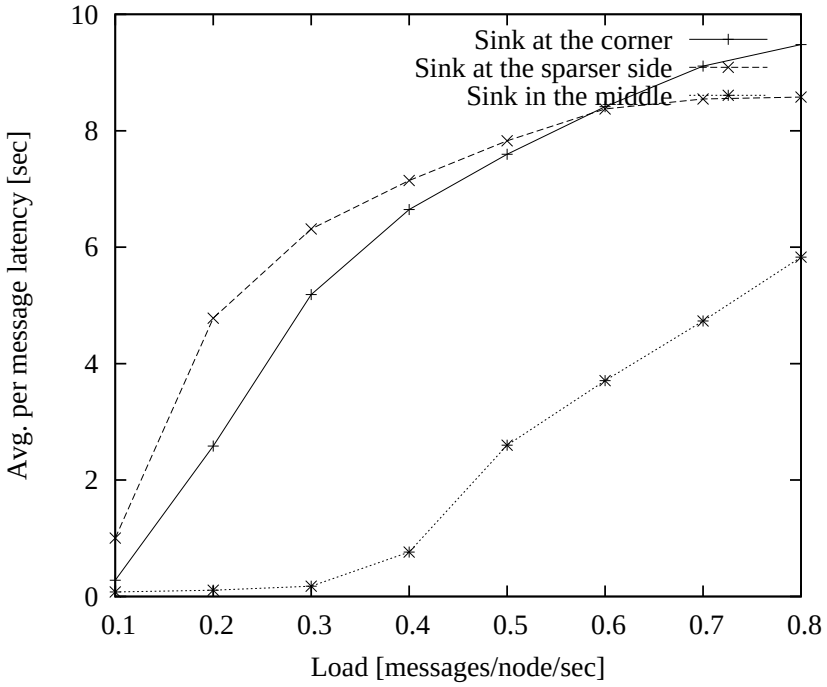

Fig. 6.30: Crankshaft: Changing sink position - Latency

From Figure 6.28, the energy consumption when the node was in the middle was the best until medium loads, and at high loads, the energy consumption worsened (0.4 messages/node/sec being the pivot). This is due to the fact that there are more collisions at high loads when all nodes try to send through nodes close to the sink. When the sink is in the middle, the overall number of multi-hop packets sent due to the sink is reduced, but the contention due to the base station is increased. This shows that studying WSN MAC protocols in multi-hop traffic is non-trivial as there is contention at the senders and more collisions at hops close to the base station.

Both the delivery ratio (Figure 6.29) and latency (Figure 6.30) performance were the best when the sink is in the middle of the WSN. The delivery ratio results show that multi-hop packets have a lower probability of reaching the destination, as the other two cases simply send more packets. The latency when the sink is in the middle is reduced by almost 66%. This is a major result in that topology and routing have a bigger influence than many of the protocol changes itself.

The worst case is when the sink is at the sparser side. Path fading plays a role for nodes that are far away from the sink, apart from the increased number of hops, which leads to congestion and retries.

Varying the number of nodes

Understanding the effect of varying the number of nodes is important because it compares a WSN MAC protocol's ability to scale and react to network density. Moreover, correlation between the simulation parameters and the number of nodes can be studied. The next set of simulations changed the WSN from 96 nodes to include runs with 24, 48 and 192 nodes. Figures 6.31 - 6.33 show the network topology at these densities. Note that Figure 6.22 contains the original topology.

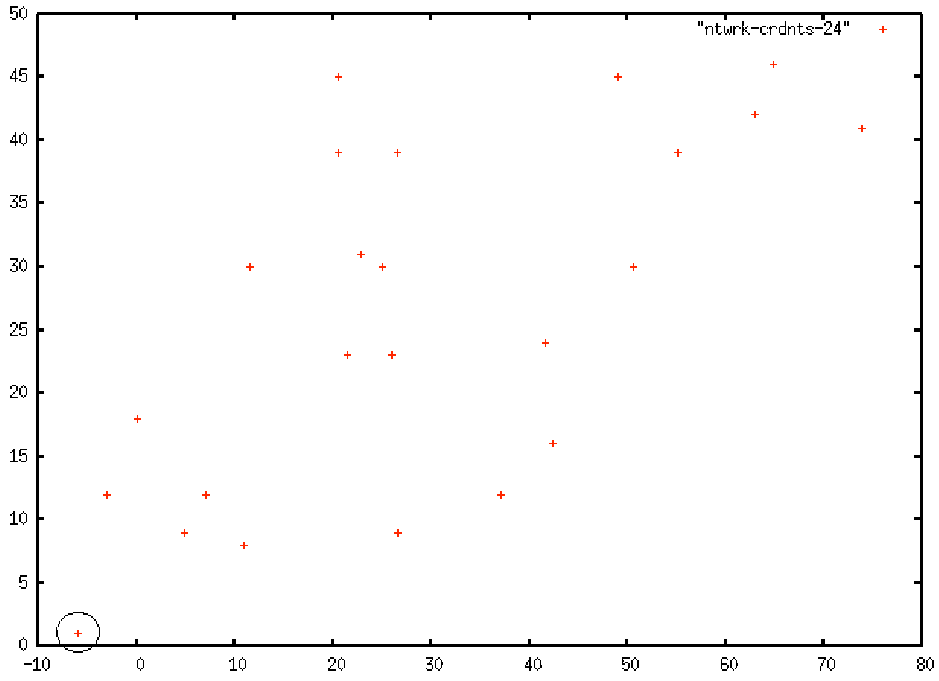

Fig. 6.31: 24 nodes in total

(Average number of hops: 2.01)

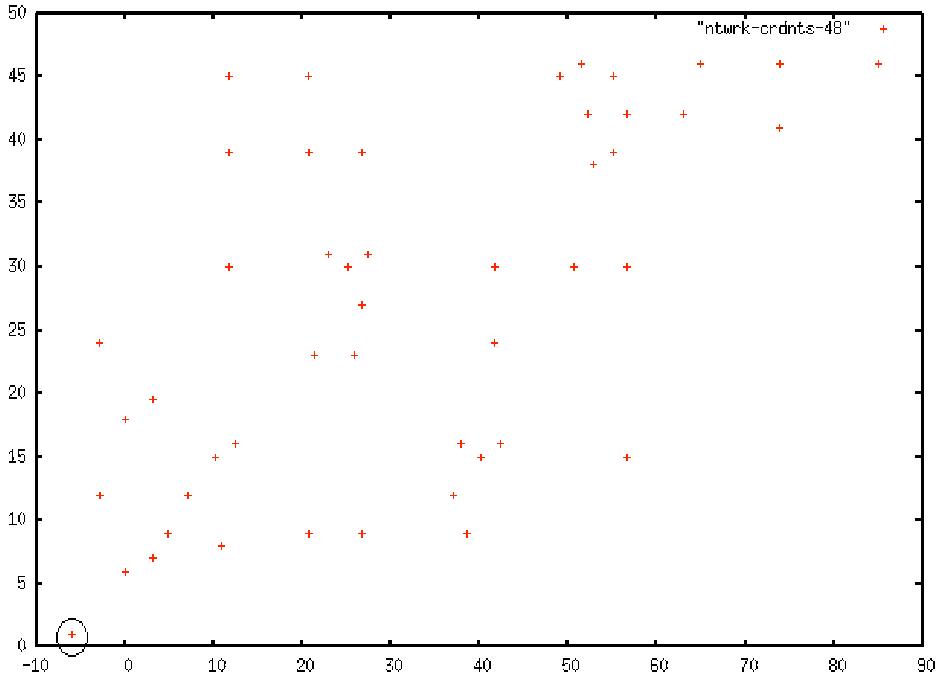

Fig. 6.32: 48 nodes in total

(Average number of hops: 2.2)

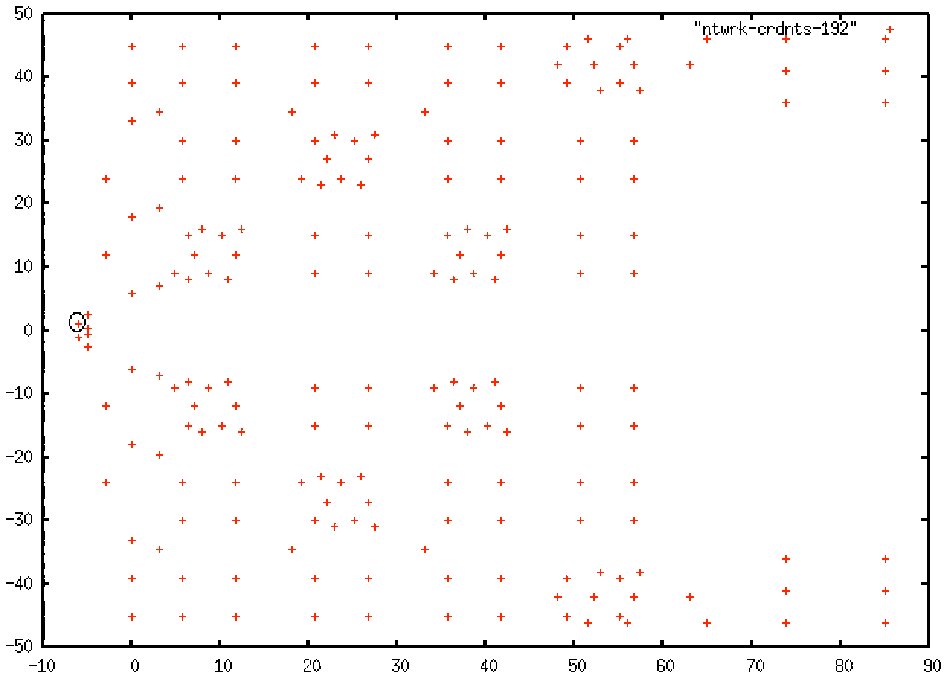

Fig. 6.33: 192 nodes in total

(Average number of hops: 2.5)

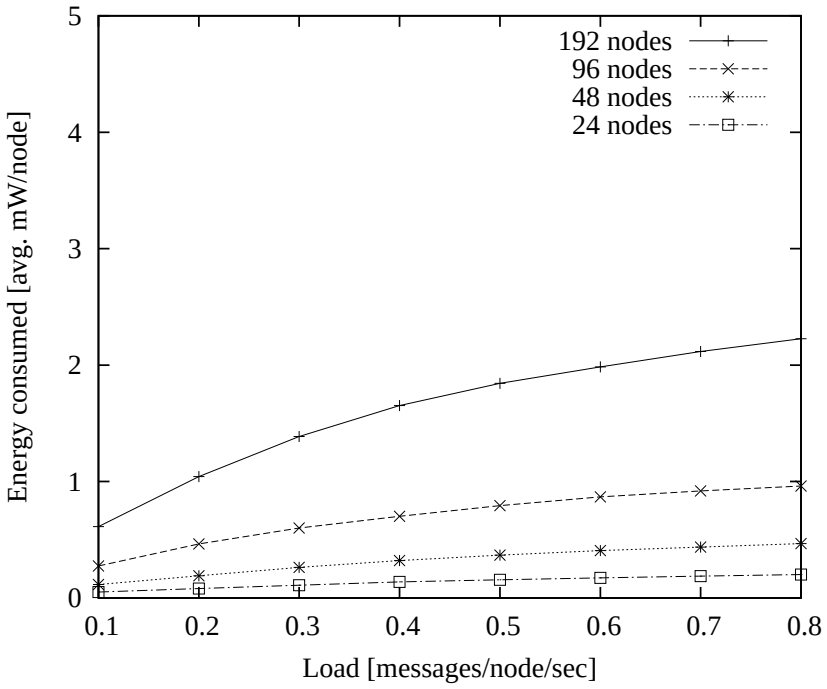

Fig. 6.34: Crankshaft: Varying number of nodes - Energy consumption

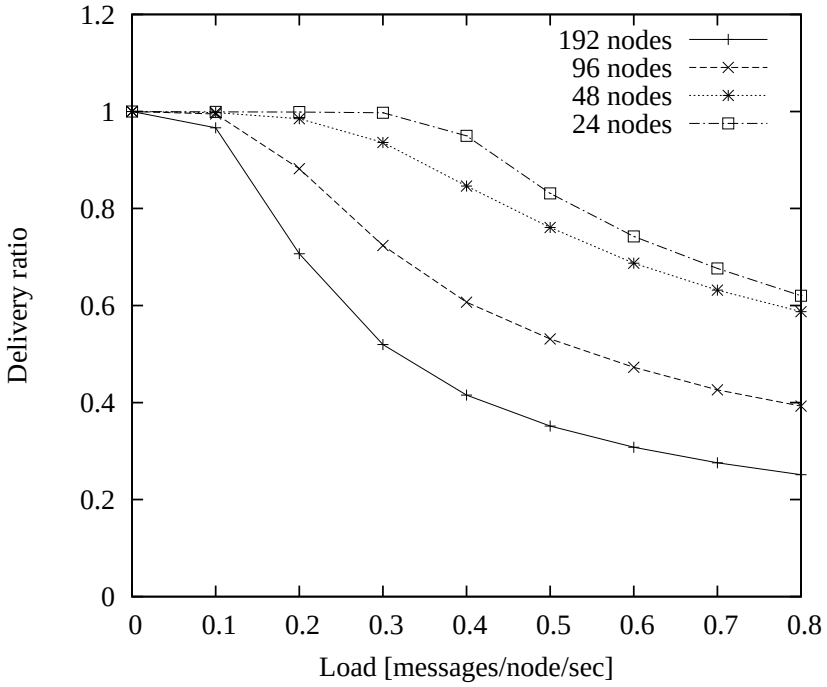

Fig. 6.35: Crankshaft: Varying number of nodes - Delivery ratio

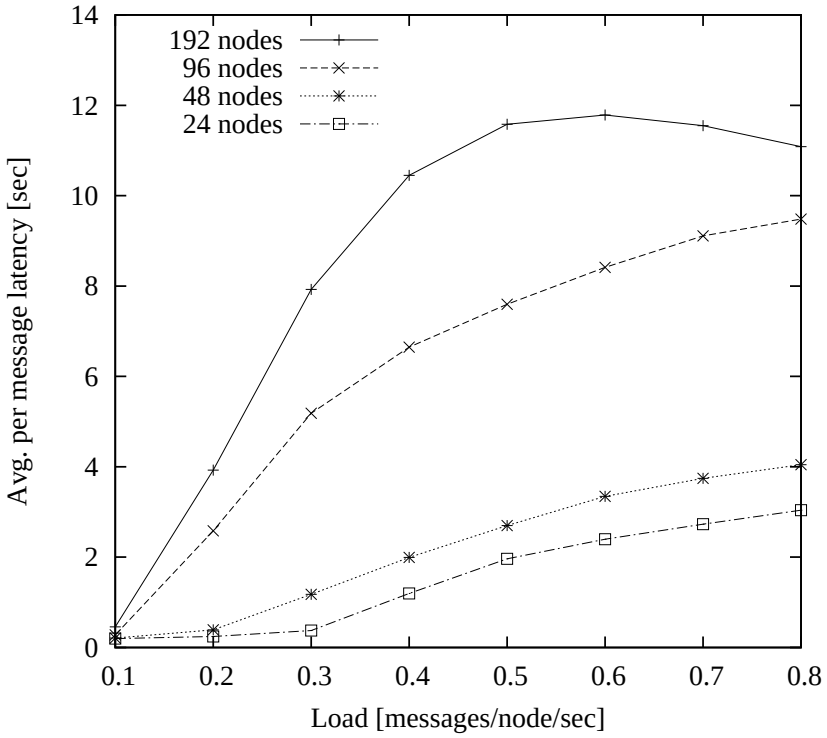

Fig. 6.36: Crankshaft: Varying number of nodes - Latency

The energy consumption (Figure 6.34) gets worse as the number of nodes increases. It is important to note that as the number of nodes increases and the contention window size stays the same, there are more contention window collisions, apart from more collisions in the air due to congestion in the network. The delivery ratio results (Figure 6.35) re-inforce the results from the earlier section that hop count impacts delivery ratio, but here, collisions at high loads also play a part. The dip in the latency for the network with 192 nodes is due to the very low delivery ratio, a smaller group of messages reaching the destination comparatively quicker.

We can make the case for three different directions based on these results: Careful placement of nodes, aggregating data or adjusting transmission power based on topology. We observe that the latency for Crankshaft is the best when the base station is located in the middle. For the default case of the base station at the sink, removing ACKs from the protocol improves latency the most.

6.3 Crankshaft vs. SCP-MAC*

The next set of simulations compares Crankshaft and SCP-MAC* both with and without ACKs, to obtain a thorough comparison of both protocols. Both protocols revert to their defaults (except for turning ACKs ON/OFF). This means simulating convergecast (multi-hop) traffic. For both protocols, when ACKs are enabled, the retry parameters are three retries with 70% retry probability. Figures 6.37 - 6.39 provide the consolidated performance results of both protocols with and without ACKs. This set of simulations were run with the original multi-hop traffic, and 96 nodes.

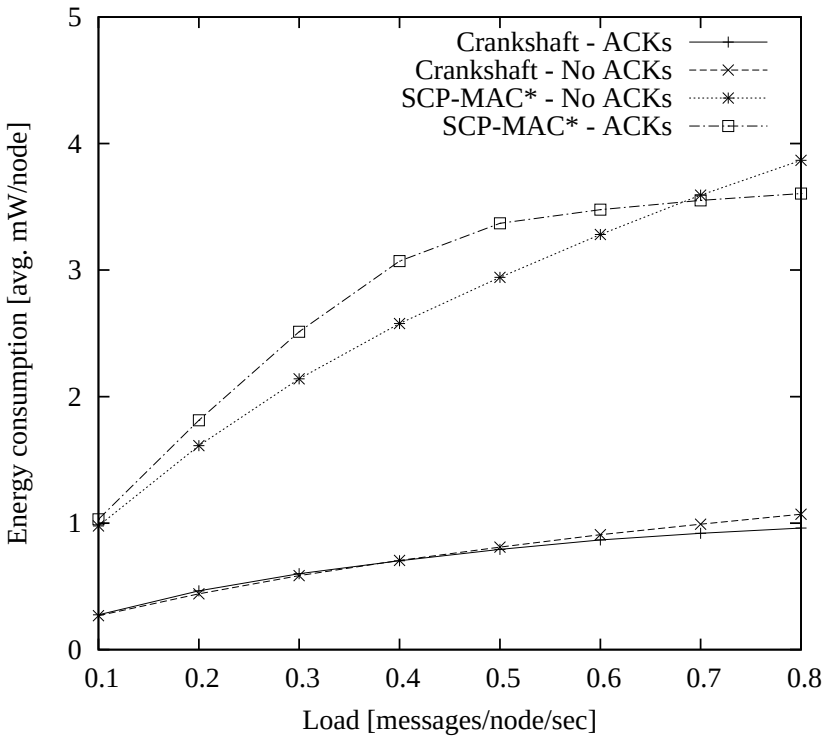

Fig. 6.37: Crankshaft and SCP-MAC* with and without ACKs - Energy consumption

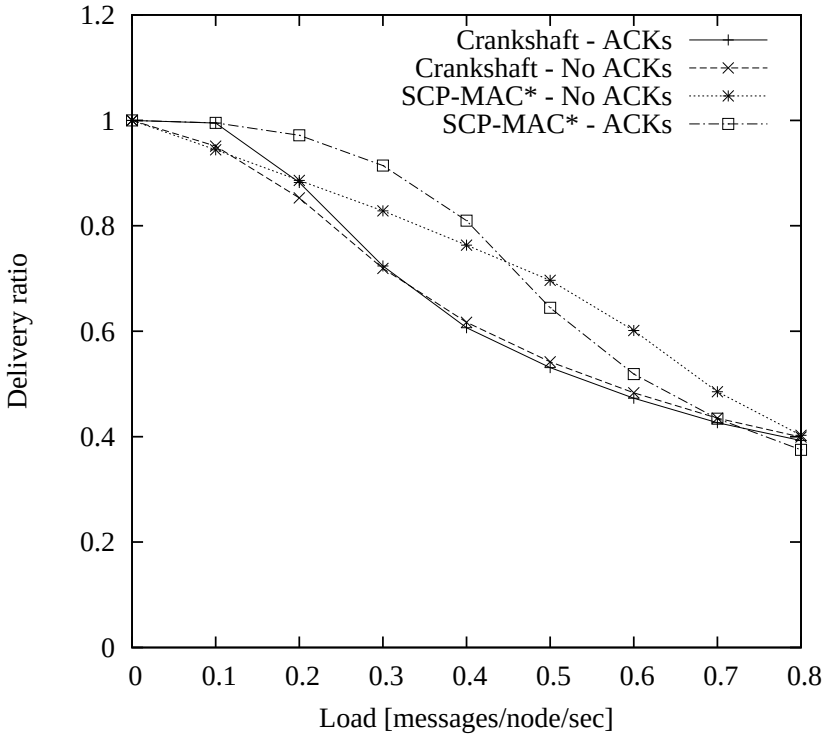

Fig. 6.38: Crankshaft and SCP-MAC* with and without ACKs - Delivery ratio

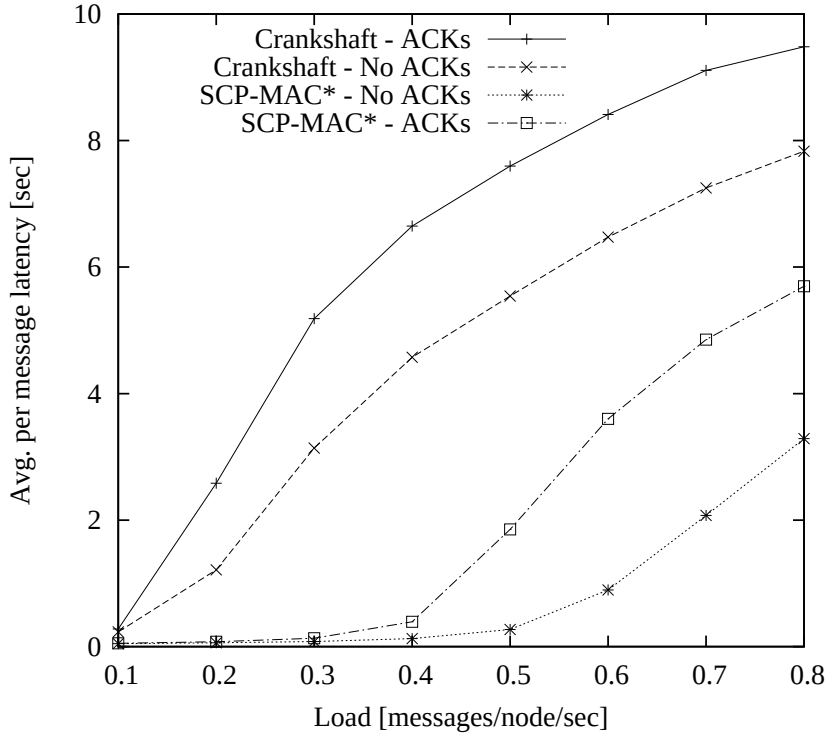

Fig. 6.39: Crankshaft and SCP-MAC* with and without ACKs - Latency

While there is not much of a performance difference for Crankshaft, it is interesting to note that SCP-MAC* is impacted highly by enabling ACKs. A fair comparison of both protocols would mean comparing both with ACKs enabled or with ACKs disabled. SCP-MAC* was originally designed without ACKs. So it is fair to compare it to Crankshaft with ACKs disabled, and we notice that we do achieve an improvement in the context of comparison with SCP-MAC*. Note that the cycle times for Crankshaft are much higher than SCP-MAC* (On average, a node that has to send has to wait a full 10 slots, compared to one slot in SCP-MAC*) so this is a considerable improvement in the latency of Crankshaft.

6.4 Crankshaft vs. SCP-MAC* - Other traffic patterns

This section compares the performances of Crankshaft and SCP-MAC* under various WSN traffic patterns.

6.4.1 One hop traffic

In the next set of simulations, the goal is to simplify the WSN environment to try to understand Crankshaft and SCP-MAC* by limiting the communication to single hop convergecast traffic. The motivation here is to understand the protocols better without the complications of multi-hop traffic. The radio range was set to the original 25m setting, but now we simulate only 24 nodes, all within one-hop distance of the base station.

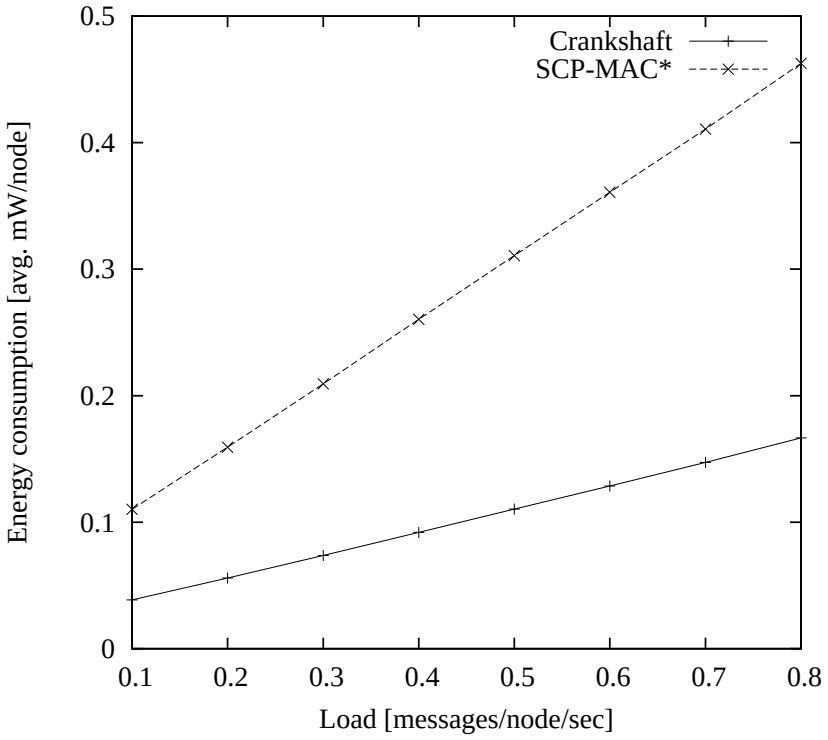

Fig. 6.40: Crankshaft vs. SCP-MAC*: Energy consumption under one hop traffic

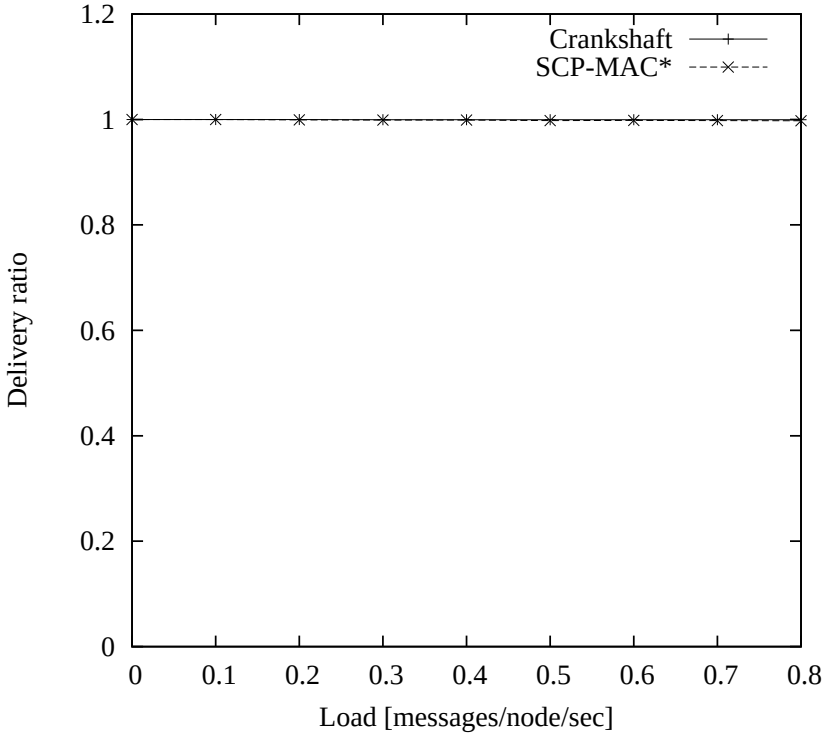

Fig. 6.41: Crankshaft vs. SCP-MAC*: Delivery ratio under one hop traffic

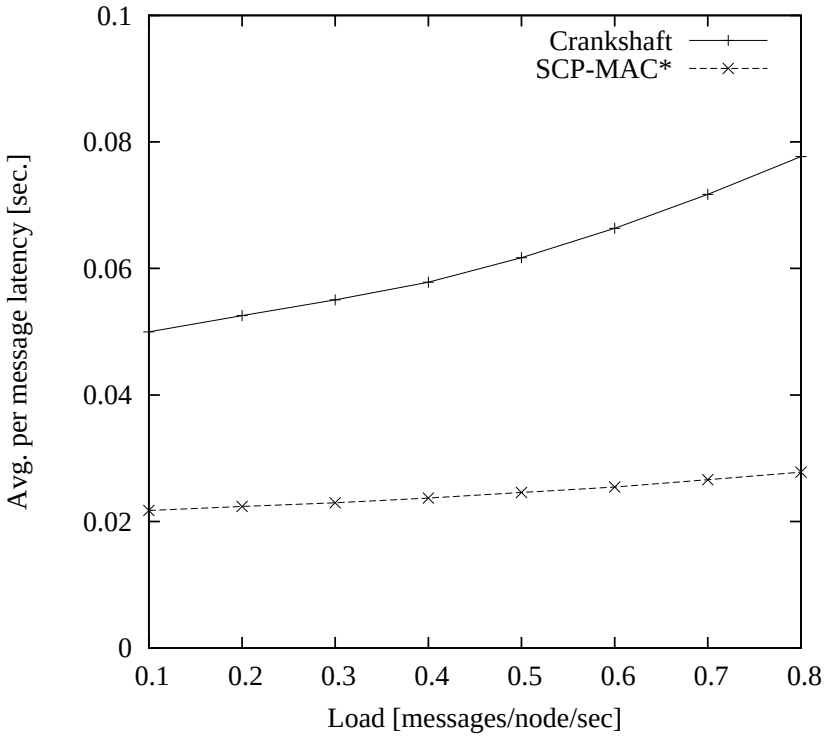

Fig. 6.42: Crankshaft vs. SCP-MAC*: Latency under one hop traffic

In figures 6.40 - 6.42, we notice similar differences to the multi-hop case in terms of both energy consumption and latency, but the delivery ratio is almost 100% for both protocols. Due to the contention window size (90) being much higher than the number of one-hop neighbors (24), there are very few collisions and very low contention, therefore all of the messages are sent successfully.

In the following figures (6.43 - 6.44), we compare delivery ratio and latency results against the corresponding multi-hop results with 24 nodes (for Crankshaft only) from Figure 6.31. Note that though the number of nodes is the same, the topology itself is different. The comparison re-iterates the fact that multi-hop causes collisions, and there are more packets to send, due to which a comparatively less number reaches the destination. This is due to a combination of farther paths and more packet collisions. This also impacts the latency (the latency starts to grow sharply at 0.4 messages/node/sec) for Crankshaft.

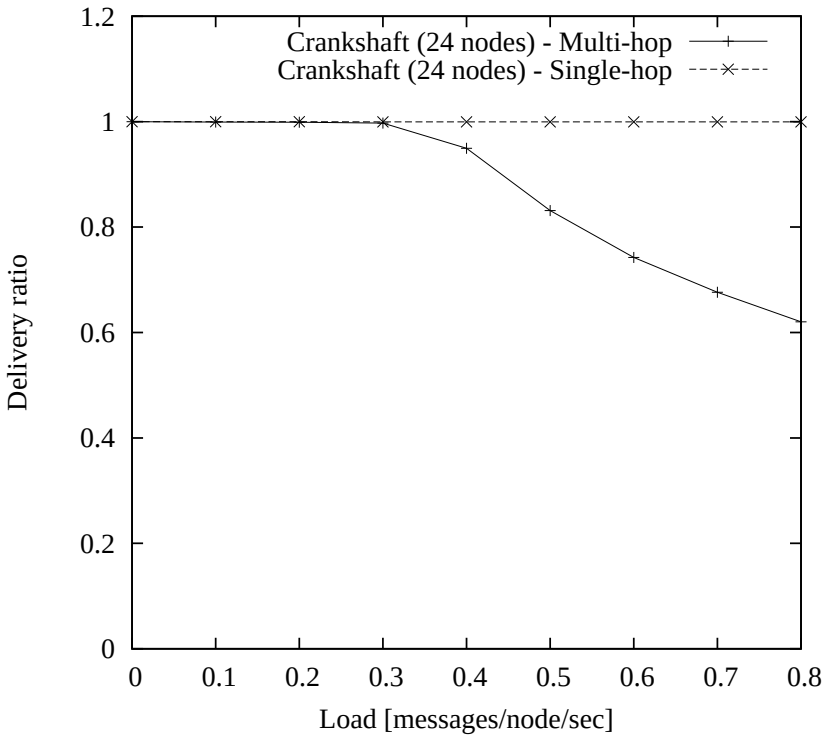

Fig. 6.43: Crankshaft Delivery ratio: 24 nodes (Multi-hop) vs. 24 nodes (Single-hop)

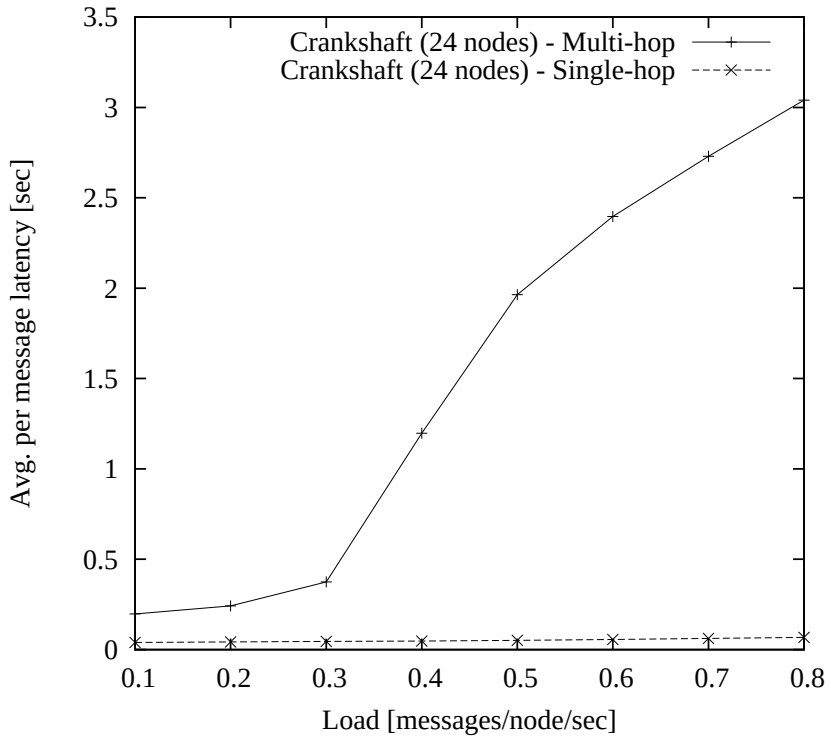

Fig. 6.44: Crankshaft Latency: 24 nodes (Multi-hop) vs. 24 nodes (Single-hop)

6.4.2 Local gossip pattern

To compare the performance of Crankshaft and SCP-MAC* under the local gossip traffic pattern, instead of the convergecast traffic all going to the sink, we use a local gossip traffic pattern. There are different classes of local gossip patterns, but here the *local unicast* model is used, where each node randomly chooses one receiver from its set of one-hop neighbors. This means that nodes without one-hop neighbors do not participate in communication. This also means that there can be more than one node sending to the same receiver, possible leading to collisions. In our simulations it is observed that for the 96 nodes, there are 48 send-receive pairs, but there are only 28 unique receivers. The results are presented in Figures 6.45 through 6.47.

Note that these values are averaged only for nodes participating in the communication.

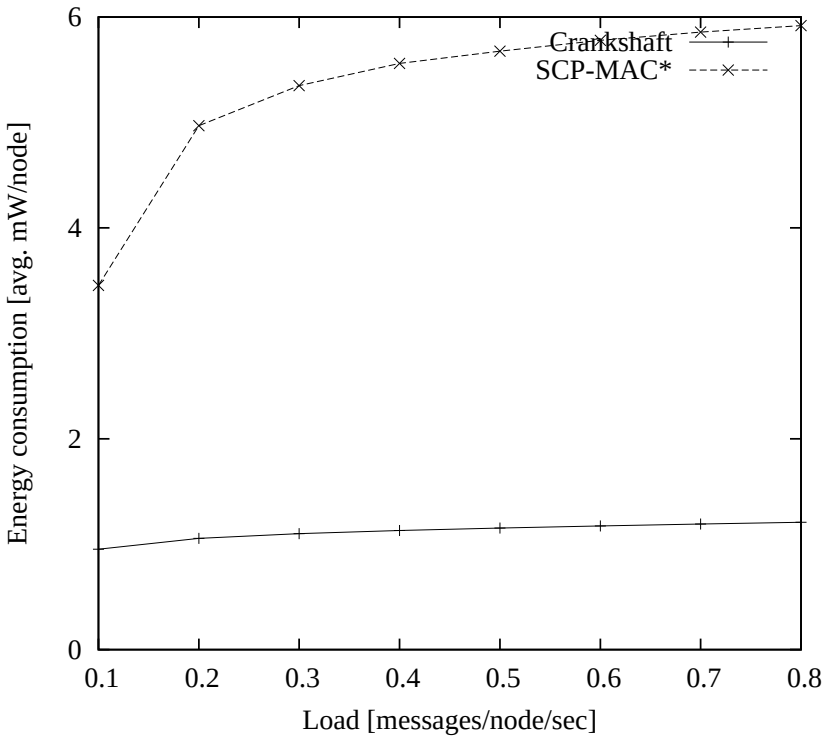

Fig. 6.45: Crankshaft vs. SCP-MAC*: Energy consumption under Local Gossip pattern

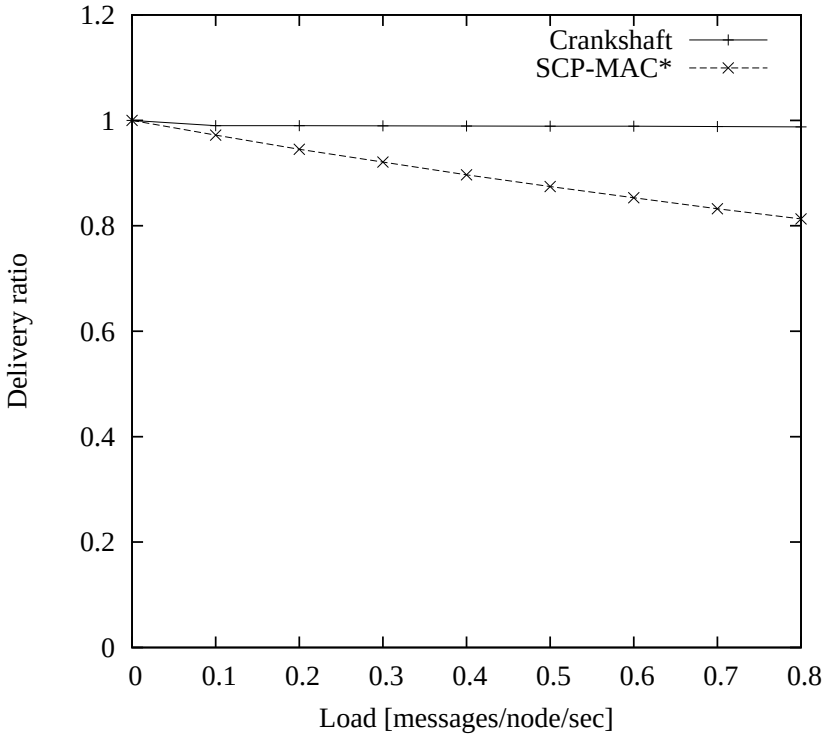

Fig. 6.46: Crankshaft vs. SCP-MAC*: Delivery ratio under Local Gossip pattern

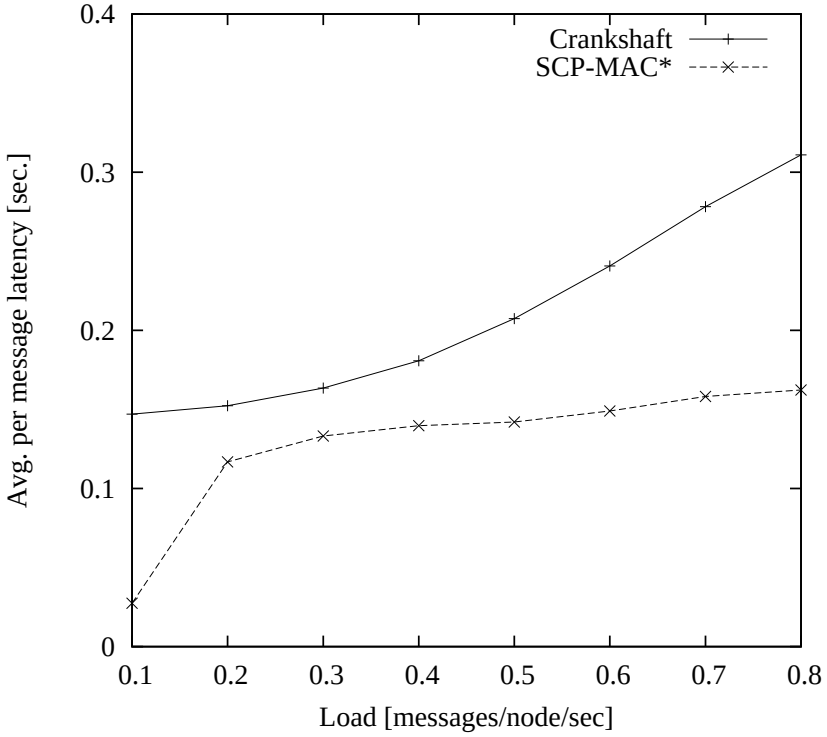

Fig. 6.47: Crankshaft vs. SCP-MAC*: Latency under Local Gossip pattern

The energy consumption (Figure 6.45) for Crankshaft is much better than SCP-MAC*. There is around a 20% gap in results for the delivery ratio (Figure 6.46) and this is due to ACK/No ACK trade-off. ACKs increase delivery ratio in one-hop traffic (Local gossip is one hop traffic). The increase in latency (Figure 6.47) for Crankshaft is due to ACKs increasing collisions due to congestion caused by retries. There is an anomaly at very low loads for the SCP-MAC* latency values, and this can be attributed to the collisions steadily accumulating after no collisions at the lowest loads.

6.4.3 Flooding traffic

Finally, the performances of Crankshaft and SCP-MAC* are compared under the flooding traffic pattern, instead of the standard sink relay (convergecast). As defined previously, in the flooding traffic pattern, the base station sends messages to its one-hop neighbors, and these are propagated in turn by the receivers to their one hop neighbors. Duplicates are filtered out. There are no ACKs in both cases here. We vary the number of slots as follows. The original case for Crankshaft is

8 unicast and 2 broadcast slots. All nodes use the broadcast slots for Crankshaft. We keep the number of unicast slots the same, but decrease the broadcast slots to 1. To get a fair comparison of the cycle times, we also increase the number of unicast slots to 9, keeping the number of broadcast slots to 1 (so that we have the original cycle of 10 slots). These variants of Crankshaft are compared with SCP-MAC*.

We observe that the energy consumption (Figure 6.48) for Crankshaft is low as nodes are awake for receiving only in the broadcast slots. The delivery ratio (Figure 6.49) is impacted negatively due to this. Crankshaft also has much worse latency (Figure 6.50) in the broadcast case as nodes have to wait a full cycle before sending or receiving. In SCP-MAC*, each broadcast message is sent per slot. This difference is also exaggerated due to the already vast difference in cycle times between Crankshaft and SCP-MAC*.

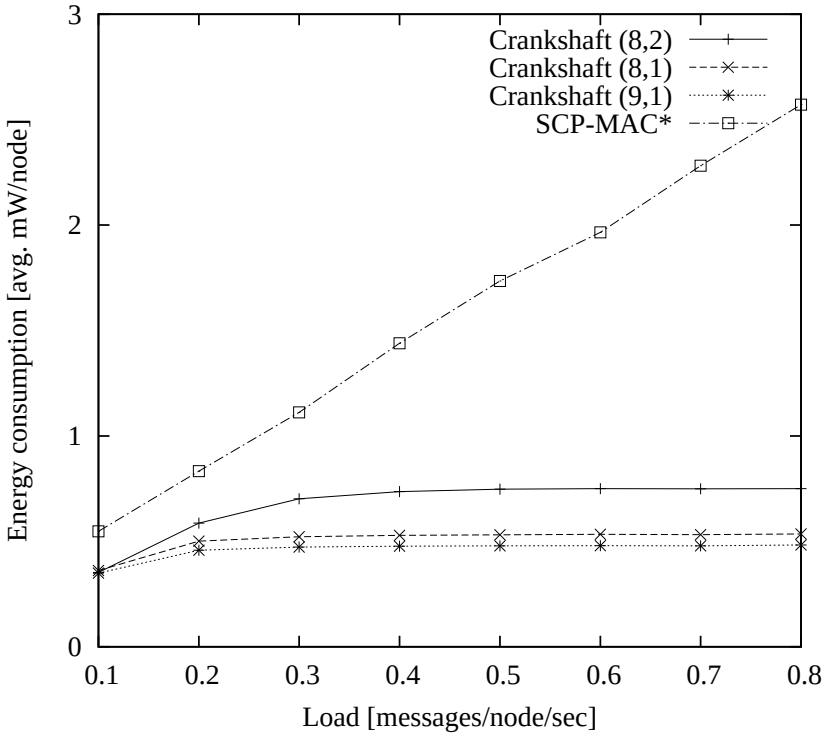

Fig. 6.48: Crankshaft vs. SCP-MAC*: Energy consumption under Flooding traffic

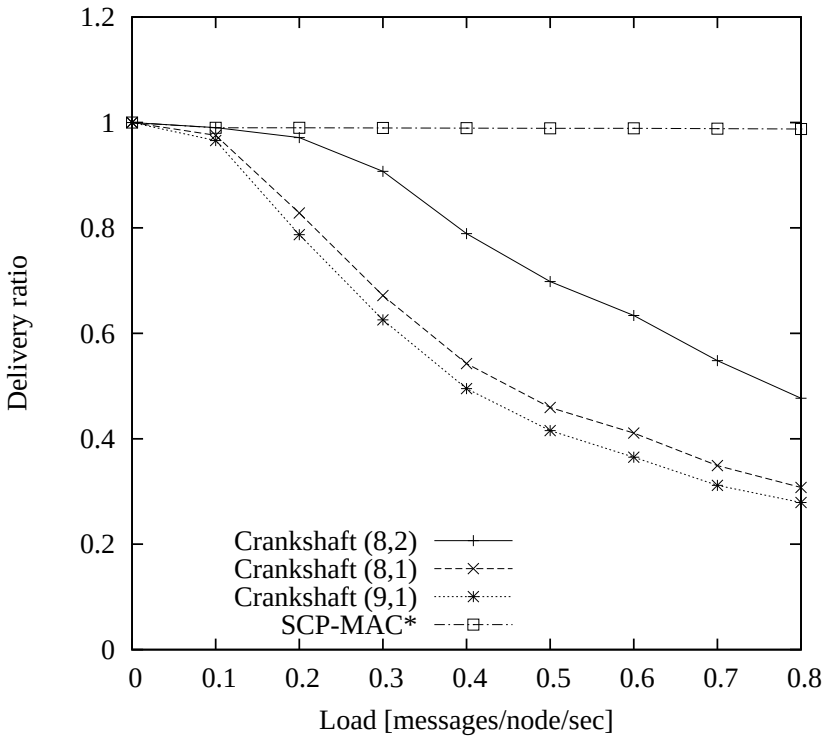

Fig. 6.49: Crankshaft vs. SCP-MAC*: Delivery ratio under Flooding traffic

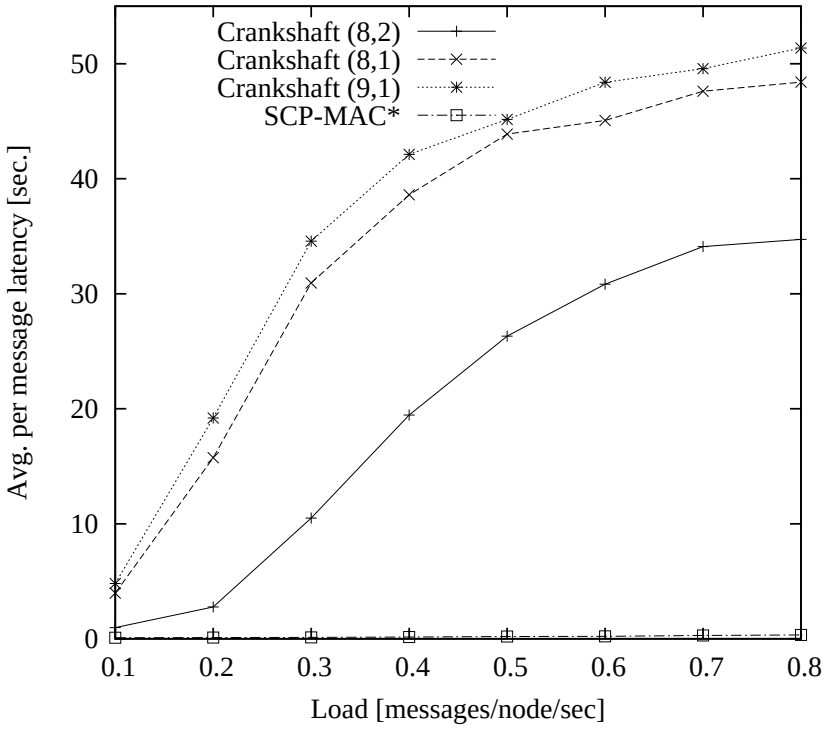

Fig. 6.50: Crankshaft vs. SCP-MAC*: Latency under Flooding traffic

Chapter 7

Conclusions

7.1 Observations

In this thesis, the domain of Wireless Sensor Networks was discussed in detail, by analyzing the characteristics, architectural decisions and the application space germane to the design of an energy-aware MAC protocol. To this end, a formal definition of an energy-aware MAC protocol was provided before discussing some of the prominent published protocols over the last decade. Specifically, this thesis investigated two of the recently published energy-aware MAC protocols, Crankshaft and SCP-MAC. The design and performance of both protocols were analyzed with various settings of the protocol parameters, and under various network conditions. The goals of this thesis introduced in **Section 1.2** are re-iterated:

- Investigate the design of Crankshaft to improve its latency. Provide the best conditions for Crankshaft that bring the latency down without adversely impacting its energy consumption.
- Provide a thorough analysis of Crankshaft with different settings of its various parameters. Since the Crankshaft protocol provides a lot of novel ideas for WSN MAC protocol design, this analysis will be helpful for future research.
- Provide a thorough comparison of Crankshaft and SCP-MAC under different traffic patterns, so as to understand the effect of the network topology and traffic on MAC protocol design.

To aid this study, simulations were performed using OMNeT++, an open source and component-based simulation framework. The framework provided a readily available variant of SCP-MAC, called SCP-MAC*. SCP-MAC* does not use two-phase contention, but uses Sift for contention resolution, and ACKs are disabled. Simulations were performed with the default settings used by the Crankshaft authors in their simulations, in a network with 96 nodes (including the base station). This default setting imposes ACKs (with three retries) for Crankshaft and no ACKs for SCP-MAC*. The default traffic pattern used was multi-hop convergecast. All results were based on calculations for three important network metrics: Energy consumption (averaged over all nodes), delivery ratio and latency at the sink (calculated at the packet layer).

In **Section 6.1**, the results from the original Crankshaft paper were reproduced, with a deviation in the SCP-MAC* results for loads above 0.5 messages/node/sec (usually unseen [25]). In **Section 6.2.1**, the effect of Sift was investigated by replacing it with a uniform contention resolution method. Sift was proven to be a better choice, with considerable improvement in terms of energy consumption (10% at 0.4 messages/node/sec). In Section **6.2.2**, Crankshaft was simulated without ACKs. The latency consequently improved considerably (28% at 0.5 messages/node/sec). This is due to the fact that there were no retries due to no ACKs, leading to a lower traffic bottleneck near the sink. In **Section 6.2.3**, Crankshaft was simulated with ACKs, but the number of retries was reduced. This improves Crankshaft latency due to the fact that there is less overall network congestion, especially at the sink. In **Section 6.2.4**, the number of broadcast slots in Crankshaft was reduced, and the latency improved as the number decreased. This is due to a reduction in the overall cycle time for Crankshaft. In **Section 6.2.5**, Crankshaft was simulated by replacing its Sift mechanism with the two-phase contention resolution of SCP-MAC, without any considerable improvement.

The important conclusion from this section of the results is that Crankshaft latency can be reduced by using no ACKs, or reducing the number of retries. This ties in to the hypothesis that a multi-hop network leads to congestion. When congestion occurs, reducing traffic (even if it is ACKs) by reducing retries can improve the performance. Thus, the hypothesis is that Crankshaft latency depends on network parameters such as number of hops to the sink and the sink location.

This hypothesis was tested in **Section 6.2.6**. The first set of simulations involved moving the sink. The best case was when the sink is in the middle, because the overall number of multi-hop packets sent due to the sink is reduced. Though the contention due to the base station increases,

this does not have too much of an adverse effect on energy consumption, but in fact improves the latency by almost 66%. The delivery ratio results here showed that multi-hop packets have a lower probability of reaching the destination. The worst case here was when the sink was at the sparser side of the network. Path fading plays a role for nodes that are far away, apart from the increased number of hops, which leads to congestion and retries. The second set of simulations involved changing the node density of the network. The results here show that the energy consumption gets worse as the number of nodes increases. The delivery ratio results re-inforce the notion that hop count impacts delivery ratio, because a higher number of nodes corresponded to a higher average number of hops to the sink. Consequently, the delivery ratio gets worse as this number increases. Latency was also better for a network with a smaller number of nodes.

It was also observed that studying WSN-MAC protocols in a multi-hop network with convergecast traffic is non-trivial as there is contention at the senders and collisions at hops close to the base station. These factors complicate the analysis of energy consumption at different states that a node radio can be in. Moreover, the designs for both Crankshaft and SCP-MAC* are extremely complicated. The SCP-MAC authors propose using an adaptive channel polling mechanism in multi-hop traffic, which was not implemented, further impacting the results. Based on this analysis, it was decided to simulate both protocols in one-hop traffic, apart from other traffic patterns.

In Section **6.4.1**, one-hop traffic was simulated by reducing the number of nodes and re-arranging the topology so that all nodes are under the radio range of the base station. Both protocols had perfect delivery ratios in this scenario, which was expected. The delivery ratio and latency results were also compared against corresponding multi-hop results. The results re-iterated the fact that multi-hop traffic causes collisions. This is observed to be one of the main reasons for the poor Crankshaft latency in multi-hop traffic: The latency starts to grow sharply at 0.4 messages/node/sec.

In Section **6.4.2**, both protocols were compared using local gossip traffic, where there are send-receive pairs over the network randomly chosen from one-hop neighbors. This also involves one-hop traffic, but with more than one receiver (i.e. other than the sink) The energy consumption for Crankshaft is still better, and the delivery ratio is impacted due to using ACKs. This gives an important result that ACKs directly improve delivery ratio in one-hop traffic.

In Section **6.4.3**, flooding traffic was simulated. The energy consumption for Crankshaft is low as nodes are awake for receiving only in the broadcast slots. The delivery ratio and latency

are also impacted negatively due to the fact that nodes have to wait a full cycle before sending or receiving in Crankshaft. This difference also brings to light the vast difference in cycle times between Crankshaft and SCP-MAC*.

The goals of this thesis were successfully met, and the study shows that Crankshaft has better latency in the following scenarios, and the WSN application under consideration defines which scenario is relevant.

- In multi-hop traffic, do not use ACKs. If ACKs are necessary, reduce the number of retries.
- Place the sink in the middle of the network topology, i.e., increase its one hop neighbors in a convergecast traffic pattern. This also ties in with careful placement of nodes and adjusting transmission power based on topology.
- Use one-hop traffic if possible so as to improve delivery ratio and latency.
- Reduce the number of broadcast slots if the application does not involve (or rarely involves) broadcast/flooding traffic. Conversely, if broadcast is important, increase the number of broadcast slots.

7.2 Future Work

The following areas of future work have been identified.

- Improve the way that the number of slots is defined in Crankshaft. Ideally, the number of slots should be adaptive, based on the traffic and the node density.
- More research is also required in determining the ideal number of slots for Crankshaft. The trade-off between slot-provisioning to reduce slot collisions and the cycle times for the protocol should be analyzed.
- Improve receive slot scheduling for Crankshaft. This is currently done by using the MAC address of a sensor node, but this can lead to multiple receivers in the same receive slot. This is not ideal in multi-hop traffic. AS-MAC [26], also published in 2007, addresses this fact, by using an algorithm to assign receive slots as the nodes join the network. This is done by

asynchronously co-ordinating the wake-up times to overhearing, contention and delay. In fact, the design of AS-MAC borrows from both Crankshaft and SCP-MAC, and can be considered novel for future research in this area.

- Per-hop latencies for Crankshaft should be analyzed. This is complicated due to multi-hop traffic, but will be useful to determine the areas of high latency with respect to the sink, so that they can be addressed.

- Another area for future research is identifying and investigating sender vs. receiver energies separately. This will especially be useful in a more thorough comparison between protocols such as Crankshaft and SCP-MAC that schedule senders and receivers differently.

Bibliography

[1] K.G. Langendoen, *Medium Access Control in Wireless Sensor Networks* in H. Wu, Y. Pan **Medium Access Control in Wireless Networks, Volume II: Practice and Standards** (pp. 535 - 560), Nova Science Publishers, January 2008.

[2] *Wireless Sensor Networks* @ Wikipedia, **http://en.wikipedia.org/wiki/Wireless sensor network**

[3] Koen Langendoen, *The MAC Alphabet Soup, served in Wireless Sensor Networks*: **https://apstwo.st.ewi.tudelft.nl/ koen/MACsoup/index.php**

[4] W. Ye, F. Silva, J. Heidemann, *Ultra-low Duty Cycle MAC with Scheduled Channel Polling* in Proceedings of the Fourth **ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)** (pp. 321 - 333), Boulder, CO, USA, November 2006.

[5] G.P. Halkes, K.G. Langendoen, *Crankshaft: An Energy-Efficient Mac-Protocol For Dense Wireless Sensor Networks* in Proceedings of the Fourth **European Conference on Wireless Sensor Networks (EWSN 2007)** (pp. 228 - 244), Delft, The Netherlands, January 2007.

[6] R. Bose, *Sensor Networks Motes, Smart Spaces, and Beyond* in **IEEE CS - Pervasive Computing** (pp. 84 - 90), September 2009.

[7] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, W. Pratt, *WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control* in Processing of the **IEEE Real-Time and Embedded Technology and Applications Symposium**, (pp. 377 - 386), Los Alamitos, CA, USA, April 2008.

[8] ISA standard 100.11a, Standards document, **Wireless systems for industrial automation: Process control and related applications**, 2009.

[9] M. Welsh, *Harvard, CS263 - Wireless Sensor Networks; Online lecture notes*, **http://www.eecs.harvard.edu/ mdw/course/cs263/intro.pdf**

[10] A. Garca-Hernando, J. Martinez-Ortega, J. Lpez-Navarro, A. Prayati, L. Redondo-Lpez, *WSN Application Scenarios*, Chapter 8 in **Problem Solving for Wireless Sensor Networks** (pp. 177 - 209), Springer London, December 2008.

[11] B. Raman, K. Chebrolu, *A Critique of Sensor Networks from a Systems Perspective* in **ACM SIGCOMM Computer Communication Review** (pp. 75 - 78), July 2008.

[12] A. Varga, *OMNeT++ Discrete Event Simulation System User Manual*, 2006, **http://www.omnetpp.org/doc/manual/usman.html**.

[13] U. M. Colesanti, C. Crociani, A. Vitaletti, *On the Accuracy of OMNeT++ in the Wireless Sensor Networks Domain: Simulation vs. Testbed* in Proceedings of **Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks** (pp. 25 - 31), Chania, Crete Island, Greece, October 2007.

[14] A. Varga, *The NED Language*, 2006, **http://www.ewh.ieee.org/soc/es/Nov1999/18/ned.htm**

[15] *MiXiM*, Official Webpage, **http://sourceforge.net/apps/trac/mixim**

[16] A. Kpke, M. Swigulski, K. Wessel, D. Willkomm, P.T. Klein Haneveld, T.E.V. Parker, O.W. Visser, H.S. Lichte, S. Valentin, *Simulating Wireless and Mobile Networks in OMNeT++: The MiXiM Vision* in Proceedings of the **OMNeT++ 4.0 Workshop, Marseille** (pp. 71 - 78), France, March 2008.

[17] J. Hill, D. Culler, *Mica: a wireless platform for deeply embedded networks*, in **IEEE Micro**, (pp. 12 - 24), 2002.

[18] A. El-Hoiydi, J. D. Decotignie, *WiseMAC: An Ultra Low Power MAC Protocol for the Downlink of Infrastructure Wireless Sensor Networks* in Proceedings of the Ninth **IEEE Symposium on Computers and Communication** (pp. 244 - 251), Alexandria, Egypt, June 2004.

[19] W. Ye, J. Heidemann, D. Estrin, *An Energy-Efficient MAC Protocol for Wireless Sensor Networks* in Proceedings of the 21st **International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)** (pp. 1567 - 1576), New York, NY, USA, June 2002.

[20] T. Van Dam, K. Langendoen, *An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks* in Proceedings of the First **ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)** (pp. 171 - 180), Los Angeles, CA, USA, November 2003.

[21] L. van Hoesel, P. Havinga, *A lightweight medium access protocol (LMAC) for Wireless Sensor Networks*, in Proceedings of the the First **International Workshop on Networked Sensing Systems (INSS 2004)** (pp. 205 - 208), Tokyo, Japan, June 2004.

[22] K. Jamieson, H. Balakrishnan, Y. Tay, *Sift: A MAC Protocol for Event-driven Wireless Sensor Networks* in Proceedings of the Third **European Workshop on Wireless Sensor Networks (EWSN 2006)** (pp. 260 - 275), Zurich, Switzerland, February 2006.

[23] A. Meier, K. Langendoen, *Analyzing MAC Protocols for Low Data-Rate Applications*. Presentation at ETHZ, Zurich, Switzerland, May 2007.

[24] Y. Li, W. Ye, J. Heidemann, *Energy and latency control in low duty cycle MAC protocols* in Proceedings of the **IEEE Wireless Communications and Networking Conference** (pp. 676 - 682), New Orleans, LA, USA, March 2005.

[25] J. Suh, M. Horton, *Current Hardware and Software Technology for Sensor Networks* in Proceedings of the First **Workshop on Networked Sensing Systems (INNS)**, San Jose, CA, USA, June 2004.

[26] B. Jang, J. B. Lim, M. L. Sichitiu, *AS-MAC: An Asynchronous Scheduled MAC Protocol for Wireless Sensor Networks* in Proceedings of the Sixth **International Conference on Networking (ICN 2007)** (pp. 434 - 441), Sainte-Luce, Martinique, France, April 2007.

Appendix 1

generic.ini:

```
ini-warnings=1
network="net"
sim-time-limit=200s
num-rngs=20 // No of unique random number generators for various runs of a
simulation
event-banners=0
net.nodeCount=96
net.nodeType="TNOde"
net.radioRange=25
net.radioType="RFM1001"
net.applicationName="AppSelector"
net.ultrasoundType="SimpleUS"
net.radioPropagationType="PathLossRadioPropagationModel"
net.usPropagationType="SimpleUSPropagationModel"
net.lpl_on=0
net.lpl_off=0
net.siftNodes=512
net.nodes[0].node.node_id=0
net.nodes[0].node.pos_x=-6.0
net.nodes[0].node.pos_y=1.0
...
```

Appendix 2

GMacF2.ini:

```
runs-to-execute=1
mixim.GMacF2.bcastSlots=2 mixim.GMacF2.slottedBcast=false
net.macType="GMacF2"
net.lpl=false
net.routingType="MagicSinkRouting"
mixim.ScenarioManager.inputfile=scenario1
```

Appendix 3

Scenario files:

The scenario files are of the following format: `active/idle class msglength msg/s [p=x, p2=x2, ... ]`

- active/idle: whether this is an "idle" or "active" pattern. An idle pattern is one that is simply turned on for the complete simulation. Active patterns are patterns that occur at random times.
- class: The name of the C++ class that will be created to generate messages and process incoming messages
- msglength: The size of the messages that get sent
- msg/s: Rate of messages to be sent out in messages/second
- p=x, p2=x2: Various module specific parameters

  replyprob: a probability value (0.0 - 1.0) determining how likely it is that a particular node will reply to a message

  ttl: max time to live i.e. maximum number of source-sink hops

  aggmode: aggregation mode. Data can be aggregated in this pattern

  0: no aggregation

  1: concatenate messages together into one big message

  2: average the data values from different messages into one message

Appendix 4

Results of feeding the OMNeT++ simulator through the perl scripts:

```
Average Outputs (Energy):
idle e =
collision e =
rx_data e =
rx_overhead e =
rx_overhear e =
tx_data e =
tx_overhead e =
sleep e =
Average Outputs (No.  of messages):
app_tx =
app_rx =
rt_tx =
rt_rx =
mac_tx =
mac_rx =
Average Outputs (Time):
delay_network =
delay_sink =
radio_sleep =
radio_tx =
radio_rx =
radio_collision =
mac_rx_data =
mac_rx_overhead =
mac_rx_overhear =
mac_tx_data =
mac_tx_overhead =
```