ABSTRACT

Physical Implementation of Synchronous
Duty-Cycling MAC Protocols:
Experiences and Evaluation

by

Wei-Cheng Xiao

Energy consumption and network latency are important issues in wireless sensor networks. The mechanism duty cycling is generally used in wireless sensor networks for avoiding energy consumption due to idle listening. Duty cycling, however, also introduces additional latency in communication among sensors. Some protocols have been proposed to work in wireless sensor networks with duty cycling, such as S-MAC and DW-MAC. Those protocols also tried to make efficient channel utilization and to mitigate the chance of packet collision and the network latency increase resulting from collision. DW-MAC was also designed to tolerate bursty and high traffic loads without increasing energy consumption, by spreading packet transmission and node wakeup times during a cycle.

Some performance comparison between S-MAC and DW-MAC has been done in previous work; however, this comparison was performed in the *ns-2* simulator only. In the real world, there are further issues not considered or discussed in the simulation, and some of those issues contribute significant influences to the MAC protocol performance. In this work, I implemented both S-MAC and DW-MAC physically on MICAz sensor motes and compared their performance through experiments. Through

my implementation, experiments, and performance evaluation, hardware properties and issues that were not addressed in the previous work are presented, and their impacts on the performance are shown and discussed. I also simulated S-MAC and DW-MAC on *ns-2* to give a mutual validation of the experimental results and my interpretation of the results. The experiences of physical implementations presented in this work can contribute new information and insights for helping in future MAC protocol design and implementation in wireless sensor networks.

Acknowledgments

First, I would like to show my sincere gratitude to my advisor, David Johnson. His guidance and support plays an extremely important role during my study in Rice University. I really benefited a lot from the long hours of discussions and invaluable suggestions about the work in this thesis. Without his help, it would be impossible for me to accomplish this thesis. I also want to thank the other two committee members, Edward Knightly and T. S. Eugene Ng, for their insightful comments and suggestions to my work.

I want to thank Yanjun Sun for his sharing of the experiences of programming in TinyOS. I also want to show my thankfulness to Chaoran Yang, Lei Tang, and Keyvan Amiri, for their help during my preparation of thesis defense. I also appreciate the encouragement from all my friends and colleagues at Rice University.

I would like to thank my wife, Xiaorui Chen, for her unconditional support and company during my hard times. I also want to thank Yimin Ge, Jia-Sheng Peng, Po-Hsiang Chang, and other friends for their comfort, encouragement, and prayers. Last but not the least, I sincerely thank God for His unchanging love and grace and give praise to Him.

Contents

Illustrations

Tables

Chapter 1

Introduction

1.1 Wireless Sensor Networks

Wireless sensor networks bring environment monitoring into a new field. The significance of the role of wireless sensor networks is increasing. Wireless sensor networks can be deployed in areas where it is difficult for human beings to reach, such as in the ocean, disaster areas, or small places, for environment monitoring or data collection. A wireless sensor network may consist of numerous nodes distributed in the environment. Message delivery in the network is realized via wireless multi-hop communication among the nodes. In addition to the processor, memory components, and radio, each sensor node is equipped with one or more sensors for environment sensing.

1.2 Duty Cycling

Due to the size, cost, and in many cases remote or inaccessible locations, the power supply of sensor nodes is usually limited with most sensor nodes being powered by batteries. Since it may be difficult to replace the batteries, achieving long battery life is important. Among the hardware components on a node, the radio consumes a significant portion of energy, and idle listening is the largest source of energy wasting. Idle listening is a state that the radio is listening to the channel but not transmitting or receiving any packets. Thus, duty cycling has been introduced into wireless sensor networks to help the nodes save energy. With duty cycling, a node turns its radio

on and off periodically. When the radio is on, the node can send or receive packets with other node, whereas the node would sleep to save energy when the radio is off. A typical value of duty cycle, i.e., the percentage of time the radio of a node is on, ranges from 1 to 10%.

Duty cycling in wireless sensor networks can be classified into two categories: *synchronous* and *asynchronous*. In asynchronous duty-cycling networks, nodes may wake up and go to sleep at different times with different schedules. Examples of asynchronous duty-cycling MAC protocols include RI-MAC [17], WiseMAC [10], PW-MAC [19], and EM-MAC [18]. A node may turn off its radio after finishing a packet transmission and does not have to follow other nodes' schedules, i.e., go to sleep with others at the same time. Also, whereas nodes running a synchronous duty-cycling MAC protocol must wake up at the beginning of the Data period and typically remain awake for all or most of the Data period, nodes running an asynchronous duty-cycling protocol are often able to go to sleep right after transmitting a packet. However, it is also possible that a receiving node wakes up earlier than the intended sender and consumes energy when waiting. In order to minimize this kind of energy wasting, an asynchronous duty cycling MAC protocol needs to develop energy-efficient node scheduling methods.

Unlike the asynchronous design, in synchronous duty-cycling MAC protocols, all the nodes start cycles at the same time with the same duty cycle setting. Examples of synchronous duty-cycling MAC protocols include S-MAC [22], DW-MAC [16], RMAC [9], and T-MAC [20]. The advantage of synchronous duty cycling is that a node can easily meet other nodes' schedules and communicate with them, which simplifies the design of the MAC protocol as all nodes are awake at the same time. The synchronous design also avoids a node from idle listening when waiting for an-

other node to wake up, helping to increase energy efficiency. On the other hand, timing synchronization across the network creates a design challenge. Some previous work [16, 9] is designed upon the assumption that time is already synchronized, while some other work [14, 13] focuses particularly on fine-grained time synchronization instead of building a complete MAC protocol. This thesis focuses on physical implementation of synchronous MAC protocols.

1.3 Other Energy-Related Issues

Another source of energy wasting comes from packet collisions. Collision of a packet not only results in time and energy wasting on failing to deliver that packet, it also causes the MAC layer to retransmit the packet, which increases the network latency and energy consumption too. In duty-cycling networks, failing to get a packet delivered usually means the node has to wait until the next cycle to make another attempt. S-MAC uses collision avoidance mechanisms similar to the RTS/CTS design in the IEEE 802.11 MAC protocol [11]. This design greatly helps prevent collisions on data packets and acknowledgements, but it also brings another problem. To avoid collision, a node that overhears a control packet, i.e., RTS or CTS, from other nodes, will defer its own control packet transmission to the next cycle. If there is high contention for the network, a node may overhear others' control packets quite often, further increasing the latency. In contrast, DW-MAC chooses a different mechanism for channel usage that results in a shorter network latency. A scheduling packet, SCH, replaces the roles of RTS and CTS in S-MAC. An SCH is used for reserving a time period in the cycle, and theoretically, it is guaranteed that once a time period is reserved, no collision for the data packet will happen. In multi-hop forwarding, an SCH can play both the roles of a time reservation control packet and a confirmation reply.

Compared with S-MAC, which needs two control packets, RTS and CTS, for channel reservation in each hop, DW-MAC saves nearly half of the control packet overhead. Through this scheduling mechanism, DW-MAC has more efficient channel usage and lower latencies and energy consumption than does S-MAC.

1.4 Timing

Accurate timing is important in duty-cycling wireless sensor networks, especially for synchronous MAC protocols. Many synchronous duty-cycling MAC protocols are designed under the assumption of accurate timing. Timing inaccuracy may cause a node to wake up too late or go to sleep too early and miss a packet targeted to it. This has similar effects to collision. Compared with S-MAC, timing is even more critical in DW-MAC. DW-MAC uses a technique called *proportional mapping*, described in Section 2.2.1, to reserve time periods for data transmission. The design of proportional mapping guarantees that no collision would occur on data packets and their acknowledgments. If there is a time difference between one node and others, i.e., the node does not start a cycle at the same time as others, the node may reserve a time period that is interleaved with others' periods, and then collision on data packets could occur. Even a small time difference can result in great confusion about the reserved time. Details about this issue will be described in later sections.

1.5 Motivation and Contribution

In the previous work published by Sun *et al.* [16], DW-MAC was proposed, and performance comparison between DW-MAC and the other two MAC protocols, S-MAC and RMAC, was also conducted. Their performance evaluation, however, was only done in the *ns-2* simulator. In this thesis, I implement DW-MAC on the Crossbow

MICAz motes [2] using TinyOS [4], which is an open-source operating system for sensor motes, and port the official S-MAC codes [3] to the same platform. I choose to implement DW-MAC because of its timing critical nature. During the implementation, many timing-related problems are revealed, and I believe the experiences of discovering and solving those problems are important for future protocol implementation and design. I also conduct both real-world experiments and simulation in the work to provide analysis of the behaviors of both MAC protocols and giving mutual demonstration of the correctness of performance evaluation results. This thesis also presents some realistic and important issues that are not addressed in the previous work but exist on physical sensor networks. The following paragraphs briefly summarize those issues and my solutions and implementation to them.

First, a perfect clock is assumed in the simulation; that is, the clock always ticks at the same rate. On a real sensor node, however, there is *clock drift*, which means the clock rate could change over time. As the CPU of a sensor node is usually cheaper and simpler than that of modern desktop or laptop computers, clock drift is more a problem. In addition to the clock drift on one node, the clock rates between two nodes could also be slightly different. Both the clock drift and clock rate difference could lead to *timing inaccuracy* among nodes, which may cause significant impact to the operation and correctness of MAC protocols. Therefore, a time synchronization mechanism is required for periodically rectifying timing of each node in the network and minimizing the impact of timing inaccuracy. While time is assumed to be synchronized across the whole network in the simulation, we need to figure out a way to handle the timing well when building a physical sensor network in the read world. In this work, a simple time synchronization mechanism is implemented. Through this implementation, time is synchronized across the whole network every other cycle so

that time differences among nodes are tolerable. The impacts of timing inaccuracy of DW-MAC are also analyzed mathematically, because DW-MAC is more sensitive to timing correctness. To meet the timing requirements and correctness of operation in DW-MAC, I add the concept *guard time*, which is a small period of time whose length is computed from the analysis. Combining the guard time with periodic time synchronization, no collision could occur on data packets and acknowledgements in DW-MAC. Details of the problems and solutions will be described in later chapters.

The second issue relates to the packet timestamping. In many cases, MAC protocols need to know the time at which a packet is sent or received. That timing information can be obtained from the timestamp in a packet. Whereas it is relatively easy for a simulator to timestamp a packet, a physical sensor node needs to detect a *start frame delimiter* (SFD) when a packet comes to its network interface and write a timestamp to that packet. This timestamp writing must be done very quickly since the packet transmission is in progress and should not be interrupted. When a packet arrives, if the node is so busy that there is not enough time to timestamp the packet, timestamping could fail. To deal with the timestamp problem, in this work, the statistics of packet processing time in the network physical layer is computed and kept every time a successful timestamping occurs. The physical-layer packet processing time can be obtained from the difference between the time the MAC layer sees a packet and the timestamp on the packet. The statistics can be used to estimate the processing time of packets whose timestamping failed and to infer the time packets are sent or received.

Third, most CPU overheads, such as instruction execution, packet processing, and timer handling, are not considered in the simulation and the DW-MAC protocol design. Ignoring these CPU overheads not only affects the correctness of latency com-

putation, it also results in improper network parameter settings and protocol designs. For example, due to the slow CPU speed, a sensor node may take much longer than SIFS on packet processing after receiving a packet. This affects the setting of the timeout value, the maximum length of time a node waits for an acknowledgement after sending a control and data packet. A long packet processing time also makes it possible that a node may hear another packet after getting a packet and before replying with the corresponding acknowledgement, which may degrade the channel efficiency or break the collision-free property of proportional mapping in DW-MAC. Considering this problem, in this work, I made a small modification to the proportional mapping in the original DW-MAC design for unicast data transmission so that it can still run correctly under such slow hardware and keep the same level of channel efficiency. Details will be discussed in Section 3.1.

Finally, in the work of Sun *et al.* [16], performance evaluation results only showed statistics at the level of end-to-end performance. Neither per-flow nor per-hop performance or behavior analysis was presented in their work. Detailed analysis would help in understanding how MAC protocols react to the environment and why some protocols outperform others in some metrics. In this work, in addition to the end-to-end level, the performance evaluation results are also shown and discussed in the per-hop and per-flow levels. The detailed results disclose drawbacks in the design of S-MAC, including the unfairness of flow competition and the channel inefficiency caused by the conservative collision avoidance mechanism of S-MAC. Through this thesis, the experiences of physical implementation and evaluation of S-MAC and DW-MAC contribute new information and insights for helping in future MAC protocol design and implementation in wireless sensor networks.

1.6 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, I present the overall design of S-MAC and DW-MAC. Both S-MAC and DW-MAC divide a cycle into three parts – the Sync, Data, and Sleep periods. They both leverage control packets to prevent collision on data packets and acknowledgements. However, the mechanisms their control packets are used for network resource reservation are quite different, which leads to different channel efficiency and other performance differences. I describe the similarity and differences of them in detail and illustrate how those two protocols work.

The details of this work are shown in Chapters 3 through 6. Chapter 3 discusses the problems mentioned in Section 1.5 and my solutions to those problems. I also describe the challenges I experienced in the implementation and experiments and the ways I addressed those challenges. Chapter 4 describes the network environment and settings in the performance evaluation of S-MAC and DW-MAC. The settings include parameters, network topologies, properties of the testbed, and traffic patterns. Performance evaluation results of the experiments and simulation are illustrated and discussed in Chapters 5 and 6, respectively. The end-to-end, per-hop, and per-flow level results are shown and analyzed in those chapters so that the detailed behavior and characteristics of S-MAC and DW-MAC can be shown.

Chapter 7 discusses some related work, including synchronous and asynchronous duty-cycling MAC protocols for wireless sensor networks. Brief descriptions and advantages and disadvantages of those protocols are given in the chapter. I also outline some previous work of MAC protocol implementation and show the relation and differences between this thesis and their work. Some time synchronization techniques are also introduced. Finally, this thesis concludes in Chapter 8.

Chapter 2

Synchronous Duty-Cycling MAC Protocols Studied

In this thesis, I chose to implement and evaluate two synchronous duty-cycling MAC protocols, S-MAC and DW-MAC. Both MAC protocols divide a cycle into three parts – the *Sync*, *Data*, and *Sleep* periods. Every node in the network stays awake in the Sync and Data periods for packet transmission. In the Sync period, nodes may send or receive time synchronization packets to coordinate their active and sleeping schedules. A node that has data packets to send may send control packets first in the Data period to compete with other nodes for channel usage. If a node does not have any data packets to send or receive, it will turn off its radio and go to sleep during the Sleep period to save energy. In S-MAC and DW-MAC, the total length of the Sync and Data periods divided by the length of a cycle is the *duty cycle*. The detailed design of S-MAC and DW-MAC is described in the following sections.

2.1 S-MAC

2.1.1 Scheduling and Time Synchronization

In S-MAC, a node chooses and maintains its schedule as follows. Initially, a node keeps listening for a short period of time. If the node does not hear any scheduling packet, i.e., Sync packet, it will start its own schedule, follow the schedule, and announce it with Sync packets. In contrast, if some Sync packet announced by others

is received first, the node will follow the schedule it hears and synchronize its time with the neighbor that announced the Sync packet. When a node receives a schedule that is different from the schedule it is following, the node will check if it is the only node that follows this schedule. If so, then the node will change to the new schedule it received. Otherwise, the node will adopt both schedules and keep awake in the Sync and Data periods of both schedules. This time synchronization and scheduling mechanism enables S-MAC to correctly act as a synchronous duty-cycling protocol in a large network, while it also brings more complexity for nodes that have to follow various schedules.

2.1.2 Contention and Collision Avoidance

Since nodes with the same schedule wake up at the same time, they have to contend for the medium when trying to send packets to their neighbors. Among contention protocols, S-MAC follows the collision avoidance mechanisms similar to what IEEE 802.11 does, including physical and virtual carrier sense and the RTS/CTS exchange for hidden nodes.

In the Data period, a node A that wants to send a data packet to a neighbor node B will first send an RTS to B. This RTS is used for requesting the medium. After receiving an RTS packet, node B will wait for a SIFS time and reply with a CTS packet back to node A, informing A that the channel is clear and that the data packet can be sent. Node A will send its data packet SIFS time after it successfully receives the CTS reply from node B, and if this is a unicast transmission, node B will reply with an ACK SIFS time after receiving the data packet. If the RTS/CTS exchange fails; that is, either one of the control packets RTS or CTS is lost during transmission or cannot be correctly decoded by the node, the node will try again later in the Data

period during the following cycle.

There is a duration field in RTS and CTS. This duration tells other nodes how long the RTS/CTS/DATA/ACK transmission sequence will last, which also means the length of time for which the medium will be busy. Upon receiving an RTS or CTS packet, if the RTS or CTS is not destined to it, a node checks the value in the duration field, sets the *network allocation vector* (NAV) to the maximum of current value and the duration, and sets a timer for the NAV. Before starting an RTS/CTS exchange, a node first checks its NAV. If the NAV is nonzero, the node regards the medium as busy and keeps silent until the NAV timer expires. This is called *virtual carrier sensing*, which is done at the link layer. RTS and CTS control packets help prevent collision on data packets and acknowledgements by reserving the medium for a period of time for their transmission. Therefore, in order to help avoid collisions of RTS or CTS, a node uses overhearing. Overhearing means that a node receives a message that is not destined to itself. Before starting an RTS/CTS exchange, if a node overhears an RTS or CTS packet, that means some other nodes have already started their transaction. In this case, the node should set its NAV timer and defer the RTS transmission until the timer expires. If a node overhears a CTS packet before sending or receiving its CTS packet, the node should turn off its radio so that it will not interfere with others' data packets and acknowledgement transmission.

Another form of carrier sensing is performed at the physical layer, called *physical carrier sensing*. On a MICAz node running TinyOS, the node makes a *clear channel assessment* (CCA) before sending a message. In a CCA, the node keeps sensing for carriers in the environment for a short and fixed period of time. After the CCA, if the medium is determined to be clear, i.e., both physical and virtual carrier sensing indicate the channel is clear, the node will start its packet transmission. In order to

decrease the chance that control packets are sent from different nodes at the same time and collide with each other, each node does a random backoff within a contention window before the CCA. In contention-based networks, multiple nodes may want to transmit in a Data period, and these mechanisms prevent collision on data packets and acknowledgements.

When the Sleep period begins, if a node has no data packets to send or receive in the current cycle, the node turns off its radio and go to sleep to save energy until the next cycle begins; otherwise, the node will finish its transaction and then go to sleep. A node may also go to sleep earlier than the end of the Data period due to overhearing a CTS packet. Through all the collision avoidance mechanisms above, S-MAC effectively reduces energy wasting due to idle listening and collision.

2.1.3 Adaptive Listening

Adaptive listening is a technique that S-MAC uses to decrease network latency in multihop unicast transmission. In the original S-MAC design [21], a packet can travel at most one hop in a cycle. If the path length of a flow is long, it will take a long network latency for the packet to be delivered to the destination. Adaptive listening turns S-MAC into a more active mode to shorten the end-to-end latency. For example, assume that node A wants to send a data packet to some destination node in the network via nodes B and C in sequence, i.e., B is the next hop of A, and C is the next hop of B in this case. First A sends an RTS to B to request a transaction. After B receives the RTS, B waits for SIFS time and replies with a CTS back to A. At the same time, C also receives this CTS because C is also in the transmission range of B. Before node C goes to sleep due to overhearing the CTS, it can check the duration field in the CTS and learn about when the transaction, i.e., the

RTS/CTS/DATA/ACK packet sequence, ends. With adaptive listening enabled, C will assume that it might be the next hop of B and wake up again after the transaction between A and B finishes. At that time, with the help of cross-layer cooperation, B can know which node is the next hop, so it can adaptively start a new RTS/CTS exchange to forward the data packet to C. An RTS/CTS exchange between B and C is still required so that collision avoidance can be ensured. The advantage of adaptive listening is that the network latency can decrease up to 50%. However, there is a drawback of wasted energy. Every node that overhears the CTS from B makes the same assumption that it might be the next hop, and all of these nodes will wake up after A and B finish their transaction. Only one of B's neighbor is the next hop, and others are just wasting their energy by waking up then and turning their radio on. In Chapter 5 and 6, I will analyze the impact of adaptive listening in detail.

2.2 DW-MAC

Similar to the design of S-MAC, the first period of a cycle in DW-MAC is the Sync period, the period of time nodes exchange their time synchronization packets. In the original DW-MAC work [16], however, a separate time synchronization protocol is assumed to exist and be used to synchronize clocks in the sensor nodes, and no synchronization mechanism is either simulated or implemented in that work.

2.2.1 Scheduling: On-Demand Wakeup and Proportional Mapping

In DW-MAC, nodes acquire the medium via control packets during the Data period, but the way the control packets are used for medium reservation is quite different from that in S-MAC. First, DW-MAC replaces the RTS/CTS control packets in S-MAC with a special packet called a *scheduling frame* (SCH), which plays roles for

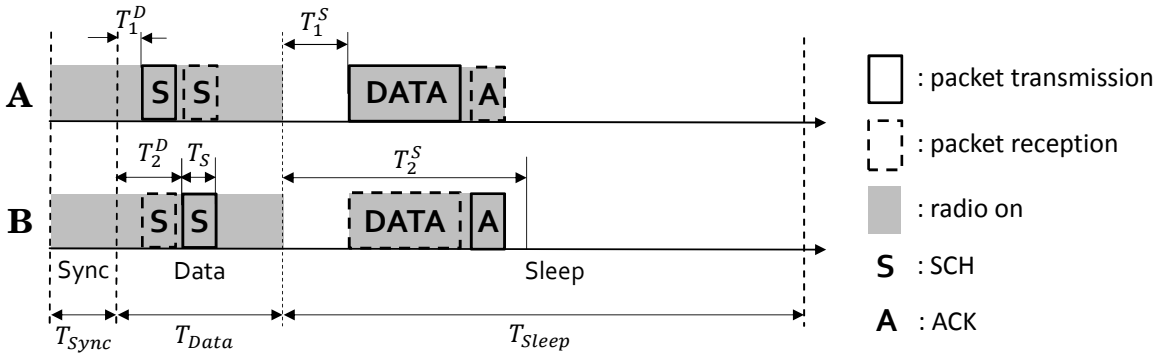


Figure 2.1 : Unicast data transmission in DW-MAC

channel competition and scheduling simultaneously. In addition, instead of sending and receiving a data packet right after this control packet, DW-MAC chooses to wake up nodes *on demand* in the Sleep period for data packet and acknowledgement transmission. Figure 2.1 shows how SCH is used for medium reservation and how on-demand wakeup works. In the Data period, node A with pending data packets first does a random backoff and physical carrier sense, the same as is done in S-MAC. If the channel is considered free, then node A sends an SCH to the next hop B for data transmission scheduling. At the same time, A records the value T_S, the transmission duration of the SCH, and T_1^D, length of time between the start of the Data period and the time the SCH was sent. With knowledge of the data rate and the size of an SCH, as node B receives the SCH, B can calculate the values T_1^D and T_S as well. Upon receiving the SCH, if this is unicast traffic, node B waits for SIFS time and then replies with another SCH as confirmation back to node A at time T_2^D after the beginning of the Data period. Similarly, node A can determine the value T_2^D after receiving the SCH from B. After a successful SCH exchange, both node A and B schedule a time in the Sleep period, and they will wake up at that time to finish

their data packet transmission. This time begins at T_1^S and extends to T_2^S after the beginning of the Sleep period, which is calculated based on *proportional mapping*:

$$\frac{T_1^D}{T_1^S} = \frac{T_2^D}{T_2^S} = \frac{T_{Data}}{T_{Sleep}}.$$

With proportional mapping, node A and B can uniquely determine the data transmission time without having any timing information in the SCH. The time between T_1^S and T_2^S also determines the maximum data and ACK transmission time.

DW-MAC relies on physical carrier sense and random backoff to reduce the chance of collision on SCHs. In the Data period, each node in the network that has pending data packets can freely send its SCH and contend for the medium any time the node find the channel is clear after its random backoff. This is different from the collision avoidance mechanism in S-MAC, especially in the case of control packets overhearing. While in S-MAC a node that overhears a CTS packet has to turn off its radio and go to sleep, in DW-MAC, the node can defer its SCH transmission until it senses the channel is free and send the SCH later in the same Data period. Theoretically, the proportional mapping mechanism guarantees that the data and ACK transmission in the mapped time period in the Sleep period is collision free if the corresponding SCH exchange in the Data period was successful. The proof is presented in the DW-MAC paper [16]. With the help of the proportional mapping and on-demand wakeup mechanisms, DW-MAC allows more nodes to deliver their data packets in a cycle than does S-MAC. Furthermore, in the Sleep period, because nodes only wake up at the necessary time, i.e., the mapped times, energy efficiency is not degraded in DW-MAC.

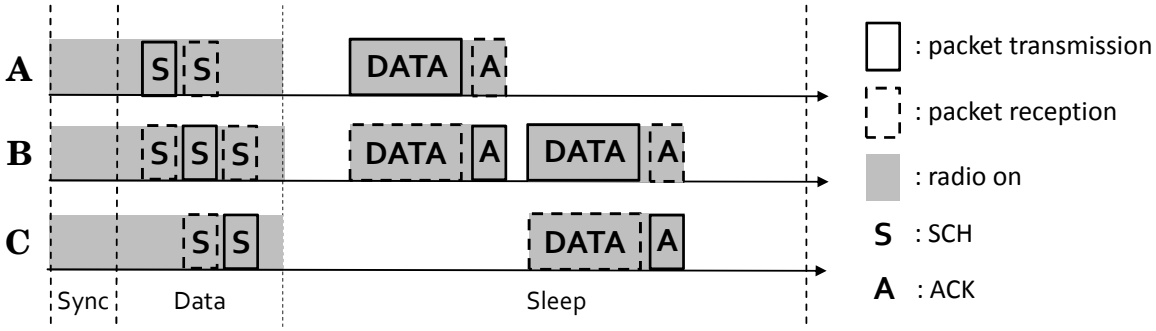


Figure 2.2 : Optimized multihop forwarding for unicast traffic in DW-MAC

2.2.2 Optimized Multihop Forwarding

In addition to the sender and receiver addresses and some other required fields, an SCH also includes the network-layer destination address. With the help of this field and cross-layer cooperation, a node receiving an SCH can learn at the link layer which node is the next hop. This design enables a data packet to go multiple hops within a cycle. Figure 2.2 shows an example of this. In the Data period, as node B replies with an SCH to node A after receiving A's SCH, the SCH B sends not only acts as a confirmation SCH for node A, it also plays the role of a scheduling request for B's next hop, node C. Because of the network destination address in the SCH, B can identify from A's SCH which node is the next hop and send another SCH to the next hop. After receiving the SCH transmitted by B, only the correct next-hop node, node C, will reply with a confirmation SCH back to B SIFS after receiving the SCH from B. Following this way, in the best case, only $n + 1$ SCHs are required for scheduling a data packet to go n hops in a cycle. Those n SCHs are mapped to n independent time periods in the Sleep period among the n hops, so the collision-free property is maintained. This is called *optimized multihop forwarding*. Compared with S-MAC, optimized multihop forwarding allows DW-MAC to deliver a data packet through

multiple hops within a cycle, whereas a packet can travel at most two hops in a cycle in S-MAC, even with adaptive listening enabled. This optimization greatly shortens network latencies in DW-MAC. Also, in S-MAC, at least $2n$ control packets are required for a data packet to go n hops, whereas in DW-MAC, only $n + 1$ SCHs are needed in the best case, which may save nearly half of the control packet overhead. Furthermore, although adaptive listening enables S-MAC to deliver a packet through two hops in a cycle, it also causes all the nodes that overhear the CTS packet to wake up. In contrast, in DW-MAC, only the correct next hop will wake up on demand in the Sleep period to receive (and forward) the data packet, further reducing energy waste.

Chapter 3

Physical Implementation Issues

As a part of the work in this thesis, I implemented DW-MAC under the UPMA framework [12, 5] in TinyOS 2.1.0 on a network of Crossbow MICAz sensor motes. All of the code is written in the *nesC* language, an extension to the C programming language. For consistency and fairness in the performance comparison, I also ported the official S-MAC code [3] from TinyOS version 1.x on Mica2 motes to TinyOS 2.1.0 on MICAz motes. The physical implementation of both protocols brings new issues and problems that were ignored in the simulated environment in previous work [16]. In the following sections, I will discuss those problems and present my solutions to them.

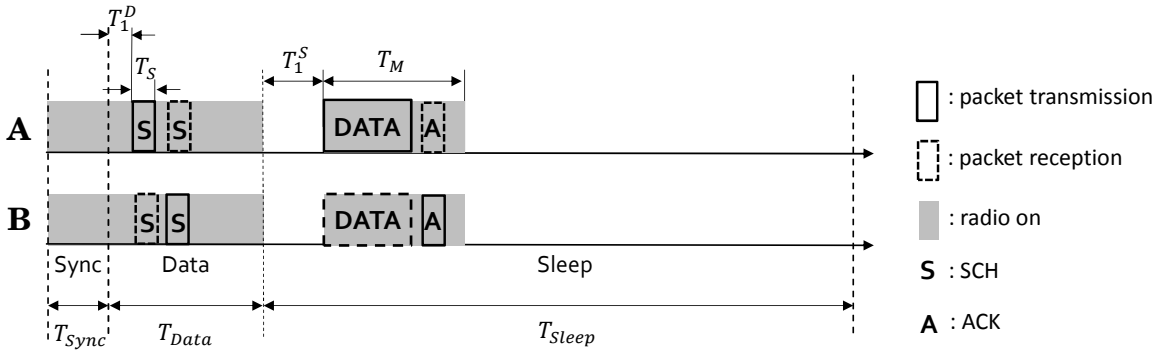
3.1 Enhanced Proportional Mapping in DW-MAC

In the original DW-MAC design for unicast transmission [16], after a node receives an SCH request, the node waits for SIFS time and then replies with a confirmation SCH. Based on my implementation experiences, however, it is nearly impossible to follow this rule. According to the definition of SIFS, which is the time the transmitting radio needs to switch back to receiving mode and be able to decode incoming packets, the value of SIFS is very short on the sensor hardware such as MICAz motes. In contrast, the CPU on a MICAz mote is relatively slow so that it takes longer than SIFS time for the CPU to process an incoming or outgoing packet. The packet processing time

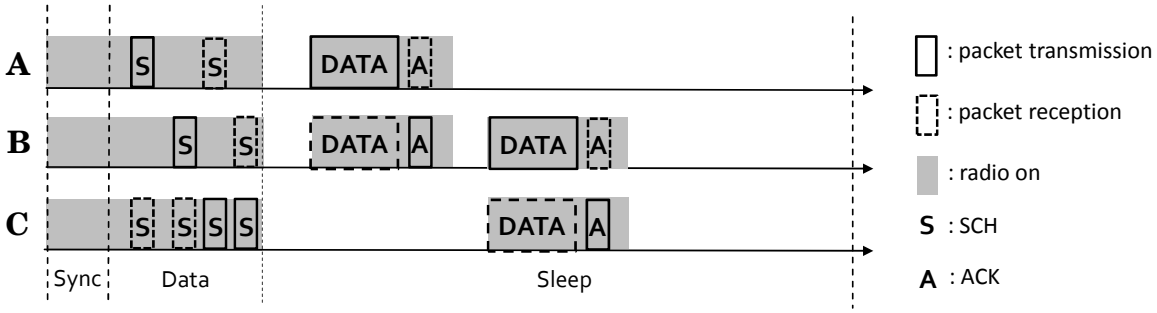
is so long that a node may receive two or more SCH requests from different senders before the node replies with any confirmation SCH. The node may then have two different policies to deal with this situation. The first policy is ignoring the second and later incoming SCH requests and just replying to the first SCH request. Although the original DW-MAC proportional mapping still works under this policy, the policy itself may cause many SCH requests be dropped in the Data period, and nodes that fail to get a successful SCH exchange have to try again, increasing overheads and network latency. The alternative policy is queuing all the incoming SCH requests and replying to them one by one when the node is not busy. This policy, however, may cause mapped time slots in the Sleep period to overlap with each other and break the collision-free property of proportional mapping. In order not to degrade the channel efficiency, I chose the second policy in this work and adapted the original proportional mapping mechanism to work correctly on the hardware with such a slow CPU.

As shown in Figure 2.1 of Section 2.2.1, the original proportional mapping maps a time period between the beginnings of an SCH request and that of its confirmation SCH in the Data period to the Sleep period. Instead, in my enhanced version of proportional mapping, a scheduled time period in the Sleep period is mapped from the small period of time when an SCH request is being transmitted on the air. The enhanced version of proportional mapping is similar to that in the original DW-MAC design for broadcast transmission. Figure 3.1(a) illustrates the design of the enhanced proportional mapping. Similar to the original version, after the successful SCH exchange, both nodes A and B wake up T_1^S after the beginning of the Sleep period. The length of the mapped time in the Sleep period, however, becomes T_M, where

$$\frac{T_1^D}{T_1^S} = \frac{T_S}{T_M} = \frac{T_{Data}}{T_{Sleep}}.$$



(a) Basic design



(b) An example of receiving two SCH requests before any reply

Figure 3.1 : Enhanced proportional mapping for DW-MAC

With this modification, a node that receives an SCH request simply has to reply to it before the end of the Data period and before the sender's SCH timeout, which provides more flexibility in time scheduling. An example is shown in Figure 3.1(b). This modification not only keeps the channel and energy efficiency and the collision-free property of proportional mapping, it also makes DW-MAC compatible with various hardware characteristics. The proof of the collision-free property for the enhanced proportional mapping is similar to that provided in the original DW-MAC paper [16].

Any proportional mapping mentioned in the remainder of this thesis refers to this enhanced version.

3.2 Timing Inaccuracy

Accurate timing is important to the correct operation of synchronous duty-cycling MAC protocols such as DW-MAC. However, in sensor nodes, there are two reasons for inaccurate timing: *clock drift* and *hardware interrupt disabling*.

Clock drift is a phenomena where a clock does not run at exactly the correct speed compared with other clocks or the same clock itself at different times. Clock drift exists in virtually all clocks in the world, including modern computers and sensor nodes. In sensor nodes, due to the relative low cost and simple design compared with desktop or laptop computers, clock drift is often more severe. Based on my observation on the MICAz motes, the clock drift could be around 20 μs per second. This is a significant amount. For unattended sensors, this magnitude of clock drift may cause 72 ms time difference among nodes after an hour. In the performance evaluation in this thesis, however, the lengths of the Sync and Data periods are 55.2 ms and 89 ms, respectively. A 72 ms time difference could lead the nodes into great confusion about the periods in a cycle.

The other cause of timing inaccuracy comes from hardware interrupt disabling. TinyOS maintains its clock time using a periodic interrupt from the hardware. If the hardware interrupt is temporarily disabled anytime when a mote is running, the periodic interrupt from clock ticks may be delayed, influencing the correctness of the clock time. One thing that may cause the hardware interrupt to be temporarily disabled is logging to the flash storage on the mote. As the mote starts to write logs to the flash, hardware interrupts are disabled until the writing finishes. Logging itself, however, is almost inevitable, especially in experiments for performance analysis, since every node in the network has to record some information about the packets sent and received.

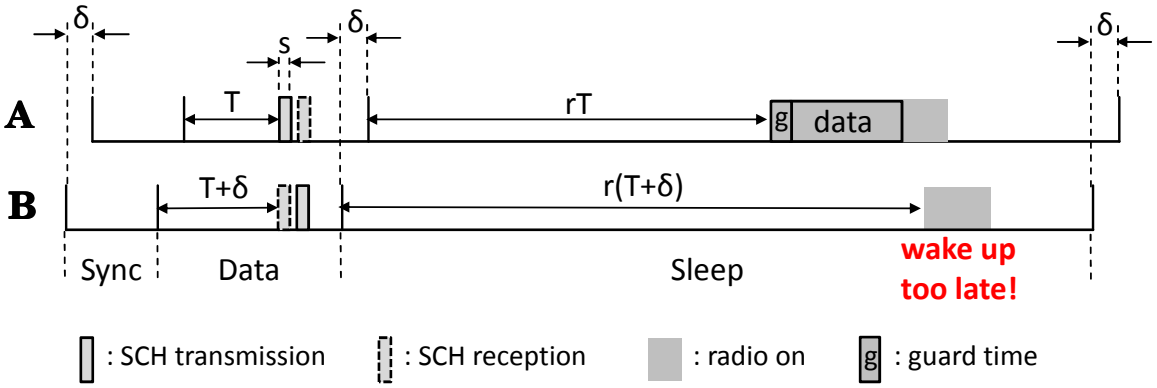
Solving either problem above directly from the hardware or operating system level is not easy. Instead of solving them directly, I choose to minimize the time difference among nodes from the view of cycles; that is, the goal is making the nodes start their cycles almost at the same wall clock time, regardless of what their clock times on the motes are. Before describing the solutions, I discuss the impacts of time differences on DW-MAC.

3.2.1 Impact on DW-MAC

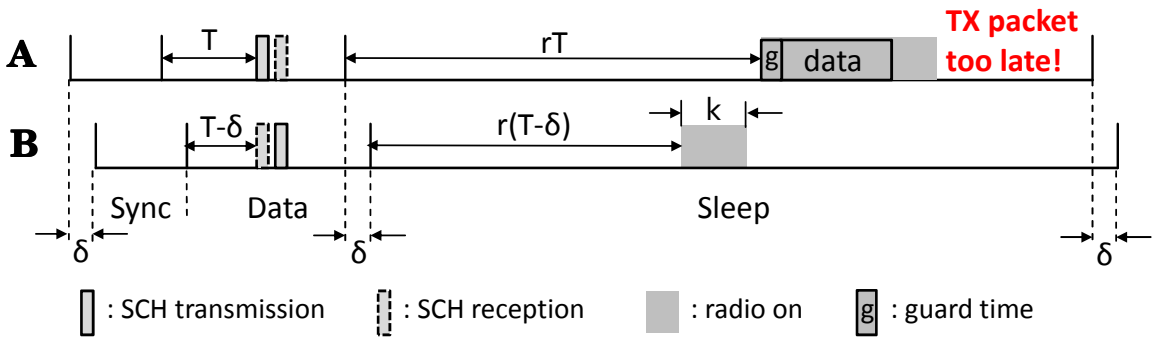
In synchronous duty-cycling MAC protocols, time differences cause nodes not to start their cycles at the same time. Whereas in S-MAC where a small time difference among the nodes does not result in significant impacts to the correctness of protocol operation, DW-MAC may suffer much more significantly from a small time difference. The reason why DW-MAC is so sensitive to the timing is that DW-MAC uses on-demand wakeup and proportional mapping when scheduling data packet transmissions. Assume the ratio of the length of the Sleep period to the length of the Data period is r:

$$r = \frac{T_{Data}}{T_{Sleep}}.$$

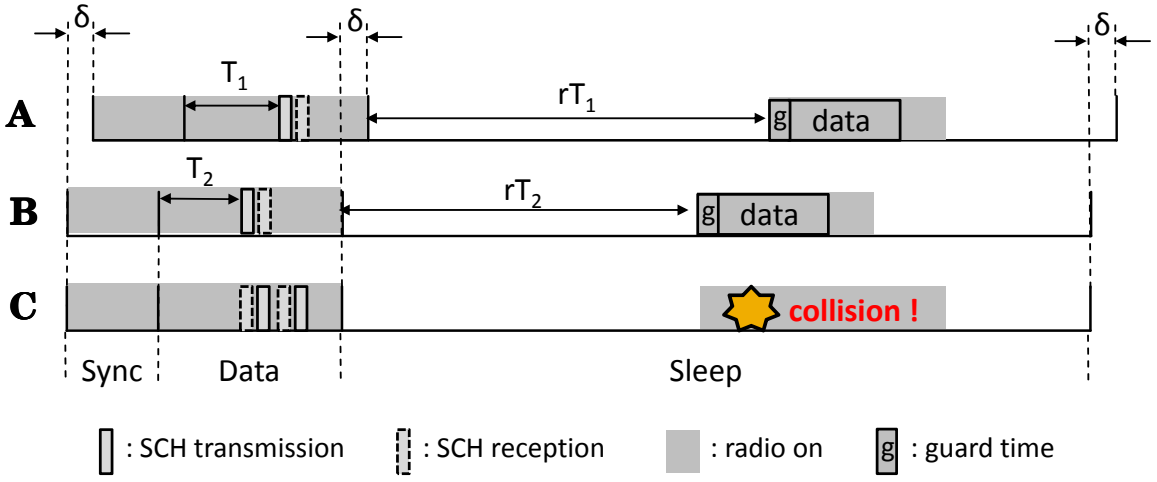
Through proportional mapping, any time in the Data period will be mapped to a period r times longer in the Sleep period. If there is a small time difference δ between two nodes, that difference δ in the Data period will also be amplified into an $r\delta$ schedule difference in the Sleep period; that is, one node may wake up $r\delta$ later than the other node. With the $r\delta$ schedule difference, the collision-free property of proportional mapping in DW-MAC is no longer guaranteed, and packets may be lost due to late wakeup, late transmission, or collision. Figure 3.2 shows these cases of packet loss.



(a) Late wakeup



(b) Late transmission



(c) Collision

Figure 3.2 : Impacts of time difference among nodes on DW-MAC

In Figure 3.2(a), node A has a packet to be sent to node B, and A first sends an SCH to B in the Data period. After the successful SCH exchange, both A and B schedule a time in the Sleep period to wake up for data packet delivery; however, the time periods they have scheduled are not the same. Due to the time difference δ between A and B, A thinks the SCH is sent at time T after the beginning of the Data period, but from B's view, the SCH is sent at time $T + \delta$. According to proportional mapping, A wakes up at time rT after the beginning of the Sleep period based on its view, whereas B wakes up at time $r(T + \delta)$ from B's view. In fact, B wakes up $(r - 1)\delta$ later than A does. At the time B wakes up, A has finished its data packet sending. Node B does not receive the data packet, and node A does not get an ACK from B.

Another case is similar, but in this case node B wakes up too early so that A wakes up after B reaches its receiving timeout and goes to sleep. This case is shown in Figure 3.2(b). In this case and the previous case, the data transmission fails, which also leads to retransmission in the next cycle and latency increase and energy waste.

In the last case, data packets collide with each other. Figure 3.2(c) shows an example of this case. Time is perfectly synchronized between nodes B and C, but A starts its cycle δ later than B and C do. Both nodes A and B have data packets to send to node C in the same cycle, and in this example they each send an SCH to C at time T after the Data period starts based on their views of time. If time is perfectly synchronized between A and B, then their SCHs should collide with each other. In this example, however, δ is larger than the time needed for an SCH exchange, and the SCH exchange between A and C is completely separated from that between B and C. Since both SCH exchanges are successful, A and B wake up at time rT after the beginning of the Sleep period and send their data packets. In this case, δ is less

than the time needed for data packet transmission, and collision occurs at C. Both A and B fail to get their data packets delivered, and they have to try again in the next cycle.

To solve these problems, the time difference δ should be kept as small as possible, which leads to the requirement of periodic time synchronization. In real-world hardware, however, it is very difficult to perfectly synchronize the time even for only two nodes. There is always an error, and the error would create nonzero probability of such packet loss. To address this problem, I add a *guard time* into the DW-MAC design. Adding the guard time allows DW-MAC to function correctly in spite of such imperfect time synchronization. Applying the two mechanisms of periodic time synchronization and use of a guard time together brings back the theoretical collision-free property of proportional mapping in DW-MAC. Details of these two mechanisms are described in Sections 3.2.2 and 3.2.3, respectively.

3.2.2 Time Synchronization

The previous section described the importance of time synchronization in DW-MAC. However, time synchronization is required for all duty-cycling MAC protocols, including S-MAC. A simple time synchronization mechanism is adopted in this work, for both S-MAC and DW-MAC, for globally synchronizing nodes in a small network. In the experiments, a *global synchronizer* was placed in the middle of the network. The radio transceiver of the synchronizer is adjusted to a higher transmit power so that all the nodes in the network can hear messages from the synchronizer. The synchronizer broadcasts consecutive Sync packets until the end of the Sync period. Each Sync packet contains a field that records the duration between the time the Sync packet is sent out and the beginning of next Data period. With the duration

field and knowledge of the radio transmission rate, each node that receives any Sync packet can calculate the time to the beginning of next Data period. For each node in the network, as long as one Sync packet is received in the current Sync period, the node can start its next Data period simultaneously with other nodes, although different nodes may have different absolute clock times in themselves.

The synchronizer must put a correct value in the duration field for every Sync packet. In TinyOS, this can be done only at the moment that the first byte of a Sync packet is just to be sent over the air, since the packet processing time in the hardware cannot be accurately predicted. When encapsulating an outgoing packet at the physical layer, the hardware adds a byte, called the *start frame delimiter* (SFD), between the preamble and the packet body. The SFD denotes the beginning of a packet. Upon seeing the SFD when sending or receiving a packet, the hardware generates an interrupt to the operating system, here TinyOS, and tells the operating system that the SFD has been processed. In the interrupt handler, the correct value in the duration field of a Sync packet can be obtained by computing the difference between the time the interrupt happens and the beginning of next Data period.

In this work, each node running S-MAC or DW-MAC is assumed to know the length of each period in a cycle. Even if a node fails to receive any Sync packet in a certain cycle, clock drift would not drive the node too far away from the correct schedule in a short period of time. The node can still follow others' schedule since the last time it was synchronized. Therefore, the global synchronizer does not have to broadcast Sync packets in every cycle. To reduce synchronization overheads and keep the schedule synchronized on each node, time synchronization is set to be performed every other cycle in the experiments in this thesis.

Through this mechanism, time can be coarsely synchronized, but not perfectly.

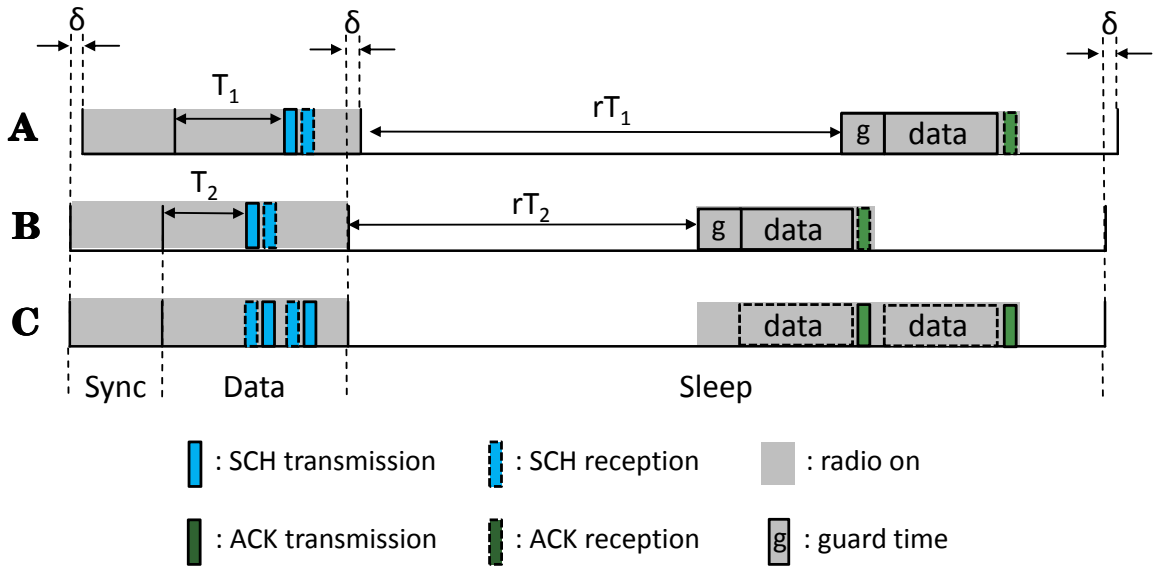


Figure 3.3 : The design of the guard time in DW-MAC

This is due to the limitation of the timer granularity in the hardware. All the time-related operations in the hardware rely on the timer. Crossbow MICAz motes provide three different levels of timing precision: millisecond, 28.8 kHz, and microsecond, where the millisecond and 28.8 kHz timers are more accurate and stable than the microsecond timer. In this work, both the former two timers are used in the implementation for different purposes and different timing requirements. For example, DW-MAC uses the 28.8 kHz timer for time synchronization, since DW-MAC is more sensitive to time difference among nodes.

3.2.3 Guard Time

Time synchronization itself is not enough for satisfying the collision-free property in the Sleep period in DW-MAC, i.e., the time difference δ mentioned in Section 3.2.1 is still nonzero. Therefore, additional mechanisms are required as well for guaranteeing

successful data transmission in the Sleep period. In this work, I add a *guard time* into the DW-MAC design. Guard time is a small period of time allocated in the front of a mapped time period in the Sleep period if the time period is scheduled for sending a data packet. Figure 3.3 depicts the design of the guard time. A node that schedules the sending of a data packet wakes up at the beginning of the mapped time period. Instead of sending the packet immediately, the node waits for the guard time first and then sends the packet. On the other hand, the intended receiver wakes up at the scheduled time and keeps listening to the channel for a while. If the receiver does not receive any packet after timeout happens, the receiver will give up and go to sleep. The cases of packet loss or collision mentioned in Section 3.2.1 can be completely avoided if the length of the guard time g is greater than or equal to $(r-1)\Delta$ and the receiving timeout value is greater than or equal to $2(r-1)\Delta$. The symbols are defined in Table 3.1. The proof of these claims is provided in Section 3.2.4. Additionally, the size of a data packet needs to be limited. Based on my experiment settings in Tables 4.1 and 4.2, the data packet size should be smaller than 150 bytes so that the collision-free property can still hold. This restriction, however, does not cause any problem on the MICAz motes because the maximum packet size the motes support is 128 bytes.

3.2.4 Proof of Lower Bounds for the Guard Time and Receiving Timeout

The proof is divided into three cases. The first case is to prevent late wakeup, which is depicted in Figure 3.2(a). To prevent late wakeup, the following condition must be satisfied:

$$r(T+\Delta) < rT + g + \Delta \Rightarrow (r-1)\Delta \leq g \tag{3.1}$$

Δ	Maximum time difference between any two nodes
s	Duration of SCH transmission
g	Length of the guard time
d	Duration of data packet and ACK transmission
k	Timeout value for waiting a data packet
r	The sleep-to-data ratio $\frac{T_{Data}}{T_{Sleep}}$

Table 3.1 : Symbols used in the proof

The value $g + d$ must be less than or equal to rs, which is the length of the mapped time period. Consider the boundary condition and assume that $g + d = rs$.

In the second case, as shown in Figure 3.2(b), node A has to transmit the data packet after node B wakes up and before B's timeout occurs so that node B can receive the whole packet from A. Thus, the following condition must hold:

$$\begin{aligned} & \Delta + r(T - \Delta) < rT + g < \Delta + r(T - \Delta) + k \\ \Rightarrow \quad & g < k - (r - 1)\Delta. \end{aligned} \tag{3.2}$$

Figure 3.2(c) illustrates the last case. To avoid collision, the packet transmission duration between nodes A and B should not overlap. Additionally, between A and B, the node that sends its SCH request first (based on node C's view) must send its data packet first as well. If A sends its SCH request prior to B:

$$\Delta + T_1 + s \leq T_2 \tag{3.3}$$

$$\Delta + rT_1 + rs \leq rT_2 + g. \tag{3.4}$$

Equation 3.3 derives

$$\begin{aligned} & r\Delta + rT_1 + rs \leq rT_2 \\ \Rightarrow \quad & (\Delta + rT_1 + rs) + (r-1)\Delta \leq rT_2 \leq rT_2 + g, \end{aligned}$$

which implies Equation 3.4. In the other case, B sends its SCH request prior to A:

$$\Delta + T_1 \geq T_2 + s \tag{3.5}$$

$$\Delta + rT_1 + g \geq rT_2 + rs. \tag{3.6}$$

Equation 3.5 derives

$$\begin{aligned} & r\Delta + rT_1 \geq rT_2 + rs \\ \Rightarrow \quad & \Delta + rT_1 + (r-1)\Delta \geq rT_2 + rs, \end{aligned}$$

and Equation 3.6 holds if

$$g \geq (r-1)\Delta. \tag{3.7}$$

To meet all the conditions in Equations 3.1, 3.2, and 3.7, the values of the guard time g and receiving timeout k must satisfy

$$\begin{cases} g \geq (r-1)\Delta \\ k \geq 2(r-1)\Delta. \end{cases}$$

■

3.3 Packet Timestamping

Packet timestamping is a feature of TinyOS that stamps the clock time on the header of a packet when an incoming or outgoing packet is processed by the hardware. This timestamp, for example, helps a MAC protocol knows the actual time a packet was sent or received by the hardware. The time information is sometimes very important

to MAC protocols. In S-MAC with adaptive listening enabled, for example, a node that overhears a CTS packet needs to know when to wake up adaptively to receive the corresponding data packet. A node that receives a Sync packet in the Sync period also has to know the clock time the packet was received so that the node can correctly predict when the next Data period begins. This timestamp is even more important to DW-MAC because of the proportional mapping and on-demand wakeup features of DW-MAC. If the sender and the receiver do not agree on the same timing of SCH transmission, then they may wakeup at different times in the Sleep period and fail to get the data packet delivered, even if their clock times are perfectly synchronized.

In TinyOS, timestamping is done in the interrupt handler when a start frame delimiter is detected. The timestamping, however, has to be done very quickly since the packet transmission is in progress. In addition, the interrupt handler is preemptable; that is, another interrupt may occur before the current interrupt processing finishes. When the node is very busy, there might not be enough time to timestamp a packet, and the timestamp field in the packet previously set to a special constant, which denotes an invalid packet timestamp in TinyOS, is not overwritten. In this case, timestamping fails, and what the link layer sees would be the invalid packet timestamp.

Designing a solution for making timestamping always successful could be very difficult or even impossible. Instead, in this work, I choose a preliminary but simpler way to mitigate the impact of invalid packet timestamps by estimating the delay between the time SFD is processed by the hardware and the time link layer receives the packet. This estimation is done statistically based on the historical experiences of processing delay by the node. When an incoming packet arrives at the link layer, if its timestamp is valid, the link layer calculates the processing delay and incorporates the

value into its records. Additionally, the link layer also computes the mean delay in the historical records and keeps updating the mean value as new records are included. When an incoming packet with invalid timestamp is received, the link layer uses the mean value as the estimate of the processing delay of the packet. Combined with the current clock time at which the link layer receives the packet, the estimated SFD time is calculated.

For outgoing packets, a similar statistical estimation method is used. Upon finishing a packet transmission, the hardware generates another interrupt to tell the upper layer the transmission is done. The link layer computes the processing delay based on the time the interrupt happens and the packet timestamp on the copy of the transmitted packet, if the timestamp is valid. On the other hand, if the timestamping fails, the link layer refers to the historical statistics to estimate the processing delay.

Chapter 4

Performance Evaluation Methodology

The purpose of my implementation of both S-MAC and DW-MAC was not only to show their differences on performance metrics but also to examine their detailed behavior and analyze how they respond to various network environments. This chapter describes the environment and methodology of the experiments and simulation, hardware properties, network topologies, traffic settings, and other network parameters used. Additionally, experiences of my implementation and experiments are also discussed. The performance results of the experiments and simulation will be covered in the following two chapters.

4.1 Physical Implementation

My implementation was done on Crossbow MICAz motes, such as depicted in Figure 4.1. A MICAz mote has an Atmel ATmega128L 8-bit microcontroller with 7.37 MHz clock, 128k bytes program flash memory, 4k bytes RAM, and 512k bytes mea-

Figure 4.1 : Crossbow MICAz mote

Bandwidth	250 kbps
Max. packet size	128 bytes
Voltage	3.3 V
TX current	$8.5 - 17.4$ mA, 8 levels
RX current	18.8 mA
Idle mode current (radio on)	18.8 mA
Idle mode current (radio off)	0.426 mA

Table 4.1 : CC2420 hardware properties

surement flash memory. The mote is powered by two AA batteries. MICAz motes also provide three different hardware clocks, which generate interrupts at millisecond, 28.8 kHz, and microsecond levels, respectively. Each MICAz mote is equipped with a 6 cm long antenna. The radio chip on MICAz motes is CC2420, which uses a 2.4 GHz IEEE 802.15.4-compliant RF transceiver. The modulation technique CC2420 uses is digital direct sequence spread spectrum (DSSS). Detailed properties of the CC2420 radio are listed in Table 4.1.

The experiments were done in a network composed of nine sensor nodes and one global synchronizer in an indoor environment with walls, server racks, chairs, and other obstacles. All the nine nodes were placed on the floor, and the synchronizer was put on a chair about half meter high. The nine nodes were configured with the lowest transmission power, which is -25 dBm, whereas the transmission power of the synchronizer was set to -10 dBm.

As shown in Figure 4.2, the nine nodes form a cross topology. The synchronizer is

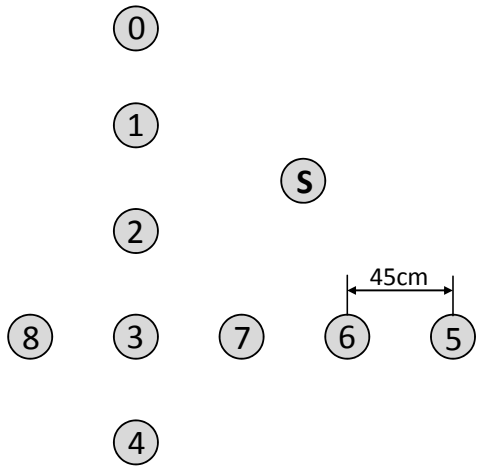


Figure 4.2 : Network topology used in the performance evaluation

located at nearly the center of the network and configured with higher transmission power so that every sensor node can hear Sync packets from the synchronizer. During an experiment, the synchronizer broadcasts Sync packets to the whole network in the Sync period every other cycle. The inter-node distance between any two consecutive nodes on a line of the cross is 45 cm. This distance guarantees that a node can almost always hear from its neighbor and decode messages if no collision or interference exists.

There are two unicast flows in the network. Those two flows fall on the two lines of the cross, respectively. Packets on one flow are generated by node 0 and go down to node 4. On the other flow, node 5 is the source, and node 8 is the destination. Static routing is used. Packets on each flow travel node-by-node from the sources to the destinations. The two flows intersect at node 3. During a round of an experiment, both nodes 0 and 5 generate data packets at the application layer simultaneously and periodically. There are totally 10 packets generated in each flow in each experiment. Simultaneous packet generation on the two flows increases the network contention, which helps analyze how S-MAC and DW-MAC respond to harsh network situations.

Data packet size	40 bytes	T_{Sync}	55.2 ms
SCH size	14 bytes	T_{Data}	89 ms
Size of RTS/CTS/ACK	10 bytes	T_{Sleep}	2739.8 ms
Min. contention window size	8	T_{Cycle}	2884 ms
Max. contention window size	64	Duty cycle	5%
SCH/CTS timeout	25 ms	Sleep-data ratio (r)	30.78
ACK timeout	10 ms	Control packet retry limit	7
Guard time	1.06 ms	Data packet retry limit	5

Table 4.2 : Network parameters used in the performance evaluation

There are four different inter-packet intervals, 2.5, 5, 10, and 20 seconds, in the experiments. The purpose of varying packet intervals is to compare the performance of S-MAC and DW-MAC under different traffic loads. In each experiment, the sources wait for 20 seconds before generating the first packet, and the experiment lasts for 240 seconds. Other network settings are listed in Table 4.2.

The settings of T_{Sync} and duty cycle are consistent with those used in the DW-MAC paper [16]. The length of the Data period, T_{Data}, is calculated by summing up the maximum required time for random backoff and RTS/CTS transaction in S-MAC. Although it might be a little unfair to DW-MAC because an SCH is a little larger than an RTS or CTS in size, DW-MAC is not favored in any way, and DW-MAC still outperforms S-MAC based on the evaluation results.

The minimum and maximum contention window sizes are also defined in Table 4.2. In each experiment, after a node sends an RTS or SCH request, if the node does not

Propagation model	TwoRayGround
Antenna model	Omnidirectional antenna
Capture threshold	8.0
Carrier sense threshold	1.29528e-10 W
Reception threshold	2.29591e-10 W
Transmission power	3.1623e-6 W
Radio frequency	2.4385 GHz
System loss factor	1.0

Table 4.3 : Physical layer parameters used in *ns*-2 evaluation

get a CTS or SCH reply after timeout occurs, the node will double its contention window size; otherwise, the contention window size will be cut half. The initial contention window size is set to the minimum.

4.2 Simulation

I also evaluated S-MAC and DW-MAC using the *ns*-2 simulator version 2.29, and most of the network settings are the same as those in the experiments. In order to simulate the radio behavior of MICAz motes, parameters at the simulated physical layer are set based on what were seen in the experiments. Table 4.3 lists those parameters. The experiment results show that nodes 0 and 4 can sometimes hear and decode packets from further hops than other nodes do. Similarly, packets sent by nodes 0 and 4 can also be delivered to further distances. I believe this is because the experiment environment is not free space, and constructive multi-path reflection might happen

at those two nodes. Considering this, I set the transmission power and reception threshold of those two nodes to different values, 4.7608e-6 W and 1.52503e-10 W, respectively. Since radio behavior in the real world is much more complicated than that in the simulator, such parameter adjusting is reasonable and makes the simulated radio behavior closer to the real hardware.

Beside the radio parameter settings, there are some aspects that are not perfectly simulated. First, the transmission range of the simulated radio is a perfect circle, and there are no obstacles in the simulated environment. In the simulation, as the receiving signal strength is above the reception threshold, the receiver can always decode incoming packets successfully if there is no collision. On the other hand, if the signal strength is below the threshold, the packets can be heard but cannot pass the CRC check. If the signal strength is even lower, i.e., lower than the carrier sense threshold, none of the packets can be heard. In addition, unlike the real physical implementation environment, there is no variable or random component to the relationship between signal strength and distance in the simulation model; that is, with a given setting of transmission power and distance, the receiving signal strength is deterministic.

Second, the computation overhead is not fully simulated. For example, the exact time needed for packet processing is not predictable on the motes. After receiving a request or a data packet, a node has to analyze the packet it received and prepare for a reply and then send the reply back to the sender of the request or data packet. During this packet processing, other events might occur, either internally or externally, that could influence the processing time of the packet. Although the range of the packet processing time can be learned from the experimental results, it is difficult to know its distribution. In the simulation, the processing time is randomized between the observed minimum and maximum packet processing time in the experiments. Similar

things happen when a node is turning its radio on or off, for which the exact duration is also not predictable. The minimum duration of radio state transition observed in the experiment is applied in the simulation. Beside the differences between the simulation and experiments described above, packet timestamping is always successful in the simulation, because time is "paused" when the simulator is processing an event.

Chapter 5

Experimental Results

This chapter presents the experimental results of S-MAC and DW-MAC with the four inter-packet intervals as described in the previous chapter. For each pair of protocol and inter-packet interval setting, the experiment is run 10 times, and the average results are presented. Performance metrics I measure include end-to-end latency, per-hop latency, energy consumption, and packet delivery ratio. I also explore the reasons that cause latencies at each hop, including analyzing how adaptive listening works under various network conditions.

5.1 Overall Performance Analysis

5.1.1 End-to-End Latency

Figure 5.1 shows the end-to-end latency of packets under different inter-packet intervals. In both MAC protocols, the end-to-end latency decreases as the inter-packet interval increases. When the inter-packet interval is small, the traffic is bursty, and the chances of collision and contention on the control packets, i.e., SCH, RTS, and CTS, is high. This contention and collision causes latencies on the data packets. In this situation, additionally, the rate at which packets are generated is faster than the rate packets are drained from the nodes. Thus, many packets are queued in the middle of the network, waiting to be forwarded to the next hop. As inter-packet interval increases, network loads are distributed over time, and the chance of contention

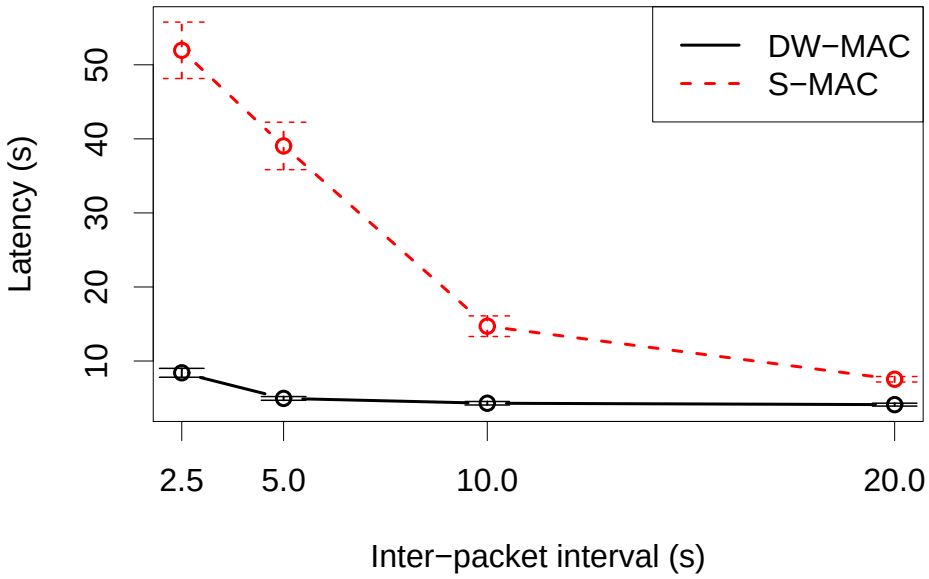


Figure 5.1 : End-to-end latency comparison

decreases. In these experiments, DW-MAC always outperforms S-MAC in terms of end-to-end latency. The influence of network loads on the latency is much smaller in DW-MAC than in S-MAC. The heights of the error bars also show that DW-MAC is more stable in performance than is S-MAC. In the case of 2.5 s inter-packet interval, the average end-to-end latency of DW-MAC is around 8.4 seconds, which is longer than the packet interval. Queuing itself contributes a significant part to the latency, as does contention. Not only packets on different flows create contention, but packets on the same flow also compete with each other for the network channel. With longer inter-packet intervals, the latency of DW-MAC is less than the packet interval, and thus almost every packet can be delivered to the destination before the next packet is generated. Thus, inter-packet interference on the same flow rarely exists. This is

why DW-MAC keeps an almost flat curve of end-to-end latency when the inter-packet interval is above 5 seconds.

On the other hand, S-MAC has larger end-to-end latency than the inter-packet interval when the inter-packet interval is less than 20 seconds. The reason why S-MAC has much longer latencies is that S-MAC suffers from much more contention, and it takes longer for each packet to get to its destination. In S-MAC, when a node overhears a CTS packet before sending any RTS or CTS packet, or when the node overhears an RTS packet before sending its own RTS packet, the node has to defer the RTS or CTS transmission or cancel its current RTS/CTS transaction. Because of this collision avoidance mechanism and how the length of the Data period is decided, the node has to wait until the next cycle and must then compete with other nodes again. In the experimental network (Figure 4.2), nodes have chances to hear packets from two to three hops away. If many nodes in the network try to start their RTS/CTS transaction in the same Data period, it is very likely for a node to overhear RTS or CTS packets. Finally, only a small portion of nodes can finish their RTS/CTS transactions and forward the data packets in a cycle, and other nodes have to make another attempt in the next cycle. Packets spend most of the time on queuing, so the end-to-end latency is linearly and inversely proportional to the inter-packet interval when the interval is between 2.5 and 10 seconds.

One reason why DW-MAC can drain packets off from the network so fast is its optimized multihop forwarding. While adaptive listening in S-MAC enables a packet to travel at most two hops in a cycle, optimized multihop forwarding in DW-MAC does not have this limitation. Without packet loss or collision, in a cycle, an SCH in DW-MAC can go any number of hops and reserve time periods for the nodes to which the SCH has traveled. The maximum number of hops an SCH can go in a cycle is

limited only by the length of the Data period, since SCHs cannot be sent outside the Data period. Based on my experimental results, many SCHs can travel three hops in a cycle. The difference of the collision avoidance mechanisms between S-MAC and DW-MAC is another important factor that differentiates their performance. In DW-MAC, if a node detects a busy channel or overhears an SCH from another node before sending its own SCH, the node simply remains silent for a short time and then does a clear channel assessment and tries again. As long as there is still enough time, the node can finish the SCH transmission in the current Data period, even if it is deferred for a little while. Additionally, a node is allowed to send multiple SCH requests or confirmations at different times in a Data period to reserve multiple time periods to forward multiple data packets in the same Sleep period. In contrast, a node running S-MAC probably has to defer its control packet transmission to the next cycle after overhearing control packets from other node, and for a node, at most one successful RTS/CTS transaction may occur in the Data period. Due to all the reasons above, packets in DW-MAC spend less time in queuing and also less time on traveling in the network. Therefore, the overall network latency of DW-MAC is much shorter than that of S-MAC.

5.1.2 Packet Delivery Ratio

The packet delivery ratios of both protocols are shown in Figure 5.2. DW-MAC maintains 100% packet delivery ratio (PDR) across all packet intervals. S-MAC always has a PDR higher than 95%, but it is still worse than that of DW-MAC under bursty traffic and higher network loads. As described in Section 5.1.1, nodes running S-MAC spend longer time on competing with each other for the channel in the Data period, but in each round only a small portion of the nodes win and get a successful

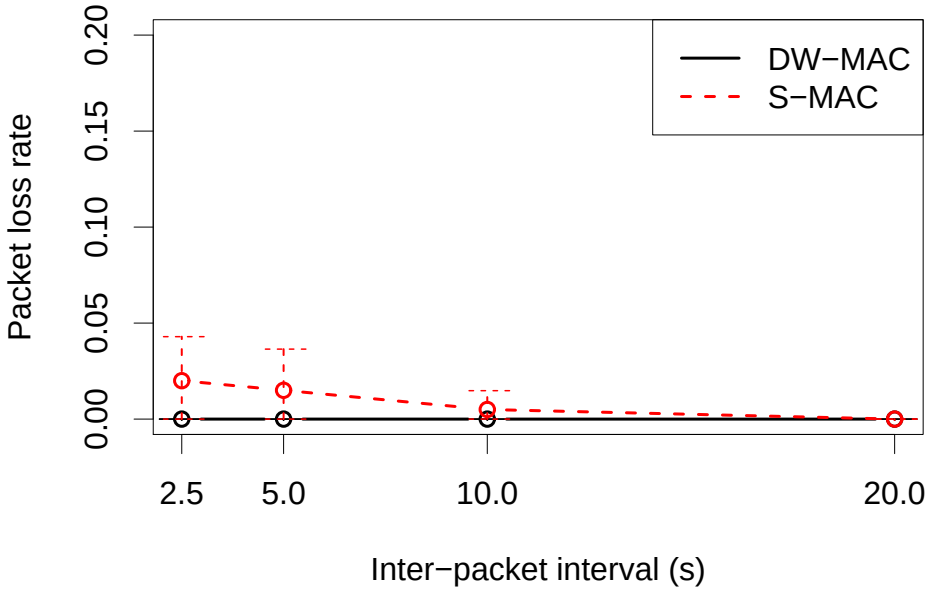


Figure 5.2 : Packet delivery ratio comparison

RTS/CTS exchange. Since many nodes are trying to send their control packets in the data Period, the more attempts a node makes on the control packet transmission, the higher chance the control packets are lost such as by colliding with each other. In the experiments, the control packet retry limit is 7 in both protocols (Table 4.2); that is, for each data packet, a node has 6 chances to fail to get a CTS reply after sending the RTS. If the node still misses the CTS reply after the seventh RTS transmission, the node would give up and discard the corresponding data packet. This is the reason why S-MAC has some data packets lost and does not reach 100% PDR.

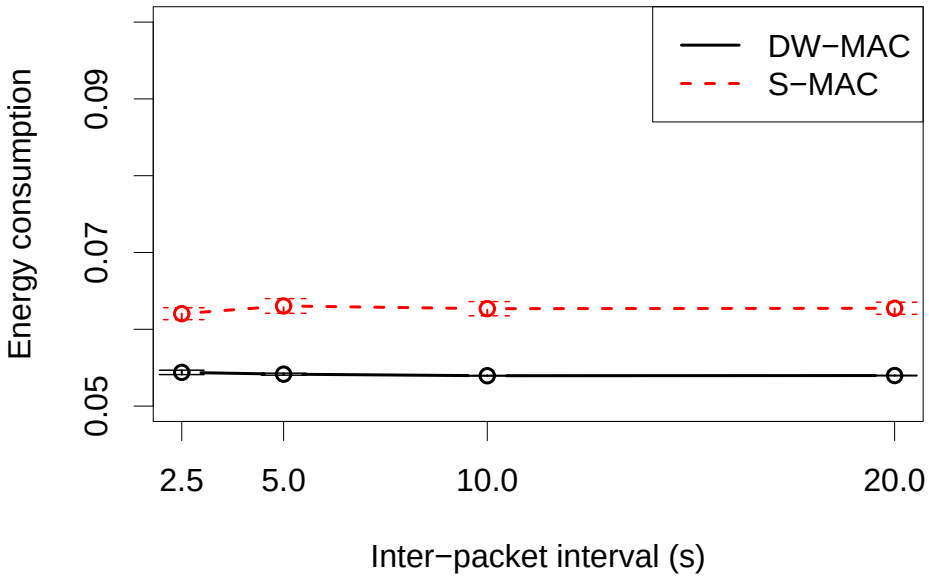


Figure 5.3 : Energy consumption comparison

5.1.3 Energy Consumption

Figure 5.3 shows the energy consumption of S-MAC and DW-MAC. The energy consumption is evaluated as the average proportion of time a node has its radio on to the total experiment time, i.e., 240 seconds. Both DW-MAC and S-MAC have energy consumption greater than 5%, which is the duty cycle. In DW-MAC, all nodes must stay awake in the Sync and Data periods even if there is no control or Sync packet to send or receive. With on-demand wakeup, nodes that have scheduled data packet transactions may turn on their radio for a little while in the Sleep period, so energy consumption is higher than the value of duty cycle. On the other hand, some nodes running S-MAC may turn off their radio before the end of the Data period due to CTS overhearing. Because of adaptive listening, those nodes and the intended senders

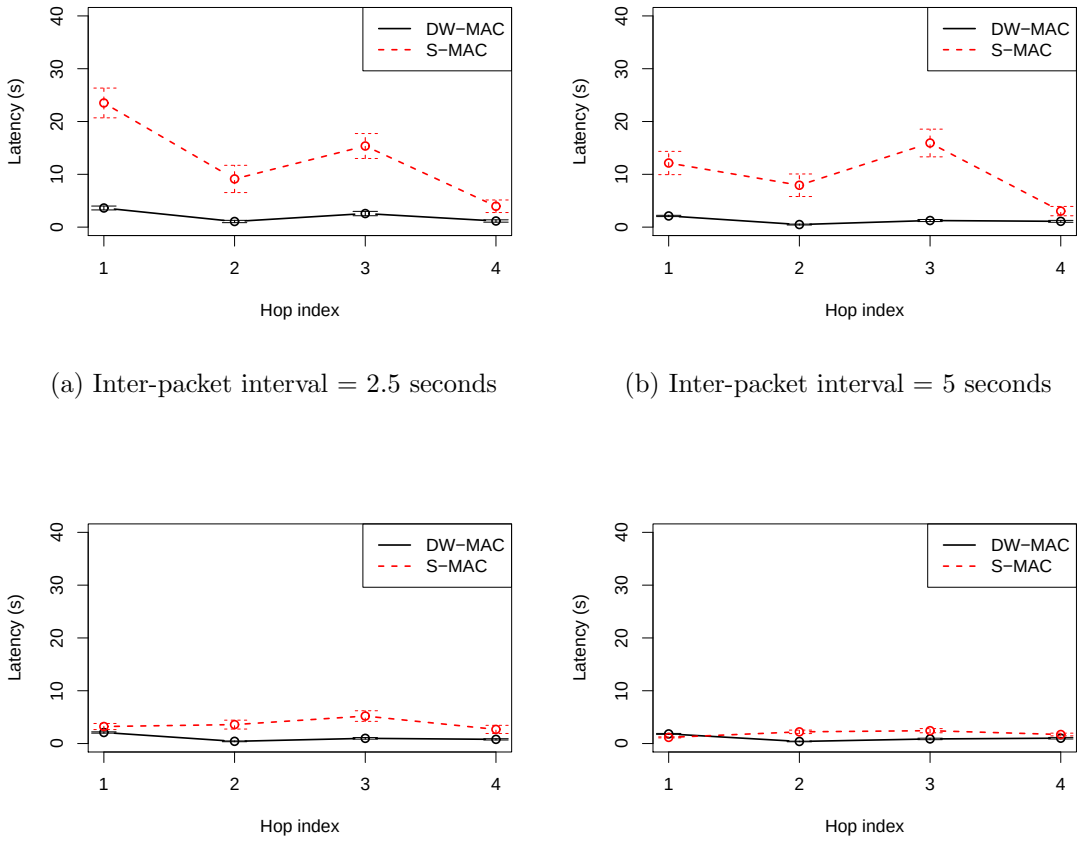
have to wake up later in the Sleep period to start another RTS/CTS/DATA/ACK transmission sequence. However, not all nodes that overhear CTS packets are the correct next-hop nodes, and those nodes are just wasting energy turning on their radio, which is quite different from the on-demand wakeup in DW-MAC. In S-MAC, the energy consumed or wasted due to adaptive listening is higher than the energy saved from CTS overhearing and early sleeping. Therefore, the overall energy consumption of S-MAC is a little higher than that of DW-MAC.

There is no significant difference in energy consumption between the different inter-packet interval rates for both MAC protocols. This is because energy consumption is related only to the amount of time a node's radio is on, not to *when* the radio is on, which affects the results of network latency. Moreover, the total number of data packets in a round of the experiment is identical (10) across all the inter-packet interval settings. Therefore, beside the Data period, the total amount of time nodes spend or waste in the Sleep period for expected data sending and receiving is almost the same, no matter how long the inter-packet interval is. This is why both energy consumption curves in Figure 5.3 are almost flat.

5.2 Detailed Performance Analysis

5.2.1 Per-Hop Latency

In the previous section, I showed the overall performance of DW-MAC and S-MAC at the end-to-end level. In this section, I detail the performance analysis at the per-hop level with separate inter-packet interval settings. Figure 5.4 shows average per-hop latencies of both protocols under different packet intervals, and Figure 5.5 explains various kinds of events that contribute latencies to each hop in the network. A hop



(a) Inter-packet interval = 2.5 seconds

(b) Inter-packet interval = 5 seconds

(c) Inter-packet interval = 10 seconds

(d) Inter-packet interval = 20 seconds

Figure 5.4 : Per-hop latency comparison

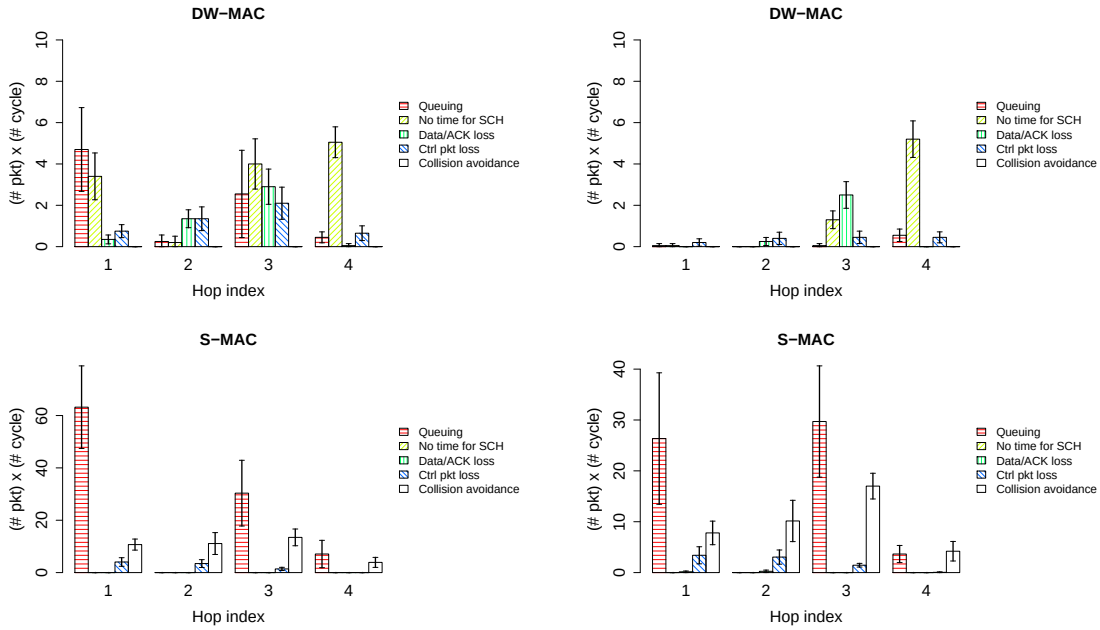
latency is defined as the duration between the time a node first receives a data packet and the time its next hop node first receives that data packet. Hop index i denotes the link between the i^{th} node and the $(i+1)^{th}$ node on each flow. The y-axis of Figure 5.5 is the number of times each kind of event occurs.

All four curves for DW-MAC in Figure 5.4 have a similar pattern. Although they are nearly flat, the average latencies at the first hop are a little larger than those at the

other hops. In the case of 2.5 s inter-packet interval, queuing is the most significant factor that causes higher latency at the first hop, shown in Figure 5.5(a). For the other cases, the time at which new packets are generated at the source nodes mostly falls in the Sleep period. This means those packets cannot be sent from the sources until the next cycle, which creates a significant portion of latencies at the first hop. Once the packets enter the network, the optimized multihop forwarding of DW-MAC helps them travel multiple hops in the same cycle, so the latencies at the following hops are smaller.

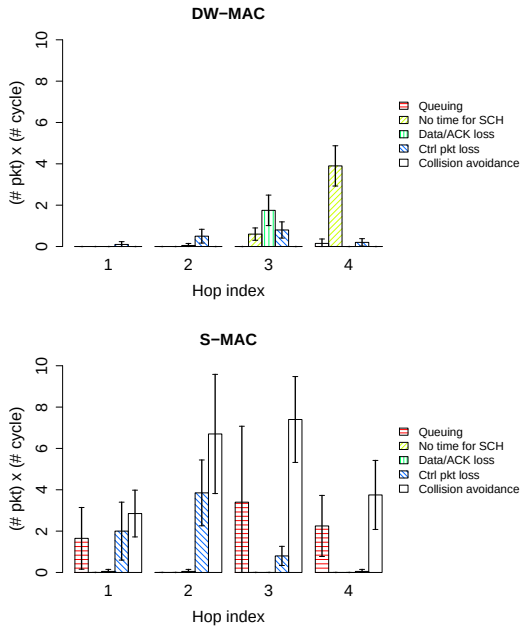
As shown in Figure 5.5, other factors like control packet loss, data or ACK loss, and insufficient time for SCH forwarding (denoted "No time for SCH") also cause latencies. In DW-MAC, control packet loss comes from collisions on SCHs. The intersection node (node 3) of the flows is the busiest point in the network. Under high traffic loads, i.e., short inter-packet intervals, the chance of SCH collision is also higher. Theoretically, proportional mapping and on-demand wakeup in DW-MAC guarantee no data or ACK loss in the Sleep period. In the implementation on real hardware, however, there might still be data packets or ACKs loss. The major reason is unsuccessful timestamping on SCHs. As mentioned in Section 3.3, if a node is very busy and does not have enough time to timestamp a packet, the timestamp of that packet will be invalid. Although I use statistical methods to estimate the sending or receiving time of packets with invalid timestamps, the estimation is not perfect. An incorrect estimation could cause two mapped time period in the Sleep period to interleave with each other, which may result in data or ACK collision. Besides, there are still small chances that data packets or acknowledgments cannot be correctly decoded by the receiver due to interference from background noise or other reasons.

A node that wants to send or forward an SCH may not have enough time near

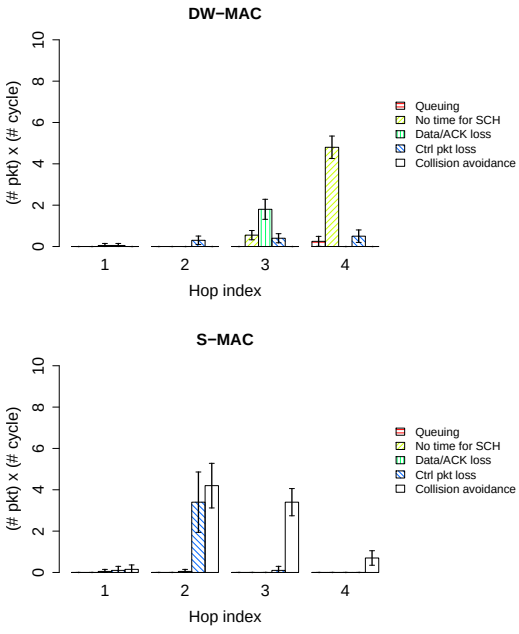


(a) Inter-packet interval = 2.5 seconds

(b) Inter-packet interval = 5 seconds



(c) Inter-packet interval = 10 seconds



(d) Inter-packet interval = 20 seconds

Figure 5.5 : Detailed per-hop latency comparison

the end of the Data period. By estimating the transmission and processing delays of an SCH, the node may decide to defer the SCH forwarding to the next cycle due to lack of time. This is denoted "No time for SCH" in the figure, and this kind of event occurs in DW-MAC only. In the experiments, this kind of event happens mostly at the last hop. This is because when an SCH goes to the fourth node of the flow (node 3), there might not be enough time for it to go to the next hop (node 4 or 8), and then the fourth node has to buffer this SCH and send it in next cycles after getting the corresponding data packet. Besides, under high traffic loads (Figure 5.5(a)), source nodes may also have insufficient time to send SCHs. The reason is that DW-MAC allows a node to send as many SCHs as it can in a Data period. Each time before an SCH is generated, a data packet is removed from the head of the queue. Under high traffic loads, a node could have multiple data packets queued in the memory and send multiple SCHs to make multiple reservations in the Sleep period. If the source node has sent a few SCHs in current Data period but does not have enough time to send the newly generated SCH, the SCH has to be sent in a later cycle. The delay on the SCH is classified as "No time for SCH", not queuing delay. For any of the reasons mentioned above, a node running DW-MAC may have to defer its SCH or data packet transmission to the next cycle or a time later in current cycle, and that creates latencies.

As expected, per-hop latencies in S-MAC are larger than those in DW-MAC, especially under high traffic loads. In the cases of 2.5 s and 5 s inter-packet intervals, there are two peaks in the curves at the first and the third hops. Similar to DW-MAC, latencies at the first hop come from packet queuing (Figure 5.5). With adaptive listening enabled, packets are likely to travel two hops in a cycle. In this situation, the packets do not have to wait for a cycle time at the second hop. However, they

have to wait and queue at the third node on the path, which results in another peak at the third hop in the curves. Queuing contributes the most to latencies when the inter-packet intervals are 2.5 and 5 seconds. In the case of 5 s inter-packet interval, packets are queued for longer time at the third hop than at the first hop. This is because the third hop is near the intersection of the two flows, and contention is more severe there. In contrast, due to the relative lower traffic loads, the queuing delay at the first hop is not as high as in the case of 2.5 s inter-packet interval.

As the inter-packet interval increases, other factors take the place of queuing and dominate the latencies. The major factor among these is the collision avoidance mechanism. Although collision avoidance protects data packets and ACKs from collision, it also forces a node that overhears RTS or CTS packets from others to defer or cancel its own RTS or CTS transmission, which brings latencies. An interesting result is that the pattern of the number of collision avoidance events in the case of 20 s inter-packet interval is quite different to the other three cases. In the 20 s case, the second hop has the most collision avoidance events, while in other packet intervals, it turns out to be the third hop. The key point is the behavioral difference at the second nodes, i.e., nodes 1 and 6, on the two paths. In the case of 20 s inter-packet interval, the time a data packet travels in the network is less than the inter-packet interval (Figure 5.1), and at most one data packet is traveling on each path at any time. Additionally, nodes 1 and 6 cannot hear each other in the experiment environment, but node 7 is in the transmission range of node 1, and node 2 can hear node 6. After nodes 0 and 5 send an RTS to nodes 1 and 6, respectively, in the same Data period, both nodes 1 and 6 will reply with CTS packets and both nodes 0 and 5 will receive the CTS packets successfully. No collision avoidance happens so far. After that, however, at most one of nodes 1 and 6 can win the channel competition and

successfully performs adaptive listening and forward the data packet to the next hop. The other node will defer its RTS/CTS transaction due to RTS or CTS overhearing, which results in a collision avoidance event at the second hop in Figure 5.5(d). In contrast, the winner node will do the next RTS/CTS transaction at the third hop in the next cycle. Because the loser node has failed an RTS/CTS transaction, its contention window will be doubled, which further decreases their chance of winning channel competition in the next cycle. The winner node keeps forwarding its data packet to the destination, and after that, the loser node also delivers the data packet to the destination after a while. As a result, the number of collision avoidance events at the second hop is larger than that at the third hop.

Figure 5.6 demonstrates this effect. When the inter-packet interval is less than or equal to 10 seconds, the chance that packets are successfully forwarded at the second and fourth hops by adaptive listening is over 60%. As the inter-packet interval increases, more often the second hop nodes have the RTS or CTS collided with other nodes after waking up adaptively, which causes the adaptive listening to be *shifted* to the third hop. In the case of 20 s inter-packet interval, almost half of the packets have their adaptive listening shifted, which means adaptive listening interleaves with each other at different hops on different paths.

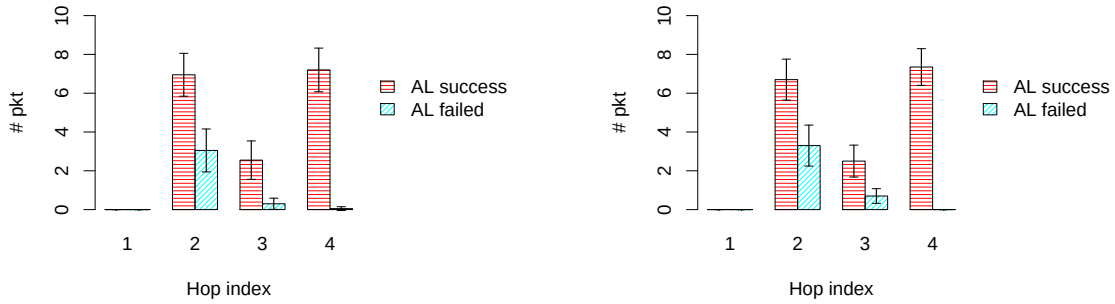
5.2.2 Fairness

Figures 5.7 through 5.10 show detailed data packet transmission and reception timelines for an example run of the experiment using S-MAC. In the timeline of a node, each vertical segment denotes a starting point of a new cycle. The numbers below each timeline graph show the indices of cycles after the first packet transmission in the network. Each cycle between two consecutive vertical segments can be divided

into two parts: the top-left part and the bottom-right part. A block at the top-left part represents an event that originates in the Data period, whereas a block at the bottom-right part represents an event that comes from adaptive listening. In particular, these graphs show where and when adaptive listening happens. The timeline of node 3, which is the node at the intersection, is shown twice in each figure so that it is easier to see the behavior of the node on both flows. In the cases of 2.5 s to 10 s inter-packet intervals, the packet transmission finishes before the fortieth cycle arrives, due to their shorter inter-packet intervals compared with the 20 s case.

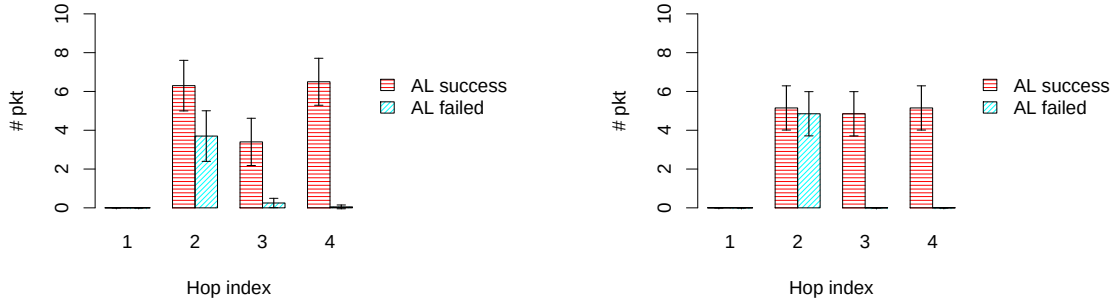
Another interesting thing that is shown in these graphs is the unfairness between the flows in S-MAC. The unfairness comes from the design of the contention window and random backoff mechanism before RTS transmission. As mentioned in Section 2, in both S-MAC and DW-MAC, before a node sends an RTS or SCH request, the node first does a random backoff within the contention window. If the node does not get a CTS or SCH reply after a timeout occurs, the node will double its contention window size. In DW-MAC, this is fine because the node that loses in the channel competition can try again later in the same Data period after other nodes finish their SCH exchanges. Therefore, the unfairness does not happen in DW-MAC. In S-MAC, however, the design of the contention window and random backoff mechanism may cause the losers difficult in getting back the channel. This is because each node has only one chance to try the RTS/CTS transaction in the Data period, and winners have smaller contention window sizes and are more likely to start their RTS/CTS transaction earlier. If the network is very busy, losers can only wait until winners finish their traffic, as shown in Figures 5.7 and 5.8. That also explains why S-MAC has larger error bars in the case of 2.5 s and 5 s inter-packet intervals in the end-to-end latency graph (Figure 5.1). On the other hand, unfairness does not exist in the

case of 20 s inter-packet interval. The nodes on each flow can deliver a data packet to the destination before the next packet is generated at the source, and the packet transmission on the two flows interleaves with each other. Therefore, the differences of end-to-end latencies between the two flows are relatively small.



(a) Inter-packet interval = 2.5 seconds

(b) Inter-packet interval = 5 seconds



(c) Inter-packet interval = 10 seconds

(d) Inter-packet interval = 20 seconds

Figure 5.6 : The number of adaptive listening happens at each hop

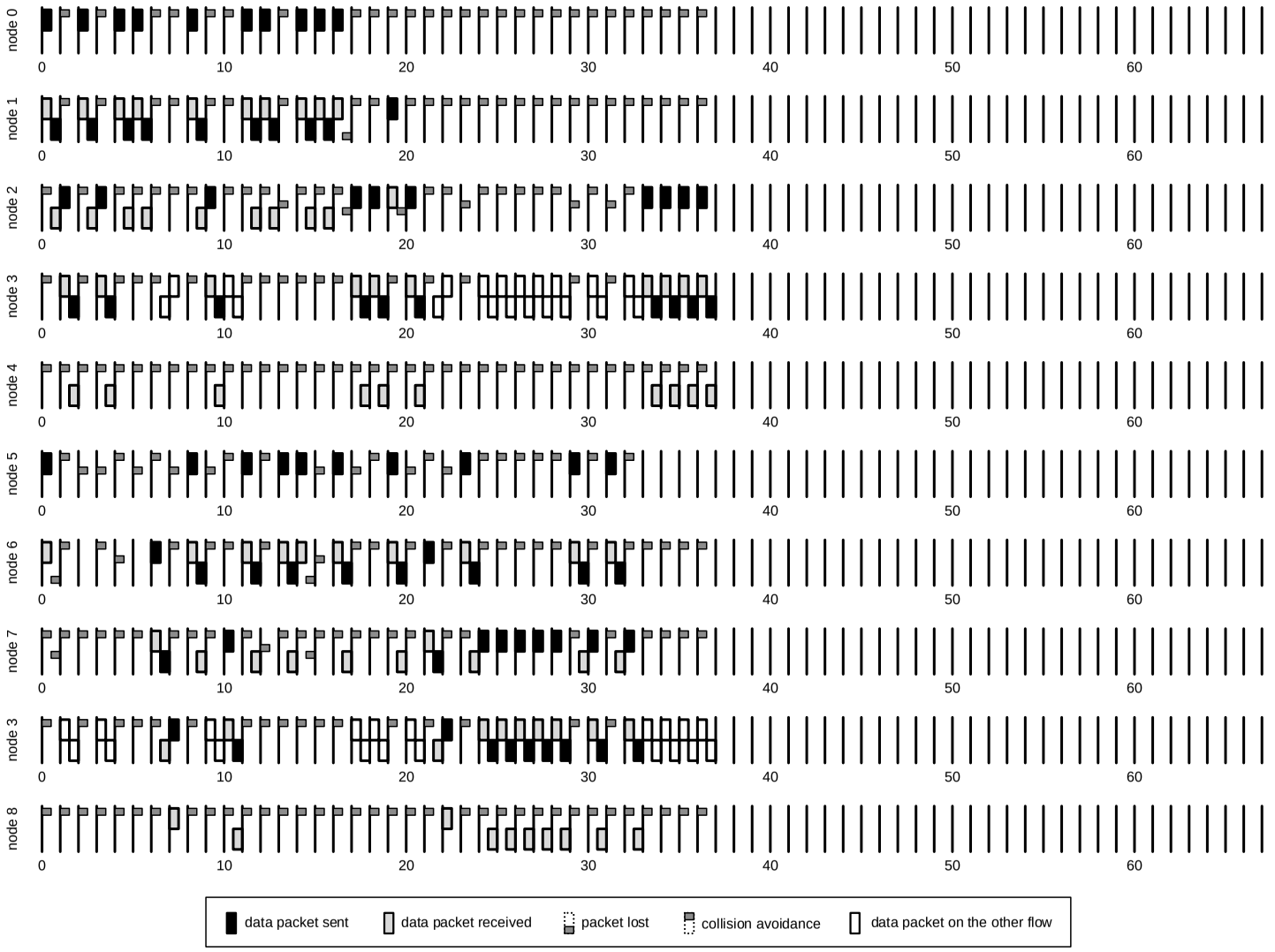


Figure 5.7 : Timeline of inter-packet interval = 2.5 seconds in S-MAC

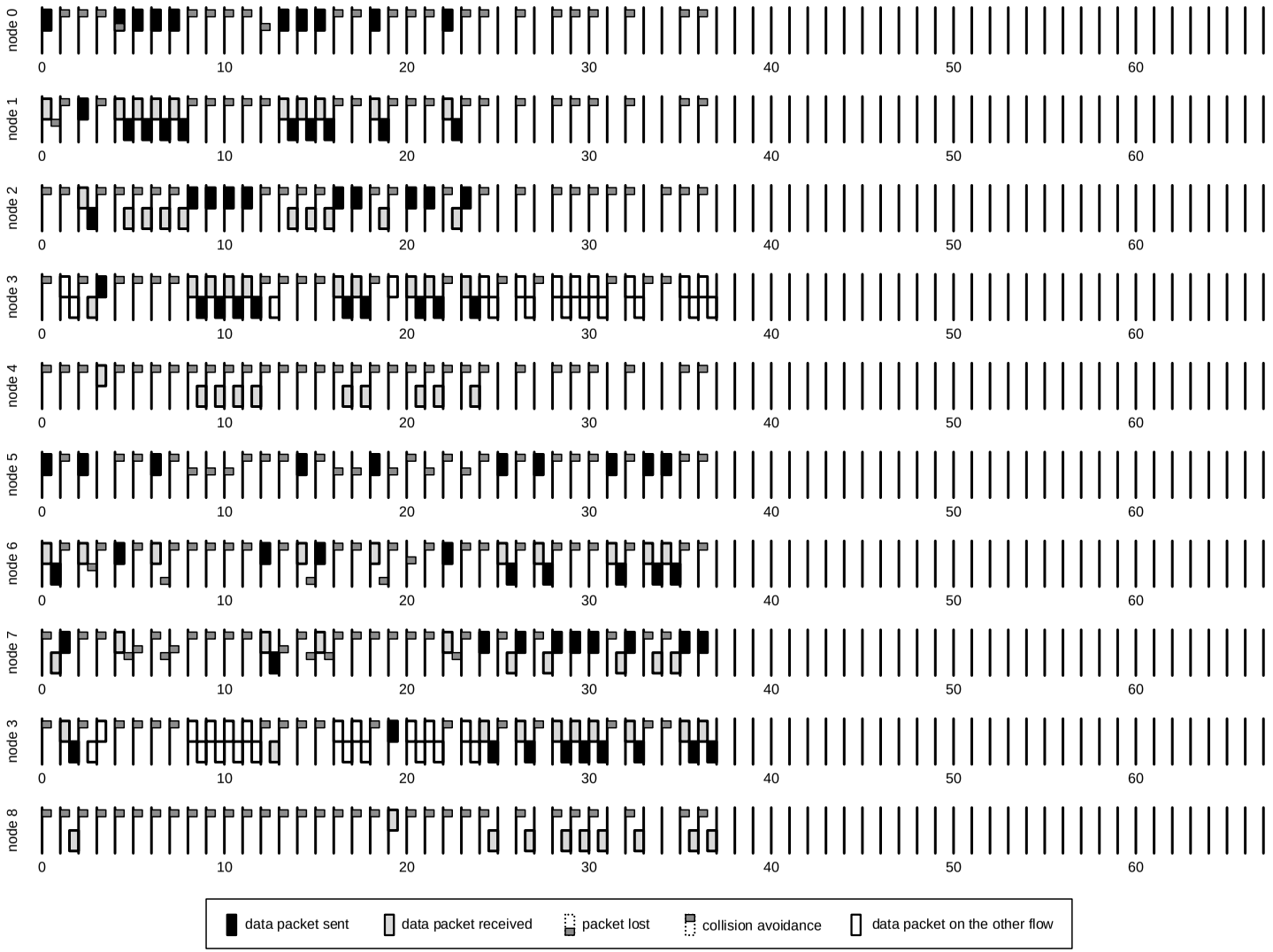


Figure 5.8 : Timeline of inter-packet interval = 5 seconds in S-MAC

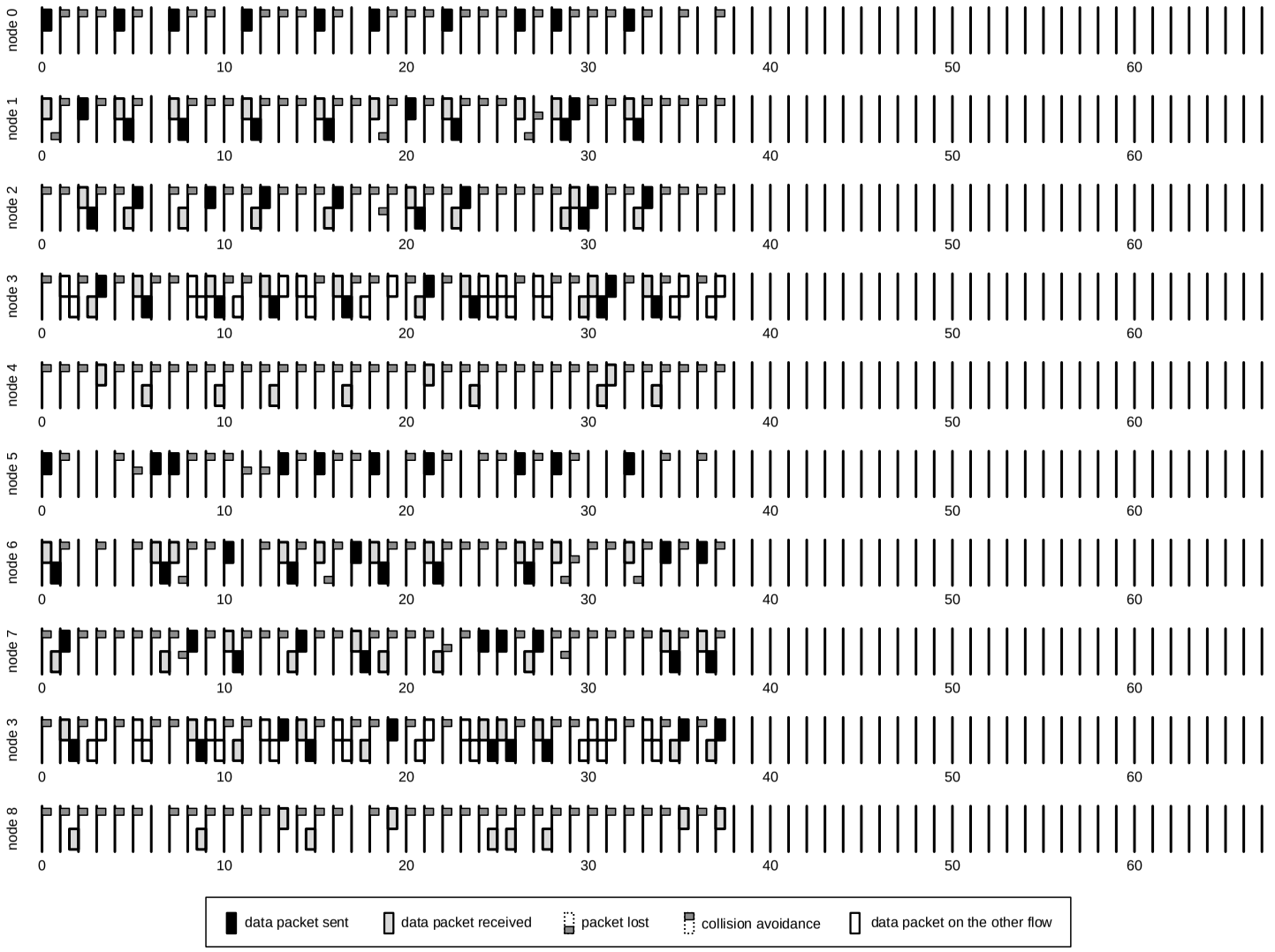


Figure 5.9 : Timeline of inter-packet interval = 10 seconds in S-MAC

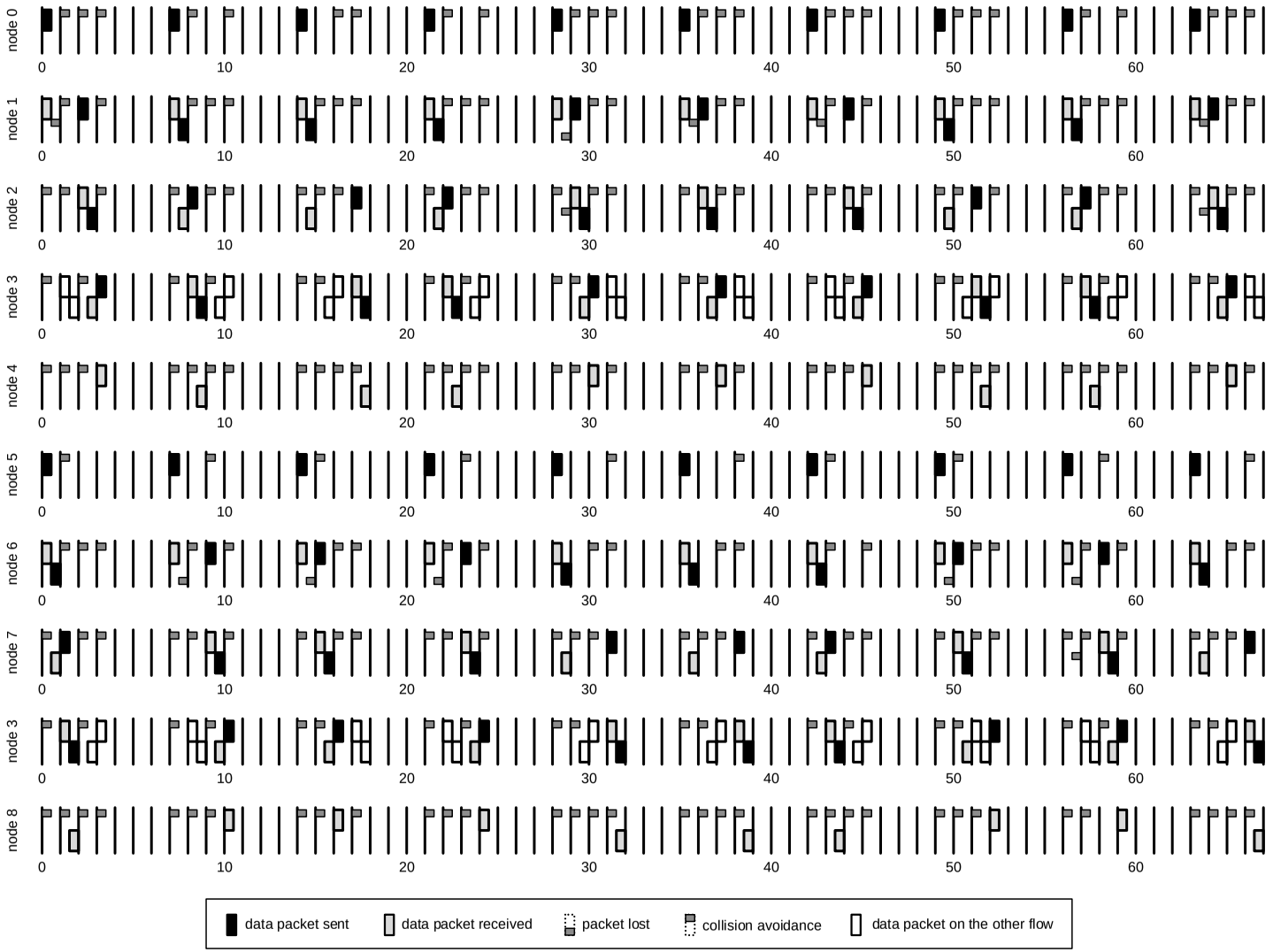


Figure 5.10 : Timeline of inter-packet interval = 20 seconds in S-MAC

Chapter 6

Simulation Results

6.1 S-MAC and DW-MAC Comparison

This chapter presents the simulation results of S-MAC and DW-MAC. For each pair of protocol and inter-packet interval setting, the simulation is run 100 times. The simulation results are similar to the results presented in Chapter 5 for the experiments using the physical implementation on MICAz motes. Figures 6.1 through 6.3 show the results of various performance metrics at the end-to-end level, and Figures 6.4 through 6.6 show detailed results at the per-hop level. The results of packet delivery ratio are quite similar between the simulation and the experiments, which means the simulated environment is close to the experiment environment. The energy consumption and end-to-end latency of DW-MAC in the simulation is a little lower than that in the experiment. This is because the simulation does not include the packet timestamping issues mentioned in Section 3.3. While invalid packet timestamps may cause collision in the Sleep period, as described in Chapter 5, packet timestamping can be considered perfect in the simulation. Without the collision on data packets or acknowledgements, a packet can be delivered to the destination in less time and fewer trials, which saves energy and decrease network latencies. The differences of the number of data or ACK collision events are also shown in Figures 5.5 and 6.5.

S-MAC has similar end-to-end latencies and energy consumption between the simulation and experiments; however, the curves of per-hop latencies are not as close.

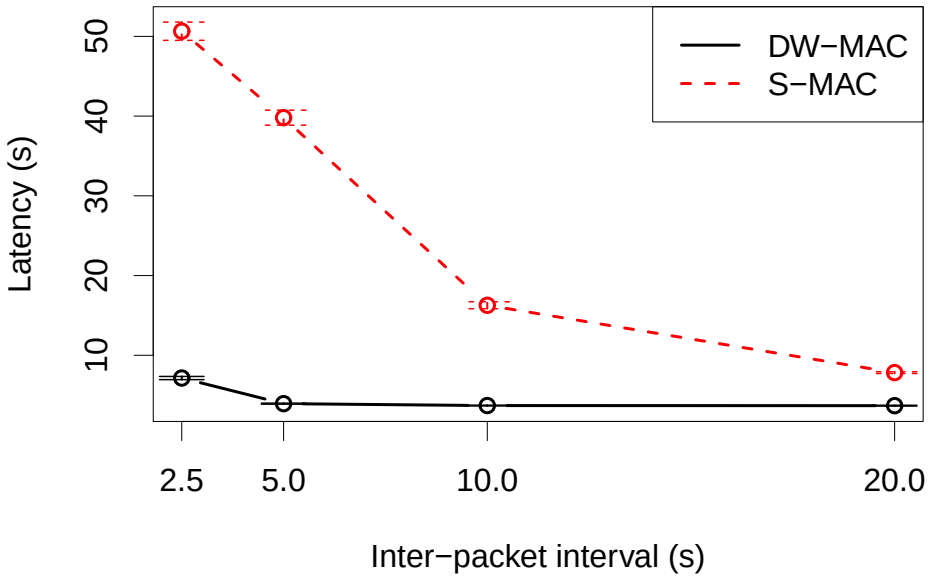


Figure 6.1 : End-to-end latency comparison (simulation)

As shown in Figures 5.4 and 6.4, average latencies at the first hop are higher whereas those at the third hop are lower in the simulation. This difference is most pronounced in the cases of 2.5 s and 5 s inter-packet intervals. In the case of 10 s inter-packet interval, similar things also happen at the first hop. This is because the behavior of the simulated radio is not exactly the same as the real radio behavior. In the simulation, more packets on one of the two flows are blocked at the first hop than in the experiments before the other flow delivers its packets to the destination, because the other flow wins the channel competition at the second hop when the first packets are sent by the sources. Once the flow wins at the beginning, packets on that flow are forwarded to the center of the network earlier than on the losing flow. The RTS/CTS transaction the winning flow performs in the center of the network is

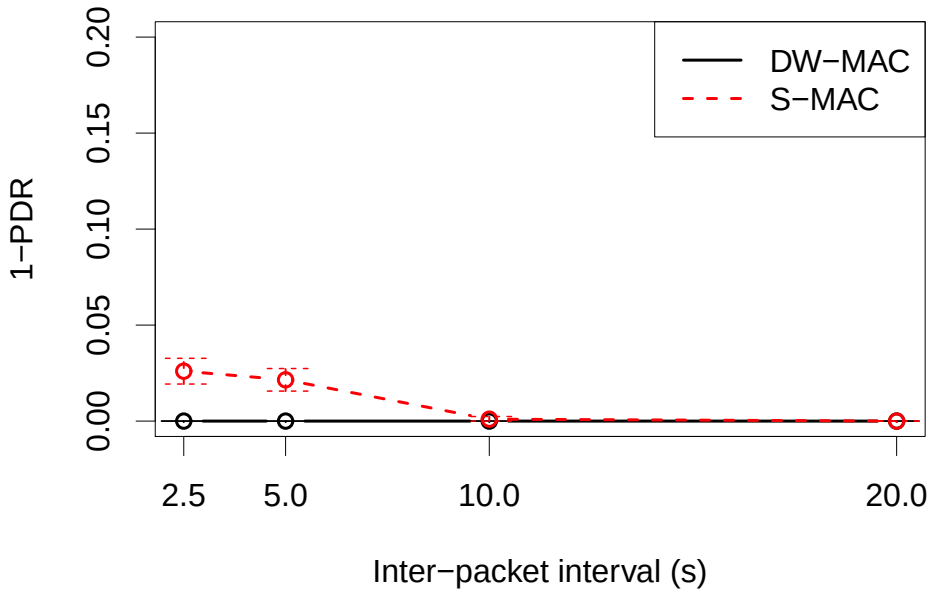


Figure 6.2 : Packet delivery ratio comparison (simulation)

overheard by the second hop on the losing flow, and the losing flow has to defer its RTS/CTS transaction based on the collision avoidance mechanism. Under high traffic loads, new packets keep being injected to the network on the winning flow, and the overhearing keeps occurring on the losing flow. Therefore, it is very difficult for the losing flow to get a chance to forward packets to the center of the network until the winning flow finishes all its packet transmissions. In the experiments, because the radio behavior is more dynamic, sometimes the losing flow does not overhear packets from the winning flow, and then the losing flow gets chances to forward its packets. This is the reason why more packets are blocked at the first hop in the simulation than in the experiments. Since more packets are blocked at the first hop, fewer packets stay in the middle of the network, so the number of packets queued at the third hop

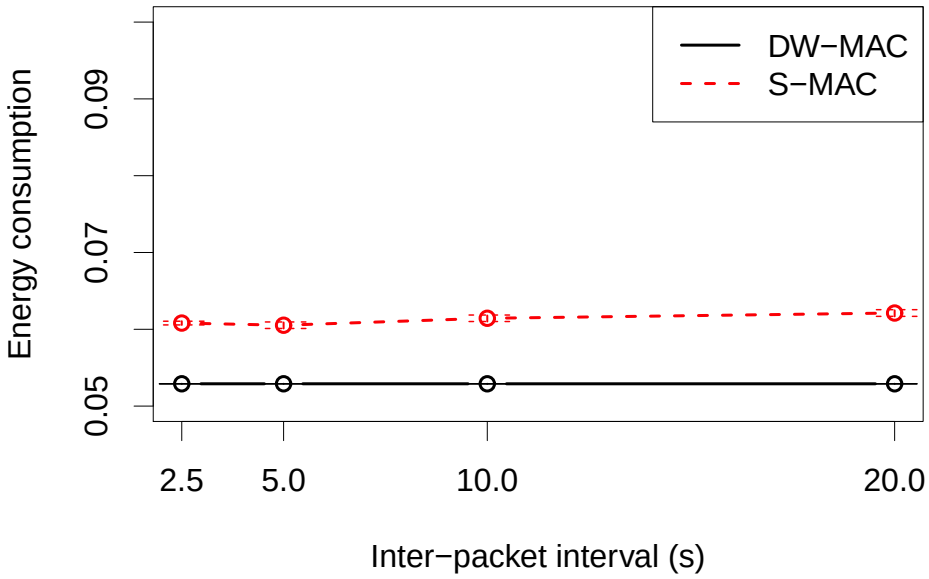


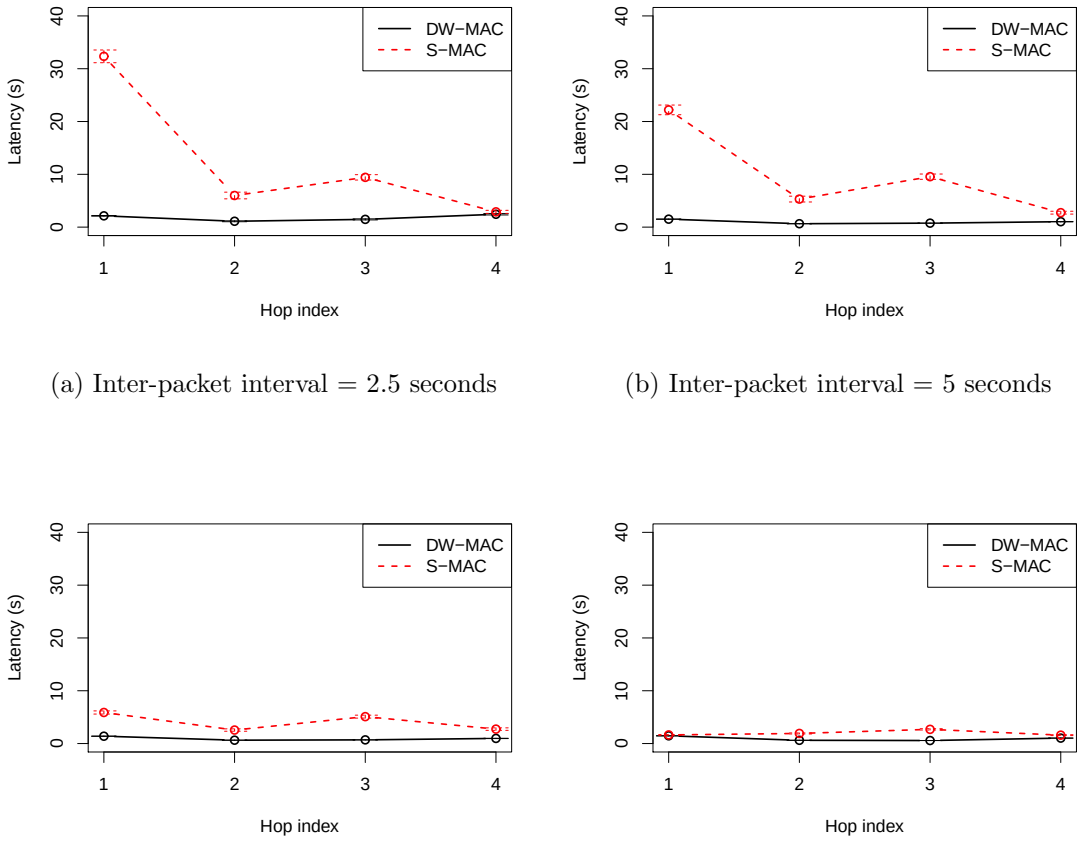
Figure 6.3 : Energy consumption comparison (simulation)

is smaller. This phenomena is also visible in the queuing delay differences between Figures 5.5 and 6.5.

Other minor differences in detailed per-hop latency (shown in Figures 5.5 and 6.5) can also be related to the difference of the simulated radio and the real radio. In spite of those miner differences, the simulation results fit the results in the experiments quite well, which serves to help validate the simulation and experimental results.

6.2 Comparison with Original DW-MAC Simulation Results

Both S-MAC and DW-MAC were also simulated in *ns*-2 in the previous DW-MAC work [16]. To compare the simulation results in this thesis with those in the previous work, I obtained the simulation code of the original DW-MAC work and ran it on



(a) Inter-packet interval = 2.5 seconds

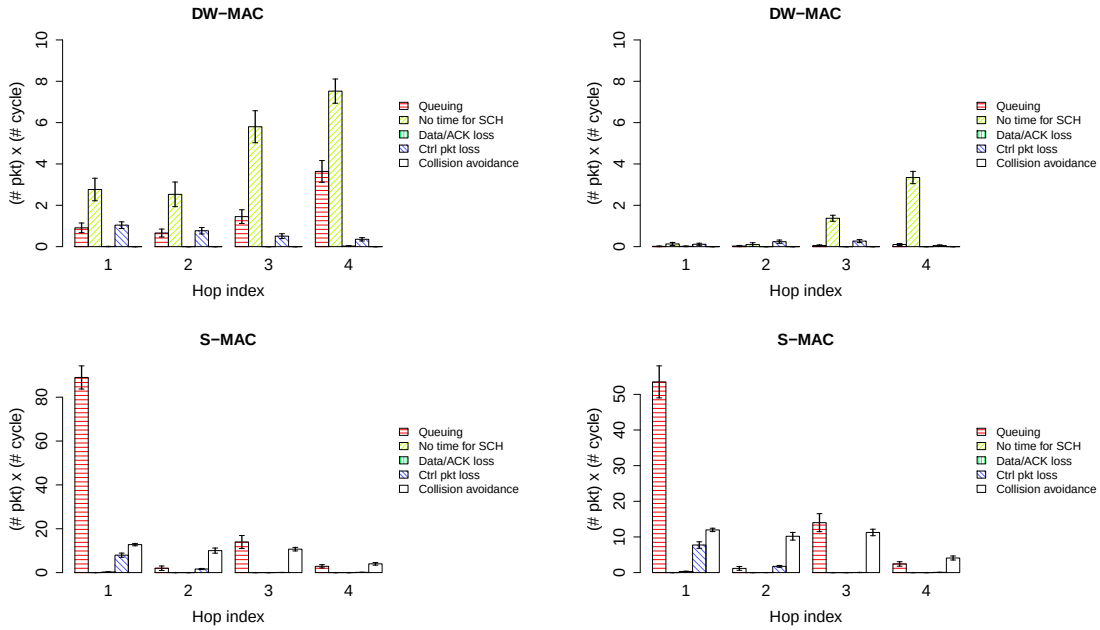
(b) Inter-packet interval = 5 seconds

(c) Inter-packet interval = 10 seconds

(d) Inter-packet interval = 20 seconds

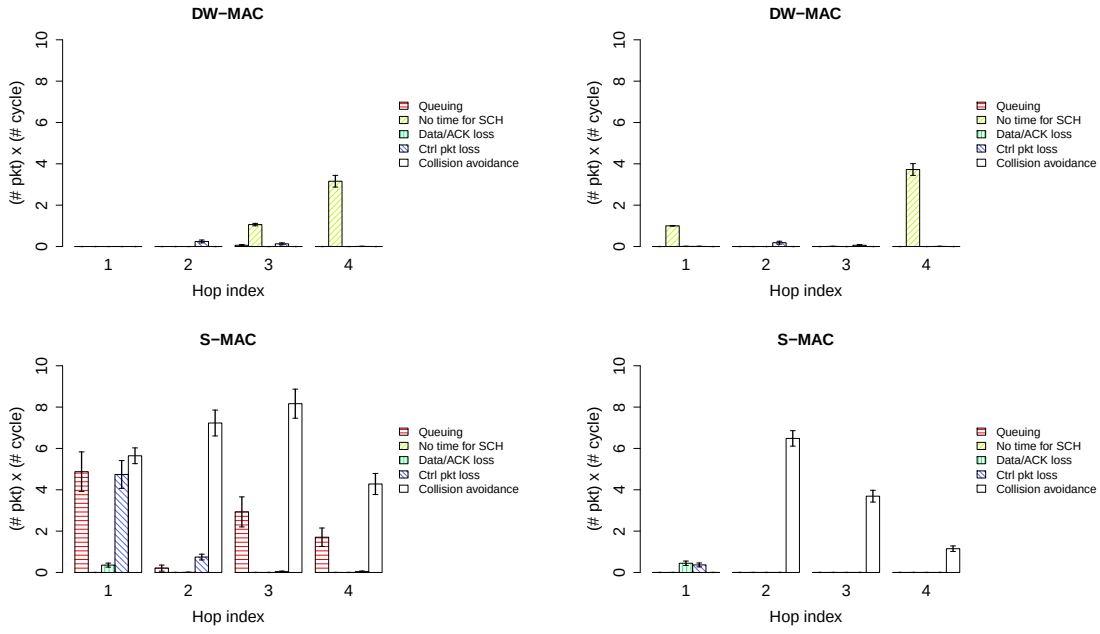
Figure 6.4 : Per-hop latency comparison (simulation)

the network topology (Figure 4.2) used in this thesis. Since the original simulation code was specifically designed for simulating the Crossbow Mica2 [1] motes, I kept some hardware-related settings unchanged in the original code, such as the maximum bandwidth of the radio (20 kbps) and the timeout value of SCH and CTS packets (14 ms). Other network parameters (Tables 4.2 and 4.3) and traffic settings used in this thesis were applied to this original code.



(a) Inter-packet interval = 2.5 seconds

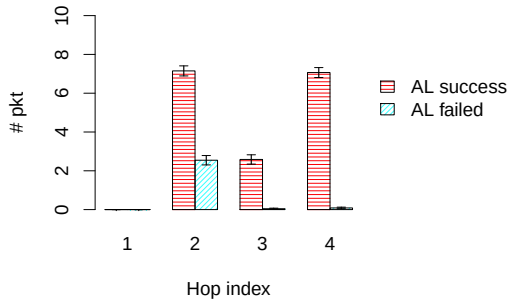
(b) Inter-packet interval = 5 seconds



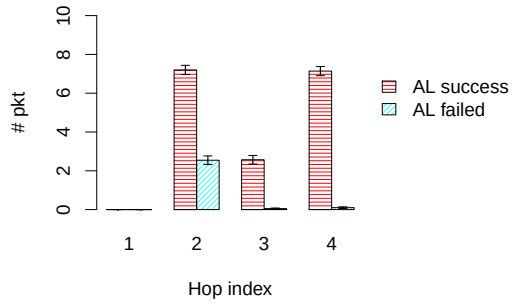
(c) Inter-packet interval = 10 seconds

(d) Inter-packet interval = 20 seconds

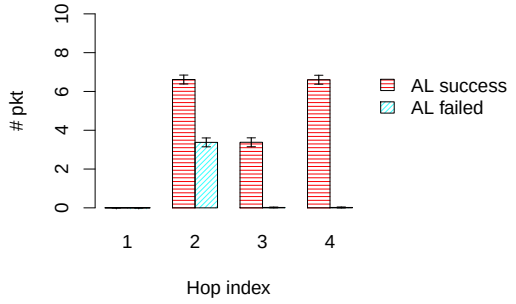
Figure 6.5 : Detailed per-hop latency comparison (simulation)



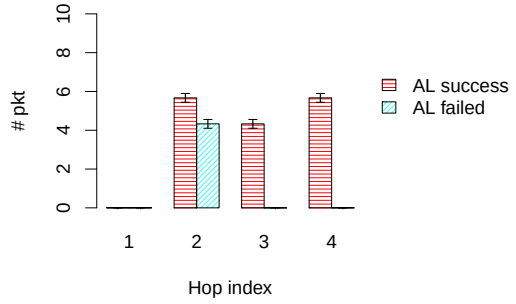
(a) Inter-packet interval = 2.5 seconds



(b) Inter-packet interval = 5 seconds



(c) Inter-packet interval = 10 seconds



(d) Inter-packet interval = 20 seconds

Figure 6.6 : The number of adaptive listening happens at each hop (simulation)

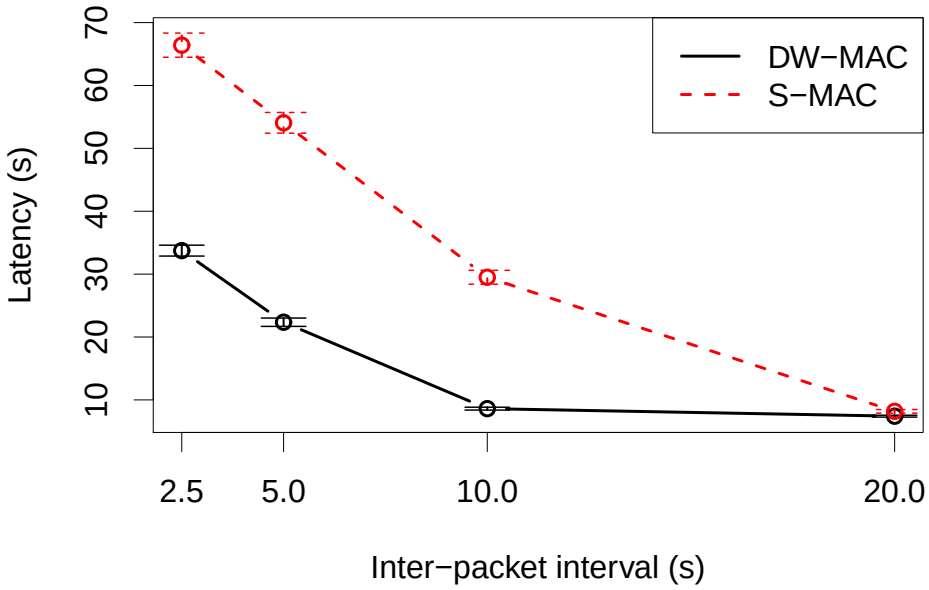


Figure 6.7 : End-to-end latencies from original DW-MAC and S-MAC simulation code

Figure 6.7 shows the results of end-to-end latency in the original simulations. Although DW-MAC still outperforms S-MAC in terms of end-to-end latency, both MAC protocols have larger end-to-end latencies under all inter-packet interval settings, compared with the simulation results in Figure 6.1. The differences are even more pronounced under smaller inter-packet intervals. When the inter-packet interval is 2.5 seconds, the difference in end-to-end latencies for DW-MAC is around 25 seconds, and for S-MAC, the difference is around 15 seconds. One reason for the larger end-to-end latencies in original simulations is the low radio bandwidth. The low bandwidth not only increases the packet transmission time, it also decreases the number of hops DW-MAC can deliver SCHs in a single cycle, which increases the

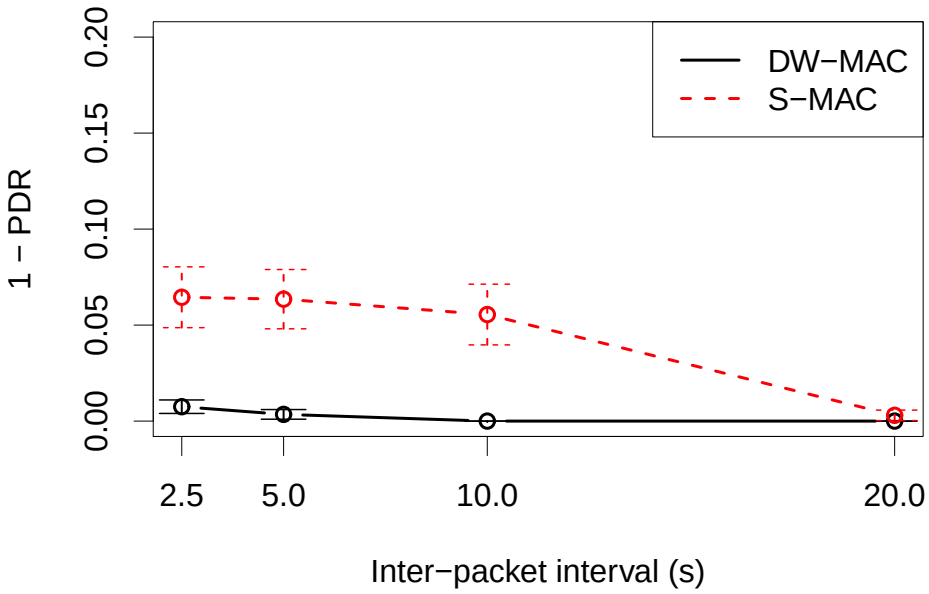


Figure 6.8 : Packet delivery ratio from original DW-MAC and S-MAC simulation code

amount of time data packets stay in the network. Another important reason for the larger end-to-end latencies is that the original simulation code uses simpler radio and network topology settings. In the original simulation, a node is assumed unable to hear from two hops or further distance away, whereas my simulation code can be applied to a more complex and difficult network environment, closer to the real network in the experiments presented in Chapter 4. The limits of this design cause the original simulation code to be sometimes unable to maintain correct states in the nodes and to handle incoming or outgoing control packets properly, resulting in unnecessary control packet discarding. This reflects higher packet loss rates on both MAC protocols, as shown in Figure 6.8, compared with the results in my simulations

(Figure 6.2). As a conclusion, my version of the simulation simulates the sensor network in a more realistic way than the original version, which is also demonstrated through the experimental results in Chapter 5.

Chapter 7

Related Work

Many duty-cycling MAC protocols have been proposed for wireless sensor networks. Some of them were synchronous duty-cycling protocols while others were asynchronous in their design. The nature of synchronous duty-cycling protocols makes all nodes stay awake at the same time, and nodes can easily broadcast messages to their neighbors or forward packets to multiple hops away within a short period of time. S-MAC [22] was one of the original synchronous duty-cycling MAC protocols in wireless sensor networks. Many MAC protocols were developed based on the design of S-MAC.

RMAC [9] divides a cycle into the Sync, Data, and Sleep periods, as with S-MAC. Instead of using the RTS/CTS collision avoidance mechanism, however, RMAC introduced another control packet, called PION, to reduce latency in multihop packet forwarding. Similar to the SCH design in DW-MAC, nodes send and forward PION frames in the Data period to inform downstream nodes about upcoming data packet transmission. Nodes that receive a PION frame will wake up accordingly in the Sleep period for receiving and forwarding data packets. In RMAC, the time a node wakes up in the Sleep period is calculated based on the duration information in the PION frame, whereas DW-MAC does not require any timing information in the SCH. Another difference between RMAC and DW-MAC is that RMAC does not use the proportional mapping mechanism. Data transmission always starts at the beginning of the Sleep period, which may cause collision between two hidden nodes that have succeeded in their scheduling through PION frames in the Data period. PRMAC [8]

and BulkMAC [7] were later proposed to enhance the performance of RMAC by allowing multiple data packets to be delivered in one cycle via one PION frame. However, none of them solved the problem of data packet collision in RMAC. Additionally, RMAC, PRMAC, and BulkMAC evaluated their performance via simulation only. No physical implementation was done in their work. In this thesis, I have implemented DW-MAC, which solves the problem of data packet collision in RMAC. I also evaluate the performance of DW-MAC through both simulation and real-world experiments.

T-MAC [20] inherits the RTS/CTS collision avoidance mechanism in S-MAC. Unlike S-MAC, T-MAC has a flexible length of the Data period, and the length is adaptively determined by the network loads. A node running T-MAC may shorten its Data period and sleep earlier if no traffic is around the node. Under high traffic, the node can also extend its Data period to accommodate more RTS/CTS transactions. With the flexible Data period design, T-MAC can preserve more energy. However, T-MAC otherwise has the same shortcomings as S-MAC. For example, T-MAC can only deliver a packet at most two hops away in a cycle. Nodes that are not the intended next hop but overhear CTS packets may still remain awake, which increases energy consumption. Dam *et al.* [20] implemented T-MAC on the EYES hardware for energy consumption testing. They mentioned the clock drift problem, and they used the Sync messages to correct time differences among nodes, similar to what I did in this thesis. However, beside the time correction, no other implementation issues were mentioned or discussed in their paper. On the other hand, in this thesis, I mainly focus on the implementation issues in sensor networks, including the problems of packet timestamping, slow CPU speed, and imperfect timing, and I also provide solutions to those problems.

Unlike synchronous duty-cycling MAC protocols, nodes running asynchronous duty-cycling MAC protocols may not start their cycles simultaneously, and thus multihop time synchronization is not required. An asynchronous duty-cycling MAC protocol is either *sender-initiated* or *receiver-initiated*. Examples of sender-initiated protocols are B-MAC [15], X-MAC [6], and WiseMAC [10]. In those protocols, before transmitting data packets, the sender sends one or more preambles to inform the receiver about the upcoming data packet. The preambles may occupy the wireless medium for a long time before the data packets are delivered. Under heavy traffic loads, the protocols may become less energy efficient. Receiver-initiated protocols, such as RI-MAC [17], PW-MAC [19], and EM-MAC [18], do not have this problem. In RI-MAC, each receiving node periodically wakes up and broadcast a beacon. Any node that receives the beacon and has data packets to send to that node can transmit the data packet after receiving the beacon. This design avoids long preambles at the sender side and thus decreases traffic loads in the wireless medium. PW-MAC enhanced the energy efficiency of RI-MAC by minimizing the time a sender spends on idle listening and waiting, through predictive wakeup. In PW-MAC, each receiving node has a pseudo-random schedule and wakes up and goes to sleep based on that schedule. A node that wants to communicate with a receiving node learns of the schedule of the receiving node and wakes up a bit earlier than the receiving node. EM-MAC further increases wireless channel utilization via multi-channel support. Different sender and receiver pairs may choose different channels for their data delivery. The multi-channel design also makes EM-MAC more resilient to problems such as ZigBee jamming attacks or Wi-Fi interference.

RI-MAC, PW-MAC, and EM-MAC were all implemented on TinyOS on MICAz motes. Both PW-MAC and EM-MAC pointed out the problem of clock drift and

proposed solutions. PW-MAC introduced an on-demand prediction error correction mechanism to correct timing errors when the error between two nodes is larger than a threshold. A sender does advanced wakeup to avoid missing the beacon from the receiver. The sender also includes in its data packet a request to the receiver for updating their wakeup schedule after detecting that the timing error is large than a threshold. On the other hand, EM-MAC used the exponential chase algorithm to re-rendezvous a sender and a receiver. After the sender misses the receiver's schedule, the sender keeps doubling its wakeup advance time to "chase" the receiver until getting the receiver's beacon. Although both timing correction mechanisms in PW-MAC and EM-MAC are energy efficient, they are only suitable for asynchronous duty-cycling protocols. For synchronous duty-cycling protocols such as DW-MAC with critical accurate timing requirements, time has to be frequently synchronized and the timing error has to be very small. Regarding the timing requirement of DW-MAC, I designed the *guard time* mechanism to complete the insufficiency of the time synchronization mechanism. Although the RI-MAC paper described the concepts of its protocol design on TinyOS, no other implementation problems were mentioned or discussed in the paper. In contrast, this thesis points out the problems existing in protocol implementation from various aspects of views and provides solutions to them, which helps future MAC protocol design and implementation.

Network time synchronization is an important part in the implementation of synchronous duty-cycling MAC protocols. Much work has been proposed for synchronizing time over wireless sensor networks. The Intel Research Lab [14] designed a power-efficient Delay Measurement Time Synchronization (DMTS) technique applicable for both single-hop and multi-hop wireless sensor networks. In DMTS, a leader is chosen among the nodes, and the leader broadcasts its local clock value to other

nodes. For single-hop synchronization, all the nodes that hear the broadcast message estimate the propagation delay from the leader and synchronize their time with the leader. The scenario can be extended to a multi-hop version by relaying a broadcast message to downstream nodes along a broadcast tree. Similar to this thesis, DMTS is implemented on TinyOS, and the synchronization accuracy is one clock tick. Although the time synchronization mechanism used in this thesis is similar to the single-hop version of DMTS, the time value the synchronizer broadcasts is not its local clock time but the duration to the beginning of the Data period. Therefore, all nodes can start their cycles simultaneously and still keep their own local clock time, which simplifies delay measurement inside each node itself. In addition, considering that the uncertainty of channel quality may cause Sync packet loss, the synchronizer consecutively broadcasts many Sync packets to the network in the Sync period to increase the chance that each node can receive at least one Sync packet in the cycle.

Maróti *et al.* [13] proposed the Flooding Time Synchronization Protocol (FTSP), which is also a broadcast-based synchronization protocol. The basic concept of FTSP is quite similar to that of DMTS. FTSP further used linear regression to predict the clock drift rate between two nodes and extrapolate a correct clock value after the last run of time synchronization. This mechanism avoided the need for frequent broadcasting for time synchronization, however, it was tested on the Mica2 motes only. For other existing hardware or future new motes, linear regression may not be able to fit the clock drift rate well. In addition, clock rates can also be influenced by the environment. Different environments may lead to different extents of clock drift. It becomes a tradeoff between time accuracy and synchronization overhead. Furthermore, it is effectively impossible to perfectly synchronize time over the network. As long as there is an error, the problem in DW-MAC mentioned in Section 3.2.1 still

exists. Since the design of guard time can solve such problem, to simplify the implementation complexity, it is acceptable and sufficient in this thesis to synchronize the time via Sync packet broadcast every other cycle.

Chapter 8

Conclusions and Future Work

In this thesis, I have revealed the problems that were not addressed in the previous DW-MAC paper, where only simulation is conducted in performance evaluation. I have proposed my solutions to those problems and implemented both DW-MAC and S-MAC on the Crossbow MICAz motes. I refined the design of proportional mapping in DW-MAC so that it can still work in the hardware with slower CPUs without degrading the channel efficiency. Considering the nature of clock drift and clock rate differences among sensor nodes, I used a simple time synchronization mechanism to periodically synchronize the time in the network. In addition, I analyzed the impacts of timing inaccuracy on DW-MAC mathematically and introduced the concept of *guard time*. With the guard time allocated to the front of a mapped time period in the Sleep period, the collision-free property of proportional mapping in DW-MAC is retained even if time inaccuracy exists among sensor nodes. I also discussed the problem of packet timestamping in the real-world implementation. With statistical analysis of historical records, the processing delay of packets with invalid timestamps could be roughly estimated. In addition to the problems and experiences in physical implementation described, I also evaluated the performance of both DW-MAC and S-MAC through the experiments and simulation in *ns*-2. I evaluated the performance not only at the end-to-end level but also at the per-hop level to reveal the detailed behavior of both MAC protocols. The fairness of channel competition between the two flows was also analyzed and discussed.

In the current implementation, DW-MAC only supports one data packet transmission in one mapped time period in the Sleep period. With small duty cycle values, i.e., less than 1%, this restriction may limit the channel efficiency and increase network latencies. In the future, DW-MAC could be enhanced to support variable SCH length and multiple data packet transmissions in a mapped time period, further decreasing the network latency. The implementation of the MAC protocols could also be extended with broadcasting support.

Bibliography

[1] Crossbow MICA2 datasheet.
http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.

[2] Crossbow MICAz datasheet.
http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf.

[3] S-MAC source codes for TinyOS 1.x.
http://www.isi.edu/ilense/software/smac/download.html.

[4] TinyOS website. http://www.tinyos.net.

[5] UPMA Package: Unified Power Management Architecture for Wireless Sensor Networks.
http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/wustl.

[6] M. Buettner, G.V. Yee, E. Anderson, and R. Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 307–320. ACM, 2006.

[7] T. Canli, M. Hefeida, and A. Khokhar. BulkMAC: a cross-layer based MAC protocol for wireless sensor networks. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, pages 442–446.

ACM, 2010.

[8] T. Canli and A. Khokhar. PRMAC: Pipelined routing enhanced mac protocol for wireless sensor networks. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–5. IEEE, 2009.

[9] S. Du, A.K. Saha, and D.B. Johnson. RMAC: A routing-enhanced duty-cycle MAC protocol for wireless sensor networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1478–1486. Ieee, 2007.

[10] A. El-Hoiydi and J.D. Decotignie. WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks. In *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, volume 1, pages 244–251. Ieee, 2004.

[11] IEEE 802.11 Working Group and others. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. 1997.

[12] K. Klues, G. Hackmann, O. Chipara, and C. Lu. A component-based architecture for power-efficient media access control in wireless sensor networks. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 59–72. ACM, 2007.

[13] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49. ACM, 2004.

[14] S. Ping. Delay measurement time synchronization for wireless sensor networks. *Intel Research Berkeley Lab*, 2003.

[15] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107. ACM, 2004.

[16] Y. Sun, S. Du, O. Gurewitz, and D.B. Johnson. DW-MAC: a low latency, energy efficient demand-wakeup MAC protocol for wireless sensor networks. In *Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, pages 53–62. ACM, 2008.

[17] Y. Sun, O. Gurewitz, and D.B. Johnson. RI-MAC: a receiver-initiated asynchronous duty cycle MAC protocol for dynamic traffic loads in wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 1–14. ACM, 2008.

[18] L. Tang, Y. Sun, O. Gurewitz, and D.B. Johnson. EM-MAC: A Dynamic Multichannel Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proc. of the 12th ACM Int. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc11). ACM*, 2011.

[19] L. Tang, Y. Sun, O. Gurewitz, and D.B. Johnson. PW-MAC: An energy-efficient predictive-wakeup MAC protocol for wireless sensor networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 1305–1313. IEEE, 2011.

[20] T. Van Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180. ACM, 2003.

[21] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference*

of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 3, pages 1567–1576. IEEE, 2002.

[22] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *Networking, IEEE/ACM Transactions on*, 12(3):493–506, 2004.