IMPLEMENTATION OF A HIGH THROUGHPUT LOW POWER MAC PROTOCOL IN WIRELESS SENSOR NETWORKS

by

Chin-Jung Liu

A THESIS

Submitted
to Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

COMPUTER SCIENCE AND ENGINEERING

2011

ABSTRACT

IMPLEMENTATION OF A HIGH THROUGHPUT LOW POWER MAC PROTOCOL IN WIRELESS SENSOR NETWORKS

by

Chin-Jung Liu

This thesis presents the design, implementation, and evaluation of TATD-MAC, a TDMA-based low duty cycle synchronous MAC protocol that improves throughput by increasing channel utilization with a traffic-adaptive time slot scheduling method. Conventional time division multiple access (TDMA) introduces significant end-to-end packet delivery delay and its throughput is limited. TATD-MAC achieves higher throughput by improving TDMA with a novel traffic-adaptive mechanism that assigns time slots only to nodes that are expecting traffic. Our traffic-adaptive mechanism is a two-phase design, which decomposes the DATA period into traffic notification part and data transmission scheduling part. The two-phase design enables TATD-MAC to optimize the control packets and improve their energy efficiencies according to the characteristics of each phase. The source nodes inform all nodes on the routing path that these sources have outgoing traffic by transmitting traffic notification packets in a "pulse" fashion. With traffic notification packets, every node on the routing path claims time slots in data transmission part. Therefore, TATD-MAC is able to forward a packet over multiple hops in a single cycle and thus reduce the end-to-end delay. The data transmission scheduling mechanism only assigns time slots to nodes with traffic through an ordered schedule negotiation scheme. This innovative traffic-adaptive scheduling mechanism assigns time slots based on traffic and totally eliminates the idle listening slots on nodes with no traffic. Moreover, if any other nodes need more time slots, they are able to claim them, which further improves channel utilization and achievable throughput. We implemented a TATD-MAC prototype on Tmote-Sky running TinyOS 2.1.0. Performance evaluation shows that TATD-MAC significantly improves throughput compared to conventional TDMA and achieves the same throughput as TDMA with slot stealing while having 70% less power consumption.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Li Xiao, who patiently motivated me to develop this thesis and guided me to complete it. Without her guidance, this thesis could not be successfully completed. My gratitude is also devoted to my review committee members Dr. Matt Mutka and Dr. Guoliang Xing for their support and guidance.

I would also like express my deepest appreciation to my parents, Jung-Hua Liu and Su-Chu Chen for their sacrifices and unconditional love. My parents are the role models to me for being persistent toward every kind of challenges. I am also grateful to every member of the ELANS lab, especially Pei Huang for answering my endless questions and for helping me to overcome all kinds of challenges. I want to thank my labmate, Kanthakumar Pongaliur, for his help in writing this thesis draft and for his precious advices to the content. Also, I would like to thank my girlfriend Boyang Tong, whose love enabled my to overcome the frustrations throughout the development of this work.

I want to further thank all faculty members of the Computer Science and Engineering department at Michigan State University, for enriching my knowledge in the field. Finally, I would like to thank everyone who directly or indirectly offered his or her help to this thesis. Especially, I would like to give my special thanks to Dr. Chung-Ta King at National Tsing Hua University. Without Dr. King's support, I would not be able to be a student at Michigan State University. Special thanks should also be given to my dear friends: Chien-Wei Chang, Dr. Chen-Lung Chan, Chien-Han Chen, Hsin-Yu Chen, Yu-Kai Huang, Su-Chen Liao, Wei-Yen Lin, and Shih-Wen Su. I appreciate them for sharing my joy, sadness, and my growth.

As for any remaining errors or deficiencies, the responsibility for this work rests entirely upon the author.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

INTRODUCTION

1.1 Motivation

Wireless sensor networks (WSN) are composed of tiny wireless sensing devices with severely constrained resources and limited energy. On most sensor nodes, wireless communication is the task that consumes the highest portion of the energy. It is essential to guarantee that only one node transmits in a radio channel at a time, otherwise neighboring nodes transmission at overlapped time may result in collision. Hence, the design of energy-efficient multiple access control (MAC) protocols on such energy-constraint devices is widely studied. A main issue against energy efficiency is *idle listening*, in which a node is active and listening to the radio channel even though no packet is being transmitted to it. One of the primary techniques to reduce idle listening is *duty-cycling*. Every node is periodically cycling between sleep state and active state. In sleep state, nodes minimize their power consumption by turning off as many hardware components as they can, especially wireless communication components. However, two nodes can only communicate with each other when they are both active. The main challenge of designing a duty-cycled MAC protocol is to achieve high throughput and low delay while maintaining good energy-efficiency.

In this thesis, we introduce a novel traffic-adaptive time division multiple access control (TATD-MAC) protocol, which improves throughput and channel utilization with low overhead. The duty cycle is reduced to a very low value and the throughput is improved, unlike in prior works. Before introducing our novel approach, we first identify the common issues of current MAC protocols.

1.2 The Design Challenges of MAC protocols

In this section, we discuss the challenges of designing an energy-efficient MAC protocol and classify duty-cycle MAC into two main categories and discuss their characteristics.

We can roughly categorize duty-cycled MAC protocols into synchronous and asynchronous MAC protocols. *Synchronous* MAC protocols [1][2][3] arrange a schedule that specify when a node should wake up to communicate with each other and when a node should sleep within a cycle. The overhead is the nodes have to be synchronize their clocks to the the global timeso that the sender and receiver can wake up at their assigned schedule. However, in applications that require time stamping, synchronization is no longer an overhead. Another MAC paradigm in wireless networks is *asynchronous* MAC protocols [4][5][6][7]. They rely on *low power listening* (LPL) to ensure reliable message delivery with low power consumption. When a node has to transmit a packet, it transmits a long control packet that consists of a preamble sequence, which has to be long enough to span a complete receive check period. Each node wakes up and senses the radio channel periodically. If a node senses preamble in the radio channel, it stays awake for the full duration of the preamble and waits for receiving the packet, otherwise it goes back to sleep. The receiver only wakes up and senses the radio channel periodically, thereby idle listening is reduced. Several hybrid MAC protocols [8][9] that combine the advantages of synchronous and asynchronous LPL techniques, are also developed.

1.2.1 Asynchronous MAC Protocols

Although the simplicity of LPL makes it a widely adopted technique to achieve energy-efficiency, the long preamble in LPL has several disadvantages and side effects. Firstly, although the sender and receiver do not have to be synchronized, LPL is not able to achieve optimal energy-efficiency at both sender and receiver ends. The sender has to transmit the long preamble before the data. The burden of long preamble is totally at the sender side. After the receiver senses the preamble, the receiver has to stay awake to wait for the preamble to end so that the data/acknowledgment exchange process can start. The preamble is not exemptible even though the receiver is already awake. These unnecessary preambles and idle wake-ups inevitably waste certain amount of energy on both sender and receiver sides. Secondly, all nodes within the transmission range of a sender are able to sense the long preamble from any sender and they stay awake to receive the data, even

though the packet is not destined for them. This is the so called *overhearing problem*, which results in waste of energy on non-recipient nodes within the sender's transmission range. Finally, because the packet recipient has to wait for the whole preamble to end before starting to receive the data, the length of the preamble bounds the end-to-end delay and limits the achievable throughput. The accumulated end-to-end delay might be quite considerable on a multi-hop routing path.

1.2.2 Synchronous MAC Protocols

Although synchronous MAC protocols are able to achieve higher throughput than asynchronous MAC protocols, they suffer seriously from high end-to-end delivery delay. For example, in S-MAC [1], time is divided into repetitive cycles and a cycle is further divided into three periods: SYNC, DATA, and SLEEP period. Every node wakes up at the beginning of SYNC period to synchronize their clock to the global time and remains awake in DATA period. In DATA period, nodes that have outgoing traffic exchange *Request-to-Send* (RTS) packets and *Clear-to-Send* (CTS) packets in a contention-based manner. In SLEEP period, nodes that are not involved in wireless communication do not wake up at all. Only nodes involved in wireless communication wake up at their time slots, exchange data/acknowledgement and then go back to sleep. It is obvious that S-MAC relays a packet only one hop each cycle and thus the latency is so high that we cannot ignore it. Although this issue is addressed and improved in S-MAC [10] by introducing the *adaptive listening*, the improvement is still limited.

The high end-to-end delay makes synchronous MAC protocols inappropriate for multi-hop time critical applications. In order to achieve lower latency, some applications [11][12] give up power management and let the nodes stay awake, while some other applications [13][14] require the nodes to remain active for a time period long enough to transmit the data. Such approaches sacrifice energy-efficiency and throughput to achieve low latency. If the events are happening infrequently, only a small portion of time is involved in reporting these events and a large portion of time is idle listening.

1.3 TATD-MAC Overview

As a TDMA-based MAC protocol, a cycle in TATD-MAC is also divided into SYNC, DATA and SLEEP period. However, in prior works, traffic notification and scheduling of data transmission for SLEEP period are both done together in DATA period. The control packets in DATA period serve as traffic notifications and are also responsible for scheduling data transmission during SLEEP period. Such control packet design limits the scheduling performance and increases the control overhead, as the scheduling is per-packet based. Therefore, in TATD-MAC, we propose a two-phase DATA period design, which further divides the DATA period into RESV and SCHED period. TATD-MAC is a hybrid of *carrier sense multiple access* (CSMA) and TDMA, where CSMA is performed in RESV period and TDMA is performed in the SCHED and SLEEP period. The two-phase design enables TATD-MAC to optimize the control packets in RESV and SCHED period according to their characteristics and to improve their energy efficiencies, respectively. In RESV period, sources inform all nodes on the routing path that they have outgoing traffic by initiating a traffic notification packet. The potential collisions of these traffic notification packets do not affect the scheduling mechanism in SCHED period and do not affect the data transmission in SLEEP period. The data transmission scheduling mechanism in SCHED period only assigns time slots to nodes with traffic through an efficient schedule exchange mechanism. Such two-phase design of DATA period in TATD-MAC enables the nodes to derive a per-flow traffic notification and data transmission scheduling method with much lower overhead and yields even higher channel utilization compared with conventional TDMA protocols.

1.4 Organization

We organize the rest of this thesis as follows. In Chapter 2, we discuss more details about related works in low duty cycle synchronous MAC protocols and TDMA protocols. In Chapter 3, we present the detailed design of TATD-MAC, including traffic notification in RESV period and data transmission scheduling mechanism in SCHED period. The implementation details, such as

component descriptions and system compositions are introduced in Chapter 4. The performance evaluations of our implementation prototypes are presented in Chapter 5. We compare the performance of our TATD-MAC prototype with TDMA with and without slot stealing. Finally, we draw a conclusion by discussing the evaluation of this work in Chapter 6.

Chapter 2

RELATED WORK

In this chapter, we discuss the most representative synchronous MAC protocols and several related TDMA protocols.

2.1 Synchronous MAC protocols

In this section, we discuss several synchronous MAC protocols and their most important characteristics.

S-MAC [10] with adaptive listening is a low-power, synchronous, RTS/CTS-based MAC protocol with adaptive listening. The nodes in the network wake up periodically and exchange synchronization and scheduling information.

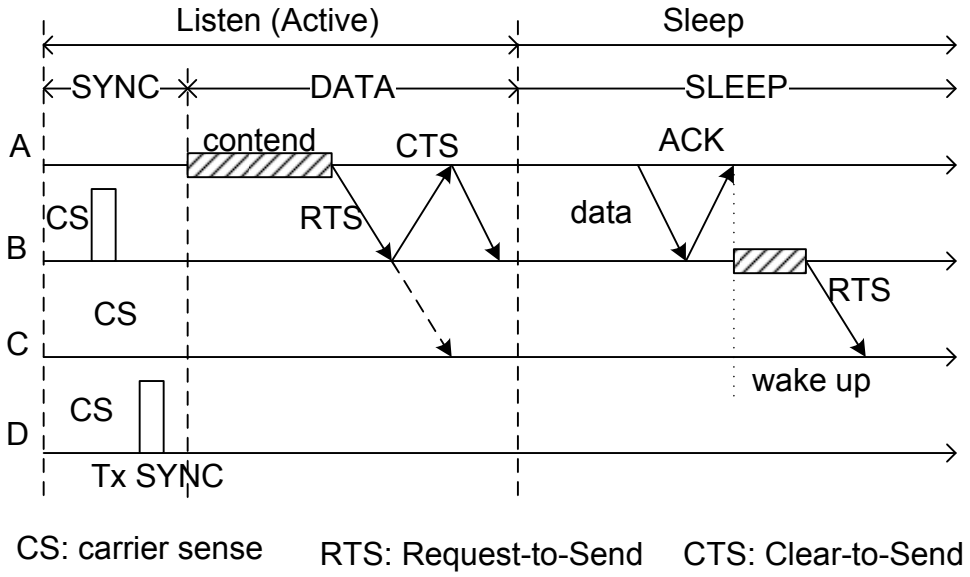

Figure 2.1: The adaptive listening in S-MAC.

As shown in Figure 2.1, during Listen/Active period, node A broadcasts a *request-to-send* (RTS) packet to notify all nodes in its communication range that node A has some data to transmit. After receiving the RTS from node A, node B transmits a *clear-to-send* (CTS) packet as a reply

to notify node A that node B knows node A has some data to transmit. In the mean time, node C overhears this CTS from node B and node C knows that node B is going to receive some data. Therefore, node C wakes up at the end of the transmission from node A to node B so that node B can forward the data to node C right away, instead of waiting until next cycle. S-MAC with adaptive listening can relay packets by at most two hops per cycle and generally cannot go beyond this because the next hop (i.e., node D) knows nothing about current transmission from node A to node C and thus will not wake up.

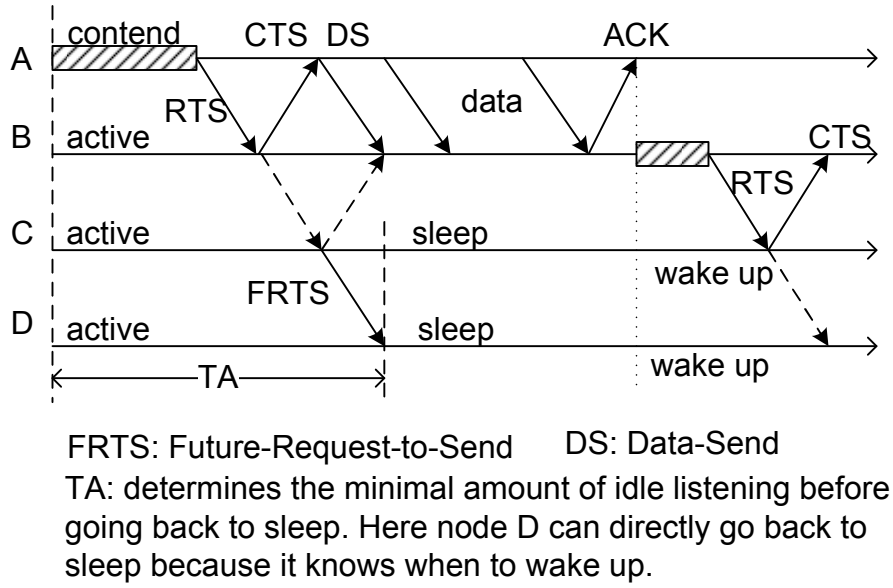

Figure 2.2: The FRTS mechanism in T-MAC.

T-MAC [8] proposes *future request-to-send* (FRTS) to extend the delivery of a packet by one hop. As shown in Figure 2.2, node C transmits a FRTS after it overhears the CTS from B. The original purpose of FRTS is to let node C with outgoing data to notify the intended receiver (i.e., node D in Figure 2.2) to not access the medium at current time. Since the FRTS packet contains the length of current data transmission, which is accessible from the overheard CTS packets, node D knows when to wake up and receive from C. Therefore, with a slight modification that node C sends a FRTS packet either when it has pending data to send or when it overhears a CTS, the FRTS can extend the delivery of a packet to up to three hops per cycle. However, nodes further downstream cannot overhear FRTS and thus do not wake up.

RMAC [2] further improves the number of hops that a packet can be relayed in a cycle by introducing PION, the *source-initiated pioneer frame*. In DATA period, any node with outgoing traffic initiates a PION and forwards it multiple hops in order to inform all nodes on the routing path when to wake up to receive possible incoming packets during the SLEEP period. A PION, while propagating on the routing path, not only serves as a RTS frame to request communication, but also confirms a request in a way similar to CTS frame. Since every node on the routing path wakes up to receive packet when its upstream node is ready to send the data packet to it, RMAC not only relays a packet multiple hops in a single cycle, but also reduces forwarding latency significantly.

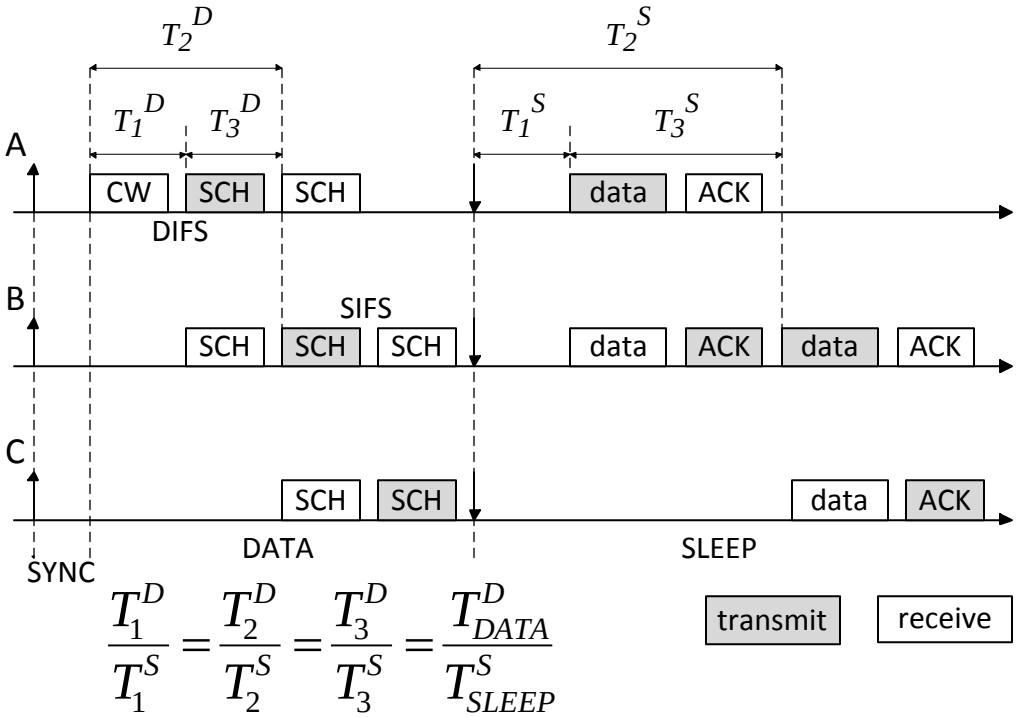

Figure 2.3: Multihop forwarding of a packet in DW-MAC

As shown in Figure 2.3, DW-MAC [3] has a mechanism similar to PION of RMAC, called SCH. The time point T_1^S in DW-MAC is 0 in RMAC. However, the PION of RMAC has a main drawback, that two hidden terminals might cause collision during data transmission although the exchanges of PION are successful. For instance, in Figure 2.3, both node C and node A initiate

their own PIONs, either at the same time or one is later than the other. Since both node A and C are the first hop and their PION are confirmed by their receiver, they start data transmission at the same time when SLEEP period starts and their data transmissions collide. If collision happens, all downstream nodes wake up but receive no data, because the expected packet is lost due to collision.

Therefore, DW-MAC [3] also introduces a one-to-one mapping function to ensure collision-free data transmission during SLEEP period. As shown in Figure 2.3, when node A receives a confirmation SCH from node B, the time length of T_3^S is assured to be collision-free, because node B receives a SCH correctly during T_3^D and they are one-to-one scaled based on the ratio between T_{SLEEP} and T_{DATA}. Therefore, if the sending and receiving SCH are successful, the reservation process is completed and data transmission during SLEEP period never collides. If the SCH sent to node B is collided and lost, node A is not going to receive a confirmation SCH and the reservation during T_3^S between node A and node B is failed. However, if collisions happen at node A and the confirmation SCH is lost, node A does not wake up to send the data at T_3^S, but node B will wake up to receive. It is hard for the sender to distinguish whether the collision is happening to the request SCH or to the confirmation SCH. The sender can only conservatively assume the reservation is failed. Therefore, if a confirmation SCH is lost, the downstream nodes will wake up unnecessarily to receive the expected packet that never arrives.

The problem with waking up nodes unnecessarily is relatively tolerable compared to the problem that the length of DATA period is wasted. Nodes can always go back to sleep if there is no channel activity for a certain amount of time. Any failure of a sequential SCH reservation not only waste the time it spent, but it also significantly limits the number of packets that can be delivered in a cycle. Those time slots that have been reserved by other nodes, but no data will be transmitted to them, whereas any other node that has data to send cannot obtain those slots. In a more complex multiple flow scenario, throughput, delivery delay, and energy consumption might get significantly compromised.

2.2 TDMA-based MACs

TDMA is widely studied and has been studied for decades. How to assign time slots in TDMA is usually modeled as a graph-coloring problem. DRAND [15], which is adopted by Z-MAC [9], is a remarkable distributed time slot assignment algorithm. Conventional TDMA protocols suffer from low channel utilization and high delay under light traffic loads, because a node only transmits in the time slots it owns. Aiming at improving channel utilization of TDMA, Z-MAC proposes a slot stealing technique, by allowing nodes to contend for the slots that are not owned by these nodes with less probability than that of the owner. Impressively, under light traffic loads, Z-MAC approaches the baseline CSMA in a dense network, because nodes with data to send contends for slots that are owned by neighboring nodes. On the contrary, TATD-MAC assigns time slots to those nodes that need time slots at the very start of each cycle, instead of each node contending for slots when they need time slots.

TDMA-ASAP [16] also studies the slot stealing technique. To achieve slot assignment at the start of each cycle, a low overhead time slot assignment algorithm must be developed. NAMA [17] proposes a neighbor-aware contention resolution (NCR) algorithm, a time slot assignment method that selects only one transmitter per two-hop neighborhood without introducing any extra communication overhead by utilizing the deterministic characteristic of pseudo-random functions. This method is a possible approach for achieving traffic-adaptive slot assignment. However, the NCR results in an unfair time slot assignment and the channel utilization is low due to inconsistent knowledge of the schedule of the nodes.

TRAMA [18] is a traffic-adaptive MAC that switches between random access mode and scheduled access mode. Neighbor discovery and schedule exchanges are performed in random access mode. Nodes exchange data packets in scheduled access mode according to the schedule they agreed in random access mode. TATD-MAC differs from TRAMA in several ways. First, TRAMA's slot assignment algorithm, which is adopted from NAMA, is not able to achieve good channel utilization. Second, because the schedule exchange is done in random access mode, the

time it takes to complete the schedule exchange process is unpredictable. On the contrary, TATD-MAC exchange schedule through TDMA, which guarantees the time spent on schedule exchange is predictable and minimal.

Third, the memory requirement of TRAMA is high. A sender uses a bitmap to indicate its intended receivers, which indicates that every node has to keep track of the one-hop neighbors of all its one-hop neighbors. The bitmap is an $n \times n$ matrix, where n is the number of all its one-hop neighbors. When an owner of a slot has done sending all its data, it selects a possible candidate neighbor to take over the slot. The candidate has to be the node with highest priority among all of the candidate's two-hop neighbors, which implies that the owner of a slot has to know the priorities of all its one-hop neighbor's two-hop neighbors and thus the memory usage is high. On the contrary, TATD-MAC only needs to store a list of two-hop neighbors. Compared with the high memory usage of TRAMA, TATD-MAC fits better to the resource constrained wireless sensor nodes. Fourth, the schedule packet of TRAMA is more complicated and is piggybacked to each data packet for consistency consideration, which makes the overhead much higher than TATD-MAC. Fifth, downstream nodes do not know whether they have data to send/receive or not and thus do not reserve time slots for data transmission until next cycle. The consequence is that a packet cannot be forwarded multiple hops in each cycle and the delay is high. With the traffic notification packet we propose, TATD-MAC notifies the whole routing paths of incoming data so that downstream nodes reserve time slots and it is possible to deliver a packet in single cycle.

Chapter 3

TATD-MAC DESIGN

In this chapter, we first present our motivation of designing a traffic adaptive MAC protocol in Section 3.1. In Section 3.2, we introduce the two-phase design, which decomposes DATA period into RESV period for traffic notification part and SCHED period for data transmission scheduling part. In Section 3.3, we describe how TATD-MAC calculate the priority for data transmission scheduling in SCHED period. In Section 3.4, we explain in detail about how TATD-MAC exchange scheduling packets efficiently using minimal overhead.

3.1 Motivation

WSN application differs from each other significantly. The traffic amount of some applications are quite dynamic. Take a typical WSN surveillance system [14] for example, when no intruder is detected, the whole network does not generate much data traffic. However, whenever a certain event is detected, the network generates a lot of data traffic. This kind of application needs a protocol that adapts from low data rate to large bursts of traffic. Moreover, such kind of applications demands a protocol that is able to provide low latency and high data throughput. Simply letting every node to stay active is not appropriate, since such kind of applications has to last for weeks, even months. Existing MAC protocols either are not able to achieve high throughput [2][3] or are not energy-efficient enough [9]. This motivates us to design a traffic-adaptive MAC protocol that not only achieves high throughput when there is large bursts of traffic, but also be energy-efficient when there is only small amount of data.

3.2 Two-Phase Design

As a synchronous duty-cycling MAC protocol, TATD-MAC divides time into repetitive cycles and each cycle is divided into SYNC, DATA, and SLEEP period (as shown in Figure 3.1). TATD-MAC further divide the DATA period into RESV period and SCHED period. In order to better distinguish the time slot during SCHED period and time slot during SLEEP period, a time slot in SCHED period is denoted as "*dslot*" and a time slot in SLEEP period is denoted as "*sslot*". All nodes wake up and synchronize with each other at the beginning of the SYNC period. The nodes remain active during DATA period.

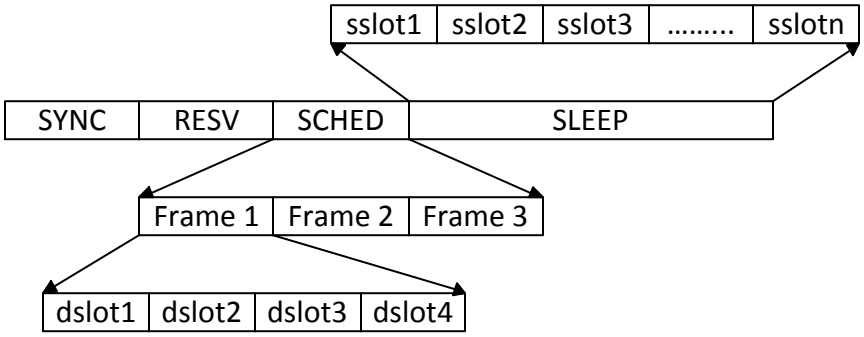

Figure 3.1: Time Division in TATD-MAC

TATD-MAC proposes a two-phase design of DATA period by dividing the DATA period into RESV period for traffic notification and SCHED period for data transmission scheduling. In RESV period, nodes with outgoing data inform all nodes on the routing path about the incoming data for them by initiating traffic notification packets. After RESV period, every node knows if it needs sslots for data transmission or not. Nodes exchange exactly three data transmission scheduling packets in the SCHED period. The time slot assignments in SLEEP period for all nodes are determined using these scheduling packets. In SLEEP period, for each sslot, a node only wakes up when it has data to send and it owns the sslot or when it is the receiver of the sslot.

Dividing DATA into two phases enables us to handle these two distinctive kinds of control

packets according to their characteristics. Data transmission scheduling packets are more complex and thus need a smart design in order to achieve high channel utilization with minimized overhead. On the other hand, the forwarding of the traffic notifications must be fast so that we can shorten the common RESV period. We make the transmission of traffic notifications faster by using as few fields as possible to make the packets short and compact.

Same as Z-MAC [9], TATD-MAC also adopts DRAND [15] to compute the time frame size and slot assignment for each node in setup phase. DRAND guarantees that no two nodes within a two-hop communication neighborhood can be assigned to the same slot and the time frame size is significantly smaller than the size of local neighborhood. However, the communication cost of performing DRAND one time is non-trivial. It is impossible to achieve traffic-adaption by applying DRAND on the set of nodes that has traffic once every cycle. Therefore, TATD-MAC only runs DRAND once at the setup phase. Using the slot assignment generated by DRAND, nodes exchange data transmission scheduling packet in TDMA in a collision-free fashion.

3.3 Priority Calculation

In this section, we introduce the neighbor-aware contention resolution (NCR)[17], which TATD-MAC adopts to determine the owner of a slot in SLEEP period. We then discuss the drawbacks of NCR and show our improvement.

A pseudo-random function with the same seed, generates the same sequence pseudo-random numbers deterministically. NCR defines the priority p_k^i for each node k in set $N2(u) \cup u$ as:

$$p_k^i = rand(k \oplus i) \oplus k, k \in N2(u) \cup u \tag{3.1}$$

where $rand(x)$ is a pseudo-random number generator that produces a sequence of random numbers that are approximately uniformly distributed with seed x, the sign $\oplus$ means concatenating its two operands, and $N2(u)$ denotes the set of two-hop neighbors of node u.

Node u wins the ith slot if for all other nodes in $N2(u)$, u is the node with highest priority:

$$\forall v \in N2(u), p_u^i > p_v^i \tag{3.2}$$

Note that although $rand(x)$ may still generate the same number with different seeds x, each priority p_k^i is concatenated with another k, the node ID. Therefore, each node is able to calculate all its two-hop neighbors' priority and thus determine which one of them is the winner of a particular time slot without exchanging any information. TATD-MAC takes advantage of this characteristic for traffic-adaptive time slot assignment. NCR is more than perfect for TATD-MAC's slot assignment algorithm, because NCR does not require any communication, which makes it a viable solution to perform traffic-adaptive time slot assignment with minimized overhead. However, NCR has two main drawbacks that may result in throughput bottleneck. Firstly, although $rand(x)$ generates random numbers that are approximately uniformly distributed with seed x, the chance of each node winning a slot is still biased. The nodes with more neighbors have lower chances to win a slot, which results in potential bottleneck on a routing path. Secondly, although a node knows which of these neighbors wins a time slot locally based on priority, the slot is wasted if the winner does not have data to send and thus channel utilization is low.

3.3.1 Fairness

As the simple chain topology shown in Figure 3.2, node C contends with four neighbors while node A and node E contend with only two neighbors. The chance of node C winning a slot is less than that of node A and node E if they all use the same uniform pseudo-random number generator. A possible solution is to use non-uniform pseudo-random number generator that produces random numbers with different distributions. TATD-MAC adopts an alternative way to achieve fairness by generating priority more times to compensate nodes with more contenders.

For the ith slot, the jth priority for a particular node $k \in N2(u) \cup u$ is modified to:

$$p_{k \oplus j}^i = [rand(k \oplus i)]_j \oplus k, k \in N2(u) \cup u, j = 1, ..., n_k \tag{3.3}$$

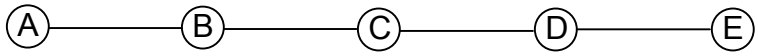

Figure 3.2: A simple chain topology

where n_k denotes the number of one-hop neighbors of node k, and $[rand(k \oplus i)]_j$ denotes the jth random number generated by $rand(k \oplus i)$. TATD-MAC only considers the number of one-hop neighbors because it is easier to maintain the consistency among one-hop neighbors.

Furthermore, in order to randomize the slot assignment of each cycle, the seed, $k \oplus i$ is added with another random number generated by using the current cycle number as seed, $rand(cycle)$.

$$p^i_{k \oplus j \oplus l} = [rand(k \oplus i + rand(l))]_j \oplus k, k \in N2(u) \cup u, j = 1, ..., n_k \tag{3.4}$$

where l is the current cycle number according to synchronization packets. Node u wins the ith slot if the following equation is true:

$$\forall v \in N2(u), \exists m, p^i_{u \oplus m} > p^i_{v \oplus j}, 1 \leq m \leq n_u, 1 \leq j \leq n_v \tag{3.5}$$

3.3.2 Channel Utilization

Depending only on pseudo-random function for time slot assignment might result in low channel utilization. As a simple example, in Figure 3.2, let the priority order of the ith slot for the nodes generated by aforementioned method is $P^i_A > P^i_B > P^i_C > P^i_D > P^i_E$. We see that only one node in the entire network wins a particular slot if the priority is sequential, which is the worst case. Although the occasions that the priority being sequential is infrequent, these occasions harm the channel utilization significantly. In this example, node D assumes that node B wins the slot as node B has the highest priority among all node D's two-hop neighbors and node E assumes node C wins the slot. However, ideally, node D or node E should be able to win this slot along with node A. Moreover, if node A does not have any data to send, all of its winning slots are wasted. Previous works [18][17] do not mention the low channel utilization problem, which substantially limits the

achievable throughput. One of our major contributions is our traffic-adaptive time slot assignment technique, which solves the problem with minimal overhead.

In order to increase channel utilization, nodes must know all their two-hop neighbors' schedule so that it can maximize the channel utilization while setting up its schedule. However, a node can only obtain its two-hop neighbors' schedule via its one-hop neighbors. To minimize the overhead, we must assign the nodes an order for their schedule broadcast. Since all node have to broadcast their schedule in the SCHED period, there is no low channel utilization issue of using conventional TDMA. Therefore, we adopt conventional TDMA scheme using the slot assignment generated by DRAND for our scheduling packet exchange process in SCHED period. As we mentioned earlier, it is acceptable to perform DRAND once in the setup phase along with the initial synchronization and neighborhood discovery.

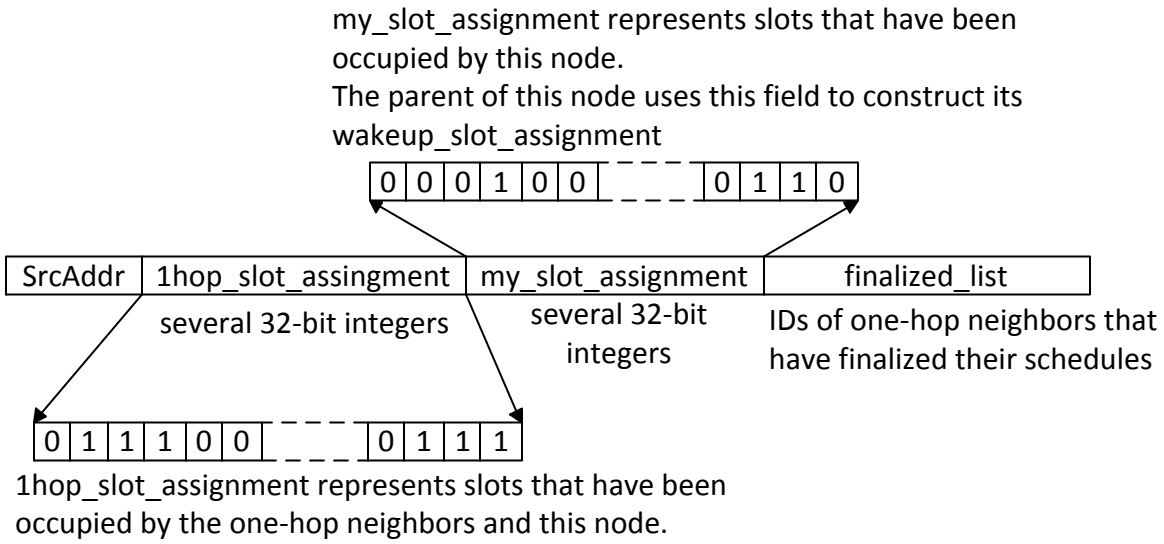

Figure 3.3: The format of schedule packets

3.4 Schedule Exchange

In this section, we discuss the three schedule exchanges during SCHED period and how TATD-MAC performs schedule exchange. TATD-MAC proposes an efficient schedule exchange algorithm using minimal overhead.

3.4.1 Schedule Exchange Algorithms

Our novel schedule exchange algorithm aims at maximizing channel utilization. In SCHED period, nodes exchange schedules in a collision-free manner using conventional TDMA. In each of the three frames, every node constructs the schedule by claiming the sslots it wins and broadcasts the schedule to all its one-hop neighbors. The decision whether a node wins a particular sslot in the SLEEP period is made based on its priority calculated by our modified NCR described in Section 3.3 and the schedules of the node's two-hop neighbors. For each sslot in the SLEEP period, a node decides whether it wins this sslot if either of the following condition holds. First, a node wins an sslot if the node has the highest priority among its two-hop neighbors. Second, a node wins an sslot if every node with higher priority than this node has already discarded this sslot. A node discards sslots by claiming its schedule has been finalized and broadcasts the scheduling packet again, which includes a finalized list containing itself and all its one-hop neighbors who have been finalized. The neighbors of this node know that the schedule of the nodes in the finalized list have been finalized and they discard any other sslots they win. There is no collision if neighboring nodes claim the unused sslots of the nodes in the finalized list, because their schedules are already finalized and they discard those sslots that are not needed.

Each sslot in the SLEEP period is represented by one bit (1: occupied, 0: free), thus the SLEEP period is represented by several 32 bit unsigned integers. The schedule of a node includes a $1hop\_slot\_assignment$ that indicates which sslot is still available, a $my\_slot\_assignment$ that helps its parent node to build the parent node's $wake\_up\_assignment$, and a $finalized\_list$ that contains its one-hop neighbors' ID whose schedules have been finalized. The format of a schedule

```
def occupySlot (i):
    my_slot_assignment[i/32] |= (1 << (i%32));
    1hop_slot_assignment[i/32] |= (1 << (i%32));
    2hop_slot_assignment[i/32] |= (1 << (i%32));

if traffic == True then
    slotAssigned ← 0;
    for i in range(0, num_of_slots) do
        if slot i is available then        /* by checking 2hop_slot_assignment */
            if highestPriority == True then
                occupySlot (i);
                slotAssigned ++;
                if slotAssigned == slotNeeded then
                    finalized_list.append(myNodeID);
                    break;
                end
            else if each node with higher priority can be found in the 2hop_finalized_list
            then
                occupySlot (i);
                slotAssigned ++;
                if slotAssigned == slotNeeded then
                    finalized_list.append(myNodeID);
                    break;
                end
            end
        end
    end
else
    finalized_list.append(myNodeID);
end
```

Algorithm 1: Schedule construction.

packet is shown in Figure 3.3. Note that a node does not regard the sslots claimed by any two-hop neighbor as occupied, because another one-hop neighbor of this node can still reuse this sslot, if this one-hop neighbor is three-hops away from the two-hop neighbor. Take node E in Figure 3.4 as an example, it does not care whether an sslot is taken by node B or not. Therefore, node D should not regard any sslot claimed by node B as occupied. The schedule construction of a node is illustrated in Algorithm 1. A node also maintains records for its own reference, including a $2hop\_slot\_assignment$ and a $2hop\_finalized\_list$.

```
1hop_slot_assignment |= pkt→my_slot_assignment
2hop_slot_assignment |= pkt→1hop_slot_assignment
if pkt→nodeID in my_child_list then
  | wakeup_slot_assignment |= pkt→my_slot_assignment
end
foreach nodeID in pkt→finalized_list do
  | if nodeID is not in 2hop_finalized_list then
  |   | 2hop_finalized_list.append(nodeID);
  |   | if nodeID is oneHopNeighbor then
  |   |   | finalized_list.append(nodeID);
  |   | end
  | end
end
```

Algorithm 2: Schedule handling.

A node knows if it has the highest priority for a particular sslot and which of its neighbors has higher priority than itself by using the pseudo-random function described in Section 3.3. When a node receives a schedule from its neighbor, it uses the bitwise OR (|) operator to aggregate the schedule with the schedule it already obtained as described in Algorithm 2.

3.4.2 A Simple Example

To simplify the presentation, we focus on the example illustrated in Figure 3.4 and Figure 3.5 (on page 21 and 22). As shown in Figure 3.4, suppose that DRAND has already generated a time frame of size 3 and node C owns the first dslot, node A and node D own the second dslot, and node B and E own the third dslot. In this example, the three SCHED frames are composed of nine dslots. Suppose our modified NCR algorithm generates a round robin priority order (as shown in Figure 3.4) and node A has no data to transmit. The goal is to assign five sslots according to nodes' traffic. In Figure 3.4, there are five sslots, which are represented by five dark blocks. We illustrate which node takes the sslot for easy understanding, but in real implementation, node ID is not part of the message exchanged and thus the overhead is low.

In the first dslot of the first SCHED frame, node C claims its schedule and broadcast the schedule to node B and node D. Node C has the highest priority for sslot 3 among all its two-hop

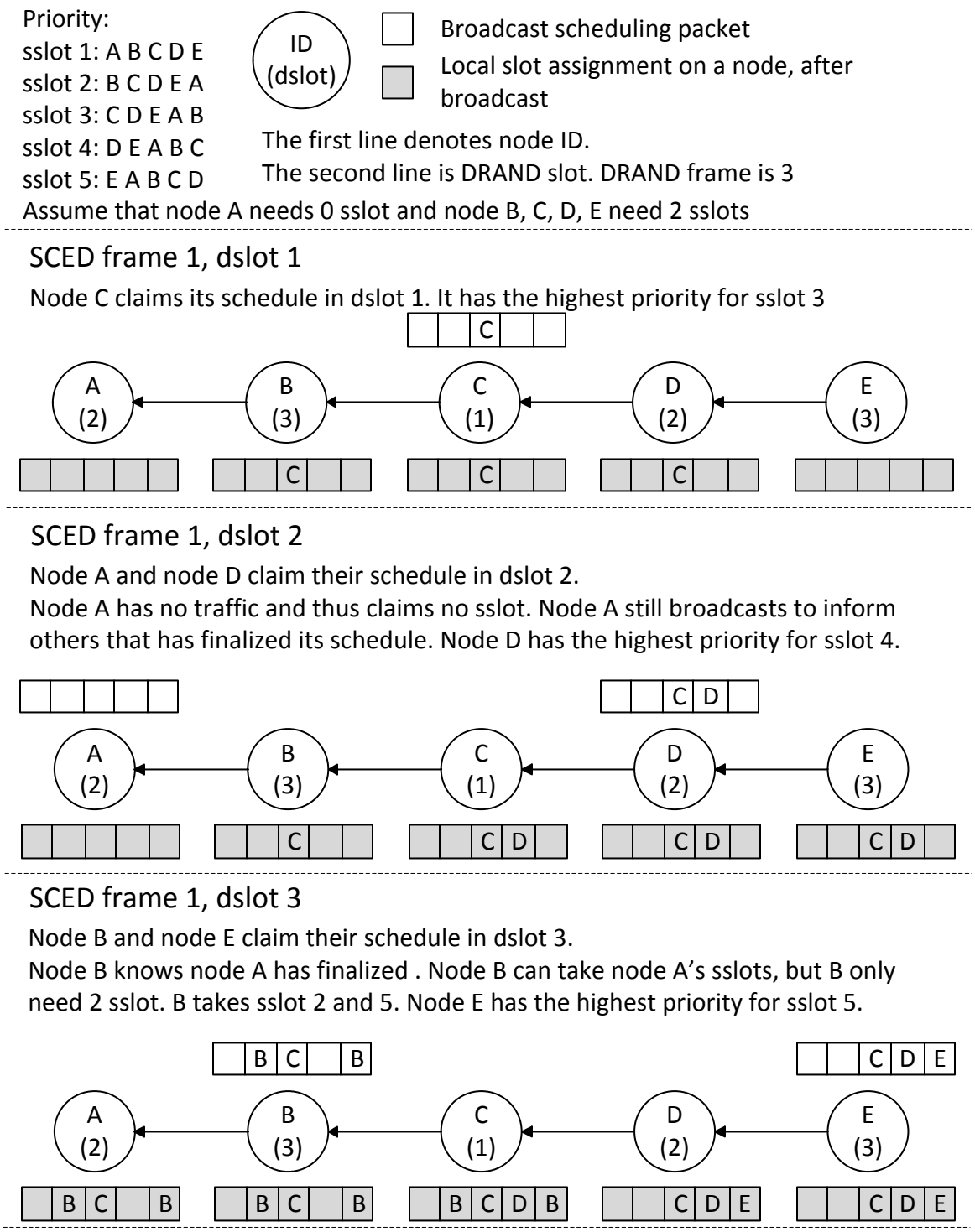

Figure 3.4: Time slot assignment in TATD-MAC, frame 1

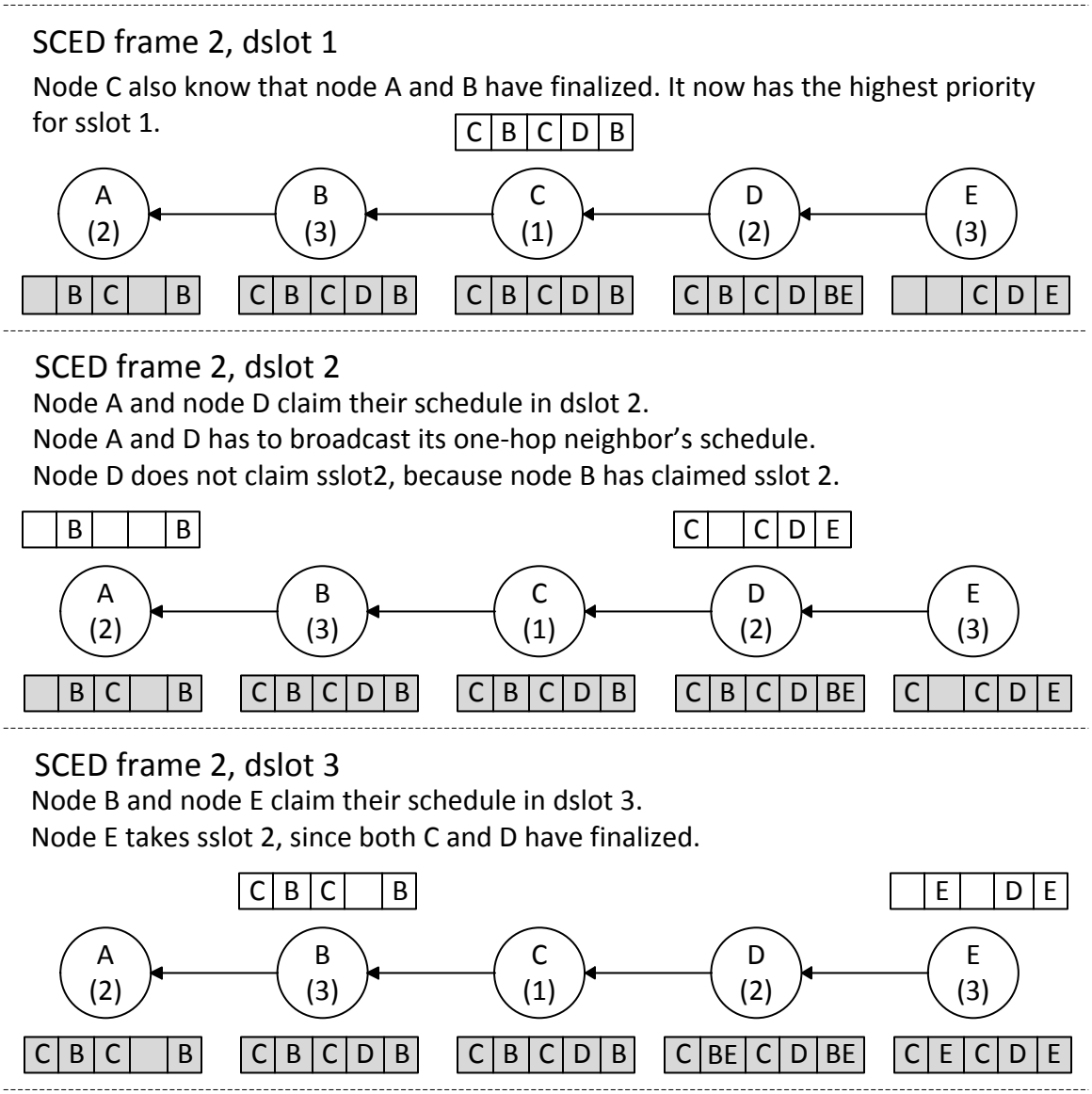

Figure 3.5: Time slot assignment in TATD-MAC, frame 2

neighbors and thus it claims that it takes sslot 3. If node C only needs one sslot, it also set its schedule as finalized, which means it needs does not need any more sslots. Otherwise, node C does not set its schedule as finalized and it waits to see if all other nodes with higher priority are finalized so that node C can take the sslots those nodes discarded. Next, node A and node D claim their schedule in the second dslot of the first frame, then node B and E, then node C, and so on.

3.4.3 Summary

Through the schedule exchange process, the adverse effect of the sequential priority order is diminished. Otherwise, for the example shown in Figure 3.4 and Figure 3.5, only one node can transmit in an sslot and none will transmit in sslot 1. The underlying principle is that for each frame, a node claims sslots once and then it broadcasts its schedule once. After the first frame, every node knows the sslot assignment of all its one-hop neighbors. After the second frame, the information is relayed to every node's two-hop neighbors and thus each node knows the sslot assignment of all its two-hop neighbors. In the third frame, every node can finalize its schedule and broadcast the schedule to its neighbors to inform them when they should wake up. Throughout these three SCHED frames, nodes that do not have data to send or do not need extra sslots set their schedules as finalized and their neighbors know that these nodes discard every sslot they win. Any other node that needs extra sslot can also claim those sslots if all other nodes with higher priority than itself are finalized. On the other hand, if a node intends to acquire more sslots, it does not set its schedule as finalized, so the nodes with lower priority cannot claim the sslots that might be taken by this node. Consequently, even though nodes claim sslots in the second or third frame, there is still no collision. To sum up, each node broadcasts three scheduling packets and this schedule exchange procedure takes three frames. This little overhead substantially increases the channel utilization.

3.5 Traffic Notification

In previous sections, we demonstrated how to achieve high channel utilization. In this section, we introduce the mechanism to inform the nodes on the routing path about the incoming packets to them, and thus they have to claim sslots in the SCHED period.

In the beginning of the RESV period, every node with data to send initiates a traffic notification packet, called NOTI, which contains an *Src* (source ID) field and an *Nxh* (next hop) field. The *Src* (source ID) field indicates the sender of this message. Based on the *Src* field, a node that receives multiple NOTIs builds a list of child nodes and then uses this child list to aggregate the *wakeup_slot_assignment* from the schedules it receives in the SCHED period. The *Nxh* (next hop ID) field gives the NOTI a dual function, to act as a traffic notification and a confirmation as well.

3.5.1 The Dual Function of NOTI

When a node receives a NOTI, it responds a NOTI immediately. The responded NOTI has two functions. First, it acts as a confirmation message to the sender of the request NOTI it receives. Second, it acts as a request NOTI to the next hop. For example, when node A sends a NOTI to node B, node B receives it and node B responds with a NOTI immediately. The NOTI sent by B is not only a request NOTI to node C, but also a confirmation to node A.

When node A initiates the NOTI, it sets *Src* as A and *Nxh* as B so that B knows itself is the intended receiver of the NOTI. When node B receives the NOTI, it sets *Src* as B and *Nxh* as C so that A knows its next hop node has received the NOTI and sends a new request NOTI to the next hop of B. Node A's notification is done when node A receives this confirmation NOTI. Node C also sets the *Src* to itself and *Nxh* to its own next hop. This procedure repeats until the NOTI reaches the sink.

3.5.2 Lost NOTI Retransmission

It is possible that a packet is lost due to various reasons, and so are NOTIs. If node A does not receive a confirmation when the NOTI times out, either the NOTI from node A to node B is lost, or the confirmation NOTI from node B to node A is lost. In either case, node A retransmits the NOTI when the last NOTI times out. In the first case, everything is still the same as if the last NOTI did not even occur. In the second case, if node B has already sent a request NOTI to C and node B receives another NOTI from node A, node B responds a confirmation NOTI with *Nxh* as -1 so that the NOTI only serves as a confirmation NOTI.

3.5.3 Multiple Flow on a Routing Path

Since traffic in wireless sensor networks converges at the sink, there might be multiple flows on a routing path. If a node has already notified its next hop and there are more NOTIs coming to it, it only responds confirmation NOTIs, instead of dual function NOTIs, because we only need to notify common paths once. We conservatively assume that the longest path of a wireless network is along the network border. Therefore, we can set the length of the RESV period as:

$$T_{RESV} = \frac{X+Y}{R} \times (durNOTI + SIFS) + CW \tag{3.6}$$

where X and Y are the length and width of the network field, R is the communication range, $durNOTI$ is the time spent to send one NOTI frame, $SIFS$ is the shortest inter-frame space, and CW is the contention window. Compared with RMAC [2] and DW-MAC [3], the most significant advantage is that the collision of the control packets no longer limits the performance of data delivery and we can set the length of the RESV period to be fixed and short.

Chapter 4

PROTOTYPE IMPLEMENTATIONS

This section describes the structure of our TATD-MAC prototype, TDMA prototype and TDMA with slot stealing prototype in detail. We implement our prototypes running TinyOS 2.1.0 on Tmote-Sky [19] mote, which integrates TI MSP430 microcontroller with 10kB RAM and an IEEE 802.15.4 compliant RF transceiver CC2420 [20].

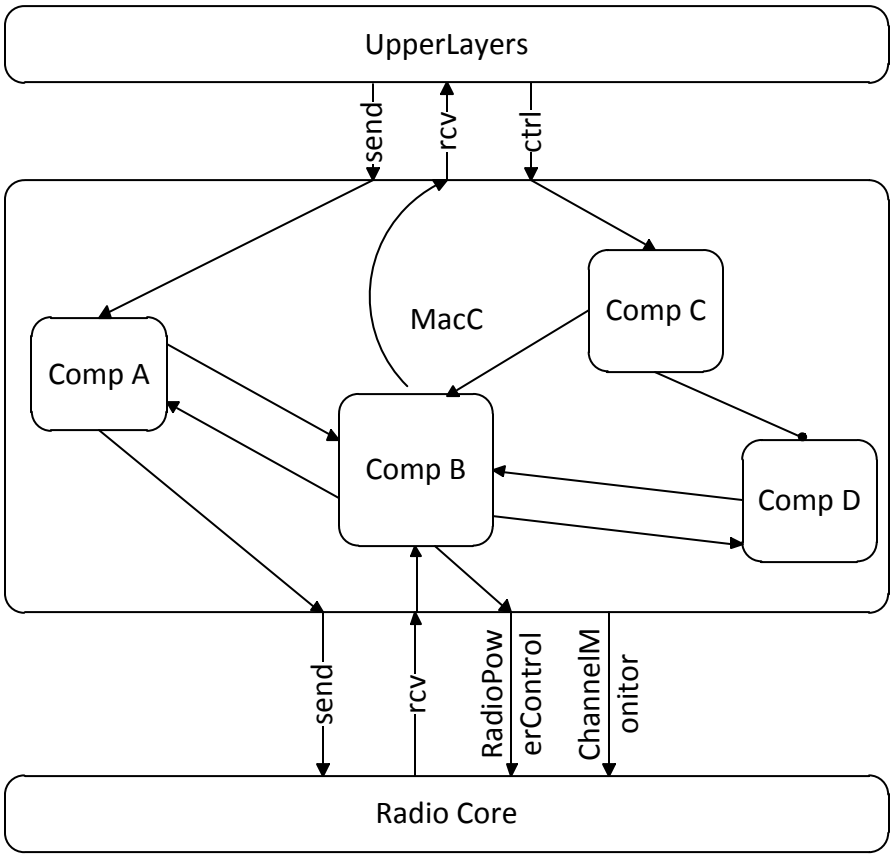

Figure 4.1: The application developer's view of MLA. Each block is a component and an arrow denotes an interface. The boxes are software components.

We implement our prototypes in MAC layer architecture (MLA) [21], a component-based architecture for power-efficient MAC protocol development in wireless sensor networks. As shown

in Figure 4.1, MLA inserts an extra layer, `MacC`, into TinyOS CC2420 radio stack above the radio core. Various components compose the `MacC`, which uses a set of unified interfaces provided by the radio core, and exposes a set of unified interfaces to the upper layers. Each MAC protocol defines their `MacC` with the components the MAC protocol needs. Since `MacC` uses and exposes a unified set of interface, MLA can switch MAC protocols simply by replacing the configuration `MacC`.

As we can see in Figure 4.1, `MacC` interacts with upper layer through the three interfaces `MacC` exposes, including `Send`, `Receive`, and `SplitControl`. `SplitControl` is the interface that upper layer uses to control `MacC`, either to start it or stop it. `MacC` interacts with radio core through `Send`, `Receive`, `RadioPowerControl`, and `ChannelMonitor`. Controlling the core radio to achieve energy efficiency is the fundamental reason of designing MAC protocol. The interface `RadioPowerControl` enables `MacC` to turn on, or turn off the radio core. `ChannelMonitor` is the interface for `MacC` to control radio core to do *clear channel assessment* (CCA).

4.1 TATD-MAC Prototype

This section describes our TATD-MAC implementation in detail. We walk through the structure of TATD-MAC. Then we identify various challenges and explain how we conquer them. Table 4.1 shows the default values of TATD-MAC prototype's parameters.

4.1.1 Structural Overview

As shown in Figure 4.2, the interactions between components in TATD-MAC is rather complex. Although we have already simplified this composition graph by showing only important component and interfaces, it is not easy to understand our implementation merely by composition graph. Hence, we choose to explain our TATD-MAC prototype in a high-level top-down approach.

There are three phases in TATD-MAC, synchronization phase, setup phase, and working phase. Every node in the network starts with synchronization phase, in which nodes synchronize with each other through the *flooding time synchronization protocol* (FTSP) [22] for a certain amount of time.

Table 4.1: The default TATD-MAC parameters

Parameter	Value
slot size (both dslot and sslot)	50 milliseconds
Number of slot (dslot and sslot) in a cycle	1280 slots / 64 sec
The length of schedule `SchedC` constructs	32 sslots
The times the schedule being repeated	40 times
Maximum neighbor count	20 nodes
Maxmum one-hop neighbor	8 nodes
SYNC period	1 DRAND frame
RESV period	1 DRAND frame
SCHED period	3 DRAND frame
Total sslot	1280 - 5 DRAND frame
Synchronization frequency	once per cycle
Synchronization protocol	FTSP
NOTI retry limit	10 times
Maximum child count	7 nodes
END_RECEIVE_TIME	25 milliseconds
`TATDMacSchedulerC` queue size	200
Synchronization phase length	120 sececonds for 10 nodes (depend on network size)
Setup phase length	240 seconds for 10 nodes (depend on network size)

After synchronization phase is over, TATD-MAC runs neighbor discovery and DRAND. In our implementation, we combine neighbor discovery and DRAND into a single component `DrandC`.

In working phase, time is divided into repetitive cycles. As we mentioned in Chapter 3, a cycle is further divided into SYNC, RESV, SCHED and SLEEP period. We simply implement one component for each of these periods, `SyncC`, `ResvC`, and `SchedC`. These components handle their jobs independently and interact with each other through various software interfaces. Meanwhile, we need another scheduler component to control when should these components do their jobs. Therefore, we implement a scheduler component `TATDMacSchedulerC`, which is also responsible for controlling the sending and receiving during SLEEP period.

After DRAND is done, `DrandC` informs `TATDMacSchedulerC` about the DRAND assignment and the neighbor information. `TATDMacSchedulerC` starts to work as a repetitive cycle. For each sslot during the cycle, `TATDMacSchedulerC` determines what should this node do and controls the

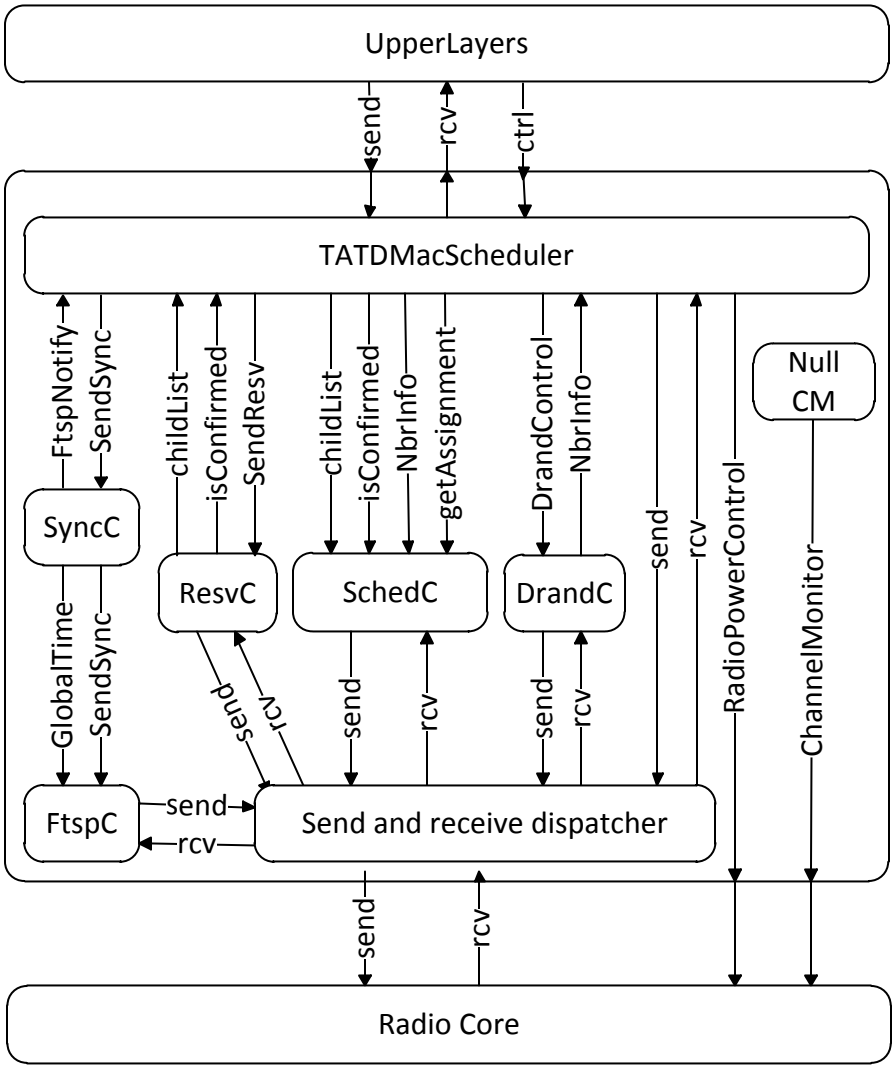

Figure 4.2: The composition graph of TATD-MAC prototype.

corresponding components. We introduce these component in the same sequence they do their jobs in a cycle.

4.1.1.1 `SyncC`

`SyncC` is the component that keeps track of current global time and determines when a new cycle should start. `SyncC` gets global time from the component `FtspC`. According to this global time, `SyncC` notifies `TATDMacSchedulerC` when it is time to start a new cycle. Each time `SyncC` notifies `TATDMacSchedulerC` that it is time to start new a cycle, it also calibrate `TATDMacSchedulerC`'s timer with the latest global time. When `TATDMacSchedulerC` is aware that a new cycle has started, `TATDMacSchedulerC` resets the current slot number of this cycle to 0 and the cycle starts over again. After the new cycle starts, it is SYNC period, `TATDMacSchedulerC` then commands `SyncC` to broadcast an FTSP synchronization packet.

4.1.1.2 `ResvC`

`ResvC` is responsible for traffic notification. `ResvC` keeps a `childList` to record which children are sending data to this node in this cycle. `ResvC` also records if the parent of this node is responding to its NOTI using a Boolean value `isConfirmed`. When the first slot for traffic notification starts, `ResvC` clears the `childList` to an empty array, and sets `isConfirmed` to false. If this node is a data source, `TATDMacSchedulerC` commands `ResvC` to initiate a NOTI packet. Throughout the whole RESV period, if a node receives a NOTI from its child, it appends the child ID to the `childList`. If a node receives a confirmation NOTI from its parent, it sets `isConfirmed` to true. When RESV period is over, `ResvC` informs `TATDMacSchedulerC` about the `childList` and `isConfirmed` it constructed. Then `TATDMacSchedulerC` informs `SchedC` about these information for `SchedC` to build the schedule for data transmission.

4.1.1.3 `SchedC`

Based on the algorithm we proposed in Section 3.3 and Section 3.4, `SchedC` is responsible for construction of the schedule for data transmission in SLEEP period. At the beginning of SCHED period, `TATDMacSchedulerC` informs `SchedC` about `childList` and `isConfirmed` and commands the `SchedC` to construct the schedule.

If a node does not have any child (i.e., `childList` is empty.) and itself is not a data source, it does not need any sslot in this cycle. On the other hand, if a node's parent does not respond to its NOTI (i.e., `isConfirmed` is false), this node does not claim any sslot either. This node's parent might be out of synchronization, or even out of order. If this node claims any sslot and its parent is not receiving, all these sslots are wasted. It not only harms channel utilization, but also wastes energy. Note that even if a node does not need any sslot, it still broadcasts its schedule packets, in order to inform its neighbors that this node is finalized and the sslot it wins are now available for other nodes to contend.

However, according to `childList` and `isConfirmed`, `SchedC` can only know about whether there is traffic or not in this cycle, but `SchedC` is not able to know how much traffic this node is expecting. Therefore, `SchedC` also needs to know how much traffic it is expecting. We discuss how `SchedC` obtains the expected traffic in next section.

4.1.1.4 `TATDMacSchedulerC`

After SCHED period is over, it is the SLEEP period. `SchedC` has constructed the SLEEP schedule already. At the beginning of every sslot, `TATDMacSchedulerC` inquires `SchedC` about the assignment of this sslot.

As shown in Figure 4.3, there are three different kinds of sslot assignments, I_SEND, I_RECEIVE, and I_SLEEP. If its current sslot assignment is I_SEND, `TATDMacSchedulerC` turns the radio on, gets a packet from its packet queue and transmits it. `TATDMacSchedulerC` repeats this process until this sslot is over. If this sslot assignment is I_RECEIVE, `TATDMacSchedulerC` turn on the

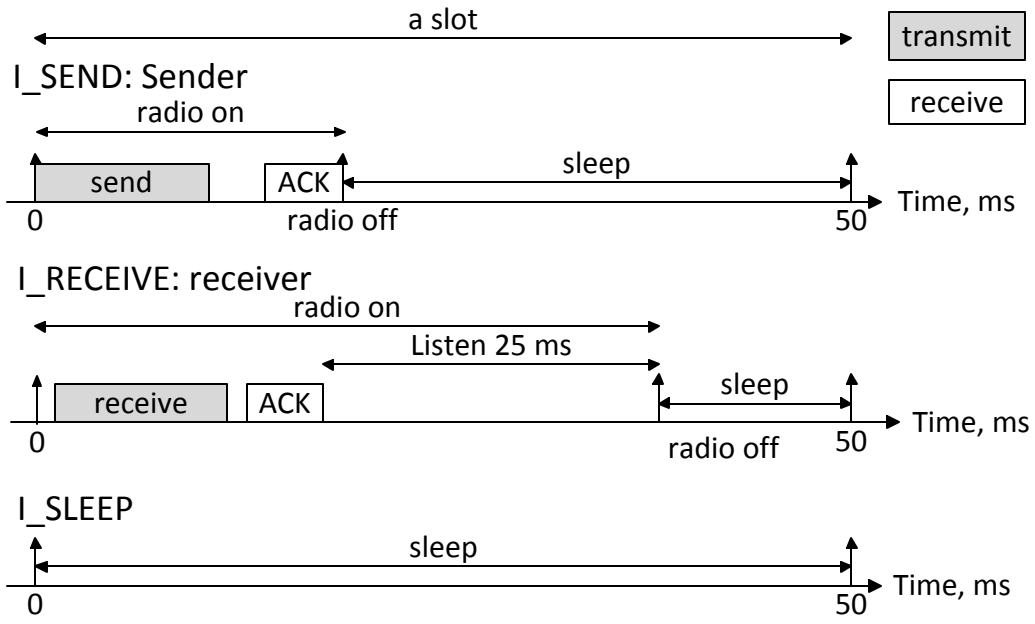

Figure 4.3: The timeline of an sslot in TATD-MAC.

radio, listens to the radio channel, and expects to receive packets. If there is no incoming packet for a certain amount of time (see Table 4.1 `END_RECEIVE_TIME`), `TATDMacSchedulerC` turns off the radio and goes back to sleep. If the sslot assignment is I_SLEEP, `TATDMacSchedulerC` does nothing and remains sleeping.

`TATDMacSchedulerC` is also responsible for estimating the amount of traffic in a cycle for `SchedC`. We implement a simple traffic estimation method by using history traffic. During these sslot in SLEEP period, `TATDMacSchedulerC` records the number of incoming and outgoing packets. Based on the history of the amount of incoming and outgoing packets, a node estimates the traffic amount for the coming cycles. At the beginning of SCHED period, `TATDMacSchedulerC` not only informs `SchedC` about `childList` and `isConfirmed`, but also informs `SchedC` a prediction of traffic in this cycle.

4.1.2 Challenge

4.1.2.1 Generating Priority

The computation power of a mote is very limited. We measured that it takes a Tmote-Sky [19] mote 75μs to generate a random number using the TinyOS built-in `RandomC` component. For each sslot, a node has to generate tens of random numbers itself and its two-hop neighbors. The computation takes time, and the length of dslot in SCHED frame is only 50 milliseconds. It is not feasible to compute the priority right before constructing schedules. We overcome this challenge by generating the priorities of cycle L in cycle $L-1$ during SLEEP period.

4.1.2.2 Optimized Concatenation

Concatenation is also performed quite often while generating priorities. Intuitively, we use `sprintf` to writes two operands into a buffer and then convert the buffer back to an integer by using `atoi`. Although this method is the most general way to do concatenation, it is extremely slow and unnecessary. In our implementation, due to the limited memory on Tmote-Sky, a priority value is a 16-bit unsigned integer. The concatenation operands are always a 16-bit unsigned integer and an 8-bit unsigned integer. Since both of the operands are unsigned integers, instead of using a general method (`sprintf` and `atoi`), we optimize our special case concatenation using multiplication and summation. For example:

$$300 \oplus 5 = 300 * 10 + 5 = 3005 \tag{4.1}$$

$$300 \oplus 10 = 300 * 100 + 10 = 30010 \tag{4.2}$$

4.1.3 The Length of the Schedule Constructed by `SchedC`

In our implementation, a cycle is composed of 1280 slots, including dslot and sslot. For each dslot in SCHED period, a node does the following things: claim the sslots belonging to it, make a

schedule packet, and broadcast the schedule packet. However, a dslot in a SCHED frame is merely 50 milliseconds in length. It is impossible for a node to scan through the priority of whole 1280 slots and claim every sslots this node wins within such a short time. Not to mention a node has to broadcast its schedule to its neighbors.

Our solution to this problem is to construct a shorter schedule and we repeat it several times. In our implementation, we only construct the schedule of thirty-two sslots and repeat it forty times to make a 1280 sslots schedule. This approach not only significantly reduces the amount of computation for constructing a scheduling packet in the SCHED period, but also reduces the amount of computation to generate priorities and the memory consumption by storing priorities. Moreover, this approach does not have any obvious drawbacks. Instead, repeating a schedule brings us a desirable characteristic, that the nodes win sslots in a more distributed way. We will discuss more about this in Section 4.1.4.

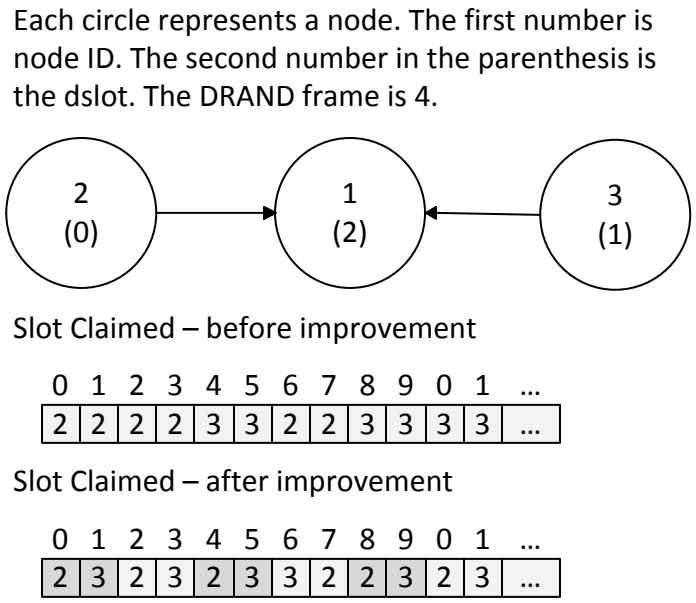

Figure 4.4: The sslot claimed that are not distributed. The dark blocks in slot claimed - after improvement, denotes that this node claims these sslot based on its dslot.

4.1.4 Claiming sslots in a Distributed Fashion

Consider a simple topology shown in Figure 4.4. Node 2 and node 3 are transmitting to node 1, the root node. The DRAND frame is 4 and the dslot of node 1, node 2, and node 3 are 2, 0, and 1, respectively. Node 1 does not transmit, only node 2 and node 3 need to claim sslots. The schedule constructed by `SchedC` before improvement might happen. Node 2 wins consecutive sslots from sslot 0 to sslot 3 and from sslot 6 to sslot 7, while node 3 wins consecutive slot from sslot 6 to sslot 7 and sslot 8 to sslot 11. Although the number of slot node 2 and node 3 claimed are still fair and the channel utilization is still high, the end-to-end latency is still high. Suppose that node 2 and node 3 generate exactly one packet per sslot, from sslot 0 to sslot 3, there will be 4 packets being blocked in the queue in node 3.

We came up with an improvement by decomposing the whole SLEEP period into repetitive DRAND frames. For each DRAND frames inside of the SLEEP period, the owner of the dslot has the highest priority to own the sslot. In Figure 4.4, sslot 0 to sslot 11 are decomposed into three DRAND frames. The dslot of node 2 is 0. Node 2 owns the first sslot of all three DRAND frames, sslot 0, sslot 4, and sslot 8. The dslot of node 3 is 1. Node 3 owns the second sslot of all three DRAND frames, sslot 1, sslot 5, and sslot 9. In this way, node 2 and node 3 only contend for sslot 2, sslot 3, sslot 6, sslot 7, and so on. In best case, node 2 and node 3 do not claim any consecutive sslot. In worst case, a node claims at most 3 consecutive sslots.

Note that we do not achieve this by forcing a node to occupy an sslot regardless of the priorities. Instead, we improve the priority generating part in `SchedC` by setting the priority of sslot i of node k to maximum priority if sslot i is mapped to the dslot of this node k. For each node k, the priority of sslot i is redefined as following equation:

$$p^{i}_{k \oplus j \oplus l} = max\ priority \quad \text{if } i \,\%\, dframe \equiv dslot_k$$

$$p^{i}_{k \oplus j \oplus l} = [rand(k \oplus i + rand(l))]_j \oplus k, k \in N2(u) \cup u, j = 1, ..., n_k \quad \text{otherwise} \tag{4.3}$$

where $dframe$ is the DRAND frame size and $dslot_k$ denotes the dslot of node k.

Such design has several advantages. First, nodes are guaranteed to claim sslots in an even more distributed fashion. Second, the computation time `SchedC` spends to generate priority is further reduced. Third, a node can still discard every sslot it does not need by declaring itself finalized so that any other node that needs more sslot can still claim those sslots.

4.1.5 Minimum sslot a Node Has to Claim

The minimum sslot a node has to claim is a two-sided blade. The delay is reduced if more sslots are claimed, but the power consumption is inevitably increased.

TATD-MAC estimates the incoming packet rates of future cycles using history traffic. As mentioned earlier, `SchedC` constructs a schedule of 32 sslots and repeats it. We can only claim exactly one sslot out of 32 if the data rate is low. Claiming as few sslots as needed can substantially improve energy efficiency in low data rate. However, less sslots also means higher end-to-end delay, which is not desirable.

As a result, we improve our TATD-MAC prototype by demanding every node to claim at least one sslot out of a DRAND frame. This limit guarantees low delay without increasing power consumption compared with conventional TDMA in low data rate.

4.2 TDMA Prototype with or without Slot Stealing

In this section, we give an introduction to our TDMA prototype and the slot-stealing component. We implement TDMA prototype as a comparison baseline. Moreover, we implement the slot stealing technique which is adopted by Z-MAC [9] to improve channel utilization. Table 4.2 show the default values of TDMA prototype's parameters. We still use the term *sslot* to denote a slot in SLEEP period.

Table 4.2: The default TDMA prototype parameters

Parameter	Value
slot size	50 milliseconds
Number of slot in a cycle	1280 slots / 64 seconds
Maximum neighbor count	20 nodes
Maxmum one-hop neighbor	8 nodes
SYNC period	1 DRAND frame
Total sslot	1280 - DRAND frame
Synchronization frequency	once per cycle
Synchronization protocol	FTSP
Maximum child count	7 nodes
END_RECEIVE_TIME	25 milliseconds
`TDMAScheduler`C queue size	200
Synchronization phase length	120 seconds for 10 nodes (depend on network size)
Setup phase length	240 seconds for 10 nodes (depend on network size)

4.2.1 Structural Overview

Although TDMA is simple and straightforward, we still follow the way we introduce TATD-MAC to introduce TDMA. Our implementation of TDMA also takes advantage of DRAND to obtain the sslot assignment of each node.

There are also three phases in our TDMA protocol, synchronization phase, setup phase, and working phase. The synchronization phase and setup phase is the same TATD-MAC, described in Section 4.1.1.

In working phase, time is divided into repetitive cycles. In TDMA protocol, a cycle is only divided into SYNC and SLEEP period. Therefore, we only need a `SyncC` component and one `TDMAScheduler`C. After DRAND is done, `DrandC` informs `TDMAScheduler`C about the DRAND assignment and the neighbor information. `TDMAScheduler`C starts to work as a repetitive cycle. For each sslot during the cycle, `TDMAScheduler`C determines what this node should do and controls the corresponding components.

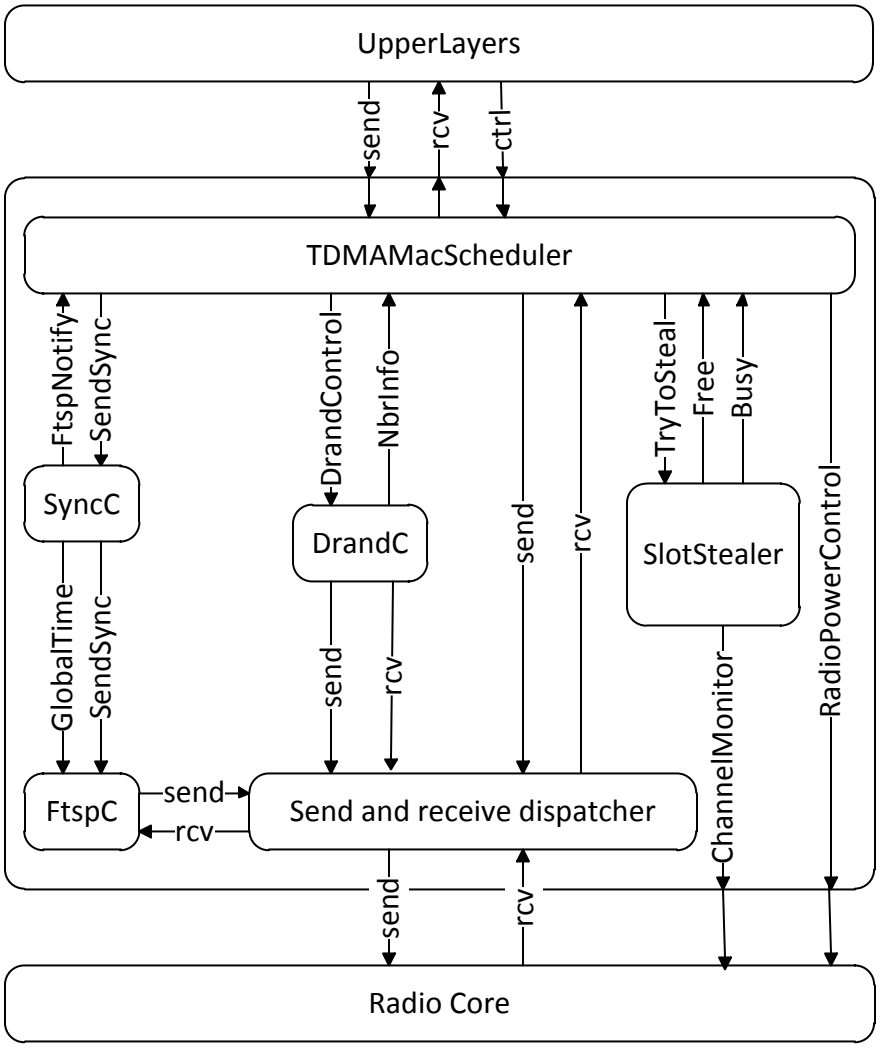

Figure 4.5: The composition graph of TDMA with slot stealing prototype.

4.2.1.1 `TDMAScheduler C`

After DRAND informs `TDMAScheduler C` about the DRAND assignment, `TDMAScheduler C` simply repeat the DRAND frame to complete the whole 1280 slots. Meanwhile, DRAND also provides all two-hop neighbors' DRAND slot assignments. A node not only knows about its DRAND assignment, but also its parent and child's assignment. A node wakes up at its DRAND slot to transmit and wakes up at its child's DRAND slot to receive from its child.

In TATD-MAC, for each sslot, the `TATDMacScheduler C` follows the schedule constructed by `Sched C` and only perform one kind of job in an sslot, either I_SEND, I_RECEIVE, or I_SLEEP. In TDMA without slot stealing, the `TDMAScheduler C` also follows the DRAND assignment and only does one kind of job, either I_SEND, I_RECEIVE or I_SLEEP. However, in TDMA with slot stealing, the `TDMAScheduler C` tries to use an sslot if the owner is not using it or is done using it.

4.2.2 `SlotStealer C`

As shown in Figure 4.6, the timeline of an sslot is more complex compared with an sslot in TATD-MAC shown in Figure 4.3. At the beginning of each sslot, if the assignment is I_RECEIVE or I_SLEEP, `TDMAScheduler C` try to steal the sslot by commanding the `SlotStealer C` to do a *clear channel assessment* (CCA). `SlotStealer C` commands the radio core to do CCA by using the `ChannelMonitor` interface. The `SlotStealer C` keeps doing CCA for 15 milliseconds plus a small random back off time and signal an event to notify `TDMAScheduler C` whether the channel is free or busy.

If the channel is free, it means that every one-hop neighbor is not transmitting, since an owner of the sslot definitely starts to transmit at the very beginning of the sslot. After `SlotStealer C` informs `TDMAScheduler C` that the sslot is free, `TDMAScheduler C` starts to transmit packets right away.

If the channel is busy, it means that either the owner is still using the channel or some other node has already stole the channel and is transmitting. Even if the channel is busy, a node should

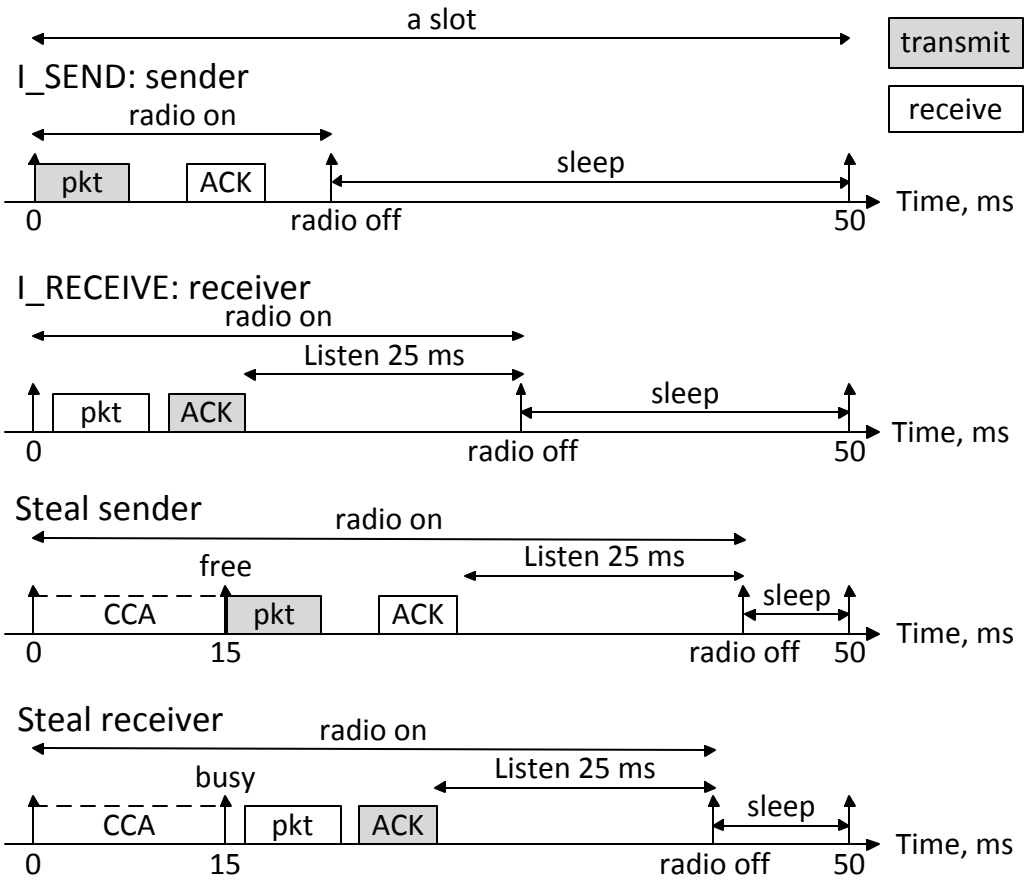

Figure 4.6: Time line of an sslot in TDMA with slot stealing.

still listen to the channel for a certain amount of time, because if the node who stole the sslot is the child of this node, this node is the intended receiver.

4.2.3 Challenge

4.2.3.1 How Much Should We Steal?

Consider the topology shown in Figure 4.7, node 2 and node 3 transmit to node 1. Node 2 and node 3 are out of each other's transmission range.

Suppose a stealing sender now sends as many packets as possible. If both node 2 and node 3 try to steal an sslot for transmission, both of their `SlotStealerC` thinks the channel is free and

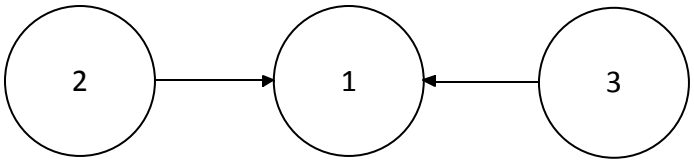

Figure 4.7: A simple topology composed of three nodes.

their `TDMAScheduler C` think they can steal the sslot. Both node 2 and 3's packets collide and node 1 receives nothing. In even worse case, node 2 is the owner of this sslot, but node 3 does not know the channel is occupied by node 2 and node 3 steals the sslot. The sslot owned by node 2 is also wasted due to collision. In this case, the slot stealing of node 3 does not improve throughput, instead, it harms.

This problem is in fact *the hidden terminal problem*. In Z-MAC [9], this problem does not affect as much as it does in our case, because Z- MAC is built on top of B-MAC. Whenever a node with B-MAC is transmitting, there is a random back off time and a congestion back off time. It is still possible to collide at node 1, but the chance is much lower in B-MAC than in TDMA. We cannot be overconfident about the CCA and transmit several packets in a single stolen sslot. Instead, we limit a node which thinks it has stolen the sslot to transmit only one packet and the retry limit is 3. If this stealing sender does not get the acknowledgement within three transmits, it stops and gives up the sslot it stole.

We have done a simple throughput test using two motes. The more detailed results are listed in Table 5.2 in Section 5.1.1.1 in page 44. In our TDMA implementation, the software acknowledgement is enabled. In a single sslot, which is 50 milliseconds long, a node can transmit 3 to 10 packets. Therefore, stealing only 1 packet in a stolen sslot is still helping the performance.

Chapter 5

PERFORMANCE EVALUATION

In this chapter, we evaluate our TATD-MAC prototype under different packet generation rates in a random topology. We also conduct the same experiments on our TDMA prototype, with or without slot stealing in order to compare the performance of all three prototypes. Moreover, we conduct another experiment to demonstrate the fairness of our modified NCR priority generation algorithm in Section 5.2.

5.1 Evaluation in a Random Topology

We first give a brief overview about the experiment topology and the experiment setting. We then examine the performances of TATD-MAC, TDMA and TDMA with slot stealing from several viewpoints, including throughput, duty cycle, and end-to-end latency. We also examine the traffic adaption feature of TATD-MAC in Section 5.1.5.

5.1.1 Experiment Overview

We evaluate our TATD-MAC prototype in a simple yet realistic and representative test bed composed of 13 Tmote-Sky motes, as shown in Figure 5.1. Node 1 is the data sink. All leaf nodes are data sources, except for node 4 and node 13. Every parent-child pair is at least 35 centimeters away from each other. We set CC2420 radio power level to minimum -25 dBm [20] so that the transmission range of Tmote-Sky is about 50 centimeters. Leaf nodes, such as node 9, node 10, and node 11 are close to each other, but it does not affect anything because they are all two-hop neighbors and they are guaranteed to get different dslot. Note that node 4 and node 13 do not generate data. They do not initiate NOTI and they do not claim any sslot, but they still wake up during the control frames and they do give up every sslot they win.

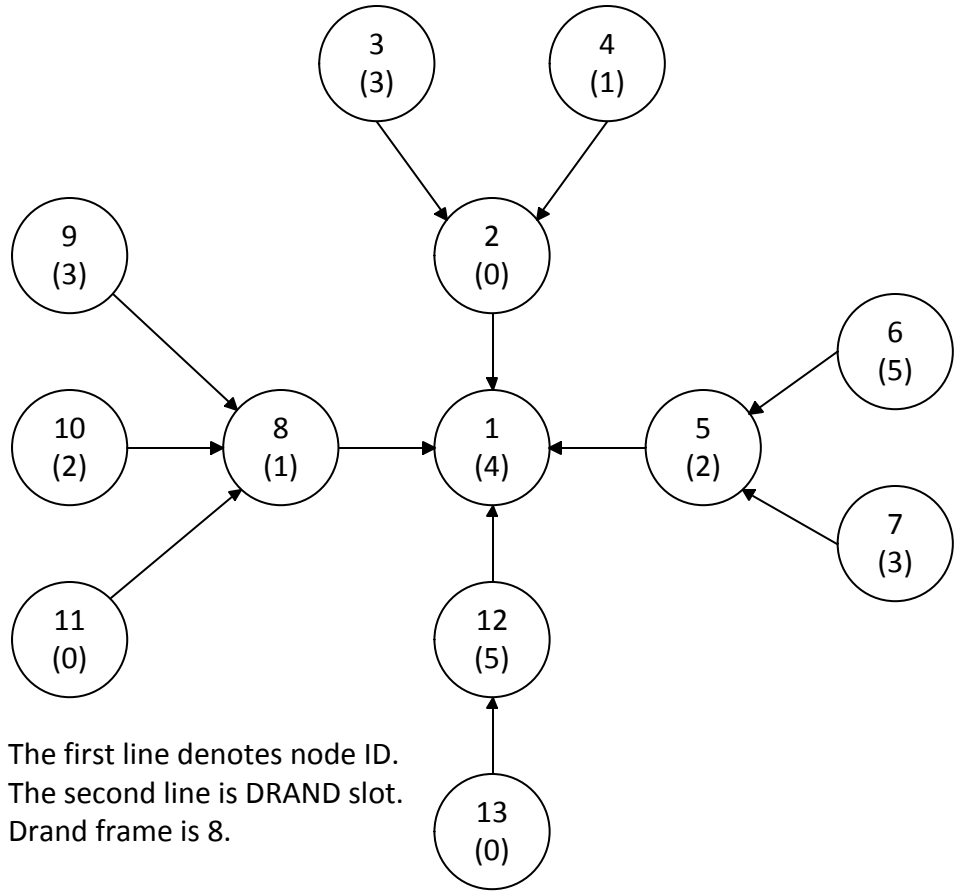

Figure 5.1: Experiment topology.

Although DRAND generates a DRAND frame of 8 due to the fact that node 1 has 4 children, the DRAND frame is still much lower than local neighbor count, which is 13. In order to compare our prototypes without being affected by DRAND results, we save the DRAND assignment and use the same DRAND assignment for all three prototypes.

The experiment parameters are listed in Table 5.1. We conduct five different experiments on each of the three prototypes, 1 packet per second, 2 packets per second, until 5 packets per second.

Table 5.1: The experiment parameters

Parameter	Value
CC2420 radio power level	-25 dBm, minimum
Packet generation rate	1 to 5 packet/second
Minimum sslots a node has to claim in a cycle	160 sslots
Packet size	128 bytes
slot size	50 milliseconds
Number of slot in a cycle	1280 slots / 64 sec
The length of schedule being constructed	32 sslots
The times the schedule being repeated	40 times
Maximum neighbor count	20 nodes
Maxmum one-hop neighbor	8 nodes
SYNC period	8 slots
RESV period	8 slots
SCHED period	24 slots
Total sslot	1240 slots
Synchronization frequency	once per cycle
Synchronization protocol	FTSP
NOTI retry limit	10 times
Maximum child count	7 nodes
END_RECEIVE_TIME	25 milliseconds
`TATDMacSchedulerC` queue size	200
Synchronization phase length	180 seconds
Setup phase length	360 seconds

5.1.1.1 Theoretical Throughput Bound

We also conduct a simple throughput test using two motes. The sender keeps sending packets to the receiver as fast it can. The CC2420 transmission power level is set to max (0 dBm). We test the throughput of a mote transmitting packets with 0 bytes payload and 114 byte payload. The results are listed in Table 5.2.

Table 5.2: The throughput test.

Software Ack	CCA/back off	Payload 0 bytes throughput	payload 114 bytes throughput
on	on	97 to 102	47 to 49
on	off	199	67
off	on	96 to 100	46 to 50
off	off	466	79

In our implementation prototypes, software acknowledgement is enabled and the payload is 114 bytes. In this throughput test, the throughput of payload of 114 bytes is 67 packet/second. Even if we assume there is no overhead to divide time into tiny time slots, a sender is able to transmit about 3.35 packets in a 50 milliseconds time slot.

For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this thesis.

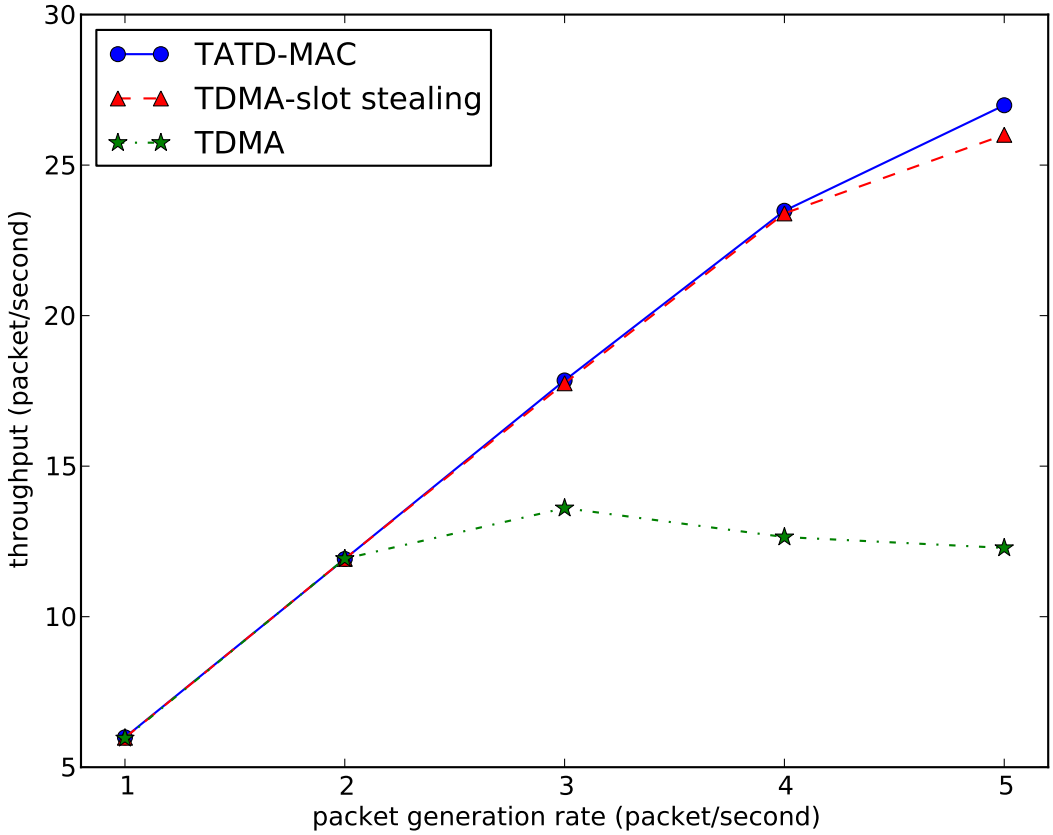

Figure 5.2: The throughput at the data sink, node 1. For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this thesis.

5.1.2 Throughput

The throughput results are shown in Figure 5.2. These throughputs are the number of packets received at the data sink, node 1. The packet received at the data sink is 6, 12, 18, 24, and 30 packets per second respectively. TATD-MAC's throughput outperforms the other two protocols on all experiments. TDMA with slot stealing's throughput is slightly less than TATD-MAC. TDMA fails to handle high throughput, because its schedule is fixed and cannot adapt to different traffic loads.

As we can see in Figure 5.2, TDMA only survived packet generation rate of 2 packets per second. TDMA simply uses DRAND assignment and repeats the DRAND frame to make a 1280-slots-schedule. Let us assume a node can transmit 3 packets per slot. Each node owns a dslot out of a DRAND frame, which means a node can transmit at most 3 packets per DRAND frame. In our topology, the DRAND frame is 8. Each second has 20 slots, which is equal to 2.5 DRAND frames. The maximum throughput on a node is cut down to 7.5 packets per second.

Node 8 has three children. Suppose the packet generation rate is r packets per second. The packet incoming rate on node 8 is $3r$ packets per second. However, the maximum packet outgoing rate is 7.5 packets per second.

$$3r \leq 7.5 \equiv r \leq 2.5 \tag{5.1}$$

Theoretically, node 8 is the throughput bottleneck and the throughput of TDMA should start to fall after 2.5 packets per second, which is confirmed by our experiment.

Let us examine our TATD-MAC prototype, which also fails to achieve ideal throughput in the experiment of 5 packets per second. Under the packet generation rate of 5 packets per second, a leaf node needs 200 sslots in a cycle. The leaf nodes are able to claim the sslots they need, since we have guaranteed a node can win at least one sslot out of a DRAND frame as described in Section 4.1.4. However, the intermediate nodes are not able to claim the sslot they need in this experiment. Node 2 has a child and it needs 200 sslots as well. Node 5 has two children and it

needs 400 sslots. Node 8 needs 600 sslots. Node 2, node 5, and node 8 are contending for the sslot to transmit to node 1, while their children are also contending for sslots. Node 5 and node 8 are not able to claim the sslot they need and they are not able to deliver the packet generated by their children. Therefore, the throughput is not able to achieve the ideal value of 30 packets per second.

TDMA with slot stealing also works well, but it is not able to achieve as high throughput as TATD-MAC. Whenever a node steals an sslot, it sends at most one packet only, while the sslots claimed in TATD-MAC are fully utilized.

5.1.3 Radio-on Percentage

The average radio-on percentage results on all nodes are shown in Figure 5.3. We measure the percentage of time a node turns its wireless communication component on, which is the main source of power consumption. TATD-MAC outperforms TDMA and TDMA with slot stealing. Although TDMA has lower radio-on percentage in higher data rate experiments, TDMA failed to achieve the same throughput as TATD-MAC.

Every node has to claim at least one sslot out of a DRAND frame as described in Section 4.1.5, which means TATD-MAC is active at least the same amount of sslot as TDMA during SLEEP period. In this experiment, a TDMA cycle is 8 control slots and 1272 sslots. Each node wakes up exactly $8+\frac{1272}{8}=167$ slots. A TATD-MAC cycle is 40 control slots and 1240 sslots. Each node wakes up at least $40+\frac{1240}{8}=195$ slots. TATD-MAC's radio-on percentage is still lower than TDMA, because node 4, node 12, and node 13 do not have packet to transmit and do not wake up during SLEEP period. The radio-on percentage of each node of TATD-MAC is still slightly higher than that of TDMA, except for node 4, node 12, and node 13. The percentage of time TDMA with slot stealing turn on the radio is significantly higher than TATD-MAC due to CCA and listen time.

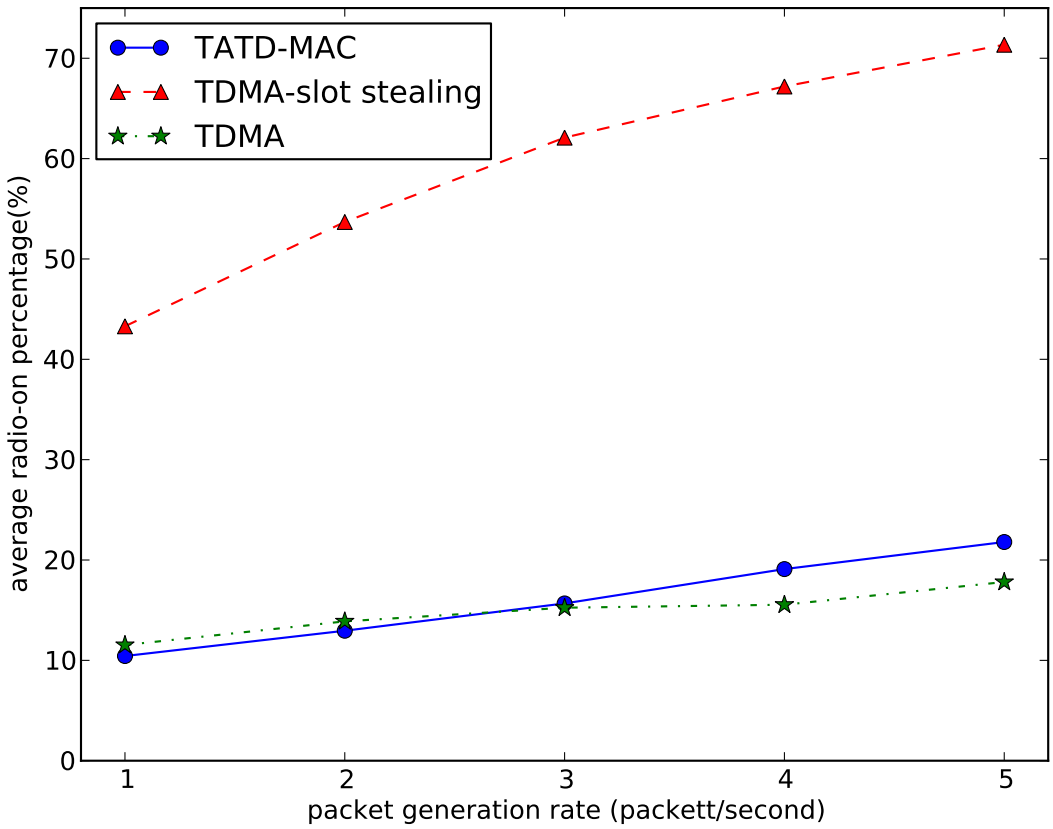

Figure 5.3: The average duty cycle on all nodes.

5.1.4 End-to-end Latency

The end-to-end latency comparison chart is shown in Figure 5.4. Although there are 5 different packet generation rates and we have three prototypes, we only show 10 experiment results in this figure. It is because in those experiments, the network is not able to deliver such a heavy data rate and we observe significant packet loss due to queue overflow. Even for the packets that were delivered to the data sink, they are blocked in certain nodes' packet queues and they are delivered at the data sink with very high end-to-end delays.

As we mentioned in last section, there are 40 control slots in TATD-MAC. During these control slots, all nodes do not transmit/receive any packet. It is inevitable to delay the packets in the packets queues during those control slots. There are 40 slots in the control frames, which means

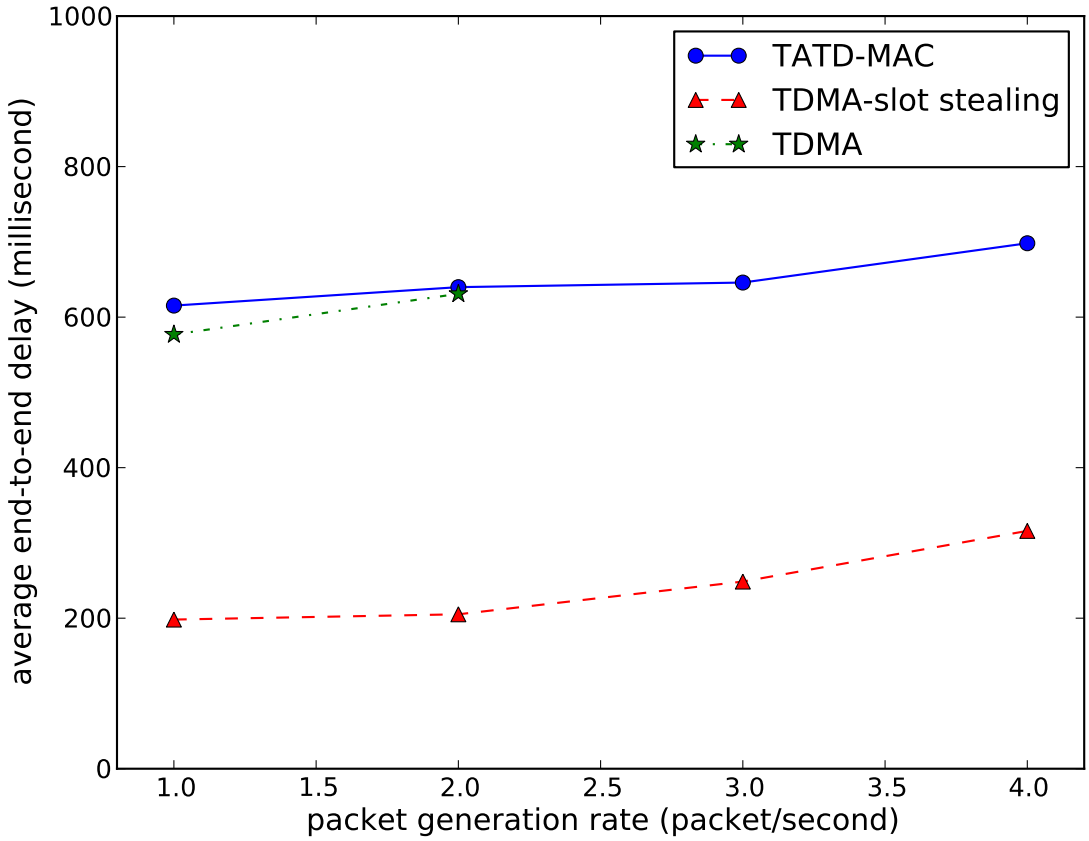

Figure 5.4: Average end-to-end delay of all packets received at the data sink.

if a packet is blocked, the delay drastically increases 2000 milliseconds. Although the chance of a packet being blocked during control slots is low, any packet get blocked in the packet queue has abnormally high delay.

On the other hand, TDMA with slot stealing has significantly lower delays, because in low data rate, successful steals happen very often. The amount of time packets being blocked in the packet queue is significantly reduced and therefore the end-to-end delay is reduced. As we can see in Figure 5.4, the delay of TDMA with slot stealing is increasing in a higher rate than TATD-MAC as packet generation rate increases. As the data generation rates at the leaf nodes increase, more nodes start to contend for unused slots and the chances of successful steal decrease. Even though the delay of TDMA with slot stealing is lower, TATD-MAC still outperforms TDMA-slot stealing

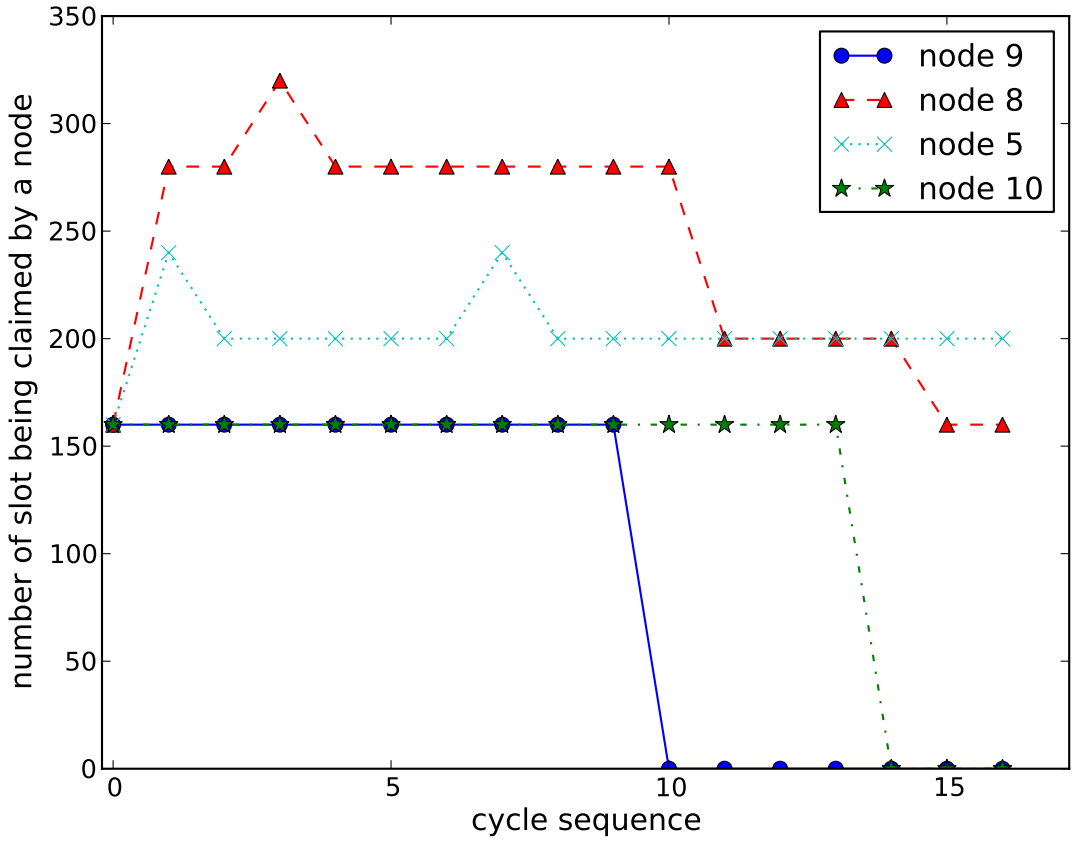

Figure 5.5: The number of sslots node 5, node 8, node 9, and node 10 claimed.

in throughput and radio-on percentage by 70%.

5.1.5 Traffic Adaption

Figure 5.5 shows the number of sslots claimed by node 5, node 8, node 9, and node10 of TATD-MAC in packet generation rate of 3 packets per second. At the beginning of the system, nodes know nothing about neighboring traffic. Therefore, they claim only the minimum amount of sslots, i.e. 160 sslots.

After the second cycle, nodes estimate the traffic from the history of previous cycle. Node 8, which has more children, claims more sslots than node 5. Note that node 9 and node 10 are children of node 8. We remove node 9 between cycle 9 and cycle 10. During the control frame of

cycle 10, node 8 is not yet aware of the removal of node 9. Throughout cycle 10, node 8 sees that the incoming traffic to it has decreased. Since the traffic in cycle 10 decreases, node 8 knows that it needs fewer sslots in cycle 11. After node 9 is removed, node 8 has 2 children, which is the same as node 5. Therefore, node 8 claims the same number of sslots as node 5 from cycle 11 to cycle 14. Then we remove node 10 as well in cycle 14. Node 8 has only one child left, and it only needs the minimum sslots per cycle, 160 sslots per cycle.

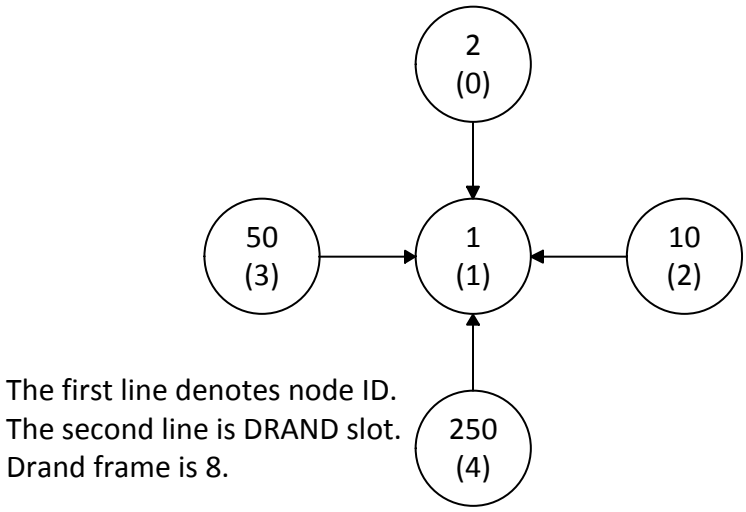

Figure 5.6: Fairness experiment topology.

5.2 Fairness of Slot Claiming

In this section, we conduct another experiment using TATD-MAC prototype to show that our modified NCR enables the nodes win sslots fairly, especially regardless of node ID. The experiment topology is shown in Figure 5.6. Node 1 is the data sink and all other nodes are data sources. The length of schedule constructed by `SchedC` is still 32 sslots and repeated for 40 times. The nodes claim as many sslots as possible.

The result is shown in Figure 5.7, the nodes contend for 1280 sslots in a cycle. On average, node 2, node 10, node 50, and node 250 claims 317.0, 339.51, 302.44, and 329.78 sslots per cycle.

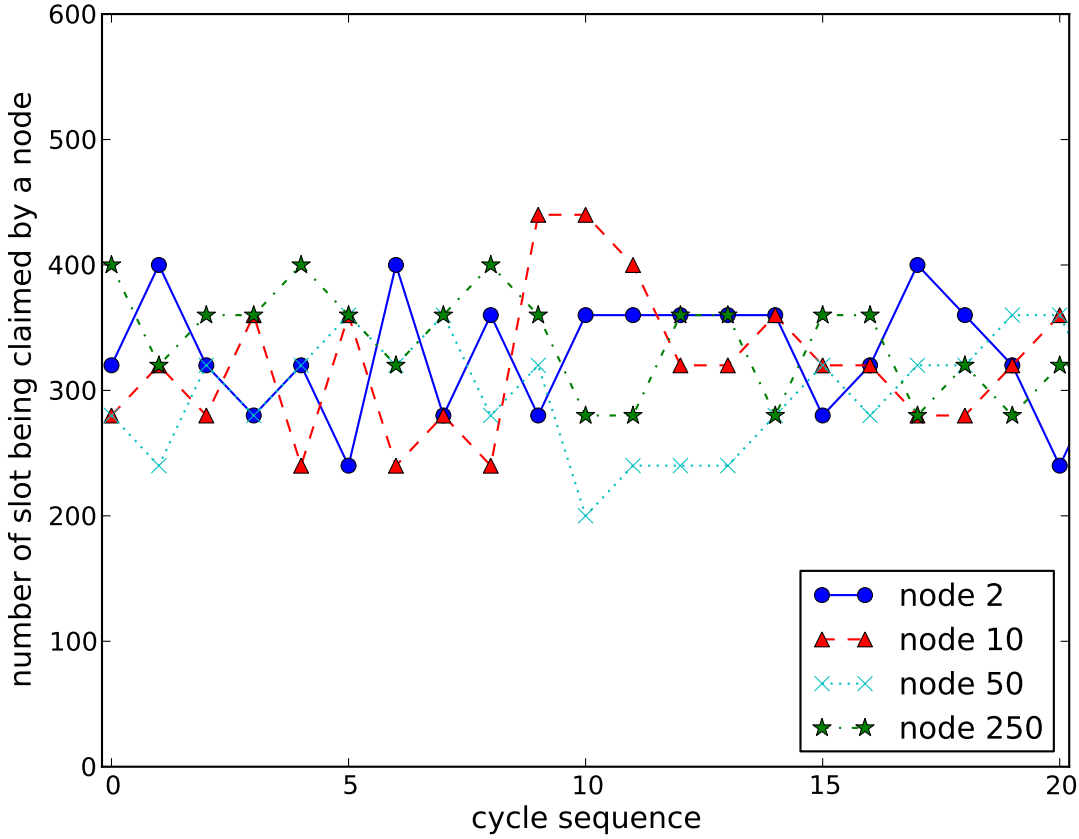

Figure 5.7: Fairness experiment result.

The average number of sslots each node claimed per node is not as biased as it seems. Note that the schedule constructed during SCHED period is repeated for 40 times. Therefore, a node claimed 320 sslots in a cycle means it wins 8 sslots out of 32 sslots in the schedule TATD-MAC contructed. Therefore, on average, node 2, node 10, node 50, and node 250 claims 7.93, 8.49, 7.56, and 8.24 sslots per 32-sslot-schedule respectively.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

High throughput and low power consumption are both critical issues in wireless sensor networks. This thesis proposes a novel MAC protocol, called TATD-MAC, which adapts itself from low data rate to high data rate with low overhead. The two-phase control packet design allows us to achieve a per-flow based traffic notification and a low overhead energy-efficient scheduling. Through our innovative scheduling method, TATD-MAC achieves the traffic adaption and high throughput with slightly increased delay. The fast traffic notification control packet informs the node on the routing path about the incoming traffic. When there is no incoming traffic, a node simply claims zero sslot and stays sleeping throughout the whole SLEEP period. When there is traffic, the traffic-adaptive characteristic of TATD-MAC brings us several advantages. Under low data rate, TATD-MAC starts to claim only minimum number of sslots it needs to deliver the traffic it is expecting. In higher data rate, TATD-MAC adjusts itself to claim more sslots to increase channel utilization and achieves higher throughput.

From the experiments we conducted, TATD-MAC is able to sustain high data rates and achieves higher throughput, compared to TDMA and TDMA with slot stealing. Moreover, TATD-MAC is more power efficient under all data rates. Nodes on the routing path that do not have traffic do not wake up at all. As the throughput increase from 1 packet per second to 5 packets per second, the radio-on percentage only increased from about 11 percent to less than 22 percent. Compared with TDMA, we see that the traffic adaptive characteristic of TATD-MAC makes TATD-MAC survive much higher traffic loads. We can see that our novel slot scheduling method increases channel utilization and throughput while consuming 70 percent less than TDMA with slot stealing.

We summarize the contribution of this work as follows:

- TATD-MAC is a novel, low overhead, low duty cycle, synchronous MAC protocol that inherits the high throughput characteristic of TDMA protocols.
- TATD-MAC assigns time slots only to nodes with traffic, which further improves the energy-efficiency. Our traffic-adaptive data transmission scheduling technique maximizes the channel utilization with minimal overhead.
- The two-phase control packets design that derives a per-flow traffic notification and the scheduling method has a lower overhead than traditional per-packet based solutions.
- We propose a simple yet efficient schedule representation and exchange method that further improves the channel utilization with extremely low overhead.

6.2 Future Work

Our implementation prototype has demonstrated that our data transmission scheduling mechanism achieves high throughput and low power consumption with low overhead. There is several potential ways to further improve performance.

Currently, TATD-MAC relies only on history traffic to estimate the traffic for future cycles. As we mentioned above, nodes in the first cycle can only claim the minimum amount the sslots, because they know nothing about their traffic. Moreover, in our traffic adaption experiment in Section 5.1.5, node 8 can only perceive the change of incoming data rate in next cycle. A possible improvement is to add an *expected traffic* field into traffic notification packets. When the source node initiate a NOTI, they not only inform the routing path that there will be traffic in this cycle, but also they inform the routing path about how much traffic they need. An intermediate node with more than one child can aggregate the expected traffic of its children and transmit the NOTI with aggregated value to its parent. The traffic adaption can be achieved in the current cycle, instead of in a future cycle.

Another possible improvement is trying to learn history traffic pattern. In many applications, such as weather monitoring, the data generation rate is periodic. In our experiments, the data sources also generate packet in a periodic fashion. We observed that the traffic a node receives from its child has certain patterns. If it is possible to learn the pattern, an intermediate node might be able to give up the sslots if it is not likely to have incoming data. For example, in our 1 packet per second experiment, claiming more than 1 slot per second is not helping performance. In our prototype, a node has to claim at least one sslot out of the DRAND frame, which is equal to 2.5 sslots per second in our experiment. If learning the traffic pattern is possible, an intermediate node might be able to predict future incoming packets and claim the sslot that is closest to next packet's expected arrival time in order to forward this packet as soon as possible.

Inspired by PIP [23], a hybrid MAC protocol based on TDMA and frequency division multiple access (FDMA), we think that using multiple channels in TATD-MAC might bring substantial improvement. Currently, TATD-MAC can only assign an sslot to a node if and only if all its two-hop neighbors are not using the sslot. If there are multiple channels, a node can still claim an sslot, even if a two-hop neighbor of this node has already claim the sslot, as long as this node is using another channel. Nodes can inform each other through the existing three scheduling frames using modified scheduling packet format, that include the channel assignments.

The first possible future work is more straightforward and most realistic. We need to analyze and investigate the traffic patterns in depth for the second option. Assigning and using multiple channel is much more ambitious and challenging. If these improvements can be implemented, our TATD-MAC will be even more traffic adaptive and will be able to achieve even higher throughput.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *Proceedings of the IEEE Infocom*. New York, NY, USA: IEEE, June 2002, pp. 1567–1576.

[2] S. Du, A. K. Saha, and D. B. Johnson, "Rmac: A routing-enhanced duty-cycle mac protocol for wireless sensor networks." in *INFOCOM'07*, 2007, pp. 1478–1486.

[3] 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 *MobiHoc'08*, 2008, pp. 53–62.

[4] 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*, ser. SenSys '04, New York, NY, USA, 2004, pp. 95–107.

[5] 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*, ser. SenSys '06, New York, NY, USA, 2006, pp. 307–320.

[6] A. El-Hoiydi and J.-D. Decotignie, "WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks," in *First Int. Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004), Lecture Notes in Computer Science, LNCS 3121*, July 2004, pp. 18–31.

[7] 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*, ser. SenSys '08, New York, NY, USA, 2008, pp. 1–14.

[8] 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*, ser. SenSys '03, New York, NY, USA, 2003, pp. 171–180.

[9] I. Rhee, A. Warrier, M. Aia, and J. Min, "Z-mac: a hybrid mac for wireless sensor networks," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, ser. SenSys '05, New York, NY, USA, 2005, pp. 90–101.

[10] W. Ye, J. Heidemann, and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 12, pp. 493–506, June 2004.

[11] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proc. USENIX OSDI*, 2006, pp. 381–396.

[12] G. Wittenburg, K. Terfloth, F. L. Villafuerte, T. Naumowicz, H. Ritter, and J. Schiller, "Fence monitoring: Experimental evaluation of a use case for wireless sensor networks," in *Proc. EWSN*, 2007.

[13] J. Koo, R. K. Panta, S. Bagchi, and L. Montestruque, "A tale of two synchronizing clocks," in *Proc. ACM SenSys*, 2009.

[14] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh, "VigilNet: An integrated sensor network system for energy-efficient surveillance," *ACM Trans. Sensor Networks (TOSN)*, vol. 2, no. 1, Feb 2006.

[15] I. Rhee, A. Warrier, J. Min, and L. Xu, "Drand: distributed randomized tdma scheduling for wireless ad-hoc networks," in *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, ser. MobiHoc '06, New York, NY, USA, 2006, pp. 190–201.

[16] S. Gobriel, D. Mosse, and R. Cleric, "Tdma-asap: Sensor network tdma scheduling with adaptive slot-stealing and parallelism," *Distributed Computing Systems, International Conference on*, pp. 458–465, 2009.

[17] L. Bao and J. Garcia-Luna-Aceves, "A new approach to channel access scheduling for ad hoc networks," in *In Proc. ACM Seventh Annual International Conference on Mobile Computing and networking*, 2001, pp. 210–221.

[18] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves, "Energy-efficient collision-free medium access control for wireless sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ser. SenSys '03, New York, NY, USA, 2003, pp. 181–192.

[19] "Telosb datasheet." [Online]. Available: http://www.xbow.com

[20] "Cc2420 datasheet." [Online]. Available: http://www.ti.com

[21] 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*, ser. SenSys '07, New York, NY, USA, 2007, pp. 59–72.

[22] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys '04, New York, NY, USA, 2004, pp. 39–49.

[23] B. Raman, K. Chebrolu, S. Bijwe, and V. Gabale, "Pip: a connection-oriented, multi-hop, multi-channel tdma-based mac for high throughput bulk transfer." in *SenSys'10*, 2010, pp. 15–28.