Ad Hoc Networks (2014), Vol. 25, Part A, pp. 54-71

DICSA: Distributed and Concurrent Link Scheduling Algorithm for Data Gathering in Wireless Sensor Networks

Behnam Dezfouli[1,2,*], Marjan Radi[1,2], Kamin Whitehouse[3], Shukor Abd Razak[1], Tan Hwee-Pink[2]

[1] *Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia (UTM), Malaysia*
[2] *Networking Protocols Department, Institute for Infocomm Research (I^2R), A*STAR, Singapore*
[3] *Department of Computer Science, University of Virginia, USA*

Abstract

Although link scheduling has been used to improve the performance of data gathering applications, unfortunately, existing link scheduling algorithms are either centralized or they rely on specific assumptions that are not realistic in wireless sensor networks. In this paper, we propose a distributed and concurrent link scheduling algorithm, called DICSA, that requires no specific assumption regarding the underlying network. The operation of DICSA is managed through two algorithms: (i) Primary State Machine (PSM): Enables each node to perform its own slot reservation; (ii) Secondary State Machine (SSM): Enables each node to concurrently participate in the slot reservation of its neighbors. Through these algorithms and a set of forbidden slots managed by them, DICSA provides concurrent and collision-free slot reservation. Our results show that the execution duration and energy consumption of DICSA are at least 50% and 40% less than that of DRAND, respectively. In terms of slot assignment efficiency, while our results show higher spatial reuse over DRAND, the maximum slot number assigned by DICSA is at least 60% lower than VDEC. In data-gathering applications, our results confirm the higher performance of DICSA in terms of throughput, delivery ratio and packet delay. We show that the network throughput achievable by DICSA is more than 50%, 70%, 90% and 170% higher than that of DRAND, SEEDEX, NCR and FPS, respectively.

Keywords: Scheduling algorithm, Interference, MAC

1. Introduction

The fundamental traffic pattern observable in sensor networks is to send the data sampled by the nodes towards a common destination called sink node. This many-to-one traffic pattern is referred to as *convergecast* [1–4]. In order to make accurate and quick decisions, data gathering applications usually require high delivery ratio with minimum end-to-end delay [5–8]. To this aim, data convergecast has been investigated and improved from various perspectives (e.g., tree structure [9, 10], data aggregation [11, 12], and channel access). From the channel access point of view, employing random access mechanisms results in significant packet collisions, which is the result of traffic direction, multihop transmissions and hidden-node problem. Since packet collisions reduce network performance in terms of effective throughput and delivery ratio [1, 13], scheduling algorithms have been proposed to eliminate the negative effects of collisions on the performance of data gathering applications [14]. In particular, in contrast to random access mechanisms, scheduled access mechanisms (a.k.a., time division multiple access (TDMA)) divide time into slots, and arbitrate channel access through slot assignment to the nodes or links. In *node scheduling algorithms*, transmission time slots are assigned to the nodes assuming each node's transmission should be received by all of its neighbors. In contrast, *link scheduling algorithms* aim to provide collision-free packet reception only at the intended receiver of each transmitter. Therefore, since link scheduling algorithms apply fewer constraints to slot assignment, they can potentially improve channel utilization, compared to node scheduling approaches [15, 16]. Besides, as the convergecast traffic pattern indicates an almost static child-parent relationship between nodes [1, 17], it justifies the benefits of employing link scheduling to improve data gathering performance [2, 3, 14]. Nevertheless, it has been shown that time slot assignment to the nodes or links of a graph is a NP-complete problem [18, 19].

Based on the decision type made for channel access scheduling, the existing scheduling algorithms can be categorized into probabilistic (e.g., [20–23]) and deterministic algorithms (e.g., [2, 3, 19, 24]). Using the probabilistic algorithms, nodes determine their behavior at each slot probabilistically (e.g., using a pseudo-random function). Therefore, although the main advantage of these algorithms is the ease of distributed implementation, they have

*Corresponding author:
Email address: dezfouli@ieee.org (Behnam Dezfouli)

570-8705/©2014 Published by Elsevier B.V.
http://dx.doi.org/10.1016/j.adhoc.2014.09.011

no guarantee on a node's channel access delay. Additionally, since each node is only aware of its two-hop neighborhood, priority chaining may appear and these algorithms cannot effectively utilize the channel. For example, a node may refrain from transmission while no node in its two-hop neighborhood is transmitting. Using the deterministic algorithms, either the transmission slots of each node within the frame is known [24], or the total data gathering duration is predetermined [2, 3]. However, while most of these algorithms are centralized (e.g., [25–29]), the rest (except DRAND [24]) rely on specific assumptions that are not realistic in sensor networks (e.g., the requirement to have an interference-free tree topology [30, 31]). In particular, while significant research has been conducted on the theoretical aspects of improving network capacity through link scheduling, much less attention has been paid to the design of practical scheduling algorithms.

Among the deterministic algorithms, DRAND [24] is a distributed implementation of RAND [18], and is suitable for networks with no significant mobility. This algorithm requires no specific assumption regarding the network, and the complexity of its execution duration and message exchange is $O(\text{maxN}^{1,2})$ (assuming no packet loss), where $\text{maxN}^{1,2}$ is maximum two-hop neighborhood[1]. In addition, DRAND employs node scheduling because it does not consider any particular traffic pattern. However, considering the convergecast traffic pattern, this algorithm cannot achieve the potential improvements of link scheduling. For example, even if the transmissions of two neighboring nodes to their parents do not cause packet collision, DRAND prevents concurrent transmission of these nodes (exposed node problem). Furthermore, using DRAND, when a node is applying for a slot reservation, no one-hop or two-hop neighbor of this node is allowed to apply for slot reservation, therefore, only those nodes with distance more than two hops can be concurrently involved in slot reservation. This low level of concurrency increases the execution duration and energy consumption of DRAND.

In this paper, we propose the DIstributed and Concurrent link Scheduling Algorithm (DICSA), which provides distributed link scheduling without requiring any specific assumption regarding the underlying network. This algorithm relies on network layer information and performs slot assignment based on child-parent relationship. DICSA is composed of two algorithms: The first algorithm enables each node to perform its own slot reservation, the second one enables the nodes to be involved in the slot reservation of their neighbors. In particular, in contrast with DRAND (which requires at least three-hop distance between those nodes applying for reservation), DICSA does not require any specific distance for concurrent slot reservation, and enables the nodes to be involved in the slot reservation of more than one neighbor at a time. This mechanism significantly reduces the execution duration and energy consumption of DICSA compared to DRAND. Additionally, it results in lower slot updating cost during the network operation. Both of the DICSA's algorithms manage a common set of *forbidden slots lists* to achieve collision-free slot assignment. Using these lists, each node is also aware of the slots in which it should receive from its children or send to its parent. Therefore, establishing the forbidden slots lists also simplifies MAC design to achieve energy efficiency. Considering probabilistic and deterministic link scheduling and node scheduling algorithms, we perform comprehensive performance evaluations from four main perspectives: (i) algorithm execution cost, (ii) slot assignment efficiency, (iii) slot updating cost, and (iv) data gathering performance. All these evaluations confirm the high performance of DICSA.

It is worth mentioning that child-parent packet transmission is not the only traffic pattern in sensor networks. For example, link estimation and route updates may require packet exchanges that do not follow the child-parent scheme [17, 32]. Nevertheless, the frequency of control packet transmissions is considerably lower than that of data packet transmissions; therefore implying the importance of collision-free unicast transmissions. Additionally, employing link scheduling algorithms in hybrid CSMA-TDMA MAC protocols enables collision-free unicast transmissions, as well as supporting other traffic types. Therefore, this paper does not aim to propose a sophisticated low-power MAC protocol, rather, the proposed scheduling algorithm can be used in the design of TDMA and hybrid MAC schemes. It should also be noted that this paper neglects data aggregation during convergecast; therefore, each generated packet should be individually delivered to the sink node. This type of convergecast is referred to as *raw-data convergecast* [3] and is different from the approaches proposed in [11, 12, 33].

The rest of this paper is organized as follows. Prior works are given in Section 2. We present the formal presentation and requirements of link scheduling and node scheduling algorithms in Section 3. The design and implementation of DICSA is described in Section 4. We present performance evaluation results in Section 5. Analysis of the storage requirement of DICSA is given in Section 6. We conclude in Section 7.

Table 1 shows the key notations used in this paper.

2. Related Work

Neighborhood-aware Contention Resolution (NCR) [20, 23] is a probabilistic scheduling algorithm. NCR requires the nodes to be aware of their two-hop neighborhood, which is referred to as the *contender set*. At each time slot, each node employs a pseudo-random function to determine its own priority and the priority of its contenders. A node is allowed to send in a time slot if it has the highest priority among its contenders. The major drawback of this scheme is its random channel access latency. In

[1] Notice that the set of the two-hop neighborhood of a sample node i, which is denoted by $\text{N}_i^{1,2}$, includes those neighbors that are one-hop or two-hop away from i. In other words, $\text{N}_i^{1,2} = \text{N}_i^1 \cup \text{N}_i^2$

Table 1: A summary of key notations

Description	Symbol
Number of nodes in the network	V
Set of the one-hop neighbors of node i	N_i^1
Set of the two-hop neighbors of node i	N_i^2
Set of the two-hop neighborhood of node i	$\mathrm{N}_i^{1,2}$
Maximum number of one-hop neighbors	maxN^1
Maximum two-hop neighborhood	$\mathrm{maxN}^{1,2}$
Maximum number of children per node	maxC
Reception slots	RS
Transmission slots	TS
Reception slots of one-hop neighbors	RSO
Transmission slots of one-hop neighbors	TSO
Transmission slots of two-hop neighbors	TST
Reservation status of node x	$S(x)$
Reservation status of node y at node x	$S(x)[y]$
One-way message delay	υ
Packet transmission duration [second]	T_{packet}
Contention window duration [second]	T_{CW}
Maximum number of slot reservations per node	κ

addition, channel capacity cannot be effectively used because each node decides about packet transmission based on its own two-hop neighborhood information. For example, assume node i is refrained from transmission due to the higher priority of node j, and node j is also refrained from transmission due to the higher priority of a node in its two-hop neighborhood. In terms of MAC design, it is hard to provide energy efficiency since a node cannot be sure about packet reception in a given time slot. A similar approach to NCR is SEEDEX [22], which is basically a link-scheduling algorithm. At each time slot, a node goes into the PT (Possible Transmit) or the L (Silent) state with probability p and $1-p$, respectively. When a node has a packet to send, it should wait for a slot in which it is in the PT state and its receiver is in the L state. Additionally, to reduce collision probability, sender node computes its transmission probability based on the number of one-hop neighbors of the receiver that are in the PT state (using a pseudo-random number generator). Although SEEDEX can potentially improve channel utilization over NCR, it suffers from packet collisions due to its probabilistic collision avoidance mechanism. Similar shortcomings can be identified for the approaches used in [34], [35], [36] and [21].

Durmaz Incel et al. [3] proposed scheduling algorithms for the convergecast minimization problem. However, they have assumed that the interfering links of the data collection tree are eliminated through mechanisms such as transmission power control or transmission frequency management. They proved that the minimum number of required slots to schedule this tree is equal to the maximum node degree. Furthermore, it has been shown that this bound can be achieved when the BFS algorithm is used for slot assignment. Generally, although there exists many scheduling algorithms for wireless networks, they usually suffer from specific assumptions that are not realistic in wireless sensor networks. For example, while some algorithms require tree topology (and cannot be used in general networks) (e.g., [30, 31, 37]), others rely on centralized operation (e.g., [25–29]). Gandham et al. [2] considered the slot assignment problem for convergecast duration minimization. Although the authors showed that their algorithm requires at most $3V$ slots for general networks, where V is the number of nodes, the validity of their algorithm depends on the following assumptions: (i) each node should generate exactly one packet, (ii) each node should be aware of the number of packets in its sub-trees, (iii) each node should know the number of its children, (iv) the tree structure should be the result of BFS traversal algorithm, (v) sink node should generate a conflict map (based on the nodes' connectivity information) and disseminate it to all the nodes. Furthermore, if the number of generated packets at each node is more than one, the number of packets that each node generates should be known at the initialization phase. In [15] a distributed link scheduling algorithm is proposed based on the Vizing's theorem. Link scheduling is performed in two phases: First, a distributed algorithm is used to assign colors to every edge of the graph. Then, directions are assigned to the links, and extra colors are used for those links without any feasible direction. The main drawback of this algorithm is its slot assignment inefficiency; the authors showed that at least $2\times(\mathrm{maxN}^1+1)$ slots are required when the maximum network degree is maxN^1. In this paper we refer to this algorithm as the Vizing-based Distributed Edge Coloring (VDEC).

Flexible Power Scheduling (FPS) [38, 39] is a tree-based scheduling algorithm that assigns transmission and reception time slots based on child-parent relationship. Assume node i is the parent of node j, establishing a child-parent slot between these nodes is as follows. At the beginning, node i marks all of its slots as idle. Then it randomly selects a slot to advertise the availability of one of its randomly selected idle slots. As long as node j has not reserved its required slot, it continuously listens to the channel to receive an advertisement packet from node i. When node j receives an advertisement packet, it performs time synchronization with node i and extracts the slot number in which node i expects to receive reservation request. During that slot, node j sends a reservation request, and expects immediate confirmation from node i. If node i receives multiple request packets, it responds to the first one; therefore, other children should reserve their slot in the next cycles. Nodes employ CSMA during the reservation phase. However, CSMA is also used during the normal network operation because FPS removes collisions partially. For example, node j can reserve slot o to send to node i only if this slot is not reserved by node i or the children of node i. Therefore, a one-hop neighbor of node i may be transmitting in that slot too. Although it can be argued that the lower bound for the number of required slots by FPS is $\mathrm{maxC}+3$, where maxC is the maximum number of children per node[2], the real number of

[2]Assuming each node reserves only one slot, a node requires: (i) maxC slots for its children, (ii) one slot for transmission to its parent,

slots may be higher because FPS does not cope with message losses during slot reservation. For example, if node j looses the confirm message sent by node i, node i has marked a slot reserved for node j, however, node j is not aware of its reservation and tries to reserve another slot. It is worth mentioning that in this paper, in order to provide a fair comparison between FPS and other scheduling mechanisms, we improved the operation of FPS to achieve the mentioned lower bound at each node. TreeMAC [40] is another child-parent based scheduling algorithm. However, this algorithm relies on each node's depth and the whole network topology. Therefore, when a change in the network traffic or topology happens, it may be necessary to update all the slot assignments towards the sink node.

DRAND (Distributed Randomized TDMA Scheduling) [24] is a distributed implementation of the RAND [18] algorithm for time slot assignment to nodes. Although DRAND's theoretical time complexity is $O(\text{maxN}^{1,2})$, where $\text{maxN}^{1,2}$ is the maximum two-hop neighborhood, it has been shown that its time complexity is $O((\text{maxN}^{1,2})^2)$, which is due to unbounded transmission delays. The operation of DRAND is as follows. At the beginning all the nodes are in the IDLE state. Each node tries to reserve a slot through entering the REQUEST state and starting a slot reservation round. In order to avoid multiple neighboring nodes enter the REQUEST state simultaneously, nodes run a lottery that its success probability is $1/\left(2\left|\text{N}_x^{noslot}\right|\right)$, where $\left|\text{N}_x^{noslot}\right|$ is the number of those one-hop and two-hop neighbors of node x that have not yet reserved their slot. When a sample node i wins the lottery, it moves into the REQUEST state and broadcasts a *request* message. When node j receives the *request* message, it moves into the GRANT state and sends a *grant* message (which includes the slots reserved by node j and its one-hop neighbors) if it is in the IDLE or RELEASE state. If node j is in the GRANT state, it replies with a *reject* message. After receiving a *reject* message, node i goes back to the IDLE state and sends a *fail* message. If no *reject* message has been received from the neighbors, node i can reserve a time slot after receiving *grant* messages from its one-hop neighbors. The selected time slot is the minimum slot number that has not been taken by its one-hop and two-hop neighbors (i.e., two-hop neighborhood). After slot selection, node i goes into the RELEASE state and broadcasts a *release* message to notify its neighbors about the reservation.

We can identify the following shortcomings for DRAND: (i) Considering the convergecast traffic pattern, DRAND cannot benefit from the potential improvements of utilizing link scheduling instead of node scheduling. (ii) Each node requires to know the time slots of its one-hop and two-hop neighbors before deciding about its own slot; therefore, slot reservation should be performed sequentially, which increases the duration and energy consumption of this algorithm. (iii) When a node sends a *grant* packet, that packet should include the transmission slots of the one-hop neighbors. This packet enlargement affects DRAND's duration and intensifies pack losses. (iv) Regardless of the next-hop node towards the sink, a node reserves a slot when the maximum number of *request* transmissions is achieved but no *grant* message has been received from the parent. This may happen when the network is sparse and link qualities are low.

(iii) one slot for advertisement transmission, and (iv) one slot for receiving reservation request.

3. Link Scheduling vs Node Scheduling

3.1. Formulation

Definition 1. *Scheduling. The scheduling problem is equivalent to slot assignment (color assignment) to all of the or a set of the nodes or links of a graph so that specific slot assignment conditions are met through employing a minimum number of slots (colors).*

As the number of assigned time slots reveals the level of concurrency, a lower slot number used to schedule a network results in higher spatial reuse in that network. Accordingly, the efficiency of the scheduling algorithms can be measured by the maximum slot number they use for graph coloring. The slot assignment conditions defined for a scheduling problem depend on the nature of transmissions. Below are the conditions defined for node and link scheduling problems.

Considering the node scheduling problem, a node's transmission should not conflict with the transmission of one-hop and two-hop neighbors. In other words, the slot number assigned to a node should be different from the slot numbers assigned to its one-hop and two-hop neighbors. This slot assignment condition guarantees a node's transmission is receivable by its entire one-hop neighbors.

Definition 2. *Slot assignment conditions for node scheduling. Assuming more than one transmission slot can be assigned to each node, slot number x can be assigned to node i if $x \notin \text{TS}(i)$ and $x \notin \text{TS}(k)$ $\forall k \in \text{N}_i^1 \cup \text{N}_i^2$. Here, for a sample node y, $\text{TS}(y)$ is the set of the transmission slots of node y, and N_y^1 and N_y^2 are the set of the one-hop and two-hop neighbors of node y, respectively.*

In the link scheduling problem, a node's transmission should be receivable at its intended receiver. With respect to data gathering application, a node's transmission should be receivable at its parent. Therefore, assigning slot x to link (i, j) indicates the assignment of slot x as the transmission and reception slot of node i and node j, respectively. The assignment conditions are as follows.

Definition 3. *Slot assignment conditions for link scheduling. Assuming each node can be assigned more than one transmission and reception slot, slot x can be assigned to link (i, j) if the followings are fulfilled: (i) $x \notin \text{RS}(i) \cup \text{TS}(i)$, (ii) $x \notin \text{RS}(j) \cup \text{TS}(j)$, (iii) $x \notin \text{RS}(k)$ $\forall k \in \text{N}_i^1$, and (iv) $x \notin \text{TS}(o)$ $\forall o \in \text{N}_j^1$. Here, for a sample node y, $\text{RS}(y)$ is the set of the reception slots of node y.*

3.2. Requirements

Scheduling algorithms require neighborhood information to achieve collision-free slot assignment. For node scheduling algorithms no information regarding the routing structure is required since all the one-hop neighbors of a node are its potential receivers. Therefore, node scheduling algorithms can immediately be started after the neighbor discovery phase. In contrast, link scheduling algorithms require routing information to assign time slots to each transmitter-receiver pair. Accordingly, in addition to the neighbor discovery phase, a collection tree construction protocol should be run before executing DICSA. After the collection tree construction phase, each node is provided with the cost of its one-hop neighbors towards the sink. This enables the nodes to forward their data packets through their minimum-cost neighbor, which is referred to as *parent* [17, 41, 42].

4. Design and Implementation of DICSA

DICSA includes two algorithms that allow the nodes to perform their own slot reservation, as well as handling neighbors' slot reservations. Both of these algorithms access to a common set of slots lists, called *forbidden slots lists*. In this section, we first introduce the forbidden slots lists that should be maintained by the nodes. Then, we present the details of the algorithms.

4.1. Forbidden Slots

Data forwarding within a tree structure indicates many nodes would have more than one reception slot. In addition, in order to provide a flexible algorithm, we assume a node can reserve more than one transmission slot. This capability allows slot assignment based on traffic demands. However, it should be noted that, to provide a fair comparison with other algorithms, this capability has not been considered in the evaluations of this paper. Having these assumptions, in this section we introduce the forbidden slots lists, which provide a slot assignment framework before presenting the operation of DICSA.

In order to perform data transmission over link (i, k), node i should execute a reservation phase that is responsible to reserve a transmission slot at node i and a reception slot at node k. However, since node i should select a slot number before initiating a reservation phase, it requires to be aware of the slots that cannot be selected. We call these slots the *forbidden slots* and we categorize them into two groups: *general forbidden slots* and *parent-specific forbidden slots*. The general forbidden slots for node i are:

- Transmission slots: A node cannot select a slot it has confirmed for packet transmission. These slots are recorded in the *Transmission Slots* (TS) list.
- Reception slots: A node cannot select a slot it has reserved for packet reception from a child node. These slots are recorded in the *Reception Slots* (RS) list.
- Reception slots of the one-hop neighbors: A node cannot apply for the slots which are used by its one-hop neighbors for packet reception. Selecting such a slot may result in packet loss due to collision. These slots are recorded in the *Reception Slots of One-hop neighbors* (RSO) list.

The parent-specific forbidden slots are as follows:

- Transmission slots of the selected parent: A node cannot select a slot its parent has confirmed for transmission. The transmission slots of one-hop neighbors are recorded in the *Transmission Slots of One-hop neighbors* (TSO) list.
- Transmission slots of the one-hop neighbors of parent: A node should not select a slot which has been selected by at least one of the one-hop neighbors of the parent for transmission. Selecting such a slot may cause packet loss due to collision at the selected parent. The transmission slots of two-hop neighbors are recorded in the *Transmission Slots of Two-hop neighbors* (TST) list.

DICSA allows the nodes to apply for slot reservation irrespective to the time and neighbors' reservation status. Since a slot reservation phase may succeed or fail, nodes cannot immediately update their forbidden slots lists unless the reservation is confirmed. Accordingly, in addition to the above forbidden slots lists, which are referred to as the *permanent forbidden slots lists*, a set of *temporary forbidden slots* lists should be kept at each node. This set includes: *Temporary Transmission Slots* (TemTS) list; *Temporary Reception Slots* (TemRS) list; *Temporary Reception Slots of One-hop neighbors* (TemRSO) list; *Temporary Transmission Slots of One-hop neighbors* (TemTSO) list. It should be noted that there is no temporary TST list because this list can only be updated after a node confirms its reservation and its one-hop neighbors broadcast $notification$ messages.

Each entry in the aforementioned slots lists is in the form of $< i, k, o >$, where i indicates the sender, k is the receiver, and o is the slot in which i sends to k. Notice that since a node can send to at most one receiver in a given slot, $< i, k, o >$ can be reduced to $< i, o >$; however, we employ the earlier form for clarity.

4.2. Algorithms

DICSA employs two algorithms (state machines) that enable the nodes to perform their slot reservation concurrently. Each sample node x should maintain two types of state variables:

- $S(x)$: Indicates the reservation status of node x;
- $S(x)[y]$: Indicates the reservation status of node y at node x.

Notice that while there is only one $S(x)$ state variable per node, the number of $S(x)[y]$ state variables at node x is

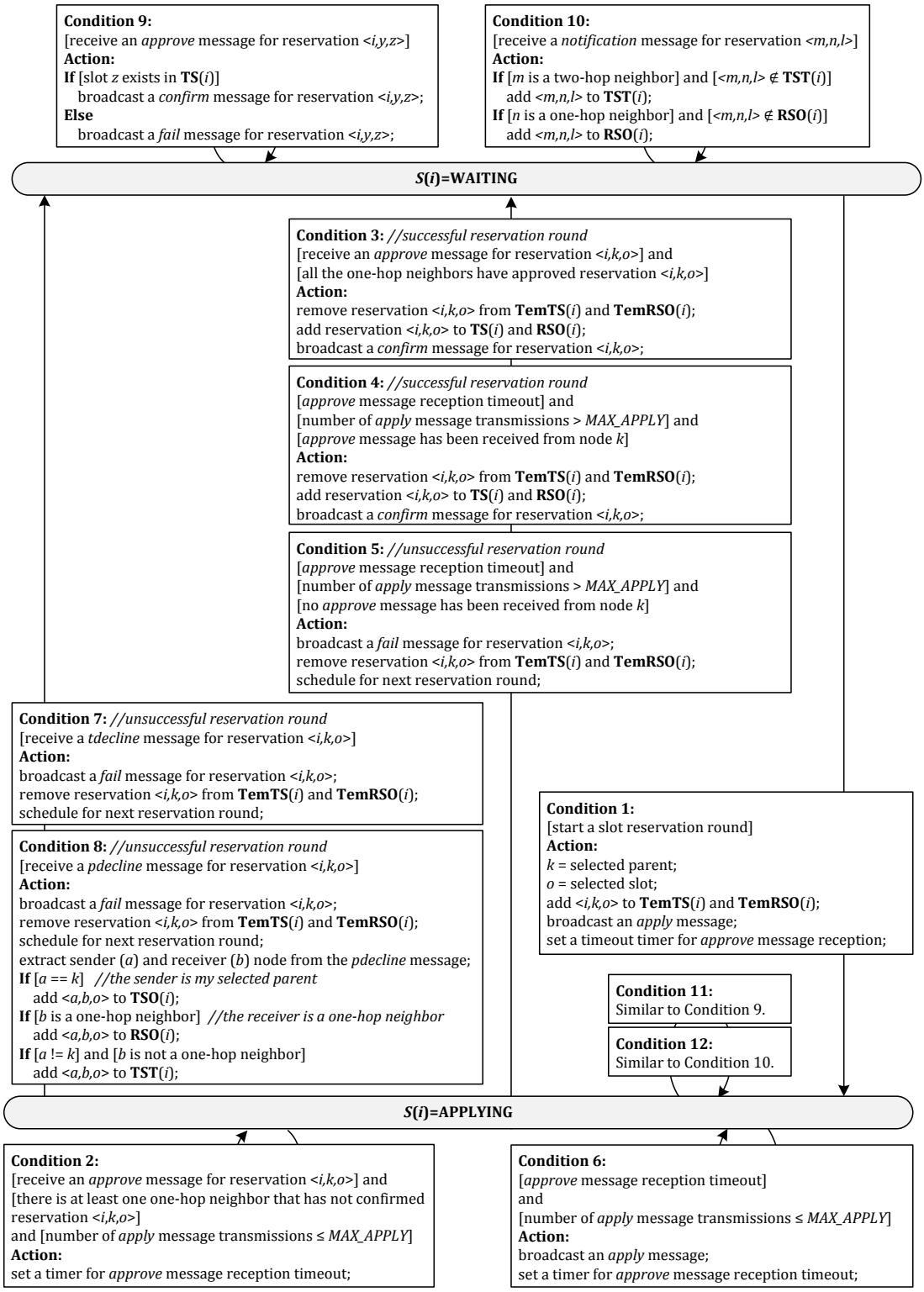

Figure 1: Primary State Machine (PSM). Nodes utilize this algorithm to manage their own slot reservation status. Node i is in the WAITING state when it is not currently requesting for slot reservation. Node i is in the APPLYING state when it is applying for a slot reservation and waits for approval from its neighbors. Note that when node i is in the APPLYING state, triplet $< i, k, o >$ shows the slot this node is trying to reserve.

$|N_x^1|$, which is equal to the number of one-hop neighbors of node x. Figure 1 shows the algorithm through which each node manages its own reservation status. Figure 2 shows the algorithm each node uses to react against the

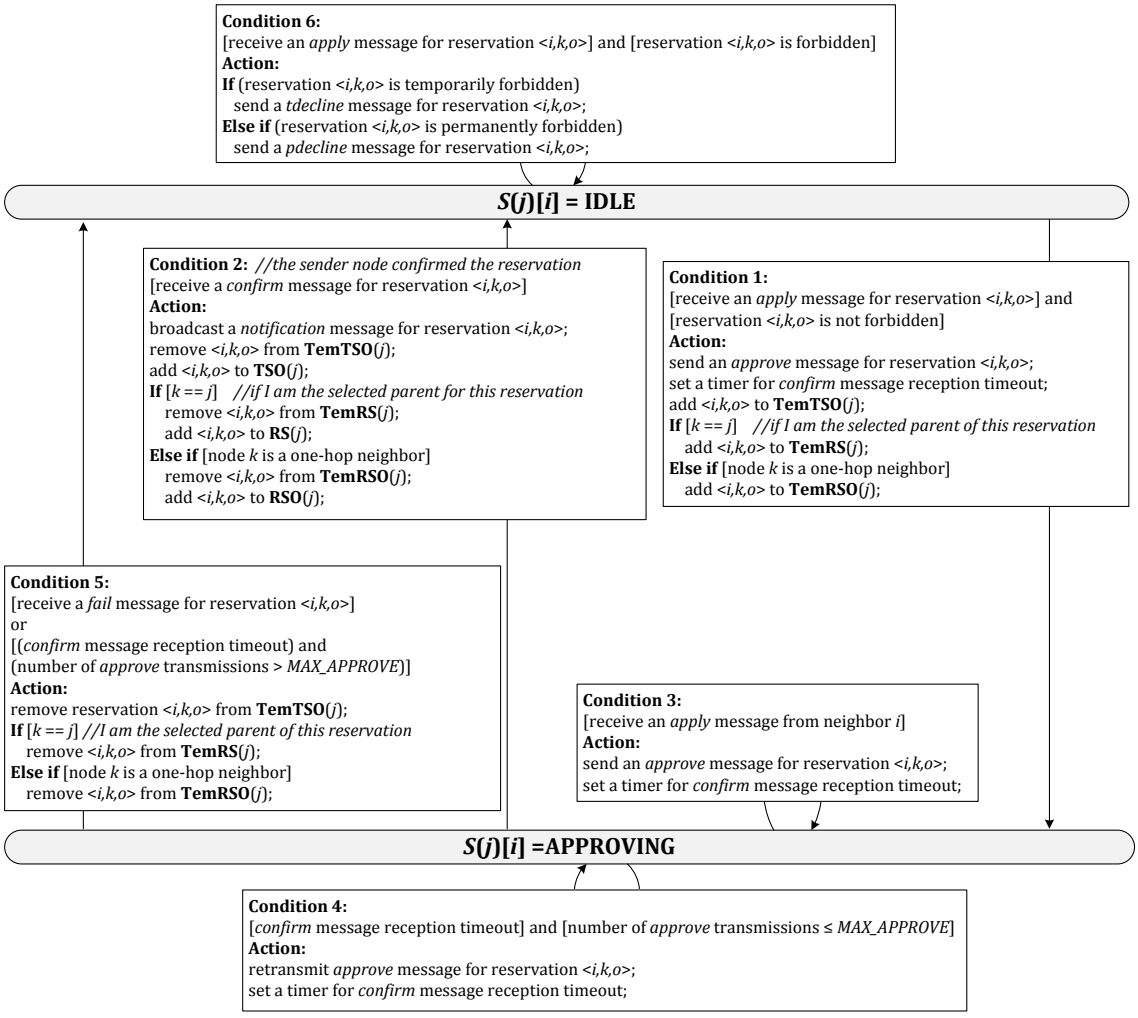

Figure 2: Secondary State Machine (SSM). Nodes utilize this algorithm to behave against the slot reservations of their neighbors. Since DICSA allows concurrent slot reservation, each node should maintain the reservation status of its neighbors separately. $S(j)[i]$ indicates the reservation status of node i at node j. $S(j)[i]$ is IDLE when node j is not involved in a slot reservation round of node i. $S(j)[i]$ is APPROVING when node j has sent an approve message to node i and is waiting for reservation confirmation.

slot reservation of its neighboring nodes. In this paper we refer to these state machines as the Primary State Machine (PSM) (Figure 1) and Secondary State Machine (SSM) (Figure 2). It should be noted that, at each node, PSM and SSM have access to a common set of permanent and temporary forbidden slots lists.

With respect to its own reservation status, i.e., $S(x)$, each node can be in one of the following two states:

- WAITING: When all the required transmission slots have been reserved, or the next slot reservation phase has not been started yet;
- APPLYING: When an *apply* message has been sent and this node is waiting for slot reservation approval from its neighbors.

Regarding the reservation status of neighboring node y, $S(x)[y]$ (which is kept in node x) can be in one the following states:

- IDLE: When node x is not currently involved in a slot reservation round of node y;
- APPROVING: When node x has sent an *approve* message to node y and waits for a response.

In the following we describe the operation of DICSA with respect to the aforementioned state machines. Assume node i wants to reserve a slot for link (i, k), and node j is a one-hop neighbor of node i. Also, statement "A: Condition B" refers to condition B of state machine A.

When node i enters a slot reservation round, firstly, it selects the minimum slot number that has no conflict with the permanent and temporary forbidden slots lists. Afterwards, it broadcasts an *apply* message including the sender address, selected parent address, and selected slot number (shown as triplet $< sender, parent, slot >$). After broadcasting the *apply* message, node i changes $S(i)$ to APPLYING, and waits for a specific time duration υ to

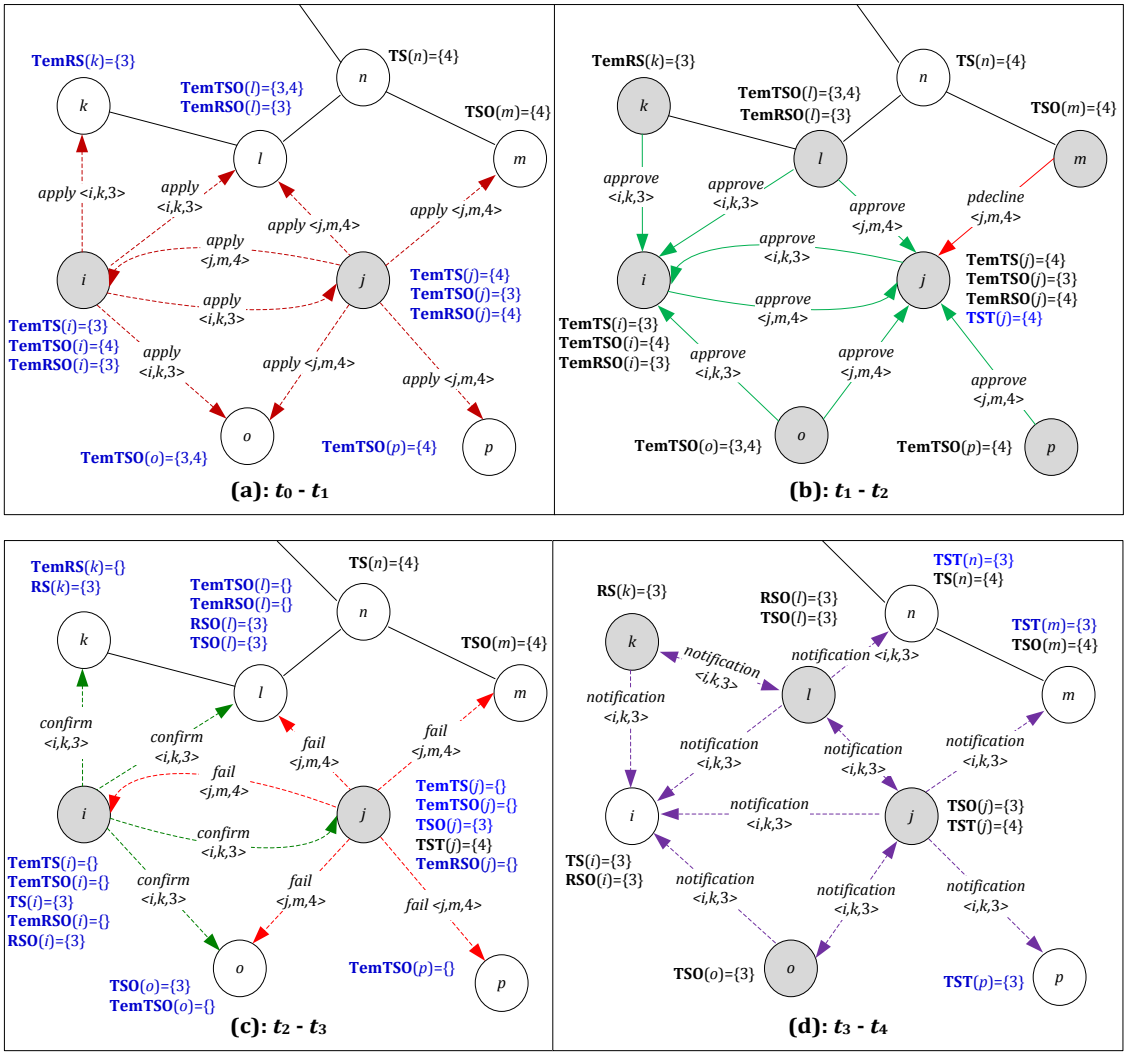

Figure 3: A sample slot reservation scenario. Gray circles show the transmitting nodes during each interval. Solid arrows and dotted arrows represent unicast and broadcast transmissions, respectively.

receive *approve* message from its neighbors (PSM: Condition 1). Computing the υ value will be described in Section 4.4. After receiving *approve* messages from the one-hop neighbors, node i makes the reservation permanent, broadcasts a *confirm* message and moves into the WAITING state (PSM: Condition 3). If node i receives a *tdecline* or *pdecline* message from at least one of its neighbors, it terminates this reservation round through sending a *fail* message and moves to the WAITING state (PSM: Condition 7,8). If node i does not receive any *approve* message within time υ, it retransmits the *apply* message (PSM: Condition 6). If one or more neighbors do not respond after MAX_APPLY transmissions, reservation can be confirmed without considering that neighbors (PSM:Condition 4). However, if the parent node is amongst the non-responding neighbors, node i terminates this reservation round through broadcasting a *fail* message and changes $S(i)$ to WAITING (PSM:Condition 5). Since DICSA allows multiple slot reservations, and because neighboring nodes may loose the *confirm* or *fail* messages, node i may receive *apply* messages during the WAITING or APPLYING state (PSM: Condition 9,11). Therefore, for example, even when node i is in the APPLYING state, it may send a *fail* message in response to the *apply* message related to its previous slot reservation. Accordingly, all the *confirm* and *fail* messages should include the information of the reservation they have been sent for.

Upon receiving the *apply* message of node i at node j, node j should refer to its forbidden slots lists and check whether the reservation can be approved. (Note that the reservation status of node i at node j is $S(j)[i]$). If the reservation request has no conflict with the permanent and temporary forbidden slots, node j sends an *approve* message (including the slot reservation information) to node i and changes $S(j)[i]$ into APPROVING (SSM:Condition 1). If the reservation is in conflict with an entry of the permanent forbidden slots lists, node j sends a permanent decline message (*pdecline*) that justifies why the slot is de-

clined (SSM:Condition 6). In other words, node j includes in the *pdecline* message the slot information entry found in the permanent forbidden slots lists. Using this slot entry, node i can update its permanent forbidden slots lists and avoid using that slot for its future slot reservations. If node j finds no conflict with the forbidden slots lists, but with the temporary forbidden slots lists, it sends a temporary decline (*tdecline*) message to node i (SSM:Condition 6). When node j goes into the APPROVING state, it expects to receive a *confirm* or *fail* message from node i within a specific time interval. Computing the timeout duration will be described in Section 4.4. After receiving a *confirm* message for this reservation, $S(j)[i]$ changes to IDLE and reservation $< i, k, o >$ is changed from temporary to permanent. In addition, node j broadcasts a *notification* message to inform its neighbors about the confirmed reservation (SSM:Condition 2). For those nodes that are in the one-hop neighborhood of the parent node, the *notification* message allows them to update their RSO list. For those nodes in the two-hop neighborhood of the sender, they can update their TST list (PSM: Condition 10,12). If node j does not receive any *confirm* message from node i, node j retransmits the *approve* message. If the *approve* message has been sent for $MAX\_APPROVE$ times or when node j receives a *fail* message for this reservation, $S(j)[i]$ changes to the IDLE state and reservation $< i, k, o >$ is removed from the temporary forbidden slots lists (SSM:Condition 5).

Figure 3 shows a sample slot reservation scenario initiated by node i and j. Packet transmissions and changes in the forbidden slots lists of nodes are demonstrated within four time intervals. During $t_0 - t_1$, both node i and node j broadcast *apply* message. After receiving these packets, the one-hop neighbors of these nodes modify their forbidden slots. For example, node l adds slot 3 and slot 4 to its TemTSO list. Additionally, node l adds slot 3 to its TemRSO list since the selected parent of node i is its one-hop neighbor. During $t_1 - t_2$, one-hop neighbors of node i and node j reply with *approve* message. However, node m, which is the selected parent of node j, replies with a *pdecline* packet because its one-hop neighbor, node n, has previously reserved slot 4 for transmission. Although $notification$ messages enable the nodes to be informed about the transmission slots of their two-hop neighbors, node j was not aware about the transmission slot of node n. This could happen due to the issues such as packet loss due to collision, or packet loss when the transceiver is in transmit mode. Notice that node j updates its TST list according to entry $< n, x, 4 >$ found in the *pdecline* message. During $t_2 - t_3$, node i broadcasts a $confirm$ message and node j broadcasts a $fail$ message to notify their neighbors about their reservation status. Finally, the one-hop neighbors of node i broadcast notification message during $t_3 - t_4$. This $notification$ messages, for example, allow node p to be aware about the transmission slot of its two-hop neighbor. Therefore, node p will not apply for reservation $< p, j, 3 >$.

4.3. Correctness

The correctness of DICSA follows from the fact that a slot number reserved by a node does not violate the conditions given in Definition 3. To this aim, a node should make sure that its selected slot is not in any of the forbidden slots lists even when a slot list is not up to date, which may happen due to packet loss. DICSA satisfies the conditions mentioned in Definition 3 as follows. Assume node i wants to reserve a slot for transmission to node j. First, node i checks the existence of x in its RS, TemRS and TS lists, and these lists are always up-to-date; therefore, $x \notin \text{RS}(i) \cup \text{TemRS}(i) \cup \text{TS}(i)$. Second, if $\text{TS}(j) \cup \text{TemTS}(j) \neq \emptyset$, node j has previously reserved or is reserving a slot, which requires the reception of an *approve* message from node i; therefore, $x \notin \text{TS}(j) \cup \text{TemTS}(i)$. Third, the exchange of *confirm* and *notification* messages allows the nodes to be aware of the reception slots of their one-hop neighbors and update the RSO and TemRSO lists. Assume node i is a one-hop neighbor of node k, but node i is not a one-hop neighbor of node m which has reserved or is reserving $< m, k, l >$. In this case, node i can be aware of the reception slot of node k through receiving a *notification* message from node k. If the RSO and TemRSO lists of node i do not include the reception slot of node k (which may happen due to the issues such as packet loss), condition $x \notin \text{RS}(k) \cup \text{TemRS}(k)\ \forall k \in \text{N}_i^1$ is satisfied through requiring node i to receive *approve* message from its one-hop neighbors. Finally, the exchange of *confirm* and *notification* messages allows node i to be aware of the transmission of the one-hop neighbors of node j; therefore, avoiding conflict with these slots. If slot x is being or has been reserved by a sample node o, which is a one-hop neighbor of node j, node j will terminate the reservation of node i. This is because node j should approve the slot reservation of its neighbors, including node i and node o. Therefore, if node j has approved slot x for node o, then it will decline the reservation of slot x by node i. Consequently, $x \notin \text{TS}(o) \cup \text{TemTS}(o)\ \forall o \in \text{N}_j^1$.

4.4. Timing

Although DICSA does not rely on time synchronization, its performance depends on the timings used during the algorithm execution. This dependency is specifically due to the concurrency of slot reservation. Here, we present the timing details used in the implementation of DICSA.

Assuming node i starts a slot reservation phase, it should wait for 1.1υ after each *apply* message transmission to receive at least one response from its neighbors. In addition, node i renews its timer after each *approve* message reception. As υ is one-way message delay, initially, node i estimates υ as $T_{packet} + T_{CW}$, where T_{packet} is the packet transmission duration and T_{CW} is the contention window size of the CSMA protocol used during the execution of DICSA. Node i can improve its estimation of one-way message delay during the network operation.

Assume node j, a one-hop neighbor of node i, receives an

apply message from node i, and replies with an *approve* message. Node j should wait for node i to receive its neighbors' responses and send a $confirm$ message. To compute this value, node i should include in its *apply* messages the number of the neighbors from which it expects response message. Assuming that the number of response-pending neighbors of node i is $|N_i^{pending}|$, node j computes its waiting time for *confirm* message reception as $\upsilon \times |N_i^{pending}| + \upsilon + \text{uniform}(0, 2\upsilon)$. Here, uniform$(0, 2\upsilon)$ is a uniform random variable in range $[0,2\upsilon]$ and is used to prevent simultaneous timer expiry and channel access by the one-hop neighbors of node i. Note that $|N_i^{pending}|$ may be changed after each *apply* message retransmission.

Due to packet transmission delay, a node's timeout timer may expire while the radio is busy with transmission. In this case, although packet buffering is a feasible solution, it may increase the algorithm overhead in terms of unnecessary sent packets. To overcome this problem, a node should reschedule its timeout timer for $\upsilon/2$ when an $approve/decline$ or $confirm/fail$ timeout happens and the radio is busy with transmission. If a node decides to enter a reservation phase while the radio is busy, the reservation is postponed to 3υ which is long enough to send and receive two packets. If the packet transmission request is not caused by a timer expiry, packet buffering can be used to send the buffered packet once the radio is idle.

5. Performance Evaluation and Discussion

Using an accurate simulation framework, we conduct extensive simulations to evaluate the performance of DICSA. We perform two types of evaluations: First, we show the performance improvements of DICSA in terms of algorithm execution duration, energy efficiency and slot assignment efficiency. Second, we show how employing DICSA in a TDMA MAC protocol can improve the performance of data gathering applications.

5.1. Simulation Configuration

We implemented the protocols and algorithms evaluated in this paper in a simulation tool developed on the OMNeT++ simulation framework [43]. Within this framework we developed an accurate wireless channel and physical layer model to precisely simulate the characteristics of low-power wireless communication [44]. In particular, we considered the following properties: (i) Amongst the interference models we used the signal-to-interference-plus-noise ratio (SINR) model due to its highest accuracy [45, 46]. (ii) Since low-power transceivers present the capture effect [47], our packet reception model employed at the physical layer of the nodes provides an accurate implementation of the capture effect [48]. (iii) In addition to the multipath channel variations, we also included noise floor variations caused by the white Gaussian noise. (iv) We considered the effects of hardware heterogeneity on transmission power and noise floor [49].

Table 2: Simulation Parameters

Parameter	Value
Radio	
Average noise power [dBm]	-106
Switch to TX/RX [us]	250
Radio sampling [us]	350
Evaluate radio sample [us]	100
Modulation	NC-FSK
Encoding	Manchester
Radio speed after encoding [bps]	19200
PL(d_0) [dB]	55
Heterogeneity of transmission powers	1.2
Heterogeneity of noise floors	0.9
Correlation of transmission power and noise floor	-0.7
TX current consumption [mA]	16.5
RX/Idle current consumption [mA]	9.6
Environment	
Path loss exponent (η) (outdoor)	4.7
Variance of multipath channel [dB] (outdoor)	3.2
Variance of white Gaussian noise [dB]	4
CSMA MAC	
Initial contention window [slot]	128
Congestion contention window [slot]	64
Packet Format	
Phy header/MAC header/Payload/CRC	8/5/29/2
Other Parameters	
Battery capacity [mAh]	2500
Packet buffer size [packets]	20

Table 2 presents the general simulation parameters of this paper. The radio parameters have been chosen based on the documentation of Mica2 motes with CC1000 radio. The environmental parameters have been chosen based on the studies of [49]. Packet format is similar to that used by TinyOS [50]. TinyOS's default CSMA MAC protocol is used during the scheduling algorithm executions. We set the maximum number of packet retransmissions to 100. Each figure point represents the median of 20 runs. Error bars indicate upper and lower quartiles.

During the neighbor discovery phase each node broadcasts 60 packets to find its one-hop and two-hop neighbors. Nodes also estimate the link qualities to their one-hop neighbors [17, 51]. Afterwards, we employ the TinyOS's Collection Tree Protocol (CTP) [1] to establish routing tables at the nodes.

It is worth mentioning that the simulation parameters used in this paper are different from those used in [24]. In particular, while the simulation parameters of [24] are valid for ad-hoc networks with high radio data rate, our parameters are valid for Mica2 sensor networks. Therefore, our reported results for the performance of DRAND comply with the multihop Mica2 experiment of [24], and not those evaluated by NS2.

5.2. One-Hop Scenario

We first evaluate the performance of DICSA and DRAND in various one-hop networks. We consider 2-20 nodes deployed in a circular topology. Nodes transmit at 5 dBm.

Figure 4 presents our evaluation results for the one-hop scenario. In terms of algorithm execution duration,

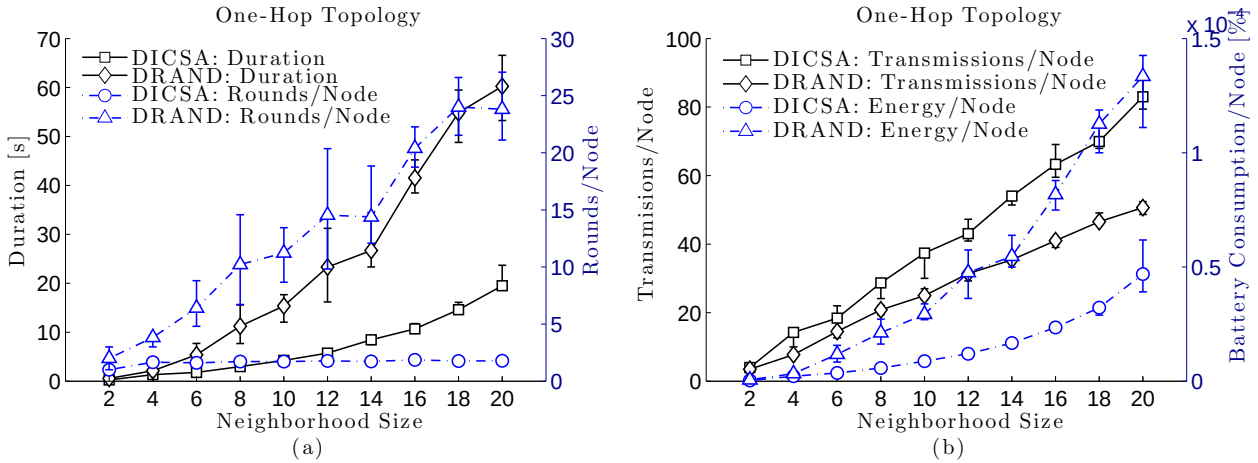

Figure 4: Evaluating the execution performance of DICSA and DRAND in various-size one-hop networks. (a): Execution duration and average number of rounds per node. (b): Average number of packet transmissions per node and average battery consumption per node. Since DICSA does not impose any distance between the nodes reserving slot, it provides shorter algorithm execution duration and lower energy consumption.

DICSA provides significantly lower duration due to its concurrency mechanism. Since all the nodes are one-hop neighbors, DRAND requires the nodes to perform their slot reservation sequentially. In contrast, DICSA allows the nodes to be concurrently involved in slot reservation. Therefore, since DICSA reduces the number of failed slot reservation tries, the average number of rounds per node is significantly lower than that of DRAND.

Despite the lower duration of DICSA, it requires more packet transmissions because of its concurrency mechanism that causes higher number of packet losses during the algorithm execution. It should be noted that one-hop networks are not collision-free. As CSMA is the employed channel access mechanism during the algorithm execution, a collision happens when two or more nodes select the same backoff slot. Therefore, the number of transmissions can be reduced through increasing the contention window size.

Regardless of its higher number of transmissions, DICSA shows lower energy consumption per node, which is due to its lower execution duration. As CSMA is used during the execution of these scheduling algorithms, and since the energy consumed by the radio in the idle and receive modes are identical and dominate the transmit mode, energy consumption of the scheduling algorithms mainly depends on the execution duration.

We do not present maximum slot number for the one-hop scenario because it is always equal to $V-1$, where V is the number of nodes.

5.3. Multi-Hop Scenario

In this section we conduct performance evaluations in multihop networks with various two-hop neighborhood densities. In a $50m \times 50m$ area we changed the number of nodes from 60 to 200 to generate various densities. Nodes transmit at 0 dBm.

Figure 5 shows our performance evaluation results in multihop scenarios. Although DRAND claims its execution duration linearly increases versus neighborhood size, the performance evaluations of [24] as well as our evaluations show quadratic relationship, which is due to the packet losses caused by collisions. Compared to DRAND, the execution duration of DICSA is also quadratic, however, with lower growth rate. Our results show that the execution duration of DICSA is at least 50% less than that of DRAND.

The one-hop and two-hop results signify that DICSA's number of rounds is independent from neighborhood size. In both DRAND and DICSA a node can confirm its slot reservation if no response is received from some neighbors after achieving the maximum number of retransmissions. In DICSA, the current round should be terminated and a new round should be initiated when: (i) a node receives a *tdecline*/*pdecline* message due to the selection of a forbidden slot, or (ii) when a node achieves the maximum number of retransmissions without receiving *approve* message from the parent node. Further evaluations have been performed to measure the effects of forbidden slot selection and the number of packet retransmissions on the number of rounds. These evaluations revealed that the maximum number of packet retransmissions is the main parameter affecting the number of rounds. In particular, when the number of retransmissions is not large enough to cope with the packet losses caused by collision, child-parent packet exchanges cannot be accomplished and therefore, the current round should be terminated and a new round should be initiated. These investigations also revealed that neighborhood density has no effect on the number of rounds when the number of retransmissions is large enough (e.g., 100 in these evaluations). However, increasing the network density intensifies packet collisions and enlarges the number of nodes from which no reply has been received after reaching the maximum number of retransmissions.

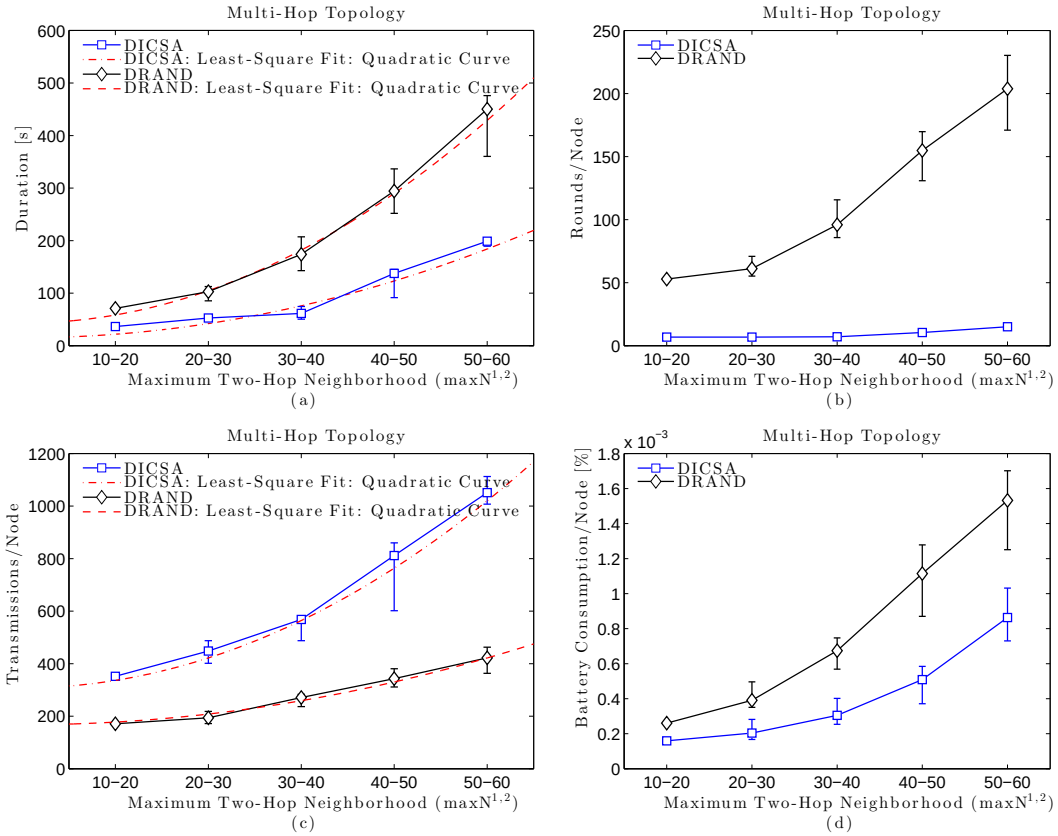

Figure 5: Evaluating the execution performance of DICSA and DRAND in multihop scenarios with various two-hop neighborhood sizes.

Nevertheless, it was observed that these links are usually asymmetric or represent low quality, consequently, they do not affect scheduling accuracy.

Our results show that the number of packet transmissions with DRAND and DICSA demonstrate quadratic growth versus the neighborhood size. However, the growth rate of DRAND is lower than DICSA. We can characterize this difference through the spacing between those nodes applying for slot reservation. In particular, in DRAND, the number of hidden-node collisions is significantly reduced as the spacing between two nodes applying for slot reservation should be at least three hops.

Similar to the one-hop scenario, the lower duration of DICSA results in its lower energy consumption. Our results indicate that the energy consumption of DICSA is more than 40% lower than DRAND.

The number of slots a scheduling algorithm uses for network scheduling is an implication of spatial reuse efficiency. Figures 6 shows the efficiency of slot number assignment achieved by various scheduling algorithms. The maximum slot number assigned by DICSA is up to 50% lower than that of DRAND at high neighborhood densities. This reduction is due to the higher spatial reuse of link scheduling compared to node scheduling. We also showed the upper bound of DRAND, which is $\text{maxN}^{1,2} + 1$. Accordingly, our results confirm that the slot assignment of DICSA is always lower than $0.4 \times (\text{maxN}^{1,2} + 1)$. The 'VDEC' curve demonstrates the lower bound of the slot assignment achieved through employing algorithm [15] (cf. Section 2). Figure 6 indicates that the slot assignment of DICSA provides more than 60% improvement over this distributed link scheduling algorithm. Figure 6 also shows the minimum slot number required to schedule the collection trees constructed from our networks. This lower bound equals

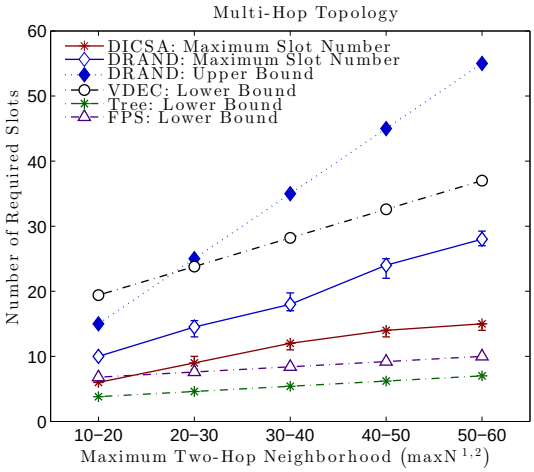

Figure 6: The slot assignment efficiency of various scheduling algorithms. Slot assignment of DICSA shows up to 50% improvement over DRAND, and more than 60% improvement over VDEC.

to maxN[1] of the tree, as reported by [3]. Note that the maximum degree of an interference-free tree can be computed as maxC + 1, where maxC is the maximum number of children per node. Although this number of slots does not hold for our network topology (because it assumes no interfering link in the tree), it can indicate the potential improvements of DICSA when all the interfering links are eliminated through mechanisms such as power control, multiple frequencies or code assignment. The minimum number of slots required for network scheduling through FPS is also demonstrated by Figure 6. This lower bound is equal to maxC + 3, which is very similar to the required number of slots for scheduling a tree that has no interference between its branches. This is because FPS schedules child-parent links and it does not care about interference between branches of a tree. In contrast with FPS, although DICSA considers all the collision scenarios affecting child to parent transmissions, nevertheless, its achieved slot assignment efficiency is close to that of FPS.

5.4. Slot Updating

When a new node is added to the network, or when the network topology changes, some nodes may require to update their slot assignment. Therefore, it is important to update slot assignment with minimum time and energy cost. In this section we evaluate the slot update cost of DICSA against DRAND. We use 100 nodes deployed in three different network densities. For each network we select different number of nodes rerunning DRAND and DICSA during the normal network operation. We report slot update cost in terms of duration and energy cost. Duration is the average time required by the nodes applying for slot update to confirm their slot reservation. Energy consumption is the total energy spent by all the nodes involved with the slot reservation of those nodes applying for slot update.

Figure 7 shows the updating costs. Both duration and energy consumption present significant variations, which is due to the randomness of packet delays and packet losses. Our results show that the recovery cost of DICSA is considerably lower than that of DRAND. In addition, with respect to the DICSA's concurrency mechanism, the improvement percentage of DICSA over DRAND increases when the network density enlarges or the number of nodes applying for slot update increases.

5.5. Data Gathering Applications

In this section we evaluate the performance of DICSA, DRAND, NCR, SEEDEX and FPS in data gathering applications. In particular, we show how a TDMA MAC protocol would behave when each of these algorithms is used as the underlying mechanism of channel access scheduling. It is worth mentioning that we did not consider FPRP [52] because the higher performance of DRAND over this protocol has been previously shown in [24]. Furthermore, we neglected VDEC because its maximum slot number is much higher than DRAND.

We introduce a TDMA MAC framework within which we evaluate these scheduling mechanisms. As DICSA, DRAND and FPS are deterministic algorithms, each node is allowed to send a data packet in a specific time slot during each frame. Therefore, time is divided into *frames* and each frame is composed of M *time slots*, where M is the maximum slot number assigned by the scheduling algorithm. For NCR and SEEDEX, time is not divided into frames, rather, it is a continuous sequence of time slots.

In addition to the efficiency of the underlying scheduling algorithms, the length of the time slots also affect network throughput. In order to determine a realistic duration for the time slots, in particular, we should consider time synchronization accuracy because, in reality, synchronization drifts are inevitable due to the issues such as oscillator inaccuracy and low traffic rate. Therefore, time slot duration should be long enough to avoid the collisions caused by slot overlap. Figure 8(a) and (b) present the minimum required time slot duration when synchronization accuracy (τ) is larger than radio switching delay (ϖ). When the packet transmitter's slot begins 2τ before the receiver (Figure 8(a)), the transmitter should start its packet transmission after $2\tau + \varpi$ to make sure the receiver is ready for reception. Considering the case in which the transmitter's slot begins 2τ after the receiver (Figure 8(b)), slot duration should be at least $4\tau + \varpi + T_{packet} + \zeta$. The value ζ is due to propagation, encoding and decoding delays. We can conclude similar time slot duration when the synchronization accuracy (τ) is shorter than radio switching delay (ϖ), as demonstrated by Figure 8(c) and (d). The obtained time slot duration holds for all the algorithms except FPS. As FPS employs CSMA at the beginning of each time slot, its minimum time slot duration should be $4\tau + \varpi + T_{packet} + \zeta + T_{CW}$.

We consider a 100-node network in which all the nodes generate packet. All the nodes send at 0 dBm. We vary the packet generation rate of the nodes to measure the spatial reuse and collision avoidance efficiency of the scheduling algorithms. Performance evaluation results are presented in terms of four metrics:

- *Sink throughput*: The number of bits successfully received by the sink node per second.
- *Network throughput*: The number of bits successfully received by the nodes per second. We avoided using the number of transmitted bits because, we wanted to measure the effective network throughput, and not those transmissions that could not be received at their intended receivers. For example, the transmission throughput and effective throughput of SEEDEX and FPS are considerably different as a result of packet collisions at the parent nodes.
- *Delivery ratio*: The percentage of the generated packets that have successfully been received by the sink.
- *Packet delay*: The average time taken for the packets from their generation until reception at the sink.

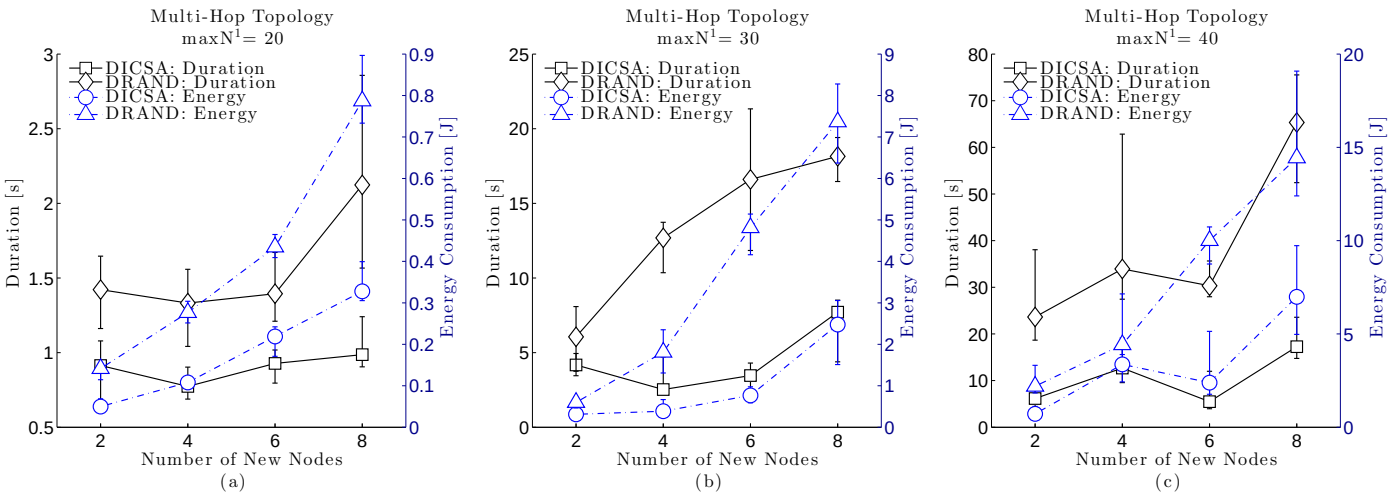

Figure 7: Recovery cost of DICSA and DRAND versus the number of nodes applying for slot update. These results indicate the significantly lower recovery cost of DICSA compared with DRAND. When the network density enlarges or the number of nodes applying for slot update increases, DICSA's concurrency mechanism becomes more beneficial, therefore, increasing the percentage of improvement over DRAND.

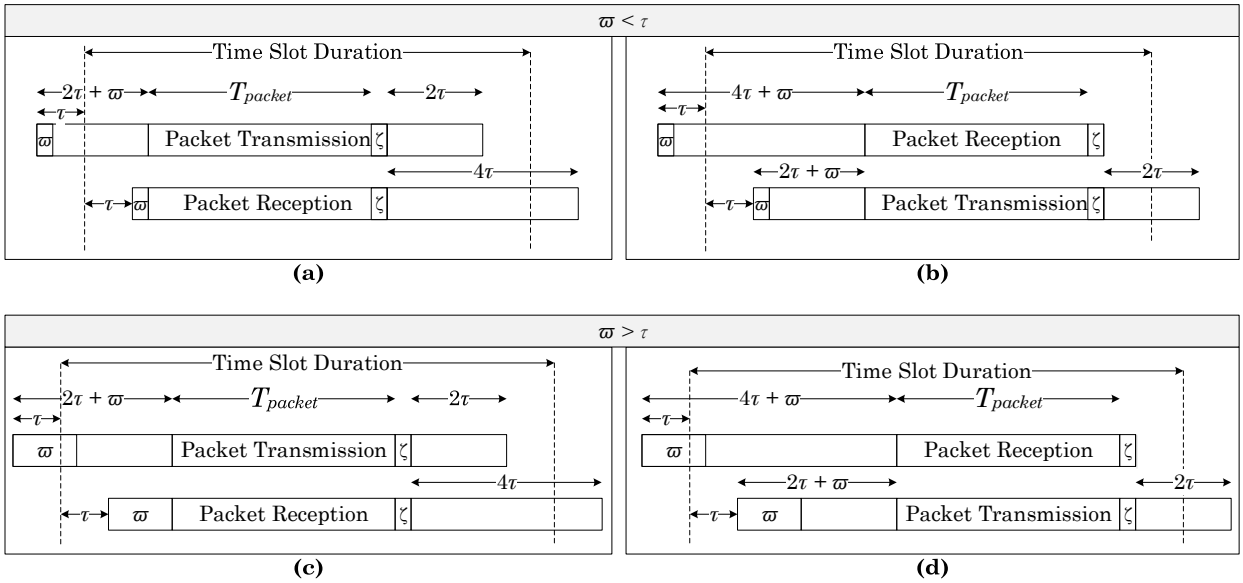

Figure 8: Minimum time slot duration with respect to time synchronization accuracy (τ) and radio switching delay (ϖ). (a) and (b): $\varpi < \tau$. (c) and (d): $\varpi > \tau$. Irrespective to the relationship between τ and ϖ, the minimum time slot duration should be $4\tau + \varpi + T_{packet} + \zeta$. In this paper we assume $\tau = 1ms$, $\varpi + \zeta = 1ms$, and $T_{packet} = 18.33ms$.

Figure 9 presents our evaluation results with respect to the data gathering application. These results confirm the higher spatial reuse and lower number of collisions achieved with DICSA. Compared with DRAND and NCR, DICSA relies on link scheduling to achieve faster channel access. Therefore, it can provide more efficient packet forwarding (higher throughput), and reduces the number of packet losses caused by buffer overflow. Consequently, we can observe higher delivery ratio and lower packet delay with DICSA. DICSA, SEEDEX and FPS all employ link scheduling, therefore they demonstrate lower packet delivery delay compared with DRAND and NCR. However, when a child transmits to its parent, both SEEDEX and FPS suffer from packet losses caused by the transmissions of the one-hop neighbors of the parent; hence, they present lower throughput and delivery ratio compared with DICSA. Assuming the transmission of a child to its parent, using SEEDEX, collisions occur at the parent node because of the probabilistic approach employed by the child node to reduce collisions. In FPS, packet collisions at the parent node are avoided through employing CSMA before commencing transmissions. However, carrier sensing can avoid collisions only when the senders can sense each other. Accordingly, since CSMA cannot avoid hidden-node collisions, increasing the contention window size does not improve the performance of FPS, as it can be observed in Figure 9. Consequently, in contrast with DRAND and NCR in which packet losses are mainly due to buffer over-

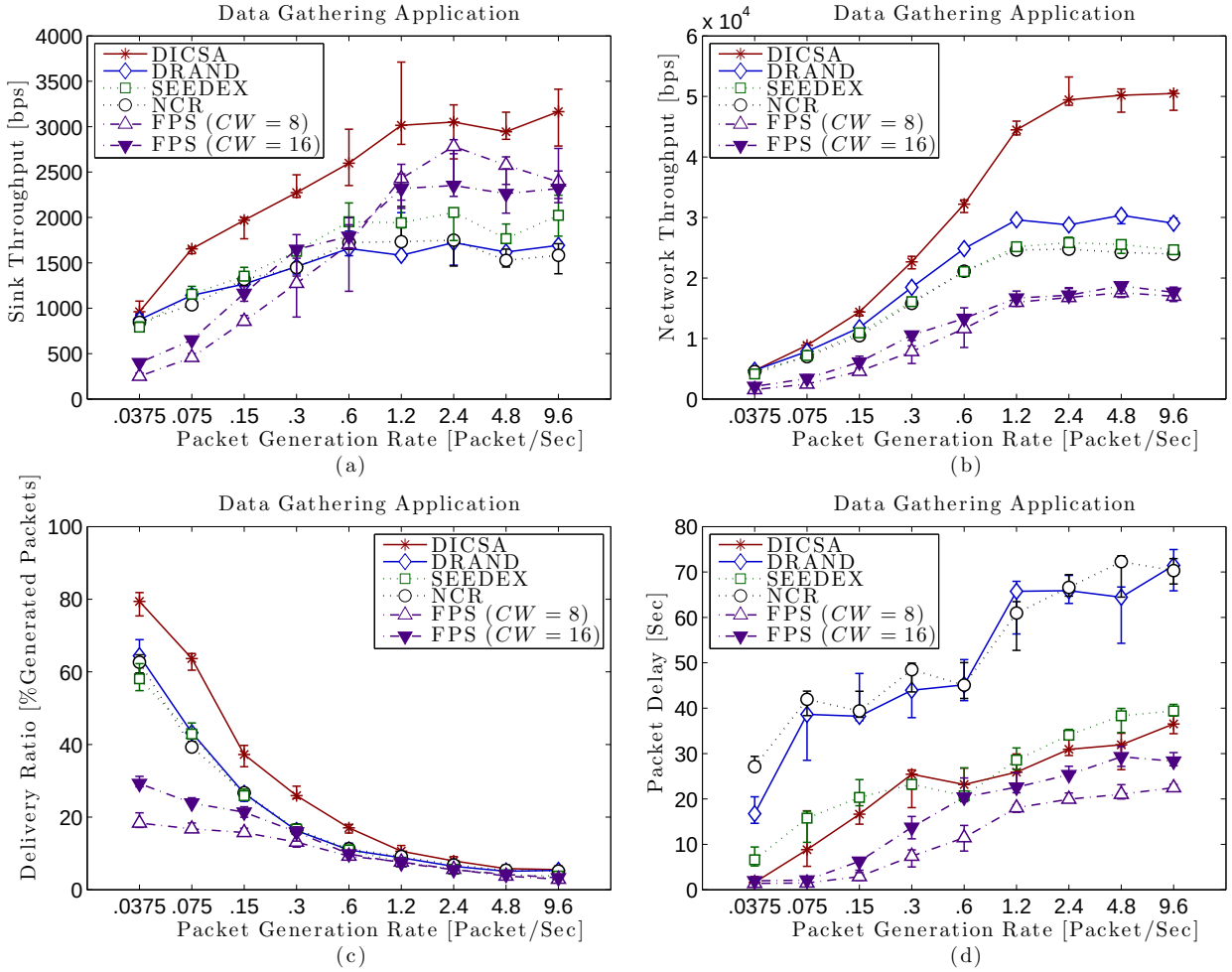

Figure 9: Performance evaluation of various scheduling algorithms versus traffic rate in data gathering applications. Compared with DICSA, the lower throughput and delivery ratio of DRAND and NCR is due to the lower spatial reuse of node scheduling which results in buffer overflow and packet loss. Packet losses caused by collisions reduce the throughput and delivery ratio of SEEDEX and FPS, compared with DICSA.

flow, the lower throughput and delivery ratio of SEEDEX and FPS are due to the packet losses caused by collisions.

Figure 10 presents network throughput versus neighborhood size. This figure shows that the throughput of DICSA is more than 50%, 70%, 90% and 170% higher than that of DRAND, SEEDEX, NCR and FPS, respectively.

6. Storage Analysis

The maximum amount of storage required by DICSA at each node mainly depends on the storage required to store the forbidden slots lists. Let $\max|x|$ be the maximum possible size of list x, and κ denote the maximum number of slots a node reserves. Therefore, $\max|\text{TS}| = \kappa$. The maximum size of list RS depends on the maximum number of children per node; therefore, $\max|\text{RS}| = (\max\text{N}^1 - 1) \times \kappa$. The maximum size of list RSO depends on the maximum number of one-hop neighbors per node and the maximum number of children per node; therefore, $\max|\text{RSO}| = \max\text{N}^1 \times (\max\text{N}^1 - 1) \times \kappa$. The maximum size of list TSO depends on the maximum number of one-hop neighbors per node; therefore, $\max|\text{TSO}| = \max\text{N}^1 \times \kappa$. Finally, the maximum size of list TST depends on the maximum number of two-hop neighbors; therefore, $\max|\text{TST}| = \max\text{N}^1 \times (\max\text{N}^1 - 1) \times \kappa$. Consequently, the maximum amount of storage required by DICSA is,

$$\Gamma = 2 \times (\max\text{N}^1)^2 \times \kappa \times \gamma \quad (1)$$

where γ is the number of bits required to store a slot number. The amount of storage required by DICSA can be reduced through two mechanisms. First, instead of using one byte for storing a slot number (which allows up to 256 slot numbers), the number of required bits can be assigned based on the maximum slot number. For example, as Figure 6 shows, the maximum slot number assigned by DICSA is less than 20 for two-hop neighborhood size up to 60. Therefore, assigning 5 bits for storing a slot number is enough. As another but more straightforward optimization we can reduce the size of TSO and TST lists. In particular, a node only stores the transmission slots of

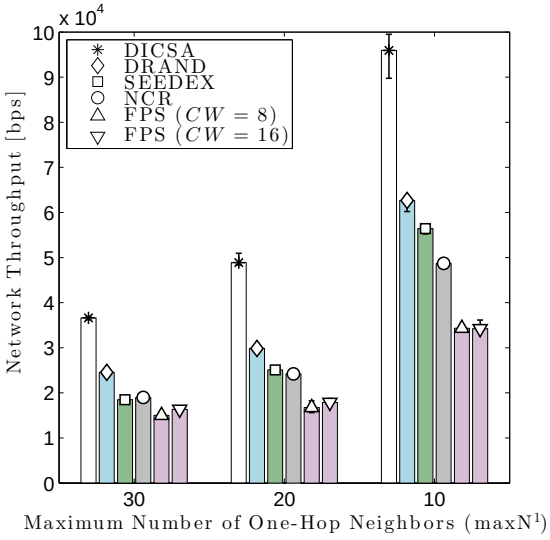

Figure 10: Throughput evaluation of various scheduling algorithms versus neighborhood size in data gathering applications.

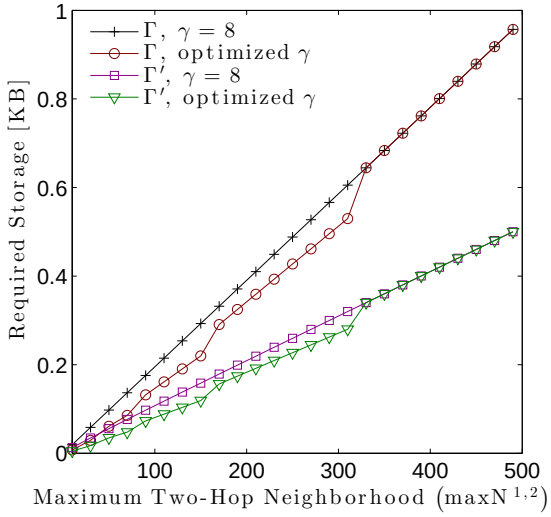

Figure 11: The storage requirement of DICSA.

its parent node in the TSO list; therefore, $\max|\text{TSO}| = \kappa$. Furthermore, a node only stores the transmission slots of the one-hop neighbors of parent in its TST list; therefore, $|\text{TST}| = (\text{maxN}^1 - 1) \times \kappa$. Consequently, the maximum amount of storage required by DICSA is reduced to,

$$\Gamma' = (\kappa \times (\text{maxN}^1)^2 + \kappa \times \text{maxN}^1) \times \gamma \qquad (2)$$

Figure 11 shows the amount of memory required by DICSA. Note that this figure shows Γ and Γ' against two-hop neighborhood size, therefore, both functions demonstrate linear behavior. This figure shows that optimizing the number of entries stored in the TSO and TST lists can considerably reduce the amount of required storage. In addition, for neighborhood sizes of lower than 300, optimizing the number of bits per slot number entry can reduce the required storage by up to about 30%. Also note that the maximum amount of storage required by DICSA is considerably lower than the RAM provided by wireless sensor nodes. For example, for neighborhood size 300, DICSA consumes about 6.5% and 2.6% of the RAM provided by Mica2 and TelosB nodes, respectively[3].

One might ask why we have only computed the storage required by slot numbers, while an entry in a list of forbidden slots should also include the address of the sender node, as described in Section 4.1. According to our previous analysis, most of the storage required by DICSA is for RSO and TST lists. In addition, as a neighbor discovery phase has been completed before the execution of DICSA (cf. Section 3.2), one-hop and two-hop neighborhood tables have already been established and therefore, the slot numbers belonging to the RSO and TST lists can be attached to the corresponding entries in the one-hop and two-hop neighbor tables.

[3]Mica2 provides 4KB and TelosB provides 10KB of RAM.

7. Conclusion

The main traffic pattern of wireless sensor networks is many-to-one, in which child-to-parent packet transmissions forward the packets towards the sink. Therefore, since link scheduling algorithms can provide higher spatial reuse over node scheduling algorithms, data gathering performance can be improved through TDMA MAC protocols that employ link scheduling. In this paper, we proposed DICSA, a link scheduling algorithm that does not require any assumption regarding the underlying network. The most important feature of DICSA is its concurrency mechanism that enables all the nodes to be concurrently involved in slot reservation. Although both DRAND and DICSA perform slot reservation in rounds, our performance evaluations confirm the lower duration and energy consumption of DICSA. Using a TDMA MAC framework, we compared the performance of DICSA with other scheduling algorithms. Our results reveal the higher performance of DICSA in terms of throughout, delivery ratio and packet delay.

References

[1] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, P. Levis, CTP: An efficient, robust, and reliable collection tree protocol for wireless sensor networks, ACM Transactions on Sensor Networks 10 (2013) 1–49.

[2] S. Gandham, Y. Zhang, Q. Huang, Distributed time-optimal scheduling for convergecast in wireless sensor networks, Computer Networks 52 (2008) 610–629.

[3] O. Durmaz Incel, A. Ghosh, B. Krishnamachari, K. Chintalapudi, Fast Data Collection in Tree-Based Wireless Sensor Networks, IEEE Transactions on Mobile Computing 11 (2012) 86–99.

[4] B. Dezfouli, M. Radi, M. A. Nematbakhsh, S. A. Razak, A medium access control protocol with adaptive parent selection mechanism for large-scale sensor networks, in: International conference on advanced information networking and ap-

plications - WINA "11, IEEE, Biopolis, Singapore, 2011, pp. 402–408.
[5] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, S. Masri, Monitoring civil structures with a wireless sensor network, IEEE Internet Computing 10 (2006) 26–34.
[6] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, D. Culler, An analysis of a large scale habitat monitoring application, in: Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04, ACM Press, Baltimore, MD, USA, 2004, p. 214.
[7] L. Yu, N. Wang, X. Meng, Real-time forest fire detection with wireless sensor networks, in: International Conference on Wireless Communications, Networking and Mobile Computing (IWCMC '05), volume 2, IEEE, 2005, pp. 1214–1217.
[8] G. Guangmeng, Z. Mei, Using MODIS Land Surface Temperature to Evaluate Forest Fire Risk of Northeast China, IEEE Geoscience and Remote Sensing Letters 1 (2004) 98–100.
[9] T.-S. Chen, H.-W. Tsai, C.-P. Chu, Adjustable convergecast tree protocol for wireless sensor networks, Computer Communications 33 (2010) 559–570.
[10] O. D. Incel, B. Krishnamachari, Enhancing the Data Collection Rate of Tree-Based Aggregation in Wireless Sensor Networks, in: 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, IEEE, 2008, pp. 569–577.
[11] E. de Souza, I. Nikolaidis, An exploration of aggregation convergecast scheduling, Ad Hoc Networks 11 (2013) 2391–2407.
[12] X. Xu, X. Y. Li, X. Mao, S. Tang, S. Wang, A Delay-Efficient Algorithm for Data Aggregation in Multihop Wireless Sensor Networks, IEEE Transactions on Parallel and Distributed Systems 22 (2011) 163–175.
[13] G.-s. Ahn, S. G. Hong, E. Miluzzo, A. T. Campbell, F. Cuomo, Funneling-MAC: a localized, sink-oriented MAC for boosting fidelity in sensor networks, in: Proceedings of the 4th international conference on embedded networked sensor systems - SenSys '06, ACM Press, Boulder, Colorado, USA, 2006, p. 293.
[14] O. D. Incel, A. Ghosh, B. Krishnamachari, Scheduling algorithms for tree-based data collection in wireless sensor networks, in: Theoretical Aspects of Distributed Computing in Sensor Networks, Springer Berlin Heidelberg, 2011, pp. 407–445.
[15] S. Gandham, M. Dawande, R. Prakash, Link scheduling in wireless sensor networks: Distributed edge-coloring revisited, Journal of Parallel and Distributed Computing 68 (2008) 1122–1134.
[16] J. Gronkvist, Assignment methods for spatial reuse TDMA, in: 1st ACM international symposium on Mobile ad hoc networking & computing, ACM MobiHoc'00, 1, IEEE, Boston, Massachusetts, USA, 2000, pp. 119–124.
[17] B. Dezfouli, M. Radi, S. A. Razak, K. Whitehouse, K. A. Bakar, T. Hwee-Pink, Improving Broadcast Reliability for Neighbor Discovery, Link Estimation and Collection Tree Construction in Wireless Sensor Networks, Computer Networks 62 (2014) 101–121.
[18] S. Ramanathan, A unified framework and algorithm for channel assignment in wireless networks, Wireless Networks 5 (1999) 81–94.
[19] S. C. Ergen, P. Varaiya, TDMA scheduling algorithms for wireless sensor networks, Wireless Networks 16 (2009) 985–997.
[20] L. Bao, J. J. Garcia-Luna-Aceves, A new approach to channel access scheduling for Ad Hoc networks, in: Proceedings of the 7th annual international conference on Mobile computing and networking - (MobiCom '01), ACM Press, Rome, Italy, 2001, pp. 210–221.
[21] L. Tang, Y. Sun, O. Gurewitz, D. B. Johnson, PW-MAC: An energy-efficient predictive-wakeup MAC protocol for wireless sensor networks, in: Proceedings of the 30th IEEE International Conference on Computer Communications, INFOCOM'11, IEEE, 2011, pp. 1305–1313.
[22] R. Rozovsky, P. R. Kumar, SEEDEX: A MAC protocol for ad hoc networks, in: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing - MobiHoc '01, ACM Press, Long Beach, CA, USA, 2001, p. 67.
[23] L. Bao, J. Garcia-Luna-Aceves, Channel access scheduling in ad hoc networks with unidirectional links, in: Proceedings of the 5th international workshop on Discrete algorithms and methods for mobile computing and communications - DIALM '01, ACM Press, 2001, pp. 9–18.
[24] I. Rhee, A. Warrier, DRAND: Distributed Randomized TDMA Scheduling for Wireless Ad Hoc Networks, IEEE Transactions on Mobile Computing 8 (2009) 1384–1396.
[25] H. Choi, J. Wang, E. a. Hughes, Scheduling for information gathering on sensor network, Wireless Networks 15 (2007) 127–140.
[26] S.-Y. Chae, K. Kang, Y.-J. Cho, A scalable joint routing and scheduling scheme for large-scale wireless sensor networks, Ad Hoc Networks 11 (2013) 427–441.
[27] Y. Li, L. Guo, S. K. Prasad, An Energy-Efficient Distributed Algorithm for Minimum-Latency Aggregation Scheduling in Wireless Sensor Networks, in: 30th International Conference on Distributed Computing Systems (ICDCS'10), IEEE, Genova, 2010, pp. 827–836.
[28] D. Chafekar, V. S. a. Kumar, M. V. Marathe, S. Parthasarathy, A. Srinivasan, Approximation Algorithms for Computing Capacity of Wireless Networks with SINR Constraints, The 27th Conference on Computer Communications, (INFOCOM'08) (2008) 1166–1174.
[29] H.-W. Tsai, T.-S. Chen, Minimal Time and Conflict-Free Schedule for Convergecast in Wireless Sensor Networks, in: IEEE International Conference on Communications (ICC'08), IEEE, Beijing, China, 2008, pp. 2808–2812.
[30] Y. Zhang, S. Gandham, Q. Huang, Distributed Minimal Time Convergecast Scheduling for Small or Sparse Data Sources, in: 28th IEEE International Real-Time Systems Symposium (RTSS '07), IEEE, Washington, DC, USA, 2007, pp. 301–310.
[31] M.-S. Pan, Y.-C. Tseng, Quick convergecast in ZigBee beacon-enabled tree-based wireless sensor networks, Computer Communications 31 (2008) 999–1011.
[32] M. Radi, B. Dezfouli, K. Abu Bakar, S. Abd Razak, Integration and Analysis of Neighbor Discovery and Link Quality Estimation in Wireless Sensor Networks, The Scientific World Journal 2014 (2014) 1–23.
[33] A. Kesselman, D. R. Kowalski, Fast distributed algorithm for convergecast in ad hoc geometric radio networks, Journal of Parallel and Distributed Computing 66 (2006) 578–585.
[34] V. Rajendran, K. Obraczka, J. J. Garcia-Luna-Aceves, Energy-efficient collision-free medium access control for wireless sensor networks, in: Proceedings of the first international conference on Embedded networked sensor systems - SenSys '03, ACM Press, Los Angeles, California, USA, 2003, p. 181.
[35] A. Barroso, U. Roedig, C. Sreenan, μ-MAC: an energy-efficient medium access control for wireless sensor networks, in: Proceeedings of the Second European Workshop on Wireless Sensor Networks, (EWSN 2005), IEEE, 2005, pp. 70–80.
[36] V. Rajendran, J. Garcia-Luna-Aceves, K. Obraczka, Energy-efficient, application-aware medium access for sensor networks, in: IEEE International Conference on Mobile Adhoc and Sensor Systems Conference (MASS'05), IEEE, Washington, DC, USA, 2005, pp. 623–630.
[37] G. Lu, B. Krishnamachari, Minimum latency joint scheduling and routing in wireless sensor networks, Ad Hoc Networks 5 (2007) 832–843.
[38] B. Hohlt, L. Doherty, E. Brewer, Flexible power scheduling for sensor networks, in: Proceedings of the third international symposium on Information processing in sensor networks - IPSN'04, ACM Press, Berkeley, California, USA, 2004, p. 205.
[39] B. Hohlt, E. Brewer, Network power scheduling for TinyOS applications, in: Second IEEE international conference on Distributed Computing in Sensor Systems, DCOSS'06, pp. 443–462.
[40] W.-Z. Song, R. Huang, B. Shirazi, R. LaHusen, TreeMAC: Localized TDMA MAC protocol for real-time high-data-rate sensor networks, Pervasive and Mobile Computing 5 (2009)

750–765.
[41] M. Radi, B. Dezfouli, K. a. Bakar, S. a. Razak, M. Lee, Network Initialization in Low-Power Wireless Networks: A Comprehensive Study, The Computer Journal (2013).
[42] M. Radi, B. Dezfouli, K. A. Bakar, S. A. Razak, M. Lee, LINKORD: link ordering-based data gathering protocol for wireless sensor networks, Computing (2014).
[43] OMNeT++, The OMNeT++ Network Simulation Framework, http://www.omnetpp.org, 2014.
[44] B. Dezfouli, M. Radi, S. A. Razak, T. Hwee-Pink, K. A. Bakar, Modeling low-power wireless communications, Journal of Network and Computer Applications (2014).
[45] A. Iyer, C. Rosenberg, A. Karnik, What is the right model for wireless channel interference?, IEEE Transactions on Wireless Communications 8 (2009) 2662–2671.
[46] G. Halkes, K. Langendoen, Experimental Evaluation of Simulation Abstractions for Wireless Sensor Network MAC Protocols, EURASIP Journal on Wireless Communications and Networking 2010 (2010) 1–11.
[47] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, D. Culler, Exploiting the Capture Effect for Collision Detection and Recovery, in: The Second IEEE Workshop on Embedded Networked Sensors, 2005. EmNetS-II., IEEE, Sydney, Australia, 2005, pp. 45–52.
[48] B. Dezfouli, M. Radi, K. Whitehouse, S. A. Razak, H.-p. Tan, CAMA: Efficient Modeling of the Capture Effect for Low-Power Wireless Networks, ACM Transactions on Sensor Networks 11 (2014) 1–43.
[49] M. Z. Zamalloa, B. Krishnamachari, An analysis of unreliability and asymmetry in low-power wireless links, ACM Transactions on Sensor Networks 3 (2007) 63–81.
[50] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, D. Culler, Tinyos: An operating system for sensor networks, in: Ambient intelligence, volume 35, Springer Verlag, 2005, pp. 115–148.
[51] M. Radi, B. Dezfouli, K. A. Bakar, S. A. Razak, M. Lee, Network Initialization in Low-Power Wireless Networks: A Comprehensive Study, The Computer Journal (2013).
[52] C. Zhu;, M. Corson, A five-phase reservation protocol (FPRP) for mobile ad hoc networks, Wirless Networks 7 (2001) 371–384.