IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 3, 2020, accepted August 3, 2020, date of publication August 6, 2020, date of current version August 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3014802

A Distributed TDMA Scheduling Algorithm Using Topological Ordering for Wireless Sensor Networks

THANH-TUNG NGUYEN, TAEJOON KIM, (Member, IEEE), AND TAEHONG KIM, (Member, IEEE)

School of Information and Communication Engineering, Chungbuk National University, Cheongju 28644, South Korea

Corresponding author: Taehong Kim (taehongkim@cbnu.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1F1A1059408).

ABSTRACT In distributed wireless sensor networks (WSNs), TDMA protocols are a widely applied solution when a reliable data-transferring mechanism is in demand. However, to produce a TDMA schedule is not an easy task. During the scheduling, there can be a large number of conflicts due to the nature of radio access, which leads to a waste of operating time and energy. In this paper, we first present and discuss the concept of Topological Ordering, which is based mainly on the local neighborhood size. It is aimed to create an order of scheduling that helps reduce the competition and conflicts. Next, we propose a fast and effective distributed scheduling algorithm using Topological Ordering, called DSTO, to create a collision-free TDMA schedule. This algorithm is promising in terms of reducing running time and message collisions during the scheduling phase, which in turn reduces overall message overhead. At the same time, efficient time slot allocation, which translates to a shorter frame, is guaranteed. We implement DSTO together with two other distributed scheduling algorithms on OPNET Network Modeler and thoroughly analyze the comparative results to prove its validity and effectiveness.

INDEX TERMS Distributed, DSTO, scheduling algorithm, TDMA, topological ordering, wireless sensor networks.

I. INTRODUCTION

Wireless sensor networks (WSNs) have received increasing interests from both the academic community and the industries as they have been proved to have many real-world applications including environmental monitoring [1]–[3], pollution monitoring [3], [4], disaster forecast, prevention and management [5]–[7], structural health monitoring [8]–[10], wildlife tracking [11], [12], cattle farm monitoring [13]–[15], and most recently the Internet of Things [16], [17]. In WSNs, carrier sensing multiple access (CSMA) has been widely used as a medium access control (MAC) protocol as it has several advantages such as robustness, flexibility, and full distributedness [18]. However, it is prone to collisions because two or more neighboring nodes may be transmitting at the same time. Even though control packets, such as RTS, CTS, and ACK, are helpful in solving this problem, they may add up to the overhead costs by 40% [19].

On the other hand, time division multiple access (TDMA) has proven to be a more reliable protocol with respect to its collision-free data-transferring. However, scheduling, or time slot allocation, is always a challenging issue, especially in multihop networks. There have been various approaches to tackle this problem. For example, the authors in [20] proposed an algorithm that can provide a distributed collision-free schedule. However, it requires all nodes to be strictly synchronized, which is not a realistic assumption for real-life networks. Rhee *et al.* proposed a distributed version of the randomized scheduling algorithm RAND [21], called DRAND [22]. Not only DRAND can allocate slots without collisions, it does not require clock-synchronization or global information. However, due to its random nature, DRAND may have a large number of collisions during the scheduling phase, which leads to long running time and generating a large amount of message overhead. For these reasons,

The associate editor coordinating the review of this manuscript and approving it for publication was Juan Liu.

DRAND may not suitable for networks with frequent changes in topology. For example, UAV networks for monitoring farm cattle [13] have to deal with sudden topology changes due to node or communication failures. For this type of networks, a TDMA scheduling algorithm with short running time and low message overhead is promising as it can quickly adapt to changes and efficiently reduce energy consumption. To overcome the problem of DRAND, the authors in [23]–[25] proposed various priority factors such as energy-topology and inter-node distance. Evidently, this helps reduce collisions, but does not completely eliminate the problem.

Our early effort to tackle this problem was presented in [37]. In the paper, we introduced Topological Ordering and a simple scheduling algorithm. The simulation results showed comparisons between DSTO and DRAND in terms of running time and number of rounds to complete scheduling. In this paper, we extend the previous work with the following contributions:

- We improve the scheduling algorithm to handle diverse cases in practical WSNs by adding the checking condition for message reception status and slot awareness of nodes. This increases completeness, efficiency, and applicability.
- We provide theoretical analysis on the complexity and running time of the algorithm, and implementation guidelines (e.g., types of messages), so that readers can have a better understanding on the concept and easily re-implement DSTO.
- We conduct more extensive evaluations to demonstrate the effectiveness of DSTO over two other scheduling algorithms. From the comparative evaluation results, diverse performance aspects of DSTO such as running time, message overhead, and slot allocation are thoroughly analyzed to prove the efficiency of our algorithm.

The rest of our paper is organized as follows. Section II analyzes the related works on scheduling in WSNs. Section III provides the theoretical and formal definition for the TDMA scheduling problem, alongside basic notations. In Section IV, we explain the idea of TO in details. Section V presents our proposed algorithm and its employed message structures. Section VI presents the theoretical analysis of DSTO. In Section VII, we describe the simulation setup and analyze the results to validate the effectiveness of DSTO. Lastly, Section VIII concludes this paper.

II. RELATED WORK

In TDMA, to produce a collision-free schedule using the minimum number of slots is a NP-Complete problem [21]. There have been several different approaches, which can mainly be categorized into centralized and distributed scheduling. The early works were mostly centralized scheduling and focused on looking for an optimal solution for the scheduling problem [26]–[29]. However, as these algorithms require full knowledge on network topology, they are not suitable for large scale multi-hop networks, even though their expected complexity is just $O(n)$, where n is the number of nodes in the network.

On the other hand, distributed scheduling algorithms work based on local information, usually on neighbors within a two-hop distance. Therefore, they are scalable and more suitable for dynamic networks with frequent changes in network topology. However, a major disadvantage is that their slot allocation performance is not as good as that of centralized algorithms. In NAMA [30], the authors proposed a hashing technique to determine contenders' priorities. Time is divided into blocks of several sections. Each section is further divided into parts and then time slots. When a node has data packets to send, it chooses a part and a time slot of that part to contend. Furthermore, the last section of a block is called *the membership section* and used exclusively for management purposes. Each slot of this section is again divided into segments for newcomer nodes to send signals that contain their IDs and a part they wish to contend. An advantage of this algorithm is that it does not require a contention phase or global topology information. However, its computational complexity can be $O(n^2)$, as the priorities of all the two-hop neighbors are to be calculated at each node every slot, and it also requires nodes to be time-synchronized. Rozovsky and Kumar [31] proposed SEEDEX, which aims to avoid collisions while not having to reserve for every packet with a hash function as well. The drawback of SEEDEX is that it is prone to collisions when two nodes pick and transmit at the same slot.

The authors in [32] proposed Five-phase Reservation Protocol (FPRP), a heuristic distributed scheduling algorithm for dynamic slot allocation. FPRP runs in cycles, each of which consists of a Reservation Frame (RF) and several Information Frames (IFs). Each RF contains a number of Reservation Slots (RSs) and each IF contains the same number of corresponding Information Slots (ISs). To reserve an IS, nodes run 5 reservation steps during their corresponding RSs. DICSA [33], a distributed and concurrent link scheduling algorithm, introduced a two-state-machine algorithm, primary and secondary, that allows each node and its neighbors to participate in each other's slot reservation process. To ensure a collision-free schedule, both of these algorithms have to maintain a common list of *forbidden slots*. Although DICSA does not require time synchronization, its performance may be varied as time difference among nodes can cause slot overlaps and collisions.

Bhatia and Hansdah [20] introduced DTSS, a distributed algorithm, to solve the *spatially correlated contention* between neighboring nodes. The authors proposed the strong and weak-conflict models to support all unicast, multicast, and broadcast. In unicast and multicast modes, the algorithm allows nodes, even two-hop neighbor nodes, to have the same time slot as long as they do not cause interference to one another's receivers. DTSS performs well in terms of scheduling time, however, since it requires all nodes to be strictly synchronized, extra time and network traffic for running time-synchronization algorithms are needed. Moreover, the frame length in DTSS must be set to be equal or larger than δ, where δ is the maximum size of two-hop neighborhoods, which leads to low channel utilization rate. Lakhlef,

Raynal and Taiani [34] proposed a frugal vertex coloring algorithm for tree-based distributed broadcast networks to avoid conflicts and collisions. This algorithm can reduce the execution time and number of broadcast messages and improve channel utilization by using the minimum number of slots. However, in the paper, the simulations were only performed on networks with the maximum node degree of 7. In large-scale networks, the algorithm may suffer from long running time and high message overhead due to a high node density.

Rhee *et al.* [22] proposed a distributed randomized time slot scheduling algorithm, named DRAND, based on a centralized algorithm called RAND [21]. DRAND is employed by the hybrid protocol named Zebra-MAC [18], or ZMAC, as its scheduling algorithm. In DRAND, nodes work in four different states *IDLE, REQUEST, GRANT* and *RELEASE*. At *IDLE* state, in each round, if a node u wins the lottery, it moves to *REQUEST* state and broadcasts a *request* to its one-hop neighbors. If this *request* is answered with a *grant* by all of the one-hop neighbors, the node can choose a minimum free time slot. After that, it moves to *RELEASE* state and broadcasts a *release* which contains the information about its chosen slot. However, if at least one of the one-hop neighbors answers with a *reject*, the round is considered unsuccessful. Node u then has to send a *fail* to its one-hop neighbor to signal a failed round and then moves back to *IDLE* state and starts again later. One-hop neighbors that have sent a *grant* to u also have to move from *GRANT* state back to *IDLE* state (if they have not decided on their slot) or *RELEASE* state (if they have). DRAND guarantees collision free slot assignment for any network with message complexity $O(\delta)$, where δ is the maximum number of nodes in any two-hop neighborhood. However, DRAND also has its drawbacks, which we define as *random competition problem* in slot scheduling in distributed WSNs.

Firstly, in DRAND, several nodes may send *requests* at the same time, which certainly results in collisions. Moreover, a node may send a *request* while some of its one-hop neighbors are in *GRANT* state and ends up being rejected by those nodes. Furthermore, due to the random nature, a node may lose the lottery several times before being allowed to send a *request*. Even when it does send, there is no guarantee that it can reserve a time slot in that round. Each node may have to go through several trials (rounds) and fails before being able to reserve a slot, which causes higher running time and more energy consumption. Secondly, when a node u is rejected by a one-hop neighbor, it has to send a *fail* to all of the one-hop neighbors. However, this message may not successfully reach some of them, which will keep waiting and periodically sending *grants* to u until they recognize u has failed. All the above-mentioned points show that while able to produce a collision-free schedule, DRAND is ineffective in terms of running time, message overhead, and energy consumption.

L-DRAND++ [25], E-T-DRAND [23] and ET-BT-DRAND [24], which are DRAND's variants, were proposed to combat the *random competition problem* of DRAND. They employ inter-node distance information and energy-topology, respectively, as priority factors to make sure that in each neighborhood, there is only one node with the highest calculated priority that can compete for its time slot at any moment and other nodes have to wait if they have lower priorities. Similarly to DRAND, after the highest prioritized node, named A, in the neighborhood has selected a time slot, it broadcasts a *RELEASE* message to notify its neighbors. Here lies a major concern on these algorithms as there seems to be no reliable mechanism to ensure this information will be successfully delivered to neighbor nodes with lower priorities. In this case, these nodes will wait indefinitely since they have not received A's message. Thus, these algorithms do not work in our simulation settings and we exclude them from the performance evaluation.

In [35], Batta *et al.* proposed an improved version of DRAND, called I-DRAND, which aims to provide a TDMA schedule for tree-based distributed networks. This algorithm follows almost exactly DRAND's four-state model. The only difference is that nodes in I-DRAND collect requests from its neighbor during one round and only send a grant to the priority node at the end of that round. The order of priority is: its parent, its child and a sibling that has the highest degree. By restricting sending of grants/rejects at the end of each round, this algorithm can reduce the number of collisions and conflicts, and thus reduce execution time. However, the *random competition problem* of DRAND as presented above is not yet solved. Another tree-based distributed TDMA scheduling algorithm is presented in [36]. This algorithm works in a bottom up manner. Each parent collects its children's calculated weights and grandchildren's assigned slots through request message from the children. Once having received requests from all of its children, the parent performs scheduling for them by itself. However, the algorithm does not provide an acknowledge mechanism for request messages from child nodes to their parent, thus they may have to send requests continuously every round until receiving a schedule from the parent, leading to more generated message overhead.

Table 1 summarizes diverse aspects of TDMA scheduling algorithms reviewed in this section. In this paper, we propose a distributed scheduling algorithm using Topological Ordering. By creating an order to tackle the *random competition problem* in slot scheduling in distributed WSNs, our algorithm allows nodes to schedule in an *orderly* fashion to reduce scheduling delay and collisions between neighbors, which helps lower running time, generated message transmissions, and numbers of rounds required to finish scheduling as well as provides efficient slot allocation.

III. PRELIMINARIES

We first define the theoretical problem, which includes the network model and fundamental concepts of graph theories in Subsection III-A. Then, we proceed to provide the basic notations used in the proposed algorithm in Subsection III-B.

TABLE 1. Summary of related works.

Algorithm	Type	Time sync	Required network information	Support mode	Frame length	Complexity	Note
GA [27]	Centralized	Not needed	Whole	Broadcast	Optimum	$O(n)$	n is the number of nodes in the network
Genetic-Fix [28]	Centralized	Needed	Whole	Broadcast	Optimum	$O(n^2)$	
ETDMA-GA [30]	Centralized	Not needed	Whole	Broadcast	Optimum	$O(n \cdot m)$	n is the number of nodes. m is the maximum number of slots
Bsch [29]	Centralized	Not needed	Whole	Broadcast	Optimum	$O(\theta)$	θ is the thickness of the graph
RAND [22]	Centralized	Not needed	Whole	Broadcast	Optimum	$O(e \cdot \rho)$	e is the number of edges. ρ is the maximum graph degree.
NAMA [31]	Distributed	Needed	Within two hops	Broadcast	Prefixed	N/A	There is no complexity since nodes compete for slots on an as-needed basis.
SEEDEX [32]	Distributed	Needed	Within two hops	Broadcast	Prefixed	N/A	
FPRP [33]	Distributed	Needed	Within two hops	Broadcast	Prefixed	N/A	
DTSS [21]	Distributed	Needed	Within two hops	All	Prefixed	$O(\delta)$ (Broadcast)	δ is the maximum size of two-hop neighborhoods.
F3C [35]	Distributed	Needed	Within two hops	Broadcast	Optimum	$O(\delta)$	
DICSA [34]	Distributed	Not needed	Within two hops	Broadcast	Flexible	$O(\delta)$	
DRAND [23]	Distributed	Not needed	Within two hops	Broadcast	Flexible	$O(\delta)$	
L-DRAND++ [26]	Distributed	Not needed	Within two hops	Broadcast	Flexible	$O(\delta)$	
E-T-DRAND [25]	Distributed	Not needed	Within two hops	Broadcast	Flexible	$O(\delta)$	
ET-BT-DRAND [25]	Distributed	Not needed	Within two hops	Broadcast	Flexible	$O(\delta)$	
I-DRAND [36]	Distributed	Not needed	Within two hops	Broadcast	Flexible	$O(\delta)$	
WB-DTSA [37]	Distributed	Not needed	Within two hops	Broadcast	Flexible	$O(\delta)$	

A. TDMA SLOT ASSIGNMENT IN WIRELESS SENSOR NETWORKS

A WSN can be represented by an undirected graph $G = (V, E)$, where V is the set of vertices, or nodes, and E is the set of bidirectional links such that link $e = (u, v)$ exists, if and only if node $u, v \in V$ are in each other's transmission range. We assume that all node have identical transmission radius. Thus, two nodes' transmitting signals simultaneously will cause interference among themselves or at a third node, if they are one- or two-hop away, respectively.

Therefore, we define the problem as producing a schedule in which no two nodes within two hops from each other have the same slot. In Graph theory, this problem can be referred to as *vertex-coloring*, which is NP-hard [21]. Two vertices $u, v \in G$ can have the same color if and only if both of the following conditions hold.

- $e = (u, v) \notin E$.
- There does not exist a vertex x such that $e_1 = (u, x) \in E$ and $e'_1 = (x, v) \in E$.

In this paper, to determine the performance of a coloring—or slot asignment—algorithm, we consider the following aspects.

- **Running Time *T*** is the amount of time taken to color all vertices in V, or for all nodes to acquire a time slot. Shortening running time is a important goal for scheduling algorithms since it would reduce energy consumption and increase QoS.
- **Number of Transmissions *C*** is the number of messages transmitted by all nodes to produce a complete schedule. A lower number of transmitted messages also reduces energy consumption and running time, which leads to better QoS.
- **Frame Length *L*** is the number of slots needed to accommodate all nodes, which is also the number of colors needed for all vertices in V. A shorter frame directly translates into better channel utilization.

In distributed WSNs, message transmissions during the process of TDMA scheduling are prone to collisions, especially in high density networks, as neighbors are not aware of one another's transmission timing. This detrimentally affects the performance of the whole network, because collided messages need to be retransmitted, which requires longer running time and consumes more energy. Therefore, with the aim of solving the *random competition problem* and improving the performance of WSNs, we propose the concept of Topological Ordering alongside a quick and efficient scheduling algorithm.

B. BASIC NOTATIONS

This subsection provides the basic notations used for our algorithm descriptions. In the interest of clarity, from hereon, all neighbors within two hops are simply referred to as neighbors. Also, neighbor nodes which are one-, two-hop away are referred to as one- and two-hop neighbors, respectively. The following basic notations are provided to give readers a better understanding of our algorithm.

- $N_1(u)$ is the set of one-hop neighbors of node u. If node $v \in N(u)$, u and v can successfully receive messages from one another. The number of one-hop neighbors is $|N_1(u)|$.

- $N(u)$ is the set of all neighbors of node u. The number of neighbors is $|N(u)|$. We have $N_1(u) \subseteq N(u)$. In addition, as $N_1(u)$ and $N(u)$ are defined, it is not necessary to define another set for two-hop neighbors. because if node $v \in N(u)$ and $v \notin N_1(u)$, v is identified as a two-hop neighbor.
- $val(u)$ is the randomly self-assigned numerical value of node u.
- $S(u)$ is the vector of slots assigned by nodes in $N_1(u)$.
- $P(u)$ is the Priority value of node u. This is explained further in Subsection IV-A.

All the notations, along with other above-mentioned ones, are summarized in Table 2.

TABLE 2. Abbreviation table.

Abbreviation		Meaning
General terms	TO	Topological Order
	TI	Topological Information
	TOT	Topological Order Table
	SAT	Slot-Awareness Table
	NDP	Neighbor Discovery Phase
	SCP	Scheduling Phase
	RL	RELEASE
	RLC	RELEASE-CONFIRMATION
	FW	FORWARD
	FWC	FORWARD-CONFIRMATION
Math and pseu-docode notations	$addr(u)$	Address of node u
	$N(u)$	Node u's list of neighbors within two hops
	$\|N(u)\|$	Total number of node u's neighbors
	$N_1(u)$	Node u's list of neighbors within one hop
	$\|N_1(u)\|$	Total number of node u's one-hop neighbors
	$val(u)$	Random numerical value of node u
	$P(u)$	Priority value of node u
	$SAT_u(i,j)$	Entry on node u's SAT specifying u is aware that node i has received information about node j's slot assignment
	$R(u)$	List of one-hop neighbors that have confirmed the reception of node u's *RL* message
	$S(u)$	List of slots already used by nodes in $N_1(u)$
	$U_u(v)$	List of node u's one-hop neighbors whose slot information node v is not aware
	$relTimer$	Waiting period between *RL* messages
	$fwTimer$	Waiting period between *FW* messages
	$relNb(u)$	Neighbor node from which node u has most recently received a *RL* message
	$fwTg(u)$	Current target node for node u's *FW* message
	dst and src	The destination and source of a message

IV. TOPOLOGICAL ORDERING AND SLOT-AWARENESS TABLE

We describe how topological information (TI), which consists of the total number of neighbors $|N(u)|$ and the random value $val(u)$, can be used by each node to create a TO for its two-hop neighborhood in Subsection IV-A. Then, in Subsection IV-B, we introduce Slot-Awareness Table (SAT)–a tool for nodes to keep track of the changes in slot assignment inside its neighborhood.

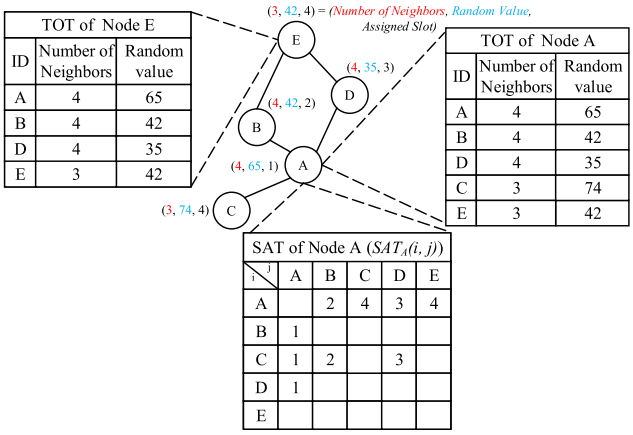

FIGURE 1. Example of SAT and TOT.

A. TOPOLOGICAL ORDERING IN DISTRIBUTED WSNs

In DSTO, the TI collected by each node is used to produce a TO, or scheduling order, consisting of the node itself and its neighbors within a two-hop distance. With this order, each node knows exactly when its turn is or in other words, how many neighbors for which it has to wait before it is allowed to reserve a time slot. TO is a list of descending priority values. A node u's priority value is denoted as $P(u)$ and determined by the u's total number of neighbors $|N(u)|$ and a randomized numerical value $val(u)$. Between any two neighbor nodes u and v, we have:

$$\begin{cases} P(u) > P(v), & if\ (|N(u)| > |N(v)|)\ or \\ & (|N(u)| = |N(v)| \& val(u) > val(v)) \\ P(u) < P(v), & otherwise \end{cases} \quad (1)$$

The numerical value, which is randomly chosen between the range of int32 by each node, is used in case u and v have the same number of neighbors to avoid possible deadlocks. In this case, the node with the higher value has higher priority. Theoretically speaking, it is possible that two or more neighbor nodes may pick the same number, but the possibility of that is extremely low. Therefore, in this paper, we assume each node has a unique value. This assumption is reasonable as in actual networks, node can be assigned a unique identification to specify their priority, roles, etc. Fig. 1 shows an example of Topological Ordering in a Distributed WSN. In A's Topological Order Table (TOT), we can see that:

$$P(A) > P(B) > P(D) > P(C) > P(E) \quad (2)$$

Thus, the sequence of scheduling nodes from A's point of view is $ABDCE$ with A as the first node and E as the last. In E's TOT, there are only four nodes as C is three-hop away from E. There, the sequence of scheduling nodes at E is

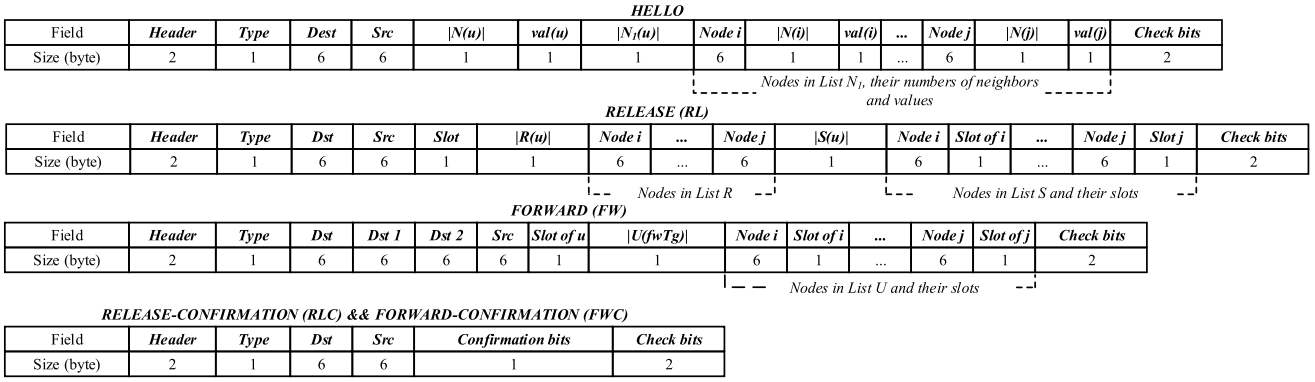

FIGURE 2. The structures of messages.

only *ABDE*. After node *A* has finished assigning a time slot, it will be removed from its own and other nodes' TOTs. Node *A*'s position on these tables will be filled by another node to keep the algorithm working continuously.

B. SLOT-AWARENESS TABLE

The correctness of DSTO requires each node to be aware of its neighbors' assigned slots and also their awareness about nodes inside the neighborhood. Thus, each node stores exchanged information inside an SAT. In a node *A*'s SAT, $SAT_A(i, j) = a$ ($\forall\ a > 0$) shows that node *A* is aware that node *i* has received information about node *j*'s taking time slot *a*. If *i* is *A*, it indicates that *A* has had information about node *j*'s time slot. In cases where *j* is *A*, it means *A* has received *j*'s confirmation on its own time slot. Fig. 1 shows an example of *SAT*. $SAT_A(C, A)$ is set to be 1 after node *C* has successfully confirmed to *A* that it is aware that *A* has chosen slot 1. In addition, $SAT_A(C, B)$ is 2 as *A* knows *C* is aware that *B* has picked slot 2. The rest of this table will be further discussed in Subsection IV-E.

V. DSTO, THE SCHEDULING ALGORITHM USING TOPOLOGICAL ORDERING

In our algorithm, there are two main phases. The first phase is Neighbor Discovery Phase (NDP), in which neighboring nodes exchange TI with their one-hop neighbors. After this phase, each node *u* has the information sufficient to construct a TO of its two-hop neighborhood. In Scheduling Phase (SCP), node *u* will perform slot assignment according to this TO.

In Subsection V-A, we first introduce the algorithm for exchanging *HELLO* messages carrying TI during the NDP. Then, the main scheduling algorithm using Topological Ordering is described in Subsection V-B. To help readers have a better understanding and avoid any confusion, Subsection V-C is spent to describe in detail the types of messages used in this paper. Next, in Subsection V-D, we describe second component of the algorithm and its logical conditions to handle all possible cases. Lastly, in Subsection V-E, an example is presented with a thorough explanation to provide a more in-depth understanding into our scheduling algorithm.

A. NEIGHBOR DISCOVERY PHASE

Each node repeatedly broadcasts *HELLO* messages, whose structure is shown in Fig. 2, that contain its TI and its one-hop neighbors' TI at random moments during the NDP. This ensures that each node is constantly updated on all neighbors within two hops. Upon reception of a *HELLO* from a neighbor node *v*, node *u* first adds *v* into its $N(u)$ and $N_1(u)$, respectively, if *v* does not yet exist in these lists (Algorithm 1, lines 2 – 8) and then updates its TOT with *v*'s TI (Algorithm 1, line 9). Next, node *u* checks if each node *x* in the list of *v*'s one-hop neighbors $N_1(v)$ are not in $N(u)$ and $N_1(u)$. If this is true, node *x* must be *u*'s two-hop neighbor and is added to $N(u)$ (Algorithm 1, line 10-15).

After transmitting a *HELLO* message, each node has to wait for a period of time, called *helloTimer*, before broadcasting again (Algorithm 1, line 17). The next *HELLO* message will contain updated information on node *u* and its one-hop neighbors (Algorithm 1, line 18-22). A similar neighbor discovery method is also used by DRAND [22] in its ns2 simulation. The length of the NDP, denoted *ndTimer*, is set to be long enough for all nodes to discover their neighbors within a two-hop range. After *ndTimer* expires, each node stops broadcasting *HELLO*s and transfers into the SCP.

B. SCHEDULING PHASE

As discussed in the previous section, each node in DSTO performs scheduling tasks according to the TO presented in its TOT. A node ranked 5 on its own TOT has to wait until the four neighbors ranked from 1 to 4 on its TOT to have assigned their slots. Function *Main*() in Algorithm 2 shows the pseudocode of DSTO's main scheduling algorithm during the SCP. The detailed explanation of this algorithm is given below.

After NDP has ended, all nodes automatically transit to the SCP. At the beginning of this phase, if a node *u* is on top of its TOT, it automatically picks the minimum slot available (slot 1) and starts broadcasting a *RELEASE (RL)* message, which contains information about the slot, to its one-hop

Algorithm 1 Algorithm for Neighbor Discovery Phase

PROCEDURE *HelloHandler*()
INPUT: Node u
$addr(u)$, $val(u)$, $N(u)$, $N_1(u)$
OUTPUT: Broadcasting *HELLO* messages
Collecting neighbors' information

1: **while** *ndTimer has not yet expired* **do**
2: **if** *u receives a HELLO message m from v* **then**
3: **if** node $v \notin N(u)$ **then**
4: **add** $v \rightarrow N(u)$
5: **end if**
6: **if** node $v \notin N_1(u)$ **then**
7: **add** $v \rightarrow N_1(u)$
8: **end if**
9: **update** $N(v)$, $val(v)$ **in** TOT
10: **foreach** node $x \in N_1(v)$ **do**
11: **if** $x \notin N(u)$ && $x \notin N_1(u)$ **then**
12: **add** $x \rightarrow N(u)$
13: **end if**
14: **update** $N(x)$, $val(x)$ **in** TOT
15: **end foreach**
16: **else**
17: **if** *helloTimer has expired* **then**
18: **set** *HELLO* **with** [$dst = 0xFFFFFFFF$, $src = addr(u)$, $|N(u)|$, $val(u)$]
19: **foreach** node $v \in N_1(u)$ **do**
20: **update** *HELLO* **with** [$addr(v)$, $|N(v)|$, $val(v)$]
21: **end foreach**
22: **send** *HELLO*
23: **end if**
24: **end if**
25: **end while**

Algorithm 2 Main Algorithm in Scheduling Phase

PROCEDURE *Main*()
INPUT: node u, $addr(u)$, $slot(u)$, $R(u)$, $S(u)$, $N_1(u)$
u's *SAT* and *TOT*

1: **while** *node u or any one-hop neighbor does not have a time slot* **do**
2: **if** *u receives a message m from node v* || *SCP starts* **then**
3: **update** SAT
4: **update** TOT
5: **if** *u is top of TOT* **then**
6: pick a minimum available slot
7: **send** *RL* (dst, src = $addr(u)$, $slot(u)$, $R(u)$, $S(u)$)
8: **set** *relTimer* $\leftarrow 4d_{tx}$
9: **else**
10: **if** *type(m)* == *RL* **then**
11: *relNb* $\leftarrow v$
12: Execute *fwAndRelConfirm*(*relNb*)
13: **else if** *type(m)* == *FW* **then**
14: **send** *FWC*(dst = $addr(src(m))$, src = $addr(u)$, 1)
15: **end if**
16: **end if**
17: **end if**
18: **if** *relTimer has expired* && $|R(u)| < |N_1(u)|$ **then**
19: **send** *RL* (dst, src = $addr(u)$, $slot(u)$, $R(u)$, $S(u)$)
20: **set** *relTimer* $\leftarrow 4d_{tx}$
21: **end if**
22: **end while**

neighbors (Algorithm 2, lines 2–8). The detailed descriptions of this message will be provided in Subsection V-C. Node u then waits for the neighbors to confirm they have received the message. The amount of waiting time is set by the release timer, denoted *relTimer*. After that, if there are still one-hop neighbors that have not replied confirming the reception of u's *RL* message, or $|R(u)| < |N_1(u)|$, it rebroadcasts another *RL* (Algorithm 2, lines 18–22). At the time of retransmission, as u have received confirmations from some of its one-hop neighbors, it adds the list of these neighbors, $R(u)$, into the message to ensure that they do not reply more times than necessary. The *RL* procedure only stops once $|R(u)| = |N_1(u)|$, which indicates all one-hop neighbors are aware of u's slot assignment. Additionally, the list of one-hop neighbors and their assigned time slots, $S(u)$, is piggy-backed in *RL*s as well, if it does exist.

On the other hand, if it is not yet node u's turn to schedule when the SCP starts, or it has already finished scheduling, it waits for neighbors' messages. When u receives a *RL* from a neighbor, denoted as *relNb*, it first adds slot information of that neighbor into its SAT and removes that neighbor from its TOT (Algorithm 2, lines 3–4). Subsequently, if u has become the top of its TOT, it chooses a minimum slot available and broadcasts *RL*s exactly as described above (Algorithm 2, lines 5–8). Otherwise, u may have to pass the information on *relNb*'s slot to a target neighbor, denoted *fwTg*, with a *FORWARD (FW)* message or just simply reply *relNb* with a *RELEASE-CONFIRMATION (RLC)* message. Therefore, it executes *fwAndRelConfirm*() in Algorithm 3, which will be described subsequently (Algorithm 2, lines 10-12). Otherwise, if u receives a *FW* from a neighbor, it simply replies with a *FORWARD-CONFIRMATION (FWC)* to the sender (Algorithm 2, lines 13–14). The algorithm ends once all of nodes inside u's one-hop neighborhood has finished their scheduling.

C. COMMUNICATION MESSAGES

Detailed structures of the messages used in the SCP of DSTO are shown in Fig. 2. All messages including *RL, FW, FWC*, and *RLC* are broadcast to all neighbors within a one-hop range. Since broadcasting does not provide an acknowledgment mechanism, we use *RLC* and *FWC* as a way to confirm the receptions of *RL* and *FW*, respectively. This is further explained in the next three paragraphs.

***RL* messages** are broadcast to all one-hop neighbors of the sender u to announce that it has successfully reserved

a time slot. The destination address field (*dst*) is set to be 0xFFFFFFFF. Furthermore, the lists of the sender u's one-hop neighbors that have confirmed the reception of u's *RL* and u's one-hop neighbors that have assigned their time slots (or $R(u)$ and $S(u)$ respectively) are also added as well if they exist. For example, the first *RL* does not contain R because there has not been any node's replying. In addition, S may be empty as well, if the sender is the first node to schedule in its one-hop neighborhood. Due to collision, *RL* messages may not be successfully broadcast to all the neighbors. Thus, retransmission is necessary. After broadcasting a *RL*, node u waits for an amount of time called *relTimer* equal to $4d_{tx}$, in which d_{tx} is the maximum one-way transmission delay between u and its one-hop neighbors, and it is measured during the NDP. The length of *relTimer* is determined based on experimental results.

***FW* messages** can be used to acknowledge the reception of a *RL* from a neighbor–*relNb*, as well as to forward slot information to another one-hop neighbor called *fwTg*. Similar to *RL* messages, its *dst* field is also set to 0xFFFFFFFF. The addresses of two explicit recipients, which are for *relNb* and *fwTg*, are stored in *dst*1 and *dst*2, respectively. In case where the sender no longer has to reply to *relNb*, field *dst*2 will be set at 0×00. The list of two-hop neighbors whose slots *fwTg* are not yet aware of, denoted as U, is also included when it does exist. This type of messages also requires retransmission. The waiting period, *fwTimer*, is set to be $5d_{tx}$ also based on experimental results.

***RLC* and *FWC* messages** are used only to confirm the reception of a *RL* or a *FW*, respectively, so besides identificative and checking bits (e.g, *header* and *src*), they both only have one other field called Confirmation bits to signal that they have successfully received the last message. Both of these messages are only transmitted promptly upon the reception of a *RL* or *FW*. Therefore, they do not require retransmission.

D. FW AND RLC HANDLING ALGORITHM

In Subsection V-B, we mentioned that when u receives a *RL*, it is required to decide whether to only reply to *relNb* with an *RLC* or forward the slot information to *fwTg* with a *FW*. While *RLCs* are just for sending confirmation on the reception of *RL*s, *FW*s are used for confirmation as well as to notify *fwTg*, which is a one-hop neighbor of u that has not decided on its slot and is ranked higher than u and all the other one-hop neighbors in u's TOT. Because if *fwTg* is two-hop away from *relNb* and other one-hop neighbors of u, it should be updated on their scheduling information to avoid choosing a collided slot. If such node cannot be found, there is no need to send a *FW*, so u only confirm to *relNb* with an *RLC* (Algorithm 3, lines 1–4).

Otherwise, if there exists a target node *fwTg*, node u examines two conditions (Algorithm 3, line 7). The first one is

$$\begin{cases} SAT_u(u, relNb) > 0 \\ SAT_u(fwTg, relNb) = 0 \end{cases} \tag{3}$$

Algorithm 3 Algorithm to Handle *FW* and *RLC* in Scheduling Phase

PROCEDURE *fwAndRelConfirm*(*relNb*)
INPUT: node u, $addr(u)$, $slot(u)$, $R(u)$, $S(u)$, $N_1(u)$ u's *SAT* and *TOT*

1: **if** *fwTimer has expired* || *being called from Main()* **then**
2: **find** *fwTg*
3: **if** *fwTg is not found* **then**
4: **send** $RLC(\text{dst} = addr(relNb), \text{src} = addr(u), 1)$
5: **else**
6: **foreach** node $x \in N_1(u)$ && $x \notin N_1(fwTg)$ **do**
7: **if** $SAT_u(u, x) > 0$ && $SAT_u(fwTg, x) == 0$ **then**
8: **if** $x == relNb$ **then**
9: $relFwFlag \leftarrow 1$
10: **else**
11: $otherFlag \leftarrow 1$
12: **add** $[addr(x), SAT_u(u, x)] \rightarrow U_u(fwTg)$
13: **end if**
14: **end if**
15: **end foreach**
16: **if** $otherFlag == 1$ || ($relFwFlag == 1$ && $otherFlag == 0$ && *u is fwTg's one-hop neighbor with least neighbors)* **then**
17: $\text{dst1} \leftarrow addr(fwTg)$
18: $\text{dst2} \leftarrow (u \notin R(relNb))\ ?\ addr(relNb) : 0x00$
19: **send** $FW(\text{dst1}, \text{dst2}, \text{src} = addr(u), SAT_u(u, relNb), U_u(fwTg))$
20: **set** $fwTimer \leftarrow 5d_{tx}$
21: **else**
22: **if** $u \notin R(relNb)$ **then**
23: **send** $RLC(\text{dst} = addr(relNb), \text{src} = addr(u), 1)$
24: **end if**
25: **end if**
26: **end if**
27: **end if**

Here, $SAT_u(u, relNb) > 0$ indicates that node u is aware of the slot assignment of *relNb*. Therefore, if according to u's knowledge, *fwTg* has not been aware of that, which is signaled by $SAT_u(fwTg, relNb) = 0$, it has to send a *FW* to *fwTg*. Thus, it sets *relFwFlag* to 1 (Algorithm 3, lines 7-9). Moreover, condition 2 is

$$\begin{cases} SAT_u(u, x) > 0 \\ SAT_u(fwTg, x) = 0 \end{cases} \quad \forall x \in N_1(u)\ \&\ x \notin N_1(fwTg) \tag{4}$$

This condition holds if there is at least one node x that is a common one-hop neighbor of u and *fwTg*, whose slot information still remains unknown to *fwTg*. In this case, u sets *otherFlag* to 1 to indicate that u has to send a *FW* to notify *fwTg* about x's slot information as well. Additionally, u adds each node x and its slot information into a list denoted as $U_u(fwTg)$ (Algorithm 3, lines 7, 10-12). This check results in

three possible cases, for which a simple graphical illustration is presented in Fig. 3(a).

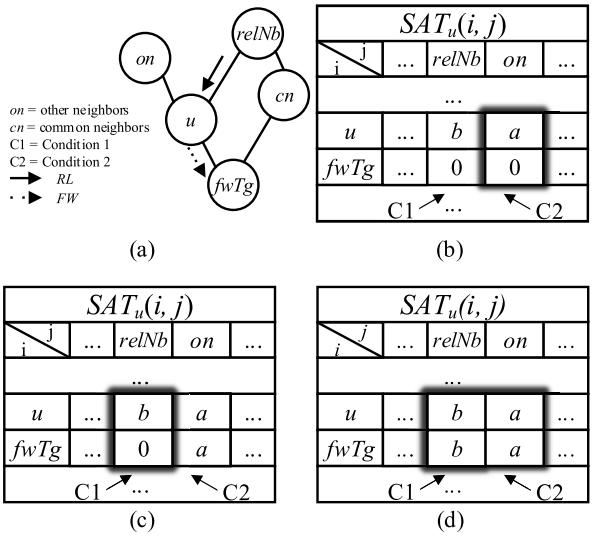

FIGURE 3. Graphical illustration for logical conditions of 3 cases in *fwAndRelConfirm()* algorithm.

Case 1: At least condition 2 is met, or $otherFlag == 1$ (Algorithm 3, line 16). This is illustrated in Fig. 3(b), in which $SAT_u(u, on) = a$ and $SAT_u(fwTg, on) = 0$, where on represents x. In this case, besides $relNb$, there is slot information from other two-hop neighbors of $fwTg$ of which it is yet unaware, which makes it necessary to send a FW to $fwTg$. Therefore $dst1$ is set to address of $fwTg$. Node u also checks if it is required to answer to the node that previously sent it a RL, $relNb$. If yes, $dst2$ is filled with $relNb$'s address. Otherwise, it will be set to 0×00 (Algorithm 3, line 17-18). Next, it broadcasts the FW message to both notify $fwTg$ and confirm on the reception of $relNb$'s previous RL message (Algorithm 3, line 19-20).

Case 2: Only condition 1 is met, or $relFwFlag == 1$ and $otherFlag == 0$ (Algorithm 3, line 16) as illustrated in Fig. 3(c), in which $SAT_u(u, relNb) = b$ and $SAT_u(fwTg, relNb) = 0$ while both of $SAT_u(u, on)$ and $SAT_u(fwTg, on)$ are a. This indicates there is only slot information of $relNb$ to be forwarded. Here, if u is the node that has the least number of neighbors amongst common one-hop neighbors, denoted cn in Fig. 3, of $fwTg$ and $relNb$, u will send a FW. Because, only if there is at least one more common one-hop neighbor between $fwTg$ and $relNb$ besides u, only of them has to forward to $fwTg$ (Algorithm 3, line 16-20). This ensures there are no redundant messages and thus reduces the total number of generated messages.

Case 3: In this case, none of two above-mentioned cases are matched as illustrated in Fig. 3(d). Here, $SAT_u(u, relNb) = SAT_u(fwTg, relNb) = b$ and $SAT_u(u, on) = SAT_u(fwTg, on) = a$. This indicates it is not necessary to send a FW to $fwTg$. In this case, u only checks if it is already included in $R(relNb)$. If no, u sends an RLC to $relNb$ to confirm the reception of the last RL (Algorithm 3, lines 21–23).

E. EXAMPLE

Fig. 4 shows how the proposed scheduling algorithm can be applied on the network presented earlier in Fig. 1. In this example, for the sake of simplicity and clarity, only changes made to node A's SAT and TOT, or in other words, from A's point-of-view, are shown. In this case, the SCP can be roughly divided into 4 steps as below.

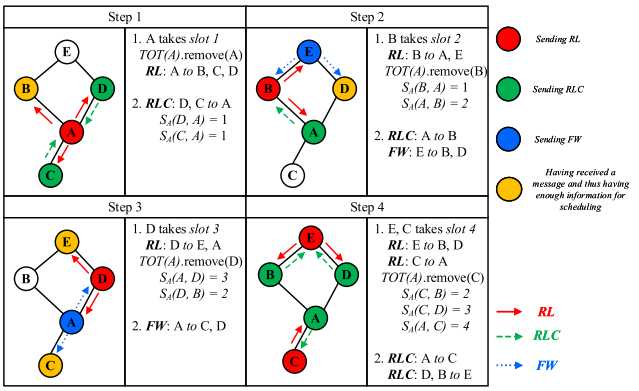

FIGURE 4. Example of applying DSTO into a wireless sensor network.

Step 1: Since node A has the highest position on its and its neighbors' TOTs, it takes slot 1, removes itself from its TOT, and broadcasts this information along with a RL. Both node D and C do not have a $fwTg$, therefore they reply to A with an RLC. A sets $SAT_A(C, A) = SAT_A(D, A) = 1$.

Step 2: However, in B's case, after receiving A's slot information, it has enough information for scheduling as A is the only node that is ranked higher than itself. B removes A from its TOT. It is now on top of the TOT and allowed to schedule. Therefore, B takes the minimum available slot, which is slot 2 and broadcasts its RL. It should be noted A's slot information is piggy-backed in B's RL because A is a one-hop neighbor of B. Therefore, node A is still able to know that its RL has successfully arrived at B. Upon the reception of B's RL, A removes B from its TOT and set $SAT_A(B, A)$, $SAT_A(A, B)$ to 1 and 2, respectively. Now, both A and E have the same $fwTg$, which is D because D is two-hop away from B and has to know B's slot information before it can start scheduling. In addition, since there is only B's slot information to be forwarded and E has less neighbors than A does, E sends a FW to D, which also serves as a confirmation to B. Node A, on the other hand, simply replies B with an RLC.

Step 3: After receiving the FW from E, node D picks slot 3 and broadcasts the information to A and E. C now becomes A's forwarded target node, thus, A has to forward D's information to C. Additionally, in Step 2, A only forwards B's information to D, thus C is still not aware of B's assigned slot. However, because B is a one-hop neighbor of A, its slot information is also piggy-backed in A's FW. As a result, C

can know about B and D's assigned slots even though they are two-hop away.

Step 4: As stated earlier, the scheduling order from A's point of view is A, B, D, C and E. However, C and E are three-hop away from each other, which allows them to schedule at the same turn without causing conflicts. Both C and E take slot 4 and broadcast their *RL*s. Next, node B and D reply to E with an *RLC* to confirm the reception of E's *RL*. Node A also receives the *RL* from C and then updates it TOT and SAT. Even though C's *RL* does not piggy-back information about B and D as they are its two-hop neighbors, A still knows that C has successfully received its *FW*. This is because C is only allowed to schedule once all of the higher-ranked nodes, which are A, B and D, have decided on their time slots. Therefore, $SAT_A(C, B)$, $SAT_A(C, D)$ and $SAT_A(A, C)$ are set to 2, 3 and 4, respectively. Lastly, A confirms to C with an *RLC*.

VI. THEORETICAL ANALYSIS

In DSTO, any incorrectness or conflicts among neighbor nodes' TOs would directly result in scheduling failure. Moreover, it must be ensured each node will always be able to receive enough slot information from all of the neighbors ranked higher than itself on its TOT. Otherwise, DSTO fails to provide a complete schedule. In Subsection VI-A, we use theoretical analysis to prove that DSTO can always complete scheduling and provide collision-free slot allocation. Furthermore, in Subsection VI-B, we analyze the complexity of DSTO in terms of message transmissions and computation.

A. DSTO's CORRECTNESS

Theorem 1: DSTO produces collision-free slot allocation for all nodes in the network. In other words, each node has a unique time slot within a two-hop distance.

Proof: The following lemmas are to prove Theorem 1.

Lemma 1: Assuming that each node has a numerical value unique inside its two-hop neighborhood, there are no conflicts between nodes' TOTs.

Proof: Let N be the number of nodes in the network and the list $\mathbb{P} = (P_1, P_2 \dots P_N)$ be the lists of priority values of all nodes which satisfies:

$$P_1 > P_2 > P_3 > \dots > P_{N-1} > P_N \tag{5}$$

It can easily be seen that all the TOs produced by individual nodes are sub-lists of $\mathbb{P}$. Thus, they retain the descending property of $\mathbb{P}$. This can prove that there would be no conflicts between nodes' TOs.

Lemma 2: Assuming that the network topology is stable for a sufficient amount of time, each node is able to collect enough information about neighbors' slot assignment to acquire a time slot at its turn in accordance with the its TOT.

Proof: In DSTO, there are two cases in which a node can receive neighbors' slot information.

- **Case 1** (receiving a *RL*). After acquiring a time slot, node u broadcasts *RL*s until all of its one-hop neighbors have successfully confirmed their awareness about its slot assignment with an *RLC* or *FW*. A list of the neighbors that have responded, or $R(u)$ is piggy backed inside the outgoing *RL*s. If a neighbor finds its ID not listed on $R(u)$, it has to respond again to u's *RL*s. Therefore, in this case, it is guaranteed all neighbor nodes of one-hop distance will be aware of each other's slot information.
- **Case 2** (receiving a *FW*). Upon the reception of a *RL*, node u checks if it has to notify one of its one-hop neighbors, denoted v, that has not acquired a time slot with a *FW*. If u sees that v has not been entirely aware of slot information of u's one-hop neighbors (or v's two-hop neighbors) who have finished scheduling, or:

$$\begin{cases} SAT_u(u, x) > 0 \\ SAT_u(v, x) = 0 \end{cases} \quad \forall x \in N_1(u) \;\&\&\; x \notin N_1(v) \tag{6}$$

node u will send *FW*s until receiving a confirmation from v. Therefore, in this case, it is also guaranteed that neighbor nodes of two-hop distance will be aware of each other's slot information. Thus, we can prove that each node is always able to collect enough neighbors' slot information.

By Lemma 1 and 2, we can prove that there are no conflicts between TOs created by individual nodes and each node will always receive enough neighbors' slot information. This ensures that each node can follow its TO to choose a time slot that has not been used by its neighbors within two hops. ■

B. DSTO's COMPLEXITY

Theorem 2: Assuming that the maximum delay of any messages is limited by a constant, DSTO's message complexity is $O(\delta)$, where δ is the maximum size of two-hop neighborhoods.

Proof: In DSTO, each node has to send $O(1)$ messages in each of the following cases: announcing its slot assignment, receiving a release message, and receiving a forward message. All of these cases result from the node's and its neighbors' slot assignments. Since there are a maximum of δ nodes in any two-hop neighborhood and each node only performs slot assignment once, the expected message complexity is $O(\delta)$. ■

Theorem 3: The expected internal computational complexity to run DSTO is $O(\delta_1)$, where δ_1 is the maximum size of one-hop neighborhoods and $0 < \delta_1 \leq \delta$.

Proof: Our algorithm is divided into two phases, which are NDP and SCP. In the NDP (Algorithm 1) and the SCP (Algorithms 2 and 3). In both phases, there are no nested loops through out both algorithms 2 and 3 and all of the loops go through only the nodes' lists of one-hop neighbors, the complexity can be expected as $O(\delta_1)$ as well. ■

Theorem 4: Let N be the number of nodes in the network. Assuming that the maximum delay of any messages is limited by a constant with c's being the maximum number of retransmission attempts to deliver a message successfully ($c \geq 0$) and d_{tx}'s being the maximum one-way transmission

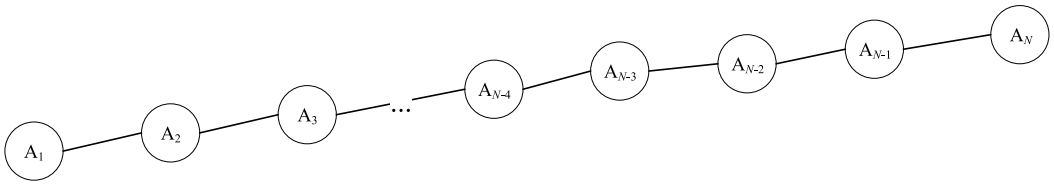

FIGURE 5. An example network for the worst-case scenario.

delay of any message. DSTO's worst-case running time:

$$T_{max} = d_{tx} \cdot \left[(5c+1) - c \cdot \lceil [] \frac{N-3}{2} \rceil \right] \tag{7}$$

Proof: As stated earlier, in DSTO, a node has to wait for all neighbors ranked higher in its two-hop neighborhood to finish their scheduling before its can acquire a time slot. Also, these neighbors also need to wait for their neighbors in the same manner. In the worst-case scenario, the network forms a *line* topology as shown in Fig. 5. In this case, nodes $A_3 \ldots A_{N-2}$ all have 4 neighbors more than those of A_1, A_2, A_{N-1} and A_N, which are 2, 3, 3, 2, respectively. Therefore, there is possibility that A_{N-2} has the highest priority in the network and that node A_1 has to wait for node A_{N-2} and several nodes between them. From Section V, we know that some of these nodes including A_{N-2} have to perform slot allocation and notify its neighbors with *RL* messages, while the others have to forward neighbors' slot information to another node with *FW* messages. Thus, we call n_1 the number of nodes that send *RL* messages and n_2 the number of nodes send *FW* messages ($0 \leq n_1, n_2 \leq N-3$).

Now, assuming amount of time needed for internal calculation is negligible (*Theorem 3*), the maximum running time can be calculated as

$$T_{max} = n_1 \cdot d_{tx} + n_2 \cdot d_{tx} \tag{8}$$

However, messages may require retransmission to be delivered successfully. As introduced in Subsection V-C, nodes wait for $4d_{tx}$ and $5d_{tx}$ before retransmitting *RL* and *FW* messages, respectively. Therefore, if c is maximum number of retransmission attempts, the running time becomes

$$T_{max} = n_1 \cdot (4c+1) \cdot d_{tx} + n_2 \cdot (5c+1) \cdot d_{tx} \tag{9}$$

Moreover, since a node's forwarding is only triggered by another neighbor's *RL* message, the number of forwarding nodes is always equal or smaller than that of releasing nodes. Additionally, there are $N-3$ hops between A_{N-2} and A_1, we can establish this as an optimization problem.

$$\textit{Maximize } f(x) = n_1 \cdot (4c+1) \cdot d_{tx} + n_2 \cdot (5c+1) \cdot d_{tx} \tag{10}$$

$$\textit{Subject to the constrains } n_2 \leq n_1 \tag{11}$$

$$n_1 + n_2 = N - 3 \tag{12}$$

By substituting $n_2 = N - 3 - n_1$ into $f(x)$, we have:

$$f(x) = (5c+1) \cdot d_{tx} - c \cdot d_{tx} \cdot n_1 \tag{13}$$

It is obvious that, for $f(x)$ to maximize, we need n_1 to be minimum. From equation (12) and (13), we can easily find that

$$n_1 \geq \frac{N-3}{2} \tag{14}$$

Because $n_1 \in \mathbb{N}$, we have

$$n_{1_{min}} = \lceil [] \frac{N-3}{2} \rceil \tag{15}$$

As a result, the maximum running time is

$$T_{max} = d_{tx} \cdot \left[(5c+1) - c \cdot \lceil [] \frac{N-3}{2} \rceil \right]. \tag{16}$$

■

VII. PERFORMANCE EVALUATION

In this section, we study the performance of DSTO in terms of running time, message overhead, slot assignment, and numbers of trials (rounds) when being applied to WSNs. To study the performance of DSTO over that of DRAND [22] and DTSS [20], we set up the simulations in OPNET Network Modeler. The number of nodes in a network ranges from 50 to 250 and all nodes are deployed randomly in an area of 300 × 300 m^2. The communication range is set at 40m. This configuration leads to two-hop neighborhood sizes' ranging from 8 to 60. This simulation setup aims to prove the effectiveness of DSTO on both small- and large-scale wireless networks with different neighborhood densities.

A. RUNNING TIME

Fig. 6(a) plots the average running time according to the neighborhood size (node density) to compare the trends of results from three algorithms. It can be seen that DSTO performs better than DRAND by 30 to 60%. This is because, in DSTO, nodes assign slots in turn according to its TOT, and as soon as they have received enough TI from their neighbors. The number of message collisions between neighbor nodes is also reduced, which leads to less execution time wasted on unnecessary retransmissions. DRAND, on the other hand, is prone to collisions due to its random nature, which also leads to significant variations of simulation results. Furthermore, Fig. 6(b) displays the variations of slot-assigning time by nodes according to network sizes. In DSTO, the duration from the first node that assigns its slot to the last in each network is remarkably lower than that of DRAND. In DRAND, nodes wait and request at a random time and especially they

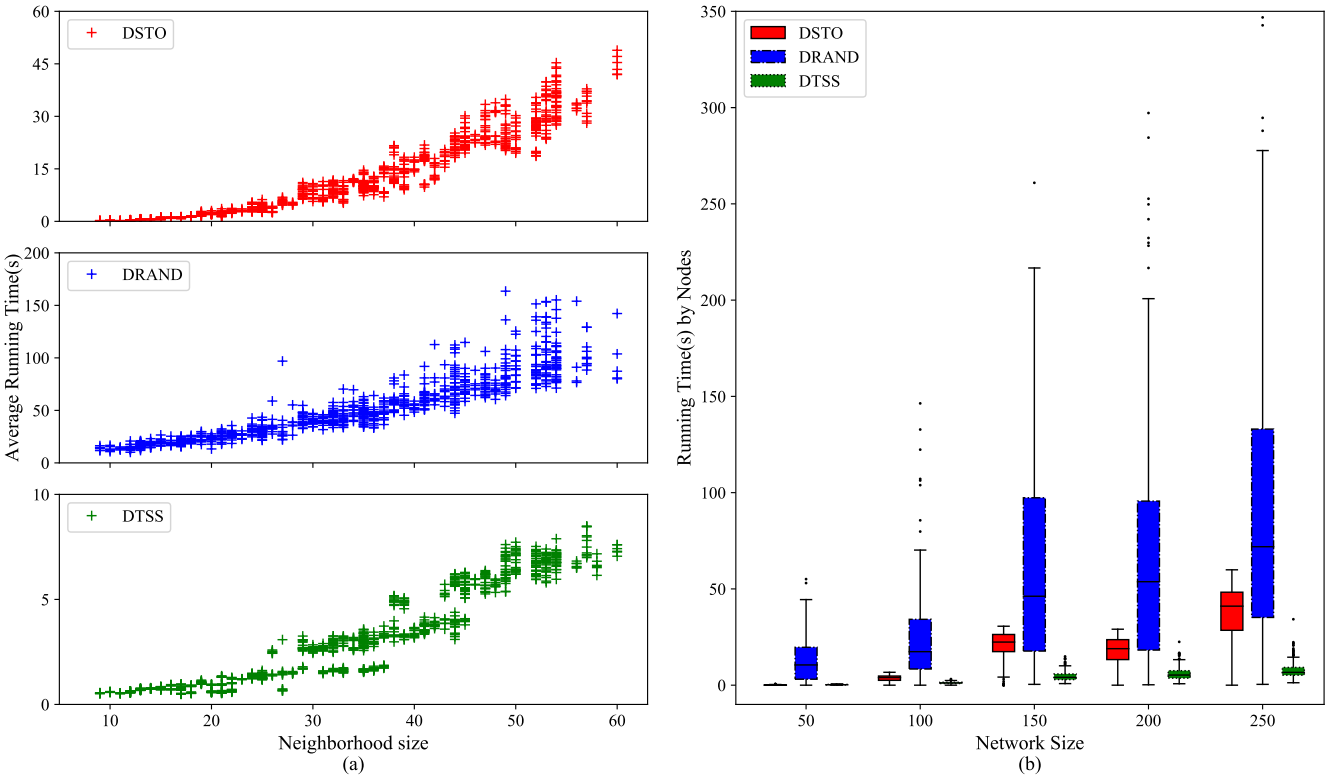

FIGURE 6. **(a) Average running time to acquire a slot and (b) Running time of individual nodes to acquire a slot.**

may fail several times before acquiring a time slot. More importantly, when a node fails, it does not only waste its own time but also time of its one-hop neighbors that has sent *grant* to it and their one-hop neighbors as well, because DRAND nodes can only send grants to only one node at any moment.

On the contrary, DSTO's performance is inferior to that of DTSS in both Fig. 6(a) and (b). Since DTSS nodes are time-synchronized, they can save a significant amount of time spent on waiting. However, it is important to note that in the case of DTSS, the actual running time needed will be higher than what is shown in the figures, as extra running time needs to be spent on time-synchronization before the scheduling phase's being able to start. Moreover, errors in time-synchronization may result in more delay or in the worst case, scheduling failure.

B. MESSAGE TRANSMISSIONS

Fig. 7(a) compares the trends of average message transmissions of three algorithms as the neighborhood size increases. It proves DSTO's effectiveness on reducing the number of generated messages. In terms of average numbers of message transmissions, DSTO outperforms DTSS by 25-30% and DRAND by 30-40% in most cases. Moreover, the output variance's size of DRAND can be higher than that of DSTO by 3-4 times in some cases. By employing a TO, DSTO allows nodes to assign time slots in an orderly fashion, which helps reduce the number of message losses due to collisions, whereas DRAND nodes have to compete with each other and thus generate more message losses. In DTSS, each node has to, at least, request twice and as a consequence, response twice to each of its one-hop neighbors. This leads to a high amount of message overhead and thus high energy consumption.

Fig. 7(b) shows the number of message transmissions by individual nodes in five networks of 50 – 250 nodes. The differences between DSTO and the other algorithms are more noticeable as the network size increases. In the network of 250 nodes, 50% of nodes in DSTO, DRAND and DTSS transmitted more than 48, 63 and 57 messages, respectively. The node that had to transmit the most in DSTO accounts for 80 messages, while those in DRAND and DTSS generated roughly 190 and 130 messages.

Fig. 8 illustrates the total number of message transmissions according to the network size. Similar to the previous result, it can be observed that there are only small differences between the results of DSTO and the others in low-density networks (50 – 100 nodes), whereas high density networks (150 – 250 nodes) have significant differences due to high possibility of collisions. DSTO can save up to 35 and 25% of total messages compared to DRAND and DTSS, respectively, as the network density increases. This proves that Topological Ordering is effective in terms of avoiding collisions between neighbors.

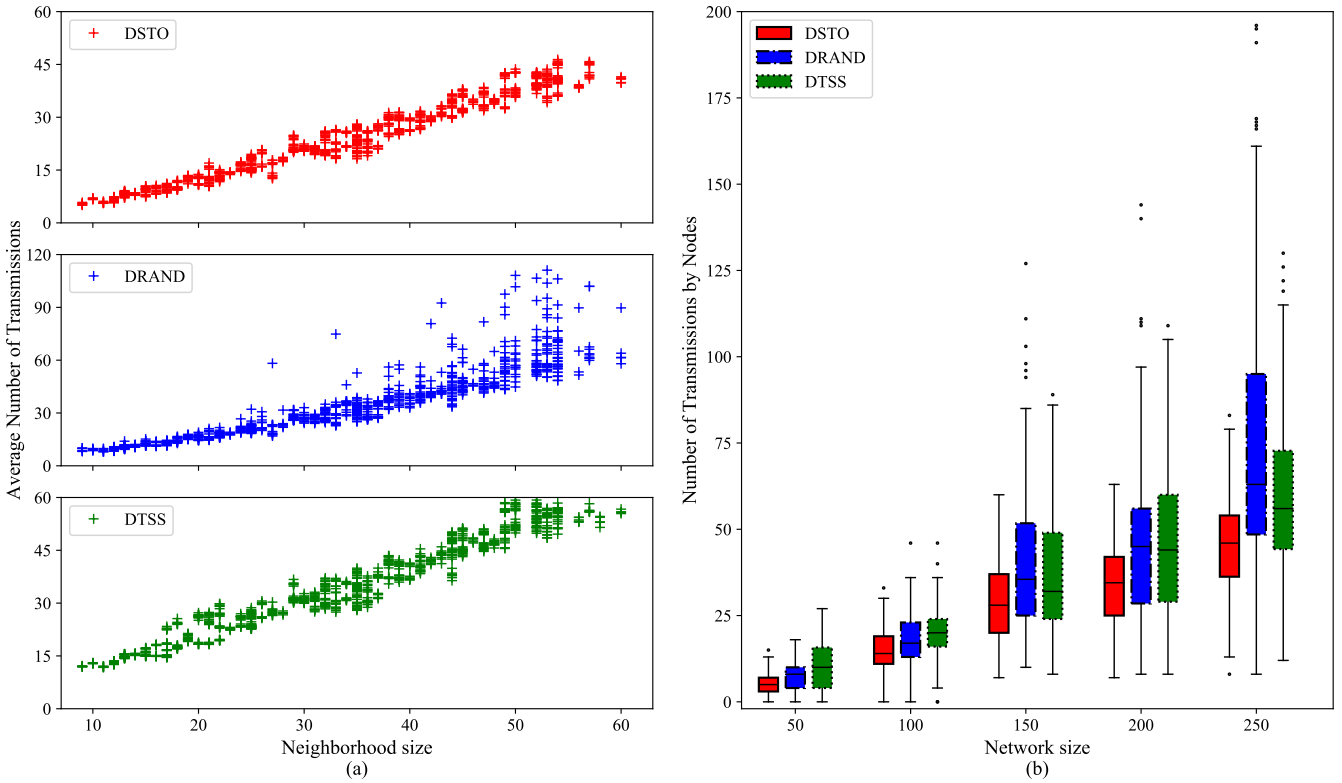

FIGURE 7. (a) Average number of message transmissions, (b) Average number of message transmissions by individual nodes.

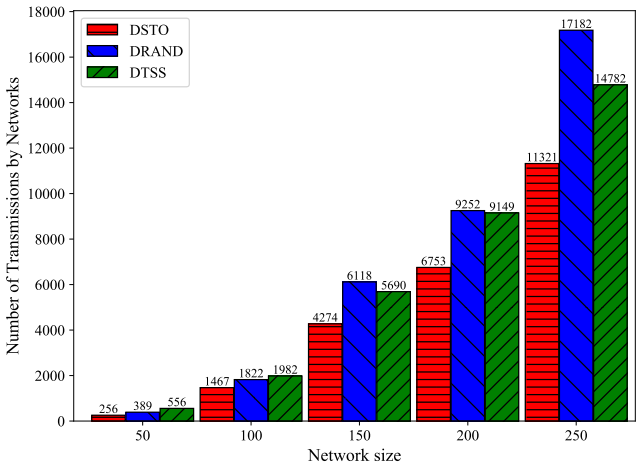

FIGURE 8. Total number of message transmissions by networks.

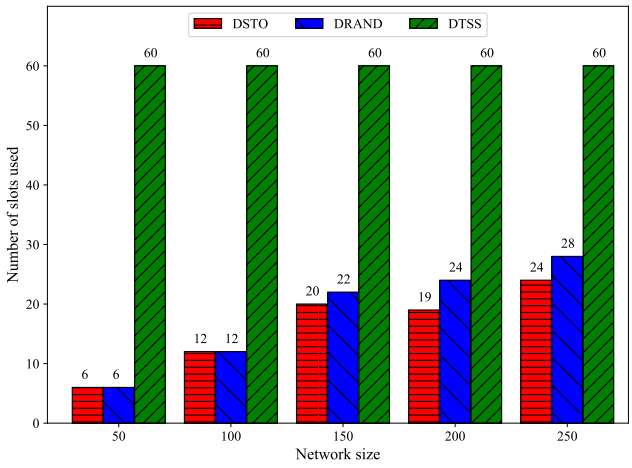

FIGURE 9. Maximum number of assigned slots.

C. SLOT ALLOCATION

Fig. 9 shows the number of used slots in each frame of three algorithms. In both DSTO and DRAND, when choosing a time slot, nodes always try to pick the minimum slot that does not cause collisions with any of its neighbors within a two-hop distance. Therefore, we can see only small differences between two algorithms. However, in the case of DTSS, the frame size must be set before the scheduling starts. In this simulation, we choose the frame of 60 slots, which is equal to the maximum neighborhood size. Because any frame shorter than this would result in an incomplete scheduling phase, in which there are nodes that cannot assign a slot. Moreover, it can be seen that DRAND appears to have used 2, 5 and 4 slots more than DSTO in 150-node, 200-node and 250-node networks, respectively. DSTO frames are 60-90% shorter than those of DTSS in all simulations.

Fig. 10 shows the slot distributions and CDFs for networks of 50, 150 and 250 nodes, respectively. DTSS has poor

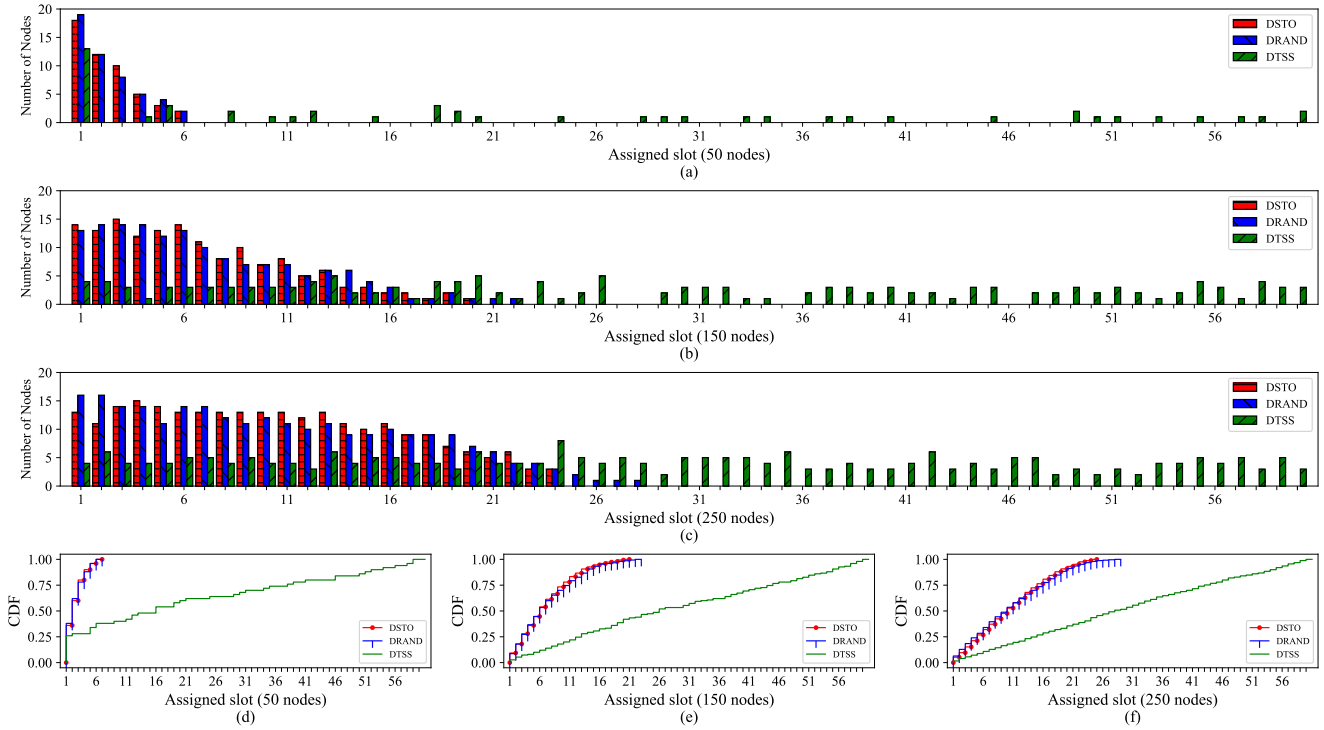

FIGURE 10. Assigned slot distributions and CDFs for networks.

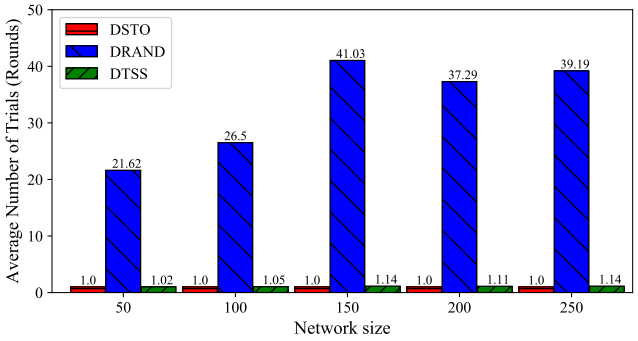

FIGURE 11. Average number of trials (rounds) taken to allocate a time slot.

channel utilization rates, especially in cases of low-density networks with a large proportion of unused slots. These slots can certainly be reassigned to raise the channel utilization rate. However, it would have to be a tradeoff as more running time and message transmissions are needed.

D. NUMBER OF TRIALS (ROUNDS)

In this subsection, we compare the average number of trials taken in order for each node to acquire a time slot. As mentioned earlier, in DRAND, at each round, each node plays a lottery and if it loses, it has to wait before being allowed to play again. Otherwise, if it wins, the node will start broadcasting requests. Thus, we can consider each round is a trial. When nodes lose the lottery or are rejected by their neighbors, the trial is considered a failure resulting in waste of resources such as time, message transmission and energy. The case is similar for DTSS. When a node randomly chooses a slot that has been blocked or taken by one of its neighbor, it has to start again from the beginning.

Fig. 11 shows that in DRAND, each node, in average, has to try and fail several times before being able to acquire a slot. In DTSS, the numbers of trials are significantly smaller due to the large frame size. On the other hand, in DSTO, since the scheduling order has been decided beforehand, each node has to try only once when they have received enough information on neighbors' slot assignments.

VIII. CONCLUSION

In this paper, we introduce the concept of Topological Ordering and use it to produce a scheduling order for WSNs. We also propose a distributed TDMA scheduling algorithm using Topological Ordering called DSTO. Our algorithm is designed to be scalable in a fully distributed manner and to reduce running time and message overhead, which is the key to energy preservation. From theoretical analysis, the message and internal computation complexities are $O(\delta)$ and $O(\delta_1)$ and, where δ and δ_1 and is the maximum two- and one-hop neighborhood sizes, respectively. This helps prove the algorithm's scalability. Moreover, we thoroughly evaluate the performance of DSTO in various aspects and compare it to those of two scheduling algorithms called, DRAND and DTSS. The evaluation results confirm the validity and proves DSTO's effectiveness in terms of scheduling time and the

number of generated messages regardless of the network size and density.

ACKNOWLEDGMENT

A preliminary version of this article was presented at the 28th International Conference on Computer Communications and Networks (ICCCN), Spain, July 2019 [37].

REFERENCES

[1] L. Lombardo, S. Corbellini, M. Parvis, A. Elsayed, E. Angelini, and S. Grassini, "Wireless sensor network for distributed environmental monitoring," *IEEE Trans. Instrum. Meas.*, vol. 67, no. 5, pp. 1214–1222, May 2018.

[2] G. Xu, W. Shen, and X. Wang, "Applications of wireless sensor networks in marine environment monitoring: A survey," *Sensors*, vol. 14, no. 9, pp. 16932–16954, Sep. 2014.

[3] A. J. Watt, M. R. Phillips, C. E.-A. Campbell, I. Wells, and S. Hole, "Wireless sensor networks for monitoring underwater sediment transport," *Sci. Total Environ.*, vol. 667, pp. 160–165, Jun. 2019.

[4] W. Yi, K. Lo, T. Mak, K. Leung, Y. Leung, and M. Meng, "A survey of wireless sensor network based air pollution monitoring systems," *Sensors*, vol. 15, no. 12, pp. 31392–31427, Dec. 2015.

[5] D. Chen, Z. Liu, L. Wang, M. Dou, J. Chen, and H. Li, "Natural disaster monitoring with wireless sensor networks: A case study of data-intensive applications upon low-cost scalable systems," *Mobile Netw. Appl.*, vol. 18, no. 5, pp. 651–663, Oct. 2013.

[6] B. Rashid and M. H. Rehmani, "Applications of wireless sensor networks for urban areas: A survey," *J. Netw. Comput. Appl.*, vol. 60, pp. 192–219, Jan. 2016.

[7] M. Erdelj, M. Król, and E. Natalizio, "Wireless sensor networks and multi-UAV systems for natural disaster management," *Comput. Netw.*, vol. 124, pp. 72–86, Sep. 2017.

[8] X. Hu, B. Wang, and H. Ji, "A wireless sensor network-based structural health monitoring system for highway bridges," *Comput.-Aided Civil Infrastruct. Eng.*, vol. 28, no. 3, pp. 193–209, Mar. 2013.

[9] M. Z. A. Bhuiyan, G. Wang, J. Wu, J. Cao, X. Liu, and T. Wang, "Dependable structural health monitoring using wireless sensor networks," *IEEE Trans. Dependable Secure Comput.*, vol. 14, no. 4, pp. 363–376, Jul. 2017.

[10] M. Z. A. Bhuiyan, G. Wang, J. Cao, and J. Wu, "Deploying wireless sensor networks with fault-tolerance for structural health monitoring," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 382–395, Feb. 2015.

[11] R. Vera-Amaro, M. E. Rivero-ángeles, and A. Luviano-Juárez, "Data collection schemes for animal monitoring using WSNs-assisted by UAVs: WSNS-oriented or UAV-oriented," *Sensors*, vol. 20, no. 1, pp. 262–293, 2020.

[12] A. Naureen, N. Zhang, S. Furber, and Q. Shi, "A GPS-less localization and mobility modelling (LMM) system for wildlife tracking," *IEEE Access*, vol. 8, pp. 102709–102732, 2020.

[13] J. G. A. Barbedo, L. V. Koenigkan, P. M. Santos, and A. R. B. Ribeiro, "Counting cattle in UAV images-dealing with clustered animals and animal/background contrast changes," *Sensors*, vol. 20, no. 7, pp. 2126–2139, 2020.

[14] D. Kandris, C. Nakas, D. Vomvas, and G. Koulouras, "Applications of wireless sensor networks: An up-to-date survey," *Appl. Syst. Innov.*, vol. 3, no. 1, pp. 14–37, 2020.

[15] D. Thakur, Y. Kumar, A. Kumar, and P. K. Singh, *Applicability of Wireless Sensor Networks in Precision Agriculture: A Review*, vol. 107, no. 1. New York, NY, USA: Springer, 2019, doi: 10.1007/s11277-019-06285-2.

[16] J. A. Stankovic, "Research directions for the Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 3–9, Feb. 2014.

[17] L. Da Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[18] I. Rhee, A. Warrier, M. Aia, J. Min, and M. L. Sichitiu, "Z-MAC: A hybrid MAC for wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 511–524, Jun. 2008.

[19] A. Woo and D. E. Culler, "A transmission control scheme for media access in sensor networks," in *Proc. MobiCom*, 2001, pp. 221–235.

[20] A. Bhatia and R. C. Hansdah, "A distributed TDMA slot scheduling algorithm for spatially correlated contention in WSNs," *Mobile Inf. Syst.*, vol. 2015, pp. 1–16, Mar. 2015.

[21] S. Ramanathan, "A unified framework and algorithm for (T/F/C) DMA channel assignment in wireless networks," in *Proc. INFOCOM*, 1997, vol. 2, no. 2, pp. 900–907.

[22] I. Rhee, A. Warrier, J. Min, and L. Xu, "DRAND: Distributed randomized TDMA scheduling for wireless ad hoc networks," *IEEE Trans. Mobile Comput.*, vol. 8, no. 10, pp. 1384–1396, Oct. 2009.

[23] Y. Li, X. Zhang, J. Zeng, Y. Wan, and F. Ma, "A distributed TDMA scheduling algorithm based on energy-topology factor in Internet of Things," *IEEE Access*, vol. 5, pp. 10757–10768, 2017.

[24] Y. Li, X. Zhang, T. Qiu, J. Zeng, and P. Hu, "A distributed TDMA scheduling algorithm based on exponential backoff rule and energy-topology factor in Internet of Things," *IEEE Access*, vol. 5, pp. 20866–20879, 2017.

[25] K. Sato and S. Sakata, "A power-efficient distributed TDMA scheduling algorithm with distance-measurement for wireless sensor networks," *Wireless Pers. Commun.*, vol. 75, no. 2, pp. 1511–1528, Mar. 2014.

[26] G. Chakraborty, "Genetic algorithm to solve optimum TDMA transmission schedule in broadcast packet radio networks," *IEEE Trans. Commun.*, vol. 52, no. 5, pp. 765–777, May 2004.

[27] C. Y. Ngo and V. O. K. Li, "Centralized broadcast scheduling in packet radio networks via genetic-fix algorithms," *IEEE Trans. Commun.*, vol. 51, no. 9, pp. 1439–1441, Sep. 2003.

[28] S. Ramanathan and E. L. Lloyd, "Scheduling algorithms for multi-hop radio networks," *IEEE/ACM Trans. Netw.*, vol. 1, no. 2, pp. 166–177, 1993.

[29] W. Osamy, A. A. El-Sawy, and A. M. Khedr, "Effective TDMA scheduling for tree-based data collection using genetic algorithm in wireless sensor networks," *Peer-Peer Netw. Appl.*, vol. 13, no. 3, pp. 796–815, May 2020.

[30] L. Bao and J. J. Garcia-Luna-Aceves, "A new approach to channel access scheduling for ad hoc networks," in *Proc. MobiCom*, 2001, pp. 210–221.

[31] R. Rozovsky and P. R. Kumar, "SEEDEX: A MAC protocol for ad hoc networks," in *Proc. 2nd ACM Int. Symp. Mobile Ad Hoc Netw. Comput. (MobiHoc)*, vol. 7, 2001, pp. 67–75.

[32] C. Zhu and M. S. Corson, "A five-phase reservation protocol (FPRP) for mobile ad hoc networks," *Wireless Netw.*, vol. 7, no. 4, pp. 371–384, 2001.

[33] B. Dezfouli, M. Radi, K. Whitehouse, S. A. Razak, and T. Hwee-Pink, "DICSA: Distributed and concurrent link scheduling algorithm for data gathering in wireless sensor networks," *Ad Hoc Netw.*, vol. 25, pp. 54–71, Feb. 2015.

[34] H. Lakhlef, M. Raynal, and F. Taïani, "Vertex coloring with communication constraints in synchronous broadcast networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1672–1686, Jul. 2019.

[35] M. S. Batta, S. Harous, L. Louail, and Z. Aliouat, "A distributed TDMA scheduling algorithm for latency minimization in Internet of Things," in *Proc. IEEE Int. Conf. Electro Inf. Technol. (EIT)*, Brooklings, SD, USA, May 2019, pp. 108–113.

[36] M. S. Batta, Z. Aliouat, and S. Harous, "A distributed weight-based TDMA scheduling algorithm for latency improvement in IoT," in *Proc. IEEE 10th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, New York, NY, USA, Oct. 2019, pp. 0768–0774.

[37] T.-T. Nguyen, L.-A. Phan, T. Kim, T. Kim, J. Lee, and J. Ham, "Distributed TDMA scheduling using topological ordering in wireless sensor networks," in *Proc. 28th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2019, pp. 1–2.

THANH-TUNG NGUYEN received the B.S. degree in electronics and telecommunications from the University of Engineering and Technology, Vietnam National University, in 2017. He is currently pursuing the M.S. degree in communication and information engineering with Chungbuk National University, South Korea, under the Korean Government Scholarship Programme (KGSP). His research interests include the Internet of Things, cloud/edge computing, open-source software, and ad-hoc networks.

TAEJOON KIM (Member, IEEE) received the B.S. degree in electronics engineering from Yonsei University, Seoul, South Korea, in 2003, and the Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2011. From 2003 to 2005, he was a Researcher with LG Electronics, Seoul. From 2011 to 2013, he was a Senior Researcher with ETRI, Daejeon. He is currently an Associate Professor with the School of Information and Communication Engineering, Chungbuk National University, South Korea. His research interests include the analysis and optimization of wireless networks and communication theory.

TAEHONG KIM (Member, IEEE) received the B.S. degree in computer science from Ajou University, South Korea, in 2005, and the M.S. degree in information and communication engineering and the Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), in 2007 and 2012, respectively. He worked as a Research Staff Member with the Samsung Advanced Institute of Technology (SAIT) and the Samsung DMC Research and Development Center, from 2012 to 2014. He also worked as a Senior Researcher with the Electronics and Telecommunications Research Institute (ETRI), South Korea, from 2014 to 2016. Since 2016, he has been an Associate Professor with the School of Information and Communication Engineering, Chungbuk National University, South Korea. His research interests include edge computing, SDN/NFV, the Internet of Things, and wireless sensor networks. He is also an Associate Editor of IEEE Access.