Hindawi Publishing Corporation
Mobile Information Systems
Volume 2015, Article ID 234143, 16 pages
http://dx.doi.org/10.1155/2015/234143

Research Article

A Distributed TDMA Slot Scheduling Algorithm for Spatially Correlated Contention in WSNs

Ashutosh Bhatia and R. C. Hansdah

Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India

Correspondence should be addressed to Ashutosh Bhatia; ashutosh.b@csa.iisc.ernet.in

Received 3 July 2013; Accepted 25 February 2014

Academic Editor: David Taniar

In WSNs the communication traffic is often time and space correlated, where multiple nodes in a proximity start transmitting simultaneously. Such a situation is known as *spatially correlated contention*. The random access method to resolve such contention suffers from high collision rate, whereas the traditional distributed TDMA scheduling techniques primarily try to improve the network capacity by reducing the schedule length. Usually, the situation of *spatially correlated contention* persists only for a short duration, and therefore generating an optimal or suboptimal schedule is not very useful. Additionally, if an algorithm takes very long time to schedule, it will not only introduce additional delay in the data transfer but also consume more energy. In this paper, we present a distributed TDMA slot scheduling (DTSS) algorithm, which considerably reduces the time required to perform scheduling, while restricting the schedule length to the maximum degree of interference graph. The DTSS algorithm supports unicast, multicast, and broadcast scheduling, simultaneously without any modification in the protocol. We have analyzed the protocol for average case performance and also simulated it using Castalia simulator to evaluate its runtime performance. Both analytical and simulation results show that our protocol is able to considerably reduce the time required for scheduling.

1. Introduction

A wireless sensor network (WSN) is a collection of sensor nodes distributed over a geographical region to monitor events of interest in the region. To effectively exchange data among multiple sensor nodes, WSNs employ the medium access control (MAC) protocol to coordinate the transmission over the shared wireless radio channel. Many times in WSNs, communication traffic is space and time correlated; that is, all the nodes in the same proximity transmit at the same time. Such a situation is known as *spatially correlated contention*. There exist many applications and protocols in WSNs, where the situations of *spatially correlated contention* can occur. Some of them are as follows.

(i) *Event Detection*: whenever an event occurs, all the nodes that sense the event will start transmitting the details of the event to the base station. Typical examples of such situations are the detection of earthquake and wildfire in WSNs for disaster recovery, fall-and-posture detection in healthcare WSNs [1], and intrusion detection [2] in WSNs for military applications, in which sensor nodes only have data to send when a specific event occurs. As multiple nodes that detect the event are quite possibly in close proximity of each other, they would share the same transmission medium. Eventually, if all the nodes report the event at the same time, the situation would lead to *spatially correlated contention*.

(ii) *Multicast Communication*: in WSNs, the applications should be configured and updated in the sensor nodes multiple times during the lifetime of the network. An update by transmitting the content to each individual sensor node separately would be very inefficient and would consume a lot of resources such as bandwidth and energy. In this situation, multicasting provides an efficient configuration and update of applications running over sensor nodes by reducing the number of transmitted packets. Another example of multicast communication in WSN is on-demand data collection, where the base station (sink node) sends a data query to a prespecified group of nodes asking

them to send their sensory data. The WSN is usually multihop in nature, and therefore direct transmission of multicast messages from the sink is not possible. To achieve this, the sensor nodes also work as routers and forward the received multicast packet to their one-hop neighbors. This simultaneous forwarding of the same packet in a proximity by multiple routers leads to *spatially correlated contention*. A detailed discussion on multicast in WSN can be found in [3].

(iii) *Routing Protocols*: the on-demand routing protocols, for example, Ad Hoc On-Demand Distance Vector Routing (AODV) [4], try to find the appropriate path from source to destination only when the data transfer is required. This process is called route discovery and it is typically achieved by broadcasting a route request message in the network and consequently leads to the collision of request message due to its simultaneous forwarding by the neighboring nodes.

(iv) *Clock Synchronization*: clock synchronization protocols, for example [5], typically use message passing mechanism to share their local time information with other neighboring nodes. Since, initially, there is no coordination between the nodes, they may transmit the protocol message simultaneously with high probability, and therefore the transmitted messages might collide. This can considerably delay the process of synchronization.

(v) *Tree Based Convergecasting*: convergecast, that is, gathering of information towards a central node, is important communication paradigms across all application domains in WSN. This is mainly accomplished by constructing an efficient tree in terms of delay, energy, and bandwidth. The algorithm for construction of such a tree typically involves simultaneous transmission of protocol messages by the sensor nodes and hence causes *specially correlated contention*.

Thus the above discussion suggests that the MAC protocols for WSNs should effectively handle the correlated contention. MAC protocols for WSNs can be mainly classified into two major categories, namely, random access based and schedule access based. Random access methods do not use any topology or clock information and resolve contention among neighboring nodes for every data transmission. Thus, it is highly robust to any change in the network. But its performance under high contention suffers because of high overhead in resolving contention and collisions [6]. Contention causes message collisions, which are very likely to occur when traffic is *spatially correlated*. This, in turn, degrades the data transmission reliability and wastes the energy of sensor nodes.

A MAC protocol is contention-free if it does not allow any collisions. Assuming that the clocks of sensor nodes are synchronized, data transmissions by the nodes are scheduled in such a way that no two interfering nodes transmit at the same time. Early works [7–9] on scheduling are centralized in nature and normally need complete topology information, and therefore, they are not scalable. To overcome the difficulty of obtaining global topology information in large size networks, many distributed slot assignment schemes [10–14] have been proposed. The primary objective of traditional distributed TDMA scheduling techniques is to improve the network capacity by reducing the schedule length. It is effective for the kind of applications where a fixed schedule can be used for a sufficiently longer time. All the scenarios for *correlated contention* discussed previously occur in form of sessions and the nodes require a time slot to transmit a sequence of data, only during these sessions. Moreover, the same schedule cannot be reused for multiple future sessions, because at that time the network topology might have changed, due to dynamic channel conditions and occasional sleeping of sensor nodes to conserve their energy. For example, in [15], a different set of nodes are selected as routers (to equally distribute the consumption of energy among sensor nodes) every time the algorithm for construction of data collection tree is executed, and therefore this changes the network topology. This suggests that the scheduling has to be performed for every instance of *correlated contention*, and therefore the effective benefit of reducing schedule length vanishes. If an algorithm takes too long to perform scheduling, as compared to the duration of *correlated contention*, it will not only degrade the QoS (e.g., delay in detection of event at the base station) but will also lead to poor bandwidth utilization and higher energy consumption. The preceding discussion emphasizes that in order to effectively handle the *correlated contention*, the TDMA scheduling algorithms should take very less time to perform scheduling.

In this paper, we propose a distributed TDMA slot scheduling (DTSS) algorithm for WSNs. The primary objective of DTSS algorithm is to reduce the time required to perform scheduling while restricting the schedule length to maximum degree of interference graph. The proposed algorithm is unified in the sense that the same algorithm can be used to schedule slots for different modes of communication, namely, unicast, multicast, and broadcast. In addition, the DTSS algorithm also supports heterogeneous mode of communication, where simultaneously a few nodes can take a slot for unicast, while other nodes can take it for multicast or broadcast purpose. In DTSS algorithm, a node is required to know only the IDs of its intended receivers, instead of all its two-hop neighbors. Also, in DTSS algorithm, the nodes in a neighborhood can take different slots simultaneously, if the resultant schedule is feasible. This is unlike the class of greedy algorithms where ordering between the nodes puts a constraint on the distributed algorithm to run sequentially and restricts the parallel implementation of the algorithm. The DTSS algorithm does not make use of any ordering among the nodes.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 gives the assumptions we make in the design of our algorithm, introduces some definitions, and explains the basic idea of our algorithm. In Section 4, we present a detailed description of the DTSS algorithm. Section 5 gives the proof of correctness of the DTSS

algorithm. Section 6 presents the average case complexity analysis of the DTSS algorithm. Section 7 discusses the simulation results and performance comparison of DTSS algorithm with existing work. Section 8 concludes the paper with suggestions for future work.

2. Related Work

The broadcast scheduling problem to find optimal solution is NP-complete [16]. A different, but related, problem to TDMA node slot assignment is the problem of TDMA edge slot assignment, in which radio links (or edges) are assigned time slots, instead of nodes. Finding the minimum number of time slots for a conflict-free edge slot assignment is also an NP-complete problem [17]. In [18], another specific scheduling problem for wireless sensor network converge-cast transmission is considered in which the scheduling problem is to find a minimum length frame during which all nodes can send their packets to access point (AP), and it is shown to be NP-complete. Previous work [7–9, 19] on scheduling algorithms primarily focuses on decreasing the length of schedules. They are centralized in nature and normally need complete topology information and are, therefore, not scalable.

Cluster based TDMA protocols in [20, 21] prove to be having good scalability. The common feature of these protocols is to partition the network into some clusters, in which cluster heads are responsible for scheduling their members. However, cluster based TDMA protocols introduce intercluster transmission interference because clusters created by distributed clustering algorithms are often overlapped and several cluster heads may cover the same nodes.

Moscibroda and Wattenhofer [11] have proposed a distributed graph coloring scheme with a time complexity of $O(\rho \log n)$, where ρ is the maximum node degree and n is the number of nodes, in the network. The scheme performs distance-1 coloring such that adjacent nodes have different colors. Note that this does not prevent nodes within two hops of each other from being assigned the same color potentially causing hidden terminal collisions between such nodes. The NAMA protocol in [10] has proposed a distributed scheduling scheme based on hash function to determine the priority among contending neighbors. A major limitation of this hashing based technique is that even though a node gets a higher priority in one neighborhood, it may still have a lower priority in other neighborhoods. Thus the maximum slot number could be of $O(n)$. Secondly, since each node calculates the priority of all of its two-hop neighbors for every slot, it leads to $O(n^2)$ computational complexity, and hence, the scheme is not scalable for large network with resource constraint nodes. SEEDEX [22] uses a similar hashing scheme as NAMA based on a random seed exchanged in a two-hop neighborhood. In SEEDEX, at the beginning of each slot, if a node has a packet ready for transmission, it draws a "lottery" with probability p. If it wins, it becomes eligible to transmit. A node knows the seeds of the random number generators of its two-hop neighbors, and hence it also knows the number of nodes (including itself) n, within two hops which are also eligible to transmit. It then transmits with probability $1/n$. This technique is also called topology independent scheduling. In this case, collisions may still occur if two nodes select the same slot and decide to transmit.

Another distributed TDMA scheduling scheme, called DRAND [12], proposes a distributed randomized time slot scheduling algorithm based on centralized scheduling scheme RAND [9]. DRAND is also used within a MAC protocol, called Zebra-MAC [23], to improve performance in sensor networks by combining the strength of scheduled access during high loads and random access during low loads. The runtime complexity of DRAND is $O(\delta)$, where δ is the maximum size of a two-hop neighborhood in a wireless network. The simulation results presented by the author show that the runtime actually becomes $O(\delta^2)$ due to unbounded message delays. FPRP [14], Five-Phase Reservation Protocol, is a distributed heuristic TDMA slot assignment algorithm. FPRP is designed for dynamic slot assignment, in which the real time is divided into a series of pairs of reservation and data transmission phases. For each time slot of the data transmission phase, FPRP runs a five-phase protocol for a number of times (cycles) to pick a winner of each slot. In another distributed slot scheduling algorithm, DD-TDMA [13], a node i decides slot j as its own slot if all the nodes with ID less than the ID of node i have already decided their slot, where j is the minimum available slot. The scheduled node i broadcasts its slot assignment to one-hop neighbors. Then the one-hop neighbors of node i broadcast this information to update two-hop neighbors. This process is repeated in every frame until all nodes are scheduled.

The protocol in [24] proposes a contention-free MAC for correlated contention, which does not assume global time reference. The protocol is based on local frame approach where each node divides time into equal sized frames. Each frame is further divided into equal sized time slots; a time slot corresponds to the time duration of sending one message. The basic idea is that each node selects a slot in its own frame such that selected slots of any two-hop neighbor nodes must not overlap. The protocol assumes that a node can detect a collision if two or more nodes (including itself) within its transmission radius attempt to transmit, which has its own practical limitations with wireless transceivers. These scheduling algorithms [12–14] commonly have the following issues.

(i) All algorithms use greedy approach for graph colouring which is inherently sequential in nature and put a constraint on distributed algorithm to run sequentially. This restricts the parallel implementation of the algorithm. Because of large runtime of these protocols, they are more suitable for wireless networks where interference relationship or network topology does not change for a long period of time.

(ii) They perform two-hop neighbor discovery, which adds considerable additional cost to runtime to perform scheduling. Additionally, the two-hop neighbors are calculated based on transmission range instead of interference range, which is normally higher than the transmission range.

(iii) They perform either broadcast (node) scheduling or unicast (link) scheduling but not both and also do not consider multicast scheduling separately.

Finally, a classification of different scheduling algorithms based on problem setting, problem goal, type of inputs, and solution techniques can be found in [25].

3. Our Approach to TDMA Scheduling in WSNs

In many applications such as weather monitoring, intrusion detection, sensor nodes are usually static. In this work also, we assume them to be static. Also, it is assumed that, for any task in an application, every node knows its receivers. Before a task begins its execution, the DTSS algorithm is executed to generate a TDMA schedule. After the task is finished, the TDMA schedule is discarded. We assume that each node in the WSN has a unique identifier. All the nodes in a WSN have some processing capability along with a radio to enable communication among them. Each node uses the same radio frequency. The communication capability is bidirectional and symmetric. The mode of communication between any two neighboring nodes is half-duplex; that is, only one node at a time can transmit. The transmission is omnidirectional; that is, each message sent by a node i is inherently received at all the nodes determined by its transmission range.

Timeline is divided into fixed size frames and each frame is further divided into fixed number of time slots, called schedule length. The nodes are assumed to be synchronized with respect to slot 0 and are aware of the slot size and the schedule length. The time of slot 0 is defined by the node which starts the scheduling process. To better understand the proposed algorithm, we introduce the following definitions.

Definition 1. The interference set N_i of a node i is defined as the set of nodes which are within the interference range of node i. That is, we say that a node $k \in N_i$ if it cannot successfully receive any message transmitted by any other node, at the same time when node i is also transmitting. Moreover, if only node i has transmitted in a slot, then a node in N_i may or may not receive the message successfully.

Note that the interference set N_i is different from the set of one-hop neighbors which depends upon the transmission range of node i. Usually, the interference range is higher than the transmission range.

Definition 2. The receiver set R_i of a node i is defined as the set of intended receivers of node i.

The size of the set R_i, $|R_i|$ depends upon the type of communication, namely, unicast, multicast, or broadcast transmission. Note that $R_i \subseteq N_i$. The DTSS algorithm assumes that only the subset R_i is known to the node i instead of all its two-hop neighbors.

Definition 3. The sender set S_i of a node i is defined as the set of nodes j such that $i \in R_j$.

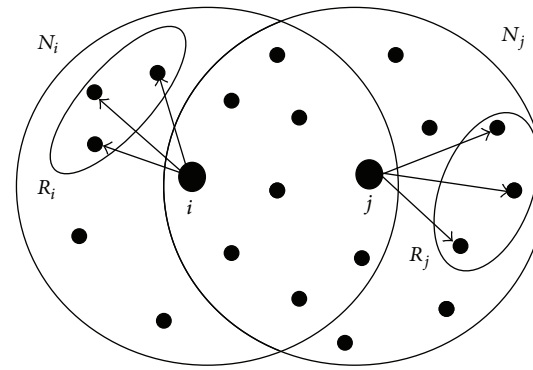


Figure 1: Example of nodes i and j that do not conflict even if they are in the interference range of each other.

A node need not know the set S_i before the start of the algorithm. It can be populated when the node receives protocol messages with destination ID as its own ID.

Definition 4. The interference graph $G = (V, E)$ of a WSN is defined as follows. V is the set of nodes in the WSN, and E is the set of edges, where edge $e = (i, j)$ exists if and only if $N_i \cap R_j \neq \phi \vee N_j \cap R_i \neq \phi$. The number of edges with which a node is connected to the other nodes is called the degree of the node.

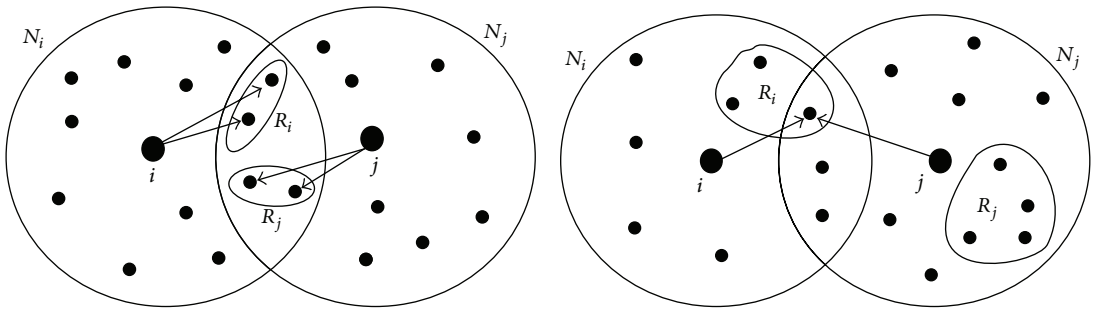
Note that $i \in R_j$ or $j \in R_i$ is also possible. We say that node i and node j conflict and are adjacent to each other, if there exists an edge between them. An edge $e = (i, j)$ exists if and only if node i and node j cannot take the same slot. Two nodes cannot take the same slot, if the transmission of one node interferes at one of the receivers of the other node. The conflict between nodes depends not only upon their respective positions and transmission power but also on the type of communication, namely, unicast, multicast, or broadcast. Two nodes within the interference range of each other ($i \in N_j \vee j \in N_i$) can even take the same slot, if their transmissions do not interfere at each other's receivers (Figure 1). Therefore, our definition of interference graph is free from well known exposed-node problem. This fact is usually ignored by most of the existing algorithms. On the other hand, two nodes which are not in the interference range of each other ($i \notin N_j \wedge j \notin N_i$) cannot transmit simultaneously, if their transmissions interfere at each other's receivers. In this manner, our definition of interference graph is also free from the hidden-node problem.

A sensor node requires a slot to transmit data packets such that data can be received successfully at all of its receivers without any interference.

The following two types of conflict relations can exist, between a pair of nodes.

Strong-Conflict Relation. Two nodes i and j have strong-conflict relation if $N_i \cap R_j \neq \phi \wedge N_j \cap R_i \neq \phi$.

Weak-Conflict Relation. Two nodes i and j have weak-conflict relation if either $N_i \cap R_j \neq \phi$ or $N_j \cap R_i \neq \phi$, but not both.



(a) Strong-conflict relation (b) Weak-conflict relation

Figure 2: Conflict relationship between nodes.

Figures 2(a) and 2(b) depict the situation when node i and node j have strong-conflict and weak-conflict relations, respectively. In case of weak-conflict relation, if $N_j \cap R_i \neq \phi$ but $N_i \cap R_j = \phi$, we say that node j is stronger than node i and denote it by $j \rightarrow i$.

Definition 5. The interference degree Δ of a WSN is defined as the maximum of all degrees of nodes in the interference graph G of the WSN.

The DTSS algorithm runs in $O(\Delta)$ time. In case of broadcast transmission; Δ and δ are roughly the same, where δ is the size of two-hop neighborhood set. But for unicast or multicast transmission $\Delta \leq \delta$.

The TDMA slot scheduling problem can be formally defined as the problem of assignment of a time slot to each node, such that if any two nodes are in conflict (strong or weak), they do not take the same time slot. Such an assignment is called a feasible TDMA schedule. A feasible TDMA schedule which takes minimum number of slots is called an optimal TDMA schedule. Our goal in this paper is to develop an algorithm which can find a feasible but not necessarily optimal TDMA schedule and to minimize the time required to perform scheduling.

The basic idea of the DTSS algorithm is as follows. For each slot s in a frame, each node i checks whether it can take the slot s, by sending a request message to the first receiver in R_i with slot probability $P(s)$, which depends upon the remaining number of free slots in the frame (slots which are not taken by others) known at node i at the current time. If node i receives response message from the first receiver, then it blocks the slot and tries to get the responses from the remaining receivers in R_i using the same slot in subsequent frames. After receiving the response from all the receivers, it assigns the time slot s to itself; otherwise, it unblocks the slot, as soon as the response from one of the receivers is not received, and repeats the above process all over again. In case of unicast communication, node i can directly assign the slot to itself as soon as it receives the response message from its receiver j instead of blocking the slot. This is because receiving a response message from node j tells that no other node k adjacent to i in G has also blocked the same slot; otherwise, the REQ message transmissions of nodes i and k would have collided at node j. An adjacent node of a node i in a graph is a node that is connected to i by an edge.

Once a slot is assigned to a node i, it continuously transmits at the same slot in subsequent frames. This would ensure that a conflicting node j in G cannot assign the same slot to itself, because of the collision between the transmission of node i and node j at one of the receivers in R_i. Furthermore, the nodes in R_i also propagate this information to next hop through their own transmissions. Note that the receivers of messages transmitted by the nodes in R_i are adjacent to i in G.

When a node j hears, from one of the receivers of node i, that slot s is blocked by node i, it leaves the slot temporarily and avoids further collisions to increase the chance of getting the slot by node i. Similarly, when node j hears, from one of the receivers of node i, that slot s is assigned to node i, it leaves the slot permanently and increases its slot probability for other free slots.

4. The DTSS Algorithm

In this section, we describe the proposed DTSS algorithm for TDMA slot scheduling problem as defined in Section 2. The number of slots, $\mathcal{N}$, in a frame is taken to be at least Δ. The slots are numbered from 1 to $\mathcal{N}$. Table 1 summarizes the set of data structures and variables maintained by a node i, to implement the algorithm. The DTSS algorithm uses two protocol messages, namely, request (REQ) and response (RES), for signaling purpose. The RES message is sent by a node, whenever its ID is the same as the destination ID in the received REQ message. The REQ/RES messages contain four fields, namely, source ID, destination ID, $L2$, and state. The value of field $L2$ in both REQ and RES messages is the copy of corresponding local variable. The value of field state in REQ message is the same as the value of the local variable *nr* while its value in RES message contains the value of field state as received in the corresponding REQ message. The field $L2$ in REQ/RES message is used to inform a node j about the slots which are already taken by other nodes conflicting with node j, whereas the field state helps the nodes to know the status of the node from where the REQ message has been transmitted. The higher level description of the DTSS algorithm is shown in the pseudocode given in Algorithm 1. The pseudocode

```
if i.slot = i.b_slot = null and s ∉ (L3 or L1) then
   With probability, P(s) do
        send REQ(i, rx_ID, L2, |R_i|)
        rx_ID = R_i → next
   End do
end if
if i.slot = s or i.b_slot = s then
   send REQ(i, rx_ID, L2, nr)
   rx_ID = R_i → next
end if
//perform channel listening
if i receives a REQ(j, dest_ID, L2, state) then
   if REQ.dest_ID = i then
      add REQ.j in S_i,
      send RES(i, j, L2, REQ.state)
   end if
   if j ∈ R_i and REQ.state = 0 then
      slot s has been taken by node j, add s to L1
      if j ∈ S_i add s to L2
      end if
   end if
   if j ∈ S_i or j ∈ R_i and REQ.state ≠ 0 then
      slot s is blocked, add s to L3
   end if
end if
if i receives a RES(j, dest_ID, L2, state)
   if RES.dest_ID = i then
      if nr = |R_i| then i.b_slot = s
      end if
      nr = nr − 1
      if nr = 0 then i.slot = s
      end if
   else
      if RES.state = 0 then
         slot is taken by RES.dest_ID, add s to L1
         if RES.dest_ID ∈ S_i add s to L2
         end if
      else
         slot s is blocked, add s to L3
      end if
   end if
end if
if i.slot = null and not received the RES for transmitted REQ message then
   i.b_slot = null, nr = |R_i|
end if
if s ∈ L3
   Remove s from L3 if blocked duration has expired
end if
```

Algorithm 1: DTSS algorithm.

describes the DTSS algorithm as executed on each node i at the current slot s.

Each node i, contending for a time slot, passes through several states. Figure 3 shows the finite state transition diagram for a node i. Initially, node i enters *contention state* (CS), where it sends a REQ message in the current time slot s, with probability $P(s)$. We call this probability as slot probability and it is equal to $1/(\mathcal{N} - |L1|)$. On receipt of a REQ message at a node j from node i, it sends a RES message immediately in the current slot s and also adds the node i to S_j if its ID is the same as the destination ID in the received REQ message. The duration of slot is kept sufficiently large to carry out the transmission of a pair of REQ and RES messages. The destination ID j of REQ message transmitted by node i in CS state can be any node from the set R_i. If a node i receives a RES message at time slot s in response to the REQ message sent by it and $|R_i| > 1$, then it blocks the time slot s and enters the *verification state* (VS). However, if $|R_i| = 1$, it assigns

Table 1: The set of data structures and variables maintained by a node i.

Notation	Description
R_i	A list of receivers of node i, maintained as circular linked list.
S_i	A list of transmitters of node i, constructed dynamically on receipt of REQ messages.
$L1$	A list of slots which are already taken by the nodes adjacent to i in G.
$L2$	A list of slots which are already taken by the nodes in S_i. Note that $L2 \subseteq L1$.
$L3$	A list of slots which are currently blocked by the nodes adjacent to i in G.
$\mathcal{N}$	Number of slots in a frame.
i.slot	Slot assigned to node i.
i.b_slot	Slot blocked by node i.
rx_ID	Current receiver node in R_i.
nr	Remaining number of receivers from where responses are required to be obtained by node i.
$P(s)$	Probability of transmitting a request message in slot s by node i, while contending for the slot.

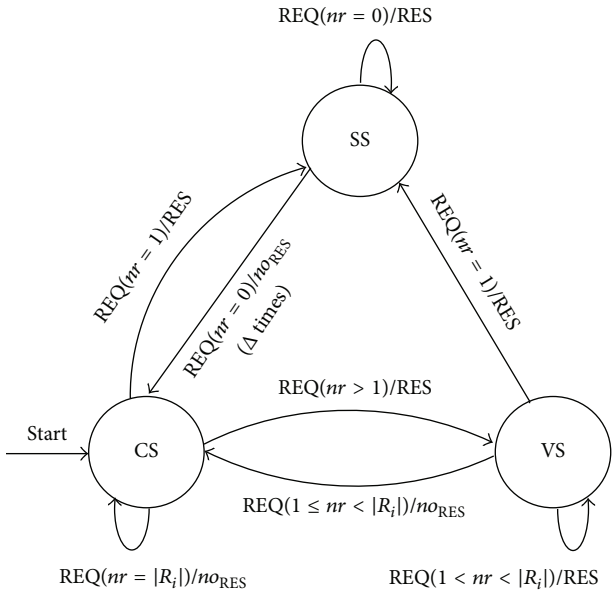


Figure 3: State transition diagram of a node i.

the slot to itself and enters the scheduled state (SS) directly. In VS state, it sends REQ messages one by one to the remaining nodes in R_i at the same time slot s in subsequent frames, by setting the pointer rx_ID to the next node in the list R_i. Furthermore, it does not transmit in slots other than s, while it is in VS state. If the node i successfully receives the RES messages from all of its receivers in R_i, it assigns the slot to itself and enters the *scheduled state* (SS); otherwise, it goes back to the CS state and starts the process all over again. In SS state, the node i always transmits REQ message in slot s so that no other node can take the same slot and also it does not transmit at slots other than s. The destination of REQ message in SS state is selected in a round robin fashion among the nodes in R_i. Moreover, it does not progress to the next receiver until it receives a RES message from the current receiver. If a node i does not receive RES message consecutively Δ times from the same receiver node, then it goes back to the CS state.

If a node i in CS state receives a REQ message from node j at slot s with state > 0 and $j \in S_i$ or $j \in R_i$, then it adds the slot s in the list $L3$. If a node i in CS state receives a REQ message from node j at slot s with state $= 0$ and $j \in R_i$ it adds the slot s in the list $L1$ and also updates the slot probability of other slots not in $L1$ as $1/(\mathcal{N} - |L1|)$. Additionally, if $j \in S_i$, then it also adds the slot s in the list $L2$.

If a node i in CS state receives a RES message in response to REQ message from itself and $nr = |R_i|$, then it blocks the slot. Then it decreases nr by 1, and if nr becomes zero, it assigns the slot to itself. If a node i in CS state receives a RES message from node j in response to REQ message from node k at slot s with state > 0, then it adds the slot s in the list $L3$. A node does not transmit its own REQ messages in the slots belonging to $L3$ for the number of subsequent frames specified in the state field of the received RES message. This allows node k in VS state to successfully transmit its remaining REQ messages and subsequently move to SS state. If a node i in CS state receives a RES message from node j in response to REQ message from node k at slot s with state $= 0$, it adds the slot s in the list $L1$ and also updates the slot probability of other slots not in $L1$ as $1/(\mathcal{N} - |L1|)$. Node i permanently leaves the slots in $L1$ and does not transmit any further REQ messages in these slots. Additionally, if $k \in S_i$, then it adds the slot s in the list $L2$.

It could be possible that a node j does not receive the transmission of a REQ message from node i or RES message in response to the REQ message from node i in slot s because REQ/RES messages could get lost. In this situation, node j would not come to know that the slot s is either blocked or taken by node i until the transmissions of REQ/RES message at the same slot in the next frame. To avoid this delay, the nodes in R_i convey the same information through the field $L2$ of their own REQ messages transmitted in slots other than s. In this case, while a node is trying to take a slot, it is helping others to know the slots which are already taken by other conflicting nodes.

Finally, a node j, with $j \rightarrow i$, can enter into state SS, while node i is already in state SS. This is because the transmission of REQ messages from node i in slot s cannot interfere at any of the nodes in R_j, and therefore, node j will receive the RES messages from all of its receivers and move to state SS. On the other hand, the REQ messages sent by node i in SS state will collide at one or more receivers in R_i due to transmission from node j and therefore node i will not receive the corresponding RES messages. The above situation is shown in Figure 2(b). However, this can only happen if node j is not aware that the slot is already taken by node i. To avoid this, if the RES is not received by node i for consecutive Δ times, it leaves the slot by adding it to the list $L1$ and comes back to the CS state. If node j is not in SS state, it cannot collide with the transmission of node i in the same slot consecutively Δ times. This ensures that node i will only leave the slot s, if $j \rightarrow i$ and j is in SS state.

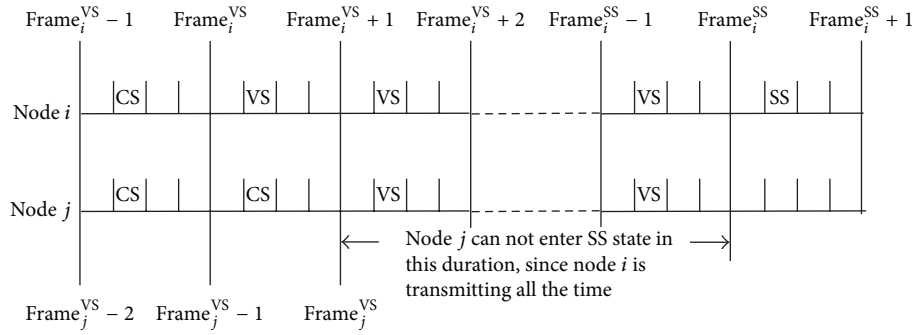


FIGURE 4: Node j cannot enter SS state since node i is continuously transmitting REQ messages from frame index frame$_i^{VS}$.

5. Correctness of the DTSS Algorithm

In this section, we prove that the schedule created by the DTSS algorithm is a feasible TDMA schedule. In a feasible schedule, two conflicting nodes will not transmit in the same time slot. That is, two conflicting nodes will not be assigned the same time slot by the DTSS algorithm. This happens because after the execution of the DTSS algorithm is completed, only one node (among the conflicting nodes) will remain in the SS state for a particular time slot. In the following, we prove this fact as Theorems 6 and 7 for strong-conflict and weak-conflict relationship, respectively.

Theorem 6. *If two nodes i and j have strong-conflict relationship, then they cannot be in SS state for the same time slot, at any time during the execution of DTSS algorithm.*

Proof. Let frame$_i^{VS}$ and frame$_i^{SS}$ be the frame indexes when node i enters VS and SS state, respectively, for a time slot k. It is possible that frame$_i^{VS}$ = frame$_i^{SS}$ if $|R_i| = 1$. Furthermore, let frame$_j^{VS}$ and frame$_j^{SS}$ be the corresponding frame indexes for node j for the same time slot k. We can assume that once a node enters VS state from CS state, it remains in VS state until it enters SS state. If it is not so, then it goes back to CS state, and the argument can be repeated. It is to be noted that both cannot enter SS state at the same frame index without at least one of them going into VS state first. Now the following three cases arise.

Case 1 (frame$_i^{VS}$ < frame$_j^{VS}$). In this case, only node i can enter SS state provided it has got response from all nodes $k \in N_j \cap R_i$ prior to frame$_j^{VS}$. There is no way node j can enter SS state since node i will be continuously transmitting REQ messages from frame index frame$_i^{VS}$, and node j cannot get response from any node in $N_i \cap R_j$ (Figure 4). Hence, in this case only node i will be able to enter SS state.

Case 2 (frame$_i^{VS}$ = frame$_j^{VS}$). In this case, both node i and node j will be transmitting REQ messages from frame index frame$_i^{VS}$ onwards. Therefore, node i will not be able to get response from any node in $N_j \cap R_i$. Similarly, node j will not get response from any node in $N_i \cap R_j$. So, the node which does not receive the response first will go back to CS state. As a result neither node i nor node j will be able to enter SS state as Case 2.

Case 3 (frame$_i^{VS}$ > frame$_j^{VS}$). This case is similar to Case 1 except that now node j can enter SS state provided it satisfies the corresponding condition.

From the above argument, it is clear that only one of nodes i or j will be in SS state for the same time slot during the execution of the DTSS algorithm. Hence, the theorem is proved. □

In case of weak-conflict relationship, it could be possible that while a node i is already in SS state, another node j (stronger than node i) can enter SS state for the same time slot, during the execution of DTSS algorithm. Therefore, Theorem 6 does not sufficiently prove the correctness of DTSS algorithm when weak-conflict relationship also exists between the nodes.

Theorem 7. *If two nodes i and j have weak-conflict relationship, then eventually only one of nodes i and j will remain in SS state for the same time slot after the execution of the DTSS algorithm is completed.*

Proof. Let frame$_i^{VS}$, frame$_i^{SS}$, frame$_j^{VS}$, and frame$_j^{SS}$ be frames indexes of nodes i and j as in Theorem 6. Also, without loss of generality assume that $N_i \cap R_j \neq \phi$ and $N_j \cap R_i = \phi$. That is, node i is stronger than node j. As in Theorem 6, we also assume that, after entering VS state, both remain there until they enter SS state. It is to be noted that both cannot enter SS state at the same frame index directly from CS state, without at least one of them going into VS state first. In this case also, the following three cases arise.

Case 1 (frame$_i^{VS}$ < frame$_j^{VS}$). This case is similar to Case 1 of Theorem 6, and only node i will be able to enter SS state, and node j will not be able to enter SS state.

Case 2 (frame$_i^{VS}$ = frame$_j^{VS}$). In this case also, only node i will be able to enter SS state, and node j will not be able to enter SS state.

Case 3 (frame$_i^{VS}$ > frame$_j^{VS}$). In this case, node i can enter SS state anyway. However node j can also enter SS state provided it has got response from all the nodes in $N_i \cap R_j$ before

Table 2: The set of notations used in Section 6.

Notation	Description
X_i	The time slot at which ith node enters SS state.
Y_i	Number of time slots between times X_i and X_{i-1}
P_{succ}	The probability that only one node transmits a REQ message in a slot after and the message is received successfully.
q_i	The probability of a node i entering SS state, in a round (frame).
q	The probability of a node with ID 1 entering SS state, in a round (the subscript 1 omitted from sake of clarity).
$q_{\min}$	Minimum value of q.
$q(k)$	The probability of node with ID 1 entering SS state at slot k in a round.
q_{sum}	$\sum_{k=1}^{\mathcal{N}} q(k)$.
β_i	The number of 1's in row i of matrix B.
α_j	The number of 1's in column j of matrix B, excluding first row.
$\pi_{i,j}$	The transition probability from state i to state j of DTMC, presented in Section 6.
τ_i	The number of rounds required to reach state "n" in DTMC, starting from state "i."

frame index $\text{frame}_i^{\text{VS}}$. Let us assume that node j satisfies this condition. Now nodes i and j can enter SS state in any order. Assume that both nodes i and j are in SS state; then node j would not be able to get response continuously Δ times from a node in $N_i \cap R_j$. As a result, node j would move back to CS state, and it would not be able to enter SS state again. □

6. Complexity Analysis of DTSS Algorithm

In this section, we evaluate the expected runtime of DTSS algorithm, that is, the time when all nodes in the network reach SS state. Table 2 summarizes the set of notations used in this section. First, we consider the situation, when all nodes in the network interfere with each other's transmission; that is, the interference graph G is complete. This situation mainly occurs in single-hop WSNs. In this case, only a single transmission of REQ message in a slot can be successful, and therefore, nodes can enter SS state one at a time in each slot, as shown in Figure 5. Furthermore, we assume that $|R_i| = 1$, for each node i. In this case nodes directly enter SS state without entering the VS state. The analysis can be further extended for the case when $|R_i| > 1$. Initially, every node transmits REQ message with probability $1/\mathcal{N}$ in every slot.

Let the time slot at which ith node enters SS state be X_i. Note that X_i is a random variable. Clearly, X_n is the time slot when last node enters SS state, which is exactly the desired runtime of DTSS algorithm. Let $Y_i = X_i - X_{i-1}$. In this case,

$$X_i = X_{i-1} + Y_i,$$
$$EX_i = EX_{i-1} + EY_i = \sum_{j=1}^{i} EY_j. \tag{1}$$

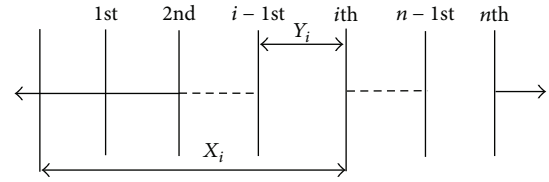


Figure 5: Sequence of slot assignment in a single-hop WSN. The nodes cannot enter state simultaneously in a slot.

Theorem 8. *EX_n is $O(n)$ for single-hop WSNs.*

Proof. At time slot X_{i-1}, exactly $i-1$ nodes are in SS state and for the remaining $n-i+1$ nodes which are not in SS state, set their slot probability to $1/(n-i+1)$, for unoccupied slots. Let P_{succ} be the probability that only one node transmits a REQ message in a slot after time slot X_{i-1} and the message is received successfully at the intended receiver. Note that the REQ message could be lost not only due to collisions but also because of channel impairment. Therefore, a successful packet transmission also depends upon packer error rate (PER). Y_i is a geometric random variable with success probability P_{succ} and expectation $1/P_{\text{succ}}$. The upper bound of EY_n (runtime) can be calculated as follows:

$$\begin{aligned} P_{\text{succ}} &= \binom{n-i+1}{1} * \frac{1}{n-i+1} \\ &\quad * \left(\frac{n-i}{n-i+1}\right)^{n-i} * (1-\text{PER}) \\ &= \left(\frac{n-i}{n-i+1}\right)^{n-i} * (1-\text{PER}), \\ EY_i &= \frac{1}{P_{\text{succ}}} = \left(\frac{n-i+1}{n-i}\right)^{n-i} * \frac{1}{(1-\text{PER})} \\ &= \left(1+\frac{1}{n-i}\right)^{n-i} * \frac{1}{(1-\text{PER})}, \end{aligned} \tag{2}$$

$$EY_i \le \frac{e}{(1-\text{PER})},$$

$$\because \left\{\left(1+\frac{1}{n-i}\right)^{n-i}\right\},$$

is monotonically increasing and converges to e

$$EX_n = \sum_{j=1}^{n} EY_j \le \frac{1}{1-\text{PER}} \sum_{j=1}^{n} e = \frac{ne}{1-\text{PER}} = O(n).$$

□

Now, we will consider a more generalized situation when not all nodes in the network interfere with each other's transmission; that is, the interference graph G is not necessarily complete. Again we assume that $|R_i| = 1$. The above situation mainly occurs in multihop WSNs. Further, we assume that the graph G is regular with degree $\mathcal{N} - 1$. Note that $\mathcal{N}$ is always taken to be greater than Δ, the maximum degree

of interference graph. Therefore, assuming the graph to be regular with degree $\mathcal{N}-1$ will give the worst case analysis; that is, the expected runtime of DTSS algorithm for a nonregular interference graph with $\Delta = \mathcal{N} - 1$ will always be less than or equal to the expected runtime for a regular interference graph of degree $\mathcal{N} - 1$.

We will first find out for an arbitrary node i what the minimum value is that q_i can take, irrespective of the slot probabilities of other nodes in the network. This will help us to set an upper bound on the expected time, required by any node, to reach SS state.

Let us rearrange the IDs of the nodes in the following manner.

(i) The ID of node i is changed to 1.

(ii) The IDs of nodes adjacent to node i in G would be changed from 2 to $\mathcal{N}$. The ordering among these nodes could be arbitrary.

(iii) The IDs of all other nodes would become $\mathcal{N} + 1$ to n. The ordering among these nodes could be arbitrary.

Note that, the above rearrangement will not change the probability of node i entering SS state in a round. Our task is to find out q_1 instead of q_i. Further, we can also omit the subscript 1 from q_1 for sake of clarity.

Let the probability of node with ID 1 (after rearrangement) entering SS state at slot k in a round be $q(k)$. The value of $q(k)$ depends upon the transmission probability of node 1 in slot k and the transmission probabilities of its adjacent nodes in the same slot. Consider

$$q(k) = p_1(k) * \prod_{j=2}^{\mathcal{N}} \left(1 - p_j(k)\right). \tag{3}$$

The probability that node 1 can enter SS state in a round is equal to the probability that it can enter SS state in at least one slot of the round. Therefore, q can be written in terms of $q(k)$ as

$$q = 1 - \prod_{k=1}^{\mathcal{N}} \left(1 - q(k)\right). \tag{4}$$

In order to find the minimum value of q, we define another term q_{sum} as a function of $q(k)$'s as follows:

$$q_{\text{sum}} = \sum_{k=1}^{\mathcal{N}} q(k). \tag{5}$$

We know that, for a constant sum, the product can be maximized when the sum is partitioned equally [26]. Therefore, for a constant value of q_{sum}, q can achieve its minimum value q_{min}, if $q(l) = q(m)$, $1 \le l, m \le \mathcal{N}$. Obviously, q_{min} is a function of q_{sum}.

Theorem 9. *Let $q_{\text{min}} = \mathcal{F}(q_{sum})$. Then $\mathcal{F}$ is a monotonically increasing function.*

Proof. Let x and y be the two values of q_{sum}, such that $x > y$. We know that q_{min} is achieved when $q(l) = q(m)$, $1 \le l, m \le \mathcal{N}$. Let $c1$ and $c2$ be the corresponding values of $q(k)$, for all k with respect to x and y. In this case, the value of $\mathcal{F}(x)$ and $\mathcal{F}(y)$ would be $1 - (1 - c1)^{\mathcal{N}}$ and $1 - (1 - c2)^{\mathcal{N}}$, respectively (4). Therefore,

$$\begin{aligned} x > y &\Longrightarrow \mathcal{N}c1 > \mathcal{N}c2 \Longrightarrow c1 > c2 \\ &\Longrightarrow (1 - c1) < (1 - c2) \Longrightarrow (1 - c1)^{\mathcal{N}} < (1 - c2)^{\mathcal{N}} \\ &\Longrightarrow 1 - (1 - c1)^{\mathcal{N}} > 1 - (1 - c2)^{\mathcal{N}} \\ &\Longrightarrow \mathcal{F}(x) > \mathcal{F}(y). \end{aligned} \tag{6}$$

□

It is clear from Theorem 9 that, to find q_{min}, we need to first minimize the q_{sum}. Let us define a binary square matrix, B, of size $\mathcal{N}$, in the following manner:

$$b_{i,j} = \begin{cases} 1, & \text{if } p_i(j) > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

The matrices P and B show an example of probability matrix and its corresponding binary transformation for $\mathcal{N} = 3$. Consider

$$P = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}. \tag{8}$$

Let $\beta_i = \sum_{j=1}^{\mathcal{N}} b_{i,j}$ (number of 1's in row i of matrix B) and $\alpha_j = \sum_{i=2}^{\mathcal{N}} b_{i,j}$ (number of 1's in column j of matrix B, excluding first row). The $q(k)$ can be rewritten in terms of $b_{j,k}$ and β_j, $1 \le j \le \mathcal{N}$, as follows:

$$q(k) = \frac{b_{1,k}}{\beta_1} \prod_{j=2}^{\mathcal{N}} \left(1 - \frac{b_{j,k}}{\beta_j}\right). \tag{9}$$

Let B_{min} be the matrix for which the value of q_{sum} is minimum. To find out the properties of B_{min}, we start with the hypothesis that q would be minimum, if none of the nodes adjacent to node 1 is in SS state. This implies that node 1 is still transmitting in all the slots with probability $1/\mathcal{N}$; that is, $b_{1,k} = 1$, for all k, $1 \le k \le \mathcal{N}$. Now, we will present two lemmas based on the above hypothesis; this hypothesis will be used to find out the properties of B_{min} in Theorem 12, where we also explain the need for it.

Lemma 10. *For a given instance of matrix B, let $b_{1,k} = 1$, for all k, $1 \le k \le \mathcal{N}$, and for a slot j, $q(j) \le q(k)$, for all $k \ne j$. Then, for any row i, q_{sum} reduces or remains the same, if $b_{i,j}$ is changed from 1 to 0.*

Proof. Let $q_{\text{sum}}^{\text{old}}$ and $q_{\text{sum}}^{\text{new}}$ be the respective sums before and after the conversion of $b_{i,j} = 1$ to 0. We need to show that $q_{\text{sum}}^{\text{old}} \ge q_{\text{sum}}^{\text{new}}$. Similarly, $q^{\text{old}}(j)$ and $q^{\text{new}}(j)$ can be defined.

Since $b_{1,k} = 1$, for all k, $1 \le k \le \mathcal{N}$, the $q^{\text{old}}(j)$ can be written as

$$\begin{aligned} q^{\text{old}}(j) &= \frac{1}{\mathcal{N}} \prod_{k=2}^{\mathcal{N}} \left(1 - \frac{b_{k,j}}{\beta_k}\right) \\ &= \frac{1}{\mathcal{N}} \left(\prod_{k=2,k\neq i}^{\mathcal{N}} \left(1 - \frac{b_{k,j}}{\beta_k}\right)\right) * \left(1 - \frac{1}{\beta_i}\right) \end{aligned} \tag{10}$$

and since $b_{i,j}$ becomes 0, after the conversion, $q^{\text{new}}(j)$ would be

$$q^{\text{new}}(j) = \frac{1}{\mathcal{N}} \left(\prod_{k=2,k\neq i}^{\mathcal{N}} \left(1 - \frac{b_{k,j}}{\beta_k}\right)\right). \tag{11}$$

Therefore, from (10) and (11), we get

$$q^{\text{new}}(j) - q^{\text{old}}(j) = \frac{q^{\text{old}}(j)}{\beta_i - 1}. \tag{12}$$

Similarly, for all other slots $k \neq j$ and $b_{i,k} = 1$,

$$q^{\text{new}}(k) - q^{\text{old}}(k) = -\frac{q^{\text{old}}(k)}{(\beta_i - 1)^2}. \tag{13}$$

To show that $q_{\text{sum}}^{\text{old}} \ge q_{\text{sum}}^{\text{new}}$, we calculate $q_{\text{sum}}^{\text{new}} - q_{\text{sum}}^{\text{old}}$ as follows:

$$\begin{aligned} &q_{\text{sum}}^{\text{new}} - q_{\text{sum}}^{\text{old}} \\ &= \sum_{k=1}^{\mathcal{N}} q^{\text{new}}(k) - \sum_{k=1}^{\mathcal{N}} q^{\text{old}}(k) \\ &= \sum_{k=1}^{\mathcal{N}} \left(q^{\text{new}}(k) - q^{\text{old}}(k)\right) \\ &= \left(\sum_{k\neq j, b_{i,k}=1} q^{\text{new}}(k) - q^{\text{old}}(k)\right) + \left(q^{\text{new}}(j) - q^{\text{old}}(j)\right) \\ &= \left(\sum_{k\neq j, b_{i,k}=1} -\frac{q^{\text{old}}(k)}{(\beta_i - 1)^2}\right) + \frac{q^{\text{old}}(j)}{\beta_i - 1} \\ &= \left(\sum_{k\neq j, b_{i,k}=1} \frac{-q^{\text{old}}(j) + \left(q^{\text{old}}(j) - q^{\text{old}}(k)\right)}{(\beta_i - 1)^2}\right) + \frac{q^{\text{old}}(j)}{\beta_i - 1} \\ &= \left(\sum_{k\neq j, b_{i,k}=1} \frac{-q^{\text{old}}(j)}{(\beta_i - 1)^2}\right) \\ &\quad + \left(\sum_{k\neq j, b_{i,k}=1} \frac{\left(q^{\text{old}}(j) - q^{\text{old}}(k)\right)}{(\beta_i - 1)^2}\right) + \frac{q^{\text{old}}(j)}{\beta_i - 1}. \end{aligned} \tag{14}$$

Since the number of 1's in row i is β_i, the number of terms in the first summation of above equation would be exactly $\beta_i - 1$. Therefore,

$$\begin{aligned} q_{\text{sum}}^{\text{new}} - q_{\text{sum}}^{\text{old}} &= \sum_{k\neq j, b_{i,k}=1} \frac{q^{\text{old}}(j) - q^{\text{old}}(k)}{(\beta_i - 1)^2} \le 0, \\ &\because q(j) \le q(k), \quad \forall k \neq j. \end{aligned} \tag{15}$$

□

Lemma 11. *For a given instance of matrix B, let $b_{1,k} = 1$, for all k, $1 \le k \le \mathcal{N}$, and $\beta_i = 2$, for all i, $2 \le i \le \mathcal{N}$. Then the following holds. For any two columns j and k and, for any row i, such that $\alpha_j > \alpha_k$, $b_{i,j} = 1$ and $b_{i,k} = 0$, q_{sum} either reduces or remains the same if the values of $b_{i,j}$ and $b_{i,k}$ are interchanged.*

Proof. Consider $q_{\text{sum}}^{\text{old}}$, $q_{\text{sum}}^{\text{new}}$, $q^{\text{old}}(j)$, and $q^{\text{new}}(j)$ as defined in Lemma 10. We need to show that $q_{\text{sum}}^{\text{old}} \ge q_{\text{sum}}^{\text{new}}$. Here, $q^{\text{old}}(j) = (1/\mathcal{N}) * 1/2^{\alpha_j}$, $q^{\text{old}}(k) = (1/\mathcal{N}) * 1/2^{\alpha_k}$, $q^{\text{new}}(j) = (1/\mathcal{N}) * 1/2^{\alpha_j - 1}$, and $q^{\text{new}}(k) = (1/\mathcal{N}) * 1/2^{\alpha_k + 1}$. Therefore,

$$\begin{aligned} q_{\text{sum}}^{\text{new}} - q_{\text{sum}}^{\text{old}} &= \left(q^{\text{new}}(j) + q^{\text{new}}(k)\right) - \left(q^{\text{old}}(j) + q^{\text{old}}(k)\right) \\ &= \left(q^{\text{new}}(j) - q^{\text{old}}(j)\right) + \left(q^{\text{new}}(k) + q^{\text{old}}(k)\right) \\ &= \left(\frac{1}{2^{\alpha_j - 1}} - \frac{1}{2^{\alpha_j}}\right) + \left(\frac{1}{2^{\alpha_k + 1}} - \frac{1}{2^{\alpha_k}}\right) \\ &= \frac{1}{2^{\alpha_j}} - \frac{1}{2^{\alpha_k + 1}} \le 0 \quad \because \alpha_j > \alpha_k. \end{aligned} \tag{16}$$

□

Now we will try to prove that $B_{\min}$ should satisfy a few constraints, in terms of α_i and β_i, $1 \le i \le \mathcal{N}$, with the help of Lemmas 10 and 11.

Theorem 12. *$B_{\min}$ has the following properties:*

(1) *$\beta_i = 2$, for all i, $2 \le i \le \mathcal{N}$;*

(2) *for exactly two columns $j1$ and $j2$, $\alpha_{j1} = \alpha_{j2} = 1$ and for all other columns $k \neq j1, j2$, $\alpha_k = 2$.*

Proof. We prove both the properties for two different cases: $\beta_1 = \mathcal{N}$ and $\beta_1 \neq \mathcal{N}$.

Case 1 ($\beta_1 = \mathcal{N}$). The property (1) can be proved by contradiction. First, we show that $\beta_i \ge 2$, $2 \le i \le \mathcal{N}$. If $\beta_i = 1$ with $b_{i,j} = 1$, for some row i, then node i is in SS state. Therefore, node 1 should have stopped transmitting in slot j, that is, $b_{1,k} = 0$, which contradicts our assumption that $\beta_1 = \mathcal{N}$. Now, we show that $\beta_i \le 2$, $2 \le i \le \mathcal{N}$. Let $\exists i : \beta_i > 2$ and $\mathcal{A}$ the set of column indexes k for which $b_{i,k} = 1$; then $\exists j \in \mathcal{A}$, such that $q(j) \le q(k)$, for all $k \in A$, $j \neq k$. Therefore, by the virtue of Lemma 10, q_{sum} reduces or remains the same, if $b_{i,k}$ is changed from 1 to 0. The same process can be repeated till $\beta_i = 2$.

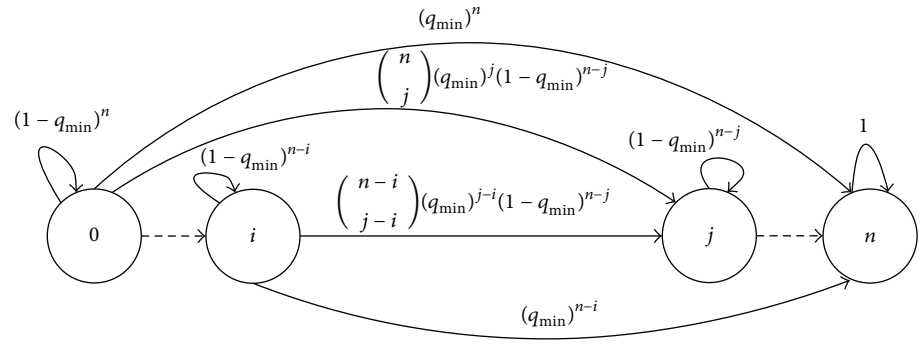


Figure 6: Discrete time Markov chain (DTMC) with number of nodes in SS state as random variable, assuming the probability of entering SS state, as $q_{\min}$, for each node in the network.

The property (2) can also be proved by contradiction. We know that $\beta_1 = \mathcal{N}$ and $\beta_i = 2$, $2 \le i \le \mathcal{N}$. Therefore, $\sum_{j=1}^{\mathcal{N}} \alpha_j = 2(\mathcal{N}-1)$. First, we show that $\alpha_j \le 2$, $1 \le j \le \mathcal{N}$. For a column j, $\alpha_j > 2 \Rightarrow \exists k : \alpha_k < 2$; otherwise $\sum_{j=1}^{\mathcal{N}} \alpha_j$ would become less than $2(\mathcal{N}-1)$. In this case, for any row i, such that $b_{\{i,j\}} = 1$ and $b_{\{i,k\}} = 0$, the value of $b_{\{i,j\}}$ and $b_{\{i,k\}}$ can be interchanged by virtue of Lemma 11. This proves that α_j could be either 0, 1 or 2, $1 \le j \le \mathcal{N}$. Since, any column can have at most two 1's, this implies that at most one column of type $\alpha_i = 0$ can exist and that also can be increased to 1 by virtue of Lemma 11. Furthermore, the number of columns of type $\alpha_i = 1$ cannot be one, since $2(\mathcal{N}-1)$ is even. Finally, we can say that number of columns of type $\alpha_i = 1$ is exactly 2; otherwise, the total sum will be less than $2(\mathcal{N}-1)$.

Case 2 ($\beta_1 \ne \mathcal{N}$). Let $q_{\text{sum}}^{\text{case 1}}$ and $q_{\text{sum}}^{\text{case 2}}$ be the corresponding summation for Cases 1 and 2, respectively. The value of $q_{\text{sum}}^{\text{case 1}}$ would be $(\mathcal{N}+2)/4\mathcal{N}$. We will prove that $q_{\text{sum}}^{\text{case 1}} < q_{\text{sum}}^{\text{case 2}}$ by showing that any perturbation in the matrix corresponding to Case 1 will increase the value of q_{sum}. We have already proved, in Case 1, that any modification in any of the rows from 2 to row $\mathcal{N}$ and leaving row 1 unchanged will increase q_{sum}. Now, let us change a single entry $b_{1,k} = 1$ to 0; that is, node 1 has decided not to transmit in slot k. This only happens when at least one adjacent node i in G has gone to SS state for slot k, which implies that $b_{i,k} = 1$ and $b_{i,j} = 0$, for all $j \ne k$. Let us interchange the row i with row $\mathcal{N}$ and column k with column $\mathcal{N}$. In this case, $b_{1,\mathcal{N}} = 0$, $b_{1,j} = 1$, for all $j \ne \mathcal{N}$, and $b_{\mathcal{N},\mathcal{N}} = 1$ and $b_{\mathcal{N},j} = 0$, for all $j \ne \mathcal{N}$. Consider the submatrix of size $\mathcal{N}-1$ times $\mathcal{N}-1$. The minimum value of q_{sum} which can be achieved by this submatrix would be $(\mathcal{N}+1)/4(\mathcal{N}-1)$. Moreover, $q(\mathcal{N}) = 0$, because $b_{1,\mathcal{N}} = 0$. Therefore, $q_{\text{sum}}^{\text{case 2}} = (\mathcal{N}+1)/4(\mathcal{N}-1) > (\mathcal{N}+2)/4\mathcal{N} = q_{\text{sum}}^{\text{case 1}}$. □

From Theorem 9, we know that the $q_{\min}$ can be achieved when q_{sum} is minimum and $B_{\min}$ should satisfy the properties as given in Theorem 12. Therefore,

$$q_{\min} = 1 - \left(\frac{4 * \mathcal{N} - 1}{4 * \mathcal{N}}\right)^{(\mathcal{N}-2)} * \left(\frac{2 * \mathcal{N} - 1}{2 * \mathcal{N}}\right)^2. \quad (17)$$

The following matrix shows one of such $B_{\min}$ matrix for $\mathcal{N} = 4$:

$$B_{\min} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}. \quad (18)$$

To calculate the expected runtime of DTSS algorithm, we model the behavior of the system using a discrete time Markov chain (DTMC), with the number of nodes in SS state, X_t, at the beginning of round t, as a random variable. The transition probabilities, $\pi_{i,j}$, are defined as follows:

$$\pi_{i,j} = \begin{cases} \binom{n-i}{j-i} (q_{\min})^{j-i} (1-q_{\min})^{n-j}, & j \ge i \\ 1, & i = j = n \\ 0, & \text{otherwise.} \end{cases} \quad (19)$$

In this DTMC (see Figure 6), all states are transient except state "n" which is an absorbing state. The probability of leaving a transient state "i" is always greater than 0; that is, $1 - \sum_{j>i} \pi_{i,j} > 0$. A transient state cannot be visited again, once it is left. This shows that the DTSS algorithm converges in a finite time. Let τ_i be the number of rounds required to reach state "n" starting from state "i." Our goal is to find $E[\tau_0]$, which can be calculated using the following recurrence relation:

$$E[\tau_i] = \begin{cases} 1 + \sum_{j=0}^{n} \pi_{i,j} E[\tau_j], & 1 \le i \le n-1 \\ 0, & i = n. \end{cases} \quad (20)$$

Note that the above DTMC is the approximation of actual stochastic process, where the transition probabilities not only depend upon the number of nodes in SS state, but also depend on exact nodes belonging to SS state.

We show that the value of $E[\tau_i]$ is greater than actual expected time required to reach state "n" starting from state "i" in DTMC, by proving that the transition probability of

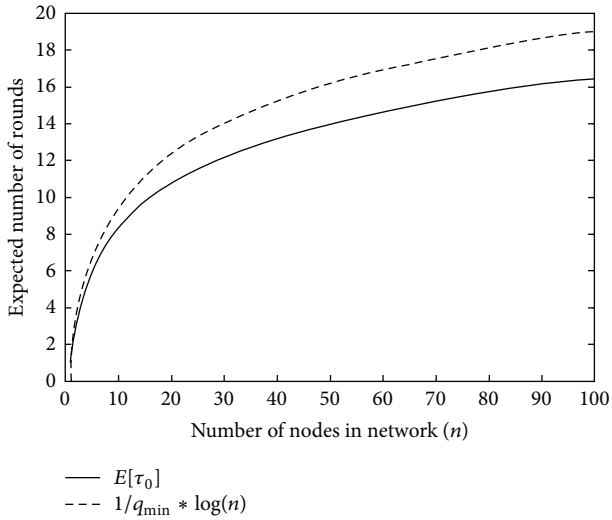


Figure 7: Runtime performance of DTSS algorithm in terms of number of rounds (frames) with respect to number of nodes in the network and its comparison with the function, $\log n/q_{\min}$.

moving from i nodes in SS state to j nodes in SS state, in an actual stochastic process, is always greater than $\pi_{i,j}$. We know that the probability of each node moving from CS state to SS state in a round is always greater than $q_{\min}$(17) and therefore the probability that, out of $n-i$ nodes in CS state, exactly $j-i$ nodes enter into the SS state is greater than $\binom{n-i}{j-i}(q_{\min})^{j-i}(1-q_{\min})^{n-j} = \pi_{i,j}$.

Figure 7 shows the graph for $E[\tau_0]$ along with function $\log n/q_{\min}$. The graph shows that $E[\tau_0]$ is upper bounded by $\log n/q_{\min}$, and therefore, for a fixed frame size, $E[\tau_n]$ is $O(\log n)$. We know from (17) that $q_{\min}$ depends only upon $\mathcal{N}$, which is a measure of two-hop network density (δ).

Another method to analyze the expected runtime of DTSS algorithm is to calculate the expectation of maximum of all X_i's, where X_i is the time taken by node i to reach SS state. Consider

$$E\left[X_{\max}\right] = E\left[\max\left(X_1, X_2, X_3, \ldots, X_n\right)\right]. \tag{21}$$

The X_is can be assumed as i.i.d (independent and identically distributed) geometric random variable with parameter $q_{\min}$. In this case the $E[X_i]$ would be higher than the actual expected time to enter SS state, by node i. The value of $E[X_{\max}]$ can be calculated as

$$\begin{aligned} E\left[X_{\max}\right] &= \sum_{k\geq 0} P\left(\left[X_{\max} > k\right]\right) \\ &= \sum_{k\geq 0} \left(1 - P\left(X_{\max} \leq k\right)\right) \\ &= \sum_{k\geq 0} \left(1 - P\left(X_i \leq k\right)^n\right) \\ &= \sum_{k\geq 0} \left(1 - \left(1 - \left(1 - \overline{q_{\min}}\right)^k\right)^n\right), \end{aligned} \tag{22}$$

where $\overline{q_{\min}} = 1 - q_{\min}$. By considering the above infinite sum as right and left hand Riemann sum approximations [29] of the corresponding integral, we obtain

$$\begin{aligned} \int_0^\infty \left(1 - \left(1 - \overline{q_{\min}}^k\right)^n\right) &\leq E\left[X_{\max}\right] \\ &\leq 1 + \int_0^\infty \left(1 - \left(1 - \overline{q_{\min}}^k\right)^n\right). \end{aligned} \tag{23}$$

With the change of variable $u = 1 - \overline{q_{\min}}^k$, we have

$$\begin{aligned} E\left[X_{\max}\right] &\leq 1 + \frac{1}{\log \overline{q_{\min}}} \int_0^1 \frac{1-u^n}{1-u} du \\ &= 1 + \frac{1}{\log \overline{q_{\min}}} \int_0^1 \left(1 + u + \cdots + u^{n-1}\right) du \\ &= 1 + \frac{1}{\log \overline{q_{\min}}} \left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right) \\ &\approx 1 + \frac{\log n}{\log \overline{q_{\min}}}. \end{aligned} \tag{24}$$

From (24), we can conclude that $E[X_{\max}]$ is the $O(\log n)$, for a fixed neighborhood density, δ. We know from (17) that $q_{\min}$ depends only upon $\mathcal{N}$, which is a measure of δ. A more rigorous analysis on expectation of the maximum of IID geometric random variables can be found in [30].

7. Simulation Results

We have used Castalia simulator [27] to study the performance of DTSS algorithm. A multihop network, based on TelosB node hardware platform that uses CC2420 transceiver [28] for communication, is used in the simulation. The transceivers run at 250 kbps data rate and dbm transmission power which approximately gives 40 m of transmission range in the absence of interference. All nodes are distributed randomly within 250 m × 250 m area. Note that, at 250 kbps, it takes about 0.5 ms to transmit a packet of size 128 bits (80 bits for the MAC header, and 48 bits for $L2$ and state payload). Hence, we set TDMA time slots to a period of 1 ms, which is sufficiently long for the transmission of REQ/RES messages. The performance of protocol has been averaged over 100 simulation runs. The neighborhood size of the network is changed by varying the number of nodes from 50 to 300. This setup produces topologies with different neighborhood density, δ, values varying between 5 and 50.

Figure 8 shows the average number of slots taken by all the nodes to decide their slot in case of broadcast scheduling for frame sizes δ and 1.3δ, respectively. The error bars denote 95% confidence intervals. Figure 8 shows that runtime increases linearly with neighborhood density, δ. Given slot size as 1 ms, the total runtime for very high density network with $\delta = \mathcal{N} = 50$ is approximately 7 s. Furthermore, if we take more slots per frame, then runtime decreases and also confidence interval improves.

Figure 9 shows the average of the number of slots taken by all the nodes to decide their slot in case of broadcast

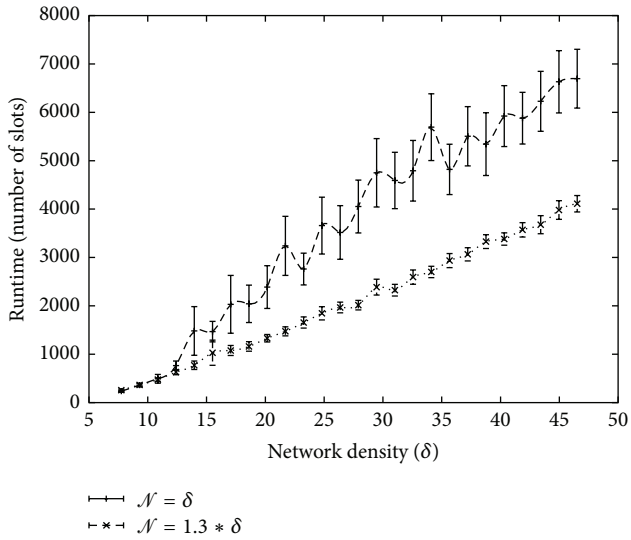


Figure 8: Runtime performance of DTSS algorithm with respect to network network density, δ, and the effect of taking the number of slots more than δ.

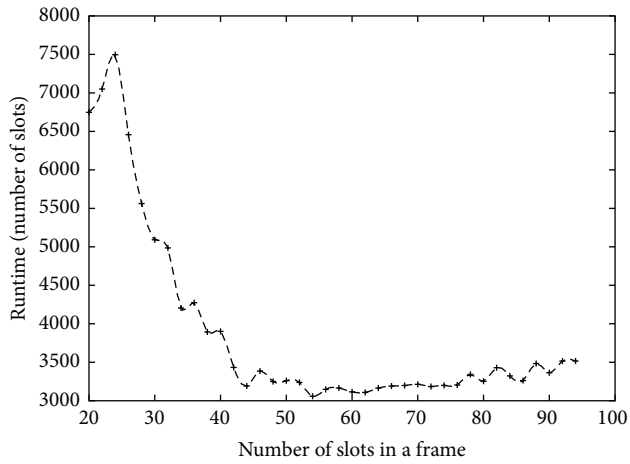


Figure 9: Runtime performance of DTSS algorithm with respect to number of slots in frame, $\mathcal{N}$.

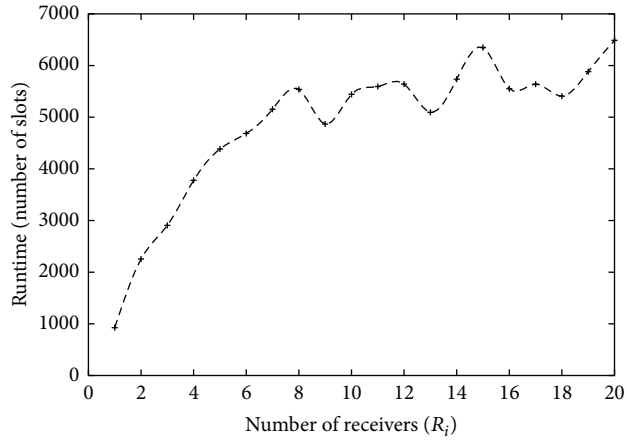


Figure 10: Runtime statistics of DTSS algorithm to show the performance with respect to unicast ($R_i = 1$), multicast, and broadcast ($R_i > 1$) mode of transmissions.

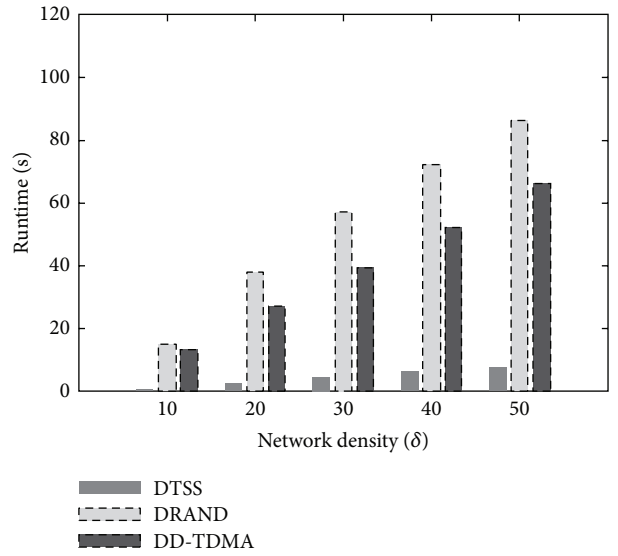


Figure 11: The runtime performance comparison of DTSS algorithm against DRAND and DD-TDMA, with respect to network density, δ.

scheduling for varying $\mathcal{N}$ values starting from δ. Figure 9 shows that the runtime reduces rapidly with small increase in $\mathcal{N}$ and further increase in Δ does not have much impact on runtime. This fact can be utilized as a tradeoff between runtime and frame length.

Figure 10 shows the average of the number of slots taken by all the nodes to decide their slot for varying the number of receivers (unicast to broadcast) with $\delta = \mathcal{N} = 40$. Figure 10 suggests that unicast or link scheduling can be performed in less than one second for a network with fairly high network density.

We now compare DTSS with DRAND [12] and DD-TDMA [13]. Figure 11 shows the performance results of DTSS along with DRAND and DD-TDMA with respect to runtime of each algorithm. The comparison is based on broadcast transmission because both DRAND and DD-TDMA only implement this mode of transmission. The primary reason of getting less runtime is because the DTSS generates a feasible schedule when the number of available slots is already fixed, whereas other algorithms try to generate a suboptimal schedule by using greedy approach, which is inherently sequential. In case of unicast and multicast scheduling, the DTSS even takes lesser time to compute the schedule as compared to broadcast transmission. The number of slots taken by DTSS is always δ as shown in Figure 12, whereas the number of time slots taken by DRAND and DD-TDMA can be less than δ.

8. Conclusions and Future Work

For many applications in WSNs, efficiently handling the spatially correlated contention is an important requirement. The DTSS takes very less time to perform the scheduling

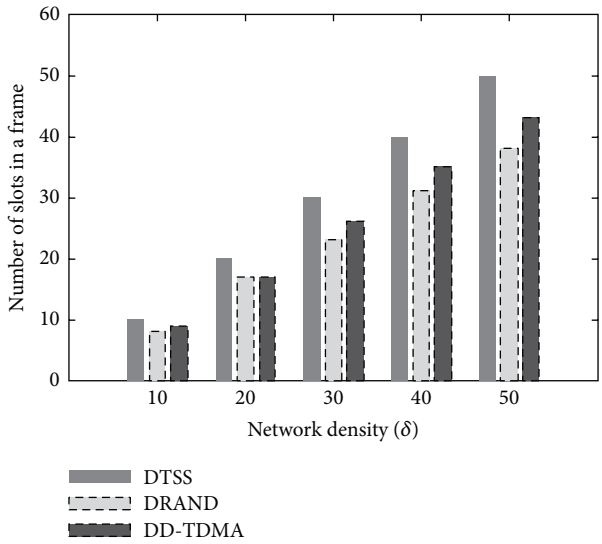


Figure 12: A comparison on frame size of DTSS algorithm against DRAND and DD-TDMA, with respect to network density, δ.

as compared to other existing distributed scheduling algorithms. We have shown that the runtime of DTSS algorithm is $O(n)$ and $O(\log(n))$ for single-hop and multihop WSNs, respectively, and therefore it is scalable for WSNs with large number of nodes. The interference model used by DTSS is more realistic than conventional protocol interference model. Additionally, the DTSS has a unique feature of unified scheduling in which simultaneously a few nodes can take a slot for unicast, while other nodes can take it for multicast or broadcast purpose. Although the number of slots taken by DTSS is bounded by Δ, further efforts can be applied to reduce the number of slots. In future, we plan to work on the variation of DTSS algorithm, for the situation, when nodes are not assumed to be synchronized before performing the slot scheduling.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] H. Alemdar and C. Ersoy, "Wireless sensor networks for healthcare: a survey," *Computer Networks*, vol. 54, no. 15, pp. 2688–2710, 2010.

[2] M. A. Maarof, M. A. Rassam, and A. Zainal.

[3] J. Silva, T. Camilo, A. Rodrigues, M. Silva, F. Gaudêncio, and F. Boavida, "Multicast in wireless sensor networks the next step," in *Proceedings of the 2nd International Symposium on Wireless Pervasive Computing (ISWPC '07)*, pp. 185–190, February 2007.

[4] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, pp. 90–100, New Orleans, La, USA, February 1999.

[5] A. R. Swain and R. C. Hansdah, "A weighted average-based external clock synchronisation protocol for wireless sensor networks," *International Journal of Sensor Networks*, vol. 12, no. 2, pp. 89–105, 2012.

[6] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating congestion in wireless sensor networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pp. 134–147, ACM, Baltimore, Md, USA, November 2004.

[7] G. Chakraborty, "Genetic algorithm to solve optimum TDMA transmission schedule in broadcast packet radio networks," *IEEE Transactions on Communications*, vol. 52, no. 5, pp. 765–777, 2004.

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

[9] S. Ramanathan and E. L. Lloyd, "Scheduling algorithms for multihop radio networks," *IEEE/ACM Transactions on Networking*, vol. 1, no. 2, pp. 166–177, 1993.

[10] L. Bao and 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)*, pp. 210–221, ACM, 2001.

[11] T. Moscibroda and R. Wattenhofer, "Coloring unstructured radio networks," in *Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '05)*, pp. 39–48, ACM, July 2005.

[12] I. Rhee, A. Warrier, J. Min, and L. Xu, "DRAND: distributed randomized TDMA scheduling for wireless ad-hoc networks," in *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '06)*, pp. 190–201, ACM, May 2006.

[13] Y. Wang and I. Henning, "A deterministic distributed TDMA scheduling algorithm for wireless sensor networks," in *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '07)*, pp. 2759–2762, Shanghai, China, September 2007.

[14] C. Zhu and M. S. Corson, "A five-phase reservation protocol (FPRP) for mobile ad hoc networks," in *Proceedings of the IEEE 17th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '98)*, vol. 1, pp. 322–331, IEEE, San Francisco, Calif, USA, March–April 1998.

[15] A. R. Swain, R. C. Hansdah, and V. K. Chouhan, "An energy aware routing protocol with sleep scheduling for wireless sensor networks," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA '10)*, pp. 933–940, IEEE, Perth, Australia, April 2010.

[16] E. Arikan, "Some complexity results about packet radio networks," *IEEE Transactions on Information Theory*, vol. 30, no. 4, pp. 681–685, 1984.

[17] S. Ramanathan, "A unified framework and algorithm for (T/F/C)DMA channel assignment in wireless networks," *Proceedings the 16th IEEE Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '97)*, vol. 2, pp. 900–907, 1997.

[18] S. C. Ergen and P. Varaiya, "TDMA scheduling algorithms for wireless sensor networks," *Wireless Networks*, vol. 16, no. 4, pp. 985–997, 2010.

[19] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic aware scheduling algorithm for reliable low-power multi-hop IEEE 802.15.4e networks," in *Proceedings of the IEEE 23rd International Symposium on Personal, Indoor*

and Mobile Radio Communications (PIMRC '12), pp. 327–332, September 2012.

[20] S. Waharte and R. Boutaba, "Performance comparison of distributed frequency assignment algorithms for wireless sensor networks," in *Network Control and Engineering for QoS, Security and Mobility, III*, vol. 165, pp. 151–162, Springer, New York, NY, USA, 2005.

[21] O. Younis and S. Fahmy, "HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," *IEEE Transactions on Mobile Computing*, vol. 3, no. 4, pp. 366–379, 2004.

[22] R. Rozovsky and P. R. Kumar, "SEEDEX: a MAC protocol for ad hoc networks," in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '01)*, pp. 67–75, October 2001.

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

[24] C. Busch, M. Magdon-Ismail, F. Sivrikaya, and B. Yener, "Contention-free MAC protocols for wireless sensor networks," in *Distributed Computing: Proceedings of the 18th International Conference, DISC 2004, Amsterdam, The Netherlands, October 4–7, 2004*, vol. 3274 of *Lecture Notes in Computer Science*, pp. 245–259, Springer, Berlin, Germany, 2004.

[25] V. Gabale, B. Raman, P. Dutta, and S. Kalyanraman, "A classification framework for scheduling algorithms in wireless mesh networks," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 1, pp. 199–222, 2013.

[26] I. M. Niven, *Maxima and Minima without Calculus*, Mathematical Association of America, Washington, DC, USA, 1981.

[27] Castalia: A simulator for Wireless Sensor Networks, https://castalia.forge.nicta.com.au/index.php/en/documentation.html.

[28] CC2420 Data Sheet, http://www.stanford.edu/class/cs244e/papers/cc2420.pdf.

[29] Riemann sum, http://en.wikipedia.org/wiki/Riemannsum.

[30] B. Eisenberg, "On the expectation of the maximum of IID geometric random variables," *Statistics & Probability Letters*, vol. 78, no. 2, pp. 135–143, 2008.