A distributed TDMA slot scheduling algorithm for spatially correlated contention in WSNs

Ashutosh Bhatia and R. C. Hansdah
Dept. of Computer Science and Automation
Indian Institute of Science, Bangalore
Email: {ashutosh.b, hansdah}@csa.iisc.ernet.in

Abstract—In wireless sensor network (WSN) the communication traffic is often time and space correlated, where multiple nodes in a proximity start transmitting at the same time. Such a situation is known as spatially correlated contention. The random access methods 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 persist only for a short duration and therefore generating an optimal or sub-optimal schedule is not very useful. On the other hand, if the algorithm takes very large time to schedule, it will not only introduce additional delay in the data transfer but also consume more energy. To efficiently handle the spatially correlated contention in WSNs, we present a distributed TDMA slot scheduling algorithm, called DTSS algorithm. The DTSS algorithm is designed with the primary objective of reducing the time required to perform scheduling while restricting the schedule length to maximum degree of interference graph. The algorithm uses randomized TDMA channel access as the mechanism to transmit protocol messages, which bounds the message delay and therefore reduces the time required to get a feasible schedule. In DTSS algorithm, a node is required to know only the ids of its receivers instead of ids of all the two hop neighbors. Finally, DTSS algorithm supports unicast, multicast and broadcast scheduling, simultaneously without any modification in the protocol. The protocol has been simulated using Castalia network simulator to evaluate the run time performance. Simulation results show that our protocol is able to considerably reduce the time required to schedule.

Index Terms—TDMA Scheduling; Media Access Control; Wireless Sensor Networks

I. 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. Monitoring of events is carried out by sensing environmental parameters such as vibration, temperature, pressure, sound etc. 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, i.e., all the nodes in the same proximity transmit at the same time. Such a situation is known as spatially correlated contention. For example, 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 earthquake monitoring or intrusion detection 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. Another example of spatially correlated contention is broadcast communication in WSNs, where multiple sensor nodes simultaneously forward the received frame in order to spread the broadcast information over the network. Many tasks in WSNs depend on broadcasting, including critical tasks like querying. Consequently, reliable broadcast communication over sensor networks has become an important research challenge. Finally, protocols like route discovery [1], clock synchronization [2] and construction of data collection tree [3] in WSN, are also the examples of spatially correlated contention. Thus, it is desirable that the MAC protocols for WSNs should effectively handle the correlated contention.

MAC protocols for WSNs can be mainly classified into two major categories, viz., 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 [4]. A previous study [5] has reported that successful packet delivery ratio (PDR) in an 802.15.4 network can drop from 95 to 55 percent as the load increases from 1 to 10 packets/sec. Contention causes message collisions, which are very likely to occur when traffic is spatially correlated. This, in turn, would degrades the data transmission reliability and waste the energy of sensor nodes.

A MAC protocol is contention-free if it does not allow any collisions. A contention-free MAC is achieved by defining a schedule for data transmission to avoid collisions. It uses topology information as a basis for scheduling and this topology information comes in the form of neighborhood and interference relations among the nodes. 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 [6]–[8] 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 very large size networks, many distributed...
slot assignment schemes [9]–[13] 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 nodes have data to send all the time. In WSNs, with spatially correlated contention, nodes occasionally require a slot to transmit a sequence of data and leave the slot once the data transfer is over. Therefore, the effective benefit of reducing schedule length vanishes. Moreover, if the algorithm takes very large time to schedule, it will not only introduce additional delay in the data transfer but also consume more energy of the sensor nodes. The preceding discussion emphasizes that in order to effectively handle the spatially correlated contention, the MAC protocols in WSNs should be distributed, contention-free and 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, where \( \delta \) is the network density in terms of the size of two hop neighborhood. The proposed algorithm is unified in the sense that the same algorithm can be used to schedule slots for different modes of communication, viz., 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 impose any ordering among the nodes.

The rest of the paper is organized as follows. Section II discusses the related work. Section III gives the assumptions we make in the design of our algorithm, introduces some definitions, and explains the basic idea of our algorithm. In section IV, we present a detailed description of the DTSS algorithm. Section V gives the proof of correctness of the DTSS algorithm. Section VI discusses the simulation results and performance comparison of DTSS algorithm with existing work. Section VII concludes the paper with suggestions for future work.

II. RELATED WORK

The optimized broadcast scheduling problems is NP-complete [14]. A different, but related problem to TDMA node slot assignment, is the problem of TDMA edge slot assignment, where 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 [15]. In [16], another specific scheduling problem for wireless sensor network converge-cast transmission is considered where the scheduling problem is to find a minimum length frame during which all nodes can send their packets to access point (AP), and is shown to be NP-complete. Previous work [6]–[8] on scheduling algorithms primarily focus on decreasing the length of schedules. They are centralized in nature and normally needs complete topology information and are, therefore not scalable.

Cluster-based TDMA protocols in [17], [18] 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 inter cluster transmission interference because, clusters created by distributed clustering algorithm are often overlapped and several cluster heads may cover the same nodes.

Moscibroda et al. [9] have proposed a distributed graph coloring scheme with a time complexity of \( O(\rho \log n n^2) \), where \( \rho \) is the maximum node degree 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 [10] proposed a distributed scheduling based on hash function to determine 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 [19] 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 [11], proposes a distributed randomized time slot scheduling algorithm based on centralized scheduling scheme RAND [6]. DRAND is also used within a MAC protocol called Zebra-MAC [20] to improve performance in sensor networks by combining the strength of scheduled access during high load and random access during low loads. The DRAND algorithm includes two major phases: Neighbor Discovery and DRAND Slot Assignment. Nodes exchange control messages like Request, Grant, Release, or Reject to determine the time slots of sensors. With this scheme, the runtime complexity is
O(n), where n is the maximum size of a two-hop neighborhood in a wireless network. The simulation results presented by the author show that the run time actually becomes \( O(n^2) \) due to unbounded message delays.

FPRP [12] 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 a pair of reservation and data transmission phases. During the reservation phase, the protocol assigns the slots of the next data transmission phase to the nodes who have data to send. Thus, the actual data transmission occurs using the slots assigned in the previous reservation phase. 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 decide j as its own slot if all the nodes with id less than id of node i has already decided their slot, where j is the minimum available slot. The scheduled node broadcasts its slot assignment to one-hop neighbors. Then those one-hop neighbors broadcast this information to update two-hop neighbors. This is repeated in every frame until all nodes are scheduled.

These scheduling algorithms [11]–[13] commonly have following issues.

- 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 run time of these protocols they are more suitable for a wireless network where interference relationship or network topology does not change for a long period of time.
- They perform two hop neighbor discovery which adds considerable additional cost to run time to perform scheduling. Additionally, the two hop neighbors are calculated based on transmission range instead of interference range which is normally higher that the transmission range.
- They either perform broadcasting (node scheduling) or unicasting (link scheduling) but not both and also do not consider multicast scheduling separately.

### III. OUR APPROACH TO TDMA SCHEDULING IN WSNs

In many applications such as whether monitoring, intrusion detection etc., 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 a WSN has a unique identifier. All nodes in the WSN have some processing capability along with a radio to enable the 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, i.e., only one node at a time can transmit. The transmission is omnidirectional, i.e., each message sent by 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, if node i and any node j other than i, transmit at the same slot, then no node in \( N_i \) can successfully receive the message transmitted by node j. Moreover, if only node i has transmitted in the slot, then a node in \( N_i \) may or may not receive the message.

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, viz., 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 intended senders of node i.

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 \( N_i \cap R_j \neq \phi \) \( \lor \) \( N_j \cap R_i \neq \phi \). 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 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 the communication, viz., unicast, multicast or broadcast. Two nodes within the interference range of each other can even take the same slot, if their transmissions do not interfere at each others 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 cannot transmit simultaneously, if their transmissions interfere at each others 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.

![Diagram](image1)

**Fig. 1:** i and j do not conflict even \( i \in N_j \) and \( j \in N_i \)

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

**Strong-Conflict Relation:** Two nodes i and j have strong-conflict relation if \( N_i \cap R_j \neq \emptyset \) and \( N_j \cap R_i \neq \emptyset \).

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

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

**Definition 5:** The **Interference degree** \( \Delta \) 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, \( \Delta \) and \( \delta \) are roughly the same, where \( \delta \) 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.

**IV. THE DTSS ALGORITHM**

In this section, we describe the proposed DTSS algorithm for TDMA slot scheduling problem as defined in section II. The number of slots, \( \text{numSlots} \) in a frame is taken to be at least \( \Delta \). The slots are numbered from 1 to \( \text{numSlots} \). Each node i maintains the following data structures and variables to implement the algorithm.

- \( R_i \): 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.
- \( L_1 \): A list of slots which are already taken by the nodes adjacent to i in G.
- \( L_2 \): A list of slots taken by the nodes in \( S_i \). Note that \( L_2 \subseteq L_1 \).
- \( L_3 \): A list of slots which are currently blocked by the nodes adjacent to i in G.
- \( \text{numSlots} \): 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 \( R_i \).
- \( \text{nr} \): Remaining number of receivers from where responses needs 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.
In addition to the above data structures, 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 message contains four fields, viz., 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 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 \( 1/\text{numSlots} - |L1| \). 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 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 \( \Delta \) 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 to the list L3. If a node i in CS state receives a REQ

```
Algorithm 1 DTSS Algorithm

if i.slot = i.b_slot = null and s \notin \{L3 or L1\} then
    send REQ(i.rx_id, L2, |R_i|) with probability P(s)
else
    send REQ(i.rx_id, L2, nr)
end if

if rx_id = R_i \rightarrow next
//perform channel listening
if i receives a REQ(j,dest_id, L2, state) then
    if REQ.dest_id = i then
        add j in S_i
        send RES(i,j, L2, REQ.state)
    end if
if j \in R_i and REQ.state = 0 then
    slot is gone, add s to L1
if j \in S_i then
    add s to L2
end if
end if
if j \in S_i or j \in R_i and REQ.state \neq 0 then
    slot s is blocked, add s to L3
end if
end if

if i receives a RES(j,rx_id, L2, state) then
    if RES.rx_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 rx_id, add s to L1
    if RES.rx_id \in S_i then
        add s to L2
    end if
else
    slot s is blocked, add s to L3
end if
end if
end if
```

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 to 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 \(L_1\) and also updates the slot-probability of other slots not in \(L_1\) as \(1/(num\text{Commands} - |L_1|)\). Additionally, if \(j \in S_i\), then it also adds the slot \(s\) in the list \(L_2\). 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, 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 \(L_3\). A node does not transmit its own REQ messages in the slots belonging to \(L_3\) 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 SS 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 \(L_1\) and also updates the slot-probability of other slots not in \(L_1\) as \(1/(num\text{Commands} - |L_1|)\). Node \(i\) permanently leaves the slots in \(L_1\) 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 \(L_2\).

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 \(L_2\) 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 node in \(R_i\), 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 2b. Although, 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 \(\Delta\) times, it leaves the slot by adding it to the list \(L_1\) and come 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 \(\Delta\) times. This ensures that node \(i\) will only leave the slot \(s\), if \(j \rightarrow i\) and \(j\) is in SS state.

V. 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 theorem 1 and 2 for strong-conflict and weak-conflict relationship respectively.

**Theorem 1:** 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 \(frame_{j}^{VS}\) be the frame indexes when node \(i\) enters VS and SS state respectively for a time slot \(k\). It is possible that \(frame_{i}^{VS} = frame_{j}^{VS}\) if \(|R_i| = 1\). Furthermore, let \(frame_{i}^{VS}\) and \(frame_{j}^{VS}\) 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 SS 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. Now the following three cases arise.

**Case1:** \(frame_{i}^{VS} < frame_{j}^{VS}\): In this case, only node \(i\) can enter SS state provided it has got response from all nodes \(k \in N_i \cap R_i\) 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 \(N_i \cap R_i\). Hence, in this case only node \(i\) will be able to enter SS state.

**Case2:** \(frame_{i}^{VS} = frame_{j}^{VS}\) : 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 a case 2.
Case3: $frame^{VS}_i > frame^{VS}_j$: 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 node $i$ or $j$ will be in SS state for same time slot during the execution of the DTSS algorithm. Hence, the theorem is proved.

Theorem 2: 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 $frame^{VS}_i$, $frame^{SS}_i$, $frame^{VS}_j$ and $frame^{SS}_j$ be frames indexes of node $i$ and $j$ as in theorem 1. Also, without loss of generality assume that $N_i \cap R_j \neq \emptyset$, and $N_j \cap R_i = \emptyset$. That is node $i$ is stronger than node $j$. As in theorem 1, 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 same frame index without at least one of them going into VS state. In this case also, the following three cases arise.

Case1: $frame^{VS}_i < frame^{VS}_j$: This case is similar to case 1 of theorem 1, and only node $i$ will be able to enter SS state, and node $j$ will not be able to enter SS state.

Case2: $frame^{VS}_i = frame^{VS}_j$: 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.

Case3: $frame^{VS}_i > frame^{VS}_j$: 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 frame index $frame^{VS}_i$. Let us assume that node $j$ satisfies this condition. Now node $i$ 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 $\Delta$ 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.

VI. SIMULATION RESULTS

We have used Castalia Simulator [21] to study the performance of DTSS algorithm. A multihop network, based on TelosB node hardware platform that uses CC2420 transceiver [22] for communication is used. The transceivers run at 250kbps data rate and 0dbm transmission power which approximately gives 40m of transmission range when there is no interference. All nodes are distributed randomly within 250mX250m area. The lognormal shadowing channel model is used to get realistic interference and path loss at different receivers. The slot size is taken as 1 ms so that transmission of REQ/RES message can be completed within a slot. Note that the Mica2 radio (CC2420 [22]) takes about .5 ms to transmit a packet of size 128 bits (80 bits for the MAC header, and 48 bits for L2 and state payload, at the data rate of 250 kbps. 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 have been of protocol, 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, $\delta$ value varying between 5 and 50.

Figure 4 shows the average number of slots taken by all the nodes to decide their slot in case of broadcast scheduling for frame size $\delta$ and 1.3$\delta$ respectively. The error bars denote 95% confidence intervals. The Figure 4 shows that runtime increases linearly with neighborhood density, $\delta$. Given slot size as 1ms, the total runtime for very high density network with $\delta = numSlots = 50$, is approximately 7s. Furthermore, if we take more slots per frame, then runtime decreases and also confidence interval improves. Figure 5 shows the average of the number of slots taken by all the nodes to decide their slot in case of broadcast scheduling for varying numSlots values starting from $\delta$. The Figure 5 shows that the run time reduces rapidly with small increase in numSlots and further increase in $\Delta$ does not have much impact on run time. This fact can be utilized as a trade off between runtime and frame length. Figure 6 shows the average of the number of slots taken by all the nodes to decide their slot for varying number of receivers (unicast to broadcast) with $\delta = numSlots = 40$. The Figure 6 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 [11] and DD-TDMA [13]. Figure 7 shows the performance results of DTSS along
The number of slots taken by DTSS is always less than the number of time slots taken by DRAND and DD-TDMA can be bounded by \( \Delta \), further efforts can be applied to reduce the number of slots. Although, the number of slots available are already fixed, whereas other algorithms try to generate a sub-optimal schedule using greedy approach, which is inherently sequential. In case of unicast and multicast the DTSS even takes lesser time to compute the schedule as compared to broadcast transmission. The number of slots taken by DTSS is always \( \delta \), whereas the number of time slots taken by DRAND and DD-TDMA can be less than \( \delta \).

VII. 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 as compared to other existing distributed scheduling algorithms. The interference model used by DTSS is more realistic than conventional protocol interference model. Additionally, the DTSS has a unique feature of uniform scheduling where 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 \( \Delta \), further efforts can be applied to reduce the number of slots.

REFERENCES


