
Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
 Volume 21, Issue 2, pp. 233–245, DOI 10.12694:/scpe.v21i2.1672

AN EFFICIENT DYNAMIC SLOT SCHEDULING ALGORITHM FOR WSN MAC: A DISTRIBUTED APPROACH

MANAS RANJAN LENKA* AND AMULYA RATNA SWAIN†

Abstract. In the current scenario, the growth of IoT based solutions gives rise to the rapid utilisation of WSN. With energy constraint sensor nodes in WSN, the design of energy efficient MAC protocol along with timeliness requirement to handle collision is of paramount importance. Most of the MAC protocols designed for a sensor network follows either contention or scheduled based approach. Contention based approach adapts well to topology changes, whereas it is more costly in handling collision as compared to a schedule based approach. Hence, to reduce the collision along with timeliness, an effective TDMA based slot scheduling algorithm needs to be designed. In this paper, we propose a TDMA based algorithm named DYSS that meets both the timeliness and energy efficiency in handling the collision. This algorithm finds an effective way of preparing the initial schedule by using the average two-hop neighbors count. Finally, the remaining un-allotted nodes are dynamically assigned to slots using a novel approach. The efficiency of the algorithm is evaluated in terms of the number of slots allotted and time elapsed to construct the schedule using the Castalia simulator.

Key words: WSN, TDMA Slot Scheduling, Correlated Contention, Data Delivery Latency, Feasible Schedule.

Abbreviations. Internet of Things (IoT), Wireless Sensor Network (WSN), Media Access Control (MAC), Time Division Multiple Access (TDMA), Carrier Sense Multiple Access (CSMA), Randomized TDMA (RAND), Distributed RAND (DRAND), Randomized Distributed TDMA (RD-TDMA), Hybrid based Distributed Slot Scheduling (HDSS), and Dynamic Slot Scheduling (DYSS).

AMS subject classifications. 68M14

1. Introduction.

In the real world, the widespread use of WSN mainly owes to monitor and control various devices in several industrial and home automation systems. A sensor network is unique in the sense that the nodes are battery powered and mostly can not be recharged from time to time. Hence, every application in a WSN environment must be designed in such a way that energy consumption must be minimal. Along with energy efficiency, other requirements like timeliness, collision handling, and reduced latency during data transmission also need to be considered. To handle collision along with the above requirements, the design of an efficient MAC protocol is of paramount importance. Studies carried out by Ergen et al. [1] revealed that TDMA base MAC protocols perform better than CSMA based MAC protocols in a WSN to satisfy these stringent requirements [14, 16, 19, 20, 21, 22].

In TDMA based MAC protocols [15, 17, 18, 24, 25], a schedule with several time slots is prepared where each sensor node is assigned to a particular time slot. The assignment of slots to each sensor node is carried out in such a manner that collision is handled with reduced data delivery latency. Further, to minimise energy consumption, the nodes are put to sleep for it's allocated time slot as proposed in [2, 3, 26]. The delivery latency is reduced through proper scheduling of the TDMA slots as proposed by Moriyama et al. [4], Ahmad et al. [5], and Ahmad et al. [6]. The collision during data transmission is handled in such a way that the nodes interfering with each other are not assigned to the same slot. Most of the TDMA based slot scheduling algorithms prepare an optimal schedule that normally takes more time to prepare a schedule. However, in the case of correlated contention, preparing an optimal schedule may not be of much use as the contention exists for a short duration. Hence, a feasible schedule needs to be prepared in a quick time to handle the correlated contention, as proposed by Lenka et al. [7], Lenka et al. [8], and Bhatia et al. [9].

*KIIT Deemed to be University, Bhubaneswar, India (manasy2k3@gmail.com).
†KIIT Deemed to be University, Bhubaneswar, India (swainamulya@gmail.com).

Preparing a feasible schedule in a quick time may lead to latency during data transmission. Therefore, design of a scheduling algorithm has to take care of both correlated contention and the latency during data transmission to enhance efficiency. Based on the above facts, this paper proposed an efficient TDMA based dynamic slot scheduling algorithm that handles both correlated contention and delivery latency. The proposed algorithm uses the average two-hop neighbors count as opposed to the maximum two-hop neighbors count used in earlier approaches. The maximum two-hop neighbors count varies significantly from the average two-hop neighbors count in most of the sensor networks due to the random deployment of the sensor nodes. As a result, the use of average two-hop neighbors count helps to reduce the number of slots to be attached to a schedule. However, the use of average two-hop neighbors count to prepare a schedule may leave a few nodes to remain un-allotted. The remaining un-allotted nodes are then dynamically assigned to the best possible slots using a novel message passing technique in a quick time to prepare the final schedule. Finally, the proposed algorithm is implemented using Castalia simulator to evaluate its performance. The performance analysis shows that our proposed scheme outperforms DRAND, RD-TDMA, and HDSS in terms of number of slots and time to allocate the slots in a schedule.

The rest of the paper is structured as follows. A review of the related works has been summarised in Section 2. Section 3 describes our proposed algorithm. The correctness of the algorithm has been analysed through various scenarios in Section 4. The simulation results are analysed in Section 5. The conclusion is drawn in Section 6.

2. Related Work. In today's world, the recent trend is to automate everything for living a relaxed and comfortable lifestyle. WSN plays a vital role in achieving the same. In a resource constraint sensor network, the design and development of an efficient application has to deal with several challenges such as energy efficiency, collision handling, reduction in latency, reliability, etc. In this paper, we mainly focus on TDMA based slot scheduling approach that helps to handle the collision and reduce the latency during data transmission.

Ahmad et al. [6] proposed a centralised TDMA scheduling for clustered based tree topology in WSN. This algorithm prepares a collision-free clustered based schedule to satisfy the timeliness for several data flows. The timeline for each data flow is expressed based on the length of the schedule period. The algorithm takes care of minimum utilisation of energy by putting the nodes into low power mode to the maximum possible extent. As opposed to the centralised approach, Ahmad et al. [10] proposed a distributed version of the clustered based TDMA algorithm, where each cluster is capable enough to prepare its time slot for the TDMA schedule. This distributed nature of the algorithm aid to the WSN as the resources in the WSN environment is scarce.

Severino et al. [11] proposed a self-adaptive clustered based dynamic scheduling algorithm for WSN. Based on the nature of the traffic flow, this algorithm adapts different required bandwidth and latency by changing the scheduling algorithm attached to the clusters. This adaption happens in quick time by exerting a small downtime for the WSN.

Wang et al. [18] proposed a deterministic TDMA based slot scheduling approach to avoid collision during data transmission. In this algorithm, each sensor node calculate it's own time slot in a distributed manner as per the available neighborhood information. However, each sensor node need synchronisation among each other to calculate their own time slot and as a result the collision handling model was not too realistic.

Long et al. [12] proposed a multi-hop TDMA scheduling algorithm for WSN that extends the one-hop TDMA scheduling to multi-hop scheduling. This extension helps in balancing the energy consumption among the nodes in a WSN which ultimately prolongs the network lifetime.

Rhee et al. [13] proposed a distributed version of RAND, a centralised random slot scheduling algorithm. In this algorithm, each node presents in one of the four states, i.e. REQUEST, RELEASE, IDLE, and GRANT. Each node starts with IDLE state and goes for a lottery. The node that wins the lottery sends a request message for allotment of a slot and enters into the REQUEST state. The node that receives this request message enters into GRANT state provided the node is in IDLE or RELEASE state. In case, the receiver node is either in REQUEST or GRANT state while receiving this request message then it sends back a reject message to the sender. When the sender node receives a reject message, it goes back to IDLE state. Once a node, who has started the slot allotment request, receives a grant message from all of its neighbors then it enters into the RELEASE state and the requested slot is allocated to that node.

Li et al. [23] proposed a distributed TDMA slot scheduling that improves upon the DRAND algorithm.

This algorithm prepares the TDMA schedule based on the residual energy and topology associated with a sensor network. In order to reduce the energy consumption and execution time of a schedule, initially it defines the energy-topology factor and then applies the same to arrange the priority of the time of a slot in a schedule to deal with the energy consumption and execution time.

Most of the slot scheduling algorithms focus on preparing an optimal schedule to handle the collision during data transmission. In case of co-related contention (i.e. the collision that occurs at the receiver end for a very short period), preparing an optimal schedule may not help a lot. Keeping the above requirement in mind Bhatia et al. [9] proposed a feasible slot scheduling algorithm named RD-TDMA that handles the co-related contention and at the same time reduces the latency during data transmission.

Lenka et al. [8] proposed a HDSS algorithm that initially prepares a feasible schedule based on maximum two-hop neighbors count using the DRAND algorithm. The number of slots attached to the prepared feasible schedule is further reduced by reallocating certain nodes based on the ratio of average two-hop neighbors count to the maximum two-hop neighbors count. Finally, a sub-optimal schedule, with less number of slots as compared to DRAND and RD-TDMA, is prepared in quick time which ultimately handles the collision due to co-related contention and at the same time reduces the latency during data transmission.

Although the existing HDSS algorithm performs better as compared to DRAND and RD-TDMA with respect to the number of slots attached to a feasible schedule, still this paper finds a way to further fine-tuned the slot scheduling approach in a novel way to reduce the number of slots. The proposed approach focuses on preparing a feasible schedule based on the average two-hop neighbors count instead of maximum two-hop neighbors count as used in HDSS. Moreover, the proposed approach uses a novel dynamic slot scheduling technique to allot slots for the remaining nodes in the best possible way to achieve the desired goal.

3. Proposed Dynamic Slot Scheduling Algorithm. In the earlier proposed HDSS algorithm for WSN MAC, a feasible schedule is prepared based on the maximum two-hop neighbors count. Each sensor node in WSN is assigned to a particular slot in the feasible schedule and all its two-hop neighbors are assigned to different slots so that the collision during data transmission will be avoided. Nevertheless, in the real scenario, during the deployment of the sensor nodes, there is every chance that some of the regions of WSN may have very high node density as compared to other regions. In such scenarios, there will be a significant difference between the maximum two-hop neighbors and the average two-hop neighbors count due to the presence of the outliers (i.e. specific regions with high node density). Therefore, preparing a feasible schedule based on the maximum two-hop neighbors count leads to a significant rise in the number of slots required for the same. In order to get rid of the above issue, in the proposed approach, the schedule is initially prepared based on the average two-hop neighbors instead of maximum two-hop neighbors count.

To start with, in the proposed approach a feasible schedule is initially prepared based on the average two-hop neighbors count. As per the earlier discussion, since there is more possibility of higher density in some regions of WSN, the actual two-hop neighbors count in those regions will be more than the average two-hop neighbors count in the whole network. Hence, there is every possibility that some sensor nodes may remain un-allotted due to the lack of availability of slots as the feasible schedule is initially prepared based on the average two-hop neighbors count. The nodes which remain un-allotted during the preparation of the feasible schedule, those nodes will be allotted to a feasible slot through these following three phases:

1. Slot Allotment Request
2. Slot Allotment Message Handler
3. Slot Allotment Status

The notations used for the set of information maintained at each node for slot allocation in the proposed algorithm is summarises in Table 3.1.

3.1. Phase 1: Slot Allotment Request. Let N_{Avg} is the average two-hop neighbors count calculated during the preparation of feasible schedule at the very beginning and a node i has not been allotted to a feasible slot during the preparation of feasible schedule. In order to get a feasible slot, the node i selects a slot k, where $k = N_{Avg} + 1$ and verifies at its end whether this slot k has already been allotted to any one of it's one-hop or two-hop neighbor nodes. In case, the slot k has already been allotted then it tries with the next slot, i.e. $k + 1$ and so on till the chosen slot is fit for him. Once the node i finds a feasible slot say k then it stores the information, i.e. $Orig_{(i,k)}, Sac_{(j,k)}, HopCnt_{(i,k)}, SlotReqTime_{(i,k)}$ into a vector V at it's own end. Finally, node

TABLE 3.1
Set of information maintained at each node for slot allocation of the remaining nodes.

Notation	Description
`Orig_{(i,k)}`	Node i, the originator of dynamic allotment procedure for slot k
`Sac_{(j,k)}`	Intermediate node j, who sacrificed the slot k for another node
`HopCnt_{(i,k)}`	Hop distance at node i for slot k
`SlotReq`	Slot requested for Allotment
`SlotReqTime_{(i,k)}`	Time at which the request for allotment of slot k has been initiated by node i
`V`	A Vector of size S containing $Orig_{(i,k)}$,$Sac_{(j,k)}$, $HopCnt_{(i,k)}$, $SlotReqTime_{(i,k)}$

i broadcasts a '***DynamicSlotAllocation***' message that contains the stored information to all its neighbors. The slot allotment request of i^{th} node is given in Algorithm 5.

Algorithm 5: Slot allotment request for remaining nodes at node i

```
1 Procedure SLOT_ALLOTMENT_REQUEST
2 begin
3     k = N_Avg + 1 ;
4     while (k already allotted to one of it's one-hop or two-hop neighbors) do
5         k = k + 1 ;
6     HopCnt_(i,k) = 0 ;
7     Store (Orig_(i,k), Sac_(j,k), HopCnt_(i,k), SlotReqTime_(i,k)) into vector v ;
8     Send(DynamicSlotAllocation(Orig_(i,k), Sac_(j,k), HopCnt_(i,k), SlotReqTime_(i,k))) to all
      neighbors ;
```

3.2. Phase 2: Slot Allotment Message Handler. After receiving the 'DynamicSlotAllocation' message from node i, a node j checks whether there exist an entry in the stored vector V at it's end for the requested slot k and goes through these following steps.

[Step 1:] In case the entry related to the requested slot k does not exist then the received information is pushed into the vector V at it's end but with a updated value for the hop count, i.e. $HopCnt_{(j,k)} = HopCnt_{(i,k)} + 1$. Then the j^{th} node broadcasts the updated received information to all of it's neighbors provided the $HopCnt_{(j,k)}$ is less than 2.

[Step 2:] If the entry related to the requested slot k exist then increment the received $HopCnt_{(i,k)}$ by one and check whether the updated received $HopCnt_{(i,k)}$ is less than the stored $HopCnt_{(j,k)}$ for the requested slot k.

[Step 3:] If the above condition holds good then update the existing entry for the slot k with the received information and broadcast the same to its neighbors.

[Step 4:] If the above condition does not hold then it again checks whether the received $SlotReqTime_{(i,k)}$ is less than the stored $SlotReqTime_{(j,k)}$ for the slot k. If it is found true then it checks for whether the receiver node j is the originator of slot allotment procedure for the same slot k or not. In case, the receiver node j is the originator of slot allotment procedure for the slot k then update the entry in the vector V with the received information but with modified $Sac_{(j,k)}$ value.

[Step 5:] If the receiver node j is not the originator of slot allotment process for the slot k then it checks whether $Sac_{(i,k)}$ information in the received message is either same as the information stored at the originator or the sacrificed intermediate node. If it matches then the existing entry for the slot k is updated with the received information but with increment the value of the $HopCnt_{(i,k)}$ by one and broadcast the same to it's neighbors provided the incremented $HopCnt_{(j,k)}$ value is less than 3.

Algorithm 6: Slot allotment process after receive of slot allotment request message at node i

```
1  Procedure SLOT_ALLOTMENT_MESSAGE_HANDLER
2  begin
3      msg = Receive_Message();
4      if (msg == DynamicSlotAllocation(Orig_(i,k), Sac_(j,k), HopCnt_(i,k), SlotReqTime_(i,k))) then
5          for m ← 0 to S − 1 do
6              if (V_[m].SlotReq == k) then
7                  break;
8          if ((S == 0) || (requested slot k ∉ V)) then
9              HopCnt_(i,k) = HopCnt_(i,k) + 1 ;
10             push the updated received information into the vector v ;
11             if (HopCnt_(i,k) < 2) then
12                 Send(DynamicSlotAllocation(Orig_(i,k), Sac_(j,k), HopCnt_(i,k), SlotReqTime_(i,k))) to
                    all neighbors ;
13         else
14             HopCnt_(i,k) = HopCnt_(i,k) + 1 ;
15             if (HopCnt_(i,k) < V_[m].HopCnt_(n,k)) then
16                 func1();
17             else
18                 if (SlotReqTime_(i,k) < V_[m].SlotReqTime_(n,k)) then
19                     if (V_[m].HopCnt_(n,k) == 0) then
20                         func2();
21                     else
22                         if (Sac_(j,k) == V_[m].Orig_(i,k)) || (Sac_(j,k) == V_[m].Sac_(j,k)) then
23                             if (HopCnt_(i,k) < 2) then
24                                 func3();
25                             else
26                                 remove the entry at index m from the vector v ;
```

```
1  Procedure func1()
2  begin
3      HopCnt_(i,k) = HopCnt_(i,k) + 1 ;
4      Update the existing entry in vector V for slot k with the received information but modified
        HopCnt_(i,k) value ;
5      if (HopCnt_(i,k) < 2) then
6          Send(DynamicSlotAllocation(Orig_(i,k), Sac_(j,k), HopCnt_(i,k), SlotReqTime_(i,k))) to all
            neighbors ;
```

3.3. Phase 3: Slot Allotment Status. After certain period of time, node i checks the content of vector V at its own end. In case the vector V contains an entry with a $HopCnt_{(i,k)} = 0$, then it indicates that the node i is the originator for the slot k and hence node i is allotted with the slot k. In case no entry with a $HopCnt_{(i,k)} = 0$ is found then try with the next slot.

```
1 Procedure func2()
2 begin
3     V_[m].Sac_(j,k) = n ;
4     HopCnt_(i,k) = HopCnt_(i,k) + 1 ;
5     Update the existing entry in vector V for slot k with the received information but modified
        HopCnt_(i,k) value and Sac_(j,k) node ;
6     Send(DynamicSlotAllocation(Orig_(i,k), Sac_(j,k), HopCnt_(i,k), SlotReqTime_(i,k))) to all
        neighbors ;
```

```
1 Procedure func3()
2 begin
3     HopCnt_(i,k) = HopCnt_(i,k) + 1 ;
4     Update the existing entry in vector V for slot k with the received information but modified
        HopCnt_(i,k) value ;
5     Send(DynamicSlotAllocation(Orig_(i,k), Sac_(j,k), HopCnt_(i,k), SlotReqTime_(i,k))) to all
        neighbors ;
```

Algorithm 7: Verification of slot allotment status at node i

```
1 Procedure SLOT_ALLOTMENT_STATUS
2 begin
3     for j ← 0 to S − 1 do
4         if (v_[j].HopCnt_(i,k) == 0) then
5             Allot the slot k to node i; Send(DynamicSlotAllotementSucceeded(i, k)) to all the nodes
                in the whole WSN ;
6             break;
7         if (j == S) then
8             Call SLOT_ALLOTMENT_REQUEST procedure for the next slot k + 1;
```

4. Proof of the proposed Algorithm. The correctness of the proposed algorithm is verified through various scenarios as described here.

The above steps for slot allotment are explained through an example given in Fig. 4.1. As per the scenario given in Fig. 4.1, node x_1, x_2, and x_7 have not been allotted to any feasible slot during the preparation of the initial schedule based on the average two-hop neighbors count. Here, x_1 and x_2 are the one-hop neighbors to each other whereas node x_7 is neither one-hop neighbor nor two-hop neighbor of node x_1 or x_2. Let us assume that the node x_1 first starts the slot allotment procedure for the slot k at time t_1 and stores the information such as originating node, sacrificed intermediate node, hop count, requested slot, and request time stamp as $(x_1, x_1, 0, k, t_1)$ in the vector V and broadcast this information to its neighbors. When the node x_2 receives the message from node x_1 and by that time if node x_2 has already started the slot allotment procedure for the slot k at time t_2 which is latter than t_1 then it stores information such as the originating node, sacrificed intermediate node, hop count, requested slot, and request timestamp as $(x_2, x_2, 0, k, t_2)$ in the vector V at its own end. As per the proposed algorithm, since the received timestamp t_1 is less than the stored timestamp t_2 at node x_2, so the stored information in the vector V at x_2 is updated to $(x_1, x_2, 1, k, t_1)$ and broadcast the updated information to it's neighbor. Here, the sacrificed intermediate node information remains same as x_2 as the node x_2 has sacrificed the slot k for node x_1 which starts the allocation procedure for slot k earlier than x_2. Hence, x_2 can not be allotted to slot k and further it has to try with the next available slot.

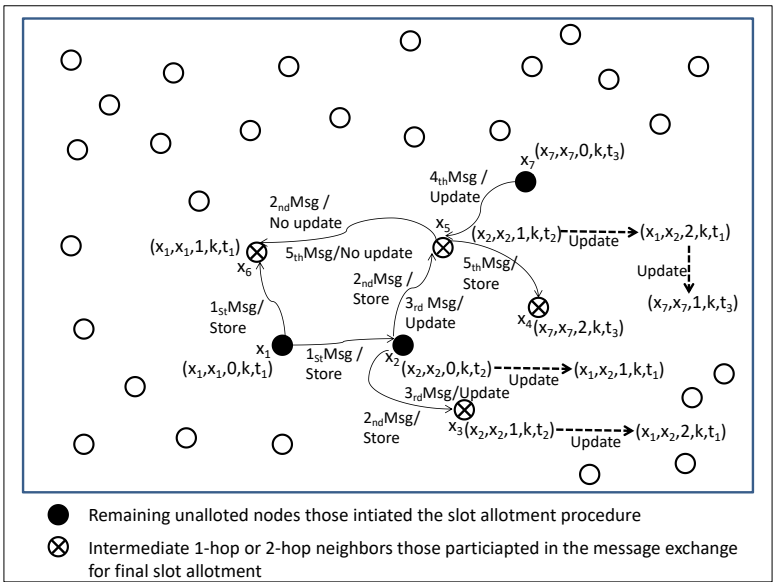

FIG. 4.1. *Slot Not Alloted To One-hop Neighbors*

At another instance of time t_3, which is later than t_2, node x_7 has started its slot allocation procedure for slot k by storing the information $(x_7, x_7, 0, k, t_3)$ into the vector V and broadcast this information to its neighbors. When a node x_5 received the message from node x_7, by that time the node x_5 has already stored the information $(x_1, x_2, 2, k, t_1)$ in its vector V. According to the proposed algorithm, as the incremented received hop count is less than the stored hop count at node x_5, the stored information in the vector V at node x_5 is updated to $(x_7, x_7, 1, k, t_3)$ and broadcast the updated information to its neighbor. Finally, node x_1 and x_7 are allotted to the same slot k as both these nodes are not neighbors to each other. Whereas, node x_2 is allotted to a slot other than k as it is a one-hop neighbor of x_1.

According to the scenario given in figure 4.2, node x_1 and x_5 have not been allotted to any slot during the preparation of the initial schedule based on the average two-hop neighbors count. Here, x_1 and x_5 are two-hop neighbors to each other. Assume that the node x_1 starts the slot allotment procedure first for the slot k at time t_1 and stores the information such as originating node, sacrificed intermediate node, hop count, requested slot, and request time stamp information as $(x_1, x_1, 0, k, t_1)$ in the vector V and broadcast this information to its neighbors. Now, when the node x_2 receives the message from node x_1 and by that time node x_2 has no stored information regarding the slot k in its vector V then the node x_2 stores the received information as $(x_1, x_1, 0, k, t_{21})$ in the vector V at its own end. In another instance at time t_2, which is later than t_1, the node x_5 has started its slot allocation procedure for slot k and stored the information $(x_5, x_5, 0, k, t_2)$ in vector V and broadcasted this information to its neighbors. Now, when the node x_2 receives the message from node x_5, as per our algorithm, neither the received hop count nor the received timestamp is lesser than the stored one. Hence, there is no update made at the node x_2. Now, when the node x_5 receives the message from node x_2 and as per our algorithm, since the received timestamp t_1 is less than the stored timestamp t_2 at the node x_5, so, the stored information in the vector V at node x_5 is updated to $(x_1, x_5, 2, k, t_1)$ and broadcast the updated information to its neighbors. Here, the sacrificed intermediate node information became x_5 as the node x_5 has sacrificed the slot k for node x_1 which starts the allocation procedure for slot k earlier than x_5. Hence, the node x_5 can not be allotted to slot k and try with the next slot. Now, when the node x_7 receives the message from node x_5 and as per our algorithm, if the received timestamp t_1 is less than the stored timestamp t_2 at node x_7 and the received sacrificed intermediate node information is same as stored originator node information then the stored information at node x_7 needs to be removed as node x_5 can no more be allotted to the slot k. Finally, based on our algorithm, the node x_1 is allotted to slot k and the node x_5 can not be allotted to the same slot k as both are the two-hop neighbors to each other. This proofs the correctness of our algorithm.

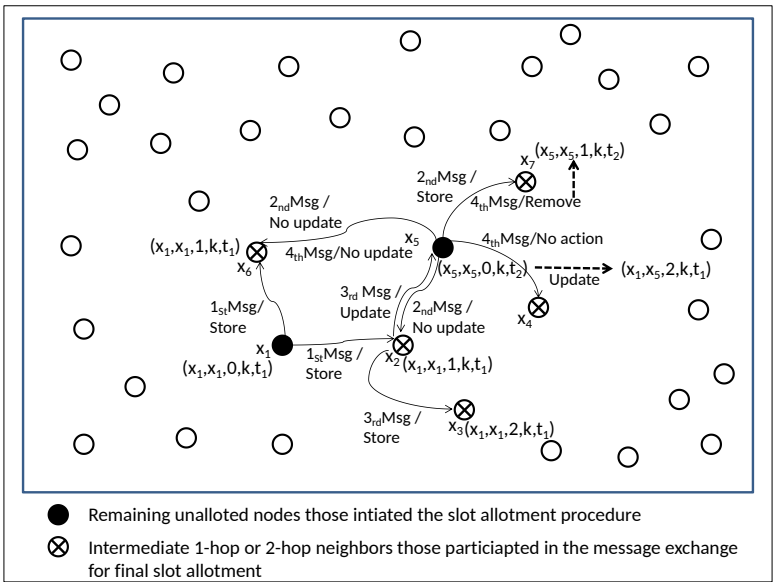

FIG. 4.2. *Slot Not Alloted To Two-hop Neighbors*

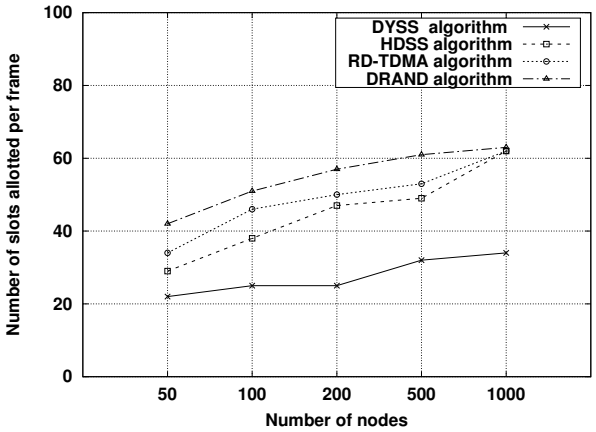

FIG. 5.1. *Number of slots attached to the feasible schedule among various algorithms using uniform random distribution*

5. Simulation Results. The simulation of the proposed dynamic slot allotment algorithm is carried out using the Castalia simulator[27]. The efficiency and correctness of the algorithm are tested in various deployment scenarios like uniform random distribution, grid, and randomised grid. The correctness of the algorithm is also carried out in fixed as well as varying node density with the number of sensor nodes varying from 50 to 1000. Important parameters like sensing range, data transmission range, etc. are taken into account based on the facts given in the datasheet of cc2420 [28] and TelosB [29].

Figure 5.1 shows the comparison of the efficiency of the proposed algorithm with other protocols with respect to the number of slots attached to the feasible schedule. This figure indicates that the proposed algorithm outperforms over others in terms of the number of slots allotted for preparing the feasible schedule. This result owes to the significant difference in the average two-hop neighbors count and the maximum two-hop neighbors count in real-time which is depicted in Fig. 5.2.

Figure 5.2 shows that the value of the average two-hop neighbors count and the maximum two-hop neighbors count differ from each other significantly. This happens because the deployment of sensor nodes in a WSN

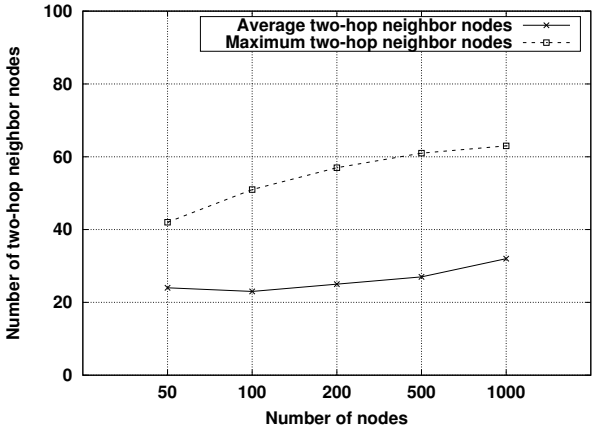

FIG. 5.2. *Average two-hop neighbors vs maximum two-hop neighbors with variable number of nodes using uniform random distribution*

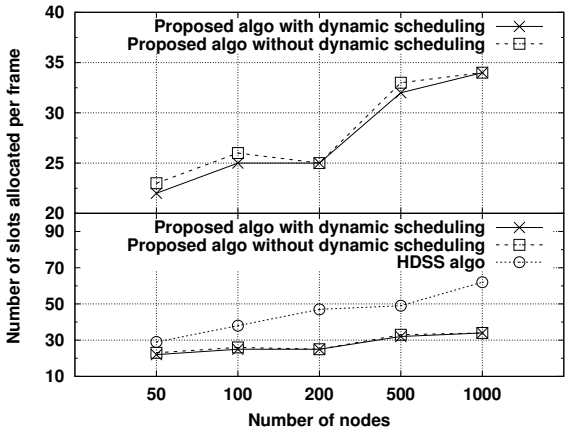

FIG. 5.3. *Number of slots attached to a feasible schedule with and without applying dynamic slot scheduling algorithm for the un-allotted nodes*

can not be 100% uniform. As there is a significant difference between the average two-hop neighbors and the maximum two-hop neighbors count, so, preparing the feasible schedule initially based on average two-hop neighbors will yield a better result in terms of the number of slots in the feasible schedule. This is already proved in the simulation result of the proposed algorithm as depicted in Fig. 5.1.

Figure 5.3 shows that the proposed algorithm prepares a feasible schedule with fewer slots as compared to the HDSS algorithm even if all the remaining un-allotted nodes are assumed to be allotted to separate slots. This graph also shows that the number of slots attached to the feasible schedule are further reduced when the dynamic slot scheduling is applied to allot the slot for the remaining un-allotted nodes. The further reduction in the number of slots is possible as some of the un-allotted nodes may not be the one-hop or two-hop neighbors to each other.

The time spent in the allocation of slots for the remaining un-allotted nodes in the proposed algorithm is compared with the reallocation of slots in the HDSS algorithm. This comparison is depicted in Fig. 5.4 which shows that our proposed algorithm takes very little time as compared to the HDSS algorithm in the allocation of slots to the remaining nodes. The initial slot allotment process for both the algorithms uses a similar approach except the proposed algorithm uses average two-hop neighbors count instead of maximum two-hop neighbors count as used by the HDSS algorithm. Accounting for the above fact, in this figure the initial slot allotment time is not depicted rather the time where both the algorithms significantly differ from each other is depicted

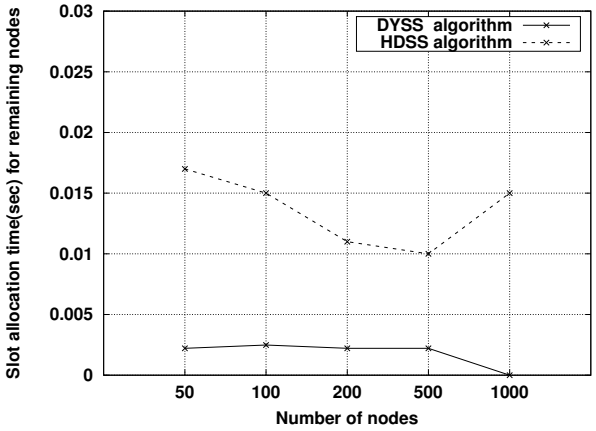

FIG. 5.4. *Time spent in allocation of slots for the remaining nodes*

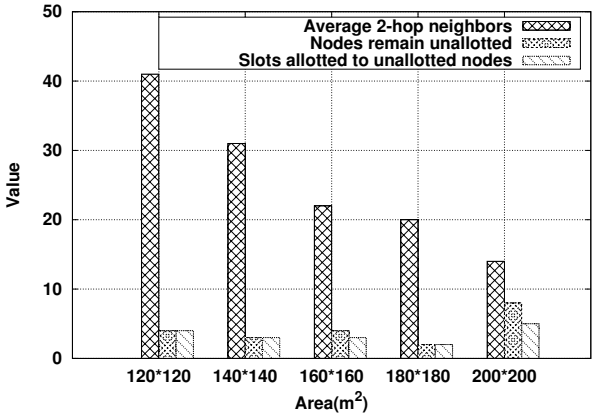

FIG. 5.5. *Comparison of number of average two-hop neighbors, nodes remain un-allotted, slots allotted to un-allotted nodes with varying node density for 100 sensor nodes using uniform random distribution*

to show a clear distinction between both the algorithms. As the time spent on the allocation of slots by the proposed algorithm is very less as compared to the HDSS algorithm. Hence, the energy consumption will also be proportionately less.

Figure 5.5 shows a comparison among the number average two-hop neighbors, nodes remain un-allotted, and slots allotted to un-allotted nodes with varying node density. Whereas Fig. 5.6 shows a similar comparison with varying the number of nodes using random uniform distribution. Figure 5.7 and 5.8 give a similar comparison using Randomised Grid. In all the cases, it is clearly shown that the nodes remain un-allotted are very less as compared to the average two-hop neighbor nodes as the distribution is randomly uniform. Since the number of remaining un-allotted nodes is very less, hence the reduction in the number of slots using the dynamic slot scheduling algorithm is also very less which is already depicted in Fig. 5.3.

6. Conclusion. In this paper, a dynamic slot scheduling algorithm for WSN has been proposed, which prepares a feasible schedule with less number of slots in a quick time. This ultimately helps to handle the collision due to correlated contention and at the same time minimizes latency during data transmission. Initially, slots are allotted to each node based on the average two-hop neighbors count as opposed to the maximum two-hop neighbors count as used in the earlier proposed HDSS algorithm. The use of average two-hop neighbors count reduces the number of slots in the schedule to a greater extent. Then the remaining un-allotted nodes are attached to slots in the best possible way using a novel dynamic slot allocation procedure to prepare the

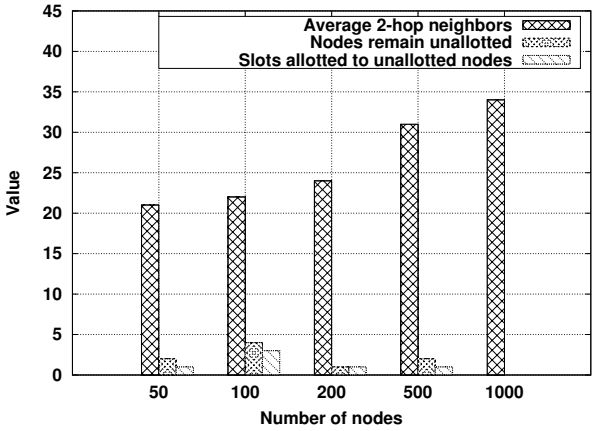

FIG. 5.6. *Comparison of number of average two-hop neighbors, nodes remain un-allotted, slots allotted to un-allotted nodes with varying number of nodes using uniform random distribution*

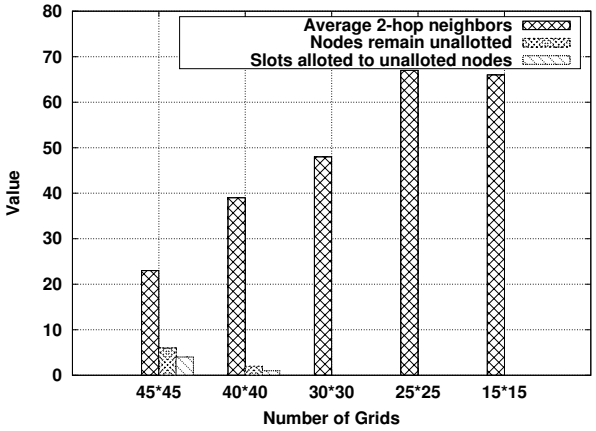

FIG. 5.7. *Comparison of number of average two-hop neighbors, nodes remain un-allotted, slots allotted to un-allotted nodes with varying number of grids with Area 490*490 and number of nodes 1000 using Randomised Grid*

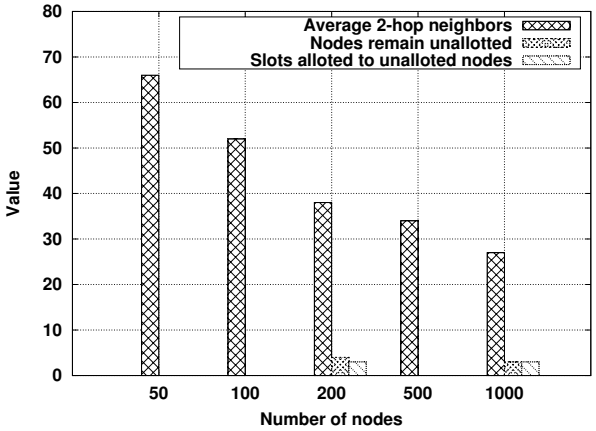

FIG. 5.8. *Comparison of number of average two-hop neighbors, nodes remain un-allotted, slots allotted to un-allotted nodes with varying number of nodes using Randomised Grid*

final schedule. The algorithm is tested in multiple environments like fixed and variable node density with uniform random distribution as well as the randomised grid to ensure it's correctness. The efficiency of the proposed algorithm has been compared with HDSS, DRAND, and RD-TDMA based on the number of slots. The comparison clearly shows that our algorithm outperforms others in terms of the number of slots attached to the final feasible schedule, which ultimately reduces the latency and also handles the collision during data transmission. The performance of the proposed algorithm is studied in an ideal scenario, i.e., a noiseless channel. In the future, we will further extend this algorithm to work in a real environment where every packet transmission is noisy in nature.

REFERENCES

[1] S. C. Ergen and P. Varaiya, *PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks*, IEEE Transactions on Mobile Computing, vol. 5, no. 7, pp. 920-930, 2006.

[2] W. Ye, J. Heidemann, and D. Estrin, *An energy-efficient mac protocol for wireless sensor networks*, Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 1567-1576, 2002.

[3] S. Kumar and H. Kim, *Energy efficient scheduling in wireless sensor networks for periodic data gathering*, IEEE Access, vol. 7, pp. 11410-11426, 2019.

[4] K. Moriyama and Y. Zhang, *An efficient distributed tdma mac protocol for large-scale and high-data-rate wireless sensor networks*, IEEE 29th International Conference on Advanced Information Networking and Applications, pp. 84-91, 2015.

[5] A. Ahmad and Z. Hanzlek, *Distributed real time tdma scheduling algorithm for tree topology wsns*, 20th IFAC World Congress, vol. 50, no. 1, pp. 5926-5933, 2017.

[6] A. Ahmad and Z. Hanzlek, *An energy efficient schedule for ieee 802.15.4/zigbee cluster tree wsn with multiple collision domains and period crossing constraint*, IEEE Transactions on Industrial Informatics, vol. 14, no. 1, pp. 12-23, 2018.

[7] M. R. Lenka, A. R. Swain, and M. N. Sahoo, *Distributed slot scheduling algorithm for hybrid csma/tdma mac in wireless sensor networks*, IEEE International Conference on Networking, Architecture and Storage (NAS), pp. 1-4, 2016.

[8] M. R. Lenka, A. R. Swain, and B. P. Nayak, *A hybrid based distributed slot scheduling approach for wsn mac*, Journal of Communications Software and Systems, vol. 15, no. 2, pp. 109-117, 2019.

[9] A. Bhatia and R. C. Hansdah, *Rd-tdma: A randomized distributed tdma scheduling for correlated contention in wsns*, 28th International Conference on Advanced Information Networking and Applications Workshops, pp. 378-384, 2014.

[10] A. Ahmad and Z. Hanzalek, *An energy-efficient distributed tdma scheduling algorithm for zigbee-like cluster-tree wsns*, ACM Transactions on Sensor Networks, vol. 16, no. 1, pp. 3:1-3:41, 2019.

[11] R. Severino, N. Pereira, and E. Tovar, *Dynamic cluster scheduling for cluster-tree wsns*, 16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC), pp. 1-8, 2013.

[12] J. Long, M. Dong, K. Ota, and A. Liu, *A green tdma scheduling algorithm for prolonging lifetime in wireless sensor networks*, IEEE Systems Journal, vol. 11, no. 2, pp. 868-877, 2017.

[13] I. Rhee, A. Warrier, J. Min, and L. Xu, *Drand: Distributed randomized tdma scheduling for wireless ad hoc networks*, IEEE Transactions on Mobile Computing, vol. 8, no. 10, pp. 1384-1396, 2009.

[14] W. Ye, J. Heidemann, and D. Estrin, *An energy-efficient MAC protocol for wireless sensor networks*, IEEE Infocom, pp. 1567-1576, 2002.

[15] S. Li, D. Qian, Y. Liu, and J. Tong, *Adaptive distributed randomized TDMA scheduling for clustered wireless sensor networks*, International Conference on Wireless Communications, Networking and Mobile Computing, pp. 2688-2691, 2007.

[16] F. Dobslaw, T. Zhang, and M. Gidlund, *End-to-End reliability-aware scheduling for wireless sensor networks*, IEEE Transactions on Industrial Informatics, vol. 12, no. 2, pp. 758-767, 2016.

[17] D. Yang, Y. Xu, H. Wang, T. Zheng, H. Zhang, and M. Gidlund, *Assignment of segmented slots enabling reliable real-time transmission in industrial wireless sensor networks*, IEEE Transactions on Industrial Electronics, vol. 62, no. 6, pp. 3966-3977, 2015.

[18] Y. Wang and I. Henning, *A deterministic distributed TDMA scheduling algorithm for wireless sensor networks*, International Conference on Wireless Communications, Networking and Mobile Computing, pp. 2759-2762, 2007.

[19] I. Slama, B. Jouaber, and D. Zeghlache, *Priority-based hybrid MAC for energy efficiency in wireless sensor networks*, Wireless Sensor Network, vol. 2, no. 10, pp. 755-767, 2010.

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

[21] S. Zhuo, Y. Song, and Z. Wang, *Queue-MAC: A queue-length aware hybrid CSMA/TDMA MAC protocol for providing dynamic adaptation to traffic and duty-cycle variation in wireless sensor networks*, 9th IEEE International Workshop on Factory Communication Systems; pp. 105-114, 2012.

[22] J. Oller, I. Demirkol, J. Casademont, J. Paradells, G. Gamm, and L. Reindl, *Has time come to switch from duty-cycled MAC protocols to wake-up radio for wireless sensor networks?*, IEEE/ACM Transactions on Networking, vol. 24, no. 2, pp. 674-687, 2016.

[23] Y. Li, X. Zhang, J. Zeng, Y. Wan, and F. Ma, *A Distributed TDMA Scheduling Algorithm Based on Energy-Topology Factor in Internet of Things*, IEEE Access, vol. 5, pp. 10757-10768, 2017.

[24] I. Slama, B. Shrestha, B. Jouaber, D. Zeghlache, and T. Erke, *DNIB: Distributed neighborhood information based*

TDMA scheduling for wireless sensor networks, 68th IEEE Vehicular Technology Conference, 2008.
[25] J. Lee and S. Cho, *Tree TDMA MAC Algorithm Using Time and Frequency Slot Allocations in Tree-Based WSNs*, Wireless Personal Communications, vol. 95, no. 3, pp. 2575-2597, 2017.
[26] T. Danmanee, K. Nakorn, and K. Rojviboonchai, *CU-MAC: A Duty-Cycle MAC Protocol for Internet of Things in Wireless Sensor Networks*, Transactions on Electrical Engineering, Electronics, and Communications, vol. 16, no. 2, pp. 30-43, 2018.
[27] *Castalia a simulator for wireless sensor networks*, http://castalia.npc.nicta.com.au/pdfs/Castalia User Manual.pdf.
[28] *Cc2420 data sheet*, http://www.stanford.edu/class/cs244e/papers/cc2420.pdf.
[29] *Telosb data sheet*, http://www.xbow.com/Products/Product pdf files/Wireless pdf/TelosB Datasheet.pdf.

Edited by: P. Vijaya
Received: Dec 10, 2019
Accepted: May 21, 2020