Stabilizing Interference-Free Slot Assignment for Wireless Mesh Networks

Mahesh Arumugam
Cisco Systems, Inc.,
San Jose, CA 95134
USA

Arshad Jhumka
Department of Computer Science
University of Warwick
UK

Fuad Abujarad
Sandeep S. Kulkarni
Department of Computer Science and Engineering
Michigan State University
3115 Engineering
East Lansing, MI 48824, USA

Abstract

In this chapter, we focus on stabilizing interference-free slot assignment to WMN nodes. These slot assignments allow each node to transmit its data while ensuring that it does not interfere with other nodes. We proceed as follows: First, we focus on infrastructure-only part where we only consider static infrastructure nodes. We present three algorithms in this category. The first two are based on communication topology and address centralized or distributed slot assignment. The third focuses on slot assignment where infrastructure nodes are deployed with some geometric distribution to cover the desired area. Subsequently, we extend this protocol for the case where there are mobile client nodes that are in the vicinity of the infrastructure nodes. And, finally, we present an algorithm for the case where a client node is only in the vicinity of other client nodes.

1 Introduction

Wireless Mesh Networks (WMNs) are one type of wireless networks constructed with mesh routers and mesh clients. Mesh routers are the backbone of the WMNs and form a mesh network within themselves. Mesh clients can serve as hosts to their application(s) and at the same time serve as routers to other clients. Mesh clients can also form WMNs within themselves to provide connections between nodes that are not within the transmission range of each other.

Mesh routers are typically stationary. They form the infrastructure of the WMNs. They are different from traditional wireless routers in that they are

often equipped with multiple wireless interfaces. This increases their transmission compatibilities and capabilities. The mesh routers are connected with each other in a way to form an infrastructure through which their clients can connect to other larger networks such as the Internet. One of the advantages the mesh routers have is that they require less transmission power, since they can use the multi-hop connections.

Mesh clients can be of several types such as desktops, laptops, phones, sensors, etc. They are mostly mobile and have wireless interfaces to connect them to mesh routers and other clients. Although mesh clients without wireless interfaces can connect to wireless mesh routers via Ethernet connections, we do not consider them in this chapter since our focus is on MAC layer for wireless channel. Clients in WMNs can form a network within themselves without mesh routers. The clients, while serving their users application, also serve as routers and forward other clients messages to the requested destination. The difference between the mesh routers and the mesh clients is that mesh clients do not have the gateway or bridge functions.

There are a wide variety of applications that can benefit from WMNs. These applications include: broadband home networking, community and neighborhood networking, enterprise networking, metropolitan area networking, transportation systems, building automation, health and medical systems, and security surveillance systems. To illustrate the use of WMNs consider for example the community and neighborhood networking: The typical way of setting up community and neighborhood networks is by connecting a wireless router to the Internet though cable or DSL modem. These types of networks include many points of access to connect their clients to the Internet. This architecture suffers from many limitations: communication between clients in the same network have to go through the Internet, expensive and high bandwidth routers are required to cover the neighborhood, and many dead zones may exist in between homes. WMNs can overcome these problems. Another example of the applications of WMNs is the enterprise networking. Although there was a significant increase in the use of wireless networks in enterprise networking, wireless units are being used as isolated groups with no link between them except Ethernet connections, which are expensive to setup. WMNs will eliminate the need for any Ethernet connections between wireless units by using wireless mesh routers.

WMNs are different from traditional wireless networks. In WMNs clients are connected to more than one point of access. However, in traditional wireless networks, nodes are connected to single point of access. Providing direct connections to every node can be expensive and impractical, because it requires setting up many points of access. WMNs use their routers and clients transmission power to overcome such problems. The advantage of the WMNs over the conventional wireless networks is that, it provides reliable, cost effective, robust, self-configuring, and self-organizing networks.

WMNs are broadly classified [1] in terms of Infrastructure/backbone WMN, Client WMNs, and Hybrid WMNs depending upon the types of nodes participating in them.

- **Infrastructure/backbone WMN.** In infrastructure/backbone WMN, networks are built with routers only. Routers are connected to each other with many links to form backbone network that can be utilized by clients. Routers in this group use two types of communications: infrastructure communication, to connect within themselves and other networks, and user communication, to connect with the their clients. Also, this group uses the gateway and bridge functionality of the routers to connect clients to each other and to existing networks such as the Internet.

- **Client WMNs.** This group consists of peer-to-peer networks among clients. Nodes can serve as hosts to the user application(s), and can provide routing functionality to connect other nodes to each other. Therefore, it is possible for the client WMNs to be built without routers. In this group, packets will be transmitted from the source to the destination though multiple nodes. Clients will forward packets from one node to another until the packet reaches its destination. Nodes in this group are equipped with routing and transmission capabilities to help them perform their tasks.

- **Hybrid WMNs.** This type combines both the infrastructure and the client WMNs together in one network. Both, routers and clients will provide point of access to the network. In the hybrid WMNs, clients will provide more capabilities to the network. They will be used to connect other clients, who are not in any transmission range of any router, to the network.

Our focus is on MAC layer for WMNs. The computation of the MAC needs to be distributed using a collaborative protocol among the WMN nodes. Furthermore, the MAC layer needs to adapt to the mobility of the nodes. However, whenever possible, such mobility should be assisted with static infrastructure nodes that are typically present in WMNs. Given these requirements, there is a need to design a new MAC layer for WMNs that meets them. In this chapter, we propose a TDMA based MAC protocol that meets such requirements.

In particular, in this chapter, we propose interference free slot assignment (TDMA/FDMA algorithms) for three types of WMNs. Our first algorithm focuses on the infrastructure nodes. This algorithm focuses on the case where nodes are static and (relatively) stable. Being relatively stable and static allows one to efficient slot assignment algorithms that utilize algorithms from traditional networking. In the second algorithm, we extend it to deal with the client nodes that are mobile but are in the vicinity of infrastructure nodes. In the third algorithm, we present an algorithm for the case where client nodes are not in the vicinity of infrastructure nodes and, hence form client WMN.

Organization of the chapter. The rest of the chapter is organized as follows: In Section 2, we present the model and assumptions made in this chapter. We also define the problem of slot assignment and what it means to be stabilizing. In Section 3, we present our algorithm for the case where only infrastructure nodes are considered. In Section 4, we extend it to deal with the

case where client nodes are present but only in the vicinity of infrastructure nodes, i.e., they do not form a client WMN by themselves. In Section 5, we extend it to deal with the case where all client WMNs are permitted. Finally, we summarize in Section 6.

2 Model and Problem Statement

In this section, we describe the WMN model and the assumptions in our algorithm. We also identify why these assumptions are reasonable or how the can be met using existing techniques[1]. In other words, many of the assumptions are made for brevity of presentation and can be removed by incorporating existing techniques.

Existence of a leader. We assume that there exists a unique mesh router, denoted as the leader, initiates the TDMA slot assignment computation. There are several ways in which this leader could be chosen. In particular, one may use any of the existing leader election algorithms [3, 5, 12, 14, 20]. The use of these algorithms also ensures that if the leader fails then another unique leader would be elected. Moreover, the leader election algorithm can be tuned to identify the leader most appropriate for a given WMN. For example, one may prefer the leader to be a node with high degree or a node that is centrally located or a node that is expected to have high availability (e.g. bridge node provided by a service provider). Furthermore, the electing of a leader could be done independent of the normal node operation since infrastructure nodes are typically equipped with multiple wireless interfaces. Also, if the network provides it, the leader could also be chosen from available centralized servers (e.g., for authentication). Finally the exact leader and its location are not crucial in that the amount of work that the leader is performing is limited to the initial slots assignment and to the reorganization cycles.

Faults/ Recovery of Infrastructure. The leader makes slots assignment based on its knowledge about working nodes in the network. After slot assignment, if a mesh router fails, then it will be eliminated from the list of infrastructure nodes in the next reorganization cycle. All the slots that were assigned to faulty nodes will be reclaimed and reassigned to the other active routers. If a new mesh router joins the network it will be treated as mesh client until the next organization cycle is started. Hence, it will run the same algorithm that the client node would use until it receives its new slots in the next reorganization cycle.

Neighborhood Discovery. There are no requirements on the way in which the mesh routers will be arranged in their network space. For simplicity, we assume that each node knows its neighbors. This could be implemented in several ways. For example, each node could maintain a list, say listen-from, of nodes is can hear from. It can communicate this list to the leader during the leader election process and before the initial slots assignment, the leader

[1] We expect techniques from other chapters in the proposed book would be applicable here. A proper reference can be added after details of these chapters are known

will compute the topology of the network. All the nodes in the network will send a message to the leader containing the list of the nodes they can listen to. The leader then will be able to compute two important sets: The first set is the listen-from set, which contains the list of all nodes that a specific node can listen to. The second set is the talk-to set, which contains the set of nodes that are in the transmission range of a specific node. After computing these two sets the leader will share this information with all other nodes. Therefore, each mesh router will be able to compute its communication range and with which group of mesh routers its messages may collide. Also the nodes will know the network they are in, and the routing layer can use this information to compute the best route.

Note that in the first two cases (infrastructure-only and clients in the vicinity of infrastructure nodes), only infrastructure nodes need to know other infrastructure nodes that it can communicate with. Clients (in the case where they are in the vicinity of infrastructure nodes) only need to know the infrastructure nodes that they are close to. For the third case (clients forming WMN by themselves), clients need to know their neighbors but this discovery can be easily achieved using standard techniques such as those in [7, 16, 22].

2.1 Problem Statement

In this section, we precisely define the problem of slot assignment. For sake of simplicity, first, we consider the case where only one frequency is available and, hence, all nodes are transmitting on the same frequency. With a single frequency, the problem of slot assignment is the problem of time division multiple access (TDMA) where each node is assigned a set of slots in which it can transmit.

Now, consider the case where two nodes, say A and B are transmitting and both messages could be received by a common neighbor C. Now, if both A and B transmit simultaneously then there will be a collision at C thereby preventing C from receiving either message. The goal of the slot assignment algorithm is to assign slots to each node so that no two nodes transmit simultaneously if their messages will collide at some node. To define the problem statement, we view the WMN as a graph $G = (V, E)$ where V consists of all nodes (infrastructure nodes, clients etc) and E denotes the links between them. The pair (v_1, v_2) is in E iff v_1 can communicate with v_2. Note that the relation E is reflexive, i.e., for any node v_1, $(v_1, v_1) \in E$. It may not be symmetric, i.e., it is possible that $(v_1, v_2) \in E$ and $(v_2, v_1) \notin E$.

First, we define the notion of collision group in WMN; the collision group of a node, say j includes those nodes that should not transmit when j is transmitting. Based on the above discussion, we define collision group as follows:

Definition 1 (Collision group) *The collision group of j is $CG(j)$, where*

$$CG(j) = \{k | \exists l : (j, l) \in E \wedge (k, l) \in E\} - \{j\}$$

Now, using the collision group, we can define the problem of slot assignment for a single frequency (i.e., time division multiple access (TDMA)). The slot assignment problem is to assign each node a set of slots such that two nodes transmitting simultaneously are not in the collision group of each other. Thus, the problem of slot assignment is as follows:

Using this definition, the problem statement for slot assignment is as shown in Figure 1.

Problem statement for slot assignment. Assign a set of slots, say $slot_j$ to each node j such that

$$\forall j, k : j \in CG(k) \Rightarrow slot_j \cap slot_k = \phi$$

Figure 1: Problem Statement for Slot Assignment

Note that this definition of collision group and TDMA takes into account unidirectional nature of the links. In other words, if there are two nodes j and k such that l can communicate with j and k although neither j nor k can communicate with l then the above problem statement allows j and k to transmit simultaneously. Ideally, one should solve the problem of slot assignment by considering the existence of unidirectional links. However, in certain cases, for sake of simplicity, an algorithm may treat all links as bi-directional and assign slots accordingly. For such an algorithm, we define the notion of symmetric collision group,

Definition 2 (Symmetric collision group) *The symmetric collision group of node j is $SCG(j)$ where*

$$SCG(j) = \{k | \exists l : (j,l) \in E' \wedge (k,l) \in E'\} - \{j\}, \text{ where}$$
$$E' = \{(j,k),(k,j) | (j,k) \in E\}$$

Using this definition, the problem statement for slot assignment is as shown in Figure 2.

Problem statement for slot assignment with symmetric links. The problem of slot assignment with bi-directional links is to assign a set of slots, say $slot_j$ to each node j such that

$$\forall j, k : j \in SCG(k) \Rightarrow slot_j \cap slot_k = \phi$$

Figure 2: Problem Statement for Symmetric Slot Assignment

In case of WMNs, a node may be able transmit on multiple frequencies. Hence, the slot assignment not only has to deal with assignment of timeslots but assignment of frequencies as well. In such a model, we view each slot assigned to a node to be of the form (f,t), where (f,t) denotes that the node

is allowed to transmit at time t on frequency f. With such a definition of slots, the problem of slot assignment remains the same as above with the exception that the slot identifies both frequency and time slot.

Additional restrictions on slot assignments in WMNs. Note that the above problem statement imposes no restrictions on which frequencies should be assigned to which nodes. In practice, however, such restrictions could be in place due to hardware limitations. To illustrate this, consider the case where there are two infrastructure nodes, say x_1 and x_2 that communicate with their respective clients y_1 and y_2. Furthermore, in such a network, assume that the infrastructure nodes use one frequency to communicate with other infrastructure node and another frequency to communicate with clients. Then, such a network could use one frequency, say f_1, for communication between x_1 and x_2, another, say f_2, for communication between x_1 and y_1 and a third, say f_3 for communication between x_2 and y_2. Observe that in this scenario, x_1 does not (respectively, cannot) communicate with frequency f_3. Hence, it should not be assigned slots of the form $(f_3, -)$. Likewise, all slots assigned to y_1 would be of the form $(f_2, -)$.

Furthermore, in our solution for the case where clients are always in the vicinity of an infrastructure node, the infrastructure nodes are assigned their slots and the clients *borrow* these slots from them. Given such a model and the scenario in the previous paragraph, x_1 would be assigned all slots in frequency f_2. In this case, we say that frequency f_2 is assigned to node x_1.

Unidirectional antennas. The above problem statement assumes that communication is omnidirectional, i.e., when a node sends a message, it is broadcast to all neighbors that can listen it. The problem (and the solutions in this chapter) could be easily modified to deal with the case where communication uses unidirectional antennas. However, this issue is outside the scope of this chapter.

2.2 Defining Self-Stabilization of Slot Assignment

A solution to the slot assignment problem from the previous subsection would ensure that when a node transmits its message would not collide with other messages and would be received by intended receiver(s). However, if certain faults occur then the interference freedom property may be violated. Examples of such faults include clock drift, variable communication characteristic, etc. For example, if clocks drift then two slots of neighboring nodes may correspond to the same time. Or, if communication characteristics change, e.g., if the slots are assigned under the assumption that j is not in the collision group of k. Now, if communication range of j increases (due to hardware changes or variable nature of communication characteristic) or a new node is present in the area covered by communication ranges of j and k then the time slots must be reevaluated so that the collision freedom is guaranteed.

While certain changes in topologies would be handled explicitly, e.g., when nodes move their slots would be recomputed to ensure collision freedom, unexpected faults could cause the system to reach states where collision freedom is

violated. In these situations, it is necessary to restore the system to a legitimate state. In particular, we argue that the slot assignment algorithms should be self-stabilizing [10, 11], where

Definition 3 (Self-stabilization) *A system is self-stabilizing iff starting from an arbitrary state, it (1) recovers to legitimate state, and (2) upon recovery continues to be in legitimate states forever.*

Thus, a self-stabilizing slot assignment algorithm would ensure that even if faults cause corruption of slots assigned to nodes, the network would eventually recover to states where correct slot assignment is reestablished.

Our approach for providing stabilization is based on periodic update. In particular, in periodic update, each node would update its slots so that they are conflict free with other nodes in its collision group. The update would also ensure that the number of unused slots is reduced/eliminated. Thus, the maximum time for restoring time slots is directly proportional to the time between these updates. And, the overhead of stabilization is inversely proportional to the time between updates. Hence, the value of the period should be chosen based on system needs, level of acceptable overhead, probability of faults, etc. We note, however, that while worst case depends upon the period, many faults would be handled locally whenever feasible. For example, in most situations, node failure, repair or movement would be handled locally and immediately.

3 Self-Stabilizing Frame Assignment for Infrastructure Network

In this section, we present algorithms for assigning frames to infrastructure nodes of a WMN. In order to achieve collision-free communication, we need to ensure that the frames assigned to a node are unique within its distance-2 neighborhood. This can be achieved using distance-2 coloring. As an illustration, consider the communication topology shown in Figure 3. Figure 3(a) considers bidirectional links among the nodes. In this example, if nodes a and c transmit simultaneously then it causes a collision at node b. Hence, a and c should transmit in different frames. In other words, a and b should get different colors. Similarly, node c cannot transmit simultaneously with neighbors of a or b as it leads to collision at a or b. Figure 3(b) considers some unidirectional links among the nodes. In this example, if two nodes can send messages to a common neighbor then they should get different colors in order to ensure collision free communication. On the other hand, node c may transmit simultaneously with nodes that can send messages to a or b directly. This does not cause a collision at a or b as c cannot talk to a or b directly.

Distance-2 coloring. Given a communication graph $G = (V, E)$ for the infrastructure network, compute E' such that two distinct nodes a and b are connected in E' if they are connected in E or if they share a common neighbor c that can hear from both a and b. To obtain distance-2 coloring, we require that

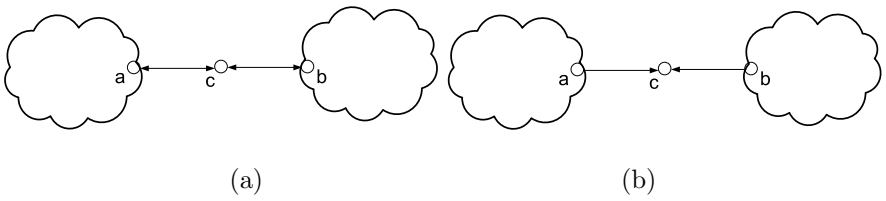


Figure 3: Distance-2 coloring

$(\forall(i,j) \in E' :: color.i \neq color.j)$. Formally, the problem statement is defined in Figure 4.

Problem statement for Distance-2 coloring
Given a directed communication graph $G = (V, E)$; assign colors to V such that the following condition is satisfied:
$(\forall(i,j) \in E' :: color.i \neq color.j)$
where, $E' = \{(a,b) | (a \neq b) \wedge ((a,b) \in E \vee (\exists c \in V :: (a,c) \in E \wedge (b,c) \in E))\}$

Figure 4: Problem statement for distance-2 coloring

Frame assignment. The algorithms presented in this section assign complete frames to each node. Each frame consists of x (≥ 1) slots. The infrastructure node will choose the first slot within its assigned frames for sending messages. The node will assign a subset of other slots to the clients attached to it. (We refer the reader to Section 4 for a discussion on the algorithm that assigns slots to the clients.)

We present two methods for frame assignment in infrastructure networks. The first method considers a network with arbitrary topology where each node is aware of only its local neighborhood (i.e., the nodes that it can directly communicate with). This method is suitable for the case where the application can tolerate network initialization time (to setup the interference-free frames). Moreover, this method does not require any global knowledge and location information. By contrast, the second method considers a network where the location of the nodes are known up front and the network is deployed in some geometric topology to cover a given region. In such networks, the frames can be assigned offline to a *location* and each node can statically determine its frames by virtue of where it is located. This method is suitable for the case where each node is equipped with GPS for determining its location (i.e., its global coordinates). Such networks allow the nodes to start functioning immediately (without significant network initialization overhead) since the frames are computed statically and time synchronization is achieved using GPS. Moreover, addition of new nodes to the network is much faster.

With this introduction, we present our algorithms in more detail, next.

3.1 Frame Assignment Algorithms Using Graph Coloring

In this section, we present two frame assignment algorithms in infrastructure network with arbitrary topologies. We use graph coloring to assign frames to the nodes. Specifically, we obtain distance-2 coloring of nodes that identifies the initial frame assignments. Distance-2 coloring ensures that the colors assigned to nodes i and j are different if i is in the collision group of j (or vice versa). This will ensure that the frame assignment (and, therefore, the slot assignment) would meet the problem statement in Figure 1. Once a node determines its color (i.e., the initial frame), it can compute subsequent frames assigned to it by using the number of colors required to obtain distance-2 coloring. Suppose $color_i$ is the color assigned to node i. Node i gets $\forall c : c \geq 0 : color_i + c \times K$ frames, where K is the number of colors required to obtain distance-2 coloring. In this chapter, K is also referred as the frame period.

Based on the model in Section 2, there is a leader in the network that is responsible for frame assignment. The leader could be chosen by algorithms such as those in [3, 5, 12, 14, 20]. Now, we present the frame assignment algorithms. The first algorithm is centralized where the leader assigns colors to all the nodes in the network. In this algorithm, the leader can optimize the number of colors required. However, in this algorithm, the leader has to learn the network topology before it can assign colors. The second algorithm is distributed in the sense that each node chooses its color depending on the colors chosen by its neighborhood. The main advantage of this algorithm is that, unlike the centralized algorithm, addition of new nodes does not involve the leader. Additionally, the distributed algorithm does not require a node to learn the entire network topology.

Next, we discuss these two algorithms in detail.

3.1.1 Algorithm 1: Centralized Coloring

In this algorithm colors are assigned to the nodes in a centralized fashion. This is achieved using the following three step process: (1) computing the global network topology, (2) coloring the nodes such that two nodes that are within distance-2 of each other have unique colors, and (3) distributing the colors and the frame period to all the nodes in the network. Next, we discuss these steps in detail.

Step 1: Computing the network topology. As mentioned earlier, each node is aware only of its local neighborhood. As discussed in Section 2, all nodes communicate their local neighborhood to the leader. Since many leader election algorithms actually construct a spanning tree that is rooted at the leader, these messages could be sent along this tree. Alternatively, similar to algorithms in [7, 16], these messages could be sent using broadcast primitives. Furthermore, by allowing nodes to combine messages of different nodes, number of messages could be reduced. Thus, once the leader election is complete and a leader is decided, the leader will be aware of the entire network topology.

Step 2: Distance-2 coloring of the nodes. The leader can then apply [19]

to obtain distance-2 coloring of the network. Specifically, in [19], Lloyd and Ramanathan present *minimum degree last* algorithm for distance-2 coloring. First, the algorithm assigns a unique label to each node in a *progressive* fashion. Suppose node i is labeled p. The next node the algorithm chooses to label will be the one with the least number of neighbors in the subgraph formed by all unlabeled nodes. The label of that node is $p + 1$. Once all the nodes are labeled, the algorithm then colors the nodes starting with the highest labeled node. When a node is selected for coloring, the algorithm assigns the lowest numbered color that does not conflict with previously colored nodes. Lloyd and Ramanathan show that ordering obtained using the labeling of nodes is crucial in bounding the worst-case performance of the algorithm. Also, they prove that obtaining an optimal distance-2 coloring of planar graphs is NP-complete (even in an offline setup).

Note that since WMN nodes could transmit on multiple frequencies, the algorithm in [19] would be repeated for each frequency. Moreover, the graph being considered for each frequency would be different based on the nodes that can actually transmit on that frequency. Thus, the number of colors required for each frequency may be different. It follows that the period used for different frequencies may be different.

Step 3: Distributing colors and frame period to the nodes. Once the leader computes the colors of the nodes, it distributes them to the nodes. Towards this end, the leader communicates the color assignments and the frame period (which is equal to the number of colors required to obtain distance-2 coloring) in the slots allocated to it. Whenever a node receives color assignments, it does the following: (i) determines its initial frame assignment (from the color assigned to it), (ii) computes its subsequent frame assignments using the frame period, and (iii) communicates the color assignments it received to its neighbors in slots assigned to it. Continuing in this fashion, the color assignments and frame period are distributed to the nodes and each node determines its frames.

Self-Stabilization. We sketch the outline of how self-stabilization is achieved. As mentioned in Section 2.2, the stabilization is provided by periodic revalidation of frames. This revalidation ensures that frames remain collision free and in case of (controlled) topology change such as addition or removal, frames are recomputed. In case of arbitrary failures, the validation messages may collide preventing a node from receiving its revised frame assignment. In this case, after a timeout, the node reverts to using CSMA and restricts application traffic so as to minimize the network traffic so that revalidation messages would succeed. Once the revalidation of frames is complete, the node subsequently resumes application traffic. The value of the timeout depends on the frequency of update messages and number of nodes. The details of computing this timeout are available in [4].

3.1.2 Algorithm 2: Distributed Coloring

In this section, we present our distributed coloring algorithm for frame assignment in infrastructure network. We propose a layered architecture that includes:

(1) distance-2 neighborhood layer, (2) token circulation layer, (3) distance-2 coloring layer or the TDMA layer, and (4) application layer. The distance-2 neighborhood layer is responsible for maintaining distance-2 neighbor path-list at each node. Path-list of a node identifies the path to all the distance-2 neighbors of the node. The token circulation layer is responsible for circulating a token in such a way that every mesh router is visited at least once in each circulation. The token-circulation layer assumes that the subgraph obtained using only the bi-directional links of the network is connected. The token circulation algorithm uses only the bi-directional links to circulate the token. The distance-2 coloring layer is responsible for determining the initial frame of the node. Whenever a node receives the token, it can choose or validate its color. As before, the color of the node identifies the initial frame of the node. Finally, the application layer is where the actual application resides. All application message communication goes through the TDMA layer. Next, we discuss the first three layers in more detail.

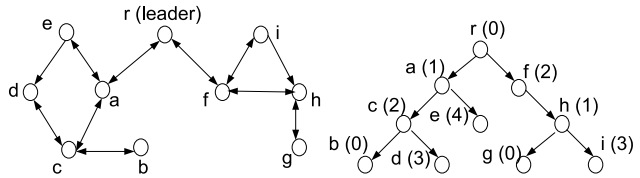
Distance-2 neighborhood layer. As mentioned before, this layer is responsible for maintaining distance-2 neighbors of a node. Towards this end, each node sends distance-2 neighborhood discovery messages. More specifically, each node communicates the information about its immediate neighbors (i.e., nodes that can send messages to this node and nodes that can hear from this node) up to certain distance in the network. The distance up to which a node forwards the information is a tunable parameter. Before the frames are assigned to each node, nodes communicate using CSMA mechanism and rely on back-off schemes for reliability. Once the frames are assigned, this layer sends distance-2 neighborhood discovery messages in the slots assigned to a node.

Token circulation layer. The token circulation later is responsible for maintaining a spanning tree rooted at the leader and traversing the graph infinitely often. The leader initiates the token circulation in the network. As mentioned earlier, we assume that the subgraph formed with bi-directional links in the network is connected. The token traverses the network using bi-directional links (as it provides acknowledgment to the node which forwards the token). In this section, we do not present a new algorithm for token circulation. Rather, we only identify the constraints that this layer needs to satisfy. This layer should recover from token losses and presence of multiple tokens in the networks. Existing graph traversal algorithms [8, 15, 23, 24] satisfy these constraints and, hence, any of these can be used.

Distance-2 coloring/TDMA layer. We use the token circulation protocol in designing a distance-2 coloring algorithm. As before, each node is aware of its local neighborhood. Whenever a node receives the token (from the token circulation layer), it chooses its color. Towards this end, node i first computes the set $used_j$ which contains the colors used in its distance-2 neighborhood. Once it determines this set, it chooses its color such that $color_j \notin used_j$. Subsequently, it reports its color to its distance-2 neighbors (using the slots assigned to it). This action is important since it lets the nodes in the distance-2 neighborhood of j that are not yet colored to compute their *used* sets appropriately. Finally, j forwards the token to one of its distance 1 neighbors (using the token circulation

layer).

As an example, consider Figure 5. Each node first computes the path-list to their distance-2 neighbors. Table 1 identifies the distance-2 neighbors of the nodes for the topology shown in Figure 5(a). The token circulation layer maintains a depth first tree rooted at the leader, i.e., node r. An example depth-first tree is shown in Figure 5(b). Whenever a node receives the token, the distance-2 coloring/TDMA layer computes the colors used in its distance-2 neighborhood.



(a) (b)

Figure 5: Color assignments using distributed coloring. (a) Topology of the network and (b) Traversal and color assignments

Suppose in Figure 5(b) colors assigned to nodes r, a, and c be 0, 1, and 2 respectively. The colors of other nodes are not yet assigned (i.e., *undefined*) since the token has not reached them yet). When b receives the token, it knows the colors that have been taken by nodes in its distance-2 neighborhood. The colors assigned to nodes in the distance-2 neighborhood of b are $\{1, 2\}$. Now, b chooses a color that does not conflict with this set. In the example shown in Figure 5, b sets its color to 0, the minimum color that does not conflict with the colors used in its distance-2 neighborhood. Similarly, other nodes choose their colors.

Table 1: Distance-2 neighborhood for the topology shown in Figure 5

Node	Distance-2 neighborhood
r	$\{a, c, e, f, i, h\}$
a	$\{r, b, c, e, d, f\}$
b	$\{c, a, d\}$
c	$\{r, a, b, d, e\}$
d	$\{a, b, c, e\}$
e	$\{r, a, c, d\}$
f	$\{r, a, i, g, h\}$
g	$\{f, h, i\}$
h	$\{r, f, g, i\}$
i	$\{r, f, h\}$

Once a node determines its color, it can compute frames assigned to it. As discussed in Section 3.1.1, color of a node identifies the initial frame. A node can compute the subsequent frames assigned to it using the frame period (which is equal to the number of colors required to obtain distance-2 coloring). Once the token circulation is complete, the leader knows the number of colors required to obtain distance-2 coloring in the network. In the subsequent token circulation, it forwards this information to the nodes and each node compute its frames accordingly. More precisely, the color of the node identifies the initial frame assigned to it and subsequent frames are computed using the frame period (as discussed earlier). In the example shown in Figure 5, the frame period is 5. Hence, frames assigned node to b are: $\forall c : c \geq 0 : 0 + c \times 5$. Similarly, other nodes determine their frames.

Self-Stabilization. We sketch the outline of how self-stabilization is achieved in this algorithm. In the above algorithm, if the nodes are assigned correct frames then validating them is straight-forward. For example, we can use a simple diffusing computation to report the colors to distance-2 neighborhood and ensure the frames are consistent. In this chapter, for simplicity of presentation, we let the token circulation be used for validation of frames assigned to each node. In the absence of faults, the token circulates the network successfully and, hence, frames are revalidated. However, in the presence of faults, token may be lost due to variety of reasons, such as, (1) frames assigned to nodes are not collision-free, (2) the set containing colors of neighbors is corrupted, and/or (3) token message is corrupted. There may also be transient faults in the network that leads to the presence of multiple tokens or cycles in the network.

Dealing with cycles. To deal with the issue of cycles, we add a time-to-live (TTL) field to the token. Whenever the leader initiates token circulation, it sets TTL to the number of hops the token traverses during one circulation. Since the token traverses an edge twice (once during visiting a node and once during backtracking), the leader sets TTL to $2 \times |E_t|$, where $|E_t|$ is the number of edges traversed in one circulation. Remember that token uses only the bidirectional links and the network formed by the bidirectional links is connected. At each hop, the token decrements its TTL value. When it reaches zero, the token circulation is terminated. Thus, this ensures that the token is caught in a cycle, token circulation terminates and the token is lost.

Dealing with multiple tokens/lost tokens. This is achieved by keeping a timeout at the leader. The value of the timeout is chosen in such a way that any token sent by the leader would return back before the expiry of the timeout. The value of this timeout depends upon the number of nodes in the network. For detailed analysis of the timeout computation, we refer the reader to [4]. Thus, if a token is lost then the leader can generate it by sending another token. If there are multiple tokens then either they will get lost (due to expiry of TTL) or they will return to the leader before the expiry of the timeout. If the leader receives multiple tokens before the expiry of timeout then it implies that there were several tokens in the network. The leader can destroy them. Finally, each node also keeps a timeout to deal with the possible loss of token. Upon expiry

of this timeout, similar to previous subsection, it reverts to using CSMA and blocking the application traffic so that the new token circulation would succeed. It follows that upon expiry of the timeout when the leader sends a token, there is only one token in the system and this token would circulate to reestablish the frames assigned to each node.

3.1.3 Addition/Removal of Nodes in the Network

In this section, we address the issue of addition/removal of mesh routers to/from the infrastructure network. Dealing with removal of nodes is straight-forward. Whenever a node is removed or fails, the frames assigned to other nodes still remain collision-free and, hence, normal operation of the network is not interrupted. However, frames assigned to the removed/failed node are wasted.

New mesh routers are typically added to the network to improve the footprint of the network and to reduce the load on mesh routers. To address the issue of frame assignment to the newly added nodes, we discuss two approaches. The first approach requires the new nodes to behave like client nodes. The second approach requires the nodes to choose conflict-free frames by listening to token circulation and distance-2 neighborhood discovery messages.

Approach 1: Adding new mesh routers as clients. In this approach, whenever a new node is added to the network, it becomes a client of one of its neighbors. Frames of the added node is assigned by its parent infrastructure node using the approach presented in Section 4.

Approach 2: Passive addition of new mesh routers. This approach requires that whenever a node forwards the token (as part of the revalidation process to verify the colors assigned to the nodes), it includes its color and the colors assigned to its distance-1 neighbors. Suppose a new mesh router, say, q is added to the network. Before q joins the network and starts communicating application messages, this approach requires q to learn the colors assigned to its distance-2 neighborhood. One way to achieve this is by listening to token circulation of its distance 1 neighbors. Once q learns the colors assigned to the nodes within distance 2, it selects its color. Thus, q can subsequently determine frames assigned to it. Now, when q sends a message, it announces its presence to its neighbors.

With this approach, if two or more nodes are added simultaneously in the same neighborhood then these new routers may choose conflicting colors and, hence, collisions may occur. However, since the distributed coloring algorithm is self-stabilizing, the network self-stabilizes to states where the color assignments are collision-free. Thus, controlled addition of new routers can be achieved.

3.1.4 Claiming Unused Frames

The algorithms discussed in this section assign uniform bandwidth to all nodes. In this section, we discuss an extension where nodes can claim unused frames/slots in the network, if available. This approach embeds information about the

frames/slots that a node requests in the token and relies on the token circulation layer.

Each node is aware of the frames used by the nodes in its distance-2 neighborhood. Hence, a node can determine the unused slots and if necessary request for the same. A node (say, j) that requires additional bandwidth does the following. Suppose j requires unused slots identified by the set $request_j$. Upon receiving the token, j embeds $request_j$ to the token along with a timestamp that indicates when j made the request. Towards this end, the token contains three tuple information in the set $token.requestSet$: (i) ID of the node, (ii) unused slots requested by it, and (iii) the timestamp. To request for unused slots, j sets $token.requestSet = token.requestSet \cup (j, request_j, timestamp_j)$.

Now, when a node receives the token, it checks $token.requestSet$ to determine if there are any requests for unused slots by nodes in its distance-2 neighborhood. Suppose node k receives the token and finds request from a neighbor j that is within distance 2 of it. Before k decides about the fate of this request, it checks $token.requestStatus$ to determine if other neighbors within distance 2 of j have accepted or denied this request. Towards this end, the token contains three tuple information about the status of each request in $token.requestStatus$: (i) ID of the node, (ii) timestamp when the request was made, and (iii) status (accept or deny). If $(j, timestamp_j, deny)$ is already present in $token.requestStatus$ then k simply ignores j's request as the request cannot be satisfied by some neighbor within distance 2 of j. Otherwise, k proceeds as follows. If there are no other conflicting requests or j's timestamp is earlier than other requests then k lets j claim the slots. To accept j's request, k sets $token.requestStatus = token.requestStatus \cup (j, timestamp_j, accept)$ if no other neighbor within distance 2 of j already added this information. If k finds j's request conflicting then it updates $token.requestStatus$ with $(j, timestamp_j, deny)$.

When the token reaches j in the next token circulation round, it contains the status of its request. It checks $token.requestSet$ to determine the status of its request. It maps the $timestamp_j$ to $request_j$ to identify the slots it has been allowed to use. Once j identifies the additional slots, it removes its request and status information from the token. Specifically, if j is allowed to use slots it requested in $requestj$ then it sets: (i) $token.requestStatus = token.requestStatus - (j, timestamp_j)$ and (ii) $token.requestSet = token.requestSet - (j, request_j, timestamp_j)$. Now, j can start using the slots in $request_j$.

Thus, infrastructure nodes can request for unused slots when necessary using the token circulation layer. Furthermore, when a node requests unused slots, it learns the status of its request within one token circulation round. Additionally, to deal with starvation, we can use lease mechanisms (e.g., [25]) where a node is required to renew the additional slots within a certain period of time.

3.2 Frame Assignment in Infrastructure Network with Known Locations

In this section, we consider infrastructure networks where the locations of the mesh routers are known up front. In this algorithm, frames are assigned offline

and each node statically identifies the frames assigned to it by determining its physical location. To obtain a TDMA schedule, we proceed as follows: (1) impose a (virtual) grid on top of the deployed region, (2) compute the interference range of the network (in terms of grid distances), (3) determine the initial frame assignment, and (4) compute the frame period.

3.2.1 Step 1: Impose a Virtual Grid

As mentioned earlier, in this algorithm, we assume that the infrastructure network is deployed in some geometric topology. Each node determines its physical location using some mechanism (e.g., GPS). This assumption is reasonable since the infrastructure nodes are powerful with sophisticated hardware. The node can then map this physical location in to virtual grid coordinates. Since the node knows the physical location where the virtual grid origin is located and the grid dimensions, it is straight-forward to calculate the virtual grid coordinates. Figure 6 shows an example of imposing a virtual grid on top of the deployed network. Observe that more than one node may be present in a given grid location.

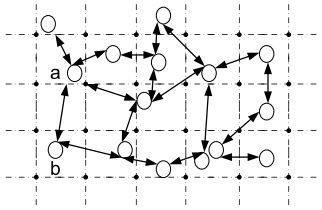


Figure 6: Imposing a virtual grid over the deployed network

3.2.2 Step 2: Compute the Interference Range

In this algorithm, communication ranges are defined in terms of grid distances. For example, in Figure 6, distance between neighbors a and b is 2 grid hops. Furthermore, in this algorithm, we restrict nodes to communicate only with its grid neighbors, i.e., nodes that are 1 grid hop away. Now, we define the notion of interference range in the context of this algorithm. The maximum grid distance up to which a node (say, j) can successfully communicate is called the *interference range* of j. The interference range of the entire network is the maximum of interference ranges of all nodes in the network. This value can be computed either statically (before deployment) or dynamically (after deployment).

An infrastructure network with known locations will be typically used to cover a geometric area, it is expected that the deployment is performed systematically, i.e. it is known up front. As a result, the interference ranges of the nodes can be computed easily. Also, ideally, the interference range should be

as close to 1 as possible. This could be achieved by using appropriate signal strength at each node. Using the interference range of each node, the interference range of the network is determined statically. Alternatively, similar to previous subsection, the interference range of the network could be computed using a leader. In particular, in this case, each node is aware of its local topology and can compute its own interference range during network initialization. This value would then be communicated with the leader who would determine the overall network interference.

3.2.3 Step 3: Assign Initial Frame to Grid Locations

Once a node determines its interference range it knows the frames it is allowed to use. The frames are assigned to grid locations rather than nodes. To ensure connectivity with this algorithm, we need to ensure that the network formed by links that are 1 grid hop distance is connected. This can be achieved by fine-tuning the virtual grid distances used in imposing a grid on top of the network.

Interference range of 1 grid hop. Let us know consider frame assignment to the grid locations for interference range of 1 grid hop. Suppose grid location $\langle 0, 0\rangle$ is assigned frame 0. Locations $\langle 1, 0\rangle$ and $\langle 0, 1\rangle$ will hear messages from a node in $\langle 0, 0\rangle$. Without loss of generality, suppose $\langle 1, 0\rangle$ is assigned frame 1. Location $\langle 0, 1\rangle$ cannot be assigned frame 1 as it will cause a collision at location $\langle 1, 1\rangle$. Therefore, $\langle 0, 1\rangle$ is assigned frame 2. In general, for interference range of 1 grid hop, location $\langle i, j\rangle$ is assigned frame $i + 2j$.

Interference range of y grid hops. Now, we consider the general case where the interference range is y grid hops. Suppose grid location $\langle 0, 0\rangle$ is assigned frame 0. Locations $\langle 1, 0\rangle$ and $\langle 0, 1\rangle$ will hear messages from a node in $\langle 0, 0\rangle$. Additionally, nodes in locations that are within y grid hops may also receive the message. Suppose $\langle 1, 0\rangle$ is assigned frame 1. Location $\langle 0, 1\rangle$ is assigned frame $y + 1$. In general, for interference range of y grid hops, location $\langle i, j\rangle$ is assigned frame $i + (y + 1)j$.

We refer the reader to [17, 18] for detailed discussion on the collision-free property of the above initial frame assignment algorithm. Figure 7 illustrates the initial frame assignment for interference range of 2 grid hops. The numbers marked in each grid location identifies the initial frame assigned to that location.

Thus, given its physical location in the network and the interference range of the network, each node determines its initial frames statically.

3.2.4 Step 4: Compute Frame Period

To compute TDMA frames, we need to determine the period between successive frames assigned to a location. As mentioned before, two locations $\langle i_1, j_1\rangle$ and $\langle i_2, j_2\rangle$ are assigned same frame if nodes located at these locations do not interfere the communication of each other. In this context, we use the notion of *collision-group* (cf. Definition 1). As defined in Section 2, collision-group of

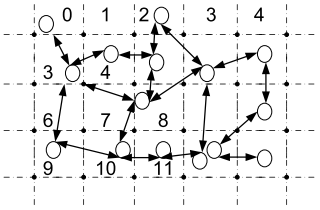


Figure 7: Initial frames of some grid locations for the network with interference range of 2 grid hops

a location (say $\langle i, j \rangle$) identifies the locations that could potentially affect the communication of $\langle i, j \rangle$).

Consider the example shown in Figure 7. The collision-group of a node at $\langle 0, 0 \rangle$ includes nodes located at $\{\langle 0, 0 \rangle, \langle 1, 0 \rangle, \langle 2, 0 \rangle, \langle 3, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 1 \rangle, \langle 2, 1 \rangle, \langle 0, 2 \rangle, \langle 1, 2 \rangle, \langle 0, 3 \rangle\}$. As a result, $\langle 0, 0 \rangle$ gets another slot only after the locations identified in its collision-group have been assigned a slot. In this example, maximum frame assigned to locations in the collision-group of $\langle 0, 0 \rangle$ is 9. Hence, $\langle 0, 0 \rangle$ can choose to transmit in the next frame, i.e., frame 10. In other words, the TDMA frame period for the network shown in Figure 7 is 10.

In general, if the interference range of the network is y then the TDMA frame period is $(y + 1)^2 + 1$. Therefore, location $\langle i, j \rangle$ is assigned the following frames: $\forall c : c \geq 0 : t_{\langle i,j \rangle} + c \times P$, where $t_{\langle i,j \rangle}$ is the initial frame assigned to that location and $P = (y + 1)^2 + 1$ is the TDMA frame period. (We refer the reader to [17, 18] for a formal proof of correctness of this algorithm.)

Thus, each node statically determines its initial TDMA frame and the TDMA frame period.

3.2.5 Dealing with Multiple Nodes in One Grid Location

When a virtual grid is imposed over the deployed network it is possible that multiple nodes fall into a single grid location. As discussed before, the algorithm presented in this section assigns TDMA frames to a grid location rather than the nodes. To deal with the case where multiple nodes are present in a single grid location, we discuss two approaches. The first approach lets the nodes share the frames assigned to that location. Whereas the second approach elects a leader and requires the other nodes to behave like clients.

Approach 1: Share the TDMA frames. In this approach, the nodes share the TDMA frames assigned to that location. Each node is aware of other nodes that fall in its grid location. Again, this information is statically available (based on the location of nodes). In this approach, nodes share the frames uniformly. Suppose 3 nodes i, j, k fall in the same grid location and $i < j < k$. Assuming the frame has x slots, i gets slots $0 \ldots \frac{x}{3} - 1$, j gets slots $\frac{x}{3} \ldots \frac{2x}{3} - 1$, and k gets slots $\frac{2x}{3} \ldots x - 1$.

Approach 2: Elect a leader. In this approach, each node is aware of other

nodes that fall in its grid location. The nodes, however, do not share the frames assigned to that location. Instead, they elect a leader for the location. Existing leader election algorithms [3, 5, 12, 14, 20] can be used to elect a leader. The leader becomes the mesh router of that location. All the other nodes in that location become clients of that leader.

4 Slot Assignment for Infrastructure Nodes and Clients in their Vicinity

As discussed in Section 1, WMNs are classified in terms of Infrastructure WMN, Client WMN and Hybrid WMN. In Section 3, we presented a slot assignment scheme for infrastructure network. In this section, we extend it to deal with hybrid WMNs where clients are in the vicinity of infrastructure nodes. We visualize such a network as a set of clusters. Each cluster consists of one infrastructure node that is the cluster head for that network and a collection of clients that communicate with it. Of these, infrastructure nodes are static and reliable, i.e., they rarely crash. Client nodes, however, are mobile and are subject to crash.

Since a client node may be close to multiple infrastructure nodes, it can be part of multiple clusters. To deal with this situation, we let the client consist of several *virtual* clients, each of which is in one cluster. In particular, each virtual client would receive slots based on the cluster it is in. The client can use any of the slots assigned to its virtual client(s). As far as the application is concerned, it can send/receive its messages through any virtual client. However, certain control messages (such as for allocating slots, requesting slots etc) would be forwarded to the appropriate virtual client based on the infrastructure node involved. It follows that each virtual client will only get control messages from the cluster it is in. For brevity of presentation, whenever it is clear from the context, we drop the word *virtual* in the presentation of the algorithm and use the *client* to mean *virtual client*. The algorithm presented in this section would in fact be executed by each virtual client.

We consider two types of mobility for a client node: (1) intra-cluster mobility and (2) inter-cluster mobility. To accurately capture the notion of cluster mobility, we, first, define the notion of *cluster state*.

Definition 4 (Cluster state) *The* state of a cluster $C_i = (V_i, E_i)$ *at instant* $t \geq 0$*, denoted by* $S(C_i, t)$*, is a tuple consisting of the set of nodes and links in* C_i *at* t*, i.e.,* $S(C_i, t) = (V_i^t, E_i^t)$*, where* V_i^t *(resp.* E_i^t*) is the set of nodes (resp. links) in* C_i *at* t*.*

Intra-cluster mobility captures the fact that, even though nodes are mobile, the mobility pattern is not totally random. The nodes, rather, move in a somewhat coordinated way. Note that when nodes are mobile, if they remain within their assigned cluster, the set of nodes within that cluster remains unchanged, though the links between nodes can change. We now define intra-cluster mobility.

Definition 5 (intra-cluster mobility) *Given two consecutive states of a cluster C_i, i.e., given $S(C_i, t)$ and $S(C_i, t+1)$, we say that there is* intra-cluster mobility in cluster C_i *at instant $(t+1)$ if $V_i^t = V_i^{t+1}$ and $E_i^t \neq E_i^{t+1}$.*

Inter-cluster mobility can occur if the mobility pattern of the nodes is random. When a node leaves its cluster, it is because it has either crashed or has joined a new one due to mobility, causing the *set* of nodes in its original cluster to change (contrast with intra-cluster mobility). Further, when a cluster head crashes or joins a new cluster, its original cluster ceases to exist, and a new cluster needs to be generated. Using our notation for cluster states, we denote the remaining set of nodes by $(\emptyset_i^{t+1}, \emptyset_i^{t+1})$ - note that this set of nodes is not a cluster since it does not have a head.

Definition 6 (inter-cluster mobility) *Given cluster C_i with two consecutive states $S(C_i, t)$ and $S(C_i, t+1)$, we say that a* node p has left cluster C_i at instant $(t+1)$ if $\exists p \in V_i^t$ *s.t* $p \in V_i^t \wedge p \notin V_i^{t+1}$. *We also say that* node p has joined cluster C_i at instant $(t+1)$ if $\exists p \in V_i^{t+1}$ *s.t* $p \notin V_i^t \wedge p \in V_i^{t+1}$. *We say that there is there is* inter-cluster mobility from cluster C_i to cluster C_j *if there exists instants t_1 and t_2, and a node p such that p has left C_i at t_1, and joined C_j at t_2.*

Note that there need not be any relationship between t_1 and t_2 since a cluster head may detect the presence of a new node n before the previous head detects n's absence.

Recall that a client consists of several virtual clients. Consider the case where a client is moving from an infrastructure node A to another infrastructure node B. In this case, initially, it would have only one virtual client that receives slots from A. When it is in the overlapping region between A and B, has two virtual clients and receive slots from both A and B. Once it is out of contact with A, the virtual client corresponding to A would be terminated and it would receive slots only from B. Thus, the *hand-off* between clusters can be handled smoothly.

Our approach is as follows: First, we observe that the infrastructure nodes are more powerful than client nodes. Hence, if given two clients, say c_1 and c_2 if c_1 is in the collision group of c_2 then the infrastructure node(s) associated with c_1 are also in the collision group of the infrastructure node(s) associated with c_2. Hence, to provide collision-free slot assignment, we can rely on the slot assignment to infrastructure nodes.

Hence, for the scenario where clients are in the vicinity of infrastructure nodes, we first run the frame assignment algorithm from Section 3 to assign individual frames to each infrastructure node. Now, client nodes can borrow slots from these frames. To enable such *borrowing* of slots by clients, we adopt a service-oriented perspective for the slot assignment problem: Client nodes can either request slots assigned to their respective cluster head and they can return them. In turn, the cluster head (an infrastructure node) allocates slots to requesting nodes, or returns an updated schedule after nodes have relinquished some slots. To achieve this, the cluster head offers four methods, namely:

1. *Request_slot(id)* slots.
2. *return_slots()* slots.
3. *Allocate_slot(id)* slots.
4. *send_schedule()*.

First, notice that if every node in the cluster is assigned a different slots set, then collision-freedom is ensured within the cluster. The algorithm in Section 3 assigns each cluster head a set of frames. The head then allocates slots from this frame in non-overlapping manner. Since the original frames are interference-free, it follows that assignment of slots to clients also preserves this interference-freedom.

Program for cluster head. We present the program for the cluster head in Figure 8. The cluster head is responsible for scheduling access to the medium. It services requests for slots, and also de-allocates slots when nodes do not use them. Also, this code is executed only in the frames assigned to it.

The first event is a timing event, whereby after each clock tick, the head updates the current slot. After a given number of slots, i.e., max_slots which is decided by the size of the frame, the value of the current slot is reset to 1. In the first slot of every frame, the head sends the schedule for the current frame, which every node in its cluster follows. Then, from slot 2 onwards, the head listens for messages from clients. If it does not hear from an expected client for a threshold number of frames, it decides that the client is no longer in its cluster, and reclaims its slot. In the last slot of the frame, the head listens to requests for slots from new nodes and allocates a slot to the node, by updating the schedule for the next frame.

Program for clients. We now develop the program for the client nodes.

Similar to the cluster head, the client nodes need to keep track of slot count to determine when they can transmit.

Since the cluster head sends schedule information in the first slot of the frame, client nodes wait to hear the message to determine their slot assignment. Due to the interference freedom property of the solution in Section 3, the message of the cluster head does not collide with other infrastructure nodes. Hence, every correct node will hear it. However, since the head is also susceptible to transient failures, the nodes may end with corrupted information. Furthermore, because nodes are mobile, they may end up hearing from a different heads, and they keep track of the assigned slots with each head. Note that $\oplus$ denotes overwrite.

```
variables:
% Timing information
current_slot int init 0;

%State information
node_id: int; % Personal head information
max_slots: int;
TYPE node_access: int×int×int×int; % (id, slot, start frame, rate)
schedule[1...max_slots]: array of node_access sequences
frame_schedule: array of int % array indexed by slot number returning id.

algorithm M_TDMA:
    do forever{
    case ⟨event⟩ of

% updating timing information.
    1. ⟨tick()⟩
        current_slot:=current_slot + 1;
        if (current_slot mod max_slots = 1) then
            current_slot := 1;fi

% first slot of frame reached; head transmits cluster information.
    2. ⟨current_slot = 1⟩
        send_schedule();fi;

% head does not hear an expected message from slot 2 onwards.
    3. ⟨(3 ≤current_slot ≤ max_slots))⟩
        if (frame_schedule[current_slot - 1] ≠ ⊥) ∧
        not rcv(⟨frame_schedule[current_slot - 1], payload⟩)) then
            frame_schedule[current_slot - 1] := return_slots();fi;

% head receives a slot request.
    4. rcv⟨request_slot(id)⟩) then
        frame_schedule:=allocate_slot(id);fi
```

Figure 8: Program for the cluster head (infrastructure node)

When a node reaches its transmission slot, it sends it payload with its id tagged to it.

In the case the message has collided, then the node does not hear its own message. It then knows that its head will remove it from its cluster (by removing its slot entry).

In slot (max_slots), i.e., the one before last, a node(s) new to a cluster requests a slot in the next frame. It checks whether it already has a slot assigned

variables:
% **Node information**
node_id, head_id: int, node_slot: int $\cup\{\perp\}$;
node_slots: head_id $\rightarrow$ int

% **Timing information**
current_slot:int init 0;

algorithm **M_TDMA**:
    do forever{
    case $\langle$event$\rangle$ of

Figure 9: Program for Client

% timing information being updated at client nodes.
    1. $\langle$tick()$\rangle$
        current_slot:=current_slot + 1;

Figure 10: Program for Client (continued)

with the current head, and if not, makes the request.

After the end of the final slot, i.e., the start of the first slot of the next frame, if a requesting node does not hear its own request, then it knows it has collided, and execute exponential back off when making another request.

Self-stabilization. Next, we sketch the proof of self-stabilization. Observe that if the state of a client node is corrupted, then a collision may occur when the client transmits its payload message. The failure to detect the message will cause the cluster head to remove the node's entry for the following frame. The client will thus have a assigned slot $\perp$. The client will thus need to request for new slots again. Hence, the fault can be corrected. In case the state corruption does not lead to collision, the head will still detect an expected message, and will remove the node's entry. Ultimately, the node will need to request slots again, leading to correction.

Likewise, if the state of the head is corrupted such that nodes have conflicting information, collisions will occur. The head will reassign new slots to the requesting nodes, thus correcting earlier faults.

4.1 Example

In Figure 16, there are two infrastructure nodes, namely *head i*, and *head j*, whose region coverage overlap. Each client node belonging to a unique head has a unique slot number. For example, nodes belonging to node i's cluster only

```
% a non-head receive cluster information from head
    2. ⟨rcv(⟨hd_id, rnd_sched, curr_slot⟩)⟩
        current_slot := curr_slot;% slot synchronization with head.
        head_id := hd_id;
        node_slot := lookup(rnd_sched, node_id);
        if (hd_id ∈ dom(node_slots) ∧ (node_slot ≠ ⊥)) then
            node_slots := node_slots ⊕ {hd_id ↦ node_slot};
        else
            node_slots := node_slots ⊕ {hd_id ↦ ⊥};fi;
```

Figure 11: Program for Client (continued)

```
% a non-head transmits in its slot.
    3. ⟨current_slot = node_slot⟩
        bcast(⟨node_id, payload⟩);
```

Figure 12: Program for Client (continued)

will the following slot information: $(i, 2)$, meaning the node belongs to cluster i, and can transmit in slot 2 in the cluster.

Now, consider a node A, belonging to cluster j and having slot 5. In Figure 17, node A has moved, and is now both within the communication range of heads i and j. When it reaches the cluster j, it requests a new slot (event 5), which is allocated by head j. Thus, after its request for slots have been serviced, the slots for node A can be {(j,5), (i,6)}, where slot 6 has been allocated to it in cluster i. Thus, node A now has two virtual clients, one is cluster i, and another in cluster j. Each virtual client can transmit within its cluster's timeframe, in its assigned timeslot.

5 Slot Assignment without Infrastructure Node Support

In this section, we extend the previous algorithms in Sections 3 and 4 to deal with the case where client WMNs are not close to infrastructure nodes and, hence, form an ad-hoc network. In particular, in 3, we had focused on the infrastructure WMN where only infrastructure nodes are considered. In 4, we considered the case where client nodes are introduced but only in the vicinity of the infrastructure nodes. Then, the network was partitioned into clusters where each cluster consisted of one infrastructure node and a collection of client nodes. Since the cluster head was an infrastructure node, we could assume that the possibility of their failure/movement is low and, hence, such crash/movement was ignored. In this section, we further extend the setup to permit client nodes

% A client does not hear its own message.
4. ⟨not rcv(node_id, payload)⟩
node_slots := node_slots \ {head_id ↦ node_slot};

Figure 13: Program for Client (continued)

% head does not hear an expected message
5. ⟨(current_slot = max_slots)⟩
if (node_slots(head_id) = ⊥) then
request_slot(node_id);fi;

Figure 14: Program for Client (continued)

% after nonhead transmits a request
6. ⟨(current_slot > max_slots) or (current_slot = 1)⟩
if not (rcv⟨request_slot(id)⟩) then
execute exponential backoff();fi;

Figure 15: Program for Client (continued)

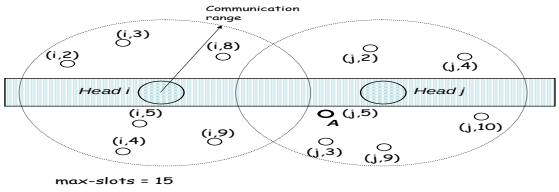


Figure 16: State of a hybrid WMN at time t_0.

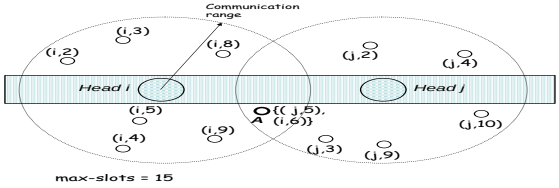


Figure 17: State of a hybrid WMN at time $t_1 > t_0$.

that are not in the vicinity of infrastructure nodes. Hence, a cluster may no longer contain an infrastructure node, i.e., the cluster head could be a client node. With this change, all nodes are subject to crash or move and the mobility can be intra-cluster or inter-cluster.

Based on the above discussion, in this section, we present a MAC protocol that guarantees collision freedom in spite of node mobility. Now, consider the case where nodes are GPS-enabled. In this situation, we could apply the algorithm in Section 3.2; clustering would be done in such a way that cluster heads are equipped with GPS devices and determine the frames assigned to them using their location. For networks where clients do not have GPS devices, we can perform hierarchical clustering using algorithms in [2,6,13,21]. For example, the algorithm in [13] establishes hierarchical clusters and identifies gateway nodes between cluster heads. Such hierarchical clustering could be used along with algorithm in Section 3.1 and allow the cluster head at one level in the hierarchy to assign frames to cluster heads at lower level. Finally, the cluster heads would assign the corresponding slots to clients within their cluster in a manner similar to the algorithm in Section 4. Another scenario in this category is where the clusters are far apart from each other and, hence, can have overlapping slots. Hence, when nodes move across cluster, collisions could occur. Since the scenario where nodes are equipped with GPS devices is straightforward, in this section, we consider the other two scenarios.

We describe the setup first: A clustering algorithm is executed to create clusters (Note that a hierarchical clustering algorithm may be executed, if needed). Then, each node within a cluster is assigned a unique slot. Further, we split the time frame into two parts, namely (i) data part, and (ii) a control part, see Figure 18. In the control part, the head and nodes execute only control events, whereas in the data part, the nodes send their payload. The MAC protocol we develop guarantees collision-freedom in the data part of the cluster schedule. Although collisions are possible in the control part, such collisions are unlikely except in case of high number of nodes moving into the given cluster. Specifically, if more than one node moves into a cluster within any time frame, then collision will occur. And, in these situations, the existing cluster nodes will continue to function even though new nodes may not be able to participate in the cluster immediately.

Before presenting the algorithm, and the assumptions, we present two definitions that we will use in the algorithm:

Definition 7 (i-band) *A node i is said to be in the* i-band *of node j if i is in the communication range of j.*

Definition 8 (o-band) *A node i is said to be in the* o-band *of node j if i can communicate with low probability with j.*

Before presenting the MAC protocol, we make the following assumptions:

1. Nodes within a cluster have non-overlapping slots.

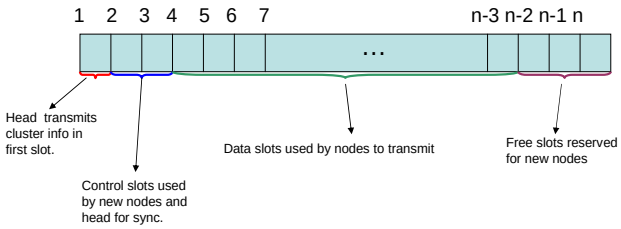


Figure 18: A frame consisting of control slots and data slots

2. All nodes including the cluster head(s) are (potentially) mobile. They can exhibit both intra- and inter-cluster mobility.

3. Every node, including the head, can crash.

4. Any node remains within a cluster for a least one round, unless it crashes. This is to capture the fact that nodes cannot move beyond a certain speed. What this also implies is that nodes will get the chance to transmit. If nodes do not satisfy this mobility constraint, then the algorithm we propose will not guarantee that the node can transmit its messages.

5. If in a given round r, a node is in the i-band of its cluster head, then in round $r + 1$, it will at most be in the o-band of the head, unless the head crashes. Again, this is used to capture the fact that nodes cannot move beyond a certain speed. What it also allows is to distinguish between message collision and a node being far away.

6. No more than 1 round elapses for a node to not hear from a head i.e., a node may not go for more than 1 round without hearing from a head. (Thus could be trivially extended to k rounds where k is an arbitrary number.)

7. Messages sent by cluster heads do not collide on $k \geq 2$ consecutive rounds. This is reasonable because either the clusters are far apart or the slots

assigned to them are not overlapping. But due to mobility the cluster heads may have come close to each other.

The cluster schedule needs to be available even if the cluster head crashes. In our protocol, the cluster schedule is replicated at all the nodes within the cluster, so that one of them can take over once the head has crashed. When a head is detected to have crashed, a clustering algorithm [9] is executed to elect a new head. The property then is that client nodes belong to at most one cluster.

```
on every client node
variables:
% Node information
node_id, node_slot: int; head:{0,1, ⊥};
cluster_id: int; band:{i,o}; status:{⊥, waiting, ⊤, ?} init ⊤;
node_access: int × int × int × int; %(node_id,node_slot, start,rate)
% timing information
current_slot, next_slot init 0;  round init 1;
%cluster information
max_slots: int; new_slot: int init ⊥;
schedule[1...max_slots]: array of node_access sequences
algorithm M_TDMA:
    do forever{
    case ⟨event⟩ of
```

Figure 19: Algorithm at each client node: state information

The variables that a node stores relate to (i) an individual node's information, (ii) timing information and (iii) cluster information (cf. Figure 19). A node stores its own id, and the node slot it is to transmit in a given frame. It also keeps track of whether it is a cluster head or not. It also keeps track of its transmission parameters, i.e., its node_access rights. A node keeps track of timing information by determining the current slot and the current round. Furthermore, every node keeps track of cluster information such as cluster schedule and the maximum number of slots in a frame.

```
1. ⟨tick()⟩
     current_slot:=current_slot + 1;
     if (current_slot mod max_slots = 1) then
          round := round + 1;fi
```

Figure 20: Algorithm at each client node: event 1

The first event (cf. Figure 20) that a node listens to is a timing event. It increments the current slot number with every slot interval. Then, once the current slot number reaches a predefined value, the slot number wraps around, and

the node increments the round value, i.e., once the current slot reaches a predefined value, the node knows that it has reached the next round of transmission, and increments the *round* variable accordingly.

```
2. ⟨current_slot = (max_slots * (round - 1) + 1)⟩
     if (head) then
          bcast(⟨node_id, head, schedule[],round⟩); fi
```

Figure 21: Algorithm at each client node: event 2

When the first slot in any round is reached (second event), a cluster head will broadcast the current schedule for the cluster, together with the value of the current round (Figure 21). This is the second event a node waits for.

```
3. ⟨rcv(⟨hd_id, hd, sched[], rnd⟩)⟩
     if (head ∧ status = ?) then
          status := ⊤;
     if (head != 1 ∧ cluster_id = hd_id) then
          head, schedule,cluster_id, band, round, status := 0, sched,
hd_id, i|o, rnd, ⊤; fi
     if (head != 1 ∧ cluster_id ≠ hd_id) then
          head, schedule,cluster_id, band, round, status := 0, sched,
hd_id, i|o, rnd, ⊥; fi
     if (status = ⊤) then
          node_access := look_up(schedule,node_id); fi
```

Figure 22: Algorithm at each client node: event 3

When a head node hears its own message (Figure 22), it knows there has been no collision at the sender, and it sets its status as "OK" ($\top$). When a non-head node receives the message, its sets itself as a non-head, and updates its version of the cluster schedule. It also keeps track of whether it is in the head i-band or o-band, depending on the strength of the signal. It also sets its status as "OK".

If the second slot is reached (Figure 23), and a cluster head has not yet heard its own message, it knows that there is something wrong, such as a collision, and it sets is status to undecided ("?"). On the other hand, if a non-head node does not hear the message, and it was initially in the o-band of the head, then it assumes that it is now too far from the cluster head, and it resets all of its cluster information, i.e., it assumes it is no longer part of the cluster. Also, the non-head node was in the i-band, then it assumes that some problem, such as collision, occurred, and sets its status to "?". On the other hand, if a node had its status as undecided, and did not receive the message from the head, it concludes that the head as crashed. So, it resets its status to unassigned, and

```
4. ⟨current_slot = (max_slots * (round - 1) + 2)⟩
     if (head ∧ not rcv(⟨cluster_id,1,sched[], current_round⟩)) then
          status := ?; fi
     if (status = ⊤∧ not rcv(⟨cluster_id,1,sched[], current_round⟩) ∧ band=o)
          head,cluster_id,node_access, band, status:= ⊥, ⊥, ⊥, ⊥, ⊥; fi;
     if (status = ⊤∧ not rcv(⟨cluster_id,1,sched[], current_round⟩) ∧ band=i)
          status:= ?; fi;
     if (status =? ∧ not rcv(⟨cluster_id,1,sched[], current_round⟩) ∧ band=i)
          status:= ⊥; cluster_id, head := run_cluster [9];fi;
     if (status = ⊥) then
          bcast(⟨node_id⟩);
          status := waiting; fi
```

Figure 23: Algorithm at each client node: event 4

the node will take part in re-electing a new cluster head. Furthermore, if a node has status unassigned ("⊥"), then it broadcasts a message, informing the head that it is a new node, and sets its status to "waiting" for transmission rights.

```
5. ⟨rcv⟨(new_id)⟩⟩
     if (head ∧| {k:4..max_slots| schedule[k] = ⟨⟩}| ≥ 2) then
          new_slot:=choose{k:4..max_slots| schedule[k] = ⟨⟩}
          rate:=1;% rate=1 denotes every round
          schedule[new_slot]:= ⟨(new_id, new_slot, round, rate)⟩; fi

     if (head ∧| {k:4..max_slots| schedule[k] = ⟨⟩}| = 1) then
          new_slot:=choose{k:4..max_slots| schedule[k] = ⟨⟩}
          rate:=2;% rate=1 denotes every second round
          schedule[new_slot]:= ⟨(new_id, new_slot, round, rate), ⊥⟩; fi

     if (head ∧| {k:4..max_slots| schedule[k] = ⟨⟩}| = 0) then
          new_slot:=choose{k:4..max_slots| |schedule[k]| > 1};
          schedule:=update_schedule(schedule,new_id,new_slot);
              % rate increases exponentially
```

Figure 24: Algorithm at each client node: event 5

When the head receives a request for slots from a new node (Figure 24), it does a non-colliding allocation. If there are empty slots, then the node is assigned one of these slots in every round. However, if there is only one empty slot, then, to be able to tolerate future mobility, the remaining bandwidth is halved by allowing the new node to transmit at half the rate of the empty slot.

When the third slot in a given round is reached (Figure 25), the head

```
6. ⟨current_slot = (max_slots * (round - 1) + 3)⟩
     if (head ∧ new_slot ≠ ⊥) then
          bcast(⟨new_id,new_slot,round,rate⟩);
          new_slot:= ⊥; fi
```

Figure 25: Algorithm at each client node: event 6

then broadcasts the new transmission information, which is then picked up by the requesting node. Note that, because of assumption 4 above, the requesting node will still be in the cluster at when the head broadcasts its new transmission information.

```
7. ⟨rcv⟨(id,slot,start,rate)⟩⟩
     if (node_id = id) then
          node_access:= (id,slot,start,rate);
          next_slot:= (start - 1)*max_slots + slot;
          status:= ⊤; fi
```

Figure 26: Algorithm at each client node: event 7

When the requesting node receives its transmission information (Figure 26), it calculates its set of assigned slots, and sets its status as "OK" ($\top$).

```
8. ⟨current_slot = (max_slots * (round - 1) + 4)⟩
     if (not rcv(⟨(id,slot,start,rate)⟩ ∧ status = waiting) then
          exponential_backoff();
```

Figure 27: Algorithm at each client node: event 8

If the fourth slot is reached (Figure 27), and a new node does not receive its new transmission information, then it concludes that either (i) the head has crashed, or (ii) its request has collided with another potential new node's request. Hence, the new nodes will execute exponential backoff, within the current cluster, before requesting transmission slots again.

After the fourth slot (Figure 28), any node will transmit its payload in an allowed slot. Note that a node is only allowed to transmit when its status is "OK". A node will still retain its transmission information even in the presence of head crashes.

If the head does not hear from a node (Figure 29), it assumes the node to have crashed, and it reclaims the slots allocated to the node.

```
9. ⟨current_slot = next_slot⟩
     if (status ≠ ?) then
          bcast(⟨node_id, cluster_id, payload⟩); fi
     next_slot:= current_slot + (max_slots * rate);
```

Figure 28: Algorithm at each client node: event 9

```
10. ⟨(max_slots * (round - 1) + 4) ≤ current_slot ≤ (max_slots * round)⟩
     if (head) ∧ not (rcv⟨(id, cluster, payload)⟩) ∧ status ≠ ? then
          schedule:= reclaim_slots(schedule); fi
```

Figure 29: Algorithm at each client node: event 10

5.1 Informal proof of correctness

Under a fault free scenario, the correctness proof can be reasoned as follows: A head h knows that node m has joined its cluster at instant $n+1$ when it assigns a slot to m, i.e., m has an entry in schedule. This occurs in the 3rd slot of a round.

Under M_TDMA, when a node hears from a new head, it knows that it is in a cluster (without joining). This occurs in the first slot of a round, i.e., instant $(n-1)$, and it updates its state. In the next slot, i.e., instant n, it broadcasts its id, and sets it status to "waiting". When the head receives a message about a new node (hence there is no collision), event 5 is triggered. It assigns a slot to the new node according to whether there is a free slot or whether the last slot is to be shared. Because the remaining bandwidth is always halved, it means that, theoretically, there is always available bandwidth, half of which is then allocated.

In the 3rd slot, i.e., instant $(n+1)$, it broadcasts the new node access information to the new node, indicating which slot is to be accessed when and what rate. At this point, the node joins the cluster. The cluster remains collision-free since, in any given round, no slot is shared by two nodes. From Lemma 1, the cluster remains collision free.

When a non-head node crashes, the head detects it when no message is obtained in a given allocated slot. In this case, only the allocated slots are reclaimed. Since the slot assignment was non-colliding, reclaiming unused slots maintain the non-interference property of the assignment. If a head is absent (crashes or move beyond its cluster), a clustering algorithm is executed, and a new head is elected. Since the new head has the cluster schedule, it can maintain it. Later, once it detects that the previous head is absent, it will reclaim the unused slots. Again, non-interference is preserved in the data part of the schedule.

5.2 Example

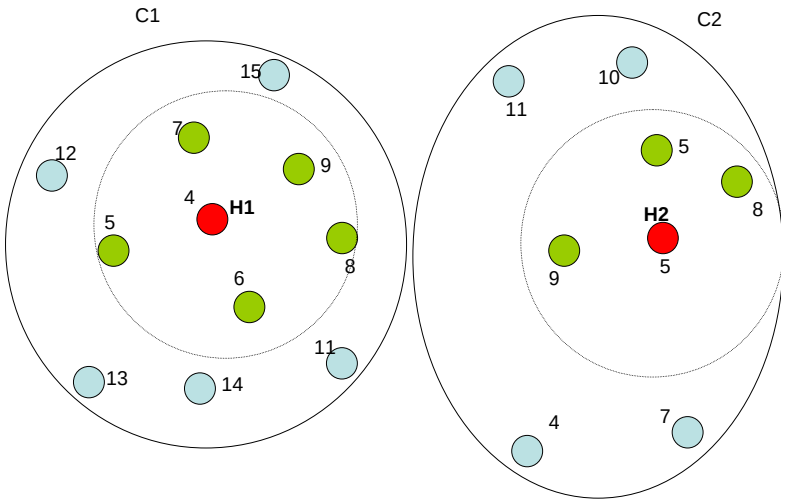


Figure 30: Setup: Two clusters, with each node in each cluster having non-overlapping slots. Nodes across clusters can have overlapping slots.

In Figure 30, two clusters are shown at some point during the execution of the protocol. Within each cluster, no pair of nodes have overlapping slots. However, nodes across clusters may have overlapping slots. Node $H1$ (resp. $H2$) is the head of cluster $C1$ (resp. $C2$). Nodes within the dotted circle with $H1$ (resp. $H2$) as center are in the i-band of the head, whereas nodes outside are in the o-band of the head. The numbers by the side of the nodes represent the node's assigned slot number. Also, we assume that a frame is 15 slots long (3 control slots and 12 data slots). Since there are 11 nodes in the cluster, it means that $H1$ cannot assign the remaining bandwidth (1 slot) completely to a new node.

If nodes display intra-cluster mobility only, then no collision will occur since nodes do not have overlapping slots. If a node moves from the i-band to the o-band of the head, then the probability of correct message transmission to the head is low, and the node may be incorrectly interpreted as crashed or not present. In either case, this will cause the node to lose its slots (event 10), which will preserve the collision-freedom property of the protocol. On the other hand, if a node moves from the o-band to the i-band of its cluster head, then two situations arise: (i) if the node still has its slot, no new slots are assigned to it, thus collision-freedom is ensured, or (ii) if the node did not have a slot,

it will eventually be assigned a slot that do not overlap within the cluster, thus collision-freedom is ensured.

So, we will focus on inter-cluster mobility. In Figure 31, a node moves from the o-band of the head of cluster C2 to the i-band of the head of cluster C1.

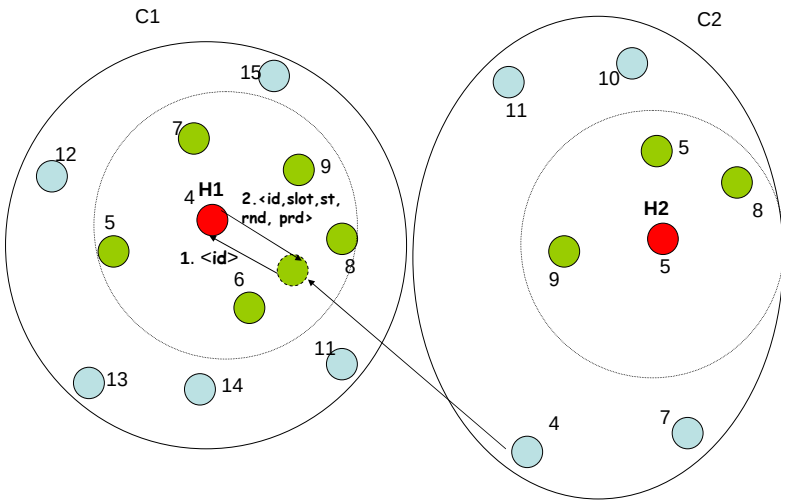


Figure 31: A node moving from the o-band of the head of cluster C2 to the i-band of the head of cluster C1

When the node reaches its new cluster C1, it broadcasts its id (arrow 1). When the head receives this message in slot 2, it sends to the node its transmission information, viz its slot number, rate, and the starting round (arrow 2). Here, the head will detect that only slot 10 is free. Upon detecting that there is only one slot available, the head halves the bandwidth by allowing the requesting node to access the medium every second round, according to event 5 (second if statement). Since the head calculates the transmission information in such a way that there is no slot overlap, collision-freedom is ensured. Thus, assume the execution is in round 2. The slot number in the current round will range from 16 to 30. Assume that node *sid* (read some id) makes a request in slot 17 (2nd slot in round 2), and is the only requesting node, then *sid* receives the following information in the 3rd slot of the round: $\langle sid, 10, 2, 2\rangle$. Thus, node *sid* will start transmitting when the slot reaches $(2-1)*15+10=25$, and it will also calculate its next transmitting slot by $((2-1)*15+10)+(2*15)=55$ (i.e., in slot 10 of round 4). On the other hand, when the head of C2 does not hear *sid* transmit in the 4th slot of round 2, according to event 10, it reclaims

the assigned slot.

In general, it can be shown that the algorithm tolerates node crashes, unrestricted node mobility, as well as new external nodes joining clusters.

6 Summary

In this chapter, we presented stabilizing algorithms for interference-free slot assignment in WMNs. First, we considered WMNs that only consist of infrastructure nodes. We presented three algorithms in this category. The first two algorithms relied on knowledge of local communication topology to determine how slots should be assigned to each node. Of these, one relied on centralized calculation of slots and the other relied on distributed calculation. These approaches provide a tradeoff between the time to add a new node to the network and bandwidth utilization. The third algorithm focused on situations where nodes are deployed to cover a geometric region. In this case, the knowledge about the node location was utilized to assign bandwidth efficiently as well as permitting quick addition of nodes in the network.

Subsequently, we extended this algorithm to deal with the case where mobile client nodes are added although they are close to infrastructure nodes. In this case, we first assigned slots to infrastructure nodes and provided an approach for clients to *borrow* these slots as needed.

We also presented an algorithm for slot assignment for clients are not close to infrastructure nodes and, hence, form a client WMN. This algorithm allows client nodes to form clusters and permits cluster heads to assign slots to different nodes within the cluster. Since an arbitrary hybrid WMN can be viewed as a union of client WMNs and a WMN where some clients are in the vicinity of the infrastructure nodes, the last two solutions can be applied for such WMNs.

The use of such slot assignment algorithms would be especially valuable when some quality of service needs to be provided to applications and where the data rate is moderate. By ensuring that communication of one node does not collide with that of other nodes allows one to provide guarantees on communication delay and guarantees on successful delivery. By contrast, if the load is low, CSMA based approach may work better since low load makes it less likely that collisions would occur.

For simplicity of presentation, in this chapter, we considered the case where the number of slots assigned to all nodes is the same (or close). However, it can be easily extended to the case where some nodes are given larger bandwidth than others. For example, in case of solutions with graph coloring, such nodes could be assigned multiple colors thereby providing them more bandwidth. Letting such nodes have higher priority in claiming unused slots could also result in preferential treatment for some nodes.

References

[1] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Computer Networks*, 2005.

[2] A.D. Amis and R. Prakash. Load-balancing clusters in wireless ad hoc networks. *ASSET*, 2000.

[3] A. Arora. Efficient reconfiguration of trees: A case study in the methodical design of nonmasking fault-tolerance. *Proceedings of the Third International Symposium on Formal Techniques in Real Time and Fault-Tolerance*, pages 110–127, 1994. *Science of Computer Programming* to appear.

[4] M. Arumugam and S. S. Kulkarni. Self-stabilizing deterministic time division multiple access for sensor networks. *AIAA Journal of Aerospace Computing, Information, and Communication (JACIC)*, 2006.

[5] Susmit Bagchi and Purnendu Das. A round-2 randomized leader election algorithm and latency for mdvm system. In *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, page 103, Washington, DC, USA, 2005. IEEE Computer Society.

[6] S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. *Proceedings of Vehicular Technology Conference*, 1999.

[7] G. Chakrabarti and S. S. Kulkarni. Load balancing and resource reservation in mobile ad-hoc networks. *Ad Hoc Networks*, 2006.

[8] A. K. Datta, C. Johnen, F. Petit, and V. Villain. Self-stabilizing depth-first token circulation in arbitrary rooted networks. *Distributed Computing*, 13:207–218, 2000.

[9] M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani. A fault -local self -stabilizing clustering service for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 2006.

[10] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11), 1974.

[11] S. Dolev. *Self-Stabilization*. The MIT Press, 2000.

[12] Stefan Dulman, Paul Havinga, and Johann Hurink. Wave leader election protocol for wireless sensor networks. *International Symposium on Mobile Multimedia Systems and Applications*, 2003.

[13] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. *Proc. 17th ACM Symp. on Computational Geometry,*, 2001.

[14] F. Gartner. A survey of self-stabilizing spanning tree construction algorithms. Technical Report IC/2003/38, Swiss Federal Institute of Technology (EPFL), 2003.

[15] C. Johnen, G. Alari, J. Beauquier, and A. K. Datta. Self-stabilizing depth-first token passing on rooted networks. *In Proceedings of the Workshop on Distributed Algorithms*, 1997.

[16] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing*, pages 153–181, 1996.

[17] S. S. Kulkarni and M. Arumugam. SS-TDMA: A self-stabilizing MAC for sensor networks. In *Sensor Network Operations*. Wiley-IEEE Press, March 2006. `http://www.cse.msu.edu/~sandeep/publications/ka05IEEEPress/`.

[18] Sandeep S. Kulkarni and Umamaheswaran Arumugam. Collision-free communication in sensor networks. *In Proceedings of the Sixth Symposium on Self-Stabilizing Systems (SSS)*, June 2003.

[19] E. L. Lloyd and S. Ramanathan. On the complexity of distance-2 coloring. *In Proceedings of the International Conference on Computing and Information*, 1992.

[20] N. Malpani, J. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. *Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 2000.

[21] A. B. McDonald and T. Znati. A mobility based framework for adaptive clustering in wireless ad-hoc networks. *IEEE Journal on Selected Areas in Communications*, 1999.

[22] M. R. Pearlman, Z. J. Haas, P. Sholander, and S. S. Tabrizi. On the impact of alternate path routing for load balancing in mobile ad hoc networks. *Proceeding of 2000 First Annual Workshop on Mobile and Ad Hoc Networking and Computing, Mobihoc 2000, Boston, MA, USA*, pages 3–10, August 2000.

[23] F. Petit. Fast self-stabilizing depth-first token circulation. *In Proceedings of the Workshop on Self-Stabilizing Systems, Springer*, LNCS:2194:200–215, 2001.

[24] F. Petit and V. Villain. Color optimal self-stabilizing depth-first token circulation. *In Proceedings of the Symposium on Parallel Architectures, Algorithms, and Networks*, 1997.

[25] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, March 2003.