EasyMAC: A New and Simple Protocol for Slot Assignment

Jerrolyn Brees and Sukhamay Kundu
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803, USA

Abstract

We give a new distributed algorithm EasyMAC for the slot assignment for media access control (MAC) of nodes in a sensor network. Our algorithm reduces both the number of messages and the time required to complete the slot assignment compared to existing algorithms. We achieve this by maintaining additional information at each node about the collisions that it detects and sending that information to its neighbors.
Keywords: slot assignment, wireless MAC protocol

1 Introduction

We consider a sensor network composed of individual nodes which communicate through wireless broadcast transmission. We assume that every node has a distinct id, which is included in every message it sends. Two nodes are neighbors (adjacent) if they can receive each other's messages, i.e., if they are within their broadcast range.

If two or more messages reach a node at the same time, then a message collision occurs at the node, and the node fails to decipher any of the colliding messages. Since these messages will have to be rebroadcast by the senders in order to transmit their data, the collision amounts to wasted power, and as sensor nodes have limited available power, collisions should be avoided. Media access control (MAC) algorithms reduce or eliminate collisions.

A particular form of MAC is to assign each node x a slot s_x in a frame, where a frame consists of a consecutive set of, say, f slots (i.e., as a form of Time Division Multiple Access (TDMA) [6]). Given a network N, a slot assignment $x \rightarrow s_x$ is valid if every node in N can send a message in the same frame during its assigned slot without causing a collision. That is, $s_x \neq s_y$ whenever x and y are at distance ≤ 2. Note that collisions occur when two adjacent nodes or two nodes who share a neighbor (i.e., hidden terminal [8]) attempt to send a message during the same slot.

If we think of the slots as a resource, then the slot assignment problem for MAC can be regarded as a distributed mutual exclusion problem [4] for the modified network N' which has the same nodes as N with two nodes x and y considered adjacent if they are within distance 2 from each other in N. However, we cannot use the algorithms in [4, 5] for our purpose because those algorithms assume the existence of an underlying communication mechanism (such as MAC slot assignment) so that messages can be transmitted from a node to a neighbor without interference and message loss. The MAC slot assignment problem considered here is also different from the distributed graph coloring problem [3] for the same reason.

Our main contribution is a new slot assignment algorithm EasyMAC, which is simpler and more efficient than the LooseMAC algorithm [2] in terms of both the number of messages sent and the number of time slots used. In both EasyMAC and LooseMAC, we begin with a random slot assignment that is gradually modified via the exchange of collision (and other) messages until a valid assignment is reached. Some other slot assignment algorithms include [1, 7, 9].

2 Definitions

Table 1 summarizes the notations used in this paper. When a sensor node x sends (broadcasts) a message, all of the neighbors N_x^1 of x receive that message. The two-neighbors of node x are the nodes z that are not neighbors of x but are able to send and receive messages directly to and from neighbors of x, i.e., $z \in N_x^2$ if and only if $\exists y \in N_x^1 \cap N_z^1$ and $z \notin N_x^1$. In Figure 4, N_x^2=y. Clearly, $y \in N_x^i$ implies $x \in N_y^i$.

We assume that time is discretized into equal sized slots and that each successive set of f slots are grouped together into frames. We refer to the i-th frame as F_i and refer to the j-th slot in F_i as $t_{i,j}$. See Figure 1. We use a simpler notation t_j for the j-th slot when the frame number is implied. We assume that frames are synchronized across all nodes, i.e., each frame begins at the same time for all nodes. The problem of unsynchronized frames can be handled by techniques discussed in [2].

It is known that there is always a valid slot as-

Table 1: EasyMAC Algorithm Notation with nodes x and $y \in N_x^1$ during F_i.

Notation	Description
N_x^1	Nodes that receive messages directly from x, i.e., the neighbors of x.
N_x^2	Nodes that receive messages directly from some $y \in N_x^1$ but not directly from x, i.e., the nodes at a distance 2 from x or two-neighbors of x.
$N_x^{1,2}$	$N_x^1 \cup N_x^2$ (disjoint union).
F_i	ith frame (see Section 2 and Figure 1).
$t_{i,j}$	Time slot j in F_i.
$m_{x,i}$	Message sent by x during F_i (based on data gathered by x during F_{i-1}).
$s_{x,i}$	Slot chosen by x during F_i.
$s_{x,y,i}$	Slot that x believes y has chosen for F_i based on messages $m_{y,j}$ received by x in frames $F_j, j \le i$.

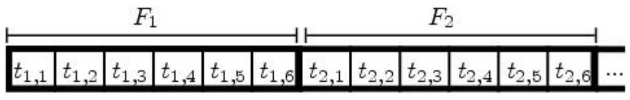


Figure 1: Frames with f=6 slots.

signment when the frame size $f \geq 1 + \Delta_2$, where $\Delta_2 = \max\{|N_x^1| + |N_x^2|\}$, for all nodes x. Figure 2 shows a network with $\Delta_2 = 4$ and a valid slot assignment for $f = 3 < \Delta_2$. On the other hand, Figure 4 shows a network with $\Delta_2 = 3$ where the minimum f to allow a valid slot assignment is $4 = 1 + \Delta_2$.

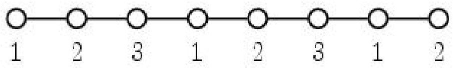


Figure 2: A network with $\Delta_2 = 4$ and a valid slot assignment for $f = 3$. Next to each node x is shown its assigned slot s_x.

At any point in time, a node x has a particular slot selected, and it sends messages only during that slot (once per frame, if any). We say that a node x *receives* a message (clearly), without collision, from $y \in N_x^1$ during a slot if no other neighbor of x nor x itself sends a message in the same slot.

2.1 Collisions

A message *collision* occurs at a node x when two or more messages reach that node at the same time, i.e., during the same slot. When this happens, x detects the presence of multiple messages but cannot decipher the content of the messages. Additionally, x cannot determine the identity of the nodes sending the colliding messages. As shown in Figure 3, a collision can happen at x in two ways:

1. A node x and one of its neighbors $y \in N_x^1$ send messages during the same slot. Here, both x and y detect the collision during that slot.
2. Two nodes y, $z \in N_x^1$ send messages during the same slot. The collision is detected at node x during that slot; and if y and z are not adjacent, then they will not detect the collision.

In the second case, x broadcasts a message (in the next frame) to inform its neighbors about the collision.

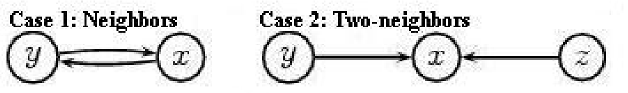
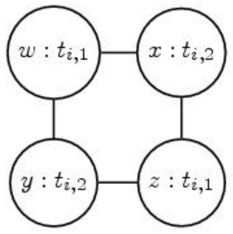


Figure 3: Example of collisions at a slot s. The arrows indicate the directions of colliding messages sent during s.

Since the collision message from x to a node w might collide with another message at w, w assumes that any collision it hears is potentially a message about a collision that it has caused. This assumption is necessary to avoid the problematic situation shown in Figure 4.



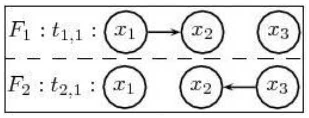
If all nodes send a message in F_i, nodes w and z detect the collision at slot $t_{i,2}$ caused by nodes x and y, and nodes x and y detect the collision caused at slot $t_{i,1}$ by nodes w and z. The collision messages that are sent in F_{i+1} also collide. If the nodes do not pick new slots after hearing the collisions, their messages will continue colliding.

Figure 4: Problem that can occur if colliding messages are ignored.

2.2 Conflict

Node x recognizes a *conflict* when it detects that there is a potential for future message collisions. Consider two nodes y, $z \in N_x^1$. Suppose in frame F_i node x receives a message from y during slot $t_{i,m}$. Now suppose in frame F_j $j > i$, x receives a message from z during the slot $t_{j,m}$, which was previously claimed by y, while y does not send a message in F_j. Node x recognizes that message collisions could occur between y and z if they both send a message during the same frame F_k, $k > j$. Thus, x takes a preventative action by broadcasting a conflict-message in F_{j+1}. See Figure 5.

Since the purpose of both conflict and collision messages is to inform other nodes to change their current slots, we use collision messages to warn about both collisions and conflicts.



- x_1 sends a message to x_2 during $t_{1,1}$ of F_1.
- x_3 sends a message to x_2 during $t_{2,1}$ of F_2.
- If x_1 and x_3 send a message during the same frame, a collision would occur at x_2.

Figure 5: Simple example of how a conflict can occur given the three node network.

2.3 Ready Node

A node x is considered *ready* when it determines that it has picked a slot that has not been chosen by any $y \in N_x^{1,2} = N_x^1 \cup N_x^2$. In EasyMAC, x can determine this if it is able to keep the same slot for two consecutive frames. The details of this are discussed in Section 3.4.

Once x is ready, it keeps its chosen slot regardless of future collisions heard or collision-messages received; however, it continues to participate in the slot selection for other nodes by informing its neighbors of any collisions or conflicts that it hears. When a node determines that all of its neighbors are ready, it can then quit the algorithm. At any point in time, all ready nodes in N_x^1, including x if ready, have distinct slots.

3 EasyMAC Algorithm

The basic structure of the EasyMAC algorithm at a node x is as follows:

1. In the beginning of frame F_0, select a random slot $s_{x,0}$ from frame F_0 and let $m_{x,0} = bcn(x)$.
2. Now do the following for each frame F_0, F_1, ...
   (a) Send the message $m_{x,i}$ in slot $s_{x,i}$ for the current frame F_i.
   (b) Collect information on all collisions detected by x and all the messages received by x from its neighbors N_x^1 during F_i.
   (c) Based on the information collected in 2(b), choose a new slot $s_{x,i+1}$ for frame F_{i+1} as described in Section 3.2, if necessary; otherwise, keep the same slot, i.e., $s_{x,i+1} = s_{x,i}$.
   (d) Choose the message $m_{x,i+1}$ to be sent, if any, in the next frame F_{i+1} based on 2(b) as described in Section 3.3.
   (e) For all $y \in N_x^1$, update $s_{x,y,i}$ based on the message received from y in F_i or from $s_{x,y,i-1}$ if no messages were received from y in F_i.
   (f) Determine whether x can consider itself ready based on the information collected in 2(b) during F_i and F_{i-1} as in Section 3.4.
   (g) If x is ready and no beacon or collision messages have been received and no collisions have been heard for two consecutive frames, then terminate. This implies that all neighbors of x and x itself are ready; thus, x no longer needs to participate.

EasyMAC begins with each node x selecting a random slot $s_{x,0}$ in the first frame F_0. In each successive frame F_{i+1}, a node x may either keep its current slot $s_{x,i}$ from frame F_i or choose a new slot $s_{x,i+1}$ based on the collisions detected by x itself and the other collision-messages received by x from its neighbors during the previous frame F_i.

On termination of EasyMAC in frame F_j, each node x has a slot $s_{x,j}$ which is collision-free, i.e., $s_{x,j} \neq s_{y,j}$ for all $y \in N_x^{1,2}$.

3.1 Message Types

EasyMAC uses two types of messages: a bcn (beacon) message and a col (collision) message; see Table 2. A node x broadcasts $bcn(x)$ in F_i if it has chosen a new $s_{x,i}$ for F_i and neither heard a collision nor detected a conflict during F_{i-1}. Assuming that the *bcn*-message is received clearly, it lets each $y \in N_x^1$ know that the slot $s_{x,i}$ has been chosen by one of its neighbors.

Table 2: EasyMAC messages sent by node x during F_i.

Message	Description
Beacon	$bcn(x)$: x has chosen a new slot for F_i
Collision	$col(x, min, max)$: x detected one or more message collisions and/or conflicts during F_{i-1}, where min is the earliest slot of collisions/conflicts detected and max is the latest slot

The node x sends a collision-message during F_i when it detects collisions and/or conflicts during F_{i-1}. Since x knows when the collisions occurred (i.e., during which time slots) but does not know which of its neighbors were involved, the collision-message format allows x to report the range of slots in which collisions or conflicts occurred (as in Table 2). In $col(x,min,max)$, min is the earliest time slot in the previous frame where x heard a collision or conflict, while max is the latest.

Although the collision-message is primarily used to alert others of collisions, it can also serve as a beacon if x chooses a new slot for F_i because the collision-message would be sent during the new time slot.

3.2 Picking a New Time Slot

At the beginning of frame F_{i+1}, each node x which is not ready chooses a new slot $s_{x,i+1}$ for the frame F_{i+1} based on messages received from neighbors N_x^1 and collisions detected by x itself during F_i; to be specific, if x determines during F_i that it is possible that for some $y \in N_x^{1,2}$, $s_{x,i} = s_{y,i}$, then it will choose a new slot for F_{i+1}. If $y \in N_x^1$, then x discovers this when it receives a message from y or hears a collision during $s_{x,i}$. If $y \in N_x^2$, then x can only determine this possibility by receiving a collision-message from a node $z \in N_x^1 \cap N_y^1$.

Since it is possible for the collision-message from z to collide with a message from another neighbor of x (as in Figure 4), x assumes that if it hears a collision then the colliding message was about a collision caused by x. If x sent a message during the previous frame F_i, then x determines that there is a possibility that $s_{x,i} = s_{y,i}$ for some $y \in N_x^2$; and (if $x \notin Ready$) x picks a new slot $s_{x,i+1}$ for F_{i+1}.

If x has picked a new slot $s_{x,i}$ in F_i, then the only event that will cause x to pick a new slot for F_{i+1} is hearing a collision during the slot $s_{x,i}$. This is because the collision-messages received by x in F_i refer to collisions that occurred in F_{i-1}, when x had a different slot.

When a node x makes the decision to change its slot in F_i, it chooses a random slot $s_{x,i} \neq s_{x,y,i-1}$ for all $y \in N_x^1$. For ease of discussion in subsequent sections, we will assume that $s_{x,i} \neq s_{x,i-1}$ when x picks a new slot for F_i (in practice, there is no need for this).

3.3 Node Behavior

The following section describes the actions taken by nodes upon receiving a message or hearing a collision.

3.3.1 Upon Receiving a Message at x in F_i

Upon the successful reception of a message $m_{y,i}$ (for some $y \in N_x^1$) during $t_{i,j}$ ($s_{y,i} = j$), node x performs the following actions and updates $s_{x,y,i}$ and its own message $m_{x,i+1}$ for F_{i+1} as necessary. When $m_{y,i}$=$bcn(y)$ or $m_{y,i}$=$col(y, min_y, max_y)$:

- [Determine whether $m_{y,i}$ causes a conflict]:
  If ($\exists z \in N_x^1$ such that ($s_{x,z,i-1} = j$ and $z \neq y$)) or $s_{x,i} = j$
  - [Set collision-message for F_{i+1}]:
    If $m_{x,i+1} = null$ or $m_{x,i+1} = bcn(x)$, then let $m_{x,i+1} = col(x, j, j)$
    Else if $m_{x,i+1}$=$col(x, min_x, max_x)$, let $m_{x,i+1}$=$col(x, min\{min_x, j\}, max\{max_x, j\})$
  - [Determine if conflict is caused with x]:
    If $s_{x,i} = j$ and $x \notin ready$, pick a new random slot $s_{x,i+1}$ for F_{i+1}

  Else [No conflict caused; x stores y's claim of j] let $s_{x,y,i} = j$
- [Determine whether collision-message effects x]:
  If $m_{y,i} = col(y, min_y, max_y)$ and $min_y \leq s_{x,i-1} \leq max_y$ and $s_{x,i-1} = s_{x,i}$
  - Pick a new random slot $s_{x,i+1}$ for F_{i+1}
  - If $m_{x,i+1} = null$, let $m_{x,i+1} = bcn(x)$

3.3.2 Upon Hearing a Collision during F_i

When node x hears a collision during $t_{i,j}$, x determines if it needs to choose a new slot for F_{i+1} and prepares to inform its neighbors of the collision during F_{i+1} by taking the following actions and updating its own message $m_{x,i+1}$ for F_{i+1} as necessary.

- [Determine if x must choose a new slot for F_{i+1}]:
  If $x \notin ready$ and $s_{x,i} = s_{x,i-1}$
  - Pick a new slot $s_{x,i+1}$ for F_{i+1}.
  - If $m_{x,i+1} = null$, let $m_{x,i+1} = bcn(x)$
- [Determine if x must report the collision in F_{i+1}]:
  If $t_{i,j} \neq s_{x,i}$ or ($t_{i,j} = s_{x,i}$ and $m_{x,i} \neq null$)
  - If $m_{x,i+1} = null$ or $m_{x,i+1} = bcn(x)$, then let $m_{x,i+1} = col(x, j, j)$
    Else if $m_{x,i+1}$=$col(x, min_x, max_x)$, let $m_{x,i+1}$=$col(x, min\{min_x, j\}, max\{max_x, j\})$

3.4 Determining Readiness

A node x is ready in F_i when x determines that $s_{x,i-2} \neq s_{y,i-2}$ for all $y \in N_x^{1,2}$. When the following three conditions apply, x can consider itself ready:

- x does not detect a collision with one of its neighbors during $s_{x,i-2}$ in F_{i-2}
- x does not receive $col(y, min, max)$ with $min \leq s_{x,i-2} \leq max$ in F_{i-1}
- x does not hear messages colliding or detect a conflict at $s_{x,i-1}$ in F_{i-1}

Any direct collisions with $y \in N_x^1$ are detected in F_{i-2}; and any collisions or conflicts with $z \in N_x^2$ are reported by neighbors during F_{i-1}. Thus, if these three conditions occur, node x will be able to keep the same slot for two full, consecutive frames ($s_{x,i-2} = s_{x,i-1} = s_{x,i}$) and can consider itself ready.

If a conflict or collision occurs between a ready x and node $z \in N_x^{1,2}$ during F_j ($j \geq i$), x would keep its slot ($s_{x,i-2} = s_{x,j} = s_{x,j+1}$) while z picks a new one.

When x becomes ready in F_i, it continues to participate in the EasyMAC algorithm until all $y \in N_x^1$ also enter the ready state. The only difference in its participation is that $s_{x,j} = s_{x,j+1}$ for all $j \geq i$. In future papers, we will discuss what would occur in the event of node failures and the addition of new nodes to the system.

4 EasyMAC vs. LooseMAC

In this section, we first examine the differences in EasyMAC and LooseMAC and then compare their efficiency by running various simulations.

4.1 Differences in Collision Message

The main difference between LooseMAC and EasyMAC is in the way we maintain detailed collision information at each node x about its one and two-neighbors. EasyMAC processes collision-messages at x differently, and its collision-messages sent by x are also different from LooseMAC (as described in Sections 3.1 and 3.3). We summarize the differences by examining how the information is gathered and the format of the messages sent.

4.1.1 Information Gathering Range

With the LooseMAC algorithm, a node x that sends a collision message in F_i is indicating that it heard one or more collisions or conflicts some time between x's previous time slot $s_{x,i-1}$ and x's current slot $s_{x,i}$. If x changed its slot for F_i, the collisions or conflicts could have occurred during a single slot (if $s_{x,i-1} = f$ and $s_{x,i} = 1$), within a range of $2f-1$ slots (if $s_{x,i-1} = 1$ and $s_{x,i} = f$), or any number of slots in between. This variance in the range of the information gathered during LooseMAC leads to some difficulty in analyzing the algorithm as well as some unnecessary slot changes by the nodes.

With EasyMAC, the information on collisions and conflicts reported in F_i was gathered in F_{i-1}. Thus, if a node changed its slot for F_i, it can ignore any collision-messages sent in F_i since that collision information does not apply to its new slot. This is one way that EasyMAC can save some time and messages sent by avoiding some unnecessary slot changes.

4.1.2 Message Format

The format for collision-messages is also different between LooseMAC and EasyMAC. Since a LooseMAC collision-message sent by node x in F_i does not indicate which slots the collisions or conflicts occurred in (beyond being between $s_{x,i}$ and $s_{x,i-1}$), all neighbors N^1_x of x that receive this message and are not yet ready must pick new slots for F_{i+1}.

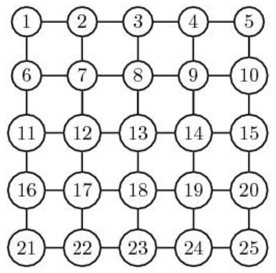
As discussed in Section 3.1, an EasyMAC collision-message indicates the range of slots between which the collisions or conflicts occurred. Upon receiving a collision-message $col(x, min_x, max_x)$ from node x in F_i, a node y can determine whether or not it needs to pick a new slot by comparing $s_{y,i-1}$ to min_x and max_x. If $s_{y,i-1} < min_x$ or $s_{x,i-1} > max_x$, then node y does not need to pick a new slot given the information received from the collision-message. This change can also save time and messages by avoiding some unnecessary slot changes.

4.2 Simulation Scenarios

We compare EasyMAC to LooseMAC [2] via simulations using different frame sizes $f = 1+\Delta_2$, $1.5*(1+\Delta_2)$, and $2*(1+\Delta_2)$ and different network sizes. As Figures 7 and 8 indicate, EasyMAC reduces both the amount of time and the number of messages needed for all nodes to become ready, i.e., to arrive at a valid slot assignment, by reducing some unnecessary slot changes.

The performance of both EasyMAC and LooseMAC improves with the increase in frame size f. We show that EasyMAC performs better than LooseMAC both for tight fit (f close to $1+\Delta_2$) and loose fit (f much larger than $1+\Delta_2$) scenarios.

The algorithms were simulated on grid networks such as the one in Figure 6 ranging in size from 5x5 (25 nodes) to 15x15 (225 nodes). In these types of grid networks, $1+\Delta_2$=13. Separate simulations were run for both the EasyMAC and LooseMAC algorithms with frame sizes of 13, 19, and 26. The averages in Figures 7 and 8 are for 1,000 runs for each f and each nxn network for $5 \leq n \leq 15$.



N^1_{13}={8,12,14,18},
N^2_{13}={3,7,9,11,15,17,19,23},
Δ_2=$|N^{1,2}_{13}|$=12, and
$1+\Delta_2$=13.

Figure 6: A 5x5 Grid Network.

4.3 Results

First, we compare EasyMAC and LooseMAC in terms of the amount of time required for all nodes in the network to become ready. Since we have simulations with various frame sizes, we use time slots as our metric for comparison. Figure 7 shows that EasyMAC (with all of the frame sizes tested) was able to complete execution in fewer time slots than either of the LooseMAC simulations. Interestingly, EasyMAC with frame size of $1.5*(1+\Delta_2)$ takes fewer time slots than EasyMAC with frame size of $2*(1+\Delta_2)$. Even though the larger frame size required fewer frames to complete, after a certain point, the number of frames for completion is not decreased at a faster rate than the increasing number of slots within the frame.

Second, we compare the average number of messages sent until all nodes in the network are ready.

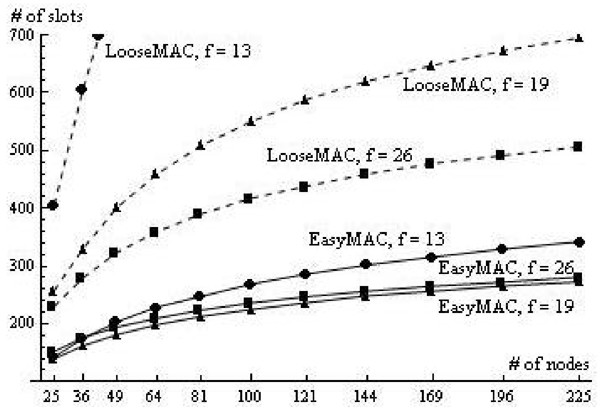


Figure 7: Average number of slots for networks to become ready over 1,000 runs using EasyMAC and LooseMAC with network sizes nxn, $5 \leq n \leq 15$, and frame sizes $f = 1+\Delta_2 = 13$, $1.5*(1+\Delta_2) = 19$, and $2*(1+\Delta_2) = 26$. The LooseMAC data for f=13 and n=15 were too large (> 2,200) to show here.

This metric is used to determine which algorithm potentially requires the nodes to expend less energy. Figure 8 shows that EasyMAC once again performed better than LooseMAC by sending fewer messages per node on average.

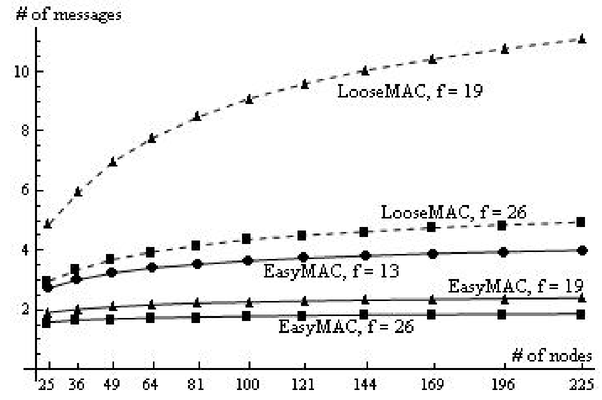


Figure 8: Average number of messages sent per node for networks to become ready over 1,000 runs using EasyMAC and LooseMAC with network sizes nxn, $5 \leq n \leq 15$ and frame sizes $f = 1+\Delta_2=13$, $1.5*(1+\Delta_2)=19$, and $2*(1+\Delta_2)=26$. The LooseMAC data for f=13 were too large to show here (13 for n=5 and 67 for n=15).

In order to determine which of our innovations discussed in Sections 4.1.1 and 4.1.2 caused the greater effect, we kept track of the slot changes avoided due to these changes. Between 60% and 80% of the avoidances were due to the addition of the minimum and maximum effected slots in the collision-message.

From this, we speculate that even in a situation with less time synchronization we should still see the majority of these savings. The min/max used in the collision-messages can be freed from the synchronized frame assumption by referring to slots based on how long ago they occurred as opposed to their slot number in the previous frame. Its effectiveness should not be decreased by doing so. On the other hand, since the information gathering range would need to be altered without the synchronization assumption, we expect that the savings from that (20% to 40%) would be decreased as synchronization is lessened.

5 Conclusion

We have described a new slot assignment algorithm EasyMAC and shown it to be more efficient than LooseMAC [2] in terms of both the number of slots (= f*the number of frames) used to reach a valid slot assignment and the total number of messages sent (hence energy saved). Our future work will expand EasyMAC to handle different timing scenarios (such as, unsynchronized frames and nonuniform frame sizes) and add fault tolerance features to handle the addition and removal of nodes from the network.

We are currently experimenting with several new methods to reduce the unnecessary slot changes and thereby make the EasyMAC algorithm more efficient. First, we let each node transmit the slots held by its neighbors. This allows a node x to keep track of the slots claimed by $y \in N_x^1 \cup N_x^2$. Second, we let the collision message include the slots involved in the conflicts and/or collisions during the previous frame instead of simply using a minimum and maximum of those slot numbers.

References

[1] L. Bao and J.J. Garcia-Luna-Aceves, "A New Approach to Channel Access Scheduling for Ad Hoc Networks," *Mobile Computing and Networking*, pp.210-221, 2001.

[2] C. Busch, M. Magdon-Ismail, F. Sinrikaya, and B. Yener, "Contention-Free MAC Protocols for Wireless Sensor Networks," *Proceedings of the 18th Annual Conference on Distributed Computing (DISC)*, pp.245-259, 2004.

[3] I. Flocchi, A. Panconesi, and R. Silvestri, "Experimental Analysis of Simple Distributed Vertex Coloring Algorithms," *ACM SODA*, pp. 606-615, 2002.

[4] J.J. Jiang, T.-H. Lai, and N. Soundarajan, "On Distributed Dynamic Channel Allocation in Mobile Cellular Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, pp. 1024-1037, 2002.

[5] S. Kundu, "Deadlock-free Distributed Relaxed Mutual Exclusion without Revoke-messages," *Proc. 7th Intern. Workshop on Distributed Computing*, no. 3741, pp. 463-474, 2005.

[6] J. Martin, *Communication Satellite Systems*, Prentice Hall, New Jersey, 1978.

[7] I. Rhee, A Warrier, J. Min, and L. Xu, "DRAND: Distributed Randomized TDMA Scheduling for Wireless Ad-hoc Networks," *MobiHoc 2006*, pp.190-201, 2006.

[8] F. A. Tobagi and L. Kleinrock, "Packet Switching in Radio Channels: Part II - the Hidden Terminal Problem in Carrier Sense Multiple-Access Modes and the Busy-Tone Solution," *IEEE Trans. Commun.*, vol. COM-23, No. 12, pp.1417-1433, 1975.

[9] C. Zhu and M.S. Corson, "A Five-Phase Reservation Protocol (FPRP) for Mobile Ad Hoc Networks," *Wireless Networks 7*, Kluwer Academic Publishers, pp.371-384, 2001.