Noname manuscript No.
(will be inserted by the editor)

Costas Busch · Malik Magdon-Ismail · Fikret Sivrikaya · Bülent Yener

Contention-free MAC Protocols for Asynchronous Wireless Sensor Networks*

the date of receipt and acceptance should be inserted later

Abstract A MAC protocol specifies how nodes in a sensor network access a shared communication channel. Desired properties of a MAC protocol are: it should be *contention-free* (avoid collisions); it should be *distributed* and *self-stabilize* to topological changes in the network; topological changes should be *contained*, namely, affect only the nodes in the vicinity of the change; it should not assume that nodes have a global time reference, that is, nodes may not be time-synchronized. We give a set of TDMA-based MAC protocols for *asynchronous* wireless sensor networks satisfying all of these requirements. The communication complexity, number and size of messages, for the protocols to stabilize is small, poly-logarithmic in the network size.

Keywords MAC protocols · Wireless Sensor Networks · TDMA protocols · Self-stabilization

1 Introduction

1.1 Motivation

Sensor networks are the focus of significant research efforts on account of their diverse applications, that include disaster recovery, military surveillance, health administration and environmental monitoring [2]. A sensor network is comprised of a large number of limited power sensor nodes which collect and

* A preliminary version of the paper appears in the *Proceedings of the 18th Annual Conference on Distributed Computing (DISC 2004)*, LNCS 3704, pp 245-259, Trippenhuis, Amsterdam, the Netherlands, October 2004.

Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, NY 12180, USA
Email: {buschc,magdon,sivrif,yener}@cs.rpi.edu
FAX: +1-518-276-4033

process data from a target domain and transmit information back to specific sites (e.g., headquarters, disaster control centers).

Sensor networks may contain many nodes dispersed with non-uniform density. Here, we consider multi-hop sensor networks in which the nodes share the same wireless communication channel. A medium access control (MAC) protocol specifies how neighbor nodes share the wireless channel. The MAC protocol plays a central role in the performance of a sensor network, since it arbitrates how message traffic is passed through the nodes.

A major cause of power consumption in a sensor network is the transmission of messages. Sensor networks typically have an event-driven communication scheme, such that spatially close nodes sense the same event and try to transmit at the same time. Hence, the message traffic is typically correlated in time and space. This causes *contention*, where nearby sensor nodes attempt to access the communication channel at the same time. Contention causes messages to *collide*, resulting in noise and wasted energy.

A MAC protocol is *contention-free* if messages do not collide during its execution. Contention-free MAC protocols are typically based on time division multiplexing access (TDMA) of the wireless medium, assuming that all the sensor nodes are time-synchronized in some way. However, time synchronization may be infeasible in large scale sensor networks, and it is better not to rely on synchronization in the design of MAC protocols. Here, we consider *asynchronous* sensor networks which do not assume any time synchronization between the nodes.

In order to conserve energy, sensors nodes may turn on and off. Nodes may fail when their batteries are depleted. Further, new nodes may join the network at arbitrary times. Thus, topological changes (node failures and additions), may occur frequently in a sensor network. A MAC protocol should be *distributed* and *self-stabilize* to such kinds of topological changes. The area affected by a topological network change should be *contained* within the vicinity of the change. In this paper, we give contention-free MAC protocols that satisfy these requirements, namely, they are distributed, self-stabilizing, with small containment areas. In addition, the network is asynchronous. These properties are essential to the scalability of sensor networks.

A contention-free MAC protocol should bring the network up from an arbitrary state (where collisions may occur) to a contention-free (collision-free) stable state. Since the protocol is distributed, during the stabilization phase message collisions may occur. We measure the quality of the stabilization phase in terms of the time it takes to reach the stable state, and the amount of control messages exchanged. When the nodes reach the stable state, they use the contention-free MAC protocol to transmit messages without collisions. In the stable state, we measure the efficiency by a node's *throughput* which we define to be the inverse of the smallest time interval between transmissions.

1.2 Network Model

A sensor network with n nodes can be represented by a graph $G = (V, E)$, in which two sensor nodes are connected if they can communicate directly. We assume that all links are bidirectional, and the graph G is undirected and unweighted. The are no other restrictions in the form of the graph.

The metric that we will use for measuring the performance of our algorithms is the *time step* which corresponds to the time it takes to send one message. A message transmitted by a node is received by all of its adjacent nodes in one time step. If two nodes u and w are adjacent and send messages simultaneously, their messages collide at u and w, and they receive noise instead of actual messages. If two nodes u and w are not adjacent and have the same common adjacent node v, then when u and w transmit at the same time, their messages collide in v (*the hidden terminal problem*). We assume for now that nodes can detect such collisions. (In section 6 we justify our assumption by providing a technique for detecting collisions.)

The *k-neighborhood* of a node v, denoted $\Delta_k(v)$, is the set of nodes whose shortest path to v has length at most k. We denote the number of nodes in the k-neighborhood by $\delta_k(v)$ (that is, $\delta_k(v) = |\Delta_k(v)|$), and the maximum k-neighborhood size by δ_k (that is, $\delta_k = \max_v \delta_k(v)$). We refer to 1-neighbors simply as neighbors. We can also define the k-neighborhood of a set of nodes S, denoted $\Delta_k(S)$, as the set of nodes that are at distance at most k away from *some* node in S. Another term that will use in our analysis is $\phi(v)$ which is the maximum 2-neighborhood size among all nodes in v's 2-neighborhood; that is, $\phi(v) = \max_{u \in \Delta_2(v)} \delta_2(u)$. Note that $\phi(v) \leq \delta_2$.

We consider a self-configuring network model where there may occasionally be nodes joining/leaving the network, rather than a completely dynamic or mobile topology. Hence we assume that the topological changes do not significantly affect the neighborhood sizes, and they can be considered as constant factors in the analysis. This allows us to use constant terms such as $\delta_1(u), \delta_2(u)$ rather than functions of time in the analysis.

1.3 Approach

Our approach for designing MAC protocols uses the concept of a *frame* (see Figure 1), which is the basis of TDMA MAC protocols. Each node divides the time into fixed length intervals, called frames. Each frame is further divided into equal-size time *slots*; a time slot corresponds to the time duration required to send one message, namely to one time step. Frames in the same node have the same size (number of slots). However, different nodes can have different frame sizes. The frames do not need to be aligned at the various nodes, and neither do the time slots. Traditionally, in other MAC protocols,

the frames are aligned, have equal sizes, and are time synchronized. However, in our protocols, we do not make these assumptions.

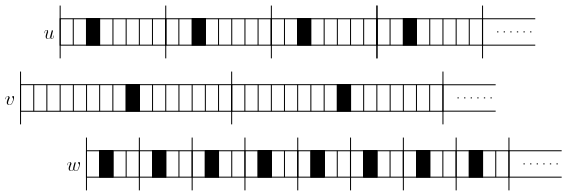

Fig. 1 Frames of three nodes. Frames at different nodes may not be aligned. Solid shaded time slots indicate the selected time slot of each node; longer vertical lines identify the frame boundaries.

The basic idea behind our approach is that each node selects a slot in its own frame which it then uses to transmit messages. The selected slots of any 2-neighbor nodes must not overlap (they should be *conflict-free*), since otherwise collisions can occur. In order to guarantee that slots remain conflict-free in any frame repetitions, the frame sizes in the same neighborhood are chosen to be multiples of each other. In our algorithms the frame sizes are powers of 2. For example, in Figure 1 three neighbor nodes u, v, and w have different frame sizes, and conflict-free time slots. Nodes continuously select slots in the stabilization phase, until they obtain a slot which is conflict-free among all their 2-neighbors, and those slots remain conflict-free thereafter, resulting in a contention-free stable state. When a topological changes occur that cause collisions, the nodes select new slots to restore the contention-free stable state. The throughput of each MAC algorithm is proportional to the inverse of the frame size, since a node can transmit one message during a frame.

1.4 Contributions

Our basic result is a TDMA-based MAC protocol that consists of two phases: the *loose* phase (Algorithm LooseMAC), where nodes set up a preliminary MAC protocol where all frames sizes are equal, followed by the *tight* phase (Algorithm TightMAC), which improves the throughput of the loose phase by tightening locally the frame sizes according to the local neighborhood sizes.

The combined MAC protocol is randomized, distributed and self-stabilizing. For the analysis, we consider an arbitrary network state I which may be the result of arbitrary topological changes. If no topological changes occur after I, for a sufficient amount of time, then the protocol converges to a contention-free stable state I'. We measure the performance of our algorithms in terms of the time and number of control messages exchanged until state I' is reached. Note that between states I and I' collisions may occur, however, after state I' the MAC protocol guarantees no collisions. New topological changes after I' are handled by the self-stabilization process.

In the loose phase, each node chooses frames with equal size $O(\delta_1^3)$. Starting from an arbitrary state I, the loose phase brings the network to a contention-free stable state I'. The time to converge is $O(\delta_1^3 \log n)$, where each node sends $O(\log n)$ control messages each of size $O(\log n)$ bits. Since the frame size is $O(\delta_1^3)$, the node throughput is $O(1/\delta_1^3)$. Table 1 summarizes the performance results of this algorithm and the algorithms below. We assume that when the nodes are initialized they are provided with an upper bound on the value δ_1, which will be preserved even when topological changes occur in the network; the nodes then use the upper bound to compute their frame sizes.

The protocol proceeds to the tight phase which improves the throughput of the nodes and brings the network to a new stable state I''. In the tight phase, each node v chooses a frame of size $O(\phi(v))$, which is related to the local 2-neighborhood size of v. The tightening phase requires $O(\delta_2 \delta_1^3 \log n)$ time until convergence with $O(\log n)$ control messages per node each of size $O(\log n)$ bits. Since the frame of a node is $\phi(v)$, the throughput of any node v is $O(1/\phi(v))$.

Note that in the tight phase, the throughput of a node is related only to the local "density" of the graph in the vicinity of the node ($\phi(v)$), and hence adapts to the varying topology of the network. This is in contrast to the loose protocol where the throughput of each node is the same and proportional to a global parameter of the network (δ_1^3). Note that $\phi(v)$ is bounded by δ_2 which is asymptotically smaller than δ_1^3, since $\delta_2 \leq \delta_1^2$. Message collisions can only occur during the loose phase, while the tight phase uses the contention-free state of the loose phase to exchange messages without collisions. We remark that the loose phase could be stand-alone, since the tight phase is used only for optimization purposes.

An important metric of the self-stabilization process is the *affected area* of a topological change. The smaller the affected area, the better the containment of the self-stabilizing algorithm. Suppose that a subset S of nodes are involved in topological changes (which result to state I), for example the nodes in S power up after being powered down. In our MAC protocol, the only nodes that are affected by the stabilization process are nodes in $\Delta_2(S)$ for the loose phase, and $\Delta_6(S)$ for the tight phase. Thus, the containment area is small and close to the topological changes.

To improve the frame size (throughput) of the loose phase, we present Algorithm pSimpleMAC where the nodes have frames of size $O(\delta_2)$ (we assume that each node has been provided with an upper bound of the value δ_2). The algorithm is parameterized by p, the collision reporting probability, which affects the transmission of control messages. This algorithm has a better throughput over LooseMAC, since $\delta_2 \leq \delta_1^2 < \delta_1^3$. However, it is hard to analyze it formally. In order to assess its performance, we first give its simpler version, Algorithm SimpleMAC, a special case where $p = 1$, for which we can give theoretical estimations of its performance. We show that under certain conditions the stabilization time is $O(\delta_2 \log n)$. We support our theoretical estimates with simulations.

Algorithm	Frame size	Stabilization Time	#Messages	Containment	Formal Analysis
LooseMAC	$O(\delta_1^3)$	$O(\delta_1^3 \log n)$	$O(\log n)$	$\Delta_2(S)$	YES
TightMAC	$O(\phi(\cdot))$ (local δ_2)	$O(\delta_2 \delta_1^3 \log n)$	$O(\log n)$	$\Delta_6(S)$	YES
SimpleMAC	$O(\delta_2)$	$O(\delta_2 \log n)$	$O(\log n)$	$\Delta_2(S)$	Partial

Table 1 Performance of the MAC protocols.

1.5 Computational Complexity of TDMA protocols

It is an NP-complete problem to find the optimal frame sizes with conflict-free slots in them. The polynomial time reduction is from the distance-2 vertex-coloring problem, where each node selects a *valid* color which is different from the color of every other node in its 2-neighborhood [22]. A graph has a valid distance-2 vertex-coloring with k colors if and only if all the nodes can select frames of size k with conflict-free slots in them. The reduction works also for variable frame sizes at the different nodes (the frame and slot selection problem is NP-hard). It can also be verified in polynomial time whether a selection of frames and slots are conflict-free, even if the frames are of variable size (the frame and slot selection problem is NP-complete).

We argue now that the optimal frame size (the smallest possible size of the maximum frame in the graph) has to be at least δ_2; thus our TightMAC and pSimpleMAC protocols are optimal within constant factors. We can show that there are graphs which require at least δ_2 colors in the distance-2 coloring problem, and therefore the corresponding frames have to be of size at least δ_2. For example a clique with δ_2 nodes requires at least δ_2 colors; however, in this simple example, $\delta_2 = \delta_1$. We can construct a better example where $\delta_2 = \omega(\delta_1)$ and requires δ_2 colors. Take k cliques each with k nodes. Order the nodes in each clique $1, 2, \ldots, k$. Connect the ith node of each clique with an edge to the ith node of each other clique. Clearly, in this graph $\delta_1 = 2k - 1$ and $\delta_2 = k^2$, thus $\delta_2 = \omega(\delta_1)$. Further, every node is a 2-neighbor with every other node and thus at least δ_2 colors are required.

1.6 Related Work

MAC protocols are either *contention-based* or *contention-free*. Contention-based MAC protocols are also known as *random access protocols*, requiring no coordination among the nodes accessing the channel. Colliding nodes back off for a random duration and try to access the channel again. Such protocols first appeared as Pure ALOHA [1] and Slotted ALOHA [30]. The throughput of Aloha-like protocols was significantly improved by carrier sense multiple access (CSMA) protocols [20]. Recently, CSMA and its enhancements with collision avoidance (CA) and request to send (RTS) and clear to send (CTS) mechanisms have led to the IEEE 802.11 [39] standard for wireless ad-hoc networks. The

performance of contention based MAC protocols is weak when traffic is frequent or correlated and these protocols suffer from stability problems [33]. As a result, contention-based protocols may result in increased energy consumption which is undesirable for sensor networks.

Our work is related to contention-free MAC protocols. In these protocols, the nodes are following some particular schedule which guarantees collision-free transmission times. Typical examples of such protocols are: frequency division multiple access (FDMA); time division multiple access (TDMA) [21]; and code division multiple access (CDMA) [35]. In addition to these protocols, various reservation based [19] or token based schemes [9,13] are proposed for distributed channel access control. Among these schemes, TDMA and its variants are most relevant to our work. Allocation of TDMA slots is well studied (e.g., in the context of packet radio networks) and there are many centralized [28,34], and distributed [3,10,29] schemes for TDMA slot assignments. All these protocols are either centralized or rely on a global time reference.

Herman and Tixeuil [15] present a distributed TDMA slot assignment algorithm which is based on a fast (with high probability) clustering technique. The time slot allocation is done by the *leaders* in clusters. This is in contrast to our approach where we consider the network as a flat (non-hierarchical) and asynchronous environment. The DE-MAC protocol in [17] uses the TDMA technique together with periodic listen and sleep schedules to avoid major sources of energy wastage. In [4], energy-aware routing and MAC protocols are presented, which are cluster based algorithms controlled by the gateway node of each cluster. A self-organizing algorithm is given in [32] to schedule the activation of links in the network, which is a combined approach for constructing a connected network structure and conflict-free communication schedule.

A synchronous deterministic self-stabilizing TDMA MAC protocol appears in [5]. Moscibroda and Wattenhofer consider the asynchronous coloring [24] and maximal independent set [25] problems for radio networks. In [24], they propose a randomized algorithm that computes a correct vertex coloring using $O(\delta_1)$ colors in time $O(\delta_1 \log n)$ with high probability. They argue that although this does not yield an entirely collision-free communication schedule, it guarantees every sender a constant success probability in each scheduled time slot. These results apply only to unit disk graphs. Our results can be viewed as coloring the 2-neighborhood graph of an *arbitrary* network.

There is considerable work on multi-layered, integrated views in wireless networking. Power controlled MAC protocols have been considered in settings that are based on collision avoidance [26,23, 37], transmission scheduling [12], and limited interference CDMA systems [27]. Some recent work on energy conservation by powering off some nodes is studied in [31,38,8,7]. While GAF [38] and SPAN [8] are distributed approaches with coordination among neighbors, in ASCENT a node decides on its own to be on or off [7]. S-MAC [40] proposes that nodes form virtual clusters based on common sleep

schedules. Sleep and wake schedules are used in [18], but based on energy and traffic rate at the nodes in order to balance energy consumption. A different approach is used in [36], with an adaptive rate control mechanism to provide a fair and energy-efficient MAC protocol. In [16], *CARES* (Connectivity Assuring Randomized Energy-Saving) algorithm is presented, which minimizes power consumption in each sensor node locally while ensuring two global (i.e., network wide) properties: (i) communication connectivity, and (ii) sensing coverage. A sensor node saves energy by suspending its sensing and communication activities according to a Markovian stochastic process.

1.7 Outline of Paper

In Section 2 we give Algorithm LooseMAC and its analysis. We proceed with Algorithm TightMAC in Section 3. Sections 4 and 5 are devoted to Algorithms SimpleMAC and pSimpleMAC. Practical considerations of the implementations of the MAC algorithms concerning the alignment of time slots and the detection of collided messages are given in Section 6. We conclude with a summary in Section 7.

2 Algorithm LooseMAC

We now present Algorithm LooseMAC and its analysis. Each node has the same frame size, which is proportional to δ_1^3 (we assume that the nodes are provided with an upper bound on δ_1). LooseMAC is a randomized algorithm and it guarantees that starting from an arbitrary state the nodes will find their conflict-free slots quickly, with low communication complexity. The algorithm is self-stabilizing with good containment properties.

2.1 Description of LooseMAC

For simplicity of presentation, we will assume that slots are aligned, however, frames do not need to be aligned. Further, a node can detect a collision at the same time that it transmits. In Section 6, we present how to justify/remove these assumptions. Recall that a time step will correspond to the duration of a slot. For notational convenience, given a set of nodes $V = \{v_1, v_2, \ldots, v_n\}$, we will denote node v_i simply as node i.

Algorithm 1 depicts the basic functionality of LooseMAC. Algorithm HandleMessages (Algorithm 2) describes how to handle received control messages or collisions. Both algorithms are running on all nodes simultaneously. For the moment, assume that there are no topological changes in the network and that all the nodes wish to obtain a time slot in their frame. The self-stabilizing version of the algorithm is discussed in Section 2.3. Consider some node i. Node i divides time into frames of size Λ

Algorithm 1 LooseMAC(node i)

1: {**Initialization**}
2: Divide time into frames of size Λ;
3: Define a marking vector $M_i[1, \ldots, \Lambda]$;
4: **for** each k **do**
5: $M_i[k] \leftarrow \perp$; {unmark all entries}
6: ready $\leftarrow$ FALSE;

7: {**Main Part**}
8: **while** not ready **do**
9: Select a new slot σ_i uniformly at random in $1, \ldots, \Lambda$;
10: Mark σ_i (set $M_i[\sigma_i] \leftarrow i$) (and unmark previous selected slot of i);
11: Let t_{σ_i} be the time step of the next occurrence of σ_i;
12: Send control message $\langle \text{beacon}, i \rangle$ at time t_{σ_i};
13: **if** during the next Λ time slots after t_{σ_i}:
($C1$) no collision is detected by i, and
($C2$) no control message $\langle \text{conflict-report}, j \rangle$ is received from any neighbor j
then
14: ready $\leftarrow$ TRUE;

15: {Use Algorithm 2 to handle all received control messages}

Algorithm 2 HandleMessages(node i)

1: **At any time step t that corresponds to slot π:**

2: {**Mark Slot**}
3: **if** control message $\langle \text{beacon}, j \rangle$ is received and $M_i[\pi]$ is unmarked **then**
4: Mark slot π with j ($M_i[\pi] \leftarrow j$) (and unmark previous slot marked with j);

5: {**Report Conflict**}
6: **if** ($C3$) a collision is detected, or
($C4$) a control message is received from j and $M_i[\pi]$ is not marked with j
then
7: Send control message $\langle \text{conflict-report}, i \rangle$ in the next occurrence of slot σ_i;
{this message may be combined with another control message sent from i at σ_i}

which is proportional to δ_1^3 (the exact value of Λ will be determined in the analysis of the algorithm). The main task for node i is to select a conflict-free time slot in its frame. When this occurs we say that the node is "ready", and its local variable ready is set to TRUE.

Initially, node i is not ready. Node i selects randomly and uniformly a slot σ_i in its frame. In σ_i, node i sends a "beacon" message m_i to its neighborhood. Let Z denote the time period consisting of the next Λ time slots after the transmission of the beacon message. During Z, node i decides whether σ_i creates any slot conflicts in its 2-neighbors. If σ_i is conflict-free, then node i keeps slot σ_i and becomes ready. After the node becomes ready it remains ready and doesn't select a new slot (with the exception of when topological changes occur, which is described in Section 2.3). Below we explain how node i can detect whether σ_i creates a slot conflict, and therefore, whether to keep or abandon the selected time slot.

If m_i creates a slot conflict which is detected in some neighbor j, then j responds by transmitting a "conflict-report" message m_j which simply states that j detected a conflict, without specifying which nodes are conflicting and to which slot (see Figure 2). Node j sends m_j during its currently selected slot σ_j. Since the frame length of j is also Λ, the message m_j is sent before the end of time period Z. If m_j is received by i without collisions, i decodes m_j to note that j detected a conflict. For safety, i assumes that the conflict in j was due to σ_i, and i abandons slot σ_i, continuing by selecting another slot in its next frame (Condition $C1$ of Algorithm 1 is not satisfied). If m_j collides at node i, i does not know if m_j was reporting a conflict or not as i cannot decode noise. Again i assumes that the collided message was reporting a conflict, abandons slot σ_i and selects another slot in its next frame (Condition $C2$ of Algorithm 1 is not satisfied). The process repeats until i does not detect any message collisions or does not receive any conflict reports during Z, at which point i becomes ready and stop sending control messages. Note that once a node becomes ready, it will not change its selected slot.

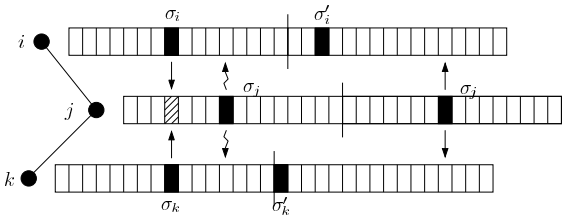

Fig. 2 Execution of the LooseMAC algorithm, where node i chooses a slot that conflicts with k. The shaded slot in j corresponds to a collision detected in j and the waived lines to a conflict-report message sent by j.

To enable a node to detect slot conflicts, the node marks the time slots that are being used by its neighbors. In order to do so Algorithm LooseMAC uses a vector (array) $M[1, \ldots, \Lambda]$ to mark the slots on which neighbors send beacon messages. Consider now node j (the neighbor of i). Initially all the slots of j are unmarked except for the slot σ_j, that is marked as being used by j. Suppose that i sends a message m_i to j during slot π in j's current frame. If node j receives m_i without collisions, and π is unmarked, then j marks π as being used by i. If later i selects another slot, node j will mark the new slot position and unmark the previous position.

Using the marking mechanism, node j can detect slot conflicts as follows. A conflict involving the slot chosen by i can occur for either of the following two reasons:

- *Message collision*: suppose that node i transmits simultaneously with a node k, which is a neighbor of j. this case is observed as a message collision by j. This conflict is detected by j when Condition $C3$ is true in Algorithm HandleMessages running in j.
- *Marking violation*: suppose that node k has chosen a slot σ_k whose corresponding position is marked in j and i chooses slot σ_i which conflicts with σ_k in j's local frame. Then j detects a beacon message

sent by i at a time slot reserved by k. This conflict is detected by j when Condition $C4$ of Algorithm HandleMessages is true running in j.

If either of the above cases occur, node j reacts to it by sending a conflict-report message in its own slot (σ_j) which causes node i to select another slot.

2.2 Analysis of LooseMAC: Correctness and Efficiency

We proceed with the time complexity analysis of the Algorithm LooseMAC. Consider an arbitrary network state I containing non-ready nodes, such that after state I no topological changes occur in the network for a sufficient amount of time (until the nodes become ready). We will give a bound on the time until every node is ready. The time analysis for the self-stabilizing version, together with message complexity and containment, are given in Section 2.3.

We say that *a collision occurs in node i at time t*: if at time t condition $C3$ holds at node i. We say that *a marking-violation occurs in node i at time t*: if at time t condition $C4$ holds at node i. We will say that *a conflict occurs in node i at time t* if either a collision or a marking-violation occurs in i at time t. A node can detect if a conflict occurs in it, and when this happens, according to the algorithm, node i sends a conflict-report message.

The conflicts are caused due to non-ready nodes. Suppose that a conflict occurs in node i at time t. If the conflict is a marking-violation, then this has to be caused by a message sent from a non-ready neighbor conflicting with a ready neighbor (due to condition $C4$). If the conflict is a collision, then from Lemma 1, the collision involves at most one ready neighbor, which has marked the respective slot. Therefore, at least one non-ready neighbor j must have sent a message to cause the collision, and once again the conflict is caused by some non-ready neighbor.

If two nodes are immediate neighbors, they can detect each other's concurrent transmission (see Section 6.4) and refrain from getting ready with overlapping time slots. This fact combined with Lemma 1 below establish our first result that if a pair of nodes are ready, then their slots do not conflict. This is an important property for the correctness of the algorithm, where the chosen slots have to be conflict-free.

Lemma 1 *If two nodes j and k are ready, and both are in the 1-neighborhood of some node i, then the slots of j and k do not conflict (and their respective slots in i are marked).*

Proof Suppose for contradiction that j and k are ready but have chosen conflicting slots. Let π be the corresponding slot in i's frame. Let m_j and m_k be the last beacon messages of j and k respectively. Both messages m_j and m_k must have been received by i without conflicts, since otherwise, node i

would have sent a conflict-report message, that would have caused j and/or k to change their time slots. Therefore, when m_j and m_k are received by i, the respective slots in i are unmarked, and, thus, the slots of j and k are marked without conflicts.

We continue by bounding the probability that a non-ready node causes a conflict in some neighbor node during a period of time.

Lemma 2 *During any time period Z consisting of at most $\alpha\Lambda$ time steps, a non-ready node j causes a conflict in a neighbor i, with probability at most $3(\alpha+1)\delta_1(i)/\Lambda$.*

Proof Let p be the probability that j causes a conflict in i during Z. We want to find an upper bound on p. The conflict in i can be caused by j due to two reasons: (1) j's slot conflicts with a marked slot in i (marking-violations or collisions with ready nodes) or (2) the beacon of j collides with beacons of i's other neighbors in i (collisions with non-ready nodes). Let p_1 be the probability that case 1 occurs and p_2 the probability that case 2 occurs. We have $p \leq p_1 + p_2$.

Note that Algorithm 2 indicates that whenever a new slot is marked for a node, the previous marking corresponding to this node is deleted. Hence node i may have at most one marking in its frame for each of its neighbors, which totals at most δ_1 markings per frame. Consequently, when j selects a slot and transmits, the probability that the corresponding slot is marked in i's frame is bounded by δ_1/Λ. Since the time period Z may overlap with at most $\alpha+1$ frames of i, we have that $p_1 \leq (\alpha+1)\delta_1(i)/\Lambda$.

We now bound p_2. Consider a frame F of j, which overlaps with Z. Frame F may overlap with at most two frames of any other node. Thus, any other node may select at most two slots during the time period of F, and send at most two beacon messages. Then, during F, the transmissions from all nodes in $\Delta_1(i)$ total at most $2\delta_1(i)$. When j selects a new slot in F, the probability that its beacon message collides with these other beacon messages during F is bounded by $2\delta_1/\Lambda$. Again, since Z may overlap with at most $\alpha+1$ frames of j, we have that $p_2 \leq 2(\alpha+1)\delta_1(i)/\Lambda$.

By applying the union bound, we can now bound the probability that a conflict occurs in a node during a time period. This bounds the probability that Condition $C1$ holds, which affects the decision of whether a node remains non-ready.

Lemma 3 *During any time period Z with at most $\alpha\Lambda$ time slots, a conflict occurs in a node i with probability at most $3(\alpha+1)\delta_1^2(i)/\Lambda$.*

Proof A conflict in i is caused by non-ready neighbors of i. From Lemma 2, a non-ready neighbor of i causes a conflict in j with probability at most $3(\alpha+1)\delta_1(i)/\Lambda$. Since there are at most $\delta_1(i)$ non-ready neighbors, and conflicts occurs in i with probability at most $3(\alpha+1)\delta_1^2(i)/\Lambda$, by the union bound.

The next lemma bounds the probability that a conflict-report is received by a node. Essentially, this bounds the probability that Condition $C2$ holds, which affects the decision of whether a node remains non-ready.

Lemma 4 *During any time period Z with at most $\alpha\Lambda$ time slots, a node i receives a conflict report with probability at most $3(\alpha+2)\delta_1^3/\Lambda$.*

Proof Let p be the probability that a node i receives a conflict-report from a neighbor during Z. Let Z' be the time period consisting of Λ slots preceding Z. If neighbor j of i reports a conflict during Z, a conflict must have occurred in j during time period $X = Z' \cup Z$. From Lemma 3, a conflict occurs in j during X with probability at most $3(\alpha+2)\delta_1^2(k)/\Lambda$ (since X has length $\Lambda + \alpha\Lambda = (\alpha+1)\Lambda$). From the union bound, by considering all the neighbors of i we have:

$$p \leq \sum_{j \in \Delta_1(i)} \frac{3(\alpha+2)\delta_1^2(j)}{\Lambda} \leq \frac{3(\alpha+2)\delta_1^3}{\Lambda}.$$

We now give the central result of this section, which shows that if the frame length is chosen proportional to δ_1^3, then every non-ready node becomes ready quickly, starting from an arbitrary state I.

Lemma 5 *If $\Lambda \geq c \cdot \delta_1^3$, for some constant c, then with probability at least $1 - \frac{1}{n}$, starting from an arbitrary state I, every non-ready node will become ready in $2\Lambda \log n$ time steps.*

Proof Suppose that node i is non-ready and selects a new slot σ_i whose first occurrence is at time t. Let Z denote the Λ time steps following t. According to the algorithm, node i will remain non-ready if one of the conditions $C1$ or $C2$ does not hold: (1) a collision occurs in i during Z, or (2) a neighbor sends a conflict-report message during Z. Let p_1 and p_2 be the respective probabilities for cases 1 and 2. By the union bound, the probability p that i remains non-ready is bounded by $p \leq p_1 + p_2$. Using Lemma 3 with $\alpha = 1$, we have $p_1 \leq \frac{6\delta_1^2}{\Lambda}$. Similarly, from Lemma 4, $p_2 \leq \frac{9\delta_1^3}{\Lambda}$. Consequently, $p \leq p_1 + p_2 \leq \frac{c'\delta_1^3}{\Lambda}$, where $c' \geq 15$.

If $\Lambda \geq 4c'\delta_1^3$, then the probability that i remains non-ready is at most $\frac{1}{4}$. Consequently, after $\log n$ independent attempts of selecting a new slot, i will not be ready with probability at most $4^{-\log n} = \frac{1}{n^2}$. Since there are at most n non-ready nodes, by applying the union bound, we obtain that the probability that some node is not ready after $\log n$ attempts, is at most $\frac{n}{n^2} = \frac{1}{n}$. Therefore, all nodes will be ready after $\log n$ attempts with probability at least $1 - \frac{1}{n}$. Every attempt corresponds to the duration of at most 2 frames, since a beacon is sent within Λ time slots after its selection, and then the node listens for Λ time steps. Therefore, every node becomes ready after at most $2\Lambda \log n$ time steps with probability at least $1 - \frac{1}{n}$.

2.3 Self-Stabilizing LooseMAC

Algorithm LooseMAC can be made to adapt dynamically to nodes joining or leaving the network. Here we discuss how the algorithm can be made self-stabilizing and capable of handling arbitrary topological changes in the network.

The situation of leaving the network is simpler than joining the network. A node may leave the network when for example it goes to sleep in order to conserve energy. The situation is more complicated when a node joins the network, due to the hidden terminal problem. Suppose node i enters the network. Nodes j and k that were previously *not* 2-neighbors may now be 2-neighbors because they both become neighbors of i. In this case, j and k may be using the same slot and creating a conflict in i. Thus, j and k may need to reselect slots. To accomplish this, node i has to force nodes j and k to become non-ready. In order to achieve this, when i joins the network, it is in a special status which is called *fresh*. Node i informs its neighbors about its special status by sending control messages. When a neighbor node j receives a control message from i indicating that i is fresh, then j becomes non-ready.

In this section, the *fresh* state should be viewed as a special mode within the non-ready state. In other words, the non-ready state is split into two distinct states *fresh* and *non-fresh*. We still use the term *non-ready* to specify nodes that are either fresh or non-fresh, but not ready. The non-fresh, non-ready state in the self stabilizing version corresponds to the non-ready state in the original version of algorithm LooseMAC.

The self-stabilizing version of LooseMAC, that incorporates fresh nodes, is depicted in Algorithm 3. When a node becomes active it is in fresh mode (phase 1). While i is fresh, it selects a random slot in its frame and transmits a message reporting that it is fresh. It continues to do so in the subsequent frames until it hears no collisions, nor any conflict reports. When this happens, it knows that every one of its neighbors has received its "fresh" message and thus, each neighbor is non-ready. Then, the node switches to the non-fresh, non-ready status (phase 2). Node i now continues with the original LooseMAC algorithm as a non-ready node. The node will remain in phase 2 forever, since nodes may join the network continuously.

We proceed with the analysis of the time it takes for all fresh nodes to become non-fresh.

Lemma 6 *If $\Lambda \geq c \cdot \delta_1^3$, for some constant c, then with probability at least $1 - \frac{1}{n}$, all fresh nodes become non-fresh within $2\Lambda \cdot \log n$ slots.*

Proof We first note that a non-ready node, whether fresh or non-fresh, selects a random slot and transmits a control message once in each of its local frames. However, a fresh node induces less conflicts on its neighbors than a non-fresh node since the fresh nodes have no slot markings in their neighbors' frames and since they do not report conflicts. Hence; as long as δ_1 represents an upper bound on the

Algorithm 3 LooseMAC(node i) (self-stabilizing version)

1: {**Initialization**}
2: Divide time into frames of size Λ;
3: Define the marking vector $M_i[1, \ldots, \Lambda]$;
4: **for** each k **do**
5: $M_i[k] \leftarrow \perp$; {unmark all entries}
6: ready $\leftarrow$ FALSE;
7: fresh $\leftarrow$ TRUE;

8: {**Phase 1: Fresh Node**}
9: **while** fresh **do**
10: Select a slot σ_i uniformly at random in $1, \ldots, \Lambda$;
11: Let t_{σ_i} be the time step of the next occurrence of σ_i;
12: Send control message $\langle \text{fresh}, i \rangle$ at time t_{σ_i};
13: **if** during the next Λ time slots after t_{σ_i}:
($C1$) no collision is detected by i, and
($C2$) no control message $\langle \text{conflict-report}, j \rangle$ is received from any neighbor j
then
14: fresh $\leftarrow$ FALSE;

15: {**Phase 2: Main Part**}
16: **repeat**
17: **while** not ready **do**
18: Select a slot σ_i uniformly at random in $1, \ldots, \Lambda$;
19: Mark σ_i (set $M_i[\sigma_i] \leftarrow i$) (and unmark previous selected slot of i);
20: Let t_{σ_i} be the time step of the next occurrence of σ_i;
21: Send control message $\langle \text{beacon}, i \rangle$ at time t_{σ_i};
22: **if** during the next Λ time slots after t_{σ_i}:
($C1$) no collision is detected by i, and
($C2$) no control message $\langle \text{conflict-report}, j \rangle$ is received from any neighbor j, and
($C3$) no control message $\langle \text{fresh}, j \rangle$ is received from any neighbor j
then
23: ready $\leftarrow$ TRUE;
24: **until forever**

25: {Use Algorithm 4 to handle received control messages **while** not fresh}

Algorithm 4 HandleMessages(node i) (self-stabilizing version)

1: {**Switch to Non-Ready**}
2: **if** a control message $\langle \text{fresh}, j \rangle$ is received **then**
3: ready $\leftarrow$ FALSE;
4: **else**
5: {Use Algorithm 2 to handle the received control message}

number of non-ready neighbors (fresh or non-fresh) of a node, our upper bounds in Lemmas 2, 3 and 4 still hold with the existence of fresh nodes and with the term "non-ready nodes" representing nodes that are either fresh or non-fresh, but not ready.

The rest of the proof is similar to the proof of Lemma 5, however we provide it in full here for completeness. Suppose that node i is fresh and selects a new slot σ_i whose first occurrence is at time t. Let Z denote the Λ time steps following t. According to Algorithm 3 (Phase 1), node i will remain fresh if one of the conditions $C1$ or $C2$ does not hold: (1) a collision occurs in i during Z, or (2) a

neighbor sends a conflict-report message during Z. Let p_1 and p_2 be the respective probabilities for cases 1 and 2. By the union bound, the probability p that i remains fresh is bounded by $p \leq p_1 + p_2$. Using Lemma 3 with $\alpha = 1$, we have $p_1 \leq \frac{6\delta_1^2}{\Lambda}$. Similarly, from Lemma 4, $p_2 \leq \frac{9\delta_1^3}{\Lambda}$. Consequently, $p \leq p_1 + p_2 \leq \frac{c'\delta_1^3}{\Lambda}$, where $c' \geq 15$.

If $\Lambda \geq 4c'\delta_1^3$, then the probability that i remains fresh is at most $\frac{1}{4}$. Consequently, after $\log n$ independent attempts of selecting a new slot, i will still be fresh with probability at most $4^{-\log n} = \frac{1}{n^2}$. Since there are at most n fresh nodes, by applying the union bound, we obtain that the probability that some node is still fresh after $\log n$ attempts, is at most $\frac{1}{n}$. Therefore, all nodes will be non-fresh after $\log n$ attempts with probability at least $1 - \frac{1}{n}$. Every attempt corresponds to the duration of at most 2 frames, since a beacon is sent within Λ time slots after its selection, and then the node listens for Λ time steps. Therefore, every fresh node becomes non-fresh after at most $2\Lambda \log n$ time steps with probability at least $1 - \frac{1}{n}$.

A network is in a stable state when all nodes are ready. Once a network is stable, it remains so until a node joins or leaves the network. We show that from an arbitrary initial state I, the network will stabilize to a stable state I' if no changes are made to the network for a sufficient amount of time. Suppose that $\Lambda \geq c \cdot \delta_1^3$, and let S be the set of non-ready nodes in state I. Let $S_f \subseteq S$ be the set of fresh nodes in state I. Lemma 6 implies that *some* node fails to become non-fresh after $2\Lambda \cdot \log n$ slots with probability at most $\frac{1}{n}$. Similarly, Lemma 5 implies that after additional $2\Lambda \cdot \log n$ slots, *some* node fails to become ready with probability at most $\frac{1}{n}$. Applying the union bound, we have that with probability at least $1 - \frac{2}{n}$, every node is ready after $4\Lambda \cdot \log n$ slots, i.e., w.h.p., the network has reached a stable state.

Consider now the containment area of the algorithm. Nodes that send control messages until stabilization are the *affected* nodes. Only nodes in $\Delta_1(S_f) \subseteq \Delta_1(S)$ become non-ready on account of the fresh nodes. Nodes that are neighbors of non-ready nodes may need to send control messages to report conflicts, thus the affected nodes are all the nodes in $\Delta_2(S)$.

We would like to note here that the fresh state is necessary, since if we only had the ready state, the containment would not be controlled. With only the ready state, the nodes would switch to the non-ready mode when detecting colliding messages. Since in the non-ready mode the nodes select new slots to transmit messages, then the collisions of messages would propagate in the network, making new nodes non-ready. This way the affected area could possibly become the whole network.

Each affected node sends at most $O(\log n)$ control messages until stabilization, since in every frame it sends at most 2 control messages (in the case that the node changes its selected slot). Each message has size $O(\log n)$ bits, since the message consists of the sender's id ($\log n$ bits), fresh or beacon status (1 bit), and conflict report (1 bit). From the discussion above, we obtain the following theorem.

Theorem 1 (Complexity of self-stabilizing LooseMAC**)** *When $\Lambda \geq c \cdot \delta_1^3$, for a constant c, then from an arbitrary state I with non-ready nodes S, the network stabilizes within $4\Lambda \cdot \log n$ slots with probability at least $1 - \Theta(\frac{1}{n})$. The affected area is $\Delta_2(S)$. Each affected node sends $O(\log n)$ messages, of size $O(\log n)$ bits.*

3 Algorithm TightMAC

We consider now the case where nodes have different frame sizes. We present the self-stabilizing Algorithm TightMAC in which each node i has a frame size proportional to $\phi(i)$; recall that $\phi(i) = \max_{j \in \Delta_2(i)} \delta_2(j)$, the maximum 2-neighborhood size among i's 2-neighbors. This algorithm runs on top of LooseMAC. We refer to the frames of TightMAC as "tight", and the frames of LooseMAC as "loose".

3.1 Transition from LooseMAC to TightMAC

A node i entering the network first runs LooseMAC. Once its 2-neighborhood is ready, it uses its selected slot in the loose frame (loose slot) to communicate with its neighbors. Using the loose slot it communicates with its neighbors to compute $\phi(i)$, which determines the size of the tight frame. Then, using the loose slot it communicates with its neighbors to find a conflict-free tight slot (in the tight frame). Then, the node starts using the tight frame's slots (tight slot). The tight frames and the loose frames are interleaved so that a node can switch between them whenever necessary. This enables algorithm TightMAC to be self-stabilizing, due to the self-stabilizing nature of LooseMAC.

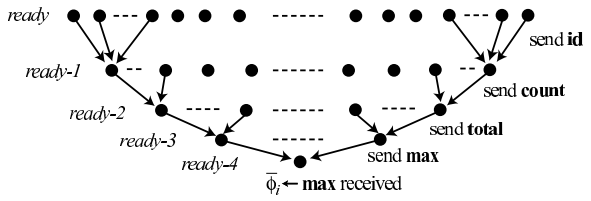

Fig. 3 Demonstration of the ready levels and computation of $\overline{\phi}(i)$ for node i. All *ready* nodes can transmit their ids collision-free, hence each *ready-1* node can compute and transmit its 1-neighborhood size. Then, each *ready-2* node adds those up to obtain an upper bound on its 2-neighborhood size, which is then conveyed to all neighbors. *ready-3* nodes then compute and transmit the maximum of such values received and finally *ready-4* nodes take the maximum of all values received to compute $\overline{\phi}(i)$.

After a node runs LooseMAC, the TightMAC algorithm requires that all nodes in its 2-neighborhood are ready (in order to compute $\phi(i)$). In order to make this possible, we modify the LooseMAC algorithm to incorporate 5 levels of "readiness": ready-0 (or ready); ready-1; ready-2; ready-3; and, ready-4.

Further, we modify the loose frame size to be the smallest power of 2 that is at least the required loose frame size, i.e., $\Lambda = 2^{\lceil \log c\delta_1^3 \rceil}$. A node becomes ready-0 as explained earlier in LooseMAC. A node becomes ready-ℓ (for $\ell > 0$) if all nodes in its neighborhood are ready at level at least $\ell - 1$. We assume that when nodes send messages, they also include their status (fresh, not-ready, ready-ℓ). Thus, when a node becomes ready-ℓ, it sends a message (in its loose slot) informing its neighbors. When a ready-ℓ node has received ready-ℓ messages from all of its neighbors it becomes ready-$(\ell + 1)$ (if $\ell < 4$).

The ready level can also decrease when topological changes occur. If a node is ready-ℓ and hears that one of its neighbors is fresh (i.e. just joined the network), then it changes to non-ready and thus it executes the loose phase. If a node hears that one of its neighbors drops down in ready level, it adjusts its ready level correspondingly. Any node that is not ready-4 is operating on the loose frame. A node starts executing the TightMAC algorithm when it is ready-4.

3.2 Description of TightMAC

Algorithm 5 gives an outline of TightMAC. A node first executes LooseMAC until it becomes ready-4 (Lines 1-3). In the main loop of TightMAC, all the control messages are sent using loose slots, until the node switches to using tight frames.

Algorithm 5 TightMAC(node i)

1: **repeat**
2: Execute LooseMAC(i) and transmit neighborhood information;
3: **until** i becomes ready-4
4: Compute $\phi(i)$;
5: Choose a frame F_i with $|F_i| = 2^{\lceil \log 6\phi(i) \rceil}$;
6: Inform neighbors for the position of F_i relative to i's loose slot; {using the loose slot}
7: Execute FindTightSlot(i);
8: Start using the tight frame and the tight slot;

After a node i becomes ready-4, its first task is to compute $\phi(i)$ (Line 4). To achieve this each ready node has to compute the size of its 2-neighborhood and transmit this information to all its 2-neighbors. This is achieved by using the ready levels. When some node j becomes ready-1, all its neighbors are ready and so by examining the marked slots in its loose frame j can infer its 1-neighborhood $\Delta_1(j)$. Next, j sends $\Delta_1(j)$ to all its neighbors. When a neighbor node k of j becomes ready-2, it must have received the 1-neighborhood information from all its neighbors (including j). With this information, node k is ready to compute the size of its 2-neighborhood $\delta_2(k)$. The next task is to transmit this information to all the 2-neighbors of k. For this purpose we need two additional ready levels. Node k sends $\delta_2(k)$ to all its neighbors. This way any ready-3 neighbor l of k must have received the 2-neighborhood sizes from all its neighbors, including k. Node l then computes the maximum of the

2-neighborhood sizes it has received and sends this information to its neighbors. Therefore, any ready-4 neighbor node i of l must received all the maximum 2-neighborhood sizes of all the 1-neighbors of its 1-neighbors. In other words, node i can compute the maximum 2-neighborhood size of all its 2-neighbors, which is $\phi(i)$.

With the above process, when a node j is ready-1 it sends a message with $\Delta_1(j)$ which has $O(\delta_1(j)\log n)$ bits, and when its ready level gets higher (ready-2 and ready-3) it sends one message per level of size $O(\log n)$. Thus, it sends in total 3 control messages. Note that a message with $O(\delta_1(j)\log n)$ bits might be large for example, it could be $\delta_1(j) = n$. In order to reduce all the message sizes to $O(\log n)$ bits, node j can send $\delta_1(j)$, instead of $\Delta_1(j)$. With this information, each ready-4 node i can compute an upper bound for $\phi(i)$, which is $\overline{\phi}(i) = \max_{j\in\Delta_2(i)} \overline{\delta}_2(j)$, where $\overline{\delta}_2(j) = \sum_{k\in\Delta_1(j)} \delta_1(k)$ is an upper bound on $\delta_2(j)$. The chain of computations for calculating $\overline{\phi}(i)$ is depicted in Figure 3. For simplicity we will present most of the results using $\phi(i)$ and $\delta_2(i)$. The same results hold for $\overline{\phi}(i)$ and $\overline{\delta}_2(i)$ with similar proofs.

After node i has computed $\phi(i)$ it chooses its tight frame size F_i to be the smallest power of 2 that is at least $6\phi(i)$ (Line 5). The slots of the tight frame and loose frame of i are aligned but, since the sizes of the two frames are different their first slots may not be aligned. Node i needs notify its neighbors about the position of F_i relative to its loose slot position (Line 6). This information will be needed for the neighbor nodes to determine whether the tight slots conflict. Next, node i executes FindTightSlot which described in Section 3.3, to compute its conflict-free tight slot in F_i (Line 7). Finally, after the tight slot is computed, node i switches to using its tight frame F_i (Line 8).

To ensure proper interleaving of the loose and tight frames, in F_i, in addition to the tight slot, node i also reserves slots for its loose slot and all other marked slots in its loose frame. This way, the slots used by LooseMAC are preserved and can be re-used even in the tight frame. This is useful when node i becomes non-ready, or some neighbor is non-ready, and i needs to execute the LooseMAC algorithm again.

3.3 Algorithm FindTightSlot

The heart of TightMAC is algorithm FindTightSlot (depicted in Algorithm 6) which obtains conflict-free slots in the tight frames. The tight frames have different sizes at various nodes which depends locally on $\phi(\cdot)$. Different frame sizes can cause additional conflicts between the selected slots of neighbors. For example, consider nodes i and j with respective frames F_i and F_j. Let s_i be a slot of F_i. The *coincidence set* $H_{i,j}(s_i)$ is the set of time slots in F_j that overlap with s_i in any repetitions of the two frames. If $|F_i| \geq |F_j|$, then s_i overlaps with exactly one time slot of F_j. On the other hand, if

$|F_i| < |F_j|$, then $|H_{i,j}(s_i)| = \frac{|F_j|}{|F_i|} > 1$. Nodes i and j *conflict* if their selected slots s_i and s_j are chosen so that $s_j \in H_{i,j}(s_i)$ (equivalently, $s_i \in H_{j,i}(s_j)$).

Algorithm 6 FindTightSlot(node i)

```
 1: SlotFound ← FALSE;
 2: repeat
 3:    Select ← FALSE;
 4:    With probability 1/φ(i): Select ← TRUE;
 5:    if Select then
 6:       Let s_i be a randomly chosen unreserved slot in the first 6δ2(i) slots of F_i;
 7:       Inform neighbors about the position of s_i in F_i; {using the loose slot}
 8:       if no tight frame conflict is reported by any neighbor for the next Λ time slots then
 9:          SlotFound ← TRUE;
10: until SlotFound
```

The task of algorithm FindTightSlot for node i is to find a conflict-free slot in F_i. In order to detect conflicts, node i uses a slot reservation mechanism, similar to the marking mechanism of LooseMAC. When frame F_i is created, node i reserves in each F_i as many slots as the marked slots in its loose frame. This way, when FindTightSlot selects slots, it will avoid using the slots of the loose frame, and thus, both frames can coexist.

Slot selection proceeds as follows. Node i attempts to select an unreserved conflict-free slot in the first $6\delta_2(i)$ slots of F_i. We will show that this is always possible. Node i then notifies its neighbors of its choice using its loose slot. Each neighbor j then checks if s_i creates any conflicts in their own tight slots, by examining whether s_i conflicts with reserved slots in j's tight and loose frame. If conflicts occur, then j responds with a conflict report message for the tight slot (again in its loose slot).

Node i listens for Λ time slots. Recall that Λ is the duration of the loose frame. If a neighbor j detected a conflict, it reports it during this period, since the conflict is reported on the loose slot which occurs no later that Λ time slots. Otherwise, if not conflict is reported, the neighbor j marks this slot as belonging to node i (un-marking any previously marked tight slot for node i). If node i receives a conflict report for the tight slot, the process repeats, with node i selecting another tight slot. If node i does not receive a conflict report, then it fixes this tight slot. Note that this communication between node i and its neighbors can occur because all its neighbors are at least ready-2 (this communication occurs in the loose slots).

In the algorithm, a node chooses to select a new tight slot with probability $1/\phi(i)$, so not many nodes attempt to select a tight slot at the same time, which increases the likelihood that the selection is succesful. This is what allows us to show stabilization with low message complexity, even with small frame sizes.

The intuition behind the frame size choice of approximately $6\phi(i)$ is as follows. Take some node j which is a 2-neighbor of node i. Node j chooses a tight slot in the first $Z = 6\delta_2(j)$ time slots of its tight frame F_j. Node i has frame size at least as big as Z, since $\phi(i) \geq \delta_2(j)$. Thus, node i cannot have more than one selected tight slot in Z. This implies that the number of potentially conflicting slots for j during Z are bounded by the 2-neighborhood size of j. This observation is the basis for the probabilistic analysis of the algorithm.

Lemma 7 *Let node $j \in \Delta_2(i)$ select tight slot s_j; s_j does not cause conflicts in any node of $\Delta_1(i)$ with probability at least $\frac{1}{2}$.*

Proof The neighbors of j have reserved at most $2\delta_1(j)$ slots in j (one loose and one tight slot). Therefore, node j chooses its tight slot s_j from among $6\delta_2(j) - 2\delta_1(j) \geq 4\delta_2(j)$ unreserved slots in its tight frame F_j. This slot can cause conflicts only in neighbors of i that are in $S = \Delta_1(i) \cap \Delta_1(j) \subseteq \Delta_1(j)$. The nodes that can possibly reserve slots in the frames of the nodes in S can be no more than the 2-neighbors of j. Each such node can mark at most two slots (one loose and one tight slot); so, at most $2\delta_2(j)$ time slots which can possibly conflict with s_j are reserved in *all* the nodes in S. Therefore, there are at least $2\delta_2(j)$ slots available to j, of the $4\delta_2(j)$ chosen from; hence the probability of causing no conflict is at least $\frac{1}{2}$.

Lemma 8 *For node i, during any period of Λ time slots, no conflicts from tight slots occur in $\Delta_1(i)$ with probability at least $\frac{1}{2}$.*

Proof Let Y be the period of Λ time slots. A conflict is caused during Y in $\Delta_1(i)$ by any slot selection of nodes in $\Delta_2(i)$. By construction, a slot selection by node $j \in \Delta_2(i)$ occurs with probability $1/\phi(j)$. From Lemma 7, a slot selection of j causes conflicts in $\Delta_1(i)$ with probability at most $1/2$. There are at most $\delta_2(i)$ nodes similar to j. Let q be the probability that any of them causes a conflict during Z in $\Delta_1(i)$, then $q \leq \delta_2(i)/(2\min_{\{j \in \Delta_2(i)\}} \phi(j))$. Since for any $j \in \Delta_2(i)$, $\phi(j) \geq \delta_2(i)$, we have that $q \leq \frac{1}{2}$; hence the probability of no conflicts is $1 - q \geq \frac{1}{2}$.

Lemma 8 implies that every time i selects a slot in its tight frame, this slot is conflict-free with probability at least $\frac{1}{2}$. Since i selects a time slot with probability $1/\phi(i)$ in every loose frame, in the expected case i will select a slot within $O(\phi(i))$ repetitions of the loose frame. We obtain the following result.

Corollary 1 *For some constant c, within $\phi(i) \cdot c\Lambda \log n$ time slots, a ready-4 node successfully chooses a conflict-free tight slot in F_i, with probability at least $1 - \frac{1}{n^2}$.*

3.4 Complexity of TightMAC

We say that the network is *stable* if all nodes in the network have selected conflict-free tight slots. Consider a state I' at which every node becomes ready in LooseMAC. Then, the nodes execute algorithm TightMAC. Consider a node i. Suppose that the algorithm uses $\overline{\phi}(i)$ and $\overline{\delta}_2(i)$ for practical considerations (smaller message sizes). Node i sends 3 messages of size $O(\log n)$ bits so that itself and its neighbors become ready-4 and compute $\overline{\phi}(\cdot)$. From Corollary 1, with high probability, node i requires $O(\overline{\phi}(i) \cdot \Lambda \log n)$ time slots to select a conflict-free tight slot. Following an analysis similar to Corollary 1, and since $\delta_2 \geq \overline{\phi}(i)$, we obtain the following theorem.

Theorem 2 (Complexity of TightMAC) *In the tight phase, the network stabilizes within $O(\delta_2 \Lambda \log n)$ time slots, with probability at least $1 - \Theta\left(\frac{1}{n}\right)$. Each node sends $O(\log n)$ messages, of size $O(\log n)$ bits.*

We can now determine the performance of the combined loose and tight phase. Starting from an arbitrary initial state I, if no changes occur in the network after I, the network first reaches a contention-free state I' at the end of the loose phase (where every node is ready), and then it reaches a stable state I'' for the tight phase. Let S be the non-ready nodes in state I. By Theorem 1, with high probability, LooseMAC requires $O(\Lambda \log n)$ time slots for all nodes to become ready. When a fresh node i arrives the nodes in $\Delta_5(i)$ drop their levels, hence the affected area is at most $\Delta_6(i)$. Therefore, from Theorem 1 and Corollary 2, since $\Lambda = O(\delta_1^3)$, we obtain:

Corollary 2 (Combined complexity of loose and tight phase) *From an arbitrary initial state I with non-ready nodes S, the network stabilizes within $O(\delta_2 \delta_1^3 \log n)$ time slots, with probability at least $1 - \Theta\left(\frac{1}{n}\right)$. The affected area is $\Delta_6(S)$ and each affected node sends $O(\log n)$ messages, of size $O(\log n)$ bits.*

4 Algorithm SimpleMAC

The frame sizes used by LooseMAC, $\Lambda = O(\delta_1^3)$, may be practically too large and asymptotically higher than the optimal δ_2, causing an under-utilization of the wireless channel. To improve the stabilization of the loose phase, we propose a simple and more practical algorithm, SimpleMAC, based on the same ideas presented earlier. Though the algorithm is simpler, its theoretical analysis is harder because of the so-called *propagation effect*, which will be explained later. Hence, we first present an analysis ignoring this effect, and then verify by simulation that our analysis does not differ much by ignoring the propagation effect. In Section 5, we will present pSimpleMAC, a more general version of SimpleMAC, which uses a *probabilistic collision reporting* technique to prevent the propagation effect. Studying SimpleMAC first, makes the understanding of pSimpleMAC much easier.

The basic idea behind SimpleMAC is similar to Algorithm LooseMAC. The frame size Λ is the same for all nodes in the network, but it can be set to a smaller value (proportional to δ_2). The basic difference is in the way in which nodes *report collisions* in the hidden-terminal case. Recall that in Algorithm LooseMAC, nodes always transmit in their own time slot, either for acquiring a time slot or to report a collision. When node j receives a garbled message due to concurrent transmission of its neighbors i and k, it transmits in its own time slot σ_j to report this collision (see Figure 2.1). However, in Algorithm SimpleMAC, each collision is reported in the next occurrence of the slot in which the collision has occurred. This is depicted in Figure 4 where node i detects a collision by concurrent transmissions from nodes j and k (the first shaded slot), and it reports this conflict exactly after Λ time slots (the second shaded slot).

To perform such conflict reports, each node i keeps an additional vector $\mathcal{C}_i$ for marking slot conflicts. The advantage of this approach is that the status of a time slot (conflict-free or not) depends only on the events occurring at this slot. If a node receives no other transmission *in its own time slot* for 2 consecutive frames, it concludes that the slot is conflict-free. In contrast, in Algorithm LooseMAC a node should receive no conflict reports or collisions in each of Λ consecutive time slots. This modification increases the chances of obtaining a conflict-free time slot, for a given frame size.

We now provide a detailed description of Algorithm SimpleMAC (see Algorithm 7). The algorithm is composed of *rounds*, each lasting at most Λ consecutive time slots. A new round starts for node i whenever it selects a new random slot σ_i. We introduce two local variables (flags) for each node, *ready* and *collision*. The *ready* flag is identical in effect to the "ready" state in Algorithm LooseMAC. Each node initializes its ready flag to FALSE, and collision flag to TRUE prior to executing the algorithm. At the end of each round, the collision flag states whether the node has collided in the selected time slot or not. If the collision flag remains FALSE after the second time a node transmits in its slot, then the node becomes "ready", i.e. it permanently obtains the current time slot.

Until node i becomes ready, it performs the following operations. If the collision flag is TRUE, then the node randomly selects a time slot σ_i. At time slot σ_i, the node broadcasts a *beacon message*. It also receives broadcasts from other nodes in each time slot of the frame, and marks the time slots in which two or more neighbors are transmitting. We assume that these will be identified as garbled messages arriving to node i, because of destructive interference.

In its selected time slot σ_i, node i checks if there is any transmission by any neighbor node. If so, i sets its collision flag to TRUE, stating that it has to select a new time slot, and hence start a new round. A necessary and sufficient condition for a node to become ready is to transmit twice in the same time slot and not detect any conflict. When node i transmits in σ_i for the first time without having a collision, it means that none of its immediate neighbors selected that time slot. Exactly Λ time slots

Algorithm 7 SimpleMAC(node i)

```
1: {Initialization}
2: Divide time into frames of size Λ;
3: Define a marking vector M_i[1, ..., Λ];
4: Define a conflict report vector C_i[1, ..., Λ];
5: for each k do
6:    M_i[k] ← ⊥;  {unmark all entries}
7:    C_i[k] ← FALSE;
8: ready ← FALSE;   collision ← TRUE;

9: {Main Part}
10: while not ready do
11:    if collision then
12:       Select a new slot σ_i uniformly at random in 1, ..., Λ;
13:    Let t_{σ_i} be the time step of the next occurrence of σ_i;
14:    Send control message ⟨beacon, i⟩ at time t_{σ_i};
15:    if sensed any other transmission at time t_{σ_i} then
16:       collision ← TRUE;
17:    else
18:       ready ← not collision;
19:       collision ← FALSE;

20: {Use Algorithm 8 to handle all received control messages}
```

Algorithm 8 HandleMessages-SimpleMAC(node i)

```
1: At any time step t that corresponds to slot π:

2: {Mark Slot / Conflict}
3: if control message ⟨beacon, j⟩ is received then
4:    if M_i[π] = ⊥ then
5:       Mark slot π with j (M_i[π] ← j) and unmark previous slot marked with j;
6:    else if M_i[π] ≠ j then
7:       Set conflict flag for slot π (C_i[π] ← TRUE);
8: else if collision detected then
9:    Set conflict flag for slot π (C_i[π] ← TRUE);

10: {Report Conflict}
11: if C_i[π] then
12:    Send control message ⟨conflict-report, i⟩;
13:    C_i[π] ← FALSE;
```

later, on the next occurrence of σ_i, i transmits again, and if this is also collision-free, then i concludes that none of its two-neighbors has selected σ_i. Hence i becomes ready. The *ready* and *collision* flags together achieve this effect.

On the other hand, if i's first transmission in σ_i has conflicted with one of its 2-neighbors, k (the hidden terminal problem), then the common neighbor of i and k, say j, would *mark* that time slot by setting $\mathcal{C}_j[\sigma_i]$. Then j would report it in slot σ_i of the next time frame, causing a collision with i's second transmission in σ_i, and forcing i to select a new slot (see Figure 4).

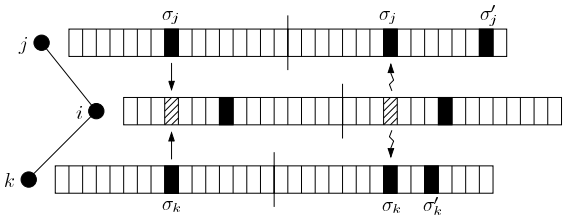

Fig. 4 Execution of Algorithm SimpleMAC, where the shaded slot corresponds to a collision and the waived lines to a conflict report message.

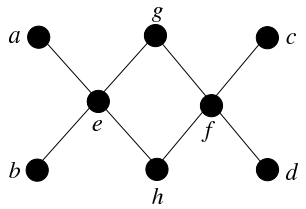

Fig. 5 A simple topology to demonstrate the propagation effect.

4.1 The Propagation Effect

It is easiest to explain the propagation effect with an example. Consider the simple topology in Figure 5. Assume that nodes a, b, c and d all select the same time slot σ. Let the global time be $t = \sigma$ at the instant when a, b, c, d all transmit for the first time in this slot. Then e detects a collision by the signals from a and b; f detects a collision by the signals from c and d, and they both report these conflicts at time $t = \sigma + \Lambda$, which corresponds to the next occurrence of slot σ. Thus at $t = \sigma + \Lambda$, the transmissions of e and f (which are actually collision reports) collide on both g and h. Since g and h have no means of differentiating an actual collision (slot conflict) from the collision of conflict report messages, they transmit conflict report messages at time $t = \sigma + 2\Lambda$. Similarly, now both e and f spuriously detect a conflict (caused by the conflict reports of g and h) and report it at $t = \sigma + 3\Lambda$. This chain of messages goes on indefinitely and makes the time slot σ useless in this neighborhood. Moreover, nodes involved in the chain waste significant amount of energy by continuously sending unnecessary conflict report messages. Another important problem that arises because of this propagation of conflict reports is that the nodes in this neighborhood will not be able to terminate the algorithm as they detect collisions in every time frame due to the slots flooded by the propagation effect.

In the next subsection, we provide a theoretical analysis of the SimleMAC algorithm, however we ignore the propagation effect in order to make the analysis tractable. Next, we show by simulations that ignoring the propagation effect in the analysis does not have a big impact, as the expectations from the analysis closely match the results of the simulation, which implicitly take the propagation effect into account.

4.2 Analysis of SimpleMAC

In this section we basically prove the following theorem.

Theorem 3 *Ignoring the effect of conflict report propagation, if $\Lambda \geq 2\delta_2$, all non-ready nodes become ready within $2\Lambda \cdot \log n$ time slots, with probability at least $1 - \frac{1}{n}$.*

Proof Assume that node i is not *ready* at the beginning of a round. At the end of this round, node i will not be ready if and only if it picks a time slot equal to the time slot selected by some other node in its 2-neighborhood. Let R_i be the number of ready nodes in node i's 2-neighborhood. Thus, $\delta_2(i) - R_i$ nodes are not ready in i's frame, and $\Lambda - R_i$ time slots are available (unoccupied). Node i will be ready if it picks one of these $\Lambda - R_i$ available time slots and all the other $\delta_2(i) - R_i$ nodes do not pick that slot. Let p_i be the probability of this event. Then, using the facts that $(1-x)^n \geq 1 - nx$ and $\delta_2(i) \geq R_i$, we have

$$\begin{aligned} p_i &= \left(\frac{\Lambda - R_i}{\Lambda}\right)\left(\frac{\Lambda - 1}{\Lambda}\right)^{\delta_2(i)-R_i} \\ &= \left(1 - \frac{R_i}{\Lambda}\right)\left(1 - \frac{1}{\Lambda}\right)^{\delta_2(i)-R_i} \\ &\geq \left(1 - \frac{R_i}{\Lambda}\right)\left(1 - \frac{\delta_2(i) - R_i}{\Lambda}\right) \\ &= 1 - \frac{\delta_2(i)}{\Lambda} + \frac{R_i(\delta_2(i) - R_i)}{\Lambda^2} \\ &\geq 1 - \frac{\delta_2(i)}{\Lambda} \end{aligned}$$

So with probability at least $\left(1 - \frac{\delta_2(i)}{\Lambda}\right)$, a node will become ready in a given round. Therefore after α rounds, the probability that a node is not ready is at most $\left(\frac{\delta_2(i)}{\Lambda}\right)^{\alpha}$, since random choices are independent. Let $q_i(\alpha)$ denote the probability that node i is not ready after α rounds:

$$q_i(\alpha) \leq \left(\frac{\delta_2(i)}{\Lambda}\right)^{\alpha}$$

Using the union bound on the sum of probabilities, we get

$$Q(\alpha) \leq \sum_i \left(\frac{\delta_2(i)}{\Lambda}\right)^{\alpha} \leq n\left(\frac{\delta_2}{\Lambda}\right)^{\alpha}$$

where $Q(\alpha)$ is the probability that some node is not ready after α rounds. If we define *success* of the algorithm as having conflict-free time slots assigned to all nodes in the network, then $Q(\alpha)$ can also be stated as the probability of *failure* after α rounds. Assume that we are given a bound ε on the

probability of failure. In order to have $Q(\alpha) \leq \varepsilon$, it suffices that

$$n\left(\frac{\delta_2}{\Lambda}\right)^{\alpha} \leq \varepsilon,$$

or that $\alpha \geq \frac{\log(\varepsilon/n)}{\log(\delta_2/\Lambda)}$.

By choosing ε arbitrarily small, we can ensure success with probability 1. For example, if we choose $\varepsilon = \frac{1}{n^{\gamma}}$, for some constant γ, then *all nodes* are ready with probability $\geq 1 - \frac{1}{n^{\gamma}}$ after α rounds, where

$$\alpha = \frac{(\gamma+1)\log n}{\log \frac{\Lambda}{\delta_2}}$$

The theorem follows by choosing $\gamma = 1$, $\Lambda \geq 2\delta_2$, in which case $\alpha = 2\log n$ rounds suffices, which corresponds to at most $2\Lambda \log n$ time slots.

Note that at the beginning of the proof we stated that $\Lambda - R_i$ slots are available in i's frame. If we do not ignore the propagation effect, $\Lambda - R_i$ slots are not available as some will be used up by conflict report propagations, and the proof breaks down. Theorem 3 signifies that without the propagation effect Algorithm SimpleMAC can work with much smaller frame sizes, $O(\delta_2)$, as compared to LooseMAC which uses frames with size $O(\delta_1^3)$ (see Lemma 5).

4.3 Simulation Study of SimpleMAC

We now investigate the impact of ignoring the propagation effect in the above analysis, and test if the upper bound on the running time of Theorem 3, still holds when the propagation effect is also taken into account. We use extensive set of simulations to measure the actual running time of the algorithm (which also includes the propagation effect).

We use random networks of 500 to 1000 nodes, which are scattered in a unit square. In order to study various node densities, we use a fixed transmission radius $r = 0.1$ units for each node in the network. Each data point used in the plots of this section is an averaged quantity over 100 random networks. In the numerical computations and simulations of this section, we set the frame size to $2\delta_2$ for each network. This is accomplished by pre-computing the maximum 2-neighborhood size for each random network generated. For each network size n, we set $\gamma = \log_n 1000$ in order to ensure a high probability result for the analysis, such that Theorem 3 holds with 99.9% confidence (i.e. with probability $1 - 1/1000 = 0.999$).

Figure 6 presents the results of both theory and simulation. We observe that for $n = 500$ and $n = 600$, the actual running times are slightly higher than the theoretical results. However, as the network size increases, the simulation results stay well below the theoretical upper bound.

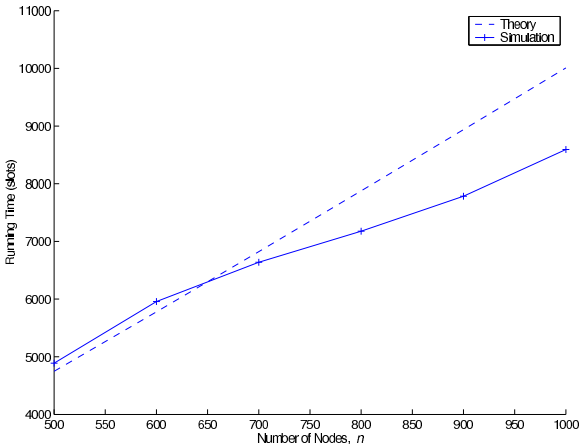

Fig. 6 Running time of SimpleMAC for different network sizes. *Theory* plots the numerical computations following the analysis of section 4.2. Though this analysis ignores the *propagation effect*, the simulation results verify that the analysis still holds asymptotically; as the network size grows, the theoretical results set an upper bound on the actual running time of the algorithm.

In the next section we propose a modification on SimpleMAC algorithm that addresses the propagation effect, and also decreases the running time of the algorithm, as supported by simulation results.

5 Algorithm pSimpleMAC - Probabilistic Collision Reporting

As explained in Section 4.1, deterministically reporting every collision may result in chains of redundant messages. Here, we present Algorithm pSimpleMAC which introduces the *probabilistic collision reporting* idea to break such chains. Probabilistic collision reporting is based on this observation: a node reports a conflict at time slot σ with a probability based on the number of consecutive collisions at σ. As the number of consecutive collisions detected at slot σ increases, it is more likely that these collisions correspond to a slot conflict, rather than the collision of conflict report messages. Spurious conflict report messages may still be generated occasionally, as in Algorithm SimpleMAC algorithm, but the probability of having long chains due to these spurious messages is dramatically reduced in Algorithm pSimpleMAC algorithm.

In pSimpleMAC (see Algorithm 9), upon detecting a collision at time t, node i reports this collision at $t + \Lambda$ with probability p_{report}; otherwise, with probability $1 - p_{report}$, it remains silent and listens the channel at $t + \Lambda$. If node i again receives a collision at this instant, which is the next occurrence of this slot, then i reports it at $t + 2\Lambda$ with probability $2 \times p_{report}$. Depending on the protocol parameter p_{report}, the probability of reporting a conflict will soon reach 1 as long as neighbors continue to collide on i at that slot. Hence, a slot conflict between hidden terminals is always reported after at most $1/p_{report}$ time frames. Note that when $p_{report} = 1$, Algorithm pSimpleMAC reduces to the SimpleMAC.

Algorithm 9 pSimpleMAC(node i)

{**Initialization**}
Divide time into frames of size Λ;
Define a marking vector $M_i[1, \ldots, \Lambda]$;
Define a conflict report vector $\mathcal{C}_i[1, \ldots, \Lambda]$;
for each k **do**
  $M_i[k] \leftarrow \perp$; {unmark all entries}
  $\mathcal{C}_i[k] \leftarrow 0$;
ready $\leftarrow$ FALSE; clear $\leftarrow 0$;
Select a new slot σ_i uniformly at random in $1, \ldots, \Lambda$;

{**Main Part**}
while not ready **do**
  Let t_{σ_i} be the time step of the next occurrence of σ_i;
  Send control message $\langle \text{beacon}, i \rangle$ at time t_{σ_i};
  if sensed any other transmission at time t_{σ_i} **then**
    Select a new slot σ_i uniformly at random in $1, \ldots, \Lambda$;
    clear $\leftarrow 0$;
  else
    if clear++ $\geq 1/p_{report}$ **then**
      ready $\leftarrow$ TRUE;

{Use Algorithm 10 to handle all received control messages}

Algorithm 10 HandleMessages-pSimpleMAC(node i)

At any time step t that corresponds to slot π:

{**Mark Slot / Conflict**}
if control message $\langle \text{beacon}, j \rangle$ is received **then**
  if $M_i[\pi] = \perp$ **then**
    Mark slot π with j ($M_i[\pi] \leftarrow j$) and unmark previous slot marked with j;
  else if $M_i[\pi] \neq j$ **then**
    $\mathcal{C}_i[\pi]$++; {increment collision counter for slot π}
else if collision detected **then**
  $\mathcal{C}_i[\pi]$++; {increment collision counter for slot π}

{**Report Conflict**}
with probability $\mathcal{C}_i[\pi] \times p_{report}$
  Send control message $\langle \text{conflict-report}, i \rangle$;
  $\mathcal{C}_i[\pi] \leftarrow 0$; {reset collision counter for slot π}

We now give some more details of the algorithm. For marking the conflicting time slots, each node i has again a local vector $\mathcal{C}_i$ of size Λ. However, $\mathcal{C}_i$ is an integer vector in pSimpleMAC, so that it not only indicates a collision but also stores the number of recent consecutive collisions at each time slot. Initially $\mathcal{C}_i[t] = 0$ for all $t = 0..\Lambda - 1$. When node i receives a garbled transmission at its local slot t_i, it increments $\mathcal{C}_i[t_i]$ by 1. Node i resets $\mathcal{C}_i[t_i]$ to zero when it does not detect any collision in t_i, or after it sends a collision report message in t_i. Each node has also a *clear* counter initialized to zero, which replaces the *collision* flag in SimpleMAC.

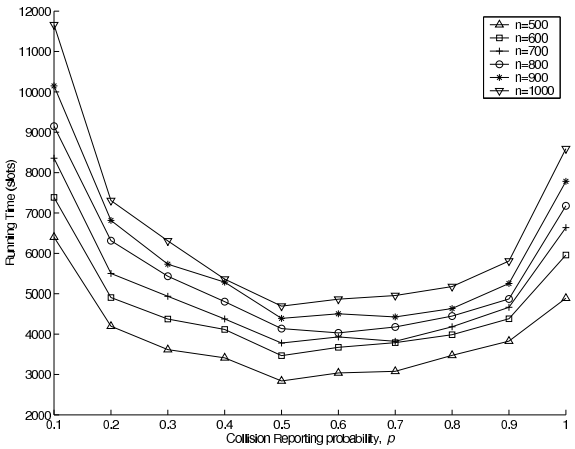

Fig. 7 Running time of pSimpleMAC with varying collision reporting probabilities, shown for 6 different network sizes. For each network size, $p = 0.5$ is approximately the optimal value that minimizes the running time.

In its selected time slot σ_i, node i checks if there is any transmission by any neighbor node. If so, the node randomly selects another time slot, and resets its *clear* counter to 0. Otherwise, the node uses the current value of its *clear* counter to decide if it can safely obtain σ_i permanently, and then increments the *clear* counter. Note that a neighbor j of node i reports a collision at time slot σ_i with probability $\mathcal{C}_j[\sigma_i] \times p_{report}$. Therefore j deterministically reports the collision after receiving $1/p_{report}$ collisions at σ_i. Thus, if node i's *clear* counter has reached $1/p_{report}$, σ_i is deterministically collision-free in the 2-neighborhood, hence, node i becomes *ready*, by obtaining slot σ_i permanently.

5.1 Simulation Study of pSimpleMAC

We now compare the running times of Algorithms SimpleMAC and pSimpleMAC, to see the effect of probabilistic collision reporting on the algorithm performance. In order to do this comparison, we parameterize with respect to the collision reporting probability and obtain the average running times for different values of p_{report}, where the case $p_{report} = 1$ is identical to the Algorithm SimpleMAC, as stated earlier. This also allows us to study the optimal value for p_{report} (the one that minimizes the running time of the algorithm).

We use the same set of networks used in section 4.3, where the number of nodes is varied from 500 nodes to 1000 nodes in an area of unit square, and the transmission radius is set as $r = 0.1$. The frame size is again set to $\Lambda = 2\delta_2$ for each random network generated. Figure 7 clearly demonstrates that the optimal value for p_{report} is around 0.5, regardless of the network size or density. Moreover, for this setting, pSimpleMAC converges much faster than SimpleMAC. This is mainly due to the fact that some time slots get flooded and become useless by the propagation effect in SimpleMAC, which in turn decreases the availability of conflict-free slots. On the other hand, very low values of p_{report} causes an

increase on the running time, since there is no further gain due to the elimination of the propagation effect, but the number of time frames required to obtain a slot increases inversely with p_{report}. E.g., for $p_{report} = 0.1$, a node needs to wait for at least 10 time frames before it can obtain a conflict-free slot, whereas in algorithm SimpleMAC a node needs to wait for only 1 frame.

6 Practical Considerations

In this section, we discuss some of the issues in a real sensor network environment which we have ignored in our analysis up to now. Essentially, we show that the simplifying assumptions we have made for easier analysis analysis and presentation can be removed, and we discuss the effect of relaxing these assumptions.

6.1 Slot Misalignment

We have assumed so far that the time slots of the nodes are aligned (even though the start times of the frames may not be aligned). In fact, this assumption can be dropped, at the expense of just a constant factor increase in the frame size. Since the slot sizes are equal, each time slot of a node v may overlap with at most two slots at some neighbor node u. Thus, whenever v acquires a time slot σ_v, u may actually mark two time slots in its own frame corresponding to σ_v in v's frame. On the other hand, if u detects a collision in any one of those two time slots, then it reports a collision forcing v to refrain from using σ_v. In effect, the previous analysis of the algorithms remain valid with the frame sizes doubled.

6.2 Clock Skew

All our discussion applies when there is no difference between the local clock speeds at the nodes. In practice, there may be a very small clock skew that can cause the relative times to change. One approach to addressing this problem in our algorithms, is to run a skew-correction algorithm (for example [6]) on top of our protocol. Another solution, which illustrates the value of the self stabilizing nature of our protocol, is to allow the algorithm to automatically address the clock skew when it leads to a collision. In such an event, the algorithm will re-stabilize and the communication can continue from there. This second approach will be acceptable providing the clock skew results in collisions at a rate that is much slower than the rate of stabilization, which will typically be the case, since clock skew is generally very small. Also with this second approach, a ready node should be able to distinguish between two types of collisions: (i) those with non-ready nodes during the stabilization phase, and

(ii) those with ready nodes due to the clock skew after stabilization. Since we can safely assume that stabilization is much faster than the rate clock skew results in a collision, a node which becomes ready may distinguish collisions (i) and (ii) by setting a timer whose value is much larger than the expected stabilization time (of its neighborhood) and much lower than the expected time that the clocks drift large enough to cause collisions. Then, any collision before the timer expires is considered as one in the stabilization phase, hence the node remains in ready state. Collisions after the timer expires are considered as due to the clock skew, thus the node becomes non-ready.

6.3 Collision Detection

Our model assumes that, in the hidden terminal case, a node detects a collision if two or more nodes (excluding itself) within its transmission radius attempt to transmit. One way to distinguish a collision from random background noise is to place a threshold on the power of the incoming signal. Distorted signals with sufficiently high power are collisions. Since wireless signals attenuate with distance rather than drop to zero at the transmission radius [11], it is possible that many nodes outside the transmission radius transmit at the same time, resulting in a collision detected at the central node, even though none of the nodes are actually colliding. The correctness of the algorithm is not affected, however the required frame size or convergence time may get affected, because (for example) a spurious collision may prevent a node from obtaining a time slot.

We now argue that the spurious collisions as described above are statistically insignificant and should only affect the convergence time by a constant factor. Assume that the transmission power level is $\mathcal{P}$ and the received signal strength is inversely proportional to the square distance from a transmitting node. Then a node may choose $2\mathcal{P}/r^2$ as the threshold for distinguishing collisions from background noise, since this is a lower bound on the total signal strength received from a set of neighbors that are transmitting simultaneously. Now consider the set of nodes, $\mathcal{S}$, that are at distance $\geq 2r$ from a receiving node u. In order for u to detect a spurious collision, at least 8 nodes in $\mathcal{S}$ must randomly select the same time slot and transmit simultaneously, which has a probability $\leq (1/\Lambda)^7$. Hence the nodes outside of the 2-neighborhood region of a node has negligible effect on the convergence time of the algorithm. The 2-hop neighbors, on the other hand, may affect the analysis only by a constant factor.

6.4 Collision Detection in Adjacent Nodes

We have also assumed that if two neighbor nodes u and v transmit at the same time, they are able to detect the collision on the wireless channel. If the receiver and transmitter of a node operate

simultaneously, the receiver of a node hears the signals from its own transmitter very strongly so that it may not be able to detect any other incoming signal. We can remedy this problem as follows.

Our approach is based on dividing each (original) time slot in our MAC algorithms into $m = \Theta(\log n)$ *mini slots*. Each mini slot has time duration of one time step, and thus each original slot expands its time duration by a factor of m; further, the frame size of each node increases by a factor of m. Consider a node u. Suppose that during the execution of any MAC algorithm, node u selects original slot σ_u. Each original slot in u, including σ_u, is divided into m mini slots. Each mini slot has associated with it a binary value. The binary sequence of the mini slots is the same for every original slot. Let b_u denote the binary sequence of the mini slots, and $b_u(k)$ denote the kth bit in the sequence, where $1 \leq k \leq m$. Suppose that node u needs to send a message during original slot σ_u. Instead, the node will send a message in every mini slot k such that $b_u(k) = 1$.

Each node has a unique binary sequence of mini slots. This allows neighbor nodes to hear each other (i.e. detect each other's transmission, possibly as a garbled signal reception) even though their original selected slots overlap. Consider a pair of nodes u and v whose selected original slots σ_u and σ_v overlap. During σ_u, there is a mini slot k with $b_u(k) = 1$, and the corresponding overlapping mini slots at v (at most 2) have their respective bits equal to 0. Thus, a transmission by u during mini slot k will be detected at v. Symmetrically, during σ_v there is a mini slot in which transmission of v will be detected by u. (Note that the detected transmission might be observed as a garbled signal if another neighbor is also transmitting.) This scheme allows the two nodes to detect each other's transmission in both directions during their overlapping original slots.

In particular, the binary sequence of mini slots in node u is $b_u = \left([\mathrm{i}d_u][00\cdots01][\overline{\mathrm{i}d_u}][11\cdots10]\right)^{(2)}$, where $\mathrm{i}d_u$ corresponds to a unique identifier assigned to node u, and $\overline{\mathrm{i}d_u}$ to its complement. We assume that there are as many identifiers as the number of nodes n. We will not allow the use of 3 specific identifiers 01, 10, and 11. Hence, we choose each of the four subsequences in b_u to have length $\lceil \log(n+3) \rceil$ (we use 0's in the leading bits of the id's). The square power means that each bit is repeated twice, resulting to a total sequence length of $m = 8\lceil \log(n+3) \rceil$. The choice of such binary sequences for the mini slots guarantees that regardless of the alignment of the original slots in any two neighbor nodes u and v, there are always mini slots which will allow the nodes to detect each other's transmission.

This scheme applies to any set of neighbor nodes (not just a pair of nodes) whose time slots overlap. Note that having a third node (or more) in the above scheme does not prevent a node from detecting a transmission in its neighborhood (possibly received as a garbled signal), since it always has a mini slot with binary value 0 in which one or more of its neighbors are transmitting. In other words, the above scheme holds mutually between any pair of nodes in a set of nodes that are transmitting simultaneously.

7 Conclusions

We have introduced and presented the theoretical analysis of a set of distributed, contention-free MAC protocols for asynchronous wireless sensor networks. The protocols are frame-based and has the desirable properties of self-stabilization and good containment, i.e., changes to the network only affect the local area of the change, and the protocol automatically adapts to accommodate the change.

We can compare the different algorithms presented in this paper. Since $\delta_2 \leq \delta_1^2 < \delta_1^3$, Algorithm pSimpleMAC improves over LooseMAC, considering the frame size and hence the throughput. Note that $\phi(v) = \max_u \delta_2(u)$ over all 2-neighbors u of v. Hence if the node density is uniform throughout the network, then throughput per node, which is proportional to $1/\Lambda$, is asymptotically the same for pSimpleMAC and TightMAC. In such a case, the pSimpleMAC algorithm may be used as stand-alone, i.e. not followed by TightMAC. However, if the nodes are dispersed with non-uniform density, the TightMAC algorithm increases the throughput in the sparse areas of the network, when compared to pSimpleMAC.

Our MAC algorithms are designed for *arbitrary* network topologies and are worst-case optimal. We can obtain better results for the special case of *disc graphs*, where every node has a fixed radius and is connected to every other node within the radius. For unit disc graphs and constant k, the maximum k-neighborhood size δ_k grows at the same rate as the 1-neighborhood size, i.e., $\delta_1 = \Omega(\delta_k)$. Since the average 1-neighborhood size required to ensure connectivity in random graphs is $O(\log n)$ [14], it follows that in our algorithms the frames are poly-logarithmic to n, hence the throughput is inverse poly-logarithmic and time to stabilization is poly-logarithmic in the network size.

References

1. Abramson, N.: The ALOHA System - Another Alternative for Computer Communications. Proceedings of the AFIPS Conference, vol. 37, pp. 295-298, 1970.
2. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A Survey on Sensor Networks, IEEE Commun. Mag., Vol. 40, no. 8, pp. 102–114, 2002.
3. Ammar, M.H., Stevens, D. S.: A Distributed TDMA Rescheduling Procedure for Mobile Packet Radio Networks. Proceedings of IEEE International Conference on Communications (ICC), pp. 1609-1613, Denver, CO, June 1991.
4. Arisha, K. A., Youssef, M. A., and Younis, M. F.: Energy-Aware TDMA-Based MAC for Sensor Networks. IEEE Workshop on Integrated Management of Power Aware Communications, Computing and Networking (IMPACCT 2002), New York City, New York, May 2002.
5. Arumugam, M., Kulkarni, S.S.: Self-stabilizing Deterministic TDMA for Sensor Networks. Proceedings of the second International Conference on Distributed Computing and Internet Technology (ICDCIT), LNCS 3816, pp. 69-81, Bhubaneswar, India, December 22-24, 2005

6. Arvind, K.: Probabilistic Clock Synchronization in Distributed Systems. IEEE Trans. Parallel Distrib. Syst., vol. 5, no. 5, pp. 474–487, 1994.
7. Cerpa, A., Estrin, D.: ASCENT: Adaptive Self-configuring Sensor Network Topologies. Proceedings of INFOCOM'02, 2002.
8. Chen, B., Jamieson, K., Balakrishnan, H., Morris, R.: Span: An Energy-efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. Proceedings of ACM International Conference On Mobile Computing And Networking (MOBICOM'01), 2001.
9. Chlamtac, I., Franta, W. R., Levin, K.: BRAM: The Broadcast Recognizing Access Method. IEEE Transactions on Communications, vol. 27, no. 8, August 1979.
10. Cidon, I., Sidi, M.: Distributed Assignment Algorithms for Multihop Packet Radio Networks. IEEE Transactions on Computers, vol. 38, no. 10, pp. 1353-1361, October 1989.
11. Crane, R.: Propagation Handbook for Wireless Communication System Design. CRC Press LLC, 2003.
12. ElBatt, T., Ephremides, A.: Joint Scheduling and Power Control for Wireless Ad Hoc Networks. IEEE Computer and Communications Conference (INFOCOM), June 2002.
13. Farber, D., Feldman, J., Heinrich, F.R., Hopwood, M.D., Larson, K.C., Loomis, D.C., Rowe, L.A.: The Distributed Computing System. Proceedings of IEEE COMPCON, pp. 31-34, San Francisco, CA, February 1973.
14. Gupta, P. and Kumar, P. R.: Critical power for asymptotic connectivity in wireless networks. in Stochastic Analysis, Control, Optimization and Applications, A Volume in Honor of W.H. Fleming. Edited by W.M. McEneany, G. Yin, and Q. Zhang, pp 547–566. Birkhauser, 1998.
15. Herman, T., Tixeuil, S.: A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks. ALGOSENSORS pp. 45-58, 2004.
16. Magdon-Ismail, M., Sivrikaya, F., and Yener, B.: Problem of Power Optimal Connectivity and Coverage in Wireless Sensor Networks. ACM Wireless Networks (in press).
17. Kalidindi, R., Ray, L., Kannan, R., and Iyengar, S. S.: Distributed Energy-Aware MAC Protocol for Wireless Sensor Networks. International Conference on Wireless Networks, Las Vegas, Nevada, June 2003.
18. Kannan, R., Kalidindi, R., Iyengar, S.S., Kumar, V.: Energy and Rate based MAC Protocol for Wireless Sensor Networks. ACM SIGMOD Record, vol. 32 no. 4, pp. 60-65, 2003.
19. Kleinrock, L., Scholl, M.O.: Packet Switching in Radio Channels: New Conflict-free Multiple Access Schemes. IEEE Transactions on Communications, vol. 28, no. 7, pp. 1015-1029, July 1980.
20. Kleinrock, L., Tobagi, F.A.: Packet Switching in Radio Channels: Part I - Carrier Sense Multiple-Access Modes and their Throughput-Delay Characteristics. IEEE Transactions on Communications, vol. 23, no. 12, pp. 1400-1416, December 1975.
21. Martin, J.: Communication Satellite Systems. Prentice Hall, New Jersey, 1978.
22. McCormick, S.T.: Optimal Approximation of Sparse Hessians and its Equivalence to a Graph Coloring Problem, Math. Programming, vol. 26, pp. 153-171, 1983.
23. Monks, J.P., Bharghavan, V., Hwu, W.: A Power Controlled Multiple Access Protocol for Wireless Packet Networks. Proceedings of the IEEE INFOCOM 2001, Anchorage, Alaska, April, 2001.
24. Moscibroda, T., Wattenhofer, R.: Coloring Unstructured Radio Networks. 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Las Vegas, Nevada, USA, July 2005.
25. Moscibroda, T., Wattenhofer, R.: Maximal Independent Sets in Radio Networks. 24th ACM Symposium on the Principles of Distributed Computing (PODC), Las Vegas, Nevada, USA, July 2005.

26. Muqattash A., Krunz, M.: Power Controlled Dual Channel (PCDC) Medium Access Protocol for Wireless Ad Hoc Networks. Proceedings of the IEEE INFOCOM 2003 Conference, San Francisco, April 2003.
27. Muqattash, A., Krunz, M.: CDMA-based MAC protocol for Wireless Ad Hoc Networks. Proceedings of the ACM MobiHoc 2003 Conference, Annapolis, Maryland, June 2003.
28. Nelson, R., Kleinrock, L.: Spatial TDMA: A Collision Free multihop Channel Access Protocol. IEEE Transactions on Communications, vol. 33, no. 9, pp. 934-944, September 1985.
29. Rajendran, V., Obraczka, K., Garcia-Luna-Aceves, J.J.: Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks. Proceedings of ACM SenSys'03, pp. 181-192, 2003.
30. Roberts, L.G.: ALOHA Packet System with and without Slots and Capture. Computer Communications Review, vol. 5, no. 2, April 1975.
31. Singh, S., Raghavendra, C.S.: Power Efficient MAC Protocol for Multihop Radio Networks. Nineth IEEE International Personal, Indoor and Mobile Radio Communications Conference (PIMRC'98), pp:153-157, 1998.
32. Sohrabi, K., and Pottie, G., Performance Of A Novel Self-Organization Protocol For Wireless Ad-HocSensor Networks. IEEE Vehicular Technology Conference, Amsterdam, Netherlands, September 1999.
33. Tobagi, F., Kleinrock, L.: Packet Switching in Radio Channels: Part IV - Stability Considerations and Dynamic Control in Carrier Sense Multiple-Access. IEEE Transactions on Communications, vol. 25, no. 10, pp. 1103-1119, October 1977.
34. Truong, T.V.: TDMA in Mobile Radio Networks: An Assessment of Certain Approaches. Proceedings of IEEE GLOBECOM, pp. 504-507, Atlanta, GA, Nov, 1984.
35. Viterbi, A.J.: CDMA: Principles of Spread Spectrum Communication. Addison-Wesley, Reading, MA, 1995.
36. Woo, A., Culler, D.: A Transmission Control Scheme for Media Access in Sensor Networks. In proceedings of Mobicom 2001, pp 221-235.
37. Wu, S.L., Tseng, Y.C., Sheu, J.P.: Intelligent Medium Access for Mobile Ad-hoc Networks with Busy Tones and Power Control. IEEE Journal on Selected Areas in Communications (JSAC), 18(9):1647–1657, 2000.
38. Xu, Y., Heidemann, J., Estrin, D.: Geography-informed Energy Conservation for Ad Hoc Routing. Proceedings of ACM International Conference On Mobile Computing And Networking (MOBICOM'01), 2001.
39. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE standards 802.11, January 1997.
40. Ye, W., Heidemann, J., Estrin, D.: Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks. IEEE/ACM Transactions on Networking, vol. 12, no. 3, pp. 493-506, June 2004.

Costas Busch

Costas Busch received a B.Sc. (1992) and M.Sc. (1995) in Computer Science from the University of Crete, Greece. He has a PhD in Computer Science from Brown University (2000). Since then, he is an Assistant Professor at the Department of Computer Science of Rensselaer Polytechnic Institute, in Troy, New York. His research interests are in the area of distributed algorithms, communication algorithms for wireless and optical networks, constructions of distributed data structures. He has several journal and conference publications in this area of research, and served in the program committees of related conferences.

Malik Magdon-Ismail

Malik Magdon-Ismail obtained a B.S. in Physics from Yale University in 1993 and a Masters in Physics (1995) and a PhD in Electrical Engineering with a minor in Physics from the California Institute of Technology in 1998. Since then, he has been a research fellow in the Learning Systems Group at Caltech (1998-2000), and is currently an Assistant Professor of Computer Science at Rensselaer Polytechnic Institute (RPI), where he is a member of the Theory group. His research interests include the theory and applications of machine and computational learning (supervised, reinforcement and unsupervised), communication networks and computational finance. He has served on the program committees of several conferences, and is an Associate editor for Neurocomputing. He has numerous publications in refereed journals and conferences, has been a financial consultant, has collaborated with a number of companies, and has several active grants from NSF.

Fikret Sivrikaya

Fikret Sivrikaya received his BSc degree in Computer Engineering from Bogazici University, Istanbul, Turkey, and is now a PhD student at Rensselaer Polytechnic Institute in Troy, New York, USA. Previously he worked for the Turkish company Bizitek Software Development and Internet Technologies, and was a freelance consultant in several projects in Turkey. His research interests are in the areas of Computer Networks, Wireless Ad-hoc Networks, Synchronization and Scheduling, and Distributed Algorithms.

Bülent Yener

Bülent Yener has received a MS. and Ph.D. degrees in Computer Science, both from Columbia University , in 1987 and 1994, respectively. He is currently an Associate Professor in the Department of Computer Science and Co-Director of Pervasive Computing and Networking Center at Rensselaer Polytechnic Institute (RPI) in Troy, New York. He is also a member of Griffiss Institute. Before joining to RPI, he was a Member of Technical Staff at the Bell Laboratories in Murray Hill, New Jersey. His current research interests include routing problems in wireless networks, Internet measurements, quality of service in the IP networks, and the Internet security. He has served on the Technical Program Committee of leading IEEE conferences and workshops. Currently he is an associate editor of ACM/Kluwer Winet journal and the IEEE Network Magazine. He is a Senior Member of the IEEE Computer Society.