Contention-Free MAC protocols for Wireless Sensor Networks

Costas Busch, Malik Magdon-Ismail, Fikret Sivrikaya, and Bülent Yener

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180, USA
{buschc, magdon, sivrif, yener}@cs.rpi.edu

Abstract. A MAC protocol specifies how nodes in a sensor network access a shared communication channel. Desired properties of such MAC protocol are: it should be *distributed* and *contention-free* (avoid collisions); it should *self-stabilize* to changes in the network (such as arrival of new nodes), and these changes should be *contained*, i.e., affect only the nodes in the vicinity of the change; it should not assume that nodes have a global time reference, i.e., nodes may not be time-synchronized. We give the first MAC protocols that satisfy all of these requirements, i.e., we give distributed, contention-free, self-stabilizing MAC protocols which do not assume a global time reference. Our protocols self-stabilize from an arbitrary initial state, and if the network changes the changes are contained and the protocol adjusts to the local topology of the network. The communication complexity, number and size of messages, for the protocol to stabilize is small (logarithmic in network size).

1 Introduction

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. A sensor network is comprised of a large number of limited power sensor nodes which collect and process data from a target domain and transmit information back to specific sites (e.g., headquarters, disaster control centers). We consider wireless sensor networks which share the same wireless communication channel. A Medium Access Control (MAC) protocol specifies how nodes share the channel, and hence plays a central role in the performance of a sensor network.

Sensor networks contain many nodes, typically dispersed at high, possibly non-uniform, densities; sensors may turn on and off in order to conserve energy; and, the communication traffic is space and time correlated. *Contention* occurs when two nearby sensor nodes both attempt to access the communication channel at the same time. Contention causes message collisions, which are very likely to occur when traffic is frequent and correlated, and they decrease the lifetime of a sensor network. A MAC protocol is contention-free if it does not allow any collisions. All existing contention-free MAC protocols assume that the sensor nodes are time-synchronized in some way. This is usually not possible on account of the large scale of sensor networks.

The preceding discussion emphasizes the following desirable properties for a MAC protocol in sensor networks: it should be *distributed* and *contention-free*; it should *self-stabilize* to changes in the network (such as the arrival of new nodes into the network), and these changes should be *contained*, i.e., affect only the nodes in the vicinity of the change; it should not assume that nodes have access to a global time reference, i.e., nodes may not be time-synchronized. These properties are essential to the scalability of sensor networks and for keeping the sensor hardware simple. In this paper, we give the first MAC protocols that satisfy all of these requirements.

A contention-free MAC protocol should be able to bring the network from an arbitrary state to a collision-free stable state. Since the protocol is distributed, during this stabilization phase collisions are unavoidable. 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*, the inverse of the time interval between its transmissions.

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, i.e., if they are within each other's transmission range. We assume that all sensor nodes have the same transmission range, hence all links are bidirectional and the graph G is undirected.

A message sent by a node is received by all of its adjacent nodes. If two nodes are adjacent and send messages simultaneously, their messages collide. 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 (*hidden terminal problem*). We assume that nodes can detect such collisions.

The *k-neighborhood* of a node v, $\Delta_k(v)$, 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 $\delta_k(v)$ (where $\delta_k(v) = |\Delta_k(v)|$), and the maximum k-neighborhood size by δ_k ($\delta_k = \max_v \delta_k(v)$). We refer to 1-neighbors as neighbors. We can also define the k-neighborhood of a set of nodes S: $\Delta_k(S)$ is the set of nodes that are at most a distance k away from *some* node in S. We assume that at the start of the algorithm, every node has been provided an upper bound on δ_1 and δ_2 (for example this information can be provided by the network administrator).

Contributions. We give a distributed, contention-free, self-stabilizing MAC protocol which does not assume a global time reference. The protocol has two parts. Starting from an arbitrary initial state, the protocol first enters a *loose* phase where nodes set up a preliminary MAC protocol. This phase is followed by a *tight* phase in which nodes make the MAC protocol more efficient. Both parts of the protocol are self-stabilizing. Since we make no assumptions about the initial state, the protocol will also re-stabilize after any network change.

During the *loose* phase, every node transmits at most $O(\log n)$ control messages, each of size at most $O(\log n)$ bits. The time duration of this phase is $O(\log n \cdot \min\{\delta_1^3, \delta_2^2\})$. The protocol has now reached a stable state in which the throughput of a node is $O(1/\min\{\delta_1^3, \delta_2^2\})$. The network may either remain in this protocol, or proceed to the next (tight) phase in which we improve the steady state throughput of the nodes. The tightening phase also requires at most $O(\log n)$ control messages per node of size at most $O(\log n)$ bits to reach the steady state. The time duration of this phase is $O(\log n \cdot \min\{\delta_1^5, \delta_1^2\delta_2^2\})$. During steady state, the throughput of node v is closely related to $1/\phi_v$, where ϕ_v is the maximum 2-neighborhood size among all nodes in v's 2-neighborhood. An important property of the tight phase is that the throughput of a node is related only to the local "density" of the graph in the vicinity of the node, and hence adapts to the varying topology of the network.

If the network changes, for example a set S of nodes suddenly power up after being powered down for some time, as already mentioned, the protocol will self-stabilize to the change. Further, the only nodes that are affected by the stabilization are nodes in $\Delta_2(S)$ for the loose phase, and $\Delta_6(S)$ for the tight phase.

Approach. Our approach is based on the concept of a *frame* (see Figure 1), which is the basis of TDMA MAC protocols. We adapt the frame approach so that it does not depend on any global time reference. Each node divides time into equal sized frames. Each frame is further divided into equal sized time *slots*; a time slot corresponds to the time duration of sending one message. 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.

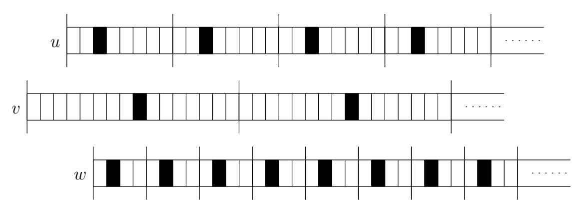

Fig. 1. Frames of three nodes. Frames at different nodes may not be aligned. Solid shaded time slots indicate the selected time slot of each node; longer vertical lines identify the frame boundaries.

The basic idea 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. Thus, nodes need to select slots only once, and the slots remain conflict-free thereafter.

The MAC protocols (algorithms) we provide find conflict-free time slots. For the loose phase, we have developed algorithm LooseMAC in which all nodes have the same fixed frame size, which is proportional to $\min\{\delta_1^3, \delta_2^2\}$. For the tight phase we have developed algorithm TightMAC in which each node v has frame size proportional to ϕ_v, which depends only on the local area density of node v. Thus, in TightMAC different nodes in the network have different frame sizes that reflects the variation of the node density in different areas of the network.

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 [21]. The throughput of Aloha-like protocols was significantly improved by the Carrier Sense Multiple Access (CSMA) protocol [14]. 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 [29] 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 [23]. As a result, contention-based protocols are not suitable for sensor networks.

Our work is most 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) [15]; Code Division Multiple Access (CDMA) [25]. In addition to TDMA, FDMA and CDMA, various reservation based [13] or token based schemes [7, 10] 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 [19, 24], and distributed [2, 8, 20] schemes for TDMA slot assignments. These existing protocols are either centralized or rely on a global time reference.

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 [17, 16, 27], transmission scheduling [9], and limited interference CDMA systems [18]. Some recent work on energy conservation by powering off some nodes is studied in [22, 28, 6, 5]. While GAF [28] and SPAN [6] are distributed approaches with coordination among neighbors, in ASCENT a node decides itself to be on or off [5]. S-MAC [30] proposes that nodes form virtual clusters based on common sleep schedules. Sleep and wake schedules are used in [12], but based on energy and traffic rate at the nodes in order to balance energy consumption. A different approach is used in [26], with an adaptive rate control mechanism to provide a fair and energy-efficient MAC protocol.

Paper Outline. In Section 2 we give Algorithm LooseMAC and an outline of its analysis (the detailed analysis can be found in [4]). We proceed with Algorithm TightMAC in Section 3, followed by some concluding discussion in Section 4.

2 Algorithm LooseMAC

Here we present Algorithm LooseMAC and an outline of its analysis. Each node selects the same frame size, which is proportional to $\min\{\delta_1^3, \delta_2^2\}$. The algorithm is randomized and guarantees that all nodes will find their slots quickly, with low communication complexity. Further, the algorithm is self-stabilizing with good containment properties.

For simplicity of the presentation, we will assume that slots are aligned (frames do not need to be aligned). All results hold immediately for when the slots are not aligned (with small constant factors, since each slot may overlap with at most two slots in a neighbor's frame). For notation convenience, given a set of nodes $V = \{v_1, v_2, \ldots, v_n\}$, we will denote node v_i simply as node i.

2.1 Description of LooseMAC

Algorithm 1 depicts the basic functionality of LooseMAC. Consider some node i. Node i divides time into frames of size Λ. The task for node i is to select a conflict-free slot. When this occurs we say that the node is "ready", and we set its local variable variable ready to TRUE.

Algorithm 1 LooseMAC(node i)

```
Divide time into frames of size Λ;
ready ← FALSE;
while not ready do
    Select a slot σ_i randomly in the frame;
    Send a "beacon" message in slot σ_i;
    Listen for a period of Λ time slots;
    if no collision is detected by i and no neighbor of i reports a conflict then
        ready ← TRUE;
```

Initially, when node i enters the network it 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 during the next Λ time slots. If σ_i doesn't create any slot conflicts in its neighbors during Z, 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 a new neighbor joins the network, which is described in Section 2.2). Below we explain how node i can detect that σ_i creates a conflict, and therefore, whether to keep or abandon the selected time slot (see also Figure 2).

If m_i creates slot conflicts in some neighbor j, then j responds by transmitting a message m_j reporting the conflict (this message simply says that j detected a conflict, without specifying which nodes are conflicting). 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 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 was with σ_i, and i abandons slot σ_i continuing by selecting another slot in its next frame. If m_j collides at node i, i does not know if m_j was reporting a conflict or not as i cannot decode two or more colliding messages. Again i assumes that the collided message was reporting a conflict, abandons slot σ_i and selects another slot in its next frame. The process repeats until i does not detect any message collisions or does not receive any conflict reports during Z. Note that a node will transmit at most twice during any arbitrary period of Λ slots, since the frame size is Λ and a node transmits at most once during every particular frame.

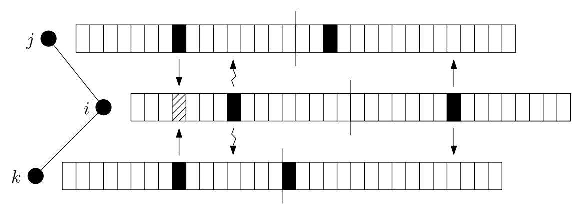

Fig. 2. Execution of the LooseMAC algorithm, where the shaded slot corresponds to a collision and the waived lines to a conflict report message.

To enable a node to detect slot conflicts, the node marks the time slots that are being used by its neighbors. Consider node j. Suppose that a neighbor i sends a message m_i to j at a slot π. 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. Suppose that a neighbor node k sends a message during slot π, which is already marked with i. Node j then detects a conflict between the time slots chosen by i and k. A conflict occurs also if nodes i and k transmit at the same time, which is observed as a message collision by j. Actually, in the LooseMAC algorithm (Algorithm 1, when a node picks a new slot in its frame it picks it among the unreserved slots. We obtain the following result.

Lemma 1. *For some constant c, if $\Lambda \geq c \min\{\delta_1^3, \delta_2^2\}$, all non-ready nodes become ready within $\Lambda \cdot \log n$ time slots, with probability at least $1 - \frac{1}{n}$.*

Sketch of proof: We sketch the case $\Lambda \geq c\delta_1^3$ (the case $\Lambda \geq c\delta_2^2$ is treated similarly). Let i be a non-ready node and let R_i denote the nodes that are ready in $\Delta_1(i)$. Suppose i selects a new slot σ_i, and let Z denote the following Λ slots, and Z' the preceding Λ slots. Let p_1 denote the probability that some neighbor node conflicts with σ_i (i.e. attempts to reserve the same slot), let p_2 be the probability that i hears a collision during Z, and let p_3 be the probability that i receives a conflict report during Z. Then the probability that i fails to become ready is at most $p_1 + p_2 + p_3$.

Node i selects a slot from among $\Lambda - R_i$ slots. σ_i can only conflict with a non-ready neighbor of i. Let j be one such neighbor. j chooses from among $\Lambda - R_j$ slots, therefore j selects the same slot σ_i with probability at most $1/(\Lambda - R_j) \leq 2/\Lambda$ for $\Lambda \geq 2\delta_1$ (since $R_j \leq \delta_1$). The union bound then gives $p_1 \leq 2(\delta_1 - R_i)/\Lambda \leq 2\delta_1/\Lambda$. Node i will hear a collision if two neighbors j, j' both transmit on the same slot σ_k. This occurs with probability at most $4/(\Lambda - R_j)(\Lambda - R_{j'}) \leq 16/\Lambda^2$, since a node may transmit at most twice during Z. Since there are at most Λ such slots hear a collision and at most $\delta_1^2/2$ different pairs of neighbors, the union bound gives $p_2 \leq 8\delta_1^2/\Lambda$.

A neighbor j will report a conflict in Z only if two of its neighbors collide in $Z \cup Z'$. A similar argument to the bound for p_2 bounds this probability by $16\delta_1^2/\Lambda$. Since there are at most δ_1 such neighbors, the union bound gives $p_3 \leq 16\delta_1^3/\Lambda$.

Thus, for some c, $p_1 + p_2 + p_3 \leq c\delta_1^3/\Lambda$. If $\Lambda \geq 4c\delta_1^3$, then the probability that i becomes ready is at least $\frac{1}{4}$. Consequently, after $\log n$ independent tries, 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, applying the union bound gives the result. ∎

2.2 Fresh Nodes

Algorithm LooseMAC adapts dynamically to nodes joining or leaving the network. When a node leaves, it informs its neighbors who can then unmark the slot they reserved for the departing node. If a node fails or crashes, it cannot inform its neighbors, and its slot will remain marked; the correctness of the algorithm and the rest of the network remain unaffected.

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 will 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.

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 "I'm fresh" message, and so it switches to the non-fresh, non-ready status. At this point, every neighbor of i has become non-ready. i now continues with the original LooseMAC algorighm as a non-ready node. The analysis of the time for fresh nodes to become non-ready is similar to the proof of Lemma 1:

Lemma 2. *For some constant c, if $\Lambda \geq c \min\{\delta_1^3, \delta_2^2\}$, all fresh nodes become non-fresh within $\Lambda \cdot \log n$ slots with probability at least $1 - \frac{1}{n}$.*

2.3 Complexity of LooseMAC

A network is in a *stable* state when all nodes are ready. Once a network is stable, it remains so until a node joins the network. We show that from an arbitrary initial state I, the network will stabilize if no changes are made to the network. Suppose that $\Lambda \geq c \min\{\delta_1^3, \delta_2^2\}$, 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 2 implies that, with probability at most $\frac{1}{n}$, after $\Lambda \cdot \log n$ slots, *some* node fails to become non-fresh. Similarily, after an additional $\Lambda \cdot \log n$ slots, Lemma 1 implies that, with probability at most $\frac{1}{n}$, after $\Lambda \cdot \log n$ slots, *some* node fails to become ready. Applying the union bound, we have that with probability at least $1 - \frac{2}{n}$, every node is ready after $2\Lambda \cdot \log n$ slots, i.e., w.h.p., the network has reached a stable state.

Containment. Nodes that send control messages until stabilization are *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 in $\Delta_2(S)$.

Communication Complexity. Each affected node sends at most $O(\log n)$ control messages, since in every frame it sends at most 1 message. Each message has size $O(\log n)$ bits, since the message consists of the sender's id ($\log n$ bits), fresh status (1 bit), and conflict report (1 bit). We thus have the following theorem.

Theorem 1 (Complexity of LooseMAC). *From an arbitrary initial state I with non-ready nodes S, the network stabilizes within $2\Lambda \cdot \log n$ slots with probability at least $1 - \frac{1}{\Theta(n)}$. 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 ϕ_i; recall that $\phi_i = \max_{j \in \Delta_2(i)} \delta_2(j)$ is 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".)

A node 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. It can thus compute the size of the tight frame, and find a conflict-free tight slot (in the tight frame). Then the node starts using the tight frame's slots. 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.

Ready Levels. After a node runs LooseMAC, the TightMAC algorithm requires that all nodes in its 2-neighborhood are ready (in order to compute ϕ_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., $\Lambda = 2^{\lceil \log c \min\{\delta_1^3, \delta_2^2\} \rceil}$. A node becomes ready-0 as explained earlier in LooseMAC. A node becomes ready-K (for $K > 0$) if all nodes in its neighborhood are ready at level at least $K - 1$. We assume that when nodes send messages, they also include their ready status (fresh, not-ready, ready-0, ready-1, ready-2, ready-3 or ready-4). Thus, when a node becomes ready-K, it sends a message (in its loose slot) informing its neighbors. When a ready-K node has received ready-K messages from all of its neighbors it becomes ready-(K+1) (if $K < 4$).

If a node is ready-K and hears that one of its neighbors is fresh, then it becomes not-ready. 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.

3.1 Description of TightMAC

Algorithm 2 gives an outline of TightMAC. A node first executes LooseMAC until it becomes ready-4. In the main loop of TightMAC, all the control messages are sent using loose slots, until the node switches to using tight frames.

Algorithm 2 TightMAC(node i)

```
1: repeat
2:    Execute LooseMAC(i)
3: until i becomes ready-4
4: Transmit neighborhood information and compute φ_i;
5: Choose a frame F_i with |F_i| = 2^⌈log 6φ_i⌉;
6: Inform neighbors for the relative position of F_i, with respect to i's loose slot;
7: Execute FindTightSlot();
8: Start using the tight frame;
```

The highest throughput is obtained by choosing ϕ to be the maximum 2-neighborhood size among a node's 2-neighbors, $\phi_i = \max_{j \in \Delta_2(i)} \delta_2(j)$. This means that nodes need to obtain their 2-neighborhood size, and each node needs to communicate to its 1-neighbors the actual IDs of the nodes in its 1-neighborhood, which is a message of size $O(\delta_1 \log n)$ bits. An alternative is to use an upper bound for ϕ_i, which is to take the maximum of an upper bound on the 2-neighborhood size over i's 2-neighborhood: $\overline{\phi}_i = \max_{j \in \Delta_2(i)} \overline{\delta}_2(j)$, where $\overline{\delta}_2(j) = \sum_{k \in \Delta_1(j)} \delta_1(k)$ is an upper bound on $\delta_2(j)$. Since the steady-state throughput is $1/\phi$, this results in lower throughput. However, the advantage is that only the 1-neighborhood size needs to be sent, which has a message length of $O(\log n)$ bits. In either event, only 1-neighborhood information needs to be exchanged (whether it is 1- neighborhood node IDs or 1-neighborhood size). When a node becomes ready-1, it knows its 1-neighborhood (by examining the number of marked slots in its loose frame), so it can send the relevant information to its neighbors. When a node becomes ready-2, it will have received the 1-neighborhood information from all its neighbors. It processes this information to compute either δ_2 or $\overline{\delta_2}$, which it transmits. When a node becomes ready-3, it will have received the δ_2 or $\overline{\delta_2}$ information from all its neighbors and hence can take the maximum which it transmits. When a node becomes ready-4, it will have received maximums from all its neighbors, so it can take the maximum of these maximums to compute ϕ or $\overline{\phi}$. This entire process requires at most 3 control messages from each node. For simplicity we will present most of the results using ϕ_i and $\delta_2(i)$. The same results hold for $\overline{\phi}_i$ and $\overline{\delta}_2(i)$ with similar proofs.

Finally, node i chooses its tight frame size F_i to be the smallest power of 2 that is at least $6\phi_i$; i notifies its neighbors about the position of F_i relative to its loose slot (this information will be needed for the neighbor nodes to determine whether the tight slots conflict); and, node i executes FindTightSlot (described in Section 3.2) to compute its conflict-free tight slot in F_i. After the tight slot is computed, node i switches to using its tight frame F_i.

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.

3.2 Algorithm FindTightSlot

The heart of TightMAC is algorithm FindTightSlot, which obtains conflict-free slots in the tight frames. The tight frames have different sizes at various nodes which depends locally on ϕ_i. 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. Every time that the frames repeat, s_i overlaps with the same slots in F_j. The *coincidence set* $C_{i,j}(s_i)$ 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. If on the other hand $|F_i| < |F_j|$, then $|C_{i,j}(s_i)| > 1$. Nodes i and j *conflict* if their selected slots s_i and s_j are chosen so that $s_j \in C_{i,j}(s_i)$ (or equivalently $s_i \in C_{j,i}(s_j)$); in other words, s_i and s_j overlap at some repetition of the frames F_i and F_j.

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

Algorithm 3 FindTightSlot()

SlotFound ← FALSE;
while not SlotFound **do**
  Select ← FALSE;
  With probability $1/\phi_i$: Select ← TRUE;
  if Select **then**
    Let s_i be an randomly chosen unreserved slot in the first $6\delta_2(i)$ slots of F_i;
    Send the position of s_i (relative to its loose slot);
    Listen for a period of Λ time slots;
    if no conflict is reported by any neighbor **then**
      SlotFound ← TRUE;

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 possible. Node i then notifies its neighbors of its choice using its loose slot. Each neighbor j then checks if σ_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 (again in its loose slot). Node i listens for Λ time slots. During this period, if a neighbor detected a conflict, it reports it. Otherwise the neighbor marks this slot as belonging to i (deleting any previously marked tight slot for i). If i receives a conflict report, the process repeats, with i selecting another tight slot. If i does not receive a conflict report, then fixes this tight slot. Note that this communication between i and its neighbors can occur because all its neighbors are at least ready-2.

In the algorithm, a node chooses to select a new slot with probability $1/\phi_i$, so not many nodes attempt to select a 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 i. Node j chooses a slot in the first $Z = 6\delta_2(j)$ of its tight frame F_j. Node i has frame size larger than Z. Thus, node i cannot have more than one slot repetition in Z. This implies that i and j conflict at most once during Z in their tight frames, and so, the possible 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 3. *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 $1/2$.*

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 slot s_j from among $6\delta_2(j) - 2\delta_1(j) \geq 4\delta_2(j)$ unreserved slots in its 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 only nodes that can reserve slots in members of S are therefore 2-neighbors of j. Each such node can mark at most two slots (one loose and one tight slot), and so at most $2\delta_2(j)$ (absolute) 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, of the $4\delta_2(j)$ chosen from, hence the probability of causing no conflict is at least $\frac{1}{2}$.

Lemma 4. *For node i, during a period of Λ time slots, no conflicts occur in $\Delta_1(i)$ with probability at least $1/2$.*

Proof. Let Z be the period of Λ time slots. A conflict is caused during Z in $\Delta_1(i)$ by any slot selection of nodes in $\Delta_2(i)$. A slot selection by node $j \in \Delta_2(i)$ occurs with probability $1/\phi_j$. From Lemma 3, 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 4 implies that every time i selects a slot in its tight frame, this slot is conflict-free with probability at least 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 $c \cdot \phi_i \cdot \Lambda \cdot \log n$ time slots, a ready-4 node successfully chooses a conflict-free tight time slot in F_i, with probability at least $1 - \frac{1}{n^2}$.*

3.3 Complexity of TightMAC

The network is *stable* if all nodes in the network have selected conflict-free tight slots. When a network stabilizes, it remains so until some node joins/leaves the network. We now show that starting from an arbitrary initial state I, if no changes occur in the network after I, the network reaches a stable state.

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. Then, $O(1)$ time is required for the nodes to become ready-4. 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 $O(1)$ messages of size $O(\log n)$ bits so that itself and its neighbors can compute $\overline{\phi}$. From Corollary 1, with high probability, node i then requires $O(\overline{\phi}_i \Lambda \log n)$ time slots to select a conflict-free tight slot. Since, $\delta_1^2 \geq \overline{\phi}_i$, the total time for stabilization is $O(\delta_1^2 \Lambda \log n)$. When a fresh node i arrives the nodes in $\Delta_5(i)$ drop in ready level, hence the affected area is at most $\Delta_6(i)$. Following an analysis similar to Corollary 1, we obtain the following theorem.

Theorem 2 (Complexity of TightMAC). *From an arbitrary initial state I, with non-ready nodes S, the network stabilizes within $O(\delta_1^2 \cdot \Lambda \cdot \log n)$ time slots, with probability at least $1 - \frac{1}{\Theta(n)}$. The affected area is $\Delta_6(S)$. Each affected node sends $O(\log n)$ messages, of size $O(\log n)$ bits.*

4 Discussion

We have introduced and presented the theoretical analysis of a distributed, contention-free MAC protocol. This protocol is frame based and has the desirable properties of self-stabilization and containment, i.e., changes to the network only affect the local area of the change, and the protocol automatically adapts to accommodate the change. Further, the efficiency of the protocol depends only on the local topology of the network, and so bottlenecks do not affect the entire network. In general, such frame based protocols tend to be preferable when traffic is constant (rather than bursty).

The time to stabilization and throughput are related to the maximum 1 or 2-neighborhood size. For unit disc graphs, the maximum k-neighborhood size δ_k grows at the same rate as the 1-neighborhood size, i.e., $\delta_1 = \Omega(\delta_k)$ for fixed k. Since the average 1-neighborhood size required to ensure connectivity in random graphs is $O(\log n)$, [11], it follows that throughput is inverse polylogarithmic and time to stabilization is polylogarithmic in the network size.

Practical Considerations. Our model assumes that a node detects a collision if and only if two or more nodes (including 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. Signals with sufficiently high power are collisions. Since wireless signals attenuate with distance rather than drop to zero at the transmission radius, it is then possible many nodes that are outside the transmission radius will 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 taking a time slot. A natural next step is to investigate such issues using simulation techniques for real sensor networks.

We have assumed that the time slots of the nodes are aligned, even though the frames may not be aligned. This simplification is not necessary, because we can accommodate misaligned time slots by having a node become ready in its time slot only if there are no collisions detected in its time slot as well as neighboring time slots. This means that a node's time slot may block off at most 4 time slots at every one of its two neighbors, which means that the frame size will have to be at most a constant factor larger.

All our discussion applies when there is no relative drift between the local clocks of the nodes. In practice there can be a very small drift (clock skew). One approach to addressing this problem is to run a clock skew algorithm (for example [3]) 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.

Future Research. Two natural directions are to improve the convergence properties or to obtain lower bounds on the stabilization time and/or the required frame size for any such distributed contention free algorithm which does not use global time synchronization. Many aspects of our protocol are hard to analyze theoretically and would benefit from experimental analysis, for example, how the performance varies with the rate at which the network topology is changing, or what the lifetime of the network will be. Further, we have made certain simplifying assumptions such as the graph is undirected, when in fact link quality may vary with time. Future directions would be to investigate how the choice of the power level would interact with such parameters as the performance of the protocol, the lifetime of the network and the connectivity of the network. It would also be useful to compare the performance of our algorithms with existing algorithms (such as IEEE 802.11 and TDMA) on real networks using simulation packages (such as OPNET and NS2).

References

1. Abramson, N.: The ALOHA System - Another Alternative for Computer Communications. Proceedings of the AFIPS Conference, vol. 37, pp. 295-298, 1970.
2. 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.
3. Arvind, K.: Probabilistic Clock Synchronization in Distributed Systems. IEEE Trans. Parallel Distrib. Syst., vol. 5, no. 5, pp. 474–487, 1994.
4. Busch, C., Magdon-Ismail, M., Sivrikaya, F., Yener, B.: Contention-Free MAC protocols for Wireless Sensor Networks. Technical Report, Rensselaer Polytechnic Institute, 2004. Available at http://www.cs.rpi.edu/research/tr.html.
5. Cerpa, A., Estrin, D.: ASCENT: Adaptive Self-configuring Sensor Network Topologies. Proceedings of INFOCOM'02, 2002.
6. 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.
7. Chlamtac, I., Franta, W. R., Levin, K.: BRAM: The Broadcast Recognizing Access Method. IEEE Transactions on Communications, vol. 27, no. 8, August 1979.
8. 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.
9. ElBatt, T., Ephremides, A.: Joint Scheduling and Power Control for Wireless Ad Hoc Networks. IEEE Computer and Communications Conference (INFOCOM), June 2002.
10. 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.
11. P. Gupta and P. R. Kumar: 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.

12. 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.
13. 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.
14. 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.
15. Martin, J.: Communication Satellite Systems. Prentice Hall, New Jersey, 1978.
16. 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.
17. 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.
18. Muqattash, A., Krunz, M.: CDMA-based MAC protocol for Wireless Ad Hoc Networks. Proceedings of the ACM MobiHoc 2003 Conference, Annapolis, Maryland, June 2003.
19. 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.
20. 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.
21. Roberts, L.G.: ALOHA Packet System with and without Slots and Capture. Computer Communications Review, vol. 5, no. 2, April 1975.
22. 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.
23. 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.
24. Truong, T.V.: TDMA in Mobile Radio Networks: An Assessment of Certain Approaches. Proceedings of IEEE GLOBECOM, pp. 504-507, Atlanta, GA, Nov, 1984.
25. Viterbi, A.J.: CDMA: Principles of Spread Spectrum Communication. Addison-Wesley, Reading, MA, 1995.
26. Woo, A., Culler, D.: A Transmission Control Scheme for Media Access in Sensor Networks. In proceedings of Mobicom 2001, pp 221-235.
27. 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.
28. 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.
29. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE standards 802.11, January 1997.
30. 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.