Contention-Free MAC protocols for Wireless Sensor Networks

Costas Busch buschc@cs.rpi.edu
Malik Magdon-Ismail magdon@cs.rpi.edu
Fikret Sivrikaya sivrif@rpi.edu
Bülent Yener yener@cs.rpi.edu

Abstract

A MAC protocol specifies how nodes in a sensor network coordinate their communication over a shared communication channel. Owing to the limited capabilities of sensor nodes, the desired properties of a MAC protocol are: it should be *distributed* and 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. In particular, we provide distributed, contention-free, self-stabilizing MAC protocols which do not assume a global time reference.

In a stable state, our protocols ensure that a node's throughput is inversely proportional to the *local* density of nodes, hence a localized bottleneck will not affect the entire network. The communication complexity until stabilization is small: $O(\log n)$ control messages per node, of size at most $O(\log n)$ bits (n is the size of the sensor network). The time it takes for the protocol to stabilize depends on the maximum density of the nodes. Further, in the event that the network changes, only nodes in the neighborhood of the change get affected.

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 access the communication channel. 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 take to reach the stable state, 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 which it is allowed to transmit messages by the MAC protocol.

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. We do not place any restrictions on G. 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*).

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 size of a nodes in the k-neighborhood by $\delta_k(v)$, and the maximum k-neighborhood size by δ_k. A 1-neighbor is simply referred to as neighbor. We can generalize this notion of a k-deighborhood to 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 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 re-stabilize after network changes as well.

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 $O(\log n)$ 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 $1/\phi_v^2$, where ϕ_v is the maximum 1-neighborhood size of all node v's 2-neighbors. An important property of the tight phase is that the throughput of a node is related only to the local "density" of the node in graph, 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 $\Delta_3(S)$ 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.

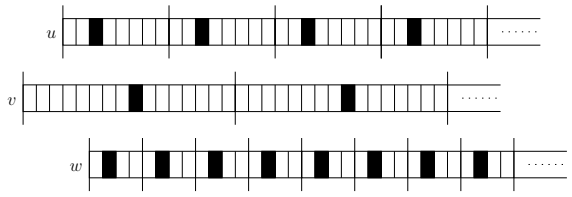

Figure 1: Frames of three nodes. Frames at different nodes may not be aligned. Solid shaded time slots indicate the selected time slot of each node; longer vertical lines identify the frame boundaries.

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 (that is, 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^2, which depends only on the local area density of node i. 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 fall into two broad classes: contention-based and 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 [17]. The throughput of Aloha-like protocols was significantly improved by the Carrier Sense Multiple Access (CSMA) protocol [10]. 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 [24] standard for wireless ad-hoc networks. The performance of contention based MAC protocols is weak when traffic is frewuent or correlated and these protocols suffer from stability problems [19]. As a result, contention-based protocols are not suitable for sensor networks.

Most related to our work are *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) [11]; Code Division Multiple Access (CDMA) [21]. In addition to TDMA, FDMA and CDMA, various reservation based [9] or token based schemes [5, 8] 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 [15, 20], and distributed [2, 6, 16] schemes for TDMA slot assignments. All these existing protocols are either centralized or rely on a global time reference.

There is considerable recent work on integrated views of several layers in wireless networking. Power controled MAC protocols have been considered by [7, 13, 14, 12, 22] in settings that are based on collision avoidance [13, 12, 22], transmission scheduling [7], and limited interference CDMA systems [14]. Some recent results suggest energy saving by powering off a subset of the nodes in an ad hoc wireless network [18, 23, 4, 3]. The common theme is to enable nodes to power off or go to low energy sleeping mode during idle time while ensuring connectivity. While GAF [23] and SPAN [4] are distributed approaches with coordination among neighbors, in ASCENT a node decides itself to be on or off [3].

Paper Outline. We continue in Section 2 with a description and analysis outline of Algorithm LooseMAC (the detailed description and analysis appears in the appendix). Then we give the description and analysis of Algorithm TightMAC in Section 3.

2 Algorithm LooseMAC

Here we give a description of Algorithm LooseMAC and its performance analysis. In the algorithm 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 fast, sending only a small number of messages. Further, the algorithm is self-stabilizing with good affected area 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 with 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"; variable *Ready* set to true signifies this event.

Algorithm 1 LooseMAC(node i)

```
1: Divide time into frames of size Λ;
2: Ready ← false;
3: while not Ready do
4:    Select a slot σ_i randomly in the frame;
5:    Send a "beacon" message in σ_i;
6:    Listen for a period of Λ time slots;
7:    if no collision is detected by i and no neighbor of i reports a conflict then
8:       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 with the next Λ time slots. If σ_i doesn't create any conflicts in its neighbors during Z, then node i keeps slot i and it 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 the 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 2.1).

If m_i creates conflicts in some neighbor j, then j responds with broadcasting a message m_j that reports the conflict (this message simply says that a conflicts occurred in j, without specifying which node caused the conflict). Node j sends the m_j during its currently selected slot σ_j. Since 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, then i infers that σ_i creates conflicts, and i abandons σ_i and continues with selecting another slot. If m_j is received with collisions, node j does not know if m_j was reporting a conflict or not (since it listens to noise). For safety, i assumes that the collided message was reporting a conflict, so node i again abandons σ_i and selects another slot. The process repeats until i does not detect message collisions in Z and no node reports a conflict in a period of Λ time slots.

To enable the nodes to detect conflicts, they use a marking mechanism. A 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 detects a conflict 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. We obtain the following result.

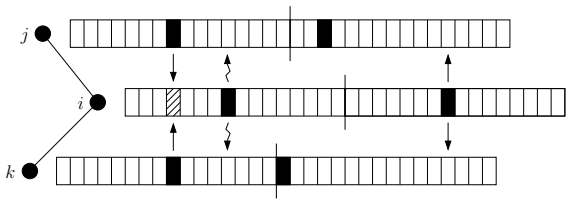

Figure 2: Execution of the LooseMAC algorithm, where the shaded slot corresponds to a collision and the waived lines to a conflict report message.

Lemma 1 *For $\Lambda = \Theta(\min\{\delta_1^3, \delta_2^2\})$, with high probability, a non-ready node becomes ready within $O(\Lambda \cdot \log n)$ time slots.*

Sketch of proof: Here, we consider only the case $\Lambda = \Theta(\delta_1^3)$ (the case $\Lambda = \Theta(\delta_2^2)$ is treated similarly). Let i be a non-ready node. When node i selects a new slot, it becomes ready if for the next period of Λ slots, which we will denote Z, no collision occurs in i and no neighbor of i reports a conflict.

During Z, a neighbor j sends at most two messages, one due to a new selected slot, and one at the old selected slot. According to the algorithm, collisions (and conflicts) are caused primarily when nodes select new slots. A new slot message of j collides with probability at most $p_1 \leq c\delta_1/\Lambda$, for some small constant c. This holds since i has at most δ_1 neighbors and each neighbor can send at most 2 messages in period Z, that could collide with the message of j. Considering now all the neighbors of i, a collision occurs in i during Z with probability at most $p_2 = p_1\delta_1 \leq c\delta_1^2/\Lambda$.

A neighbor node j reports a conflict in Z if a conflict occurs in j either during Z or in the previous Λ time slots. With a similar analysis as above, this event occurs with probability at most $p_3 \leq c'\delta_1/\Lambda$ (where $c' > c$, since we also consider the conflicts in the marked slots in the frame of j). Considering now all the $\delta_1 - 1$ neighbors, we have that a conflict is reported by any of them with a probability most $p_4 = \delta_1 p_3$.

Consequently, node j remains non-ready with probability at most $p_5 = p_2 + p_4 \leq c''\delta_1^3$, for constant $c'' \geq c'$. Since $\Lambda = \Theta(\delta_1^3)$, $p_5 \leq c''$, which is a constant. Thus, in the expected case, node i becomes ready within $1/c''$ frame repetitions. This implies that with high probability, node i switches to ready in $O(\log n)$ frame repetitions, as needed. ■

2.2 Fresh Nodes

Algorithm LooseMAC can handle the case in which nodes join and leave dynamically the network. When a node leaves, it simply informs the neighbor nodes to unmark the respective slots, and the rest of the network remains unaffected. However, even if the departing node isn't able to inform its neighbors that it is leaving, for example when a node fails or crashes, the rest of the network still remains unaffected.

The situation is more complicated when a node joins the network, due to the hidden terminal problem. Let j and k be two nodes which are not 2-neighbors. Let i be a node which joins the network and which is neighbor with both j and k. Thus, when i joins, j and k become 2-neighbors. Suppose now that j and k were ready before i joins, and have already selected slots which are overlapping (and which were conflict-free, since i and j were not 2-neighbors). When i joins, the slots of j and k conflict, and thus they need to reselect slots.

This is accomplished by having node i 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 of i receives a control message from i indicating that i is fresh, then j becomes non-ready.

While i is fresh, it selects random slots and if for a period of Λ slots no conflict occurs in it and its neighbors, then its switches to the non-fresh status. This guarantees that every neighbor of i has become non-ready before i becomes non-fresh. Then, node i continues with the algorithm as being non-ready until it becomes ready, as described in Section 2.1. With an analysis similar to Lemma 1, we obtain the following lemma.

Lemma 2 *For $\Lambda = \Theta(\min\{\delta_1^3, \delta_2^2\})$, with high probability, a fresh node becomes non-fresh within $O(\Lambda \cdot \log n)$ slots.*

2.3 Complexity of LooseMAC

A network state is *stable* if all nodes in the network are ready. When a network is in a stable state in stays in it until some new node joins the network. We show now that starting from an arbitrary state I, if no changes occur in the network after I, the network reaches a stable state. Suppose for the discussion below that $\Lambda = \Theta(\min\{\delta_1^3, \delta_2^2\})$.

Let S be the set of non-ready nodes in state I. Let S_f be the set of fresh nodes in S. Lemma 2 implies that within $O(\Lambda \cdot \log n)$ slots, the network reaches a state I' in which all nodes in S_f become non-fresh, and thus, there are no more fresh nodes in the network. In I' there are only non-ready nodes in the network. Lemma 1, implies that all the non-ready nodes become ready in $O(\Lambda \cdot \log n)$ time slots. Therefore, within $O(\Lambda \cdot \log n)$ time slots, the network reaches a stable state.

The "affected" nodes are the nodes which send control messages until stabilization. The nodes in $\Delta_1(S_f)$ become all non-ready, and in I' only the nodes of $\Delta_1(S)$ are non-ready. While those nodes become ready, they communicate with their neighbors. Therefore, the nodes affected are all members of $\Delta_2(S)$. Each affected node sends at most $O(\log n)$ control messages, since in every frame it sends at most 2 messages. 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). Therefore, we have the following theorem.

Theorem 3 (Complexity of LooseMAC) *Consider an arbitrary network state I, such that no change occurs after I. With high probability, the network stabilizes within $O(\Lambda \cdot \log n)$ slots. If S are the non-ready nodes in I, then the affected area is $\Delta_2(S)$. Each affected node sends at most $O(\log n)$ messages, each message consisting of $O(\log n)$ 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^2; recall that ϕ_i is the maximum 1-neighborhood size of any 2-neighbor of i. This algorithm runs on top of LooseMAC. (To simplify the discussion, we will refer to the frames of TightMAC as "tight", and the frames of LooseMAC as "loose".)

When a node enters the network, is first runs the LooseMAC algorithm. Using the selected slot in the loose frame (the loose slot), the node communicates with its neighbors in order to compute

the size of the tight frame, and then to find a conflict-free slot (the tight slot) in the tight frame. Then the node starts using the tight frames 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.

3.1 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 so that there are three additional ready levels: ready-1, ready-2 and ready-3. A node is ready-x if all nodes in its neighborhood are ready-$(x-1)$. A node starts executing the TightMAC algorithm when it is ready-3.

The ready status changes as follows. When a node becomes ready, it broadcasts a message informing its neighbors about it. When a ready node has received ready messages from all of its neighbors it becomes ready-1. In a similar way, a ready-1 node elevates its status to ready-2, and then to ready-3. A node does not elevate its status further. In the LooseMAC algorithm, a ready-x node has the same behavior as a ready node (for example, from any ready-x mode it becomes non-ready when a fresh node arrives in the neighborhood).

3.2 Description of TightMAC

Algorithm 2 depicts the outline of Algorithm TightMAC. When a node i joins the network, it first executes the LooseMAC algorithm until it becomes ready-3. At this point it starts the execution of the main part of the TightMAC algorithm. All the control messages are sent using the loose slots, until the node switches to using the tight frames.

Algorithm 2 TightMAC(node i)

1: Execute LooseMAC(i);
2: **Whenever** i becomes ready-3:
3: Inform the 2-neighborhood about $\delta_1(i)$ and then compute ϕ_i;
4: Choose a frame F_i with $|F_i| = 2^{\lceil \log 6\phi_i^2 \rceil}$;
5: Inform the 1-neighborhood about the relative position of F_i, with respect to its loose slot;
6: Execute FindTightSlot();
7: Start using the tight frame;

Next i computes its ϕ values. When node i becomes ready-3, it knows its 1-neighborhood, since all these nodes have marked slots in its loose frame of i. Thus, i knows $\delta_1(i)$. Then i sends this value to its neighbors. Each neighbor computes the maximum of the values received and is sends it again to its neighbors. Now i knows the maximum 1-neighborhood size, of any of 2-neighbors, and thus it computes ϕ. This process requires at most 2 control messages from each node.

Then, node i chooses the size of its tight frame F_i to be equal to the smallest power of 2 bigger or equal to $6\phi_i^2$. Then i sends a message notifying its neighbors about the position of F_i with respect to the position of the loose slot. This information is needed in order for the neighbor nodes to determine whether the slots in the tight frames conflict or not. After this step, node i executes the FindTightSlot algorithm (described in Section 3.3) which computes the conflict-free slot in the frame F_i. After the tight slot is computed, node i switches to using tight frame F_i.

In order to guarantee the proper interleaving of the loose and tight frames, in F_i, in addition to the tight slot, node i also reserves slots for the loose slot and all marked slots in the loose frame. This way, the slots used by LooseMAC are always preserved and can be used by i, even in the tight frame. This is especially 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

In the heart of the TightMAC algorithm is the FindTightSlot algorithm, which is used to find conflict-free slots in the tight frames. The tight frames have different sizes at the various nodes, which depends on their local parameter ϕ_i. The different frame sizes cause additional conflicts between the selected slots of neighbors. Take two 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.

Algorithm 3 FindTightSlot()

```
1:  SlotFound ← false;
2:  while not SlotFound do
3:    Select ← false;
4:    With probability 1/φ_i^2 set Select ← true;
5:    if Select then
6:      Let s_i be an randomly chosen unreserved slot in the first 6δ_1^2(i) slots of F_i;
7:      Send the position of s_i;
8:      Listen for a period of Λ time slots;
9:      if no conflict is reported by any neighbor then
10:       SlotFound ← true;
```

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 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 will coexist.

The procedure for selecting a slot is as follows. Node i selects an unreserved slot σ_i in the first $6\delta_1^2(i)$ slots of F_i. As the analysis shows, these are enough slots in order to obtain a conflict-free slot. Then node i notifies its neighbors about σ_i. In order to do so it uses the loose slot, which repeats every Λ time slots. Each neighbor j then checks if σ_i creates any conflicts in their own slots, by examining whether s_i conflicts with reserved slots in F_j. If conflicts occur, then j responds with a conflict report message (again in the loose slot). Node i waits for Λ time slots. In this period if a neighbor detected a conflict it reports it. If i receives a conflict report then it chooses again another slot and the process repeats; otherwise, i returns the selected slot s_i.

In the algorithm, a node select a new slot in a loose frame with probability $1/\phi_i^2$. This guarantees that the new slot selection is unlikely to cause many conflicts with other nodes choosing slots at

the same time, which helps in stabilizing with small frame sizes and small number of messages.

We would like to comment on the choice of the frame size of node i, which is proportional to ϕ_i^2. Take some node j which is a 2-neighbor of i. Node j chooses a slot in the first $Z = 6\delta_1^2(j)$ slots of F_j. Node i has frame size larger than Z. So, node i cannot have more than one slot repetition in Z. This implies that i and j conflict at most once during Z. Therefore, and so the possible conflicting slots in j during Z are bounded by the 2-neighborhood size of j. This makes it easy to bound the probabilities in our analysis which is given below.

Lemma 4 *Let $j \in \Delta_2(i)$ be node which selects a slot s_j. Slot s_j causes conflicts in $\Delta_1(i)$ with probability at most $1/2$.*

Proof: Node j chooses s_j among $q_1 \geq 4\delta_1^2(j)$ unreserved slots in frame F_j (since at most $\delta_1(j)$ by the loose frame slots, and $\delta_1(j)$ are reserved FindTightSlot algorithm). When s_j is selected, there are $q_2 \leq 2\delta_2(j)$ total slots reserved in the frames in all the 1-neighbors of j. Subsequently, s_j creates a conflict in any 1-neighbor with probability at most $q_2/q_1 = 1/2$. ■

Lemma 5 *For node i, during a period of Λ time slots, conflicts occur in $\Delta_1(i)$ with probability at most $1/2$.*

Proof: Let Z be the period of Λ time slots. A conflict is caused during Z in $\Delta_1(i)$ by any slot selection in nodes $\Delta_2(i)$. A slot selection by node $j \in \Delta_2(i)$ occurs with probability at most $1/\phi_j^2$. From Lemma 5, 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. The probability that any of them causes a conflict during Z in $\Delta_1(i)$ is at most $q \leq \delta_2(i)/(2\min_{\{j\in\Delta_2(i)\}} \phi_j^2)$. Since for any $j \in \Delta_2(i)$, $\phi_j^2 \geq \delta_2(i)$, we have that $q \leq 1/2$, as needed. ■

Lemma 5, implies that every time that 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 $1/\phi_i^2$ in every repetition of loose frame, in the expected case i will select a slot within $O(\phi_i^2)$ repetitions of the loose frame. This implies the following result.

Corollary 6 *With high probability, node i successfully chooses a conflict-free time slot in F_i within $O(\phi_i^2 \Lambda \log n)$ time slots.*

3.4 Complexity of TightMAC

A network state is *stable* if all nodes in the network have selected conflict-free tight slots. When a network is in a stable state in stays in it until some node joins/leaves the network. We show now that starting from an arbitrary state I, if no changes occur in the network after I, the network reaches a stable state. Let S ne the non-ready nodes in state I.

From Theorem 3, with high probability, the LooseMAC requires $O(\Lambda \log n)$ time slots until all nodes become ready. Then, $O(1)$ time is required until the nodes become ready-3. Consider a node i. Computing ϕ_i requires $O(1)$ messages sent from node i, where each message consists of $O(\log n)$ bits. From Lemma 6, with high probability, node i requires $O(\phi_i^2 \Lambda \log n)$ time slots until it selects the conflict-free time slot. Since, $\delta_1 \geq \phi_i$, the total time until stabilization is $O(\delta_1^2 \Lambda \log n)$. The affected slots are the ones in $\Delta_3(i)$. With an analysis similar to Corollary 6, we can show that each affected node sends at most $O(\log n)$ control messages until stabilization, each with $O(\log n)$ bits. Combining all the results, we obtain the following theorem.

Theorem 7 (Complexity of TightMAC) *Consider an arbitrary network state I, such that no change occurs after I. With high probability, the network stabilizes within $O(\delta_1^2 \cdot \Lambda \cdot \log n)$ slots. If S are the non-ready nodes in I, then the affected area is $\Delta_3(S)$. Each affected node sends at most $O(\log n)$ messages, each message consisting of $\Theta(\log n)$ bits.*

References

[1] N. Abramson, "The ALOHA system - another alternative for computer communications," Proceedings of the AFIPS Conference, vol. 37, pp. 295-298, 1970.

[2] M. H. Ammar and D. S. Stevens, "A distributed TDMA rescheduling procedure for mobile packet radio networks," Proceedings of IEEE ICC, pp. 1609-1613, Denver, CO, June 1991.

[3] A. Cerpa and D. Estrin, "ASCENT: Adaptive Self-configuring sensor network topologies," in Proc. of INFOCOM'02, 2002.

[4] B. Chen, et al, "Span: An Energy-efficient Coordination Algorithm for topology maintenance in ad hoc wireless networks," in Proc. MOBICOM'01, 2001.

[5] I. Chlamtac, W. R. Franta and K. Levin, "BRAM: The broadcast recognizing access method," IEEE Transactions on Communications, vol. 27, no. 8, August 1979.

[6] I. Cidon and M. Sidi, "Distributed assignment algorithms for multihop packet radio networks," IEEE Transactions on Computers, vol. 38, no. 10, pp. 1353-1361, October 1989.

[7] T. ElBatt and A. Ephremides "Joint Scheduling and Power Control for Wireless Ad hoc Networks," IEEE Computer and Communications Conference (INFOCOM), June 2002.

[8] D. Farber, J. Feldman, R. Heinrich, D. Hopwood, K. Larson, D. Loomis and L. Rowe, "The distributed computing system," Proceedings of IEEE COMPCON, pp. 31-34, San Francisco, CA, February 1973.

[9] L. Kleinrock and M. O. Scholl, "Packet switching in radio channels: New conflict-free multiple access schemes," IEEE Transactions on Communications, vol. 28, no. 7, pp. 1015-1029, July 1980.

[10] L. Kleinrock and F. A. Tobagi, "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.

[11] J. Martin, Communication Satellite systems, Prentice Hall, New Jersey, 1978.

[12] J.P. Monks, V. Bharghavan, and W. Hwu, "A Power Controlled Multiple Access Protocol for Wireless Packet Networks," IEEE INFOCOM 2001, Anchorage, Alaska, April, 2001.

[13] A. Muqattash and M. Krunz, "Power controlled dual channel (PCDC) medium access protocol for wireless ad hoc networks," in Proceedings of the IEEE INFOCOM 2003 Conference, San Francisco, April 2003.

[14] A. Muqattash and M. Krunz, "CDMA-based MAC protocol for wireless ad hoc networks," in Proceedings of the ACM MobiHoc 2003 Conference, Annapolis, Maryland, June 2003.

[15] R. Nelson and L. Kleinrock, "Spatial TDMA: A collision free multihop channel access protocol," IEEE Transactions on Communications, vol. 33, no. 9, pp. 934-944, September 1985.

[16] V. Rajendran, K. Obraczka, J.J. Garcia-Luna-Aceves. "Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks," in Proc. SENSYS03, 2003.

[17] L. G. Roberts, "ALOHA packet system with and without slots and capture," Computer Communications Review, vol. 5, no. 2, April 1975.

[18] S. Singh and C. S. Raghavendra, "Power efficient MAC protocol for multihop radio networks," in the Nineth IEEE ISPIMRC'98, pp:153-157, 1998.

[19] F. Tobagi and L. Kleinrock, "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.

[20] T. V. Truong, "TDMA in mobile radio networks: An pp. 504-507, Atlanta, GA, Nov, 1984.

[21] A. J. Viterbi, CDMA: Principles of Spread Spectrum Communication, Addison-Wesley, Reading, MA, 1995.

[22] S.L. Wu, Y.C. Tseng, and J.P. Sheu, "Intelligent medium access for mobile ad-hoc networks with busy tones and power control," IEEE JSAC, 18(9):1647–1657, 2000.

[23] Y. Xu, J. Heidemann and D. Estrin, "Geography-informed Energy conservation for ad hoc networks," in Proc. MOBICOM'01, 2001.

[24] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE standards 802.11, January 1997.

Appendix

A Algorithm LooseMAC

In this section, we discuss the LooseMAC algorithm in more detail. In the basic description of the algorithm, recall that nodes are referred to as being "ready" when they select a conflict-free time slot. In the actual implementation of the algorithm, this corresponds to just one the three states (modes) a node can be in: *NEWSLOT*, *WATCH* and *READY*. The *READY* mode here corresponds to the "ready" state in the basic discussion of the algorithm. All nodes start in the *NEWSLOT* mode by selecting a random time slot. After the node transmits a control message in its randomly selected slot, the node switches into the *WATCH* mode, where it checks if its randomly selected slot is collision-free. If it is, then the node goes into the *READY* mode. Else the node goes back to the *NEWSLOT* mode by selecting a new random time slot. Recall that the aim of the algorithm is to reach a network state where every node in the network is *READY*, i.e. each node has a conflict-free time slot for transmission.

The detailed pseudo-code for LooseMAC is given in Algorithm 4, with the auxiliary functions Send, Receive and UpdateMode given respectively in Algorithms 5, 6 and 7. Let us go over the algorithm by considering how node i executes the algorithm and reaches the *READY* state. Node i starts the algorithm by selecting a random time slot σ and initializing its mode to *NEWSLOT*. In the current time frame, node i listens the channel for every time slot, but broadcasts only in σ. Note that the Send function is called in every time slot, but it does nothing in effect if the current time slot is not the node's selected time slot, σ. If the current time slot is indeed the selected time slot σ, then the node broadcasts a control message including its id, a bit indicating any observed conflicts, and a bit indicating whether the node is "fresh" (see the basic algorithm discussion in section 2.1).

Unlike Send function, the Receive function is literally executed in every time slot, i.e. the node listens the channel in every time slot. When listening the channel in a time slot, it may either receive noise, or receive and decode a message successfully. Receiving noise means that two or more neighbors of i are transmitting at the same time, thus i sets its *Conflict* variable to true. If i receives a message $\langle j, Conflict_j, Fresh_j \rangle$, it checks in its current frame if this time slot was reserved for any node other than j. If so, it again sets its *Conflict* variable to true. Else, it reserves this time slot for node j. Finally, if $Conflict_j$ is true, this indicates that j has recently observed a collision. To keep this information, node i sets its $ConflictInNeighbor$ variable to true.

When node i transmits in its randomly selected slot σ, it switches its mode to *WATCH*, which lasts for at most Λ consecutive time slots. When in *WATCH* mode, node i effectively checks if σ is a free time slot in its 2-neighborhood (i.e. σ is collision-free). In order for i to determine that σ is collision-free, it first needs to ensure that no neighbor of i is transmitting at σ. Node i can simply accomplish this by checking if it receives any transmission in slot σ. Even if no node in the immediate neighborhood of i has selected σ, this time slot may still not be collision-free because of a 2-neighbor of i and because of the hidden terminal problem, as mentioned earlier. Hence, node i relies on the *reports* of its neighbors to see if its time slot is indeed collision-free. If i is in *WATCH* state and receives a message $\langle j, Conflict_j, Fresh_j \rangle$ in which $Conflict_j$ is true, the conflict observed at node j may be due to i's transmission. Moreover, if i receives a garbled message in any time slot when it is in *WATCH* mode, it can never know whether the transmitted control messages carried conflict information or not. Hence node i goes into *READY* state only if it receives no garbled

messages (collisions) during the last Λ time slots in which it is in *WATCH* mode, and each control message it receives during this period has the *Conflict* bit reset to false. With the local variables set in the Receive function, the node calls the UpdateMode method in each time slot in order to handle the discussed state transitions, if necessary.

Algorithm 4 LooseMAC(node i)

```
1: Unmark all slots in frame;
2: Fresh ← true;
3: Conflict, LastConflict, ColissionInNeighbor, FreshNeighbor ← false;
4: σ ← random slot in frame;
5: Mode ← NEWSLOT;
6:
7: Divide time into frames each with duration T, consisting of Λ time slots;
8: τ ← 1; {the current slot}
9: {this loop repeats at every time slot}
10: while true do
11:    Send();
12:    Receive();
13:    UpdateMode();
14:    τ ← τ + 1;
15:    if τ = Λ + 1 then
16:       τ ← 1;
```

Algorithm 5 Send()

```
1: if τ = σ then
2:    if Mode = NEWSLOT then
3:       Broadcast⟨i, Conflict, Fresh⟩;
4:    if Mode ∈ {WATCH, READY} then
5:       if Conflict then
6:          Broadcast⟨i, Conflict, Fresh⟩;
7:    LastConflict ← Conflict;
8:    Conflict ← false;
```

B Analysis of LooseMAC

B.1 Basics

Suppose for the moment that the time slots at each node are aligned. We will discuss later the case for when the time slots are not aligned. We will refer to the absolute positions of time slots according to some common reference, which is the first ever time slot of the first node that joined the network. Unless otherwise stated, we will refer to absolute time slots.

Consider a time slot α of a node v. Slot α has an *absolute position*, which is the number of slots passed since i became operational. and a *relative position* which is the number of time slots passed since the beginning of the current frame.

Algorithm 6 Receive()

```
1: if received noise then
2:    Conflict ← true;
3: else
4:    if receipt of message ⟨j, Conflict_j, Fresh_j⟩ then
5:       if Fresh_j then
6:          FreshNeighbor ← true;
7:       if τ = σ then
8:          Conflict ← true;
9:       else
10:         if Conflict_j then
11:            ConflictInNeighbor ← true;
12:         if (slot τ is marked with some node k, k ≠ j) then
13:            Conflict ← true;
14:         else
15:            Unmark any previously marked slot with j;
16:            Mark slot τ with j;
```

Algorithm 7 UpdateMode()

```
1: if τ = σ then
2:    NewMode ← Mode;
3:    if Mode = NEWSLOT then
4:       NewMode ← WATCH;
5:    if Mode = WATCH then
6:       AnyConflict ← LastConflict or Conflict or ConflictInNeighbor;
7:       if AnyConflict or FreshNeighbor then
8:          σ ← random unmarked slot in frame;
9:          NewMode ← NEWSLOT;
10:      else
11:         NewMode ← READY;
12:      if not AnyConflict then
13:         Fresh ← false;
14:   if Mode = READY then
15:      if FreshNeighbor then
16:         σ ← random unmarked slot in frame;
17:         NewMode ← NEWSLOT;
18:   Mode ← NewMode;
19:   FreshNeighbor, ConflictInNeighbor ← false;
```

For each node i, we will use the subscript i to denote its local variables, for example for i variable $Fresh$ is denoted $Fresh_i$. The *current* value of a variable is value of the variable at the beginning of the current time slot. We will always refer to the current values of the variables, unless otherwise stated. A variable *switches* value at a time slot if at the value of the variable is different at the beginning and at the end of the slot. We say that at slot α node i is in the X mode, or simply node i is X, if the current value of $Mode_i$ is X (where X is either $NEWSLOT$, $WATCH$, or $READY$).

An (absolute) time slot is *special* for node i if the respective relative lost is the selected slot of i. Note that a node has at least one special slot every T time steps. Special time slots are important for two reasons: (i) a node switches its mode only at special time slots, and (ii) a node broadcasts control messages only at special time slots. Let m be a control message that a node i sends any special time slot α. If in α the mode of i is X, then we say that m is X (where X is either $NEWSLOT$, $WATCH$, or $READY$). From the algorithm description we have the following result.

Lemma 8 *In any period of Λ time slots, a node has either one or two special time slots; further, it sends at most one $NEWSLOT$ message in this period.*

We say that a *conflict* occurs in node i at slot α if any of the following three events occur in α: (i) two messages collide and i listens to noise, or (ii) node i receives a message m from a neighbor j without collisions and α is special for i, or (iii) node i receives a message m from a neighbor j without collisions but slot α is marked in i with another node $k \neq j$. In either case, the variable $Conflict_i$ is set to true in α. Let β be the immediate special slot after α. According to the algorithm, if a conflict occurs in i node i will broadcast a control message with $Conflict_i = true$ in β.

B.2 Fresh Nodes

We say that at any time slot a node i is *fresh* if $Fresh_i = true$; otherwise the node is *non-fresh*. When a node joins the network it is fresh. A node becomes non-fresh at a time slot at which it is in the $WATCH$ mode and its local variable $AnyConflict$ is false, that is no conflict in the node i nor in its neighbors. We show the following lemmas.

Lemma 9 *Consider a node i which switches to non-fresh at a time slot α. Then no neighbor of i is $READY$, nor switches to $READY$, at α.*

Proof: Let $\beta = \alpha - \Lambda$. According to the algorithm, at special slot α node i is in $WATCH$ mode, while at special slot β node i is in $NEWSLOT$ mode. Thus, at slot β, node i broadcasts a control message m_i with $Fresh_i = true$. Let j be some neighbor of i at slot α. We examine two cases:

- *Node j joined the network after β.* When a node becomes active its initial mode is $NEWSLOT$, and it clearly requires at least Λ additional slots before it switches to the $READY$ mode, an event that occurs after α. Therefore, at slot α node j is not in the $READY$ mode, nor switches to the $READY$ mode.
- *Node j joined the network at or before β.* In slot β message m_i is received by j with or without conflicts. We examine each case separately.

Suppose that j detects a conflict in β. Then j broadcasts a control message m_j with $Conflict_j = true$ in the next special slot γ. Slot γ appears no later the Λ time slots after β; thus γ is before or concurrent with α. During γ, node i receives m_j with or without conflicts. In either case, $AnyConflict_i$ is set to $true$, which implies that at slot α node i remains fresh. A contradiction.

Therefore, m_i does not create a conflict in j at time slot β. Then, clearly, β is not a special slot for j. When j receives the message m_i it sets its local variable $FreshNeighbor_j$ to $true$. Let γ be the first special slot of j after β. Slot γ appears before α, since the number of slots from β (including β) up to α (excluding α) are in total Λ. If in γ node j is in either the $WATCH$ or $READY$ modes, it switches to the $NEWSLOT$ mode (since $FreshNeighbor_j = true$). This implies that j requires at least Λ more slots before it becomes $READY$, an event that can only occur after α. If in γ node j is in the $NEWSLOT$ mode, then it will require at least $\Lambda - 1$ time slots before it switches to the $READY$ mode, an event that occurs after α. Therefore, in all cases, node j is not in the $READY$ mode in α, nor switches to the $READY$ mode in α.

■

B.3 Knowing Time Slots of Neighbors

Let σ_i be the relative position of the currently selected slot of node i. Let ψ_j be the respective relative position of slot σ_i in node j. Let j be a neighbor of i. We say that node j *knows the slot* of i, if j has marked slot ψ_j with i. From the proof of Lemma 9, we obtain the following.

Lemma 10 *Let α be a time slot at which node i switches to non-fresh or $READY$. Let β be the special slot of i immediately before α. and let S be the set of neighbors of i which participate at β. At time slot a, every node in S knows the current time slot of i.*

Next, we show that if two neighbor nodes are $READY$ at the same time then they know each-other's time slots.

Lemma 11 *At any time in which two neighbor nodes are $READY$, they know each other's slots.*

Proof: Let α be any time slot in which two neighbor nodes i and j are $READY$. We will show that j knows the slot of i in α. The opposite direction (i knows teh slot of j) is symmetric.

Suppose for contradiction that j doesn't know the slot of i in α. Let β be the last special time slot before α in which i switches to the $READY$ mode. Let $\gamma = \alpha - \Lambda$. From Lemma 10, node j cannot be participating in the network in γ. Therefore, j joined the network after γ. Let ζ be the time slot at which j becomes non-fresh. Clearly, ζ occurs after β, since at least Λ slots are needed in order to switch to the non-fresh mode from the moment of joining the network. From Lemma 9, no neighbor of j will be $READY$ in slot ζ. Therefore, i is not ready in ζ, which implies that i becomes $READY$ again at a slot η after β and before α. This contradicts the choice of β. Therefore, j knows the slot of i at α. ■

B.4 Stable State

A network state is *stable* if each node is in the $READY$ state. Clearly in a stable state there are no conflicts of messages, since from Lemma 11 each node knows the slots of all of its neighbors, and thus these slots are conflict-free. Once such a stable state is reached, the network remains in the stable state while no topological changes occur (no node joins or leaves the network). We study now the conditions under which we can reach a stable state. A useful lemma in our study is derived from from the algorithm description, and Lemma 9:

Lemma 12 *A $READY$ node remains $READY$ unless it becomes adjacent to a fresh node.*

Suppose we are given an arbitrary network state I_0. We will show how we can reach a stable state from I_0, assuming that no topological changes occur after I_0. Let S denote the set of participating nodes in state I_0. We define the following subsets of S: set A contains nodes which in state I_0 are not $READY$; set B contains nodes which is state I_0 are fresh; set C contains nodes which are neighbors of B. Let $A' = A - B \cup C$, that is, A' is the set of nodes which are not $READY$ in I_0 and not neighbors with fresh nodes. Let $S' = S - A \cup B$, that is S' is the set of nodes which are $READY$ in I_0 and not neighbors with fresh nodes. From Lemma 12, we have the following containment theorem, which shows that the nodes which are $READY$ and not adjacent to fresh nodes are not affected by changes in the network.

Theorem 13 (Containment) *The nodes of S' remain $READY$ after I_0.*

Suppose now that there exists a network state I_1, such that between state I_0 and I_1 each node in A' switches to $READY$ and each node in B becomes non-fresh. After state I_1, no node is fresh. From Lemma 12, all the nodes in A' remain $READY$ after I_1. Let D denote the set of nodes which are not $READY$ in state I_1. From Lemma 9, each node of set C switch to a non-$READY$ state between I_0 and I_1. Thus, the only nodes which are not possibly ready at I_1 are the nodes of B and their neighbors C. Therefore, $D \subseteq B \cup C$.

Suppose now that there exists a network state I_2, such that between state I_1 and I_2 each node in D switches to $READY$. Then, from Lemma 12, all the nodes in D remain $READY$ after I_2. Thus state I_2 is stable, and we have the following result.

Lemma 14 *Given an arbitrary state I_0, the network reaches a stable state if first the nodes of set A' switch to $READY$ and the nodes of set B become non-fresh, and then the nodes of set D switch to $READY$.*

B.5 Becoming $READY$

Here we show that the nodes of set A' reach the ready state within $O(\Lambda \log n)$ time slots since I_0. For the discussion below, assume that node i is not adjacent to fresh nodes.

Lemma 15 *Let m be a $NEWSLOT$ message which is broadcasted by node i. Let $S \subseteq \Delta_1(i)$. The probability that message m conflicts in S is at most $4\min\{\delta_1|S|, \delta_2\}/\Lambda$.*

Proof: Let α be the slot at which message m is sent. Node i chooses α randomly among $q_1 \leq \Lambda - \delta_1$ unmarked slots in a sequence of Λ slots Z, since there are at most δ_1 marked slots from the neighbors

of i. Let $j \in S$. From the Z slots in j, $q_2^{(j)} \leq \delta_1$ of them are marked with neighbors of j. In $q_3^{(j)} \leq 2\delta_1$ slots, j receives messages from adjacent nodes (since from Lemma 8, each adjacent node of j sends at most 2 control messages in Z). Therefore, m can conflict in at most $q^{(j)} = q_2^{(j)} + q_3^{(j)} \leq 3\delta_1$ slots in j. The probability that m causes a conflict in j j is at most $q^{(j)}/q_1$.

Now we consider all nodes in S. Let $q_2 = \sum_{j \in S} q_2^{(j)}$, $q_3 = \sum_{j \in S} q_3^{(j)}$, and $q = q_2 + q_3$. The probability that m conflicts in S is at most q/q_1.

Now, we give an upper bound on q. Clearly, $q_2 \leq \delta_1|S|$, and $q_3 \leq 2\delta_1|S|$, and thus $q \leq 3\delta_1|S|$. Moreover, $q_2 \leq \delta_2$, since the number of slots marked in the nodes on S are no more than the size of the 2-neighborhood of i, that is $q_2 \leq \delta_2$. Similarly, $q_3 \leq 2\delta_2$. Therefore $q \leq 3\delta_2$, which implies that $q \leq 3\min\{\delta_1|S|, \delta_2\}$.

Subsequently, message m conflicts in S with probability at most $q/q_1 = 3\min\{\delta_1|S|, \delta_2\}/\Lambda - \delta_1 \leq 4\min\{\delta_1|S|, \delta_2\}/\Lambda$. ■

Lemma 16 *Let Z be a period of $k\lambda$ time slots. Let W a period consisting of Z and the Λ time slots before Z. Conflicts occur in a node i during Z, only if $NEWSLOT$ conflicts occur in i during W.*

Proof: In Z a conflict occurs in i by $NEWSLOT$ messages or by $WATCH$ messages. (Note that from Lemma 10, a $READY$ message cannot cause a conflict, unless it collides with either a $NEWSLOT$ or a $WATCH$ message, which are the kinds of conflicts we consider below.) We will show that the conflicts caused by $WATCH$ messages in Z can be considered as $NEWSLOT$ message conflicts in W.

Consider a $WATCH$ message m, sent by node $j \in \Delta_1(i)$, at time slot α of Z. There are the following kinds of conflicts that are caused by m in i:

(1) m collides with a $NEWSLOT$ message: this case is covered by the $NEWSLOT$ message conflicts.

(2) m collides with a $WATCH$ message: Let l be the neighbor of i which sends the $WATCH$ message m' that collides with m. Consider the time slot $\beta = \alpha - \Lambda$. Nodes j and l must have sent $NEWSLOT$ messages in β. Thus, the conflict in α is caused because of the $NEWSLOT$ message conflict in β.

(3) α is the selected slot of i: if in α node i is in $NEWSLOT$ mode, then it sends a $NEWSLOT$ message. If in α node i is not in $NEWSLOT$ mode, then in β node i had the same selected slot and j was in the $NEWSLOT$ mode. Thus, the conflict in α can be treated as a $NEWSLOT$ message conflict in either α or β.

(4) α is marked with a node different than i: this case is similar to case (3).

Subsequently, a $WATCH$ message conflict in α can be treated as a $NEWSLOT$ conflict in either α or β. This implies that conflicts from $WATCH$ or $NEWSLOT$ messages are caused in Z only if conflicts from $NEWSLOT$ messages are caused in W. ■

Lemma 17 *Consider a node i and a period Z of $k\lambda$ time slots. During Z, conflicts occur in $\Delta_1(i)$ with probability at most $4(k+1)\min\{\delta_1^3, \delta_2^2\}/\Lambda$.*

Proof: From Lemma 16, we only need to find an upper bound on the probability that conflicts are caused by $NEWSLOT$ messages in period W (this period is defined in the statement of Lemma 16). The duration of W is $(k+1)$ slots. Conflicts in $\Delta_1(i)$ are caused only from messages sent by $\Delta_2(i)$ during W. From Lemma 8, during Z, each node in $\Delta_2(i)$ sends at most $k+1$ $NEWSLOT$ messages. From Lemma 15, each of these messages causes conflicts in $\delta_1(i)$ with probability at most $4\delta_2/\Lambda$. Since, there are at most $(k+1)\delta_2$ messages, we have that conflicts occur with probability at most $4(k+1)\delta_2^2/\Lambda$.

We can also compute an alternative bound for this probability. Conflicts occur in $\Delta_1(i)$ if conflicts occur in any of the individual nodes in $\Delta_1(i)$. Let $j \in \Delta_1(i)$. Each neighbor of j, sends at most $k+1$ $NEWSLOT$ messages during W. From Lemma 15, each such message causes a conflict in j with probability at most $4\delta_1/\Lambda$. Since there are at most $(k+1)\delta_1$ such messages, a conflict occurs in W with probability at most $4(k+1)\delta_1^2/\Lambda$. Since there at most δ_1 nodes similar to j, we have that conflicts occur in $\Delta_1(i)$ with probability at most $4(k+1)\delta_1^3/\Lambda$.

Combining the two upper bounds we have that the probability of conflicts in $\Delta_1(i)$ during Z is at most $4(k+1)\min\{\delta_1^3, \delta_2^2\}/\Lambda$. ■

Lemma 18 *Let α be a special slot for node i, in which i is in $WATCH$ mode. In slot α node i does not switch to the $READY$ mode with probability at most $16\min\{\delta_1^3, \delta_2^2\}/\Lambda$.*

Proof: Let β be the immediate previous special slot of i before α. Denote by Z the time period starting at β and ending at α (period Z includes β and α). Node i does not switch to the $READY$ mode in α if either of the following two events occurs. E_1: A conflict occurs in i in Z; E_2: A node adjacent to i reports a conflict during Z.

Consider now event E_2. Let j be a neighbor of i that reports a conflict with sending a message m in Z. Suppose that message m is received by i without conflicts (the case where m conflicts is covered in E_1). Let Z' denote the period consisting of Z and the Λ slots immediately before β. Message m is sent if a conflict occurs in j during Z'. Therefore, combining E_1 and E_2, we have that i does not switch to the $READY$ state in α, only if during Z' conflicts occur in $\Delta_1(i)$.

Since Z' consists of $2\Lambda+1 \leq 3\Lambda$ time slots, from Lemma 17, conflicts occur in Z' with probability at most $16\min\{\delta_1^3, \delta_2^2\}/\Lambda$. ■

Let $\Lambda \geq 32\min\{\delta_1^3, \delta_2^2\}$. Lemma 18 implies the following corollary.

Corollary 19 *Let α be a special slot for node i, in which i is in $WATCH$ mode. In slot α node i switches to the $READY$ mode with probability at least $1/2$.*

Corollary 19 further implies the following result.

Corollary 20 *With high probability, all nodes of set A' become $READY$ within $O(\Lambda \log n)$ slots after state I_0.*

B.6 Becoming non-Fresh

Here we show that the nodes of set B become non-fresh within $O(\Lambda \log n)$ time slots since I_0. When a node $i \in B$ joins the network, it can cause two nodes which were not 2-neighbors before, to become 2-neighbors. This implies that two $READY$ neighbors of i can possibly cause conflicts in i, since they could have selected overlapping time slots. In order to avoid conflicts of this form,

fresh node i forces every neighbor to become non-$READY$. Then those neighbors select new time slots which are guaranteed to be conflict-free when they become $READY$ again. We show below that each node in C switches to the $NEWSLOT$ mode (becomes non-$READY$) after state I_0, due the fresh nodes B.

Lemma 21 *With high probability, every node in C switches to the $NEWSLOT$ mode within $O(\Lambda \log n)$ time slots after I_0.*

Proof: Consider a node $i \in B$. Let t be a time at $k\Lambda \log n$ time slots after I_0, for some $k \geq 1$. We will show that node i forces all neighbors to enter the $NEWSLOT$ mode by time t. There are two possible events:

- *By time t node i becomes non-fresh.* In this case, Lemma 9, guarantees that between I_0 and time t every neighbor of i switches to the $NEWSLOT$ mode.
- *By time t node i remains fresh.* Let α be a time slot between t and I_0, in which node i is in $NEWSLOT$ mode and broadcasts message m. Lemma 15 implies that message m create conflicts in $\Delta_1(i)$ with probability at most $4\delta_1^2/\Lambda \leq 1/8\delta_1 \leq 1/8$. Since node i is not-fresh, it is not $READY$ either, and thus it broadcasts a $NEWSLOT$ message every 2Λ time slots. Therefore, with high probability, one of these $NEWSLOT$ messages is received without conflicts by all neighbors by time t. This implies that with high probability all neighbors of i switch to the $NEWSLOT$ mode by time t.

Considering now all the nodes in B, we have that with high probability every node in C switches to the $NEWSLOT$ mode by time t. ■

Let I' ne a network state in which all the nodes in C have entered the $NEWSLOT$ mode after I_0. After state I', when the C nodes enter the $READY$ mode, their $READY$ messages will not cause conflicts in neighbors (a consequence of Lemma 10). Therefore, conflicts are caused only from $NEWSLOT$ and $WATCH$ messages. Therefore, after state I' we can use an analysis similar to subsection B.5, to obtain that with high probability all nodes of set B become non-fresh within $O(\Lambda \log n)$ slots after state I'. Lemma 21, implies the following result.

Corollary 22 *With high probability, all nodes of set B become non-fresh within $O(\Lambda \log n)$ slots after state I_0.*

B.7 Complexity

For $\Lambda \geq 32 \min\{\delta_1^3, \delta_2^2\}$, Corollaries 20 and 22 imply that within $O(\Lambda \log n)$ time slots after I_0, all nodes in A' become $READY$ and all nodes in B become non-fresh. Thus I_1 occurs no later than $O(\Lambda \log n)$ time slots after I_0. Using a result similar to Corollary 20, we can also show that the nodes in D become $READY$ within $O(\Lambda \log n)$ after I_1. Lemma 14 implies the main result for the stabilization time.

Theorem 23 (Stabilization Time) *Given an arbitrary network state I_0 with no topological changes occurring after I_0, with high probability the network reaches a stable state within $O(\Lambda \log n)$ time slots.*

Lemma 8 and Theorem 23 imply the following corollary.

Corollary 24 (Message Complexity) *Since state I_0, each node sends $O(\log n)$ control messages, each with $\Theta(\log n)$ bits, until stabilization.*