Analysis of a Self-organizing Algorithm for Time Slot Selection in Schedule-based Medium Access

Lodewijk van Hoesel, Paul Havinga

Department of Electrical Engineering, Computer Science and Mathematics, University of Twente
Postbus 217, NL-7500 AE Enschede, THE NETHERLANDS
E-mail: l.f.w.vanhoesel, p.j.m.havinga@utwente.nl

***Abstract*—To ensure a long-lived network of wireless communicating sensors, it is necessary to have a medium access control protocol that is able to prevent energy-wasting behaviour like idle listening, hidden terminal problem or collision of packets. Schedule-based medium access protocols are in general robust against these effects, but require a mechanism to establish non-conflicting schedules. We present such a scheduling mechanism, which allows wireless sensors to choose a time interval for transmission, which is not interfering or causing collisions with other transmissions. We analyze the scheduling mechanism in the case that many nodes enter the time interval selection procedure simultaneously and potentially multiple selection rounds are required before each node has a non-conflicting schedule. In our proposed solution, we do not assume any hierarchical organization in the network and all operation is localized, making the network self-configuring.**

I. Introduction

We argue that schedule-based MAC protocols are well-suited for (energy-constrained) wireless sensor networks (WSNs), since energy-wasting effects like idle listening, hidden terminal problems and collisions of packets are minimized. In addition, schedule-based medium access provides good data throughput characteristics [10]. For these reasons, this type of medium sharing is well suited for WSNs.

A drawback of schedule-based medium access protocols is that they often lack the ability to self-organize, which is a key issue for many wireless sensor network applications. In general, these networks are assumed to be large networks, both in number of nodes and coverage area [1]. This makes (manual or central) configuring during network deployment an unattractive option. Hence, to ensure scalability of the network, WSNs must be self-organizing.

Our motivation stems from application scenarios of the AWARE project (EU IST-2006-33579). The project considers self-deploying of WSNs with autonomous helicopters [8]. The AWARE platform targets to enable operation in sites which are difficult or impossible to access and which are without a pre-existent communication structure. One of the focus application scenarios of the AWARE project is disaster management and civil security, in which wireless sensors collaboratively detect critical events (such as fire), or continuously monitor physical conditions of fire brigade personnel, e.g. to prevent them from overheating. In this safety critical application, wireless sensors are the ears and eyes of the AWARE platform, they are added to the network on-the-fly and might be attached to mobile objects. Communication needs to be reliable: self-starting and self-organizing properties are key in the effectiveness of the AWARE platform.

In this paper, we propose a self-organizing medium access scheduling principle and analyse its behaviour during network initialization. With an eye to limited hardware capabilities of wireless sensors, the proposed medium access scheme is kept very simple. Each node gets periodically a time interval — a so called time slot– in which it is allowed to control the wireless medium according its own requirements and needs. Outside this interval, nodes are notified when they are intended receivers. When a node is not needed for communication, it switches its transceiver to standby and is hence able to conserve energy.

II. Assumptions

Our medium scheduling mechanism depends on the following assumptions:

1) **Spatial medium reuse** — The RTS/CTS handshaking mechanism proposed by Bharghavan et al. [3] and the associated channel assessment mechanisms are adequate to prevent collisions or interference to on-going data transmissions. Consequently, the wireless medium can indeed be spatially reused.
2) **Correctness of received information** — The physical layer is able to provide feedback to the data link layer about the correctness of the received packet and incorrect packets are discarded. In addition, the physical layer is able to provide the reason of incorrect packets. In our work, we assume that the physical layer can for example distinguish between corrupt packets due to (random) bit errors and due to collisions. In addition, we assume that transceivers used in wireless sensor nodes are half duplex (they either transmit or receive).
3) **Timing** — Nodes can derive the moment at which packets are sent to facilitate synchronization. Synchronization is key-issue in schedule-based medium access. We assume network-wide synchronization between nodes.

For self-starting of the network, we make the following assumption. We presume that at least one gateway node is present in the wireless sensor network and we require *exactly one* gateway node to initiate the scheduled medium access in

order to prevent multiple misaligned schedules. This gateway is not involved in assigning time slots to nodes; its data link layer is identical to that of other nodes.

Our medium access approach is in particular useful in multi-hop networks in which spatial reuse and autonomous configuration are important. In addition, the medium access scheme is robust against mobility and dynamic topologies (e.g. due to iterative deployment), because of the inherent characteristics of its self-configuring.

III. Scheduling mechanism for medium access

In this section, we present a lightweight medium access scheduling algorithm that allows nodes to (autonomously) choose a time slot, which is not interfering with the communication between other nodes in the network. In addition, the algorithm resolves schedule conflicts, which might occur, for example, when nodes are mobile and travel through the network. A key issue and novel in the presented work is self-organizing reuse of the wireless medium in combination with scheduled medium access.

First, we present the basic structure of medium access scheduling. Then, we discuss how nodes can autonomously choose a conflict-free time slot.

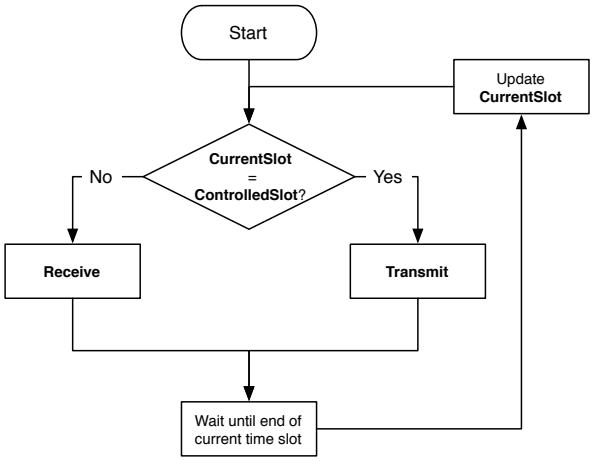


Fig. 1. Medium access scheduling principle

A. Scheduling principle

We propose a very simple scheduled medium access mechanism for wireless sensor networks. Each node takes control of (at least) one time slot. We refer to this time slot as *controlled time slot* and during this time slot, a node is allowed to *transmit* packets. Thus, a node uses only its controlled time slot to transfer data to neighbouring nodes. During the time slots of other nodes, a node *receives* packets. If a node decides that it is not required for communication during the current time slot, it switches its transceiver to low-power mode in order to conserve energy. Figure 1 illustrates the scheduling mechanism.

In our approach, we assume that the schedule in the network is fixed and is repeated periodically. For this purpose, we introduce the concept of frames —consisting of a (integer) number of time slots– to indicate the period of scheduling. The variable CurrentSlot (Figure 1) indicates the current slot position in the MAC frame.

Wireless sensor networks are assumed to be *always on* networks and thereof we implicitly require the nodes to keep the communication structure intact at all times. Nodes indefinitely (until their power source runs out) stick to their controlled time slot, unless one of the following happens:

- **Conflict/collision** — Due to, for example, mobility of nodes or dynamic quality of radio links, nodes controlling the same time slot may come too close. In this case, we speak of a collision. When a collision occurs, receiving nodes are not able to decipher packets and thus the nodes causing the collision should reconsider their controlled time slot. The topic of resolving collisions is discussed in Section V.
- **Synchronization errors** — In schedule-based MAC protocols, timing and synchronization are, for obvious reasons, of vital importance. When any synchronization error occurs, action should be taken to restore correct synchronization.
- **No neighbouring nodes** — When a node does not detect any node around it, it is not able to transfer its sensor readings or exchange any other messages. To keep trying to communicate, the node wastes energy. Therefore, it should fall back to its pre-deployment state and give up its controlled time slot.

B. Localized algorithm for conflict-free time slot choosing

The network diameter of WSNs is expected to be larger than the transmission and interference ranges of the individual wireless sensors. WSNs are thus assumed to be multi-hop networks, which allows for *spatial reuse* of the wireless medium. Obviously, this is beneficial for the network, because more data can be transported per second per meter (i.e. higher transport capacity) [2]. But it also requires the MAC protocol to take measures for ensuring successful transmissions and to prevent problems like the well-known *hidden terminal problem*. In this section, we present a localized algorithm for conflict-free time slot choosing in which time slots can spatially be reused. In general, nodes lack sufficient memory to obtain and maintain a global view of the network topology. Hence, a localized algorithm is required.

We observe that MAC protocols, built upon the handshaking mechanism proposed by Bharghavan et al. [3], prevent nodes to use the wireless medium when it would disturb ongoing communications. The handshaking mechanism of these contention-based MAC protocols consists of two steps: (1) reservation of the medium by the transmitting node and (2) reservation by the receiving node.

In step (1), the medium is reserved by the RTS message and all nodes in range of the transmitter postpone transmissions. In step (2), the receiver replies with a CTS message and all

nodes —that are able to receive this message– postpone their transmissions. Note that this method of medium reservation only works under the assumption of mutual radio coverage.

We propose a similar solution for medium reservation in schedule-based MAC protocols. In our solution, we assume the scheduling mechanism as presented in Section III-A. Nodes determine what time slots are available for use and what time slots interfere with other nodes. We require each node to transmit *at least once* during its controlled slot(s). Through this method, we make sure that all nodes in radio range are aware of the node, comparable to the reservation of the medium by the transmitting node (RTS) in the above described handshaking mechanism.

1) Medium reservation by transmitting nodes: Nodes, which are trying to find a non-interfering time slot, remove all time slots in which a message is received (or a carrier is detected) from the list of potential non-interfering candidates. There are two obvious reasons not to choose a time slot, which is already in use by a node in radio range: (1) a node would not be able to exchange messages with one or more of its neighbours, since nodes are only allowed to transmit messages in their controlled time slot. And (2) a node would potentially cause collisions, such that it might block other nodes for communicating with their neighbours.

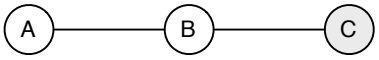


Fig. 2. Node C should not choose the time slot of node A as controlled time slot, since this would cause collision at node B. Node B should advertise the controlled slot of A

2) Medium reservation by receiving nodes: More difficult is the reservation of the medium by the receiving node. Consider the two-hop network in Figure 2. Lets assume that node A and B have different controlled time slots and node C is about to choose a controlled time slot. Given the above reasoning, C would not choose the same time slot as node B. However, when it chooses the same time slot as node A, collision occurs at node B and both A and C would not be able to transmit messages to node B. Node B should thus advertise to node C that it should not take the time slot of A, since a direct connection does not exist between A and C.

In general, nodes are required to transmit a list of their neighbours' controlled slots to give newly joined nodes in the network opportunity to determine which time slots can be used without interfering with other transmissions. In Section III-C, we give suggestions on how this list can be efficiently be broadcasted between nodes.

3) Time slot selection: A newly joined node collects time slot usage information during one complete frame. After this information is collected, it can compile a set of time slots, which are not in use by its neighbour nodes or the neighbours of its neighbours (the groups might overlap). From this list *any* slot can be chosen as a controlled time slot without causing collision to any other transmission. For now, we assume that nodes (uniformly) randomly select one (non-interfering) time slot from the list.

Note, that we assume that *communication* between the choosing nodes is not yet possible, therefore, all our strategies will have a random time slot selection nature. Consequently, (local) optimization of time slot choice is not (yet) possible.

C. Implementation aspect: Bit vector of occupied time slots

In the previous section, we argued that a node should not pick a time slot that is in use by its *first* or *second* order neighbours to ensure that transmissions are not interfering. We concluded that nodes should transmit a list of the time slots, which are in use by their neighbours. This can efficiently be implemented using bit vectors of occupied slots.

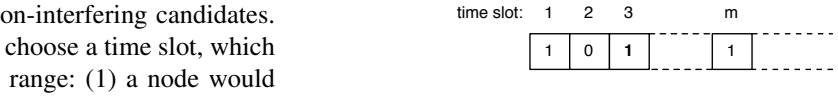


Fig. 3. Bit vector of occupied slots. A 0 indicates that the node did not detect any transmission in the respective time slot, otherwise a 1 is inserted. In addition, the node inserts a 1 at the position of its controlled slot

Consider a bit vector with a length (in bits) equal to the number of time slots per MAC frame. Each bit position in the vector represents a distinct time slot (Figure 3). A "0" is placed when no transmission is detected and otherwise a "1" is inserted. Additionally, a node adds a 1 at the bit position of its controlled time slot. All nodes internally maintain such a bit vector and transmit the result during their controlled slot.

To get a (local) two-hop view of the network, a node simply has to collect transmitted bit vectors, while it keeps its own local bit vector up to date. When a complete frame has passed, the node can pinpoint non-interfering time slots by applying *OR*-logic to all received bit vectors and the local bit vector. A "1" in the end result means that a node choosing that slot would interfere with other transmissions and a "0" in the result means that the time slot can be taken. The node scans the resulting vector for 0's and records the respective bit positions to get a complete list of non-interfering time slots.

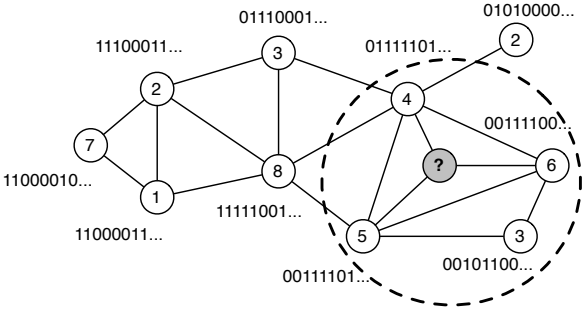


Fig. 4. Bit vector logic to determine what time slot can be used (numbers indicate controlled time slot of node). The grey node is observing the network to compile a list of non-interfering slots

We illustrate the mechanism with an example in Figure 4. Consider the grey node in the figure. It receives messages in time slots 3, 4, 5 and 6. Therefore, it constructs the following local bit vector: 00111100.... Its first order neighbours advertised the following bit vectors: 00101100... (received in time slot 3), 01111101... (slot 4), 00111101... (slot 5) and 00111100... (slot 6). The result of the OR-operation on the vectors is 01111101.... The node would find logical zero's at positions 1 and 7 in the resulting bit vector and would conclude that it can safely use time slot 1 or 7 without interfering other transmissions. In Figure 4, nodes using time slots 1 or 7 are indeed more than two hops —three and four hops, respectively– away from the grey node and the hidden terminal problem is prevented.

IV. Initiating scheduled medium usage

In the previous section, we discussed how nodes collect a two-hop view of the (local) network to choose a non-interfering time slot. However, this solves only the self-organizing aspect and not the self-starting aspects of WSNs as we sketched in Section I. In this section, we discuss how to initiate/setup the scheduling mechanism.

In traditional MAC protocols —mostly random access or contention-based protocols, like IEEE 802.11x in DCF mode– self-starting requires no additional attention. Nodes simply start listening to the wireless channel and respond to incoming requests and generate outgoing request when they see fit.

In schedule-based protocols (or any other MAC protocol that requires synchronization), initiating requires much more attention, because a random "just starting" approach —like in SMAC [12]– would result in many different timing schemes, which do not necessarily co-exist i.e. nodes between groups with different schemes might experience collisions.

One of the characteristics of wireless sensor networks is that there is only a small set of nodes that have an interest in the sensor readings. As result, most of the information is hopping towards these —so called– gateway nodes. Without any interest in the sensor readings, the network is by definition wasting energy by gathering them and keeping communication structures intact. Therefore, gateway nodes —as advocators of interest– are given a special role in our schedule-based protocol for network initialisation.

In our protocol, gateway nodes take —in contrast to non-gateway nodes– initiative in creating timing schemes. They do this by starting to take control of a time slot. Its one-hop neighbours will receive the transmission and will synchronize their clocks to the gateway. This triggers the nodes to choose a time slot themselves. Their neighbours will detect transmissions and, in this way, the synchronization event propagates through the entire network until every node is participating and controlling a time slot.

A. Operational states

To establish the above described functionality, we define four operational states for nodes not performing gateway tasks:

- **Initialisation/pre-deployment state (*I*)** — Nodes sample the wireless medium (at a slow rate to conserve energy, like in [9]) to detect transmissions of other nodes. When a neighbouring node is detected, the node synchronizes (i.e. the node knows the current slot number in the MAC frame). When a new frame is due, the node switches to the wait state *W*. However, a node remains in initialisation state when it cannot retrieve the current slot number in a message. Thus a node does not commence to the wait state when it receives a collision or an erroneous packet.
- **Wait state (*W*)** — The purpose of the wait state is to spread out the wakening times of nodes to prevent energy-consuming schedule conflicts caused by nodes that choose identical time slots (Section V). We observe that, especially at network setup, many nodes simultaneous enter the process of obtaining a time slot. This potentially leads to many collisions. Clearly, this is the case for nodes around the gateway.
  We introduce randomness in reaction time w between synchronization with the network and the actual choosing of a free time slot: $w = \{0 \dots w_{max}\}$, expressed in (integer number of) MAC frames. After the random wait time, the node continues with the discover state *D*.
- **Discover state (*D*)** — The node collects first and second order time slot usage information during one entire frame and records time slots to be occupied when the signal level is higher than a pre-defined threshold (i.e. carrier is detected).
  When all information is collected, the node chooses a time slot and advances to the active state *C*. If a node receives no useful information during the frame it is in discover state, it falls back to the initialization state *I*.
- **Continuous operation state (*C*)** — The node transmits in its time slot of choice (every MAC frame). Meanwhile, it listens to other time slots and accepts data from neighbouring nodes. The node also keeps its view on the network up-to-date. When a neighbouring node informs that there was a collision in the time slot of the node, the node immediately gives up its time slot and continues in the wait state.

The gateway nodes omit states *I*, *W* and *D*, but start directly with controlling a time slot. A gateway node can be identical to a non-gateway wireless sensor. No additional processing power or memory is required.

V. Schedule conflicts

With the algorithm presented in Section III, nodes are able to determine which time slots can be used without interfering with other nodes. However, this is not guaranteeing that schedule conflicts will not occur, as we explain below.

Schedule conflicts can occur when two or more nodes choose the same time slot to control. This can happen during network setup or when network topology changes due to —for example– mobility of nodes or variations in link quality. In this paper, we focus on the first cause. The selecting procedure of a time slot from the set of non-interfering ones has a random

nature, because the nodes without time slots are not yet able to communicate with each other. Due the random nature, nodes in the selecting process might simply select identical time slots, which causes collision. Strategies that require communication, like the node with highest ID may select a time slot first, cannot be applied, although these strategies would potentially lead to collision-free time slot allocation.

As a result of collisions, nodes are not able to communicate in a proper sense with all nodes in radio range. Both transmitting and receiving wireless sensors waste energy, since the contents of packets cannot be deciphered. Therefore, it is key issue to solve collisions preferably as quickly as possible.

Next, we introduce a collision reporting and solving mechanism. Section VI presents a statistical analysis of colliding time slots under assumption that time slots are (uniformly) randomly chosen.

A. Detecting and reporting collisions

The nodes that caused the collision cannot detect the collision by themselves; they need to be informed by their neighbouring wireless sensors, simply because they are transmitting when the event occurs.

Nodes cannot extract any information from colliding packets, so the neighbouring nodes do not have any knowledge on e.g. the ID's of the nodes that caused the collision. Therefore, nodes can only report in which time slot a collision occurred. When nodes detect a collision, they use their *own* controlled time slot to inform neighbouring nodes that they detected a collision. The neighbours on their turn check their controlled time slot against the transmitted collision information. If it matches, they conclude that they are in collision with another node, release their time slot and initiate the procedure of obtaining a non-interfering one by returning to the wait state *W* (Section IV).

To speed up the collision reporting process, nodes prevent duplicate notifications, i.e. when a collision is reported by another node and the node did detect collision in the same time slot, it will not repeat the report. Instead, it reports another collision that potentially occurred in the past frame.

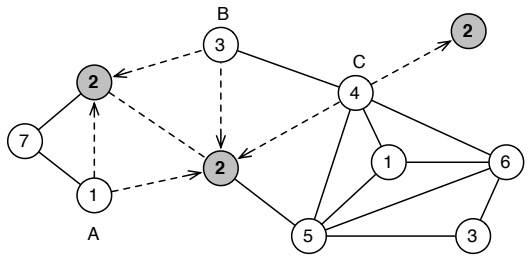


Fig. 5. The three grey nodes cause a collision in time slot 2. Nodes A, B and C detect the collision

In Figure 5, three nodes transmit during time slot 2. This causes collision at nodes A, B and C, who record the event. During time slot 3, node B is the first node that gets an opportunity to report the collision. The two colliding nodes, in direct link with node B, reinitiate the process of obtaining a non-interfering time slot. During the next time slot, node C gets an opportunity to report the collision. However, it will not do so, since it heard node B already announce the collision. At the beginning of the next frame, A reports the collision, because it did not hear another node report it.

In this collision reporting scheme, we put the responsibility of detecting interfering time slots by the receiving party and the actual solving of the interference by the transmitting party. In [4], we discuss formal verification of the schedule conflict resolving mechanisms.

VI. Statistical analysis of collisions

In this section, we analyse the statistical behaviour of nodes competing simultaneously for time slots. We determine the yield of the random time slot selection, i.e. the number of nodes that is not in conflict after choosing a slot. A low yield of unique choices makes the network slow in starting up and wastes energy in the process of self-configuration.

A. Definitions and approach

We define $k > 0$ to be the number of nodes competing *simultaneously* for time slots, u ($0 \leq u \leq k$) the number of nodes that makes a unique choice and n the number of free time slots. We assume that the number of time slots available is equal or greater than the number of competing nodes i.e. $n \geq k$ and that all k nodes make their choice from the same set of n time slots.

A first observation is that a collision always happens between two or more nodes. Thus, the probability that *exactly one* node is in collision is $P(u = k - 1) \triangleq 0$.

A second observation is that a collision is quite likely to occur (this effect is known as the *birthday paradox*). Consider the first node of the k nodes. This node can choose from n time slots without causing collision. The second can choose from $n-1$ time slots, the third from $n-2$ time slots etc. Thus the probability that all k nodes make unique time slot choices (i.e. $u = k$) is given by

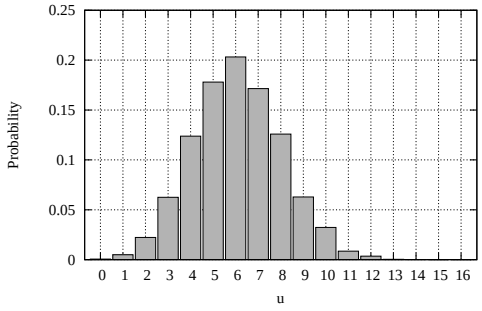
$$P(u = k) = \frac{n!}{(n-k)!n^k} \tag{1}$$

The probability of at least two nodes choosing the same time slot is

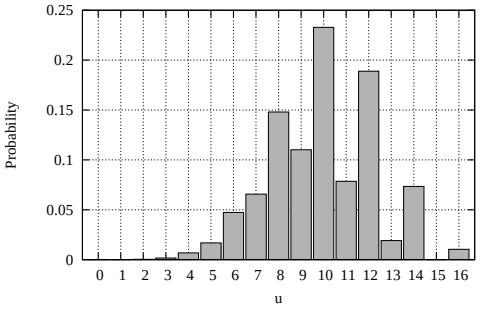
$$P(u < k) = 1 - P(u = k) = 1 - \frac{n!}{(n-k)!n^k} \tag{2}$$

In Figure 6, the above probability is plotted for different values of n. Clearly the term n^k in Equation 2 grows faster than $n!$ (assuming $n \backsim k$), resulting in an increasing probability of collision with a growing number of nodes. A higher ratio of non-interfering time slots per node results in a smaller probability of schedule conflict.

We conclude that it is quite likely that at least one collision will occur and two or more nodes will have to redo the procedures of obtaining a time slot.



(1)



(2)

Fig. 7. Probability distribution of u for (1) $n = k$, and (2) $n = 2k$. In both cases $k = 16$

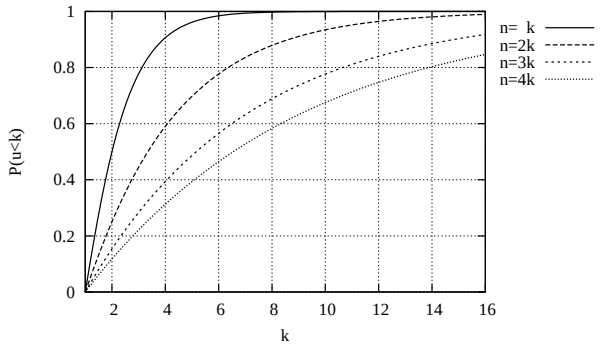


Fig. 6. Probability that at least two nodes choose identical time slots (i.e. $u < k$)

B. Average number of surviving nodes

How many nodes make on average a unique choice? We derive a method for counting the number of non-conflicting nodes based on the well known *principle of inclusion and exclusion* [6]. This principle is a generalisation of: $|A \cup B| = |A| + |B| - |A \cap B|$, with $|A|$ the number of elements in set A. Let k nodes simultaneously choose a slot from n non-interfering time slots. The number of combinations with *exactly* u surviving nodes is given by

$$E_k^n(u) = \frac{(-1)^u n!k!}{u!} \sum_{i=u}^{k} \frac{(-1)^i (n-i)^{k-i}}{(i-u)!(n-i)!(k-i)!} \tag{3}$$

Proof: The result follows directly from the principle of inclusion and exclusion. Consider a set S and conditions c_i $(1 \leq i \leq k)$ which are satisfied by some elements of S. In our case, condition c_i is satisfied when node i is not in collision after the time slot selecting process. Let S_t be the number of combinations in which t requirements are fulfilled, ignoring remaining conditions.

Let $E_k^n(u)$ be the number of elements of set S which fulfil exactly u of the conditions (including $k - u$ conditions not fulfilled), then

$$E_k^n(u) = \binom{u}{0} S_u - \binom{u+1}{1} S_{u+1} + \binom{u+2}{2} S_{u+2} - \ldots$$
$$\ldots + (-1)^{k-u} \binom{k}{k-u} S_k$$
$$= \sum_{i=u}^{k} (-1)^{i-u} \binom{i}{i-u} S_i \tag{4}$$

A thorough explanation and proof of this principle can be found in [5].

Next, we derive an expression for S_t. First, we consider t surviving nodes and place them uniquely in a time slot. We can arrange these t nodes in $k(k-1)(k-2)\ldots(k-t+1) = \frac{k!}{(k-t)!}$ different ways and select time slots for them $\binom{n}{t}$ ways. Hereby, we fulfil t conditions. The remaining $k - t$ nodes can be arranged in over the $n - t$ remaining time slots in $(n-t)^{k-t}$ different ways. Note that the later term also includes combinations in which more conditions are fulfilled; for this reason the principle of inclusion and exclusion is used.

Thus, for S_t we can write

$$S_t = \frac{k!}{(k-t)!} \binom{n}{t} (n-t)^{k-t} \tag{5}$$

Substituting Equation 5 in Equation 4 gives the result of Equation 3. ■

Equation 3 gives us exactly how many combinations there exist that result in a given number of surviving nodes. Dividing this by the total number of possible combinations i.e. n^k gives the associated probability (Figure 7):

$$P_k^n(u) = \frac{E_k^n(u)}{n^k} \tag{6}$$

The average yield of surviving nodes is given by Equation 7 (Figure 8).

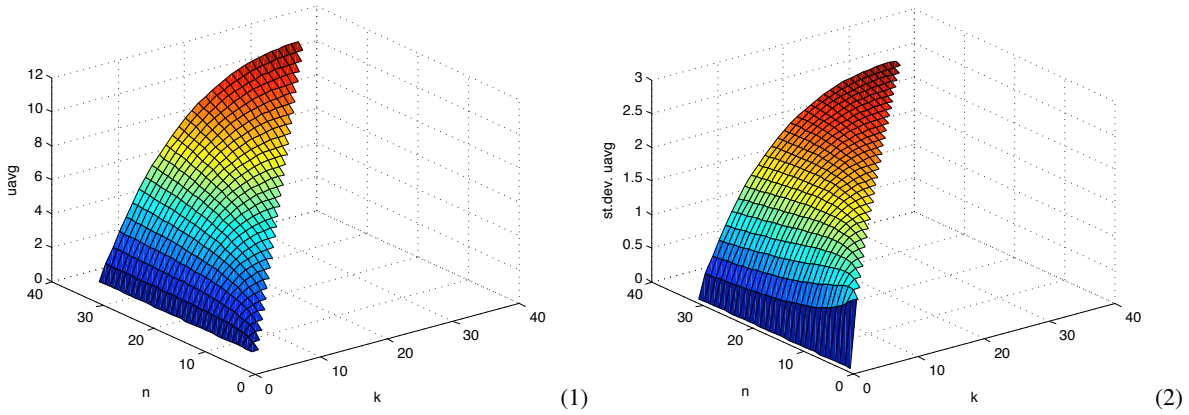
$$\bar{u}_k^n = \frac{1}{n^k} \sum_{u=0}^{k} u \frac{(-1)^u n!k!}{u!} \sum_{i=u}^{k} \frac{(-1)^i (n-i)^{(k-i)}}{(i-u)!\,(n-i)!\,(k-i)!} \tag{7}$$

Or, in a much simpler form:

$$\bar{u}_k^n = k \left(1 - \frac{1}{n}\right)^{k-1} \tag{8}$$

With standard deviation:

$$\sigma_{u_k^n} = \sqrt{\frac{1}{k} \sum_{u=0}^{k} P_k^n(u) \, (u - \bar{u}_k^n)^2} \tag{9}$$



(1) (2)

Fig. 8. (1) Average surviving nodes $\bar{u}_k^n$ and (2) its standard deviation

Let $\bar{f}_k^n$ be the average number of time slots that is not occupied after each node has selected a time slot. With simple combinatorics, we find:

$$\bar{f}_k^n = n\left(1 - \frac{1}{n}\right)^k \tag{10}$$

C. Number of rounds

Interestingly, the probability that a collision occurs in one or more time slots is quite high, as we argued in Section VI-A. This implies that it is most likely that multiple rounds are required before all k nodes use distinct time slots. By rounds we understand the following: (1) nodes make a (uniform) random choice from the set of non-interfering nodes, and (2) some nodes might pick distinct time slots, whereas others might be in collision. The latter group is notified in which time slots collisions occur and release their time slots. These two steps are repeated until all nodes have distinct time slots. We conclude that multiple rounds are necessary before each node has a distinct time slot.

On the other hand, the probability that *no* nodes survive the slot choosing process is quite small for such a large number of nodes. For example, when we look at $u = 0$ cases in Figures 7(1) and (2), we find probabilities of 0.0513% and 0.0002%, respectively, for $n = 16$ and $n = 32$. Since these probabilities exist, the time slot choosing process is not guaranteed to finish in finite time. However, the probabilities are comfortably small, that quite likely in every round some nodes survive the process and progress is made in putting every node in a distinct time slot.

The progress of the (uniformly) random time slot choosing is less evident for smaller number of nodes. For example, when two nodes can choose from two time slots, then with equal odds they both survive, or both do not survive. However, when there are slightly more non-interfering time slots than nodes, the probability of no surviving nodes at all (i.e. $u = 0$) decreases. Adding one extra time slot to the $k = 2$ case already gives a probability of $\frac{2}{3}$ that both nodes survive. Moreover, the random wait time in state *W* —the state to which colliding nodes return, Section IV– spreads the number of competing nodes over time, thereby increasing the ratio of non-interfering time slots versus nodes.

Based upon the average yield of surviving nodes (Equation 7), we determine the average number of rounds that is required to give nodes distinct time slots (Figure 9). As expected, the required number of iterations drops with an increased ratio between non-interfering time slots and nodes. Thus, to speed up the network setup process, plenty of non-interfering time slots should be available.

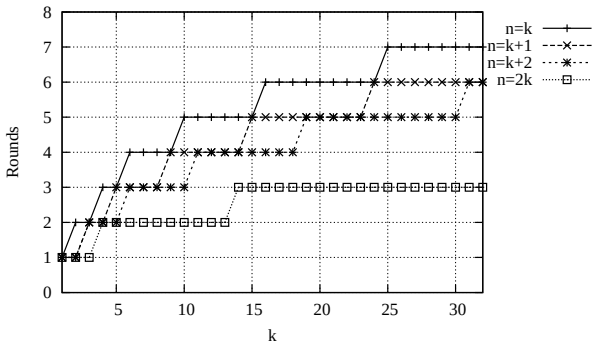


Fig. 9. Average number of rounds required before k nodes pick a non-colliding time slot

Equation 8 suggests that there is almost a linear relationship between k and $\bar{u}_k^n$; more than $\frac{1}{e}$ of the nodes would get on average distinct time slots in the $n = k$ case (i.e. worst case). We denote the average number of required rounds as $\bar{R}$. Then $\bar{R} - 1$ satisfies $k(1 - \frac{1}{e})^{\bar{R}-1} = 1$ to reduce the number of non-surviving nodes to one. The average number of rounds required can therefore be (over)estimated by

$$\bar{R} = 1 - \log_{1-\frac{1}{e}}(k) \quad \text{where } k > 0 \tag{11}$$

D. Selecting multiple time slots per round: Beneficial?

In the previous section, we concluded that plenty of non-interfering time slots should be available to speed up the network setup process, because this increases the yield of

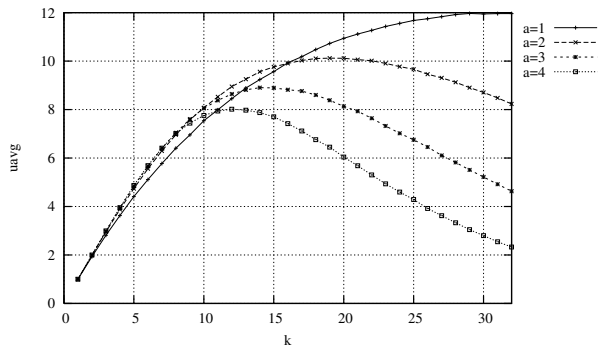


Fig. 10. Average surviving nodes when nodes select a time slots per round

surviving nodes per round. Can the yield also be increased if we let nodes select *more* time slots per round?

In Figure 7 (2), we find that abundant time slots result in 9.94 surviving nodes on average for $n = 32$ and $k = 16$, while in the case of $k = 32$, 11.96 nodes survive on average. Is a similar increase of 2.02 surviving nodes attainable if we let nodes select two time slots in the $k = 16$ case? Interestingly, when nodes select *one* time slot in the $k = 16$ case, we expect on average $\bar{f}_k^n = 19.25$ time slots to be unoccupied at all. So, there are in this example plenty of time slots that can be selected to create a higher yield of unique time slot choices. However, three effects play when nodes make their second time slot choice in a round:

1) The number of unique time slot choices can increase, when nodes select empty time slots.
2) Nodes can destroy unique time slots by choosing slots which are already occupied by exactly one node. However, a node will not collide with itself.
3) Nodes can select time slots that already collide.The yield of unique time slot choices does not change.

In Figure 10, we investigate the effect of a time slot choices per node per round.

We note that selecting multiple time slots per round does not necessarily increase the yield of surviving nodes, except for small number of nodes. In the example, for $k = 2...10$ selecting multiple time slots per round increases the yield (an increase of 0.7 surviving nodes on average), however, for larger k, the yield of selecting $a = 1$ time slot is larger and selecting multiple time slots reduces the yield considerably.

In practical wireless sensor networks, nodes have no means of estimating how many nodes will be competing for time slots and are therefore not able to decide if multiple time slot selection is beneficial for the network setup time. In addition, the number of time slots per MAC frame is likely to be tuned to the needs of the network topology, because of message latency considerations. The network wide length of the MAC frame is set to the minimum number that is required to give each node the oppurtunity to transmit once during a MAC frame (comparable to graph colouring problems). This performance consideration is the reason why in practical implementations the number of competing nodes is close to the number of non-conflicting time slots (assuming a wait time $w = 0$ in state W).

We conclude that selecting multiple time slots per round does not result in a reduction of network setup time.

VII. Related work

Sridharan et al. present a time slot assigning mechanism, in which parent nodes select time slots for their siblings [11]. The work assumes that (1) the parent/sibling relations are established before initiating the assignment procedure, and (2) nodes can communicate collision-free whilst executing the algorithm. The algorithm adapts breadth first search in assigning the time slots and parent nodes communicate with all one hop neighbours to make sure that time slots can be used without causing collisions. The algorithm is a Greedy polynomial heuristic, but according to the authors, its performance matches optimal graph colouring results in small topologies [11].

Due to its hierarchical organization, the algorithm of Sridharan [11] is less suited for dynamic topologies or iterative deployment; the network has to re-examine the time slot allocation when the topology changes. However, the algorithm is useful to obtain a coarse estimate of the required number of time slots. Our scheduling mechanism (in which we assume the number of time slots per frame fixed) can use this estimate to adapt the number of time slots to the needs in the network. This topic is left to future work.

Moscibroda et al. [7] see the assigning of time slots in a wireless network as a chicken-and-egg problem, i.e. a MAC protocol is required to establish graph colouring and vice versa. But during initialization of the network, there is no MAC or time slot allocation. The proposed algorithm elects cluster heads (i.e. a set of mutually independent nodes), which assign colour-ranges (i.e. time slots) to all its slaves. Since slaves of different clusters can be within radio coverage, they need to verify the assigned time slot(s). The slave nodes try the first time slot ϕ of the assigned range. Whenever a neighbour announces it already uses time slot ϕ, the slave node continues with verifying slot $\phi+1$ etc. When it finds a free time slot, the node claims it and periodically transmits the above mentioned announcement. The authors show that this algorithm results in correct colouring with $O(\Delta)$ time slots in $O(\Delta \log N)$ time with high probability in unit disk graphs [7] (Δ represents the maximal connectivity in the network graph and N represents the number of nodes in the network).

Interestingly, Moscibroda et al. [7] assume nodes without possibility to detect collisions, i.e. all decisions are based upon positive enforcement. Once a node has made its slot choice, it keeps using the slot. Therefore, the algorithm is only suited for static wireless networks.

VIII. Conclusion

In this paper, we have presented a medium scheduling approach in which (1) nodes are self-configuring, (2) the wireless medium is spatially reused, and (3) conflicts (i.e. collision of schedules) are resolved when detected.

In the self-organizing and self-starting schedule-based medium access method, time is organized into time slots, which are then grouped into frames. Each frame has a fixed length of a (integer) number of time slots. Each time slot has a distinct number indicating its position in the frame. Each node takes control of (at least) one time slot and a node is allowed to transmit packets during this time slot. Thus, a node uses only its controlled time slot to transfer data to neighbouring nodes. During the time slots of other nodes, a node receives packets when it is addressed. Otherwise, the node switches its transceiver to low-power mode in order to conserve energy. In our approach, we assume that the schedule in the network is fixed and is repeated on a frame basis. In principle, nodes indefinitely stick to their controlled time slot.

We required nodes to transmit at least the following information during their controlled time slots: (1) the number of the current time slot, and (2) a list of time slots used by first order neighbours to ensure medium reservation by receivers.

1) **Self-configuration** — We proposed four operational states associated with the medium access scheme: (1) the initialization state in which transmissions of other nodes act as a trigger to move to the next state, (2) a wait state, (3) a discover state in which a node assesses what time slot(s) can be used conflict-free, and (4) the continuous operation mode. In the continuous operation mode, nodes apply the proposed medium scheduling. We assumed one node in the network, which initiates the entire network. This task is well-suited for a gateway since it acts as advocator of user interest in sensor readings.
2) **Spatial reuse of medium** — In the well-known RTS/CTS handshaking mechanism, the wireless medium is reserved by both transmitter and receiver of the data message. This ensures that collision and interference to the data message are prevented. The medium can be reused outside this blocking area. We adapt a similar blocking of concurrent transmissions in time slots.
3) **Conflict resolving** — During initiation of the network by the assumed gateway, many nodes potentially enter the time slot choosing process. Since communication between these nodes is not yet possible, nodes might choose identical time slots, which obviously lead to schedule conflicts. We proposed and verified a mechanism to resolve these conflicts.
   We analyzed how many rounds are on average required to provide all nodes a non-conflicting time slot. We concluded that a higher ratio between unoccupied time slots and choosing nodes reduces the number of required rounds. To increase this ratio, the wait state W has been introduced, which spreads the choosing of nodes in time. A balance should be established between the number of time slots and the start-up time of the network.

REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Elsevier Computer Networks*, 38(4):393-422, 2002.

[2] A. Agarwal and P.R. Kumar. Improved capacity bounds for wireless networks. *Wireless Communication and Mobile Computing*, vol. 4:251–261, July 2004.

[3] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: A media access protocol for wireless LANs. *In Conf. on Communications Architectures, Protocols and Applications*, pages 212–225, Augustus 1994.

[4] A. Fehnker, L.F.W. van Hoesel and A.H. Mader. Modelling and Verification of the LMAC Protocol for Wireless Sensor Networks. *Integrated Formal Methods conference (IFM 2007)*, Oxford, UK, June 2007.

[5] R.P. Grimaldi. Discrete and Combinatorial Mathematics, Chapter 8: The Principle of Inclusion and Exclusion. Fourth edition. Addison–Wesley, 1998, ISBN 0-201-19912-2

[6] J. van Lint and R. Wilson. *A course in combinatorics*. Cambridge University Press, 2nd edition, 2001.

[7] T. Moscibroda and R. Wattenhofer. Coloring unstructured radio networks. *17th Symposium on Parallelism in Algorithms and Architectures*, July 2005.

[8] A. Ollero, M. Bernard, M. La Civita, L.F.W. van Hoesel, P.J. Marron, J. Lepley and E. de Andres. AWARE: Platform for Autonomous self-deploying and operation of Wireless sensor-actuator networks cooperating with unmanned AeRial vehiclEs. *IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR 2007)*, Rome, pages 1–6, ISBN 978-1-4244-1569-4, September 2007.

[9] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 95–107, ACM Press, November 2004.

[10] Y.E. Sagduyu and A. Ephremides. The Problem of Medium Access Control in Wireless Sensor Networks. *IEEE Wireless Communication Magazine*, 11(6)44–53, December 2004.

[11] A. Sridharan and B. Krishnamachari. Max-min fair collision-free scheduling for wireless sensor networks. *Workshop on multi-hop wireless networks*, 2004.

[12] W. Ye, J. Heidemann, and D. Estrin. Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks. *IEEE/ACM Transactions on Networking*, 12(3):493–506, 2003.