Extended Desynchronization for Multi-Hop Topologies

Clemens Mühlberger
Department of Computer Science
University of Wuerzburg
muehlberger@informatik.uni-wuerzburg.de

Reiner Kolla
Department of Computer Science
University of Wuerzburg
kolla@informatik.uni-wuerzburg.de

Abstract

Desynchronization is a biologically inspired primitive [8] for periodic but temporary exclusive access to a shared resource, like the transmission medium. In this paper, we recapitulate the single-hop desynchronization algorithm DESYNC [5, 2, 6] for Wireless Sensor Networks (WSNs) and identify its lack of handling hidden nodes in multi-hop environments. We explain in detail our decentralized and self-organizing multi-hop extension EXTENDED-DESYNC, solving the hidden node problem although it uses just locally available information. We further inspect several scenarios and topologies to prove convergence of our algorithm and to demonstrate its robustness and flexibility against dynamic changes within multi-hop Wireless Sensor Networks, like adding or removing nodes. We thus show that EXTENDED-DESYNC desynchronizes well in spite of hidden nodes. We finally outline a few useful add-ons for our algorithm and its possible fields of application.

1 Introduction

Main characteristics of *wireless sensor networks* (WSNs) are restrictions on hardware, e.g. low computational power or just little memory, as well as the ability to wireless communication. That's why several protocols for *medium access control* (MAC) already exist, mostly classified as contention-based *carrier sense multiple access* (CSMA) or as schedule-based *time division multiple access* (TDMA). We focus on TDMA protocols, which divide the radio channel into so called *time slots* with exclusive medium access for a certain node. To achieve this, most TDMA protocols like Z-MAC [4] or TRAMA [9]

- provide a base node as clock master propagating a global clock for synchronization of the time slots,
- need prior knowledge about the schedule of the time slots,

and thus

- may lose bandwidth in case of unused slots, and
- may not be able to add or remove nodes modestly.

Instead of an explicit network synchronization by a master node, Degesys et al. introduced in [5] the biologically inspired desynchronization primitive [8] as TDMA protocol for single-hop networks. That means to equidistantly spread out the time slots of all participating nodes in a self-organized and decentralized manner. Desynchronization was also implemented successfully for periodic resource scheduling like scattering of wake-up times [1]. The convergence of the DESYNC algorithm for single-hop networks was proved in [2], for multi-hop networks in [6]. The single-hop version was already used within a real-world application of an RSSI-signature-based indoor localization system in [7], however the multi-hop extension does not solve the hidden node problem and thus is not yet practical for real-world scenarios.

That's why we developed a serviceable multi-hop extension of the DESYNC algorithm: EXTENDED-DESYNC, which operates self-organized and indeed can handle hidden nodes within multi-hop networks. The main idea is utilizing locally available information about the network topology to uncover formerly hidden nodes. Furthermore, we prove its convergence and analyze the functionality of our algorithm at some complex, but realistic network scenarios.

This paper is organized as follows: In the next section we briefly introduce the *hidden node problem* and describe the DESYNC algorithm as TDMA protocol for single-hop WSNs together with a first idea of a multi-hop extension. In section 3 we first explain our multi-hop extension EXTENDED-DESYNC of the DESYNC algorithm and its handling of hidden nodes in detail, and prove afterward its convergence. An analysis of our algorithm's behavior while integrating nodes into several network scenarios follows in section 4. A conclusion and an outlook to future work in section 5 closes this paper.

2 Related Work

In this section we first outline the *hidden node problem* and then define the framework for the desynchronization algorithms used within this paper. Afterwards we recapitulate the original DESYNC algorithm and its properties for single-hop and multi-hop networks.

2.1 Hidden Node Problem

Suppose, without loss of generality, a wireless (sensor) network of three nodes L, M and R, so that node L can interact with node M, but is out of range of node R, whereas node R can also interact with node M, but is out of range of node L (see Fig. 1). If node L wants to transmit data to node M and at about the same time node R tries to send something to node M, too, both radio packets collide at node M, which then receives corrupt data. Because neither node L nor node R can overcome this packet collision by *carrier sense* (CS),

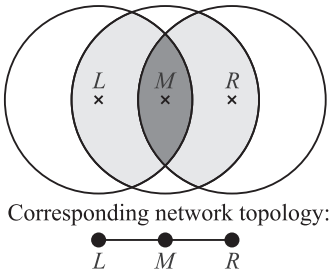

Figure 1. Hidden Node Problem

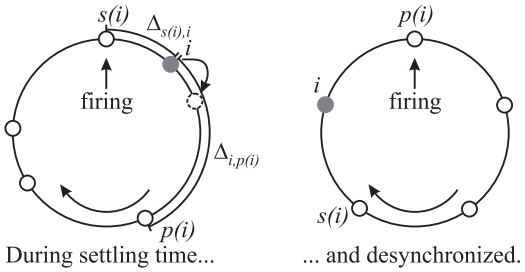

Figure 2. Snapshots of the progress of desynchronization

both nodes are hidden from each other – therefrom the name *hidden node problem*.

2.2 The DESYNC Algorithm ... in General

Next we sketch the framework generally required by the desynchronization algorithms used within this paper. These algorithms are rather simple and only depend on local information, but nonetheless form a quite complex behavior of the overall system. They do rely on neither a global clock nor an a priori known schedule. Hence, to some degree they are tolerant towards dynamical changes of the network topology and thus exemplify quite well the emergence of a distributed system by self-organization.

Again, desynchronization is a biologically inspired primitive [8] for the equidistant distribution of *oscillators*, e.g. periodically transmitting sensor nodes. So, the network is composed of a set of nodes N with symmetric communication links, where each node oscillates at an identical frequency ω within a period $T = \frac{1}{\omega}$. However, the period T must be sufficiently long, e.g. for the single-hop version T must support all $n = |N|$ possible members of the network. Each node has a unique identifier i. The phase $\phi_i \in [0.0, 1.0]$ of a node i denotes the elapsed time since its last transmission relative to T. For example $\phi_5 = 0.9$ means, that node number 5 has already finished 90% of its current period. When node i finishes its period, i.e. $\phi_i = 1.0$, it broadcasts a so called *firing packet* and immediately resets its phase to $\phi_i = 0.0$. The set of all one-hop neighbors of node i is

$$N_1(i) = \{j : i \text{ received broadcast from } j\}, \tag{1}$$

and the column vector

$$\overrightarrow{\phi} = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_i \\ \vdots \\ \phi_n \end{pmatrix}$$

denotes the system state of all n nodes. An update of the system state is only required each time a node broadcasts its firing packet.

For further understanding, two other nodes $p(i) \neq i$ and $s(i) \neq i$ are important for the following algorithms:

- node $p(i)$ broadcasts its firing packet just **before** node i and thus is called *previous phase neighbor*, whereas
- node $s(i)$ broadcasts its firing packet just **after** node i and thus is called *successive phase neighbor*.

To determine a more appropriate firing phase ϕ_i' according to an equidistant distribution, node i must first calculate the midpoint of its phase neighbors using the *delta phase*

$$\Delta_{i,j} = \phi_j - \phi_i.$$

Unfortunately, the information about the phase of the previous phase neighbor $\phi_{p(i)}$ might be stale, since $p(i)$ also tries to determine a more appropriate firing phase $\phi'_{p(i)}$ and thus could have changed its phase already – depending on the phase neighbors of $p(i)$. According to [2], this stale information won't affect the functionality of the DESYNC algorithm, but slows down a bit its convergence rate. After the broadcast of i's successive phase neighbor $s(i)$, node i can calculate the midpoint of its phase neighbors as

$$\begin{aligned} \text{mid}(\phi_{s(i)}, \phi_{p(i)}) &= \phi_i + \frac{\Delta_{i,p(i)} - \Delta_{s(i),i}}{2} \\ &= \frac{\phi_{p(i)} + \phi_{s(i)}}{2}. \end{aligned} \tag{2}$$

Finally, the new firing phase ϕ_i' of node i can be estimated by itself as

$$\phi_i' = (1-\alpha) \cdot \phi_i + \alpha \cdot \text{mid}(\phi_{s(i)}, \phi_{p(i)}). \tag{3}$$

The *jump size parameter* $\alpha \in (0.0, 1.0)$ regulates, how fast a node moves toward the assumed midpoint of its phase neighbors, e.g. $\alpha = 0.0$ would mean no movement at all, whereas $\alpha = 1.0$ forces straight movement onto the midpoint under unstable emergence of new configurations (cf. [1]). Adequate values for the jump size parameter are $\alpha = 0.95$ and $\alpha = 0.9$, according to [5], and [6] respectively.

Figure 2 exemplifies the progress of the desynchronization: on the left, node i is ready to change its phase, because its successive phase neighbor $s(i)$ is broadcasting its firing packet. That means, the calculation of the next firing phase ϕ_i' of node i can be made soonest whenever its subsequent one-hop neighbor[1] broadcasts its firing packet. Whereas on the right, the system of five nodes has reached a stable state, because all nodes are equidistant from their particular phase neighbors. Convergence to the stable state of *desynchrony* (cf. [5]) is achieved, if each node has the same distance to its phase neighbors and thus the transmission times do not change anymore – unless the system changes.

[1] Especially when using EXTENDED-DESYNC for multi-hop networks, the subsequent one-hop neighbor of a node i is necessarily not identical to the successive phase neighbor of node i.

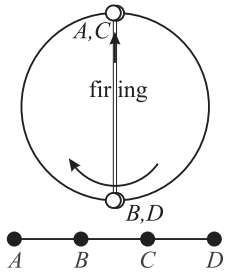

Figure 3. Desynchronization of P_4 topology by DESYNC

2.3 ... within Single-hop Networks

The first version of the DESYNC algorithm described in [5, 2] just considers single-hop networks. Here, the node identifiers – without loss of generality – are numbered consecutively so that $1 \leq i \leq n$ holds. Furthermore, the phase neighbors of a node i simply are

- node $p(i) = i - 1$ as previous, and
- node $s(i) = i + 1$ as successive phase neighbor.

The midpoint computation works in the same way as specified in section 2.2. The stable state of desynchrony is reached, if

$$\Delta_{i,p(i)} = \frac{1}{n}$$

is true for any i.

This single-hop approach converges well, as proved in [5]: the problem of desynchronization was adapted to the *graph coloring problem* and by means of it, convergence to a stable state was shown for arbitrary system initial states, as well as stability, i.e. if the system reaches a stable state, this state will be changed not until system topology is changed.

2.4 ... within Multi-hop Networks

Since single-hop networks cover just a small field of real-world applications, the developers of the single-hop DESYNC algorithm tried it on multi-hop networks [6]. The main difference is the identification of the phase neighbors of a node i:

- $p(i) = \underset{j \in N_1(i)}{\operatorname{argmin}} \Delta_{i,j}$ as previous phase neighbor, and
- $s(i) = \underset{j \in N_1(i)}{\operatorname{argmax}} \Delta_{i,j}$ as successive phase neighbor.

The new phase ϕ_i' of node i is then calculated according to equation 3, where $\alpha \in (0.0, 1.0)$ again is the jump size parameter (cf. 2.2). If each node i satisfies the equation $\Delta_{i,p(i)} = \Delta_{s(i),i}$, the stable state of desynchronization is reached here.

With it, convergence to a stable state also was proved, but not all stable states are valid and implementable, because total message loss can emerge within some scenarios due to hidden nodes. Especially in path graph[2] topologies P_n this multi-hop version of the DESYNC algorithm generates improper and impractical phases for all nodes! For example Fig. 3 shows a P_4 topology and the resulting desynchrony. Unfortunately, $\phi_A = \phi_C$ and $\phi_B = \phi_D$ and thus all messages will be lost due to collisions.

To overcome these defects, one proposal made was not to look at the communication graph but at the constraint graph of the network – however that's exactly the challenge: how to gain the constraint graph from just local view. We met the challenge and thus present our multi-hop approach for desynchronization including an answer for the hidden node problem in the following section.

[2] Two nodes have degree 1, and $n-2$ nodes have degree 2.

3 Multi-Hop Extension EXTENDED-DESYNC

According to [6], the challenge is how to gain the constraint graph. Our main idea is to use additional and locally available information about the current network topology for an equidistant spread out of the sensor nodes while solving the hidden node problem. Our approach should converge, of course, but be robust against topological changes, also. That's why we present our algorithm in the next section and demonstrate its convergence and fault-tolerance afterward.

3.1 The EXTENDED-DESYNC Algorithm

First of all, each node requires a unique identifier and symmetrical links between nodes. In addition to the definition of the set of one-hop neighbors $N_1(i)$ from equation 1, we define the set of two-hop neighbors $N_2(i)$ of node i as

$$N_2(i) = \{ j \neq i : \exists k \in N_1(i) : j \in N_1(k) \} \cup N_1(i).$$

Because our algorithm shall work fine in multi-hop networks, the period T must be now long enough to support the maximum set of possible two-hop neighbors $\max_{i \in N}(|N_2(i)|)$. With it, we can construct the desired constraint graph and define the phase neighbors of node i as

- $p(i) = \underset{k \in N_2(i)}{\operatorname{argmin}} \phi_k$ as previous phase neighbor, and
- $s(i) = \underset{k \in N_2(i)}{\operatorname{argmax}} \phi_k$ as successive phase neighbor.

Now node i simply needs to know all its two-hop neighbors and their relative phases. That knowledge can be gained, if every one-hop neighbor $j \in N_1(i)$ broadcasts within its firing packet the current list of its one-hop neighbors $N_1(j)$ together with their phases corresponding to the point of view of sender j. According to listing 1, every time node i receives such a broadcast from one of its one-hop neighbor $j \in N_1(i)$ (line 1), it first checks, whether node j is yet registered as one-hop neighbor (line 2). If not, it removes node j from the list of two-hop neighbors $N_2(i)$ (line 3) to be on the safe side, and then adds j to its one-hop neighbors $N_1(i)$ (line 4). Anyway, node i refreshes its information about the phase of node j (line 6): because node j just sent its firing packet at phase $\phi_j = 0$, node i calculates the delta phase as $\Delta_{j,i} = \phi_i$. This delta phase is now subtracted from each phase of any node $k \in N_1(j)$ as well as from node j's phase to adapt all phases relative to i's point of view (cf. line 12). Of course, if the resulting relative phase is less than 0, it has to be normalized again by adding 1. Afterwards, node i matches the received list $N_1(j)$ with its own, $N_1(i)$, and registers yet unheard nodes as two-hop neighbors: each element k of the received list $N_1(j)$ (line 7) is added to the two-hop neighbors $N_2(i)$ (line 10), if node k is not already a member

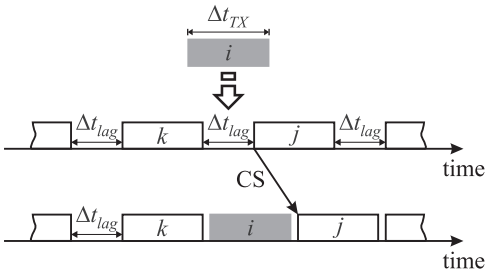

Figure 4. Node *i* joins the network, filling a time lag

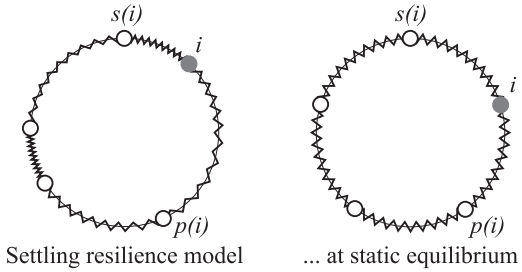

Figure 5. The example from Fig. 2 as resilience model

of the list of one-hop neighbors $N_1(i)$ (line 9). In any case, node *i* also adapts the phase of node *k* (line 12), as already described above.

```
1  receiveFiringPacketFrom(j);
2  if (j ∉ N1(i)) {
3      removeNodeFrom(j, N2(i));
4      addNodeTo(j, N1(i));
5  }
6  adaptPhaseOf(j);
7  for (k ∈ N1(j)) {
8      if (k ≠ i) {
9          if (k ∉ N1(i)) {
10             addNodeTo(k, N2(i));
11         }
12         adaptPhaseOf(k);
13     }
14 }
```

Listing 1. List operations of node *i* after receiving a firing packet from its one-hop neighbor *j*

Please note, that if a phase neighbor is a *true*[3] two-hop neighbor of node *i*, its current phase can not be measured by node *i* itself anymore. But even so, EXTENDED-DESYNC is able to handle such stale information, which will be updated at the next period. That's why node *i* can then compute its more appropriate transmission time ϕ_i' according to equation 3 – just the convergence rate may slow down a bit. Only now, node *i* can avoid collisions with one-hop neighbors, but also keeps track of its two-hop neighbors, which indeed are all potential hidden nodes. Next, we prove the convergence of our EXTENDED-DESYNC algorithm.

3.2 Convergence of EXTENDED-DESYNC

In contrast to the proof of convergence of the DESYNC algorithm, we do not try to convert the problem of desynchronization into the problem of graph coloring, but we are geared to the elastic resilience model. There are some requirements to be fulfilled – a few of them were already mentioned in 2.2:

- First of all, the settling time of the desynchronization algorithm shall be minimal, i.e. produce as few as possible collisions. Therefore, CS is mandatory every time before a node broadcasts a packet. At least, this reduces collisions with one-hop neighbors.
- The period T must be large enough. For our algorithm the period T has to hold $T > \max_{i \in N}\left(|N_2(i)|\right) \cdot \Delta t_{TX} \cdot \tau$, i.e. the maximum cardinality of any node's two-hop neighborhood times the duration of a single packet transmission Δt_{TX} times a safety factor[4] $\tau = 1.5$.
- A joining node *i* selects its first time of firing in such a manner to fit in-between the time lag Δt_{lag} of one of its one-hop neighbors $j \in N_1(i)$ and the node $k \in N_2(i)$, which is subsequent to *j*, but even must not be known by *j* (cf. Fig. 4). This way, the one-hop neighbor *j* can detect the additional overlapping packet from node *i* using CS. Node *j* thus defers its own firing backwards, because the time lag between its firing and the firing of its successive neighbor $s(j) \in N_2(j)$ – which may be a two-hop neighbor and thus can not detect an overlapping packet of *j* by using CS – is large enough per definition of T. This procedure works fine, since the joining node *i* listens for at least one period to become familiar with its neighborhood.

With it, we can define an elastic resilience model which complies to our EXTENDED-DESYNC algorithm: All observed nodes are arranged on a circle of period T so that each node *i* is connected with its phase neighbors $p(i)$ and $s(i)$ via springs, cf. the left side of Fig. 5. Keep in mind that the phase neighbors can be one hop as well as two hops away from node *i*. However, phase changes of two-hop neighbors emerge after two periods, of one-hop neighbors already after one period. That's why two-hop phase neighbors can be treated as one-hop neighbors with a just delayed exchange of information. Hence, it is legitimate not to distinguish between one-hop and two-hop neighbors for our resilience model, but to place them all along a single circle. Both types, nodes as well as springs, move frictionless solely along the circle and there is no external force at all. All springs within this elastic resilience system are identical in construction, especially they all have an equal spring constant k and an identical, unstretched length l_0. Besides, the springs are connected in series, trying to decrease their potential energy by returning to their equilibrium position. This static equilibrium looks like the right side of Fig. 5 and exactly complies with the stable state of desynchrony. After settling time, the

[3] Remember that $N_1(i) \subseteq N_2(i)$. So, node *j* is a true two-hop neighbor of node *i*, if $j \in N_2(i)$, but $j \notin N_1(i)$.

[4] The safety factor τ assures, that the time lag Δt_{lag} between two successive firing packets is always greater than half the duration of a single packet transmission $\Delta t_{lag} > 0.5 \cdot \Delta t_{TX}$ – as long as there is space for at least one more node to join.

stable state has the lowest potential energy U of all springs. It is sufficient to show that our EXTENDED-DESYNC algorithm also results in such a stable state, because every node holds the highest possible delta phase to its phase neighbors.

Assuming m nodes – and so m springs – with $1 \leq i \leq m$ along a circle as described above, the potential energy U_i stored in spring i is

$$U_i = \frac{1}{2} k_i x_i^2,$$

where k_i denotes the spring constant and x_i their displacement. With it, the potential energy U of all m springs sums up to

$$U = \frac{1}{2} \sum_{i=1}^{m} k_i x_i^2.$$

Because all springs have an arbitrary but equal spring constant k, i.e. $k_i = k$ for all i, we choose $k = 2$ and obtain

$$U = \sum_{i=1}^{m} x_i^2. \tag{4}$$

A remarkable observation for the resilience system is, if the whole system is translated (counter)clockwise along the circle without changing the relative distances of all nodes, no changes in energy will result. Paying attention to the crossing of the endpoints of the phase interval, this observation also holds for the network model. Now let us consider the displacement x_i as the delta phase $\Delta_{s(i),i} = \phi_i - \phi_{s(i)}$ between node i and its successive phase neighbor $s(i)$. Please keep in mind, that you usually have to add 1 to the resulting phase to stay consistent, if the delta phase crosses an endpoint of the phase interval. We cope with that difficulty due to our observation above by translating the whole system along the circle in such a manner that the phase of the examined node i will be $\phi_i = 0$, if i deals with its previous phase neighbor $p(i)$, and $\phi_i = 1$, if it deals with its successive phase neighbor $s(i)$. Thus equation 4 can be modified to

$$\begin{aligned} U &= \sum_{i=1}^{m} (\phi_i - \phi_{s(i)})^2 \\ &= \sum_{\substack{j=1 \\ j \neq i}}^{m} (\phi_j - \phi_{s(j)})^2 \\ &\quad + (\phi_i - \phi_{s(i)})^2 + (\phi_{p(i)} - \phi_i)^2. \end{aligned}$$

A sufficient condition for the stable state of our system is, that the difference in energy, when a single node i moves while all other nodes $j \neq i$ remain unchanged, can be obtained by the partial derivative of the total potential energy U with respect to ϕ_i:

$$\frac{\partial U}{\partial \phi_i} = 2(\phi_i - \phi_{s(i)}) - 2(\phi_{p(i)} - \phi_i). \tag{5}$$

After some settling time, the elastic resilience system entered a stable state, as soon as there is no more change in energy. Thus, it is a necessary condition to have a minimum difference of energy, that

$$\frac{\partial U}{\partial \phi_i} = 0. \tag{6}$$

This complies with the stable state of our algorithm, where node i doesn't want to change its phase any more. That is, when the phase of node i equals the midpoint of its phase neighbors, because from equation 5 and 6 follows

$$\phi_i = \frac{\phi_{p(i)} + \phi_{s(i)}}{2},$$

which indeed equals equation 2 – a central term within all desynchronization algorithms mentioned within this paper. So far, we showed that the elastic resilience system complies well with our EXTENDED-DESYNC algorithm and we characterized the stable state of that system.

Such a stable state exists, it will be attained, if the changes of the total potential energy of subsequent states is strictly monotonic decreasing. Thus, we show

$$\Delta U = U' - U < 0, \tag{7}$$

where U' denotes the potential energy of all m springs after just node i changed its phase from ϕ_i to ϕ_i', while again all other nodes $j \neq i$ remain unchanged. Due to our translation of the whole system along the circle, we insert $\phi_i = 0.0$ into equation 3, and can use the following substitution

$$\phi_i' = \alpha \cdot \frac{\phi_{p(i)} + \phi_{s(i)}}{2}$$

to prove equation 7 as

$$\begin{aligned} \Delta U &= \left[(\phi_i' - \phi_{s(i)})^2 + (\phi_{p(i)} - \phi_i')^2 \right] \\ &\quad - \left[(\phi_i - \phi_{s(i)})^2 + (\phi_{p(i)} - \phi_i)^2 \right] \\ &= \left(\alpha \cdot \tfrac{\phi_{p(i)} + \phi_{s(i)}}{2} - \phi_{s(i)} \right)^2 + \left(\phi_{p(i)} - \alpha \cdot \tfrac{\phi_{p(i)} + \phi_{s(i)}}{2} \right)^2 \\ &\quad - \left(\phi_i - \phi_{s(i)} \right)^2 - \left(\phi_{p(i)} - \phi_i \right)^2 \end{aligned}$$

Here, we get two possible solutions

$$\Delta U_1 = -\frac{1}{2} \Big(2\phi_i - \big(1 - (\alpha - 1)\big) \big(\phi_{p(i)} + \phi_{s(i)} \big) \Big)^2 \text{ and}$$

$$\Delta U_2 = -\frac{1}{2} \Big(2\phi_i - \big(1 + (\alpha - 1)\big) \big(\phi_{p(i)} + \phi_{s(i)} \big) \Big)^2.$$

But for both ΔU_x with $x \in \{1, 2\}$ holds

$$\Delta U_x < 0,$$

and thus

$$\Delta U = U' - U < 0.$$

Consequently, the convergence of our EXTENDED-DESYNC algorithm is proved. But there remain some deficiencies to be remedied, so we'll have a closer look at them in the next section.

3.3 Further Observations

Due to our experiences with real-world implementations we observed strange occurrences, which aren't totally satisfied by the requirements of our EXTENDED-DESYNC algorithm from section 3.2, but which need additional support.

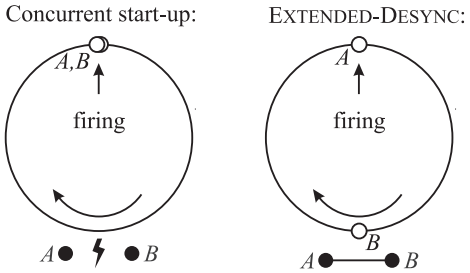

Figure 6. The path graph topology P_2

That's why we have a closer look at the impact of concurrently starting nodes in 3.3.1, the possibility of collision detection by the polluter in section 3.3.2, and unreliable links due to reduced transmission power in 3.3.3.

3.3.1 Concurrent Start-up

If a node doesn't receive any other node since start-up, maybe a neighboring node broadcasts concurrently and thus may cause collisions. For example, consider the very simple path graph topology P_2 of two nodes *A* and *B*, where node *A* is within the transmission range of node *B* – and vice versa. Both nodes fulfill the requirements of section 2.2 and 3.2, which briefly are

- unique identifiers,
- symmetric communication links,
- a period T of sufficient length,
- both nodes use CS before transmission, and
- the first time of broadcasting the firing packet is selected by each node according to section 3.2.

However, if both nodes start at the same time, they listen for one period T, and – for lack of further neighbors – broadcast their (first) firing packet concurrently, which causes collisions at each node. Because no accurate firings of further neighbors were received, each node assumes to be alone and thus listens again for one period T. Once more, no other neighboring nodes were detected and thus again nodes *A* and *B* broadcast their firing simultaneously, causing a collision. That's why, each node still assumes to be the only node within the network and listens once again for one period T. In the absence of neighbors, node *A* and node *B* again fire simultaneously, and so on. As long as there is no clock drift or topology change, both nodes still broadcast concurrently and never detect each other (cf. left side of Fig. 6).

For this reason, some sort of randomness is mandatory. Here, we install a (pseudo) random number generator, e.g. that one described in [3], using the node's unique identifier as seed to produce random numbers. Such a random number just has to be transformed into a period of time $T_{rand} \in [0.0, T]$. The complete back-off strategy at a node's start-up then works as follows:

> The node starts up, and listens for one period T.
>
> 1. If the node receives at least one firing packet, it matches the received neighbor list(s) according to listing 1 and chooses the phase of its first firing corresponding to section 3.2. The further progress follows the EXTENDED-DESYNC algorithm as described in section 3.1.
> 2. Else, the node immediately broadcasts its first firing packet and now listens for $T + T_{rand}$. This random period of time T_{rand} is generated by a (pseudo) random number generator using the node's unique identifier as random seed.
>   (a) If the node receives at least one firing packet within $T + T_{rand}$, it proceeds according to 1.
>   (b) Else, the node again broadcasts a firing packet immediately, generates the next random number $T'_{rand} \in [0.0, T]$ and listens for $T + T'_{rand}$ once more.
>
>   The further progress follows 2a or 2b, depending on whether firing packets of other nodes were received within $T + T'_{rand}$, or not.

Using the node's unique identifier as random seed assures for concurrently firing nodes, that one node listens for a shorter time and thus fires earlier than all other nodes, whereas the other nodes are still listening. With an increasing amount of periods, the probability of nodes still firing at the same time thus is exponentially decaying. Together with this back-off strategy, EXTENDED-DESYNC will converge and result in a stable state, like that one on the right side of Fig. 6.

3.3.2 Collision Detection

Next, due to symmetric links a node i can conclude that it causes a collision, if i receives its neighboring node j, but over several periods $i \notin N_1(j)$. Thus, remember the network of the nodes *L*, *M* and *R* from section 2.1. Now imagine, nodes *L* and *R* still can't receive each other, but are asleep. Just node *M* could communicate with both nodes, is yet desynchronized and broadcasts its firing packets. Now, both nodes *L* and *R* start up simultaneously, listen for one period, and receive the firing packet of node *M*. According to 3.2, the nodes *L* and *R* choose their first time of firing in-between the time lag of node *M* and its successive phase neighbor $s(M) = M$. Unfortunately, they both will choose the same phase, i.e. $\phi_L = \phi_R$, which causes a collision of their firings at node *M*. Therefore, the firing packet of node *M* contains neither nodes. But by reason of symmetric links, both nodes *L* and *R* can conclude, that they may cause a collision and thus must change their phases. On account of the definition of T in section 3.2 and while T supports further nodes, such a phase change must be possible.

Indeed, if both polluters *L* and *R* jump to equal phases again, there are still collisions. Thus, we recommend the usage of a (pseudo) random number generator once more. That is, each polluter decides randomly to *jump*, i.e. to change its phase, or not. An optimal probability would be the reciprocal value of the number of polluters. But because the number

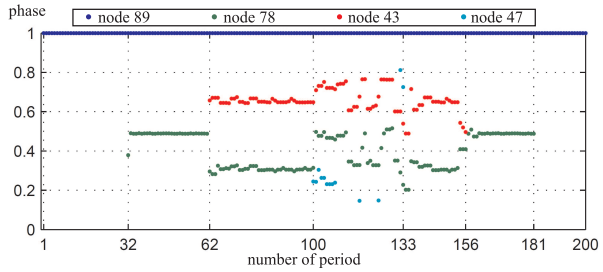

Figure 7. Logged data from our real-world indoor testbed with unreliable link

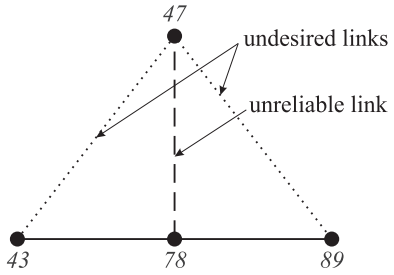

Figure 8. Topology of our real-world indoor testbed with unreliable link

of current polluters is locally unknown and in the majority of cases just two nodes are causing a collision, a probability of 50% provides a good trade-off between reliability and latency for real-world deployments.

3.3.3 Unreliable Links

From one of our real-world testbeds we received just strange data at first, as shown in Fig. 7: All phases are normalized to that one of node 89, which started first at period 1. The other nodes 78 and 43 are joining at period 32, and 62 respectively, but the system desynchronizes pretty well and fast. But as soon as node 47 starts up at period 100, whole system was not able to converge, because node 47 always changed its phase or was not even received by node 89. Exactly at period 133, when the faulty node 47 leaves the network, the remaining nodes try to desynchronize again. Remarkably, no collisions were detected by node 89 during the experimental run. A closer look at the network topology disclosed the answer to that problem: because we tried to install the multi-hop topology from Fig. 8 indoors, we had to reduce the transmission power of node 47 to release the connections between node 47 and 43 as well as between node 47 and 89. But thereby, the link between node 47 and 78 became temporarily asymmetric and thus unreliable. However, all desynchronization algorithms mentioned within this paper require symmetrical and reliable links. With the following strategy, the EXTENDED-DESYNC still can not handle asymmetrical links in general, but it now can deal with such unreliable links.

So far, if a neighbor was not received any more within the next period, it was removed immediately from the corresponding list. Far better results can be achieved, if a node, which is not received for a moment, is just removed from the corresponding list after a certain number of consecutive periods without reception. Our experiments showed that a holding time of about three periods smooths such unreliable links and produces a good trade-off between latency and fault-tolerance. But a holding time for nodes, which were not received for a while, doesn't solve the problem of asymmetrical links in general, because the holding time is fixed and definitely not sufficient for every scenario.

Indeed, the strategies for handling unreliable links and for detection of collisions are somewhat contrary, that's why node 47 changed its phase that frequently, since it followed the strategy to detect collisions. Reading the received signal strength (RSS) value of the radio chip could be one possibility to distinguish collisions from unreliable links, because for collisions a higher signal strength can be supposed. Another indicator for colliding packets could be a failed CRC check, but this failure may originate from other cases and additionally, the preamble has to be received completely at first.

4 Node Integration Behavior

The last section documented the need for additional back-off strategies for our EXTENDED-DESYNC algorithm when implementing real-world applications – despite of proved convergence. Obviously, the case when a node leaves the network, because it was removed or it failed, is easy to handle and just creates a few unused time slots. That's why we analyze some integration scenarios exemplary in this section to inspect the procedure of our algorithm. First, we'll look at a certain star topology in section 4.1, where the central node initially operates exclusive, but later on further nodes will join – even simultaneously. Next, in section 4.2 we'll observe another complex scenario of two disjoint and yet independently desynchronized networks, which become interlinked by a joining gateway node.

4.1 Integration into Star Topology

Suppose a simple star topology as shown in Fig. 9, consisting of a central node *S* and a few boundary nodes around. All boundary nodes are two-hop neighbors but don't know each other initially.

First, let us assume that all boundary nodes are yet running and desynchronized, while *S* is asleep. Now, the central node *S* starts up, listens for one period and then broadcasts its first firing packet, containing all boundary nodes received so far. Simultaneously broadcasting boundary nodes are causing collisions at the central node and thus aren't within the list of one-hop neighbors of *S*. That's why these polluters then use the collision detection algorithm from section 3.3.2 to cope with that situation. The more boundary nodes are communicated by *S*, the faster converges this setting.

Next, we expect that just the central node *S* is running and follows the strategy for concurrent start-ups from section 3.3.1, while all boundary nodes are asleep. If now some boundary nodes start up concurrently, they will cause a collision at the central node, but will receive the central node. Since boundary nodes are not within the list of one-hop neighbors of *S*, each such polluter uses again the collision detection algorithm from section 3.3.2. If the boundary nodes would start up by and by, the EXTENDED-DESYNC algorithm would not need further back-off strategies.

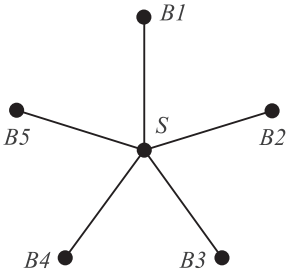

Figure 9. Star topology S_5 with central node S and five boundary nodes

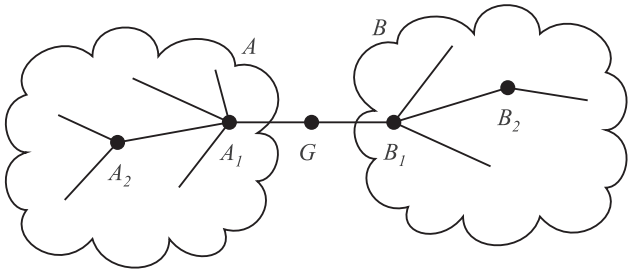

Figure 10. The disjoint networks A and B are connected by gateway G

4.2 Integration as Gateway

Now we'll analyze a more complex network topology consisting of two disjunct partial networks A and B plus a gateway node G as exemplified in Fig. 10. Both (partial) networks A and B are already independently desynchronized, while node G is asleep. When the gateway G starts up, it listens for one period.

If G receives both boundary nodes A_1 and B_1, it selects a phase according to the received neighbor lists $N_1(A_1)$, and $N_1(B_1)$ respectively. Thus, both networks are unified and no special back-off strategy was required.

Now let us assume that the boundary nodes A_1 and B_1 broadcast their firings simultaneously, which occurs with much lower probability then the first case. This time, the gateway node receives just corrupt data and thus follows the start-up strategy from 3.3.1. Granted that without loss of generality e.g. node $A_2 \in N_1(A_1)$ and G are broadcasting their firing packets at the same time over a longer period, which is even more unrealistic, but causes collisions at A_1. Because the gateway node can't detect collisions, due to a still empty list of one-hop neighbors, only node A_2 can detect its further missing in the neighbor list $N_1(A_1)$ of node A_1. That means, node A_2 detects this collision and follows the strategy from section 3.3.2.

So far, nodes A_1 and B_1 didn't change their phases and still broadcast their firings simultaneously. Because of the start-up strategy, nodes A_1 and B_1 eventually receive the gateway node G, but they aren't on the gateway's neighbor list $N_1(G)$. That's why these boundary nodes also start the collision detection algorithm from 3.3.2 to change their phases. Eventually, this makes G to receive A_1 and B_1, which then will meet each other as two-hop neighbors as soon as the gateway broadcasts its neighbor list $N_1(G)$. For this reason, we demonstrated the robustness and fault-tolerance of our EXTENDED-DESYNC algorithm together with some back-off strategies.

5 Conclusion and Future Work

In this paper we first introduced the biologically inspired primitive of desynchronization. Next, we gave a survey at the DESYNC algorithm for single-hop networks and its misuse in multi-hop topologies due to hidden nodes. In addition, we introduced our multi-hop version of a desynchronization algorithm: EXTENDED-DESYNC. Our approach considers two-hop neighbors and thus prevents hidden nodes. We therefore proved the algorithm's convergence and identified the need of additional back-off strategies for unreliable links, collision detection and concurrently operating nodes. Finally, an analysis of several real-world scenarios showed the robustness and fault-tolerance of our EXTENDED-DESYNC algorithm.

For our approach we added local information into the firing packets to gain data about the network topology. So, for our future work we plan to fully utilize the information of the firing packets for other add-ons, like time synchronization (even if this add-on is contrary to the primitive of desynchronization) or routing. We also need to analyze some of the parameter settings, e.g. a dynamically adapted jump size parameter α, a non-universal period T, or support of asymmetrical links. We also want to research possible savings in energy as a result of exclusive periods for data and periods for desynchronization. The size and structure of the firing packet, as well as possible compression techniques also remain for future investigations.

6 References

[1] Alessandro Giusti, Amy L. Murphy, and Gian Pietro Picco. Decentralized Scattering of Wake-up Times in Wireless Sensor Networks. In Koen Langendoen and Thiemo Voigt, editors, *EWSN*, volume 4373 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 2007.

[2] Ankit Patel, Julius Degesys, and Radhika Nagpal. Desynchronization: The Theory of Self-Organizing Algorithms for Round-Robin Scheduling. In *SASO*, pages 87–96, Cambridge, MA, USA, 2007. IEEE Computer Society.

[3] Deva Seetharam and Sokwoo Rhee. An Efficient Pseudo Random Number Generator for Low-Power Sensor Networks. In *LCN*, pages 560–562. IEEE Computer Society, 2004.

[4] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min. Z-MAC: a Hybrid MAC for Wireless Sensor Networks. In Jason Redi, Hari Balakrishnan, and Feng Zhao, editors, *SenSys*, pages 90–101. ACM, 2005.

[5] Julius Degesys, Ian Rose, Ankit Patel, and Radhika Nagpal. DESYNC: Self-Organizing Desynchronization and TDMA on Wireless Sensor Networks. In Tarek F. Abdelzaher, Leonidas J. Guibas, and Matt Welsh, editors, *IPSN*, pages 11–20, Cambridge, MA, USA, 2007. ACM.

[6] Julius Degesys and Radhika Nagpal. Towards Desynchronization of Multi-hop Topologies. In *SASO*, pages 129–138, Venice, Italy, 2008. IEEE Computer Society.

[7] T.-H. Lin, I.-H. Ng, S.-Y. Lau, K.-M. Chen, and P. Huang. A Microscopic Examination of an RSSI-Signature-Based Indoor Localization System. In *5th Workshop on Embedded Networked Sensors (HotEmNets 2008)*, Charlottesville, Virginia, USA, June 2008. ACM Press.

[8] Renato E. Mirollo and Steven H. Strogatz. Synchronization of Pulse-Coupled Biological Oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[9] Venkatesh Rajendran, Katia Obraczka, and J. J. Garcia-Luna Aceves. Energy-efficient, Collision-Free Medium Access Control for Wireless Sensor Networks. In Ian F. Akyildiz, Deborah Estrin, David E. Culler, and Mani B. Srivastava, editors, *SenSys*, pages 181–192. ACM, 2003.