Pulse Coupled Synchonization and Scheduling

Reinhard Gentz, Lorenzo Ferrari and Anna Scaglione

Abstract—We propose an architecture that provides decentralized synchronization and scheduling, using a network synchronization protocol that is inspired by the dynamics of pulse coupled oscillators PCO. This provides time division multiple access, for a self-organized clustered network in which cluster-heads can obtain data from all the client nodes in the cluster with bounded delays.

Index Terms—Pulse Coupled Oscillators, distributed time scheduling, clustered networks.

1 Introduction

Data acquisition is ubiquitous in sensor networks (SN), and it consists of the regular sampling of analog measurements as well as digital variables, recorded by field devices and programmable logic controllers.

As SN have grown in size and complexity, it has become prevalent to use packet switched network to connect field devices and sensors, multiplexing the same communication media for a number of control and monitoring functions. Some applications in SN are particular time sensitive: they require accurate time-stamping of information over multiple sensors, ensuring a bounded data delivery latency. In principle, what the system should emulate is the model of an ideal sensor array where the field devices sample the physical system synchronously. In practice, each individual sensor will have a local sampling clock, which has to be synchronized. One application for that could is state estimation, e.g. in the power grid.

Typically the need for network synchronization and rapid access to the communication medium are addressed through out of band control channels or message exchanges, with timing coming prevalently from a Global Positioning System (GPS) receiver. In band network synchronization protocols, such as, for instance the Precision Time Protocol (PTP) [1], use instead the same communication medium to establish a common clock. However GPS-Signals can bee spoofed [2] and PTP is not immune to cyber attacks.

Today, shared media (wireless) solutions like WiFi or Zigbee, are inherently asynchronous. Network access conflicts are prevalently resolved through Carrier Sensing Multiple Access. In applications with large number of sensors with slow duty-cycles Time Division Multiplexing (TDM) has advantages over other multiplexing methods, as nodes can have both transmitter and receiver in sleep mode while their wait their turn, conserving energy, therefore especially useful for battery powered devices.

TDM solutions which address the need to bound delivery latency are also emerging but in general found to be NP-hard [3]. Several papers proposed heuristic solutions for TDM scheduling, aimed at allocating regularly a portion of a time frame to each node while meeting a given criterion of fairness (e.g. max-min fairness in [4]) .

To do scheduling, protocols such as the USAP [5] and the DTSAP [6] use a message-passing approach, while DRAND [7] and the method in [8] formulate the time scheduling problem as an instance of graph-coloring problem. The main drawback of TDM scheduling algorithms is that, not only they often depend on preceding synchronization, but have the need for global control information (such as the nodes' ID, destination, neighbors, data rates etc.) which needs to be exchanged over two additional channels, one for synchronization and one for control signaling (presumably a random access channel) prior to eventually assigning a portion of transmission time (or a non-conflicting color) to each user in the transmission channel. The adaptation to changing channel and traffic condition is cumbersome. Their large overhead and lower resilience compared to random access protocols clearly makes them less competitive, even in sensing applications that require to meet deterministic deadlines. In the mentioned protocols do not provide, but instead rely on out of band synchronization.

2 System Setup

In this section we describe the goals of the protocol and the models used in the implementation. The goal of this protocol is to provide easy to deploy decentralized synchronization and scheduling, archiving proportional fairness among the member regarding the distribution of scheduling time. This means that a node with a high demand in communication will receive a greater share of the available transmission time. We also want

• *R. Gentz, L. Ferrari and A. Scaglione are with the Department of Electrical and Computer Engineering, University of California, Davis, CA, 95616.*

to emphasize the fact that are proposing a *decentralized* scheduling algorithm, allowing easy scalability. In addition this protocol is also designed to limit the possibilities and damage potential of a rogue node or attacker to the nodes the attacker is directly connected to, and make it hard for an attacker to propagate wrong or misleading information through the network. The allowance that an attacker can disturb the nodes he is directly connected to is very reasonable in a shared media, as one option for an attacker would be to jam the shared medium, making any communication in range of the attacker impossible, however any communication outside of the jammed area is working normally.

The medium we are using this protocol is shared, In order to avoid collisions, we need to ensure that multiple nodes are not interfering each others transfers, especially we need to take care of hidden stations,for with we are introducing cluster heads (CH). Similar to the IEEE 802.11 standard transmissions are only allowed from and to the CH. In addition each transmission requires a clear-to-send packet (start beacon s) and ready-to-send packet (acknowledge ack) pair, which ensures that each node is aware of hidden stations. We assume nodes are connected in a mesh network, consequently it is possible for nodes to be connected to multiple CH, we call these nodes shared-nodes. In contrast there are nodes which are only connected to one CH and are called non-shared-nodes.

We are opposing a cross-layer protocol connecting the physical layer with the medium access layer. In fact we are using the arrival time of a signal to estimate the internal clock of the transmitter. The accuracy we can archive with this method is limited by the bandwidth and the signal to noise ratio (SNR) as described in sec.5. For our model we are assuming the usage of wireless communication with the physical specifications of the Zigbee standard: 2.4GHz center frequency with x MHz bandwidth, using a transmission power of 30dBm with an omnidirectional antenna. We are modeling the link between nodes using the Okumura-Model[10], assuming an urban environment.

3 The CODTM protocol

3.1 Definition of communication

In the communication system each node has its own local clock, the following assumptions are explained within this paragraph: 1) We define without loss of generality that this clock can be separated in a continuous part of length T_{PCO} and a discrete part of length L, further let us assume that the continuous part has a length $T_{PCO} = 1$, i.e. time is normalized with respect to the period of the clock used for network synchronization. 2) Let us assume that that the frame duration for data packets is is large and contains several multiples of T_{PCO}. 3) We assume that there are three kind of *beacons* and each has duration strictly less than 1/2 of T_{PCO}.

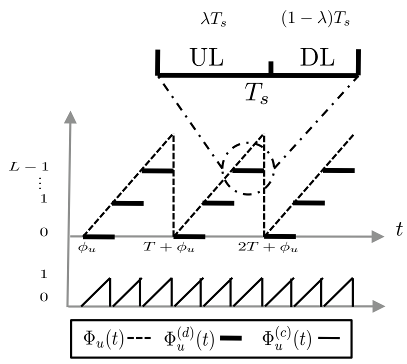


Fig. 1. Decomposition of the phase variables $\Phi_u(t)$ into discrete and continuous parts.

We can model assumption 1), the time of the node u as the following phase variable:

$$\Phi_u(t) = (t - \phi_u)\frac{L}{T_{PCO}} \pmod{L}, \tag{1}$$

where $\phi_u \in [0, T)$ represents the time offset between different nodes with respect to a virtual periodic time reference[1]. This reference can $\Phi_u(t) \in \mathbb{R}$, however due to the modulu operation will can find $\Phi_u(t) \in [0, L)$ instead.

We further can model assumptions 2) and 3) as the phase variable can be decomposed uniquely as discrete and continuous modular components:

$$\begin{aligned}\Phi_u(t) &= \Phi_u^{(d)}(t) + \Phi_u^{(c)}(t) \\ &= (\lfloor \Phi_u(t) \rfloor \pmod{L}) + (\Phi_u(t) \pmod{1}) .\end{aligned} \tag{2}$$

If $L \in \mathbb{N}$, then $\Phi_u^{(d)}(t) = \lfloor (t - \phi_u)\frac{L}{T} \rfloor \pmod{L}$ is in the set $\mathcal{L} = \{0, 1 \dots L-1\}$, and can be denoted as the current time slot which node u is experiencing. Also $\Phi_u^{(c)}(t) \in [0, 1)$ represents the continuous changes of the phase between consecutive increments of $\Phi_u^{(d)}(t)$. Fig. 1 illustrates how $\Phi_u(t)$ can be decomposed into the discrete $\Phi_u^{(d)}(t)$ and continuous $\Phi_u^{(c)}(t)$ phases. This can be seen as a phase evolution of a nested clock within a discrete clock which is represented by the discrete phase. Each completion of a period in the continuous clock, will increase the discrete clock by one unit. Another way to look at the interaction of these two parts is the relationship between minutes and hours in a regular clock dial.

In order to have a reliable communication between the regular nodes and cluster-heads, we divide each PCO period in two non-overlapping portions: *Uplink* (UL) and *Downlink* (DL) as shown in Fig. 1. It is assumed that UL can only be used by nodes for sending beacons and data packets, while DL can only be used by CH for acknowledging beacons or sending data packets through the medium. In general UL and DL can differ in their durations, depending on the data send in the UL such that for a value $0 < \lambda < 1$, UL occupies λ and DL occupies $(1 - \lambda)$. For simplicity in the following we

1. It should be noted that nodes do not have access to the value of ϕ_u, as it is a virtual reference to show the state of synchronicity in the network.

assume $\lambda = 1/2$. In our model we assume that each node has two internal timers, denoted as start and end clock, who both share the continuous part denoted as follows:

$$\Phi_u^s(t) = s_u(t) \pmod{L} + \Phi_u^{(c)}(t) \pmod{1}$$
$$\Phi_u^e(t) = e_u(t) \pmod{L} + \Phi_u^{(c)}(t) \pmod{1}$$

However as CH are only responding to requests from nodes they do not need a start and end clock but instead use (2) as a time reference for all of their operations.

In the following subsections we explain

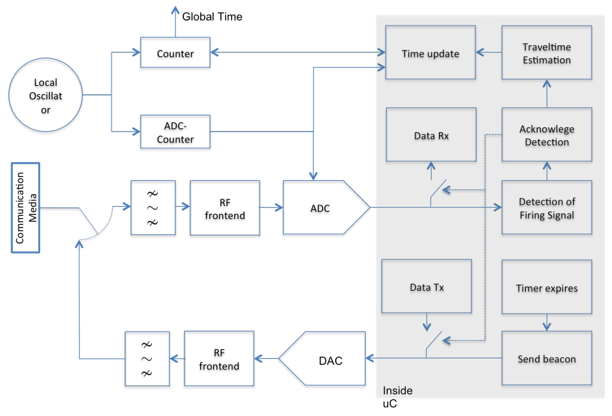


Fig. 2. Block scheme of a node

3.2 Synchronization Update

So far we have assumed that time grids in different regular nodes and cluster-heads are aligned. However, since there is not any communication scheme in other network layers, nodes are not necessarily synchronized. In this part we discuss an updating procedure which gradually achieves time synchronization. Previously, the protocol was described based on the periodic emission of start, end and ACK beacons through the network. Although initially these signals are not sent aligned to a synchronized time grid, but all of them are emitted at the starting edge of the UL or DL time slot of the sender. Thus, at the receiver, time of arrival of the detected signal indirectly represents the misalignment between the receiver's and transmitter's time grids. In order to compensate these time differences, a PCO update as described in Section **??** is performed: Upon receiving a Start/Ack beacon-, or End/Ack beacon pair each Node[2] , exempt the node initializing this communication, update their continuos time with

$$\Phi_c^{(c)}(t^+) = \min\{(1+\gamma)\Phi_c^{(c)}(t), 1\}. \tag{3}$$

for the coupling factor $\gamma > 0$. In general we want to choose the coupling factor as large as possible to allow fast convergence. However as shown in [11], γ large creates instabilities, depending on the network structure. We noticed that based on our simulation that γ has to be smaller the larger the network is. We suspect this is because time updates require more time to reach the other end of the network.

3.3 Scheduling Update

As shown in [12] for an all-to-all connected network the PCO-inspired scheduling update lead to a demand bases fairness, assigning each node u a communication time according to their demand D_u compared to the networks total demand. We use the the same update mechanism and will show that the properties also hold for non fully connected networks.

We define a nodes predecessor (pre) and successor (suc) as following: $pre(u)$ is a node, among all the nodes which conflict node u, which has sent out its end beacon right before node u. $suc(u)$ is a node, among all the conflicting nodes with u, which will be sending its start beacon right after node u. This is visualized in Figure 3.

The update for the discrete time part is done at each node u, only when receiving a start/ack beacon from the nodes predecessor. This node then performs the following, using an intended guard distance δ between the nodes. All calculations in this paper for the discrete scheduling part are modulus the global amount of time slots, which we assume given.

$$s_u^{target}(t) = \frac{\delta}{D_u + 2\delta} s_{u-1}(t) + \frac{D_u + \delta}{D_u + 2\delta} e_{u+1}(t) \tag{4}$$
$$e_u^{target}(t) = \frac{D_u + \delta}{D_u + 2\delta} s_{u-1}(t) + \frac{\delta}{D_u + 2\delta} e_{u+1}(t) \tag{5}$$

Which shows the target value of s and e. It is worth mentioning $s_{u-1}(t)$ is 0 at the time of firing. In the equation δ acts as a guardspace between the nodes, which is intended to be left empty. This acts as an entry point for new nodes and as an indication to other nodes about the internal demand.

Then the update is performed by moving the local start and endpoints towards the target:

$$s_u(t^+) = (1-\alpha)s_u(t) + \alpha s_u^{target}(t) \tag{6}$$
$$e_u(t^+) = (1-\alpha)e_u(t) + \alpha e_u^{target}(t) \tag{7}$$

Where $\alpha \in (0,1)$. However as shown in [12] it is possible that two nodes can change order with this mechanism. Therefore we alter our approach to:

$$\tilde{s_u}^{target}(t) = \min\left(s_u^{target}(t), \frac{s_u + e_{u+1}}{2}\right) \tag{8}$$
$$\tilde{e_u}^{target}(t) = \max\left(e_u^{target}(t), \frac{s_{u-1} + e_u}{2}\right) \tag{9}$$

The fix point is not altered by this mechanism, as shown in [13] it limits the maximum changing amount in order to avoid overlapping. Numerical simulations show that convergence properties are not affected.

The update equation shown in the right of (6) is not necessarily a discrete value in $\mathcal{L} = \{0, 1, \ldots, L-1\}$, and

2. This algorithm also works when a node misses some beacon pairs, due to interference, or shutdown of the Rx module in order to save energy.

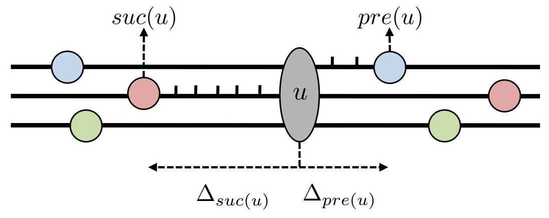


Fig. 3. In a network with three clusters, and node u which can hear all cluster-heads, definitions of $pre(u)$, $suc(u)$, $\Delta_{pre(u)}$ and $\Delta_{suc(u)}$ are shown in this figure. The horizontal axis shows the values of discrete phases ($\Phi_u^{(d)}(t)$).

thus should be projected into $\mathcal{L}$ again

$$s_u(t^+) = \mathcal{Q}[(1-\alpha)s_u(t) + \alpha s_u^{target}(t)] \tag{10}$$

$$e_u(t^+) = \mathcal{Q}[(1-\alpha)e_u(t) + \alpha e_u^{target}(t)] \tag{11}$$

where $\mathcal{Q}(\cdot)$ is a quantization function. We use dithered Quantization [14] defined as follows

$$\mathcal{Q}(x) = round(x + v) \tag{12}$$

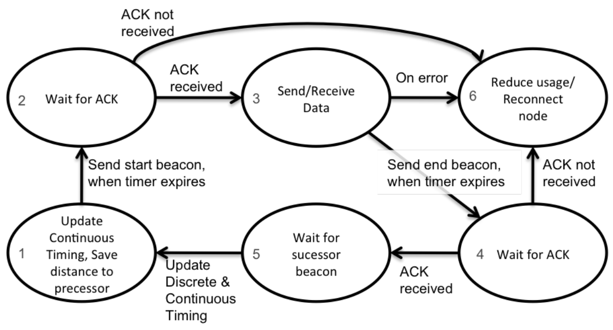
where $x \in \mathbb{R}$ and $v \sim \text{unif}(-1/2, 1/2)$ is drawn from a uniform distribution. The idea is analogous to the *probabilistic quantization* used in [15] to ensure the convergence of an average quantized consensus policy. Convergence analysis of dithered quantization in discretized DESYNC with only one cluster and synchronized nodes can be found in [16].

3.4 State transitions in nodes and cluster heads

Each node is executing a program shown in the Fig. 4. Similar to the PCO based methods in [**?**], each node $u \in \mathcal{V}$ sends out a *firing beacon* in the UL whenever its timer expires as seen in state transition from state 1 to 2. Then the node waits for an acknowledge from the CH and upon receiving the the mentioned, the node knows that the channel is free to use and starts transmitting data (3). Once the node's end timer times out then the node will stop transmission of data an send an end beacon. The CH will acknowledge this end beacon (4) and the node is now ready to update its timing information (5). This is done upon receiving the start beacon of the next node, which is acknowledged by the CH then the node updates their timing based on equation (**??**). However for this equation we need the distance from the nodes start node to the nodes precessing's node end node. Therefore a node is using the saved value from (6). There the node is constantly monitoring the the network and recoding the distance from its start beacon to the end beacon of its precessor. However as all end beacons are the same the node will take the last end beacon it heard as a reference, which is the desired node, the precessor. In (6) in addition to saving the processors position the node also can use each acknowledged start and end beacons to update the continuos time (3) as updating it does not interfere with the scheduling algorithm, therefore archiving faster convergence.

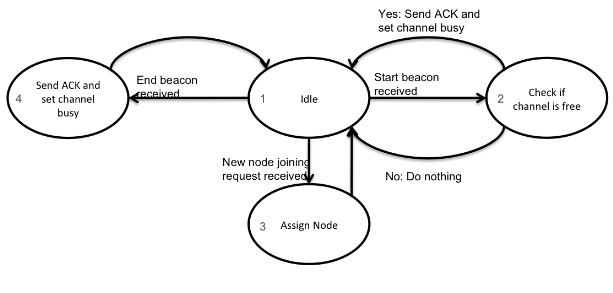
In the case of an error of the CH not acknowledging a beacon the node will go to state 6 and has to change his behavior. At the first time entering state 6) the node will reduce its time from start to end beacon to the minimum, resulting in a low overlapping chance. And then enters (2) as normal. If this reduction was successful on the next firing the CH will acknowledge and we successfully solved the problem. However in case the minimal distance is still not acknowledged the node has to reconnect to its CH, as backing up to the maximum has not brought the expected result and the easiest way is to rejoin the network as described in section 7.

One possible extension to this protocol is to add an power saving idle state, with both rx and tx shut down in between (5) and (1), as for the scheduling only the very last end beacon, the one from the precessor is of interest, reducing power consumption. The disadvantage of this approach is that we a) need to make sure we turn on the rx in time to receive the precessor end bean and b) it will slow down convergence of the continuos time part.



ACK not received
2 Wait for ACK
ACK received
3 Send/Receive Data
On error
6 Reduce usage/ Reconnect node
Send start beacon, when timer expires
Send end beacon, when timer expires
ACK not received
1 Update Continuous Timing, Save distance to precessor
Update Discrete & Continuous Timing
5 Wait for sucessor beacon
ACK received
4 Wait for ACK

Fig. 4. State transitions in nodes



Yes: Send ACK and set channel busy
4 Send ACK and set channel busy
End beacon received
1 Idle
Start beacon received
2 Check if channel is free
New node joining request received
No: Do nothing
3 Assign Node

Fig. 5. State transitions in cluster heads.

A CH acts as the coordination center of a cluster the CH has to acknowledge beacons whenever the media in the cluster is free and deny/do not ack. if the media is busy. Fig.5 describes the state flow of a CH. Initially the CH is idle waiting beacons and joining requests (1) and monitoring the network saving the current usage in his memory. Uppon receiving a start beacon the CH checks if the media is already in use. If not the beacon is confirmed with an acknowledge and the media is set to busy. In case the media is use no ack. is send and in both cases the CH goes back to the idle state. Uppon receiving an end beacon the CH will always confirm it and mark the media as free. We now describe the CH view of a node joining the network, while a node's view on joining the network can be found in section **??**. Uppon a new node joining the network the CH searches its memory for a free space to join. Depending if the new node is a non-shared node or a shared nodes, the CH will place the node at a new position or keep the nodes position for the reasons described in [**?**]. In general non-shared nodes should next to other non-shared nodes, while shared nodes should be next to other shared nodes, if possible even sharing the same other cluster. However as for a shared node two CH are involved only the first CH can set the position while the second node has to accept the position as is. Therefore if the CH want to enforce that the new shared node does not spit the existing sorting the only thing a CH can do is restart its own local nodes be next to each other, by denying the nodes beacon requests forcing the nodes into a reconnect.

4 How a beacon is designed

We now specify how a beacon is generated and what physical shape it has. The goal of the beacon is to have the smallest timing error, τ^2 as described in the next section (16) as possible. Therefore we have to maximize the SNR and the mean square bandwidth (MSB). It is beneficial to transmit more than one symbol per beacon as we can combine the Signal-power of each symbol. To do this we prefer find a sequence with the maximum autocorrelation peak-to-side lobe p ratio possible and each symbol using the maximum transmission power. Mean that each transmitted symbol has to lie on the unit-circle of the Inphase & Quadrature Phase. However we also require a large mean square Bandwidth. In other words, this means that a beacon is a synchronization sequence as in communications. Therefore we are using industry standard sequences, in our case the Zadoff-Chu Sequences, which are also used in LTE-Wireless, which have excellent properties in a power line channel, which is typically characterized by multi path propagation due to signal reflection and impedance mismatches.

When sending a beacon we say that we have one discrete time slot to send the beacon and receive the answer from the CH. Considering the time needed for the signal to travel the response in the CH being computed and send back we find that a beacon has the maximum length:

$$B_l = \lfloor \frac{(1 - t_c - t_t - t_e)g}{2L} \rfloor \tag{13}$$

where t_c is the computing time, t_t the round-trip-traveling time, t_e the error in round-trip-traveling time and g a guard factor.

The sequence is generated with the following function:

$$a(k) = \begin{cases} \exp j\mu\pi k(k+1+2q)/B_l & , \text{for } B_l \textit{ odd}. \\ \exp j\mu\pi k(k+2q)/B_l & , \text{for } B_l \textit{ even}. \end{cases} \tag{14}$$

Where $a(k) = 0, 1, \ldots, B_l - 1$ is the sequence of the symbols we want to transmit. The parameter μ is an integer prime to B_l.It is shown in [] that two sequences with the same parameters but different μ, i.e μ_1, μ_2, are orthogonal to each other if $|\mu_1 - \mu_2|$ is also prime to B_l.The parameter $q \in \mathbb{Z}$ is an offset parameter, we choose to be 0. It is interesting to note that this code will always have an amplitude of 1, therefore uses the maximum transmission power. We use the fact that we can choose μ to encode information in the beacon, in our case if it is a start or end beacon or if its an acknowledge from a CH. This idea originates from the LTE network, where multiple base station include their ID number in the signal.

Zadoff Chu sequences have good autocorrelation peak-to-side lobe as seen in fig.6. In addition this code has a flat spectrum [**?**,] This has several benefits: First it has a large mean square bandwidth, which reduces the error made in estimating time. Second the flat frequency characteristic is useful in transmissions, as regulation authorities only allow a certain energy per Hz. Therefore since we have a flat frequency use the same maximum transmission power. In addition the flat frequency can be used at the same time to estimate the channel.

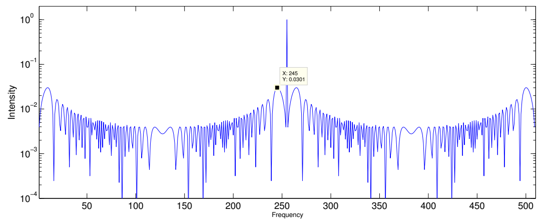


Fig. 6. Autocorrelation of the Zadoff-Cru-Code with length 255, normalized to 1. We see that the code has a peak-to-sidelobe ratio of $\approx 15dB$.

This peak-to-sidelobe ratio is equal to: SNR_{eff}/SNR_{symbol}. Therefore we found the effective SNR of the sequence, this we use in the next section to compute the timing error of a beacon.

5 Travel time estimation of signals

In previous papers the system was assumed to have infinite signal traveling speed without any uncertainty (jitter). In the following we assume a finite noisy traveling speed, which is determined by the distance of the nodes to each other plus an additional signal processing time, which is assumed to be known. This is a reasonable assumption since the processor type and therefore processing latencies are consistent, and therefore assumed to be known by each node.

The algorithm works as seen in figure 7. Node i sends a beacon with his time t_1. This signal is received by the CH and all other nodes in communication range sharing the same CH, denoted as node j. The CH replies this signal back to node i, but this answer is also received

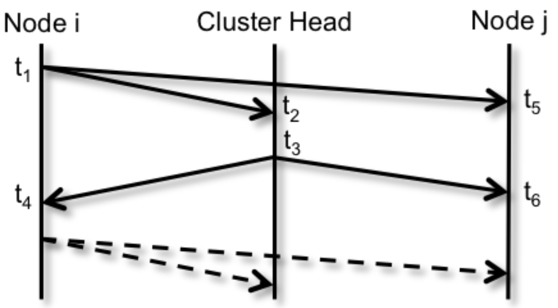


Fig. 7. Estimation of the link latency from a Node i to the CH and all other nodes denoted as Node j. Solid Line: Beacon transmission and corresponding times; Dashed: Data-flow

by node j. From the traveling time to the CH and back and the known computing time t_c in the CH, node i can compute its distance to the CH. This information is then broadcasted though the system. Out of this information the CH and node j are able to compute the time t_1 the original transmission time of node i, given that node j knows its distance to the CH from a previous measurement.

However in the physical measurement each signal (and therefore our timing information) arrives with a certain error e because a perfectly sharp delta impulse resulting in perfect time resolution , is not reasonable since it would require infinite bandwidth (see [17] chapter 3). In this book the time of a signal arrival is given by:

$$t_{rx} = t_{tx} + t_{tr} + e \tag{15}$$

where t_{tx} and t_{rx} the transmit/receiving time and t_{tr} is the noise free signal traveling time, calculated as the distance divided by the speed of the signal (speed of light in air/media). With e the time estimation error. We model the distribution of $e_j(t_i)$ as $\mathcal{N}(0, \sigma_j^2)$, with σ_j^2 equal to the Cramer-Rao-Bound for time estimation in Gaussian white noise, which we assume is achieved by the receiver estimator (see e.g. [17] Ch.3):

$$\sigma^2 \geq (SNR_{(i,j)} \cdot \bar{F^2})^{-1} \tag{16}$$

Here $SNR_{(i,j)eff}$ is the effective signal-to-noise-ratio of the beacon from node i to node j as derived in the previous section and $\bar{F^2}$ is the mean square bandwidth of the firing signal envelope $s(t)$:

$$\bar{F^2} = \frac{\int_{-\infty}^{\infty} (2\pi F)^2 |S(F)|^2 dF}{\int_{-\infty}^{\infty} |S(F)|^2 dF} \tag{17}$$

where $S(f)$ is the Fourier transform of the firing signal envelope $s(t)$. As our beacon is generated by a Zadoff-Cru sequence it has maximum bandwidth. Therefore $\bar{F^2}$ only depends on the pulse-shaping filter of the transmission system, which is typically a root raised cosine filter.

We can write the expected absolute value of timing error for an efficient receiver reaching the Cramer Rao bound can be written as:

$$\mathbb{E}(|e|) = \sigma \sqrt{\frac{2}{\pi}} \tag{18}$$

This knowledge is now used in the following algorithm to compute the timing error when nodes communicate with the PCO-Protocol. We denote the estimate of a time with t^e while the true value has no index.

We can find the estimated travel time is the following, given that the computation time $t_3 - t_2$ is common knowledge among the nodes.

$$t^e_{i,ch} = \frac{t_4 - t_1}{2} - (t_3 - t_2) = t_{i,ch} + \frac{e_{i,ch} + e_{ch,i}}{2} \tag{19}$$

We can extend this to find the relation

$$t^e_{i,j} = t5 - t1 =$$
$$t5 - t6 + t_{i,ch} + t_{ch,b} + \frac{e_{i,ch} - e_{ch,i} - e_{ch,j} - e_{j,ch}}{2} + e_{ch,j2} \tag{20}$$

The distance $t_{i,j}$ has the additional error $e_{ch,j2}$, since the distance $t^e_{ch,j}$ has been measured in a previous round. Therefore the estimation of the error is no longer correct, assuming coherence time has passed. The error is Gaussian with zero mean, and therefore symmetrical around 0. Therefore the error of a transmission is given by the following:

$$\mathbb{E}(|e_{i,j}|) = \mathbb{E}(|2e_{j,ch} + e_{i,ch}|) = (2\sigma_{j,ch} + \sigma_{i,ch}) \sqrt{\frac{2}{\pi}} \tag{21}$$

In contrast to this the CH is can update its time more accurate (19):

$$\mathbb{E}(|e_{i,ch}|) = \mathbb{E}(|e_{i,ch}|) = \sigma_{i,ch} \sqrt{\frac{2}{\pi}} \tag{22}$$

From this we can conclude that the timing estimations from node i to the CH is way more accurate than from node i to any other node j in the cluster.

6 Algorithm to Decentralized Assign Cluster Heads

In this section we describe how to assign cluster heads in a static network in a decentralized way. In practice a lot of CH are already preassigned, as in the Network is is known where data is aggregated and commands are generated. However each node needs to be assigned to a CH. The following algorithm automates this work by 'promoting' regular nodes to CH. We formulate the algorithm to have the following goals: 1) The algorithm ensures that two cluster heads are not next to each other, in order to reduce the amount of shared nodes , and 2) Minimize the amount of Cluster heads needed, to reduce interference of the clusters with respect to each other. 3) The network itself is able to assign the first CH, which can act as a starting point to the network. This is done under the constraint, that the algorithm has to run

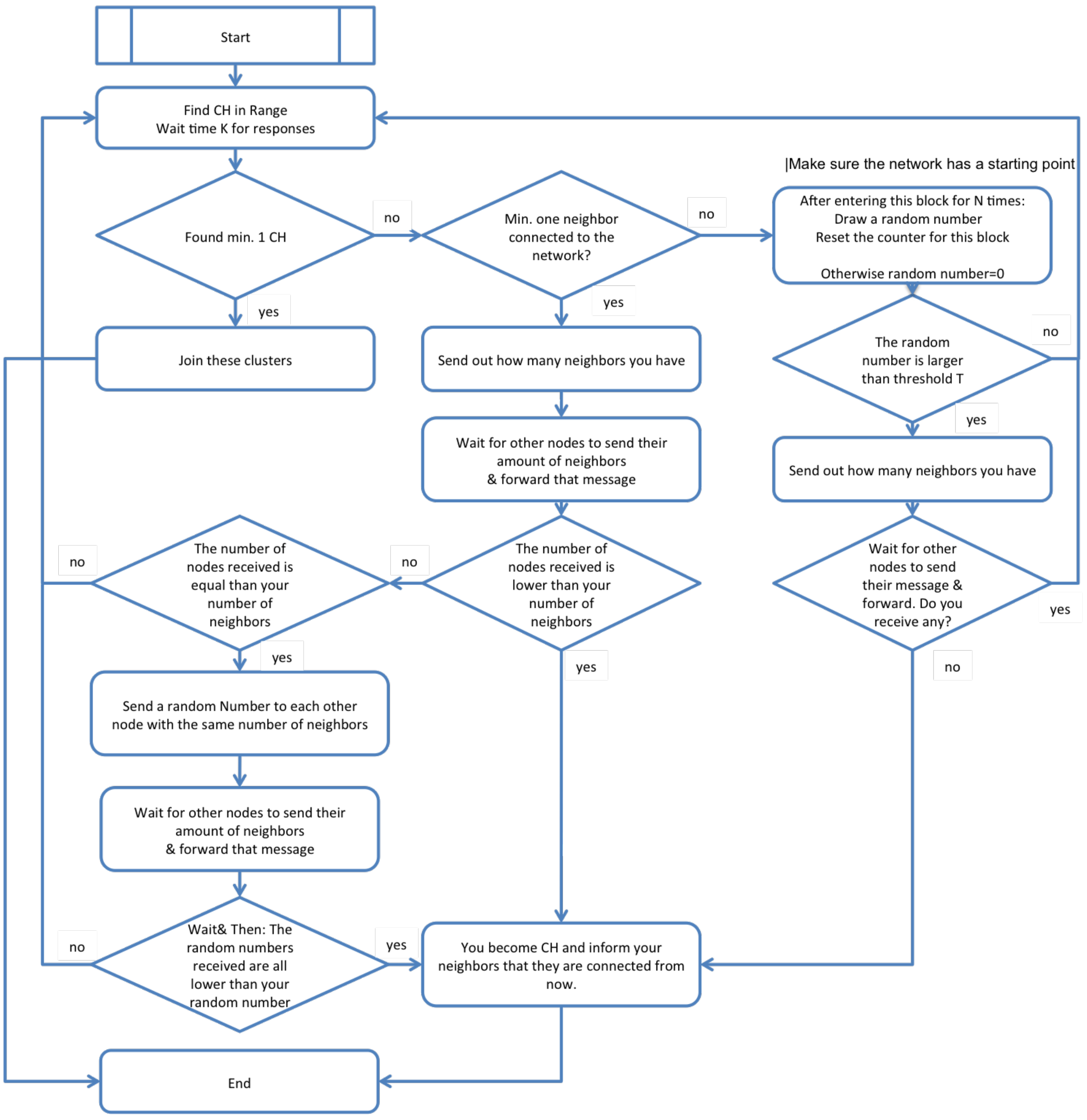

Start
Find CH in Range
Wait time K for responses
|Make sure the network has a starting point
Found min. 1 CH
no
yes
Min. one neighbor connected to the network?
no
yes
After entering this block for N times:
Draw a random number
Reset the counter for this block
Otherwise random number=0
Join these clusters
Send out how many neighbors you have
The random number is larger than threshold T
no
yes
Wait for other nodes to send their amount of neighbors & forward that message
Send out how many neighbors you have
The number of nodes received is equal than your number of neighbors
no
yes
The number of nodes received is lower than your number of neighbors
no
yes
Wait for other nodes to send their message & forward. Do you receive any?
yes
no
Send a random Number to each other node with the same number of neighbors
Wait for other nodes to send their amount of neighbors & forward that message
Wait& Then: The random numbers received are all lower than your random number
no
yes
You become CH and inform your neighbors that they are connected from now.
End

Fig. 8. Flowchart to assign cluster heads in a decentralized way.

decentralized, with every node running the same simple program.

Goal 3 is particularly useful in the case of an outage of the last communication link. For example a house has its own solar power generation, which needs a clock/phase of the grid as input, but the communication/powerline to the power grid breaks. Being able to start a new network allows the PCO to start, giving the needed clock input to the solar generation. The result is, that an island is formed which is still able to function. Once the line to the grid has been restored the PCO will automatically realign with the power grid allowing the solar generation to deliver power in phase to the grid. (However a circuit breaker or a temporary turning off of the solar generation is needed until PCO has converged. Otherwise the power generation from the grid and the solar generation from the previous island can be out of phase.)

Our algorithm works as seen in Fig.8, which we now describe: Initially a node scans its surrounding for existing PCO-traffic. If he can hear one or more clusterheads, the node will join that network. In case the node can hear some traffic but no cluster head, this means that the node is at the border of the existing network and we will need to assign another cluster head. To make sure we assign the minimal amount of CH we apply a greedy alorithm. Each node which is at the border to the existing network broadcasts the number of nodes he as a new

CH could cover. This message is forwarded by any node not yet assigned to any CH. Then after all responses arrived the node decides if he covers the most nodes. If so he becomes a new CH and informs his neighbors which will join his new cluster. If there are other nodes with more neighbors covered they first do the previous operation and the rest of the network waits and see if their situation changes. However in the case of multiple node having the same number of nodes to cover we have to make sure only one of them will get CH at a time, in order no to assign too many CH. Therefore the nodes in question send each other a random number. Whoever has the highest random number becomes a new CH.

If a node is This process gets repeated until all nodes are covered, i.e. have a CH.

However there is the case that there is not yet an initial network to join to. Therefore we need to give every node the chance to be the initial point. This is done by giving each node who cannot hear any network a small chance to start the assignment process of a CH. This ensures that we do not need to assign an initial CH to begin with as the network will itself after a certain time. However to waiting time N has to be chosen so large, that the network has enough time to form before the next initial point is set to give nodes connecting without this path priority and also avoid unnecessary traffic.

With the threshold T we can experiment. One option is to set it $NoNeighbors^{-1}$ to emphasize stating cluster which have more nodes covered. If no other metric is found we can set it to $-\infty$

7 Effect of a new node joining the network

Initially a node joining a network has only limited information about the network. The node knows what he received in this radio, which includes which and how many CH are in range and an approximate usage in each cluster. The node using this information in order to join the network as follows:

When a new node is joining the network we have to distinguish two cases, whether the new node is a shared node or not. We will see in section 8 that it is beneficial if all shared nodes are next to each other; And all non-shared nodes are. In principle a CH assigns a discrete part to the joining node, whereas the node itself choses the continuous part at random.

When adding a non-shared node, this Node will send a request to the CH, making sure the channel is not in use when sending this request, via *carrier sense collision avoidance* (CSCA). The CH receiving this request will assign a free slot next to one of the existing local nodes, which is done by transmitting the time the node needs to wait until its own first firing.

When adding a shared node we have the issue that there has to be a new slot at the same time in each cluster the node is connected to. However quite often this will not be the case, also because one goal of scheduling in general is to use the available resources as good as we can, which means there are little to no resources unused. Therefore it becomes clear that the new shared node can only confirm with one CH and has to 'brute force' its way into other clusters. The algorithm we use is as follows: The new node tries to guess which one of the CH has the highest constraints, by observing the traffic for one cycle. Then the node sends this CH a request to join the network as a shared node, communicating which other CH he can see. (Also using CSCA as above). The CH will then assign a slot, if possible next to another shared node with the same shared cluster(s). Otherwise the CH assigns a slot between the shared and non shared nodes. This ensures that existing shared nodes which share the same other CH stay next to each other.

When adding a new CH the new Ch announces this to the network. This makes nodes in range connect to the new CH either as local node or as a shared node. However when a adding a CH, previously non-shared nodes can become shared with the new CH. Therefore all the nodes converting from a non-shared to a shared node will restart their scheduling, requesting a new spot from the existing CH to ensure that they, as shared node are next to the other shared nodes.

8 Proof of Convergence of PFS

In this part we explore the behavior of the CODTM. For simplicity of the convergence analysis we analyze the model in equation 4 and 6. A proof of convergence of the COTDM for Proportional Fair Scheduling is given in [?] for a single cluster. We define $\Gamma_u(t) = \Phi^s_u(t) - \Phi^e_u(t) \pmod 1$ and $\Theta_u(t) = \Phi^e_u(t) - \Phi^s_{u-1}(t) \pmod 1$. Therefore we can describe the complete system with the following vector of size $2N$:

$$\Delta(t) = [\Theta_1(t), \Gamma_1(t), \Theta_2(t), ..., \Gamma_{n-1}(t), \Theta_n(t), \Gamma_n(t)]$$

the proof in [?] show that for an all to all connected network there exist a unique fixed point where the system can converge to which is:

$$\Delta^* = \frac{\beta}{D}(\delta, D_1, \delta, D_2, ..., \delta, D_n)^T$$

where $D = \sum_{u=1}^{n} D_u$ and $\beta = \dfrac{D}{D + n\delta}$.

In this paper we want to extend this to a multicluster scenario, and we want to show what happens if a node can hear multiple cluster heads. An example topology with two clusters is shown in fig. 9 below:

We can represent the two different clusters with two different system vectors, with two shared variables that are referred to the shared node that we can consider, without loss of generality, the last node of both the clusters, respectively the n-th one and the l-th node where N and L are the cardinality of the two clusters:

$$\Delta_N(t) = [\Theta_1(t), \Gamma_1(t), \Theta_2(t), ..., \Gamma_{n-1}(t), \Theta_n(t), \Gamma_n(t)]$$
$$\Delta_L(t) = [\Theta_1(t), \Gamma_1(t), \Theta_2(t), ..., \Gamma_{l-1}(t), \Theta_l(t), \Gamma_l(t)]$$

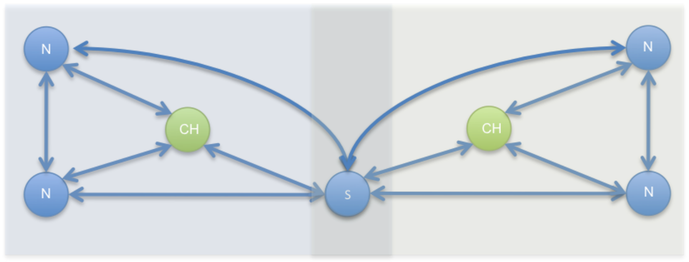


Fig. 9. Network topology with two clusters, each with a Cluster Head *CH* and Non-Shared Nodes *N* and Shared node among different clusters *S*

The shared nodes can hear all the nodes in the two clusters. However as seen in they will update only with the closest neighbors from any of the two clusters as seen in fig. 3.3.

In the following we will show that having any configuration of nodes will each lead to a unique fix point under certain conditions.

8.1 One shared Node, updating with one cluster

Assume that for now we only have one shared node. We need to recall the definition of the system matrix $\mathbf{M}$ describing one whole round of updates:

$$\mathbf{M} = \prod_{u=1}^{N} \mathbf{M}_u$$

where $\mathbf{M}_u^N$ represents the matrix for the update of node u in the cluster N. Which has the following form for all nodes updating with that cluster:

$$\mathbf{M}_u = \begin{pmatrix} \mathbf{I}_{(2u-2)} & & \mathbf{0}_{(2u-2)\times(2N-2u+2)} \\ \mathbf{0}_{3\times(2u-2)} & \mathbf{U}_u & \mathbf{0}_{3\times(2N-2u-1)} \\ \mathbf{0}_{(2N-2u-1)\times(2u+1)} & & \mathbf{I}_{(2N-2u-1)} \end{pmatrix}$$

where

$$\mathbf{U}_u = \begin{pmatrix} 1-\alpha\dfrac{D_u+\delta}{D_u+2\delta} & \alpha\dfrac{\delta}{D_u+2\delta} & \alpha\dfrac{\delta}{D_u+2\delta} \\ \alpha\dfrac{D_u}{D_u+2\delta} & 1-\alpha\dfrac{2\delta}{D_u+2\delta} & \alpha\dfrac{D_u}{D_u+2\delta} \\ \alpha\dfrac{\delta}{D_u+2\delta} & \alpha\dfrac{\delta}{D_u+2\delta} & 1-\alpha\dfrac{D_u+\delta}{D_u+2\delta} \end{pmatrix}$$

The cluster the shared node updates with, we assume as the one with cardinality N, will have the same matrix $\mathbf{M}$ described in [**?**], and hence the same dynamic. Let's show now what happens to the node not updating the shared node, say the cluster with cardinality L. Noteing the shared node as the l-th node in Cluster L, the matrix $\mathbf{M}_l^L$ will be the identity matrix and so the matrix of the cluster is: $\mathbf{M}^L = \prod_{u=1}^{L-1} \mathbf{M}_u \cdot \mathbf{I}$.

$$\mathbf{M}^L = \begin{pmatrix} & & & & 0 \\ & & & & 0 \\ & \mathbf{M}^L_{(2N-1)\times(2N-1)} & & & \vdots \\ & & & & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

We can further see that each $\mathbf{M}_u^L$ is a left stochastic matrix and so is the product $\mathbf{M}^L$. We note that if we take the submatrix of dimension $2L-1$ from the top left of $\mathbf{M}^L$, this will also be left stochastic matrix cause we just removed a 0 value from each column. Furthermore the submatrix is primitive cause it contains all positive elements

We can then apply the Perron-Frobenius Theorem to the submatrix, which ensures that the sub matrix has exacly one eigenvalue equal to 1 and $2L-2$ eigenvalues inside the unit circle. Considering the submatrix of dimension $2L-1$ we are in the same situation of the proof in [**?**] and we can say that the fixed point of the (2L-1)-subsystem is represented by:

$$\Delta^*_{2L-1} = \frac{\beta}{D}(\delta, D_1, \delta, D_2, ..., \delta, D_{l-1}, \delta)^T$$

If we look back at the complete matrix M^L, we that the last row adds an eigenvalue 1. Therefore the eigenvalue of one has multiplicity equal to 2 and we can find the two eigenvetors associated:

$$\begin{aligned} \Delta^*_1 &= \frac{\beta^{l-1}}{D^{l-1}}(\delta, D_1, \delta, D_2, \ldots, \delta, D_{l-1}, \delta, 0)^T \\ \Delta^*_2 &= \quad (0,0,0,0,\ldots,0,0,0,1)^T \end{aligned}$$

with $D^{l-1} = \sum_{u=1}^{l-1} D_u$ and $\beta^{l-1} = \dfrac{D^{l-1}}{D^{l-1}+l\delta}$.

The possibility to have this linear combination allows the shared node to reach the convergence of the cluster that make it update, without losing his slot in the other cluster. In fact, the cluster that make it update is a complete system with a unique fixed point and so

$$\lim_{t\to\infty} \Gamma_m(t) = \frac{\beta^m}{D^m} D_m$$

with $D^m = \sum_{u=1}^{m} D_u$ and $\beta^m = \dfrac{D^m}{D^m+m\delta}$.

We define:
$w^{l-1} = \dfrac{\beta^{l-1}}{D^{l-1}}$ and $w^m = \dfrac{\beta^m}{D^m}$

then the vector

$$\begin{aligned} \Delta^*_L = &((1-w^m D_m)w^{l-1}\delta, (1-w^m D_m)w^{l-1}D_1, ... \\ &..., (1-w^m D_m)w^{l-1}\delta, w^m D_m)^T \end{aligned}$$

represents a fixed point for the system L cause we can see that:

- $||\Delta^*_L||_1 = 1$
- $\Delta^*_L = (1-w^m D_m)\Delta^*_1 + (w^m D_m)\Delta^*_2$

where Δ_1^* and Δ_2^* are the two eigenvectors defining the subspace of fixed points of system L. We can see that the proportional fair scheduling is conserved between the non-shared nodes in the cluster L but we lose the proportionality between the node m and the others, cause the slot assigned to the node m is decided by the cluster M and hence is based upon the percentual weigth of its demand among the cluster M that can be different from the one in the cluster L. We now have to compare the two idle slots before the node in the two system (δ_M, δ_L) and verify the node makes the update with the closest previous:

$$\begin{aligned}\delta_M &= w^m \delta \\ \delta_L &= (1 - w^m D_m) w^{l-1} \delta\end{aligned}$$

Now, with simple calculations we can obtain:

$$\begin{aligned}\delta_M \lesseqgtr \delta_L \;&\Leftrightarrow\; D^{l-1} + l\delta \lesseqgtr D^m - D_m + m\delta \\ &\Leftrightarrow D^{L-1} + (l-1)\delta \lesseqgtr D^{m-1} + (m-1)\delta\end{aligned}$$

and see that if a cluster has a lower overall demand excluding the share node, the other cluster will have a smaller idle slot and make the shared node to update with it. So, even if we consider an initial random distribution of nodes around the clock in the two cluster, at the steady-state the shared node always update with the most demanding cluster, and this is good for us considering the previously described partial loss of fairness in this scenario. If the node made the update with the lower demanding cluster, it would take a big portion of the bandwidth dedicated to the bigger cluster and inhibite all the other nodes to reach their fair portion: it's definitely better to respect fairness in the higher-demand context.

8.2 Two or more shared nodes, updating with one cluster

If we have more than one nodes shared among two clusters, we can extend our analysis and highlight that, if we have two nodes (m,l) updating with the same cluster, the "incomplete" system has now four eigenvectors associated to the eigenvalue 1:

$$\begin{aligned}\Delta_{L_1}^* &= w^{m-1}(\delta, D_1, \delta, D_2, ..., \delta, D_{m-1}, \delta, 0, 0, ..., 0)^T \\ \Delta_{L_2}^* &= (0, 0, ..., 0, 0, 1, 0, 0, ..., 0, 0)^T \\ \Delta_{L_3}^* &= w_{m+1}^{l-1}(0, 0, 0, ..., 0, \delta, D_{m+1}, \delta, D_{m+2}, ..., \delta, D_{l-1}, \delta, 0)^T \\ \Delta_{L_4}^* &= (0, 0, 0, ..., 0, 1)^T\end{aligned}$$

with $w_{m+1}^{l-1} = \dfrac{\beta_{m+1}^{l-1}}{D_{m+1}^{l-1}}$, $D_{m+1}^{l-1} = \sum_{u=m+1}^{l-1} D_u$, $\beta_{m+1}^{l-1} = \dfrac{D_{m+1}^{l-1}}{D_{m+1}^{l-1} + (l-m)\delta}$.

The situation of having several shared node not next to each other is not desirable because we can not have a rigid constraint on the coefficient of $\Delta_{L_1}^*$ and $\Delta_{L_3}^*$, having infinite fixed points and losing the fairness between these two subgroups.

Therefore this topology our cluster heads have to ensure that these shared nodes are consecutive to obtain these four different eigenvectors:

$$\begin{aligned}\Delta_{L_1}^* &= w^{l-2}(\delta, D_1, \delta, D_2, ..., \delta, D_{l-2}, \delta, 0, 0, 0)^T \\ \Delta_{L_2}^* &= (0, 0, ..., 0, 1, 0, 0)^T \\ \Delta_{L_3}^* &= (0, 0, 0, ..., 0, 1, 0)^T \\ \Delta_{L_4}^* &= (0, 0, 0, ..., 0, 0, 1)^T\end{aligned}$$

In this situation, there is no significant difference between the previous described case. The complete cluster gives an initial constraint on the value of the three last variables of the system L, and so we can find the coefficients λ_2,λ_3,λ_4 for $\Delta_{L_2}^*$,$\Delta_{L_3}^*$,$\Delta_{L_4}^*$ and then multiply $\Delta_{L_1}^*$ with $(1 - \lambda_2 - \lambda_3 - \lambda_4)$. Then we have to verify the idle slots to find the final solution; since the last three variables are equal in both systems we can assume that if the first of them update with a cluster, the second updates with the same. If the cluster L is the incomplete one we can calculate:

$$\begin{aligned}\delta_L &= [1 - w^m(D_{m-1} + \delta + D_m)] w^{l-2} \delta \\ \delta_M &= w^m \delta\end{aligned}$$

and then:

$$\begin{aligned}\delta_M \lesseqgtr \delta_L \;&\Leftrightarrow\; D^{l-2} + (l-1)\delta \lesseqgtr D^m + m\delta - D_{m-1} - \delta - D_m \\ &\Leftrightarrow D^{l-2} + (l-2)\delta \lesseqgtr D^{m-2} + (m-2)\delta\end{aligned}$$

and find the same relation that we found for the previous case.

8.3 One Node synchronizing with two clusters

There the case when a node is not fully synchronizing with one cluster. This can happen if the cluster has multiple shared nodes but also when the cluster only has one shared node which in another cluster is next to a *non-updating* node, as seen in fig.10 .

For this section we have two separate cases we have to describe: In one cluster there are all nodes fully synchronizing with that cluster exempt one node who is synchronizing one of his beacons with another cluster. If we look at this example closer we notice that this means that one of the points in the system is fixed but all others remain flexible. Therefore the non shared nodes cluster will converge just as a cluster will all node synchronizing with that cluster, except that the cluster has one fixed node and therefore has to rotate (need to find better wording). However the shared node is now updating with two different clusters. Therefore he has its own update M_u, described as follows.

$$\begin{aligned}\Delta_1^* &= \frac{\beta^l}{D^l}(\delta, D_1, \delta, D_2, \ldots, \delta, D_{L-1}, \delta, D_S^L)^T \\ \Delta_2^* &= \frac{\beta^n}{D^n}(\delta, D_1, \delta, D_2, \ldots, \delta, D_{N-1}, \delta, D_S^N)^T\end{aligned}$$

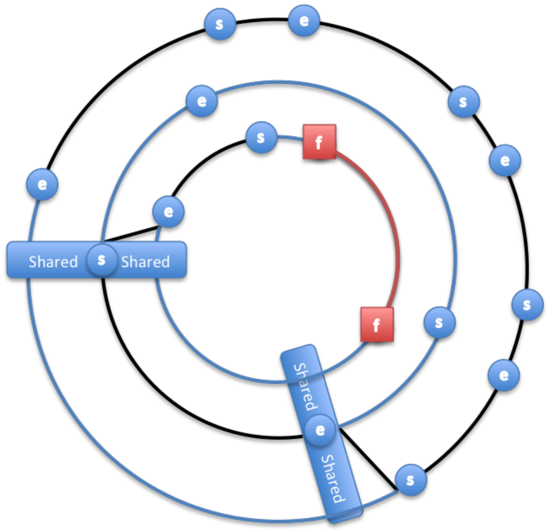


Fig. 10. Synchronizing of a shared nodes start and end beacon in different clusters. We see a fixed node f, in red, from another cluster which will not move, hence is fixed. All other nodes are non-shared nodes. We can see the Space available for the shared node as the black line, which can be in multiple clusters.

Where D_S^L and D_S^N is determined as follows:

$$D_S^L = \frac{D^L}{\beta^L} * \frac{\beta^S}{D^{SC}} * D_S * Space_{avail}$$

$$D_S^N = \frac{D^N}{\beta^N} * \frac{\beta^S}{D^{SC}} * D_S * Space_{avail}.$$

With $S^{SC} = \sum D_L + \sum D_N - D_S$ and

$$\beta^S = \frac{D^{SC}}{D^{SC} + (m + l - 1)\delta}$$ and

$Space_{avail}$ the available space in the clusters which are updating with them, normalized to one per cluster. An example can be seen as the black line in fig.10.

9 Simulation Results

We simulated our approach in Matlab for one multiple clusters.

For the simulation of one cluster we simulated 5 Nodes over 100 iterations with a coupling factor $\alpha = 0.2$. We can see in fig.13 that the system is synchronizing itself and the error after 5 iterations with respect to node1 as a reference is 1e-10. We can further see in fig.11 that the system is organizing itself without overlapping, meaning that there will be no collisions. However after a miss detection of the green node in fig.11 the node was told by the CH to back of as he would otherwise interfere. But as we can see this is not a problem for the system as it is self healing. We can see that 30 iterations later the system has normalized and is dividing the available transmission time according to the demand which we can see in fig.12.

With multiple cluster this algorithm is working as well. We distributed node randomly (fig.14) in a plane giving each node a maximum communication distance.

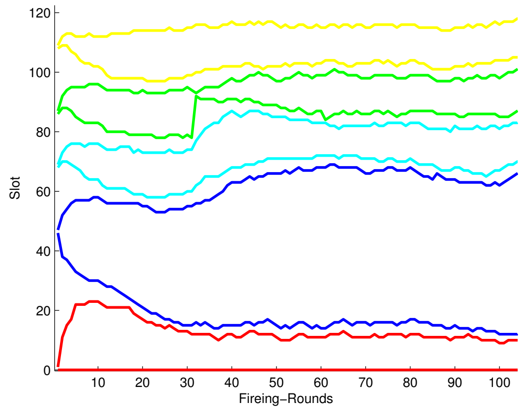


Fig. 11. Convergence of the scheduling algorithm with 5 nodes each different demand. Noise and random events can be seen, as the green Node had to back off, after a miss detection. However this is just temporary as the network is self-healing.

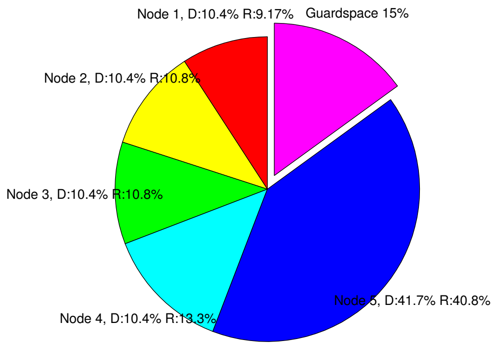


Fig. 12. Scheduling result of nodes with different demand after 100 firing rounds. Each node is demanding D% and actually receiving R%. The guardspace are free slots serving as an entry point for new nodes, and as a guard intervall between each node.

Therefore the nodes first assigned themselves the minimum number of CH and then started the scheduling and synchronization algorithm. We can see that the algorithm schedules very well limiting each node by the denser cluster (fig.15, fig.16). For the synchronization we can see that it is a lot slower that for the one cluster case presented above and also less accurate. This is because the time information needs to hop over several nodes each time addend an additional error. The hoping of the information also explains why the algorithm is slower as one cluster first has to organize himself and then slowly has to 'convince' the other cluster of the 'right' time (fig.17).

We can see that the algorithm also converges for 2

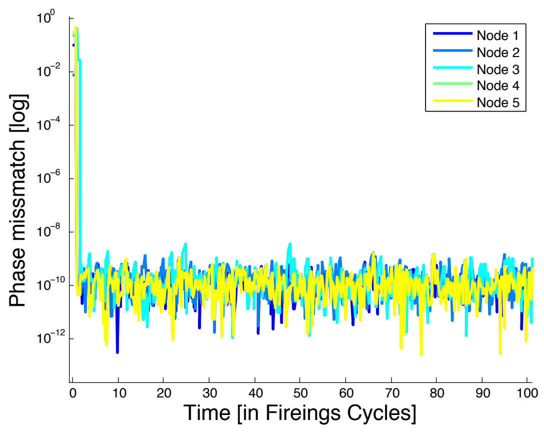


Fig. 13. Convergence of the syntonization algorithm. With respect to an arbitrary node chosen as reference.We can see that after 30 iterations the algorithm converges and we get an error of 1e-10s determined by the SNR of the transmissions

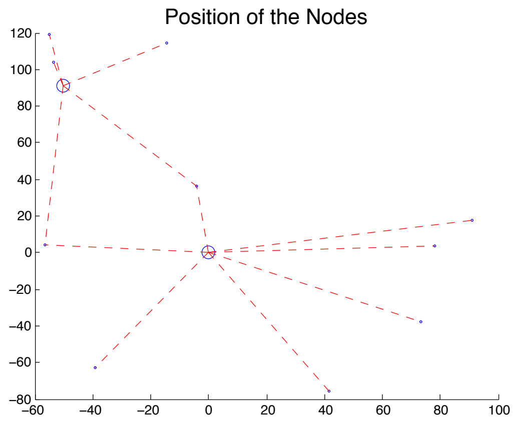


Fig. 14. Positions on the nodes randomly distributed. The circled nodes represent a cluster head.

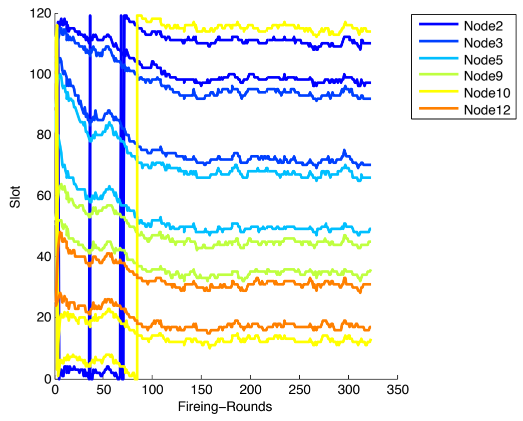


Fig. 15. Convergence of the scheduling algorithm of cluster one containing 6 out of the 12 nodes.

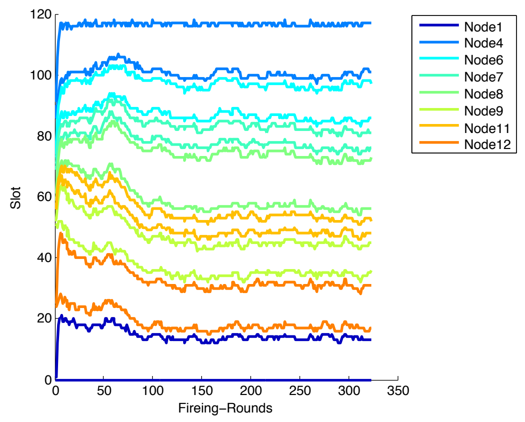


Fig. 16. Convergence of the scheduling algorithm of cluster one containing 8 out of the 12 nodes. Note that one node can be in more than one cluster.

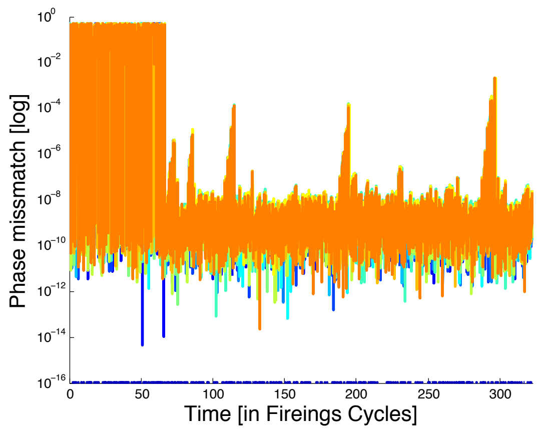


Fig. 17. Convergence of the syntonization algorithm. With respect to an arbitrary node chosen as reference.We can see that after 70 iterations the algorithm converges and we get an error of 1e-8s determined by the SNR of the transmissions

clusters and we will show in a future paper that this is also true for an arbitrary number of clusters.

References

[1] "Ieee standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. c1–269, July 2008.

[2] J. Larcom and H. Liu, "Modeling and characterization of gps spoofing," in *Proc. 2013 IEEE International Conference onTechnologies for Homeland Security (HST)*, Nov 2013, pp. 729–734.

[3] S. Ramanathan, "A unified framework and algorithm for (t/f/c)dma channel assignment in wireless networks," in *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 2, Apr 1997, pp. 900–907 vol.2.

[4] X. L. Huang and B. Bensaou, "On max-min fairness and scheduling in wireless ad-hoc networks: analytical framework and implementation," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*. ACM, 2001, pp. 221–231.

[5] C. D. Young, "Usap: a unifying dynamic distributed multichannel tdma slot assignment protocol," in *Military Communications Conference, 1996. MILCOM'96, Conference Proceedings, IEEE*, vol. 1. IEEE, 1996, pp. 235–239.

[6] L. C. Pond and V. O. Li, "A distributed time-slot assignment protocol for mobile multi-hop broadcast packet radio networks," in *Military Communications Conference, 1989. MILCOM'89. Conference Record. Bridging the Gap. Interoperability, Survivability, Security., 1989 IEEE*. IEEE, 1989, pp. 70–74.

[7] I. Rhee, A. Warrier, J. Min, and L. Xu, "Drand: distributed randomized tdma scheduling for wireless ad-hoc networks," in *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2006, pp. 190–201.

[8] T. Herman and S. Tixeuil, "A distributed tdma slot assignment algorithm for wireless sensor networks," *Algorithmic Aspects of Wireless Sensor Networks*, pp. 45–58, 2004.

[9] "Ieee std 802.11n, amendment 5: Enhancements for higher throughput," 2009.

[10] T. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[11] L. Pecora, F. Sorrentino, A. Hagerstroom, T. Murphy, and R. Roy, "Symmetries, cluster synchronization, and isolated desynchronization in complex networks," 2013.

[12] R. Pagliari, Y.-W. P. Hong, and A. Scaglione, "Bio-inspired algorithms for decentralized round-robin and proportional fair scheduling," *IEEE Journal on Selected Areas in Communications, Special Issue on Bio-Inspired Networking*, vol. 28, no. 4, 2010.

[13] A. S. Roberto Pagliari, Yao Win Peter Hong, "A low-complexity scheduling algorithm for proportional fairness in body area networks," *Bodynets*, 2009.

[14] R. Wannamaker, S. Lipshitz, J. Vanderkooy, and J. Wright, "A theory of nonsubtractive dither," *Signal Processing, IEEE Transactions on*, vol. 48, no. 2, pp. 499–516, 2000.

[15] T. Aysal, M. Coates, and M. Rabbat, "Distributed average consensus with dithered quantization," *IEEE Transactions on Signal Processing*, vol. 56, no. 10, pp. 4905–4918, October 2008.

[16] S. Ashkiani and A. Scaglione, "Discrete dithered desynchronization," *arXiv preprint arXiv:1210.2122*, 2012.

[17] S. M. Kay, *Fundamentals of Statistical Signal Processing*. Prentice Hall, 1993.