DEPARTMENT OF INFORMATION ENGINEERING
UNIVERSITY OF PADOVA

Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria dell'Automazione

corso di Programmazione di Sistemi di Controllo

Adaptive time synchronization for low energy consumption

Students:
Chiara Anselmi
Luca Magnabosco
Stefano Bizzo

Professor:
Luca Schenato

Anno Accademico 2012-2013

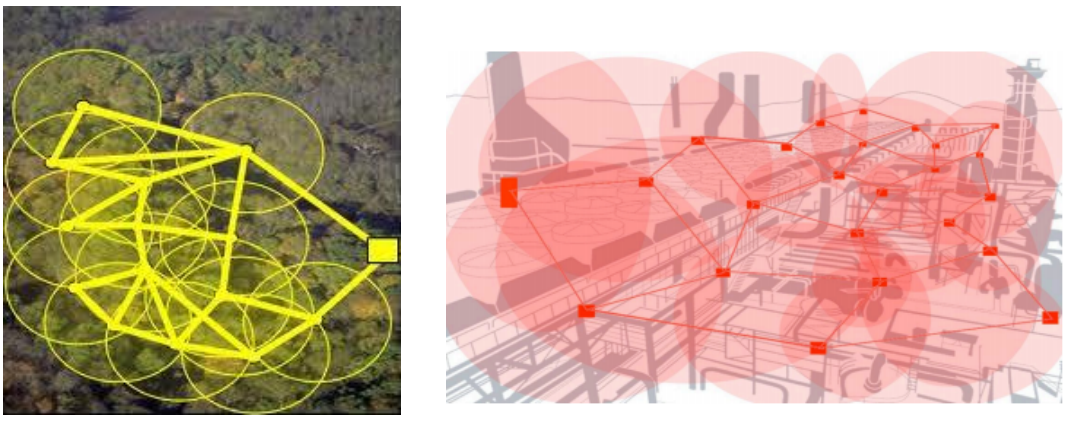
Contents

Chapter 1

Introduction

Through internet diffusion and the development of the miniaturisation of electronic components, it has been possible to interconnect a large number of tiny sensors with wireless connections. Therefore wireless sensors networks (WSN) have become more and more widespread. They permit to receive or monitor data which would be hardly managed using wired connections, even though communications bring on a considerable energy consumption. This technology is widely exploited in many applications: industrial machines control, environmental monitoring, agriculture, event detection, military applications.

For all these uses it is important to maintain all the devices of the network temporally synchronized, so that each node of the system has the same time reference of each other node. For example, if a network provides data from a plant, it is fundamental that these data refer to the same instant of time, even more this synchronism is essential in event detection, or mechanical systems.



(a) (b)

Figure 1.1: WSN application examples.

1.1 Background: problems and existing solutions

As already described, the use of WSN leads to face mainly two problems: synchronization and energy conservation. In the next paragraphs we will describe these problems and the existing solutions.

Figure 1.2: Some WSN devices.

1.1.1 Synchronization

The first problem to be solved, dealing with WSNs, is the synchronization of nodes internal clocks despite both the unknown time-varying frequency of each clock oscillator and the unknown topology of the network.

In literature various synchronization protocols have been proposed. In this work we have decided to consider the ATS and PI ones. As explained in the next chapter, these protocols are founded on a distributed approach characterized by the fact that each node runs the same algorithm and that the decisions made by the nodes are only taken on the basis of local information. For these reasons and according to the characteristics mentioned above, each node is a low power device integrating all the functions required to accomplish both the interconnection and its particular task: sensing, computing, wireless communication, storage ability. In addition, as these devices are not connected to electricity supply, the nodes of a WSN are battery powered. This characteristic leads us to the second central issue.

1.1.2 Energy conservation

As explained nodes are battery supplied. This permits a flexible installation and use (strong point of WSNs) such as the inaccessible collocation of the sensors or their possible hostile working environment or their huge number. Due to these factors, in many cases these devices are impossible or too expensive to get to, hence the impossibility for these devices of being either recharged or replaced makes the energy consumption a central issue. It is remarkable that in a typical sensor node the energy required to transmit a bit of information is approximately the same required to execute a thousand operations [1], so an efficient strategy to guarantee a massive energy conservation is to power-off the nodes (or wireless transceiver) when communication is not needed. In order to reach this goal it is necessary a sleep/wake schedule and, as a consequence, this requires the nodes synchronization. Some techniques are the following:

- fixed parameter strategy. In order to schedule the duty-cycle (as ratio between wakeup and

sleep periods) and maintain the possibility of communication, fixed parameters are used. Even though this strategy requires a simple method for synchronization, it needs the knowledge of the network topology, therefore it does not comply with a distributed approach. In addition, it is difficult to obtain an optimal performance because once the ratio between wakeup and sleep time has been fixed, it can't be modified;

- S-MAC (Sensor MAC) is implemented at the MAC layer. It defines adaptative duty-cycles coordinating the switching on and off through the use of sync data packets;
- T-MAC (Time-out MAC) defines a time-out period (a node goes to sleep after a period without events);
- other techniques use hierarchical network structures. For example in [1] the information is sent from the children to the parents during variable talk intervals (see fig. 1.3).

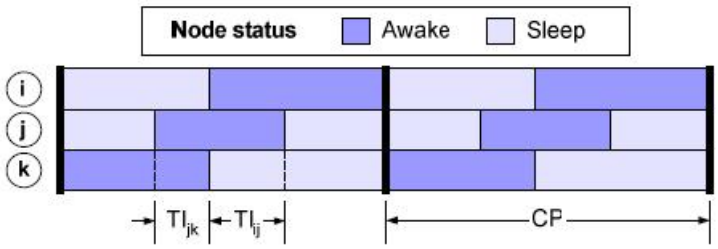


Figure 1.3: Communication periods representation in hierarchical structure.

1.2 Objectives

The aim of our work is to suggest different strategies to regulate the cycle-time in which a node can receive (ad not for example shifting them in the time, as in [1]), evaluating, on one side how the introduction of switching-off periods influences node's synchronization, and on the other side how much synchronization can favour sleeping periods.

The heart of the matter is quite evident: as mentioned before, in order to reduce the time lapse in which the nodes can communicate, it is important that all the nodes are sufficiently synchronized. However, to guarantee a robust synchronization, nodes have to be able to communicate, so they need a broad enough time window. Therefore the width of the communication periods and the synchronization of the nodes are contrasting aspects of the problem and they have to be managed in the best way from the energy consumption point of view. In any case it is fundamental that as time goes by, no nodes are excluded from future communications.For this reason in all the windows proposed a neighbours check is implemented: the window widens if no communications are recorded.

We have developed essentially two different strategies. Error based window and error proportional window consider the local synchronization error to reduce or enlarge the duration of switching on. The other two windows (received neighbours window and previous received neighbours window) instead work so that a node considers just the fact that it has received from all its neighbours or not.

The observations and the conclusions have been drawn through MATLAB simulations. In order

to simulate the behaviour of a WSN in the most realistic way as possible, we implemented the network communication taking into account the following aspects:

- more than one node can communicate at any instant;
- possible collisions during communications;
- transmission noise;
- including a new node in a synchronized WSN.

To estimate the performance of each strategy we have decided to evaluate the variance of the time estimates and the saving factor. The first one is indicative of the accuracy of the synchronism and the second one is an energy conservation index which depends on the ratio between the awake time of the nodes and the total time. In chapter 5 we will describe how these indexes vary depending both on the window and on the different conditions of simulations (communication noise, collisions, synchronization algorithm). In particular we will show that in most cases the best performance lies in a compromise between precision of synchronization and energy conservation.

1.3 Contents of the chapters

The structure of the work is divided into six chapters:

Chapter 2: it describes the communication protocol and the synchronization algorithms used in the simulations giving a brief mathematical background

Chapter 3: it explains how the sizing policies of the communication periods work. We propose here three different adaptative window algorithms;

Chapter 4: it illustrates the MATLAB implementation ideas to describe the network, the adaptative windows and the behaviour of the whole system

Chapter 5: in this chapter we show the simulations carried out, and we describe the different situations chosen to test the implemented algorithms

Chapter 6: we describe the conclusions of our work, referring to the simulation tests and hinting at the possible developments

Chapter 2

Consensus synchronization

As explained in the introduction, the aim of this work is to synchronize a net of digital clocks, so that they will all have the same value after a certain period. In this perspective, a synchronisation problem can be considered as a consensus problem, which allows us to use all the techniques developed so far in the latter field. For a detailed discussion on this subject, we refer to the existing reference. This chapter is rather conceived to illustrate some mathematical tools used later in this report.

Let us consider a set of agents, able to communicate to each other, characterised by some variables associated to the agents themselves. The aim of consensus is to guarantee that these variables converge to the same value.

At the basis of the consensus theory, we find the theorem of Perron Frobenius, which states that if a matrix A of R^{nxn} is strictly positive, then a vector v and an eigenvalue λ_0, both positive, exist so that:

1. v is an eigenvector of A
2. $\lambda_0 = 1$ and the other eigenvalues of A are inside the circle of unitary radius.

This means that there is a dominant eigenvalue, situated on the circle of unitary radius.

It can be shown that stochastic matrices satisfy the conditions of the previous theorem. Moreover if a stochastic matrix P is the characteristic matrix of a discrete system such as:

$$x(k+1) = Px(k) \tag{2.1}$$

all the components of $x(k)$ converge to a common value which is a linear combination of the initial conditions for k approaching to infinite. In other words:

$$\lim_{k\to\infty} x(k) = \alpha * 11 \tag{2.2}$$

where 11 is the vector $[11\ldots1]^T$ and α a linear combination of the initial conditions.

If, instead of a stochastic matrix, we use a bistochastic matrix the value to which $x(k)$ converges is the mean of the initial conditions.

A net of devices can always be represented by a graph, whose nodes contain the information which needs to be shared in order to reach consensus. From the graphs theory we know that it is always possible to associate to each graph a matrix; in particular some methods can be used to obtain a stochastic or a bistochastic matrix associated to the graph. Therefore both tools from graph theory and consensus theory can be applied to describe and solve the synchronisation problem we are interested in.

In the next sections we will briefly present the communications protocol and the consensus algorithms that we have used, focussing mainly into their asynchronous implementation.

2.1 Communication protocol

Since our aim is to analyse a network of devices communicating their status to each other, it is important to develop fitting communication protocols. Many examples can be found in literature. For this work, we have chosen to use the asynchronous broadcast protocol, where the term asynchronous means that the nodes communicate in instants of time, which are not pre-established. According to this protocol, at every instant an active node communicates its status to his neighbours and the active neighbours update themselves, with reference to the prefixed weight q. The transmitting node maintains its value, since a node can either transmit or receive and update; obviously if a node does not receive any information, it maintains its previous status, without updating.

The broadcast matrix $P_{broadcast}$ associated to the graph describing the network determines the evolution of the system. Let's call i the current transmitting node, $q \in (0, 1]$ the weight, e_k the vector of the canonical basis with 1 on the k-th position and Ni_{out} the set of neighbours of i which can receive from i. The broadcast matrix $P_{broadcast\,i}$ can be written as:

$$P_{broadcast\,i} = P_i = I + q \sum_{j \in Ni_{out}} e_j (e_i - e_j)^T \tag{2.3}$$

Note that every time there is a communication, a new broadcast matrix is built to update the system, and this means that these matrices are depending both on time and on the node that is communicating its status to its neighbours.

2.1.1 A Broadcast communication example

An example can clarify what has been said so far. Consider the following graph (see fig. 2.1):

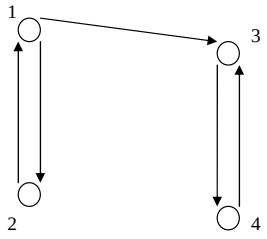


Figure 2.1: Example graph.

Suppose that the node 1 transmits its value to its neighbours and that the updating equation is eq. 2.1. In this way the broadcast matrix would be:

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ q & 1-q & 0 & 0 \\ q & 0 & 1-q & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

If node 2 transmits after node 1 has transmitted, the broadcast matrix becomes:

$$P_2 = \begin{bmatrix} 1-q & q & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

2.2 ATS

Average Time Sink (ATS) is a fully distributed, consensus based protocol, for the synchronization of wireless sensors, which guarantees both skew and offset compensation [4]. Since it is fully distributed, all nodes follow the same algorithm and there are no special nodes, such as roots, in the graph which is associated to the sensors network.

In our work this algorithm has been implemented in its asynchronous form using the Broadcast protocol for communications purposes. We have chosen this implementation, as in our analysis it would be unrealistic to suppose that all the agents communicate with all the other agents at the same time.

Since all the protocol is described in Schenato-Gamba paper [4], we will just summarize the results. Every node in the graph can be modelled by the following expression:

$$\tau_i = \alpha_i t + \beta_i \tag{2.6}$$

where τ_i is the local clock, α_i the local skew and β_i the local offset. The aim of the algorithm is to synchronize all the clocks to the same virtual clock

$$\tau_v = \alpha_v t + \beta_v \tag{2.7}$$

letting the nodes share the estimation of their own clock:

$$\widehat{\tau}_i = \widehat{\alpha}_i t + \widehat{o}_i \tag{2.8}$$

Since every node has a different skew and offset estimation, to achieve consensus all clocks have to compensate them, in order to reach a common value for both. For the offset estimation, the equation is given by:

$$\widehat{o}_i(t+1) = \widehat{o}_i(t) + (1-\rho_o)(\widehat{\tau}_j - \widehat{\tau}_i) \tag{2.9}$$

This is the update equation that is used in the algorithm to which we refer to as *ATS_O*, which provides only the offset compensation. The updating equation for the skew estimation is:

$$\widehat{\alpha}_i(t+1) = \rho_v \widehat{\alpha}_i(t) + (1-\rho_v)\eta_{ij}\widehat{\alpha}_j(t) \tag{2.10}$$

where η_{ij} is calculated as:

$$\eta_{ij}(t+1) = \rho_\eta \eta_{ij}(t) + (1-\rho_\eta)\frac{\tau_j^{new} - \tau_j^{old}}{\tau_{ij}^{new} - \tau_{ij}^{old}} \tag{2.11}$$

and represent the ratio between α_i and α_j, in fact $\lim_{t\to\infty} \eta_{ij}(t) = \frac{\alpha_i}{\alpha_j}$.

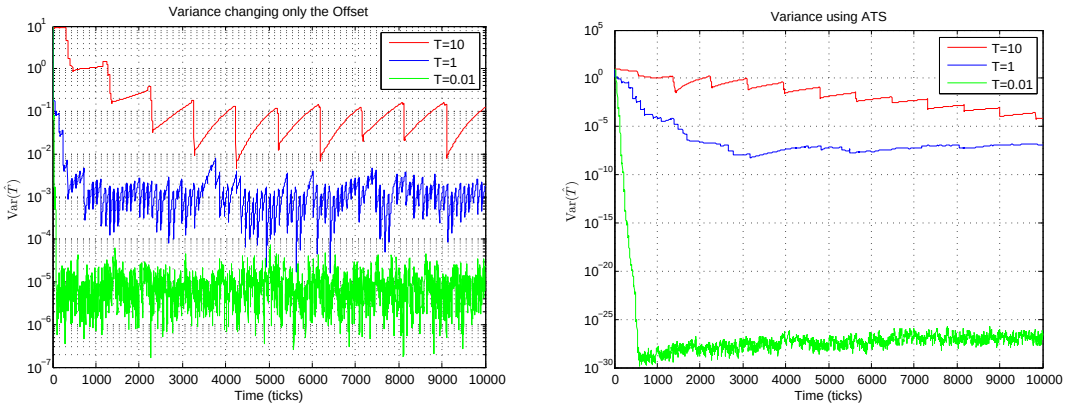
It can be demonstrated that, for these expressions of skew and offset estimations, the time estimations of the nodes of the graph approach the same value for all the clocks of the network, that is:

$$\lim_{t\to\infty} \widehat{\tau}_i = \widehat{\tau}_i, \quad \forall(i,j) \tag{2.12}$$

It can be observed that the weights ρ_o, ρ_v and ρ_η have a meaning similar to the weight q of the Broadcast matrix. For the asynchronous implementation in a WSN with N nodes we can imagine ρ_o, ρ_v and ρ_η as time-varying $N \times 1$ vectors whose cells can take only two values: 1 or $0 < \rho < 1$.

2.2.1 ATS and ATS_O comparison

In this paragraph we show the performance of the ATS and of the ATS_O, which, as said before, changes only the offset. In the following simulations the error variance is calculated without the influence of the noise and considering α constant for every node. Figure 2.2 shows the difference in terms of performance between both the asynchronous implementations. Moreover in this figure can be noticed that the different trends of the variance depend on the chosen communication period.



(a) $\widehat{\tau}$ *variance changing only the Offset.* (b) $\widehat{\tau}$ *variance using ATS.*

Figure 2.2: Comparison between ATS and Only Offset changing the communication period

2.3 PI

The PI Consensus algorithm is a distributed clock synchronization protocol for networks of clocks, which have different initial offsets and internal clock speeds. The algorithm is based on an PI-like consensus protocol where the proportional (P) part compensates the different clock speeds, while the integral part (I) eliminates the different clock offsets. This synchronization protocol has been

presented in its synchronous implementation in the paper "A PI Consensus Controller for Networked Clocks Synchronization"[3], where, if the communication graph is known, convergence is guaranteed and optimal design is studied, using optimisation tools.

However in our case, as explained before, the synchronous implementation is unrealistic, therefore we have based our implementation on the paper "A PI consensus controller with gossip communication for clock synchronization in wireless sensors networks"[2] , with the difference that our implementation is based on Broadcast communication.

If the nodes can not communicate, then the local time $x_i(t)$ can be modelled as a discrete time integrator

$$x_i(t+1) = x_i(t) + d_i \tag{2.13}$$

Clearly, different initial offset, $x_i(0)$, and different speeds d_i cause synchronization errors. To solve the problem in the article[3] they assume that it is possible to control each clock by a local input $u_i(t)$ as follows

$$x_i(t+1) = x_i(t) + d_i + u_i \tag{2.14}$$

where to make the algorithm distributed, the control action u_i is only allowed to use local information.

Moreover each node has in memory, beside the estimate $x_i(t)$, another variable denoted by $w_i(t)$ that plays an important role in compensating the different speeds d_i.

In our case we suppose that for $i \in 1, \ldots, N$ and for $k \in N$, the information $x_i(T(k))$ is sent by node i to all its active neighbours, which are selected within the set Ni taking into account which ones are turned on[1].

Now, without loss of generality, let's focus on the communication between the node i and the node j, where the node i transmits only to one of its neighbours that is j. Therefore at time $T(k)$[2] the node i transmits the information $x_i(T(k))$ to node j.

Then the updating equations for x and w are

$$x_j(T(k+1)) = \frac{1}{2}\left(x_j(T(k)) + x_i(T(k))\right) + w_j(T(k)) + d_j(T(k)) \tag{2.15}$$

$$x_h(T(k+1)) = x_h(T(k)) + w_h(T(k)) + d_h(T(k)), \quad \forall h \neq i \tag{2.16}$$

and

$$w_j(T(k+1)) = \frac{\alpha}{2}\left(-x_j(T(k)) + x_i(T(k))\right) + w_j(T(k)) \tag{2.17}$$

$$w_h(T(k+1)) = w_h(T(k)), \quad \forall h \neq i \tag{2.18}$$

Defining the vector $\overline{x}_j = \begin{bmatrix} x_j(T(k+1)) \\ w_j(T(k+1)) \end{bmatrix} = \begin{bmatrix} \overline{x}'_j \\ \overline{x}''_j \end{bmatrix}$ to identify the control action u_j these equations can be expressed as

$$\overline{x}_j(T(k+1)) = \begin{bmatrix} 1 & \Delta\alpha_j(T(k)) \\ 0 & 1 \end{bmatrix} \left[\overline{x}_j(T(k)) - u_j(T(k))\right] \tag{2.19}$$

[1] It depends from the communication window.

[2] This representation of the time is used to indicate that we use discrete time with granularity $T(k) - T(k-1)$. Anyway this time T(k) is the general time that is unknown to the nodes. For example node i at the time $T(k)$ has a time estimation $\widehat{\tau}_i(k)$

where $u_j(T(k))$ has been previously calculated as

$$u_j(T(k)) = \begin{bmatrix} u'_j(T(k)) \\ u''_j(T(k)) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ \alpha_{PI} \end{bmatrix} (\overline{x}'_i(T(k)) - \overline{x}'_j(T(k))) \tag{2.20}$$

Chapter 3

Adaptative window

As explained before, our goal is to reduce power consumption by limiting the width of the receiving window of every agent. In this chapter we are going to describe the three algorithms we have implemented in order to obtain a sort of adaptative window width.

Before the algorithms some variables, to size and to represent the receiving window, are presented.

3.1 Variables used for the windows algorithms

3.1.1 Introduced variables

In order to represent the receiving window size and to evaluate the saved power some variables are introduced:

- $\boldsymbol{T}$: the communication period of each node.
- $\boldsymbol{\delta}$: used to represent the window semi-width while the window center is a multiple of the synchronisation period T (see fig. 3.1).
- $\boldsymbol{\delta_{max}}$: δ maximum value calculated as $\delta_{max} = T/2$.
- $\boldsymbol{\delta_{min}}$: δ minimum value that depends from the time quantization.
- **SF**: it represents the *power saving factor* due to the use of the receiving windows. It is calculated as $(1 - \frac{(Agents\_active\_time}{(Number\_of\_agents * time)}) * 100$.

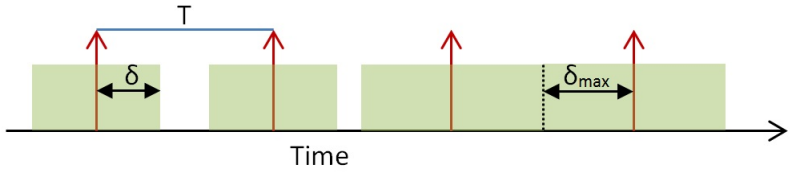


Figure 3.1: Receiving window representation.

In order to control the memory width with the adaptive windows, other variables are introduced:

- **neighbours**: information inserted by the user that represents, for each node, how many neighbours it has[1].
- **ID**: It is a number that is used to uniquely identify each node in the WSN. In our case we use the node number, but the idea is to use a sort of ID that is unique for each agent[2]. This data is sent to the neighbours within the estimated time every time that a node transmits.
- **NRIW**(Neighbours received in last window): At every instant, each active node evaluates how many *different neighbours* it has received in the *current window*. This information could be obtained by the node considering how many different ID it has received in the current window.
- **NRIW_old**: the NRIW value referred to the previous receiving window.
- $\widehat{e}$ (Estimated synchronization error): This is not the real consensus error just because each agent can not know this information that requires the knowledge of the $\widehat{\tau}$ of each agent. It is intended to "estimate" the consensus error just using local data that every agent has. It is calculated by the receiving node as $\widehat{e}_j(t) = \widehat{\tau}_i(t) - \widehat{\tau}_i(t)$, where i is the transmitting node while j is the receiving one[3].
- $\widehat{e}_{max}$: it is defined as the maximum $\widehat{e}$ received in the last window.

3.1.2 Nodes memory

To control the width of the receiving window, each node needs to know some information about the last communications. Therefore we have decided to equip each node with a certain amount of memory to save various data along the run time.

Obviously this memory is used in our implementation like a normal variable, as implemented in section 4.3. The stored variables are, for each node and for each instant of time:

1. *estimated synchronization error* $\widehat{e}$
2. *ID* of the received node
3. a label that is set to 1 only if the communication, containing $\widehat{e}$ and ID, has been performed in the current receiving window

3.2 Currently Received Neighbours Window (CRNW)

This window is based on the simple idea that, in an ideal network without nodes failures and without communication collisions, the WSN will continue working as if there is not a communication window, if every node receives all its neighbours in every communication window.

The receiving window starts δ_{min} ticks[4] before the communication time and it is closed only if

[1] The issues related to the use of this information about the real neighbours will be commented in the next paragraphs.

[2] Eventually node ID can be randomly generated by the same node.

[3] It is by using the MAC-layer [5] time-stamping, available in many sensor network devices, that the reading of the local clock at the receiving node can be assumed instantaneous

[4] One tick is the minimum time between one execution of the algorithm in the node and the next one.

all the neighbours are received in the window. An example of this window algorithm working in an ideal case is showed in fig. 3.2.

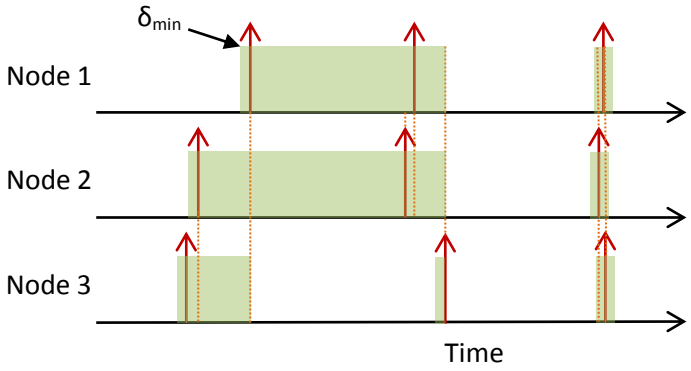


Figure 3.2: Currently Received Neighbours Window algorithm example.

As it is clear, with this window every node needs to know from the beginning how many neighbours it has. Moreover, the window is not centered with respect to the communication time and this fact can cause the loss of a lot of communications, which occur just before the instant $T(k) - \delta_{min}$. To solve this problem another algorithm to size the window is proposed.

3.3 Previously Received Neighbours Window (PRNW)

Another way of taking into account the communications received from the neighbours is to consider the number of neighbours received in the previous window and change the window width according to the fact that the node has received all its neighbours or not. In this way the receiving window is opened δ ticks before the communication moment, leading to a centered receiving windows which brings a faster and more robust synchronization (see fig. 3.3). Window width is sized according this algorithm:

- if in the last window no neighbour has been received next window width is set to δ_{max}.
- else window width is sized in this way[5]:
  - if $NRIW\_old_j(t) \geq neighbours_j$ $\delta(j) = \delta(j) - \delta_{min}$[6]
  - if $NRIW\_old_j(t) < neighbours_j$ $\delta(j) = \delta(j) + \delta_{min}$
- if $\delta(j) > \delta_{max}$ $\quad \delta(j) = \delta_{max}$
- if $\delta(j) < \delta_{min}$ $\quad \delta(j) = \delta_{min}$

As it is evident with this window every node needs to know from the beginning the number of its neighbours. Moreover, the algorithm is not so robust because in real networks nodes and communication failures may occur, causing a node not to receive one or more neighbours for a considerable amount of time.

[5] The algorithm is described for the node j.

[6] As this operation is performed for every instant that the node is active the decreasing action is proportional to the current window size.

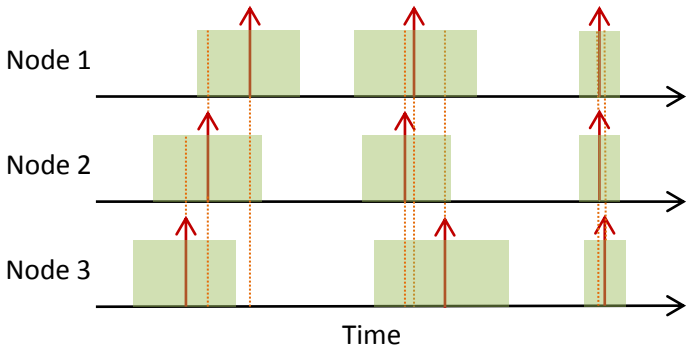


Figure 3.3: Previously Received Neighbours Window algorithm example.

Possible evolutions of this kind of window can be implemented in order to let each node estimate its neighbours. So, each node has to make this estimation when it is turned on and during the running time when it stops receiving a neighbour for more than a certain number of periods. The estimation could be done fully opening the window for a certain number of periods and considering as actual neighbours only the neighbours received in these periods. These considerations would make the adaptive window algorithm more robust, but because of the nature of the problems they pretend to solve, their advantages are difficult to be analysed with virtual simulations.

3.4 Error Based Window (EBW)

This window is based directly on the evaluation of what has been called the *estimated synchronisation error* $\widehat{e}$. We desire that this error decreases till it reaches an acceptable value, called $\widehat{e}_{accepted}$.

In every instant the maximum between the errors received in the receiving window, named $\widehat{e}_{max}$, is evaluated and if the node is active next window width is changed in this way:

- if the node estimated time $\widehat{\tau}_j \leq 3T$ next window width is set to δ_{max}.
- if in the previous window no neighbour has been received next window width is set to $\delta(t) = 2\,\delta(t-1)$.
- else when a new neighbour is received, window width is sized in this way:
  - if $\widehat{e}_{max}(t) \leq \widehat{e}_{max}(t-1)$ or $\widehat{e}_{max}(t) \leq \widehat{e}_{accepted}$ next window width is set as $\delta(t) = \delta(t-1) - \delta(t-1)/5$.
  - if $\widehat{e}_{max}(t) > \widehat{e}_{max}(t-1)$ or $\widehat{e}_{max}(t) > \widehat{e}_{accepted}$ next window width is set as $\delta(t) = \delta(t-1) + \delta(t-1)/5$.
- if $\delta(j) > \delta_{max} \quad \delta(j) = \delta_{max}$
- if $\delta(j) < \delta_{min} \quad \delta(j) = \delta_{min}$

The algorithm is performed by each node independently.

This algorithm seems to be more robust to node failures than the previous one. Moreover it does not need the information about the number of neighbours[7]. Anyway, the choice of using only

[7]This algorithm takes only care of receiving at least from one neighbours in every window and it has at least a neighbour because the graph has to be connected.

the data received in the last window could be too less conservative and in a WSN, where a large number of collisions occur, it may be preferred to use all the data that the memory described in section 3.1.2 can provide. This can be easily verified calculating $\widehat{e}_{max}$ using all the $\widehat{e}$ in memory and not only the $\widehat{e}$ referred to the current window.

3.5 Error Proportional Window (EPW)

This window algorithm is another way of using the information that nodes have about the *estimated synchronisation error* $\widehat{e}$.

To receive a neighbour every time it communicates, the receiving window needs a semi-width large at least as the real synchronization error between the two nodes[8]. Therefore we have decided to change the semi-width δ proportionally to the *estimated synchronisation error* $\widehat{e}$. The algorithm can be described in few steps:

- if the node estimated time $\widehat{\tau}_j \leq 3T$ next window width is set to δ_{max}.
- if in the previous window no neighbour has been received next window width is set to $\delta(t) = 2\,\delta(t-1)$.
- else window width is sized for the node j as: $\delta(j) = K\,\widehat{e}_{max}(j)$[9].
- if $\delta(j) > \delta_{max} \quad \delta(j) = \delta_{max}$
- if $\delta(j) < \delta_{min} \quad \delta(j) = \delta_{min}$

[8] Not considering non-idealities such as communication noise and collisions.
[9] K is the estimated synchronisation error proportional coefficient

Chapter 4

Matlab implementation

The code we have implemented in Matlab consists in a main file which calls sequentially various functions inside a `For` loop. Each iteration of this loop represents the minimum quantum of time in which every transmitting node communicates its estimated time and ID, while we evaluate communication collisions.

To decide how data flow in the WSN, we use a broadcast matrix for every node that is in its transmitting instant. These broadcast matrices are created considering not only the adjacency matrix of the graph, but also which nodes are active.

So, the synchronization algorithms use the broadcast matrices to decide which node can update its $\widehat{\tau}$ using the $\widehat{\tau}$ transmitted from another node.

4.1 Program flow

Outside the loop, the program performs some initializations. The graph and its adjacency matrix are randomly generated, as well as the coefficients α end β (see eq. (2.6)) that are used for the clock model.

The operations that the program performs during a loop iteration are:

1. The α coefficient for every agent is changed according to the variable *value_vel*, in order to execute a random walk that simulates clocks, varying their period along the run-time.
2. The noise that simulates quantization and errors during the data transmission is generated according to the variable σ_n.
3. The chosen synchronisation algorithm is performed using the estimated time of the previous instant $\widehat{\tau}(t-1)$. It is important to underline that the various algorithms use the $\widehat{\tau}(t-1)$ only of the nodes that have been received in the previous instant. As described in 4.2 it is performed evaluating the various broadcast matrices referred to the instant $t-1$.
4. The chosen adaptive window algorithm changes the semi-width of the receiving windows δ.
5. The function called `agents_communication_control`, depending from the current $\widehat{\tau}$, the chosen communication period T and the window semi-width δ, changes the adjacency matrix

in order to simulate that a node is inside or outside its communication window. Moreover this function detects which nodes are in their communication moment and calculates NRIW.

6. Depending on the fact if we decide to perform a simulation considering communications collision or not, the functions *real_broadcast* or *ideal_broadcast* are called.[1] These functions create a broadcast matrix for every transmitting node, taking into account the modified adjacency matrix.
7. Each node checks its memory and calculates how many different neighbours it has received in the current window and in the current moment.
8. Depending on the user choice, the program can plot in every moment which the active nodes are and which nodes have communicated with other nodes. This has been useful to debug the program: a node, in fact, can update its $\widehat{\tau}$ only when it has received the $\widehat{\tau}$ sent by another node while it is active. In section 4.4 it is explained how this procedure is performed.

4.2 Broadcast matrices evaluation

As described before, the synchronization algorithms need to know which node has communicated with another node, and this is performed evaluating more than a broadcast matrix in every quantum of time.

In the following paragraph we will explain how the ATS algorithm has been implemented in its asynchronous form. The PI has been implemented in a similar way.

4.2.1 Simplified WSN case

Let's focus for example on a WSN where *only the node* i can transmit in the instant of time t, while all the j neighbour nodes can receive. We can easily model the communication with the Broadcast protocol, observing that $\rho_{\eta_j}(t) \neq 1$ only if $P_i(j,i)(t) \neq 0$ and if $\exists\ t_{old}$, $P_i(t_{old})(j,i) \neq 0$, with $i \neq j$,, while $\rho_{o_j}(t) \neq 1$ and $\rho_{v_j}(t) \neq 1$ only if $P_i(j,i)(t) \neq 0$, with $i \neq j$. Therefore, we can describe how data flow, evaluating the cells of the matrix $P_i(t)$.

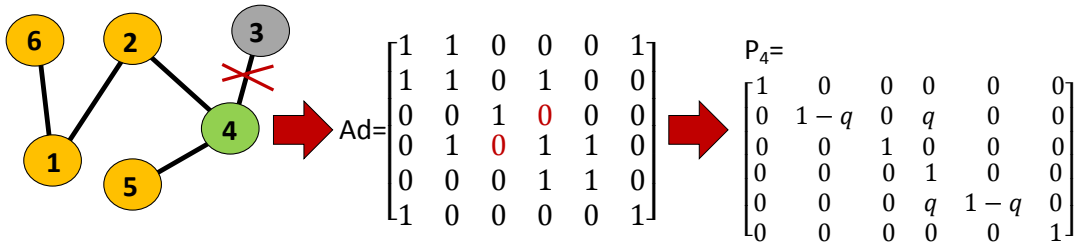


Figure 4.1: Ideal WSN with only node 4 transmitting but with node 3 switched off.

In the previous image (fig. 4.1) it can be seen how we evaluate communications in the graph, considering that only the node 4 can transmit. In this example all the nodes apart from node 3 are

[1] The way we evaluate collisions will be described in the next sections.

in their receiving windows[2], therefore all nodes except from node 3 can receive, so we can change the adjacency matrix accordingly. So, using the modified adjacency matrix, the broadcast matrix is automatically generated and it is employed, by the synchronization algorithms, only to detect which node has received from another node.

4.2.2 Real WSN case

To implement our program we have taken into account the possibility that more than a node can transmit in the same instant. Therefore, for each instant of time t we generate a P_i for every node i which can transmit, using than the matrices in sequence.

It can happen that a sequence of matrices P_i, executed at the same time, causes a certain number of collisions. A collision occurs when a node, which is transmitting, is also receiving a communication from another node, or when a node is receiving from more then a node at the same time.

We have decided to manage these collisions implementing two different function inside the program:

ideal case: Collisions are not considered. It is supposed to be possible that the nodes can transmit and receive information to and from more than a neighbour at the same time[3].

real case: The collision causes the node to fail the reception.

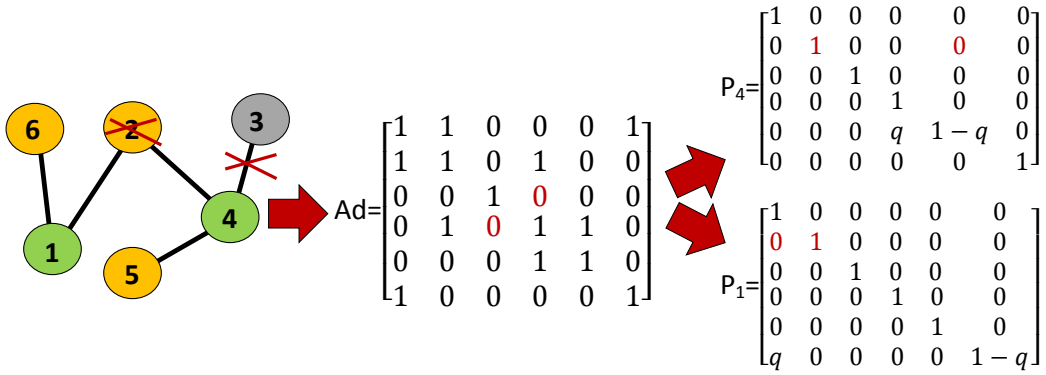


Figure 4.2: Real WSN with more than a node transmitting and node 3 switched off.

In figure 4.2 it can be seen how the program detects the nodes communications, considering that more than a node can transmit, and that collisions cause communications failure. In this example all the nodes apart from node 3 are in they receiving windows, therefore all the nodes apart from node 3 can receive. This fact is detected as described before and causes adjacency matrix to change. So, using the modified adjacency matrix, the broadcast matrices are automatically generated for each node which is in its transmitting instant and collisions are identified, evaluating the same cell for all the broadcast matrices. If the cell is not in the main diagonal and if it is different from 0

[2]This evaluation is made noticing that $|\widehat{\tau}_3(t) - T(k_3)| > \delta_3(t)$ while for the others j nodes $|\widehat{\tau}_j(t) - T(k_j)| < \delta_j(t)$. Only node 4 is in its communication instant because $|\widehat{\tau}_4(t) - T(k_4)| < \delta_{min}/2$.

[3]In this case the order of evaluation of the P_i matrices is important, therefore we have decided to randomize the evaluation order of the matrices that are referred to the same instant of time.

in more than a broadcast matrix, it is set to 0 in all the broadcast matrices, referred to the same instant.

4.3 Memory layout

In the next lines we present the "logical structure" of the variable which represents the node memory, as it will be useful to understand how the recorded data are used. This memory is used only to size the semi-width δ, depending on the chosen adaptative algorithm. In particular, it is using the labels 0 or 1, saved in the third level of every cell, that the variables $NRIW$ and $\widehat{e}_{max}$ are calculated.

We have projected a memory which has the structure of a three dimensional vector, for every node, as it can be seen in fig. 4.3. The length, which represents how many past communications are saved, can be decided by the user[4], while every cell, that refers to a single communication event, can be imagined organized into three levels:

1. The first level contains the *estimated synchronization error* $\widehat{e}$
2. The second level contains the *ID*
3. A label is set to *1* if this instant is a part of the current window for this node. The entire labels vector is set to 0 at the end of the window.

The pointer is moved from a cell to the next one, if a neighbour is received in this instant, or if the node was receiving but no neighbours has been received. When the pointer reaches the end of the vector, it starts again from the beginning, overwriting old data.

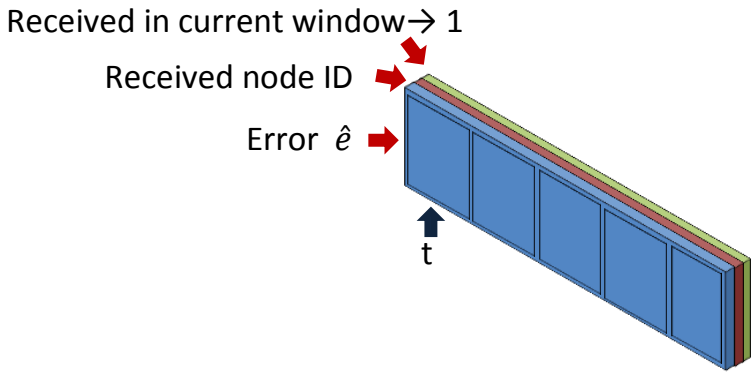


Figure 4.3: Memory scheme for one node. Every "3D" cell contains data referred to a communication received event.

[4]Generally it has been observed that the memory size has to be at least of the same order of magnitude of the number of neighbours that a node has.

4.4 WSN evolution check

In this section we will explain how the image, that represents the WSN evolution, is created in order to check if the program works right. The right part simply presents $\widehat{\tau}$ evolution where a "change" can obviously occur only if a node receives a communication. The left part is used to check if the "change" occurs only when the node has really received data from another node. This second check is independent from the $\widehat{\tau}$ and is based on the evaluation of the various broadcast matrices, referred to the same instant of time.

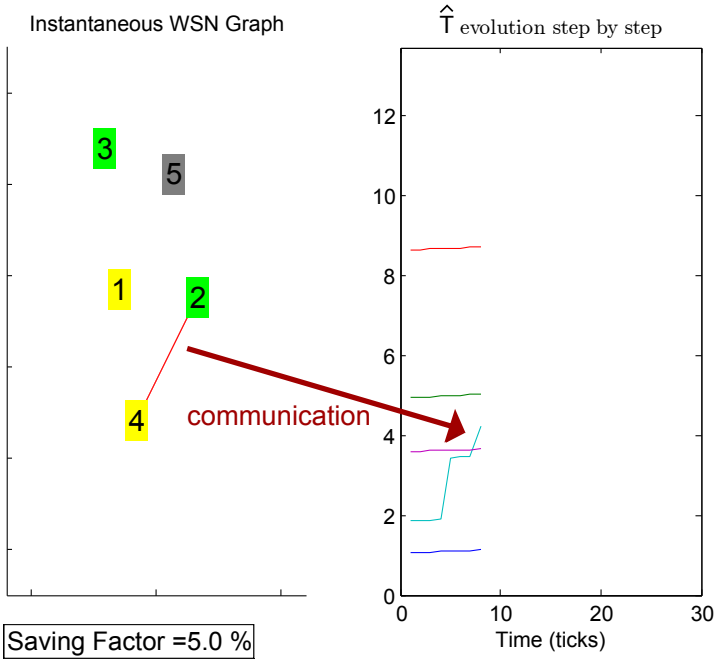


Figure 4.4: This figure is a frame produced by the function *plot_WSN*. It represents the evolution of $\widehat{\tau}$ for every node and plots the graph step by step showing active nodes in yellow and communicating nodes in green. The data flow between two nodes is represented with a red line.

Chapter 5

Simulations

In this chapter, which is the core of our work, we will analyse the behaviour of the algorithms using the four windows projected, i.e. the EPW (Error Proportional Window), the EBW (Error Based Window), the PRNW (Previous Received Neighbours Window) and the CRNW (Current Received Neighbours Window).

At the beginning we will consider ATS_O, which is ATS with only offset compensation, full ATS, and PI in the most ideal case, i.e. transmissions with no noise and no collisions; then we will introduce in the simulations noise and collisions, progressively. We have decided to work in this way in order to understand which is the most significant factor causing the performances to worsen.

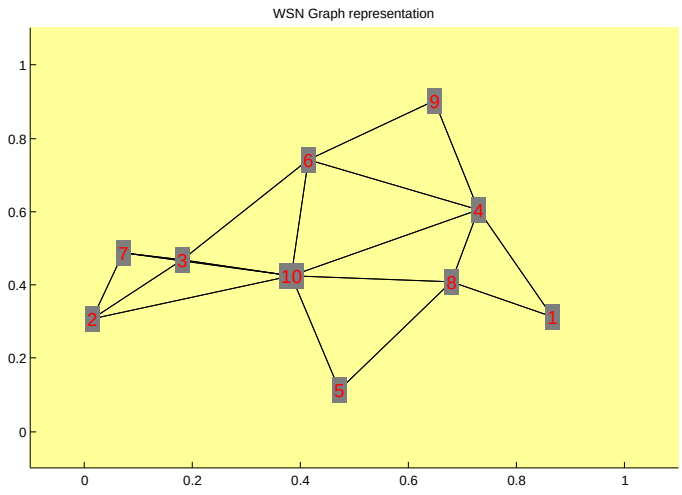


Figure 5.1: WSN graph used for the simulations.

The variables of interest in this contest are the variance of the time estimation, and the saving factor SF, which is useful to show if the chosen configuration allows energy conservation. The variance is expressed in logarithmic scale, the SF in per cent. The communication period is set to 10 seconds[1], while the time is expressed in ticks and one tick corresponds to 0.01 second s. Therefore

[1]This parameter has a great influence on the others and on the resulting saving factor. Its choice has been done to simulate as really as possible a real WSN.

a 300000 ticks simulation corresponds to 50 real minutes, with approximately 300 communications for each node.

The variables used for the ATS and the PI are:

- $\rho = 0.5$ that from previous work results to be a good value for both transient response and noise resistance
- $\alpha_{PI} = 0.01$

The variables used for the windows algorithms:

- $K = 10$ used for the EPW window.
- $\widehat{e}_{accepted} = 0.2$ used for the EBW window.

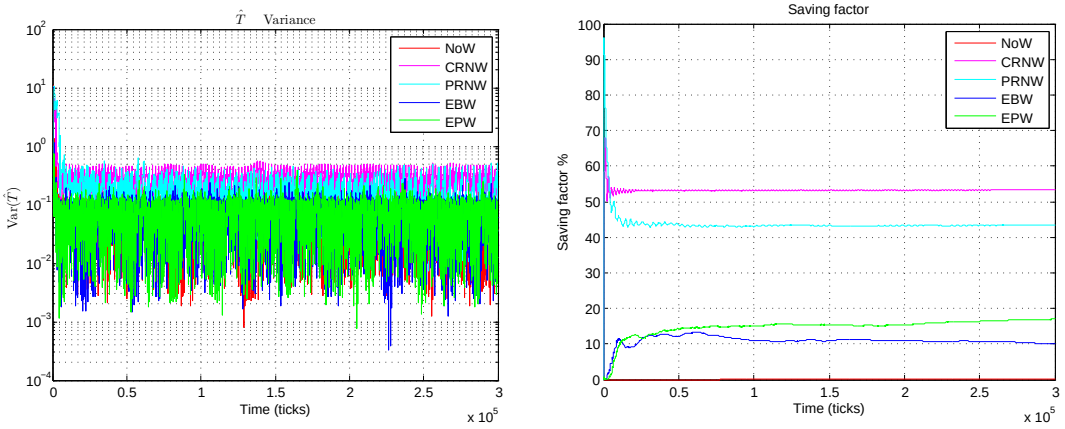
The WSN graph we have decided to use for the following simulations can be seen in fig. 5.1.

5.1 Case 1: No noise and No collisions

In this first paragraph, we will analyse the behaviour of ATS_O, ATS and PI in the ideal case with no noise and no collisions, testing the four windows described in the previous chapter.

5.1.1 ATS_O

Figure 5.2 shows the time estimation variance and the saving factor[2] obtained with the four windows and in the case with no window.



(a) $\widehat{\tau}$ *variance.* (b) *Saving factor (SF).*

Figure 5.2: $\widehat{\tau}$ variance and saving factor obtained with the algorithm ATS_O for different windows.

As it can be seen, the variance is really similar in the cases NoW, EBW and EPW, while for

[2] Note that the SF corresponding to the case with no window is always 0, because the nodes never turn off, so there is no energy conservation.

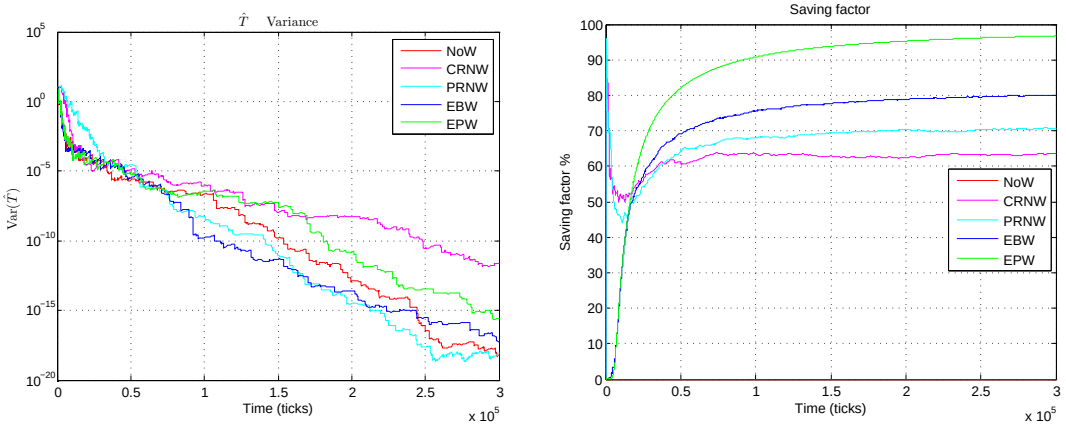
the cases CRNW and PRNW it can be noticed that the evolution is slower[3] and the steady state value is bigger.

The reason why the variance continues to oscillate between the maximum and the minimum values is the structure of the ATS_O itself. This algorithm, in fact, has no skew compensation, so clocks having different skews can never maintain their synchronisation. Eventually they will desynchronize, as they run with different speeds.

The analysis of the SF shows that the window which guarantees the best performances is the CRNW, with a SF of nearly 55%. PRNW allows to reach a value for the SF that is around 40%. The other two windows, i.e. EPW and EBW, do not attain great results: in steady state their saving factors are respectively around 20% and 10%. Because of the ATS_O structure in fact, the error can not really be reduced, as it will encrease again, once the nodes loose their synchronisation, for the lack of skew compensation.

5.1.2 ATS

The ATS algorithm is implemented to provide both the offset compensation, as the ATS_O, and the skew compensation. This should make the time estimation variance decrease all along the run-time, if no noise and no external factors, such as collisions, interfere. This behaviour is confirmed by simulations, as it can be seen in fig. 5.3.



(a) $\widehat{\tau}$ *variance.* (b) *Saving factor (SF).*

Figure 5.3: $\widehat{\tau}$ variance and saving factor obtained with the ATS algorithm for different windows.

Both PRNW and EBW give minimum values of variance included between 10^{-20} and 10^{-15}; the highest values are attained with the CRNW, and are included between 10^{-15} and 10^{-10}. In this case it is even more evident than with ATS_O, that CRNW and PRNW are slower in diminishing the variance.

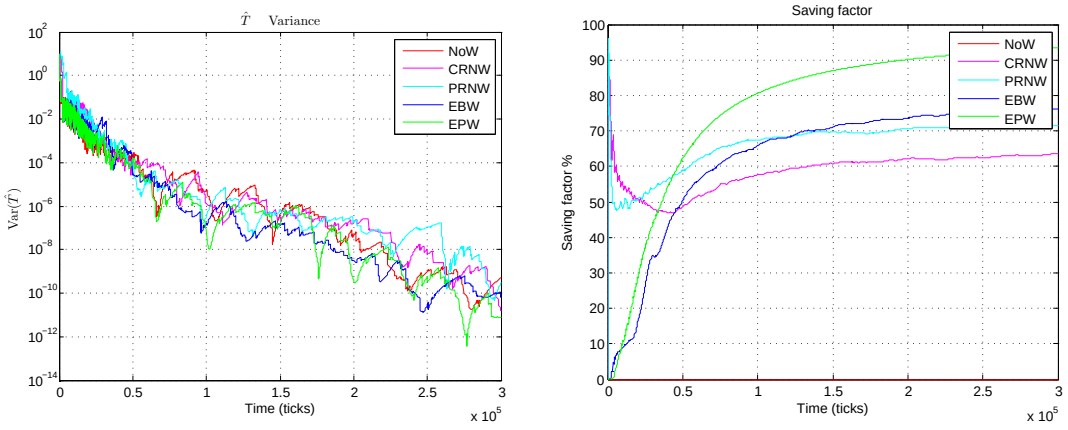
As for the energy conservation, EPW and EBW result to be the best, since their saving factors reach respectively nearly 100% and 80%. Windows based on neighbors communications have on the

[3] The slower evolution depends from the fact that these windows are not sized to the maximum delta for the first cycles.

contrary worse results: PRNW saving factor attains 70% and CRNW nearly 65%, both in steady state.

5.1.3 PI

Even with PI algorithm the time estimation variance has a decreasing trend along all the simulation.



(a) $\widehat{\tau}$ *variance.* (b) *Saving factor (SF).*

Figure 5.4: $\widehat{\tau}$ variance and saving factor obtained with PI algorithm for different windows.

However, this decreasing trend is slower than in the case of the ATS, and probably the reason is the choice of the PI coefficients. The final time estimation variance is similar in all cases. As for the saving factor, the best results are given by the EPW, 95%, and by the EBW, nearly 75%. the PRNW and the CRNW saving factors are respectively 70% and 60%, both in steady state.

Summarizing the results obtained in the ideal case, we can say that, when a high synchronization is desired, the most effective window algorithm is EPW. This particular window ensures good results with both ATS and PI algorithms, while it has bad performances with the ATS_O[4]. Actually, all the windows give better results if used to implement ATS and PI, rather than ATS_O. This last algorithm, in fact, has great performances limitations, due to the lack of skew compensation. An interesting question now is weather these results will change, adding noise to the system. This analysis will be the aim of the next paragraph.

5.2 Case 2: Considering noise but not collisions

In this paragraph we will analyse how the noise affects the system transmissions. As before, we will not consider the possibility of collisions. To simulate the noise we have used two parameters:

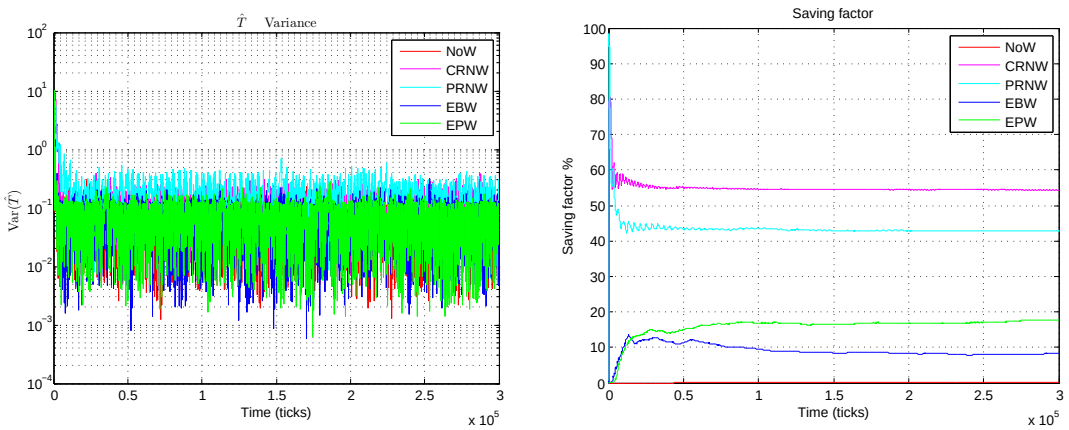
- `value_vel` $= 10^{-7}$: it simulates a clock that is not perfectly stable
- `sigma_n` $= 0.01$: it simulates the error introduced in the communication

[4]Its behaviour can be improved changing the proportional coefficient K.

These values have been chosen because, with `value_vel` higher than the one proposed, considering rho = 0.5, in the ATS algorithm the time estimation variance increases during the simulations, even without any window[5]. This does not happen with the PI algorithm, which guarantees good results even with `value_vel` $= 10^{-5}$.

5.2.1 ATS_O

As it can be seen figure 5.5, EBW and EPW have very similar time estimation variance trends, attaining the minimum variance value $6 * 10^{-4}$. The highest values of variance are reached by the PRNW, with a maximum of $7 * 10^{-1}$. As it can be observed through the saving factor analysis, the introduction of the noise does not cause the performances to worsen, as the attained values are almost the same as in the ideal case. So, the highest saving factors are reached by the CRNW, 55%, and by the PRNW, 45%. The EPW and the EBW are unable to guarantee energy conservation, and their saving factors are respectively less than 20% and 10%.



(a) $\widehat{\tau}$ *variance.* (b) *Saving factor (SF).*

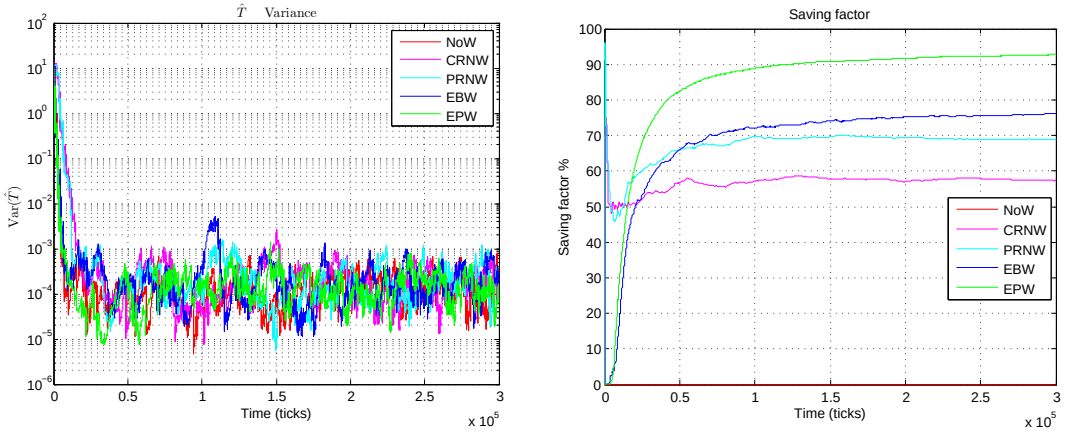
Figure 5.5: $\widehat{\tau}$ variance and saving factor obtained with ATS_O algorithm for different windows.

5.2.2 ATS

Looking at figure 5.6, it can be observed that after a $2 * 10^4$ ticks transient during which the time estimation variance decreases, its values remain included between a maximum of 10^{-3} and a minimum of 10^{-5}. The increased and low bounded variance values in steady state are due to the noise.

The variance trend shows, more clearly than in the previous case, the alternation of local maximums and minimums. This happens since the nodes can not maintain their synchronization, because of the noise effects. So, we find a variance local maximum, when the nodes are not synchronized, and a minimum, when communications ensure good synchronization.

[5]while with rho=0.9 it is almost constant if `value_vel` $= 10^{-6}$



(a) $\widehat{\tau}$ *variance.* (b) *Saving factor (SF).*

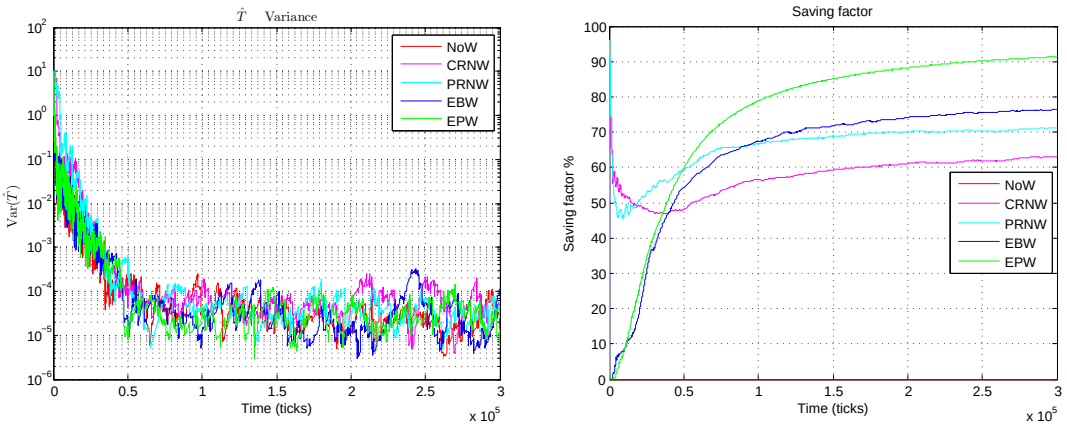
Figure 5.6: $\widehat{\tau}$ variance and saving factor obtained with ATS algorithm for different windows.

As expected, the case which guarantees the best variance values is the one without window; the reason is simple: no window means that the nodes are always listening, so it is easier to regain synchronisation, eventually.

Observing the saving factor graph, we note that the window which guarantees the highest value is still the EPW, whose maximum value is over 90%. Both EBW and CRNW are unable to mantain the same performances as in the ideal case, and their saving factors are respectively 75% and less than 60%. The noise is obviously the cause of the worsening of the results.

5.2.3 PI

As shown in 5.7, the time estimation variance trends are decreasing but low bounded. After the transient, the variance remains always below $3 * 10^{-4}$, in all the five cases considered.



(a) $\widehat{\tau}$ *variance.* (b) *Saving factor (SF).*

Figure 5.7: $\widehat{\tau}$ variance and saving factor obtained with PI algorithm for different windows.

As for the energy conservation, the window which guarantees the best performances is the EPW, with a 90% saving factor. EBW and PRNW have similar saving factors: the first is around 75%, the second 70%. The worst case is given by CRNW, whose saving factor is around 60% in steady state. Comparing these results with those obtained in the ideal case, we notice that the performances do not worsen. The reason is the good noise resistance, typical of the PI algorithm.

To summarize the results obtained so far, we stress the fact that both the windows based on the evaluation of the synchronization error give the best performances with ATS and PI. In particular, EPW ensures a saving factor over 90% with both algorithms. On the contrary, ATS_O has better performances with CRNW and PRNW, though the saving factor values are not really good.

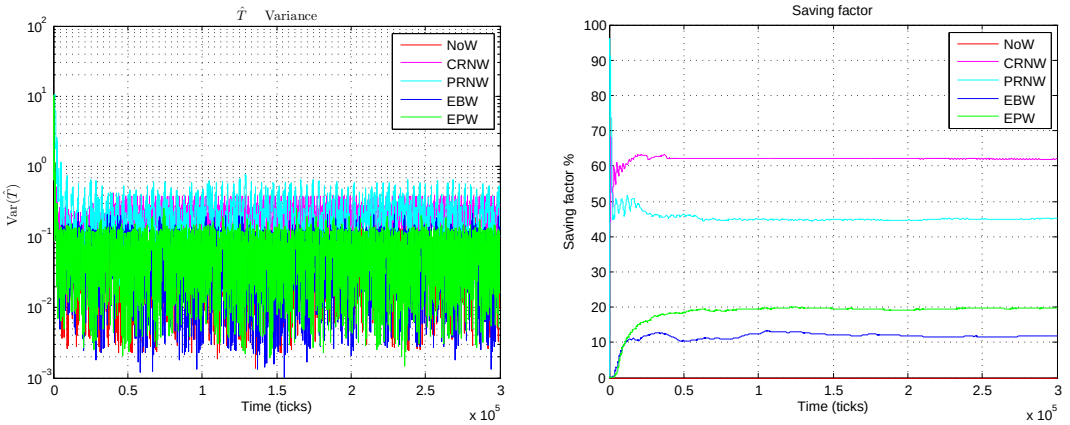
Comparing these results with those obtained in the ideal case, we notice that the noise causes a worsening in the time estimation variance values, especially for the PI and for the ATS, while the effects on the saving factor are negligible.

5.3 Case 3: considering noise and collisions

Let's now consider the effects of collisions in the behaviour of ATS_O, ATS and PI, implemented using the different windows. We could expect that the performances worsen, as the probability of collision increases when nodes are synchronized causing windows to enlarge.

5.3.1 ATS_O

The trends of the time estimation variance and of the saving factor are similar to the ones without collisions. This means that, as the algorithm ATS_O does not ensure a "perfect" synchronization, collisions do not occur frequently.

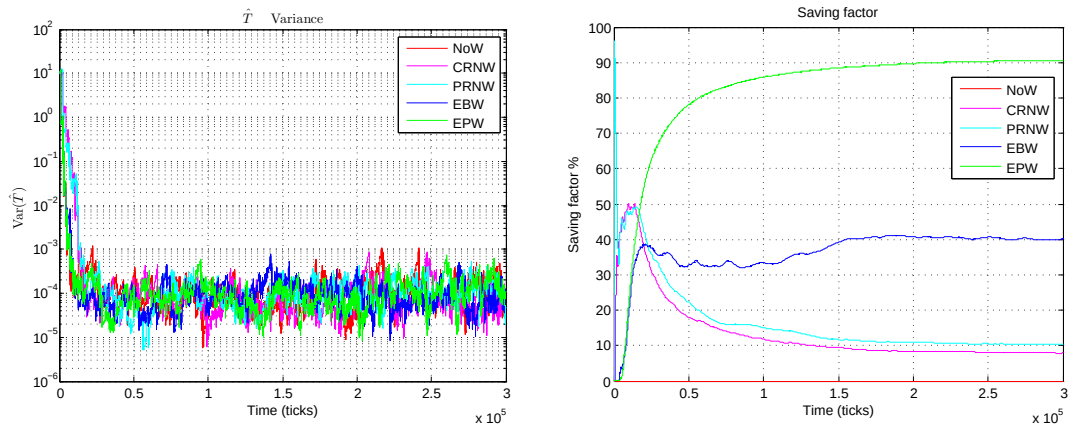


(a) $\widehat{\tau}$ *variance.* (b) *Saving factor (SF).*

Figure 5.8: $\widehat{\tau}$ variance and saving factor obtained with ATS_O algorithm for different windows.

5.3.2 ATS

As it can be seen in fig. 5.9, the trends of time estimation variance for the four windows is more irregular[6] than in the case with no collisions.



(a) $\widehat{\tau}$ *variance.* (b) *Saving factor (SF).*

Figure 5.9: $\widehat{\tau}$ variance and saving factor obtained with ATS_O algorithm for different windows.

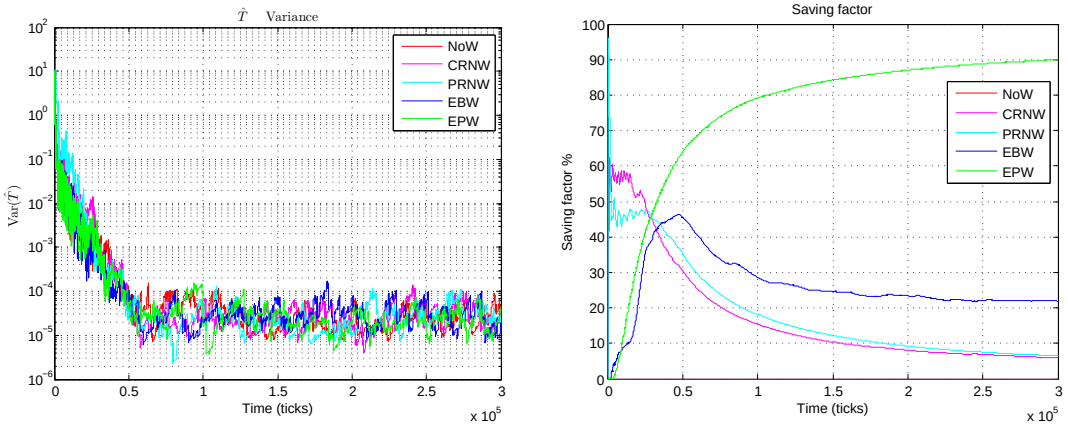
The graph of the saving factor offers interesting information. The EPW gives the best values, reaching 90%. The other window based on the evaluation of the synchronization error has not the same good performances: its saving factor attains only 40% in steady state. This means that the EBW is more affected by collisions than the EPW. As for the PRNW and the CRNW, we notice that their performances worsen drastically: in steady state their saving factors are around 10%. From this analysis we can conclude that these last two windows are unable to manage collisions, which cause the failure of a certain number of communications and the consequentially enlargement of the window.

5.3.3 PI

As for the ATS algorithm, the most interesting thing to observe is how collisions make performances worsen, when the synchronization increases. In fact, the only implementation which guarantees good results with PI is EPW: its saving factor reaches 90%. EBW, PRNW and CRNW, as the variance decreases, give bad saving factor values, meaning that, because of collisions, they are forced to open their window uselessly. This is the cause of the saving factor trends, which encrease at the beginning, while the variances are decreasing, and stabilize around low values in steady state: around 20% for the EBW and 5% for the other two.

In conclusion, as expected, the introduction of collisions brings performances to worsen. It is interesting to notice that the time estimation variance, in the case with collisions, seems to be of the same order of magnitude than the one in the case with only noise, for all the three algorithms. As for the saving factor, the ATS_O is the only algorithm which offers the same performances with

[6] The previous observed alternation of maximums and minimums is affected by collisions becoming irregular.



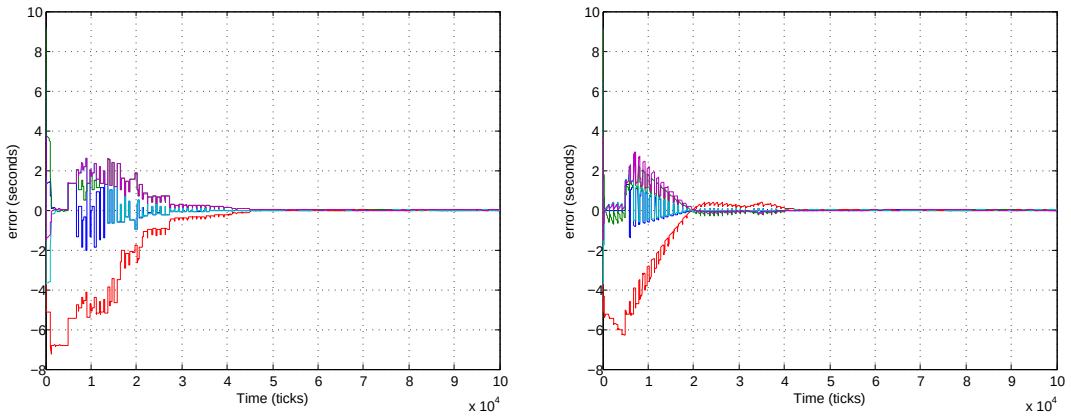
(a) $\widehat{\tau}$ *variance.* (b) *Saving factor (SF).*

Figure 5.10: $\widehat{\tau}$ variance and saving factor obtained with PI algorithm for different windows.

and without collisions because of the poor synchronization. Both PI and ATS are badly affected by this phenomenon, maintaining good results only with EPW. Especially with PI, EBW, CRNW and PRNW are completely useless, if the aim is to preserve energy. So we can say that the elements which cause the greatest problems to save energy in our simulations are collisions. In fact, just with noise it is possible to reach discrete results at least with EPW, EBW and PRNW in the PI and ATS cases.

5.4 Adding a node test

Matlab implementation gives the possibility of simulating the action of plugging a node to an active WSN therefore in these last simulations we connect a node to the WSN, only when a certain amount of time has already passed (see fig. 5.11).

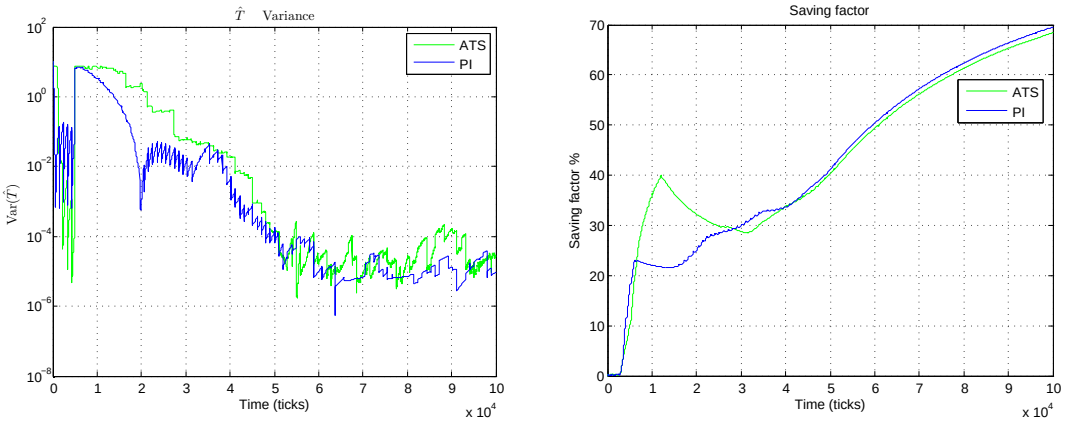


(a) *Using ATS and EPW.* (b) *Using PI and EPW.*

Figure 5.11: Error between five nodes from their instantaneous mean. The fifth node is plugged and considered for the mean after 5000 ticks.

The test is performed only with ATS and PI algorithms considering both noise and collisions. As from the previous simulations it results that the best window algorithm is EPW, we have decided to perform the test only with this one. All the algorithms parameters are set to the values used in the previous simulations. Only the total time and the WSN network are changed. The total time is set to 100000 ticks while the WSN network considered is made by four nodes. The fifth node is plugged after 5000 ticks.

Figure 5.12 shows that with this window algorithm the saving factor behaviour is good with both ATS and PI. When the node is connected, its "big" synchronisation error makes its neighbours windows enlarge. This is desirable in order to let the last connected node be received with good probability and its effect on the saving factor is quickly recovered.



(a) $\widehat{\tau}$ *variance in the WSN.* (b) *Saving factor of the WSN.*

Figure 5.12: Diagrams obtained plugging one node to the WSN network at time $t = 5000$ ticks.

This test proves that the presented adaptive window algorithm, based on the local estimated error, seems to work well even if used in a network to which a new node is plugged. This is the first step to test adaptative windows algorithms with a WSN that can dynamically change its topology. However this kind of test depends a lot from real parameters therefore we have decided that it would be much more interesting to perform them with a real WSN as descried in chapter 7.

Chapter 6

Conclusions

As already explained, the aim of our work is to project algorithms able to guarantee both synchronization and energy conservation. For this reason, we have tried to study the behaviour of four different window algorithms, starting with an ideal case, ending in a more realistic situation, where we have considered the effects of both noise and communication collisions.

These window algorithms are projected to guarantee the nodes the possibility to communicate in windows dynamically sized. The main idea has been equipping each node with a certain amount of memory to evaluate past communications. Therefore parameters responsible for the performances are the memory amount and others depending on the algorithm, such as $\widehat{e}_{accepted}$ and the estimated synchronization error proportional coefficient K.

Accordingly to available devices and desired application, some configurations of these parameters may be preferred.

Our analysis shows, as said in the previous chapter, that the most effective implementation is due to the EPW. In particular EPW and EBW provide a great saving factor without compromising synchronization precision but they require a good synchronization algorithm such as ATS or PI.

ATS_O for its own structure, which does not assure high levels of synchronizations, is less affected by collisions. For this reason it has better performances with CRNW and PRNW than with EPW and EBW.

In conclusion, windows implementations based on error evaluation are preferable than those founded on the neighbours estimation if high levels of synchronization are required. In fact, if supported by performing synchronization algorithms, they allow optimal results in energy conservation, noise resistance and, being fully distributed, they are flexible as regards the network structure.

Chapter 7

Future Works

Since our work is based on Matlab simulations, the realism is obviously limited, and it would be interesting to test the implemented routines in a real network. In this way it will be possible to consider some phenomena which could have been disregarded, or evaluate, for example, some possible hardware limitations.

An important further test would be to study how the windows performances are affected by the dynamical change of the structure of the network. For sure, the windows based on the neighbours estimation would probably need to be modified, not to maintain a maximum amplitude forever when a node dies as described on section 3.3.

Moreover during the tests, it could be useful to change some of the windows parameters, to see how the real WSN reacts. For instance, we could decide to modify the maximum accepted error[1], analysing the effect on the synchronization. In particular a node could not remain synchronized with the others.

Tests results would be of great interest to conclude our work, which as said before is basically simulative, and to project adaptive windows with high performances.

[1] Used in EBW algorithm.

Bibliography

[1] Giuseppe Anastasi et al. “An Adaptive and Low-latency Power Management Protocol for Wireless Sensor Networks”. In: *Proc. MobiWAC'06*. 2006, pp. 67–74 (cit. on pp. 6, 7).

[2] Saverio Bolognani, Ruggero Carli, and Sandro Zampieri. “A PI consensus controller with gossip communication for clock synchronization in wireless sensors networks”. 2009 (cit. on p. 13).

[3] Ruggero Carli et al. “A PI Consensus Controller for Networked Clocks Synchronization”. In: *Proc. 17th World Congress The International Federation of Automatic Control*. 2008, pp. 10289–10294 (cit. on p. 13).

[4] Luca Schenato and Giovanni Gamba. “A distributed consensus protocol for clock synchronization in wireless sensor network”. In: *Proc. 46th IEEE Conference on Decision and Control*. 2007, pp. 2289–2294 (cit. on p. 11).

[5] Moteiv inc. *TmoteSky data sheet*. 2004 (cit. on p. 16).