

Management of Low-density Sensor-Actuator Network in a Virtual Architecture

Vianney Kengne Tchendji, Blaise Paho Nana

▶ **To cite this version:**

Vianney Kengne Tchendji, Blaise Paho Nana. Management of Low-density Sensor-Actuator Network in a Virtual Architecture. Revue Africaine de Recherche en Informatique et Mathématiques Appliquées, 2018, Special issue, CARI 2016, Volume 27 - 2017 - Special issue CARI 2016, pp.75-100. 10.46298/arima.3110 . hal-01442814v3

HAL Id: hal-01442814
https://hal.science/hal-01442814v3

Submitted on 1 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Management of Low-density Sensor-Actuator Network in a Virtual Architecture

Vianney Kengne Tchendji*, Blaise Paho Nana*

*Department of Mathematics and Computer Science
Faculty of Science
University of Dschang
PO Box 67, Dschang-Cameroon
vianneykengne@yahoo.fr, blaisepaho@gmail.com

RÉSUMÉ. Les réseaux de capteurs sans fil (RCSF) font face à de nombreux problèmes dans leur mise en œuvre, notamment aux problèmes de connectivité des nœuds, de sécurité, d'économie d'énergie, de tolérance aux pannes, d'interférence, de collision, de routage, etc. Dans ce document, nous considérons un RCSF peu dense, caractérisé par une mauvaise couverture de la zone d'intérêt, et l'architecture virtuel introduite par Wadaa et al qui permet de partitionner efficacement ce type de réseau en clusters. Dans l'optique de router optimalement les informations collectés par chaque capteur jusqu'à une station de base (nœud sink, supposé au centre du réseau), nous proposons une technique d'utilisation des fréquences multiples pour limiter les interférences lors des communications. Ensuite, nous proposons un algorithme de détection de clusters vides permettant d'avoir une vue globale de la répartition réelle des capteurs dans la zone d'intérêt, et ainsi donner la possibilité de réagir en conséquence. Nous proposons également une stratégie de déplacement des capteurs mobiles (actuators) afin de: sauvegarder la connectivité du RCSF, optimiser le routage, économiser l'énergie des capteurs, améliorer la couverture de la zone d'intérêt, etc.

ABSTRACT. Wireless sensor networks (WSN) face many implementation's problems such as connectivity, security, energy saving, fault tolerance, interference, collision, routing problems, etc. In this paper, we consider a low-density WSN where the distribution of the sensors is poor, and the virtual architecture introduced by Wadaa and al which provides a powerful and fast partitioning of the network into a set of clusters. In order to effectively route the information collected by each sensor node to the base station (sink node, located at the center of the network), we propose a technique based on multiple communication frequencies in order to avoid interferences during the communications. Secondly, we propose an empty clusters detection algorithm, allowing to know the area actually covered by the sensors after the deployment, and therefore, giving the possibility to react accordingly. Finally, we also propose a strategy to allow mobile sensors (actuators) to move in order to: save the WSN's connectivity, improve the routing of collected data, save the sensors' energy, improve the coverage of the area of interest, etc.

MOTS-CLÉS : Réseau de capteurs sans fil, architecture virtuelle, déplacement des actuators, cluster vide, évitement de collision(s), connectivité, routage.

KEYWORDS : Wireless sensor network, virtual architecture, moving actuators, empty cluster, collision(s) avoidance, connectivity, routing.

1. Introduction

For few years now, many improvements have been made in domains such as micro-electro-mechanical systems (MEMS) technology [9], wireless communications, and digital electronics. This enabled the development of micro components that easily combine data collection tools and wireless communication devices, and then opens a wide scope to wireless sensor networks (WSN) [3, 5, 8, 12]. Usually called micro-sensors or simply sensors, these devices with limited resources (bandwidth, computing power, available memory, embedded energy, etc.) have revolutionized traditional networks by bringing the idea of developing sensors networks based on the collaborative effort of a large number of sensors operating autonomously, and communicating with each other via short-range transmissions [6, 7]. The vulnerability of the radio communication that sensors are using added to their resource limitations are factors that raise many problems (interference, intrusion, disconnection, data integrity, etc.).

Nowadays, this technology allows sensors to move while remaining connected. When the transmitters are relatively close and use the same frequencies, there may be interferences while sending messages. The messages that never arrive because of collisions can make the network useless. Radio frequencies should be better used to avoid collisions during the various message's broadcast. In addition, it is common to see WSN composed of several thousand units [4]. In these large networks, the sensors can be grouped into clusters based on their proximity in order to significantly increase the scalability, economy of energy, routing, and consequently the lifetime of the network. The structure provided by this partitioning allows the use of various techniques to improve the quality of a WSN, such as data aggregation [11, 12]. One of the biggest challenges of wireless sensor networks remains a fast and energy efficient routing protocols. If in a sensor network, communications are not well organized, the sensors will communicate haphazardly, and spend their energies unnecessarily. This may cause the networks to be quickly off. Similarly, in a real-time application, the data must quickly reach the base station so that decisions can be taken consistently.

Several works have already focused on this field of study. For example Wada and al [1] has set up a virtual architecture which provides a powerful and fast partitioning of the network into a set of clusters, making administration of WSN more easier. S. Faye and J. F. Myoupo [2] were also interested in this topic, showing the first way to route messages in a sparse network. They also showed how to optimize routing using mobile sensors (actuators). Their work has helped to establish a routing protocol that quickly routes the packets to the base station while limiting some redundancies in the clusters. F. Saffre and al [10] were interested in the use of communication channels to limit collisions during the broacastings.

In this paper, we consider a low-density WSN where the distribution of the sensors is poor, and the virtual architecture introduced by Wadaa and al [1]. The latter is created and orchestrated by base station, which is able to split the network into a set of clusters, depending on the strength and direction of the broadcast it can perform. Here, many empty clusters can be formed in the virtual architecture, unlike in [1] where the WSN is assumed to be dense.

In order to effectively route the data collected by each sensor to the base station (sink node, located at the center of the network), we firstly propose a technique using multiple communication channels to greatly reduce collisions during communications. Secondly, we propose an algorithm for the detection of empty clusters, allowing us to know the area

actually covered by the sensors after the deployment, and therefore, gives the possibility to react accordingly. Finally, we propose a strategy to allow mobile sensors (actuators) to move in order to : save the connectivity of the WSN, improve the routing of collected data, save the energy of our sensors, improve the coverage of the area of interest, reduce the time taken by packets to reach the base station, etc.

The rest of this paper is organized as follows : in section 2, we present the virtual architecture in which we work. Then in section 3, we present our collision(s) avoidance mechanisms. We present a technique of detecting empty clusters in section 4, followed in section 5 by our method of strengthening strategic points by the actuators, and the technique used to properly move the actuators. In section 6, we present the overall structure of our fast and energy efficient routing protocol. Examples and simulation results are presented in section 7. Finally, a conclusion ends the paper.

2. Architecture of the virtual sensor network

2.1. Anatomy of a sensor

This is the basic equipment of any WSN. It has three main tasks : information collection from the deployment area, light treatment (optional) on the collected data and sharing these data with other sensors through multi-hop routing. Despite the great diversity (temperature sensors, humidity, pressure, etc.) existing on the market, they are all mounted on the same architectural diagram mainly made of a unit of : capture, processing, storage, communication, and energy. This material may be supplemented or reduced according to the developer [3]. One can for example add a locating system such as a GPS (Global Positioning System), a mobilizer (to get an actuator). The main and optional elements (represented by dashed lines) are shown in figure 1.

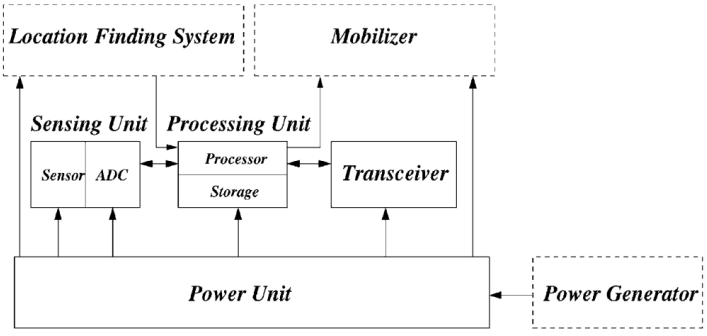
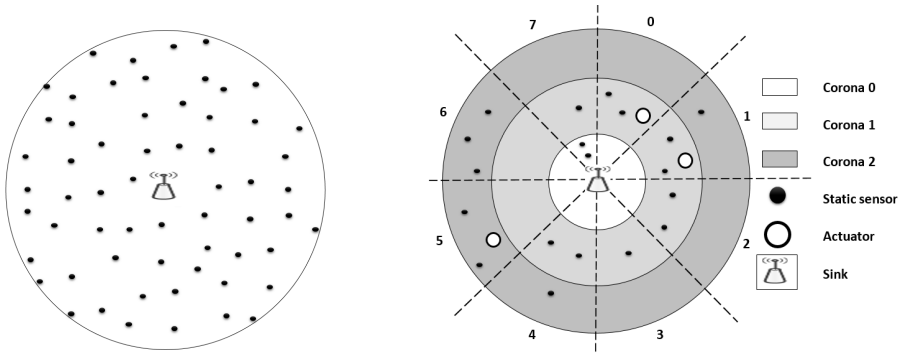


Figure 1 – Hardware architecture of a sensor.

2.2. Virtual network architecture

Let's consider a special sensor called the sink or base station (BS) unconstrained by common sensors's limits and capable of omni-directional transmissions at various transmit power that allows it to covers different transmissions radius forming coronas, and directional transmissions at various angles forming angular sectors. Once deployed in the supervised area (figure 2a), the sensors can be grouped in clusters (as described in [1])

depending on the corona and the angular sector in which they are located (see figure 2b). Thus, the intersection of the corona i and the angular sector j forms the cluster (i, j). Exceptionally, the sink is alone in its cluster and its coordinates are $(-1, -1)$. Since the network is sparse, it is important to identify the empty clusters. This allows the sink to have an overview of the area covered by the sensors, and to achieve a better monitoring.



(a) Example of a basic WSN. (b) A sparse virtual architecture.

Figure 2 – A WSN represented in a virtual architecture.

For the rest we advance the following hypotheses.

2.3. Hypotheses

The network is fully clustered as shown in figure 2b using the technique described in [1]. Furthermore,

– Each sensor has a unique identifier ID in the network. The sensors are static and form a connected network. Adding or removing a sensor is a rare event ;

– A node is able to estimate its residual energy E_r and the sink has the ability to broadcast messages in the network at different radius, or at different angles ;

– The time is divided into slots of length r. The sensors are aware of the number of coronas and angular sectors. The local clock of each sensor is synchronized with the sink's clock ;

– A message sent by a sensor reaches all the sensors located in its transmission range within a slot ;

– Clusterisation is made such that all sensors of a given cluster can communicate with each other and some sensors of neighbor clusters.

With these hypotheses we can tackle the first contribution of this paper which is to limit collisions at different message transmissions both in a cluster and amongst clusters.

3. Collision avoidance mechanisms

In this section, we show how to avoid interference between the messages exchanged amongst the sensors of the same cluster, and those exchanged between the sensors of neighboring clusters.

3.1. Intra cluster collision(s) avoidance : Sharing a communication channel amongst many sensors

To quickly and correctly route the collected data to the base station (through the cluster-heads), we must prevent messages from collision risks. For this, there are several techniques :

FDMA (Frequency Division Multiple Access) : A communication channel is a frequency range. Each sensor will then have its own sub communication channel. In this case, we will ensure to limit the number of sensors per cluster to ensure that the channel is not divided to the point of being unable to carry sensor's data.

TDMA (Time Division Multiple Access) : Each sensor will then have its own time space, large enough, to transmit its data. Here too, we should make sure we limit the number of sensors per cluster to allow cluster-head to collect data from all stations of their cluster at a time not very long. A good time synchronization that allows each sensor to retrieve his messages and send hers without interfering with other sensor's messages, and in the allotted time is one of the major problems of this method.

CDMA (Code Division Multiple Access) : The allocated frequency range supports its spread spectrum technique. But a problem would remain : the modulation and demodulation operations still require significant computing capacity, which implies more energy and expensive components.

CSMA/CA (Carier Sense Multiple Access with Collision Avoidance) is a variant of the CSMA/CD (Carier Sense Multiple Access with Collision Detection) protocol, which allows the CSMA method to work when collision detection is not possible, as in hertzian networks. Its operating principle is to resolve contention before data is transmitted using acknowledgments and timers. The transmitter tests the channel repeatedly to ensure that there's no activity (i.e. no other station is transmitting). Any received message must be acknowledged immediately by the receiver. Any other new message can be sent only after a certain period to guarantee transport without loss of information. The non-return of an acknowledgment, after a predetermined time, makes it possible to detect if there has been a collision. This strategy not only makes it possible to implement an acknowledgment mechanism at the frame level but also has many advantages such as : it is simple and economical, since it does not require a collision detection circuit unlike to CSMA/CD ; It can be used in wireless sensor networks ; Control of restraint is relatively simple ; At low traffic levels, information flow is high ; It is not subject to synchronization constraints ; It is distributed ; the bandwidth is not divided, which allows fast data transfer ; In most of WSN, the collected data is not large, so they are not usually fragmented, and even if that were the case, the CSMA/CA's fragmentation-reassembly mechanism can solve the problem ; It enables to share access to the medium, resolves interference and concurrency access problems ; his back-off algorithm gives the same probability to the various sensors to access the channel ; even if in a given cluster, the informations' harvesting time is relatively short, sensors that were unable to send their data during the allotted time can always do it at the next round thanks to the equiprobability of the back-off algorithm ; etc. For these reasons, we choose to use CSMA/CA.

In summary, CSMA/CA is an appropriate way of solving interference problems and sensor's concurrent channel access in a cluster. But, it is unwise to use this technique over the entire network, because it is possible that sensors in neighboring clusters have adjacent transmission range, and therefore, can not communicate at the same time without risk of collision. To enable them to communicate at the same time, and thus reduce the latency time due to the access time to the communication channel, an additional tip must be associated with the CSMA/CA protocol. To do this, we propose the modulation of frequency of communication between neighboring clusters.

3.2. Inter clusters collision(s) avoidance : Communication channels distribution

Our idea is to assign different communication frequencies to neighboring clusters, so that their sensors can communicate at the same time without risk of collision. To do this, we rely on the techniques of graphs coloring.

In graph theory, to color a graph refers to assign a color to each vertex such a way that two vertices joined by an edge have different colors. The frequency assignment in a network is often compared to the graph coloring problem : defining a graph where vertices represent issuers. Two emitters are connected when they are close enough to interfere. Each emitter has a color representing for us, a communication channel. The issuers represent for us the clusters of our virtual architecture. We generally try to minimize the number of colors (channels), said chromatic number while respecting the fact that two related issuers (clusters) should not have the same color (do not transmit on the same channel), but the sensors of the same cluster communicate on the same channel. This minimization of the number of colors (channels) allows the reuse of frequencies.

There are some accurate algorithms for the coloring of a graph using the minimum number of colors such as the Zykov's algorithm [18, 19] which uses the branch an bound method ; but this class of algorithms have an exponential complexity. Heuristics such as those of Welsh and Powell proposed in [16] and [17] can be used to achieve an acceptable results in polynomial time. An allocation of frequencies to clusters of a sensor network using previous techniques could give the result of figure 3a. But for our use, this frequency distribution also pose interference problems for some clusters. For example, if a sensor A from the cluster $(2, 1)$ wants to send a message to the sensors of cluster $(1, 1)$, it will listen to the communication channel of the cluster $(1, 1)$ (using channel 4) to seek if its channel is free. Note that there's also a neighboring cluster (cluster $(3, 1)$) communicating on channel 4. If the channel is indeed free in cluster $(1, 1)$ but busy in cluster $(3, 1)$ which also uses channel 4, then sensor A will think that the recipient's channel (channel of cluster $(1, 1)$) is busy (because clusters $(1, 1)$ and $(3, 1)$ are both using the same channel 4) and will try to postpone the transmission of its message. On the other hand, this inadequacy can cause enormous delays in the delivery of messages which can be very damaging especially for a real-time application.

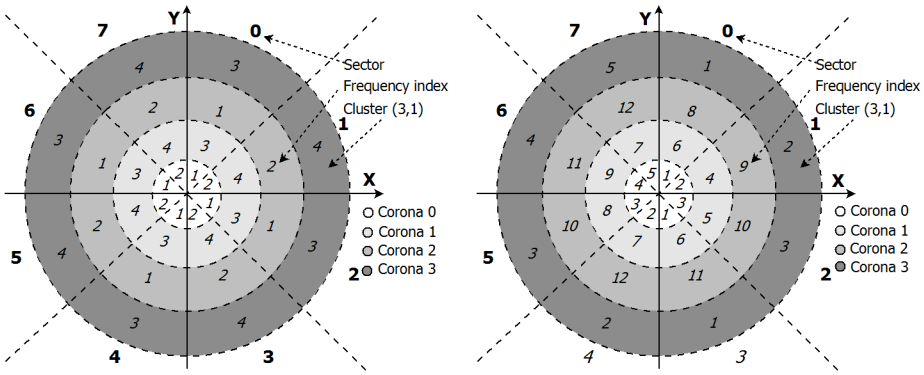
To overcome this problem, we propose a greedy heuristic (algorithm 1) allocating communication channels at a distance of two hops to different clusters of our virtual architecture. The communication channels are distributed such that two clusters separated by a single intermediate cluster use different channels. This will have for first effect to reduce the flooding of the network, because a message will be confined in a single cluster since it is transmitted using a particular frequency. We will see in the following that it can get out of this cluster only via a gateway node that controls all the outgoing messages.

The heuristic we propose performs a cluster by cluster processing. For each cluster, it lists its neighbors (including those at two hops), and derives the frequencies used by these neighbors. The channel assigned to the cluster is the smallest unused channel in the vicinity of the cluster.

When the base station has finished determining the communication channels, each cluster is then alerted by a message *UseFreq* broadcasted in an angular section (or angular wedge) covering the cluster. This message allow the sensors of the recipient cluster to update their communication channels according to algorithm 2. The message *UseFreq* consists only of small numbers. The first is a real number defining the amplitude of a channel, the second is an integer representing the index of the frequency that the cluster should use, the third is a list of small integers representing the index-numbers of the frequencies of neighboring clusters and the last is a pair of integers representing the coordinates of the cluster concerned. These index-numbers allow a sensor to calculate the frequency range on which to transmit a message towards its own cluster and also to its neighboring clusters.

Figure 3b shows an allocation of frequencies in two hops using our algorithm. If we look carefully, we can realize that the sensors of the first corona are not subjected to the constraint of distancing the channels by two hops, since they transmit their messages directly to the sink. On the other hand, when the sink wants to communicate with a sensor, it transmits on the frequency of the cluster in which the sensor is located, in the corresponding sector, with a power capable to reach the cluster. However, communication is very rare in this direction, since almost all messages travel from the sensors to the base station.

We also use the CSMA/CA protocol for inter-cluster communication for its many benefits. Each cluster has a leader node also called cluster-head (CH) which can both play the role of a common sensor and the role of the gateway by wich a message can go out of the cluster. Each cluster (apart from those of the first corona) also have an other cluster called relay cluster or cluster-relay through which passes any message it sends to the sink. When a cluster-head wants to transmit data to the sink, it transmits it on its relay cluster's channel using the back-off algorithm of the CSMA/CA protocol.



(a) One hop frequency allocation scheme. (b) Two hops frequency allocation scheme.

Figure 3 – Frequency allocation scheme.

Theorem 1 *This communication channels allocation algorithm determine a distribution, limiting interference with neighboring clusters in $O(n\log(n))$ time for n clusters.*

Proof. In fact, this algorithm is articulated in three main axes. We start by performing a sorting ; We know that this operation has a complexity in $O(n\log(n))$. Then comes two loops : the first calculates for the n clusters of the architecture, the frequency to be used ; hence a complexity of $O(n)$ for this loop. Then the second makes n broadcast to inform each cluster of the channel to be used ; Hence a complexity in $O(n)$ for this second loop. This algorithm thus has an overall complexity in $O(n\log(n)) + O(n) + O(n) = O\left(n\log(n)\right)$. ■

The available frequency range is divided into N channels and dispatched for each cluster using the directional transmissions reaching the concerned cluster. Each cluster has at must 24 neighbors (including those at two hops). After repeating the algorithm several times on various samples, we noticed that the frequencies range is beeing subdivided into $N = 9$ channels for an odd number of angular sectors and $N = 12$ channels for an even number of angular sectors.

Now that we know how to avoid collisions at various emissions, we can tackle the next goal of this paper : How to correctly and quickly route messages to the base station. For this, we will start with the detection of empty clusters, and then we will see how to move mobile sensors into these empty clusters to improve the routing.

4. Detection of empty clusters and distributed cluster-heads election

This detection of empty clusters phase must precede routing of data in order to define the best path that data should follow to quickly reach the sink (using for example the message propagation tree of figure 5b). It also allows to know the area actually covered by the sensors after the deployment, and therefore, gives the possibility of react accordingly. A cluster is considered empty if it contains no sensor, or if it contains a set of sensors disconnected from the rest of the network (or disconnected from the sink).

For c coronas and s angular sectors, the sink counts $(c \times s)$ clusters in its virtual architecture ; therefore, for each message received, it regularly updates two tables $h(c, s)$ and $relay(c, s)$. Each entry (i, j) of the table $h(c, s)$ contains 1 if cluster (i, j) is not empty and 0 otherwise, allowing the sink to get an overview of the sensors distribution ; and each entry (i, j) of the table $relay(c, s)$ contains the coordinates of the relay cluster of the cluster (i, j).

4.1. Sink's algorithm

After the partitioning and distribution of communication channels, the sink periodically broadcasts the date on which the detection algorithm will begin. All sensors are awake and the sink initiates the detection. The sink then transmits on each channel of its neighboring clusters (those of the first corona) a message $Detect$ containing its coordinates $(-1, -1)$ and an integer *numberOfHops* initialized at 0 representing the distance (in terms of number of hops) to the sink (algorithm 3). After this, it wait some acknowledgments (ACK messages) that will allow it to update the tables h and $relay$. Due to network connectivity, it is certain that at least one sensor of the first corona will receive this message. During the algorithm, each ACK message transmitted by a sensor towards the sink contains the coordinates of its cluster and those of his relay cluster. Table $h(c, s)$

is initialized to 0. At each reception of a message from a sensor of the cluster (i, j), BS puts 1 in $h(i, j)$ and, in the entry $relay(i, j)$, it assigns the value contained in the variable *relay* of the received message. The first ACK message arrives 6 slots after the launch of the algorithm (as we shall see later, 1 slot for the transmission of the $Detect$ message, 2 slots for the two timers, 2 slots for the transmission of the messages $Head1$ and $Head2$, 1 slot for the transmission of the ACK message), and then an ACK message arrives every 2 slots ; the sink knows that the detection (algorithm 3) has ended when it receives no ACK message after more than two slots.

4.2. Sensor's algorithm

The network is supposed connected, so for all cluster (i, j) considered non-empty, there is always a path from it to the sink node. Isolated clusters cannot reach the sink and are considered empty even if there are not. There are five main events in the detection of empty clusters :

1) The reception of a message *Detect* asking sensors to indicate their coordinates ;

2) The reception of a message *Detect_timer* asking sensors to be ready for the election of the cluster-head ;

3) The reception of a message *Head1* sent by a node to propose itself as a gateway node (or cluster-head), if it has the smallest distance to the sink ;

4) The reception of a message *Head2* sent by a node to notify the other sensors of his cluster that he is the cluster-head and ;

5) The reception of a message *ACK* sent by a cluster-head to the sink node to indicate their coordinates, and also the fact that the cluster is not empty.

Reception of a message $Detect$

When a sensor receives a $Detect$ message, it checks two conditions : if it has not already had to route a $Detect$ message, and if its cluster is adjacent to that of the sender of this message. The reason is the following : firstly, each sensor has to route only a single $Detect$ message, because any other similar message necessarily comes from a more distant cluster to the sink. Secondly, the sizes of the clusters of the lower coronas are smaller than the sizes of the more distant clusters, since the sensors transmission range are constant, in lower coronas, transmission can easily cover several neighbors clusters. If one of the two conditions is not satisfied, it ignores the $Detect$ message. Otherwise, if it is the first time he receives the $Detect$ message (i.e. if it has not received a $Detect_timer$ message), it builds a message $Detect_timer$ containing the starting date of the distributed cluster-head election noted End_Date, and broadcasts it on its own cluster's channel. Any sensor that receives a $Detect_timer$ message can't send an other $Detect_timer$ message. It just waits still the date End_Date to begin the cluster-head's election algorithm.

While waiting for End_Date, sensors that receive the $Detect$ message (including the one that received the first $Detect$ message), compare the value $Detect.numberOfHops$ to their local value $numberOfHops$. If $Detect.numberOfHops+1 < numberOfHops$ this means that the message comes from a nearest cluster from the sink ($numberOfHops$ is initialized at ∞ for all the sensors, forcing the sensors to route a single $Detect$ message), then the sensor puts its variable $numberOfHops$ at $Detect.numberOfHops+1$, takes the sender as relay cluster, builds an other $Detect$ message an broadcast it to permit other sensors to identify theirselves (algorithm 4).

At the time End_Date contained in the $Detect_timer$ message previously received, the distributed cluster-head election algorithm (algorithm 6) is immediately executed. At the end of this algorithm, the elected cluster-head sends an ACK message to the sink using the frequency of its relay cluster.

Before the distributed cluster-head's election in each cluster, every sensor of the given cluster that are able to join some relay cluster have received a $Detect$ message from this relay cluster, and know how far they are from the sink ($numberOfHops$) ; they are also able to estimate their residual energy E_r. We rely on these information available locally at each sensor to elect the cluster-head. The algorithm takes place in two phases, requiring only two broadcasts (messages $Head1$ and $Head2$). Each sensor with a residual energy superior or equal to the energy threshold E_s begins by calculating a first timer at the end of which it is supposed to broadcast a $Head1$ message to propose itself as the leader. The local channel is normally free and the timer is calculated such that it is the nearest sensor to the sink that will access the channel. If there are many sensors having the same distance to the sink, the one with the highest ID accesses the channel. The duration of this first timer is $\left(1 - \frac{1}{numberOfHops} + \frac{e^{(-ID)}}{\Lambda}\right)$ slots [1]. The first sensor accessing the channel broadcasts its $Head1$ message and oblige other sensors to cancel the sending of their $Head1$ message (algorithm 7 : Reception of a $Head1$ message).

Reception of a message $Head1$

The sensor that sent the $Head1$ message, and other sensors that have received the message $Head1$, and which have a residual energy greater than the threshold E_s, and have a $numberOfHops$ less or equal to the one contained in the message $Head1$, calculate and arm a second timer. This second timer rely on the residual energy E_r of each sensor, and is such that it is the sensor having the highest residual energy that accesses the channel the first. The duration of this second timer is $\left(\frac{1}{E_r} + \frac{e^{(-ID)}}{\Lambda}\right)$ slots. At the end of this second timer, a single sensor (this is actually the cluster-head) access the channel and notifies (by sending a message $Head2$) all other sensors of its cluster that it is now the cluster-head.

Reception of a message $Head2$

At the reception of the message $Head2$, the receiving sensor disarms the second timer and puts its local data up to date (algorithm 8 : Reception of a message $Head2$). This message contains the ID of the cluster-head, the coordinates of the relay cluster, as well as the distance (in terms of number of hops) of the cluster to the sink. The cluster-head is charged of building and sending an acknowledgment (ACK message) to the sink. The cluster-heads of the first corona send their ACK message directly to sink, while those of the other coronas send their ACK message through the relay cluster.

Reception of a message ACK

A sensor that receives a message ACK from the neighbor node checks whether this message is for its cluster. In this case, it sends it to its gateway node which checks if it has not already routed an ACK message from the same cluster. In this case, it broadcasts it on the channel of its relay cluster. Otherwise, it simply ignores the message.

Theorem 2 *For c coronas and s angular sectors, the empty clusters detection algorithm requires $[c \times (s-1) + 4]$ slots in the worst case.*

1. Λ is selected relatively large so that the timers of two sensors having the same distance to the sink differ by only a few microseconds. Eg 100, 1000, ...

Proof. The worst case corresponds to the one where the $Detect$ and ACK messages move from the sink to the furthest cluster following a path in the form of a spiral as in the figure 4. For c coronas and s angular sectors, the $Detect$ message needs $\left(\frac{c}{2} \times (s-1)\right)$ slots to reach the most distant sensor if c is even, $\left(\frac{c-1}{2} \times (s-1)+1\right)$ slots if c is odd, and 1 slot if $c = 1$. The ACK message sent by this sensor takes an equivalent amount of time to reach the sink. Additionally, it is easy to see that the election of a cluster-head takes 4 slots in each cluster. So the first ACK message arrives 6 slots after the launch, then another arrives every 2 slots, and the last one arrives $[c \times (s-1)+4]$ slots after the launch. ■

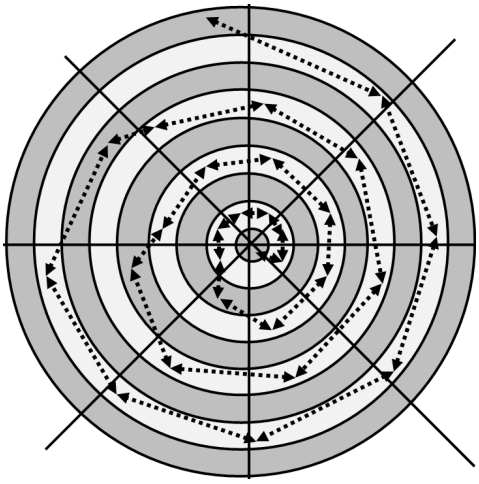


Figure 4 – Longest path traveled by a message to reach the sink node.

After the detection of empty clusters and the election of cluster-heads, the network is ready to be use. Sensors can collect data and send them to the sink through possibly non optimal paths. The following section aims to optimize routing by offering more optimal paths.

5. Routing optimization using mobile sensors

This section is inspired from [2]. Here we introduce the actuators : sensors with a mobilizer, allowing them to move on the sink's order. They can be used for many purposes, depending on the user, for example :

– Being the CH in a cluster where the residual energies of the sensors are under the threshold energy ;

– Collect and route information in isolated areas ;

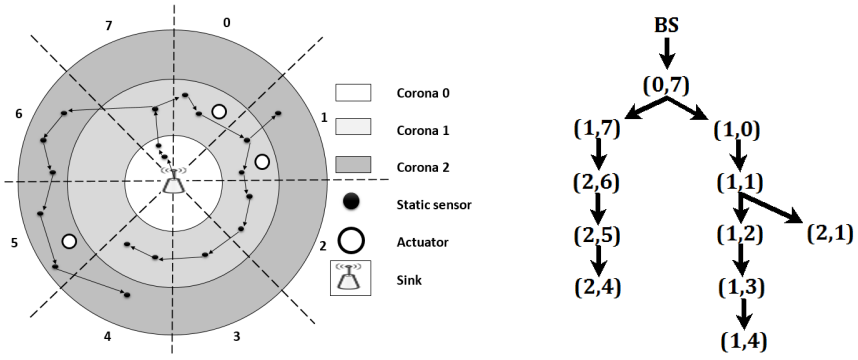
– Connect a sub isolated connected network to the main network ;

– Being sent in strategic empty clusters (purpose of this paper) to optimize the routing.

5.1. Searching and filling strategic empty clusters by actuators

From the table $h(c, s)$, the sink can identify empty clusters. In order to know which ones it is going to fill first, it should produce the messages spreading tree like in figure 5b

by using the two tables it has like this : Take BS as root and the first tree's leaf. As long as there's an unvisited leaf (i, j), search in the table $relay(c, s)$ the cluster that has the cluster (i, j) as relay cluster and add them as the sons of (i, j).



(a) Propagation of messages in the supervised area. (b) Messages spreading tree.

Figure 5 – The messages spreading tree obtained after the detection of the empty clusters.

Filling a strategic empty cluster has the effect of reducing the tree's height (figure 5b). The ideal one would be to reduce this height to the number of coronas of the virtual architecture. The routing will be optimal if for any cluster of corona k, the transmission of a message towards BS passes through k intermediate clusters.

5.1.1. Rule of detection of clusters which access can be improved

To optimize the routing to a cluster, it would be good to know whether the current access is optimizable. In figure 5, it is the case of cluster $(1, 4)$ which is at 5 intermediate clusters from BS instead of 1 if an actuator were judiciously placed in cluster $(0, 3)$.

Rule : Let $\mathbb{T}$ be a messages spreading tree similar to that of figure 5b, $prof(i, j)$ denotes the depth of the cluster (i, j) in the tree $\mathbb{T}$. The path from sink to (i, j) can be improved if there is another cluster (i', j') with a depth $prof(i', j')$, such as $i' \geq i$ and $prof(i', j') < prof(i, j)$, i.e. (i', j') is in a corona greater or equal to (i, j)'s but in the tree $\mathbb{T}$, (i', j') appears at a depth less than (i, j)'s.

5.1.2. Detection of strategic empty cluster to fill in priority

To determine this priority cluster (PC), we establish for every corona a the list $L[a]$ of clusters of this corona which access can be improved (i.e for C coronas, we should have C lists). Each list $L[a]$ contains the coordinates (a, j) of clusters of the corona a which access can be improved. It is in the form : $L[a] = [(a, j_1), (a, j_2), \ldots, (a, j_n)]$. From each list $L[a]$, *we extract the longest list* $L_s[a]$ *made of consecutive clusters of* $L[a]$. In the list $L_s[a]$, each (a, j) represents the coordinates of the clusters of corona a that follows in the message spreading tree. The coordinates (x, y) of PC are deduced from the longest sub list $L_{sp}[a]$ taken among the $L_s[a]$ extracted lists. $x = a - 1$ and y is equals to the default rounding average of j (the j are the second components items (a, j) of the list $L_{sp}[a]$). As long as there are available actuators, it is necessary to move an actuator at the cluster (x, y), another at the cluster $(x - 1, y)$ if it is empty, ... another at the cluster $(0, y)$ if it is empty. The process can be repeated until the routing is optimal, i.e. up to $prof(\mathbb{T}) \leq C$. For the example of figure 5, the determination of PC is presented in annex A.

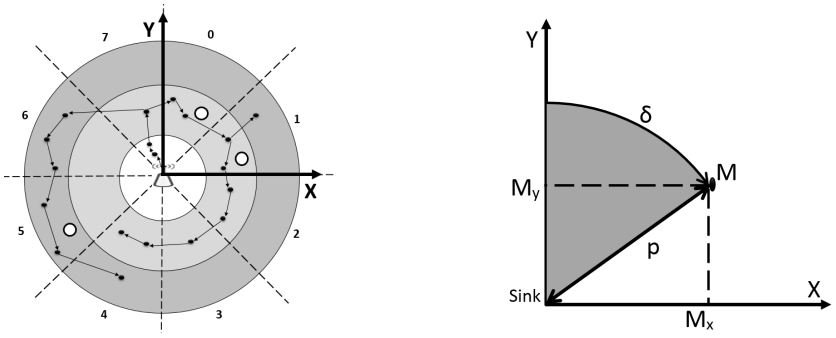
Before moving a mobile sensors, the sink node must calculate the distances from it to the target empty cluster, it is better to choose the most appropriate actuator, based on distance, residual energy, availability, etc.

5.2. Moving a mobile sensor

We propose to move an actuator from the cluster (x, y) to the empty cluster (j, k). To properly move our actuator from cluster (x, y) to cluster (j, k), it will need a distance (not in terms of number of hops) and a direction. To simplify our calculations, we should be in a Cartesian plane. For this we describe here how to transform our current coordinate system (Dynamic Coordinate System : DCS) in a Polar Coordinate System (PCS) and then in to a Cartesian Coordinate System (CCS).

5.2.1. Correspondence between the DCS, the PCS and the CCS

To get the distance and the direction to follow, we must define a reference. Thus the origin of our reference will be the sink. The Y axis is taken such that it coincides with the left edge of the first section (section 0). The X axis is at a quarter turn from the Y axis so that the angle $X, \widehat{sink}, Y$ is direct (figure 6a). Any point of the DCS is discoverable using p (its distance from the sink) and φ (angle measured from the Y axis) as in figure 6b.



(a) Definition of the reference (b) PCS and CCS

Figure 6 – Correspondence between the DCS, the PCS and the CCS

Denote 'α' the angle of a sector and 'e' a thickness of a corona, we can state corollary 1.

Corollary 1 *Let M be a sensor of the cluster (c, s) assumed at it center. In the PCS, M has the coordinates (p, φ) where* $p = c \times e + \frac{e}{2}$ *and* $\varphi = s \times \alpha + \frac{\alpha}{2}$ *;*
Let $M(p, \varphi)$ be a point in the PCS. In the CCS, M has coordinates (x, y) with $x = M_x = p\sin(\varphi)$ *and* $y = M_y = p\cos(\varphi)$.

Now we can start the necessary calculations (distance p and angles φ) to move the actuators.

5.2.2. Calculation of the distance (p)

The distance between two points A and B of the plane is given by the norm of the vector $\overrightarrow{AB}$, denoted $||\overrightarrow{AB}||$ or just AB. According to figure 6b, an actuator that moves from the sink node (with coordinates $(0, 0)$) to the point M (with coordinates (x, y)) must cover the distance $p = ||\overrightarrow{sink\ M}|| = ||(x - 0, y - 0)|| = ||(x, y)||$. But $||\overrightarrow{sink\ M}||^2 =$

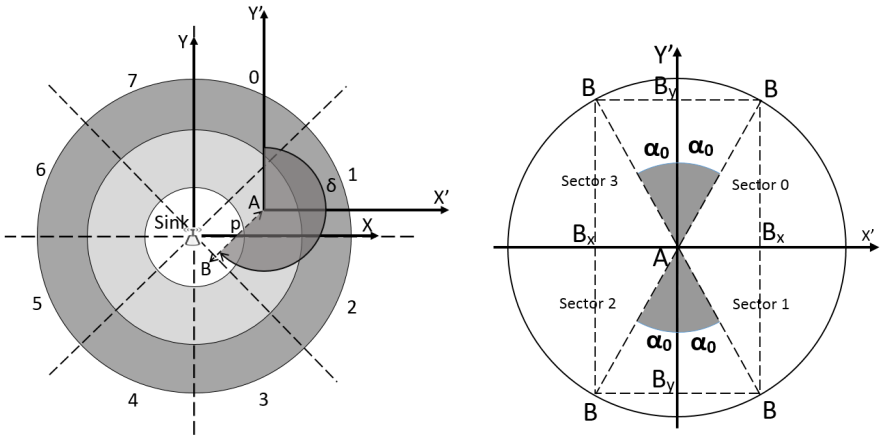
$(sink\ M_x)^2 + (sink\ M_y)^2 = x^2 + y^2$. Thus $||\overrightarrow{sink\ M}|| = \sqrt{x^2+y^2}$. So we have corollary 2.

Corollary 2 *Moving an actuator from the center of the cluster $A(c_1, s_1)$ of polar coordinates (p_1, φ_1) to the center of the cluster $B(c_2, s_2)$ of polar coordinates (p_2, φ_2) returns to move this actuator from the point $A(A_x, A_y)$ to the point $B(B_x, B_y)$ of the cartesian coordinates system on a distance $p = ||\overrightarrow{AB}||$ where $\overrightarrow{AB}(B_x - A_x, B_y - A_y)$, $p = ||\overrightarrow{AB}|| = \sqrt[2]{(B_x - A_x)^2 + (B_y - A_y)^2}$, $A_x = p_1\cos(\varphi_1)$, $A_y = p_1\sin(\varphi_1)$, $B_x = p_2\cos(\varphi_2)$ and $B_y = p_2\sin(\varphi_2)$.*

5.2.3. Calculation of the angle (φ)

To facilitate the calculation of the value of φ, let's make a change of reference.

Change of reference : We want to move a mobile sensor from a point A to a point B. For this, we define a new reference in which the base vectors are collinear with those of the previous (see figure 7a).



(a) Reference change

(b) Angle between the axis Y' and the displacement vector

Figure 7 – Angle displacement

The reference $R_1 = (sink, \overrightarrow{i}, \overrightarrow{j})$; $R_2 = (A, \overrightarrow{i}, \overrightarrow{j})$; $R_2 = t_{\overrightarrow{sink\ A}}(R_1)$ where t denotes the translation of vector $\overrightarrow{sink\ A}$. The coordinates of two points $M(x, y) \in R_1$, and $M'(x', y') \in R_2$ such that $M' = t_{\overrightarrow{sink\ A}}(M)$ are now linked by the following relations : $x' = x - x_A$ and $y' = y - y_A$.
Basis vectors of these two references are pairwise collinear, that's why the angles found in one of the references will be equivalent in the second.

Calculation of the inclination α_0 formed by the displacement vector and the Y' axis : The displacement angle φ that we want to calculate is strongly related to the angle α_0 formed by the vector $\overrightarrow{AB}$ and the Y' axis. Figure 7b presents the different situations we may encounter. We deduced that $\sin(\alpha_0) = \frac{|B_x|}{AB} \Rightarrow \alpha_0 = \sin^{-1}\left(\frac{|B_x|}{p}\right)$ where $p = AB$.

Determination of the displacement angle φ : From figure 7b, point B can be found in one of the four sectors.

Corollary 3 *The displacement angle φ of an actuator from the point $A(A_x, A_y)$ to the point $B(B_x, B_y)$ of $(A, \overrightarrow{i}, \overrightarrow{j})$ reference is given by :*

1) if B is in the sector 0*, i.e.* $B_x > 0$ *and* $B_y \geq 0$ *then* $\varphi = \alpha_0$

2) if B is in the sector 1*, i.e.* $B_x \geq 0$ *and* $B_y < 0$ *then* $\varphi = \pi - \alpha_0$

3) if B is in the sector 2*, i.e.* $B_x < 0$ *and* $B_y \leq 0$ *then* $\varphi = \pi + \alpha_0$

4) if B is in the sector 3*, i.e.* $B_x \leq 0$ *and* $B_y > 0$ *then* $\varphi = 2\pi - \alpha_0$

For a practical example of moving a sensor, see annex B.

Now that we have all the different modules of our protocol, we can now describe how it works.

6. Unrolling of the routing protocol

Once the sensors are randomly deployed around the sink, the base station performs the clustering of the deployment area in small clusters easily manageable. Note that at this phase, sensors does not communicate, therefore their energy consumption is quite insignificant. Then the base station determines and allocates communication channels to different clusters to limit inter-clusters interferences. Then comes the empty cluster detection phase initiated by the sink. This step allows the base station to get an overview of the distribution of the sensors in the area of interest. Cluster-heads and relay clusters are also defined at this step. Already at this stage, the routing of collected data can begin. The informations collected in the empty clusters detection phase will allow the sink to move the actuators in strategic empty clusters in oder to optimize the routing.

Once the data are captured, it would be wise to send them as soon as possible to the base station while avoiding network congestion. The messages spreading tree seen previously may be very useful here. This tree optimized by the sink with actuators, allows each cluster to send its data to the sink by the shortest path (number of hops).

In addition, the CSMA/CA protocol executed by each sensor in each cluster and between clusters helps to avoid network congestion. Another way of making the best results would be to attach an aggregation protocol to limit redundant messages routed in each cluster.

7. Simulation and analysis of our solution

7.1. Tools and simulation environment

Using an HP computer Intel (R) Core (TM) i7-2630QM CPU @ 2.00 GHz$\times$8, 8GB of RAM, running Windows 8 Professional ; a discrete event network simulator J-Sim ; and a sample of 1000 sensors randomly deployed within 10 km of the sink ; we also have a sufficient number of mobile sensors. To remain in the logic of low density network, we have limited number of mobile sensors to 50% of the number of clusters i.e. 40 mobile sensors for our tests, knowing that few of them will be useful and the rest will serve for possible replacements. The energy threshold is set at 40% of the total energy of each sensor ; the virtual architecture has 10 coronas and 8 sectors of 45^o each. We repeated our tests many times and in what follows we are using the averages of the obtained results. The energy

model is the one adopted by many efficient contributions [14] : $E = E_{trans} + E_{recep}$. E_{trans} and E_{recep} are respectively the total energy used for transmissions in the network and receptions, knowing that each sensor has a range of 500 meters, an initial energy of 100 joules, and needs $35.28 \times 10^{-3} joule$ per transmission and $31.32 \times 10^{-3} joule$ per reception. Each actuator has an initial energy of 250 joules and consumes $0.5 \times 10^{-3} joule$ per meter to move. A slot takes $780 \mu s$. The curves were made with version 5.0 of gnuplot software.

7.2. Analysis of the simulation results

Time taken for the election of CH

For this particular test, we also wanted to test the scalability of our cluster-head election algorithm. For this we carried out the test on an architecture made of 16 sectors and 32 coronas i.e. 512 clusters. On figure 8, each point represents the duration of the cluster-head election in a given cluster. The figure shows that our distributed cluster-head election algorithm which needs only two broadcasts in each cluster stays less than four slots in each cluster.

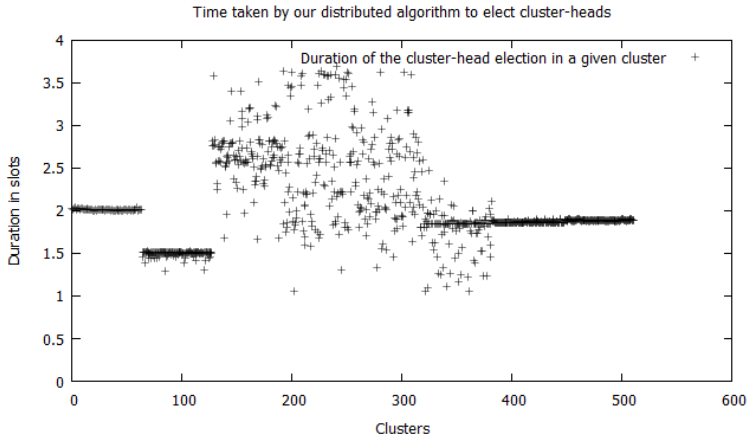


Figure 8 – Time taken for our distributed cluster-head election algorithm in each cluster.

Energy progress

Figure 9 shows the energy consumption among both ordinary sensors and cluster-heads sensors. The sharp fall of energy observed at the beginning of the curve is normal. Initially, CH are not yet elected, and this operation wastes enough energy. This drop of energy is significantly reduced once the CH are elected.

Use of actuators to improve energy consumption

Figure 10 compares the energy consumption of the cluster-heads and ordinary sensors when the routing is not optimized and when routing is optimized with actuators. An economy of energy is observed among both ordinary sensors and cluster-heads sensors. This increases the longevity of the network. The simulation is made for the detection of empty clusters, the election of cluster-head and routing. Here too, the great loss of energy observed at the beginning of the curve is due to the fact that the cluster-head are not elected at

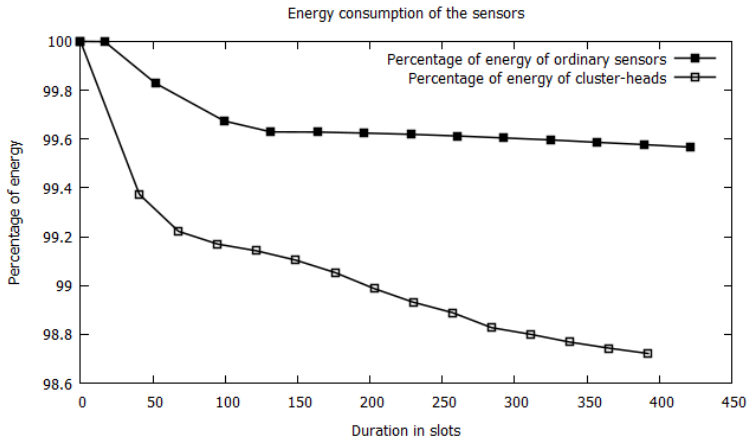


Figure 9 – Energy consumption of ordinary sensors and cluster-heads

the beginning. It is clear that this energy loss is significantly reduced once the cluster-head are elected.

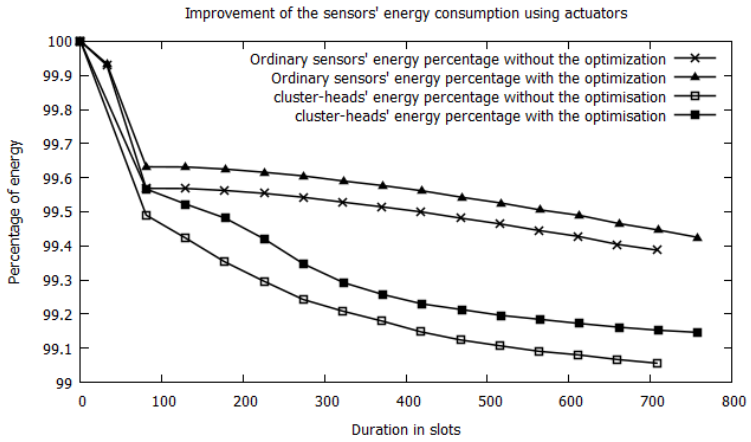


Figure 10 – Improved of the power consumption using actuators

Effect of the number of actuators

Figure 11 shows that, compared to when actuators are not used, the more we have available mobile sensors, the more routing tends to be optimal allowing enormous gains in terms of energy and messages delivery delay. Also notice that when the routing is optimal, the energy gain and the message delivery delay are also optimized. At that time, it is no longer necessary to continue increasing the number of mobile sensors. The rest of mobile sensors can then be used to replace those that are defective.

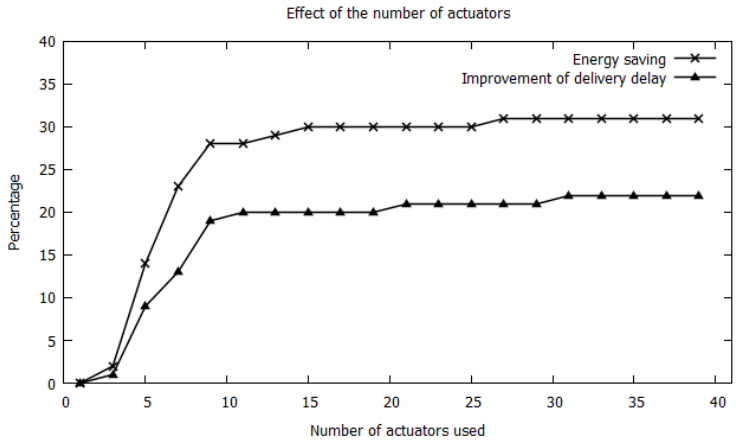


Figure 11 – Effect of the number of actuators.

8. Conclusion

In this paper we have presented a virtual architecture that facilitates the management of WSN. We have presented some technics which allows us to avoid collisions in a WSN both between clusters an inside clusters. We also introduced the gateways nodes, their economical election protocol needs only two message transmissions, we show how to limit redundant messages through them. But our main aim was to optimize the routing of collected data towards the sink. That's why we describe a method of detecting and filling strategic empty clusters in which we can send mobile sensors (actuators) in oder to shorten the road taken by messages. We also present other possible utilities of the actuators, and show a new way to perform their movements to improve the routing of the collected data.

In a very soon future, we intend to work on the mechanism of reelection of the cluster-head in a cluster ; the mechanism of changing relay cluster if the current relay cluster is no longer accessible and the mechanism of redirection of packets after the positioning of a mobile sensor. It would be also interesting to integrate data aggregation techniques to minimize the number of messages circulating in the network. Authentication and security are also important issues in this kind of network.

9. Bibliographie

[1] A. WADAA, S. OLARIU, L. WILSON, M. ELTOWEISSY, K. JONES, « Training a wireless sensor network », *Mobile Networks and Applications*, Vol. 10, N° 1-2, p. 151–168, 2005.

[2] SÉBASTIEN FAYE, JEAN-FRÉDÉRIC MYOUPO, « Deployment and Management of Sparse Sensor-Actuator Network in a Virtual Architecture », *International Journal of Advanced Computer Science*, Vol. 2, N° 12, December, 2012.

[3] I. F. AKYILDIZ, WEILIAN SU, Y. SANKARASUBRAMANIAM, E. L. CAYIRCI, « A survey on sensor networks », *IEEE Communications Magazine*, Vol. 40, N° 8, p. 102–114, 2002.

[4] B. WARNEKE, M. LAST, B. LEIBOWITZ AND K. PISTER, « Smart Dust : communicating with a cubic-millimeter computer », *IEEE Computer*, Vol. 34, N° 1, p. 44–51, 2001.

[5] S. TILAK, N.B. ABU-GHAZALEH, W. HEINZELMAN, « A taxonomy of wireless micro-sensor network models », *Mobile Computing and Communications Review*, Vol. 6, N° 2, p. 28–36,

2002.

[6] MARK A. PERILLO, WENDI B. HEINZELMAN, « Wireless Sensor Network Protocols », *Algorithms and Protocols for Wireless and Mobile Networks, Eds. A. Boukerche and al., CRC Hall Publishers*, 2004.

[7] K. SOHRABI, J. GAO, V. AILAWADHI, G. J. POTTIE, « Protocols for Self-Organization of a Wireless Sensor Network », *IEEE personal communications*, Vol. 7, N° 5, p. 16–27, 2000.

[8] C. INTANAGONWIWAT AND R. GOVINDAN AND D. ESTRIN, « Directed Diffusion : a scalable and robust communication paradigm for sensor networks », *ACM Press*, p. 56–67, 2000.

[9] B. WARNEKE, K.S.J. PISTER, « MEMS for Distributed Wireless Sensor Networks », *9th International Conference on Electronics, Circuits and Systems, Croatia*, Vol. 1, p. 291–294, 2002.

[10] SAFFRE, FABRICE, TATESON, RICHARD, GHANEA-HERCOCK, ROBERT, « Reliable sensor networks using decentralised channel selection », *Computer Networks, Elsevier*, Vol. 46, N° 5, p. 651–663, 2004.

[11] S. FAYE, J. F. MYOUPO, « An Ultra Hierarchical Clustering-Based Secure Aggregation Protocol for Wireless Sensor Networks », *AISS : Advances in Information Sciences and Service Sciences*, Vol. 3, N° 9, p. 309 – 319, 2011.

[12] A. PERRIG, R. SZEWCZYK, V. WEN, D. CULLER, J.D. TYGAR, « SPINS : Security protocols for sensor networks », *Wireless networks*, Vol. 8, N° 5, p. 521–534, 2002.

[13] C. KARLOF, N. SASTRY AND D. WAGNER, « TinySec : A Link Layer Security Architecture for Wireless Sensor Networks », *in : Proc. of the 2nd international conference on Embedded networked sensor systems*, ACM, p. 162–175, 2004.

[14] D. WEI AND S. KAPLAN AND H. A. CHAN, « Energy Efficient Clustering Algorithms for Wireless », *in : Sensor Networks, Proceedings of IEEE Conference on Communications, Beijing*, IEEE, p. 236–240, 2008.

[15] PUJOLLE, GUY, « Les réseaux : Edition 2014 », *Editions Eyrolles*, 2014.

[16] WELSH, DOMINIC JA, POWELL, MARTIN B, « An upper bound for the chromatic number of a graph and its application to timetabling problems », *The Computer Journal, Br Computer Soc*, Vol. 10, N° 1, p. 85–86, 1967.

[17] BRÉLAZ, DANIEL, « New methods to color the vertices of a graph », *Communications of the ACM*, Vol. 22, N° 4, p. 251–256, 1979.

[18] ZYKOV, ALEXANDER ALEKSANDROVICH, « On some properties of linear complexes », *Matematicheskii sbornik, Russian Academy of Sciences, Steklov Mathematical Institute of Russian Academy of Sciences*, Vol. 66, N° 2, p. 163–188, 1949.

[19] CORNEIL, DEREK G., GRAHAM, BRUCE, « An algorithm for determining the chromatic number of a graph », *SIAM Journal on Computing, SIAM*, Vol. 2, N° 4, p. 311–318, 1973.

A. Practical example for determination of priority clusters

With the example of figure 5, the determination of priority clusters is performed as follow :

Lists construction :

$L[0] = \emptyset$; $L[1] = [(1,1),(1,2),(1,3),(1,4)]$; $L[2] = [(2,1),(2,4),(2,5)]$

Extraction of clusters sublists which follow :

$L_s[0] = \emptyset$; $L_s[1] = [(1,1),(1,2),(1,3),(1,4)]$; $L_s[2] = [(2,4),(2,5)]$

The longest sub-list : $L_{sp} = L_s[1] = [(1,1),(1,2),(1,3),(1,4)]$

Calculation of the coordinates (x, y) :

$x = 1 - 1 = 0$ and $y = floor(Average(1, 2, 3, 4)) = floor(2.5) = 2$
Filling stategic clusters : The cluster $(0, 2)$ is free, an actuator should be sent into it. By repeating the process : We should put an actuator (if there are available) in the cluster $(0, 4)$ ...

B. Practical example of moving a mobile sensor

Let's move an actuator from the cluster $A(1, 1)$ to the cluster $B(0, 3)$. For our tests, let's suppose : the scope of the sink is $30.0m$ and the virtual architecture includes 3 coronas with $e = 10.0m$ each ; there are 8 angular sectors of $\alpha = \frac{\pi}{4}$ rad each.

Polar Coordinates : A(15.0, 1178 rad) and B(5.0, 2.749 rad)

Cartesian coordinates : $A(13858, 5740)$ and $B(1913, 4619)$

Distance : $p = ||\overrightarrow{AB}|| = 15.811$

Displacement angle : $B'_x = B_x - A_x = -11.945$ and $B'_y = B_y - A_y = -10.359$, then $\alpha_0 = 0.856$ rad. Since $B'_x < 0$ and $B'_y <= 0$ then B is in sector 2 and thus, $\varphi = \pi + \alpha_0 = 3.998$ rad or $\varphi = 229.065^{\circ}$.

Conclusion : To strengthen the area $(0, 3)$, the sink node asks the actuator of the cluster $(1, 1)$ to cover a distance $p = 15.811m$ with an angle of $\varphi = 229,065^{\circ}$.

C. Algorithms

Lexicon of parameters and functions

Parameters	**Description**
$numberofneighbours$	Numbers of neighbors in a cluster including those at two hops
$NumberOfFreq$	Contains the number of channels we need in our network. The frequency range will then be divided by this number
V_{c2}	Neighbors of cluster 'c' including those at two hops
$NeighbouringFreqOfC$	Set of all the indices of the channels used in the vicinity of a cluster
$neighbouringFreqs(L)$	Function that returns the indexes of the channels used by the clusters passed in parameter (L)
$UsableFreqForC$	Indexes of unused channels in the vicinity of a cluster
$cluster.freq$	Denotes the index of the channel used by this cluster
$NumberOfFreq.size()$	Returns the current number of channels required
$first(L)$	Returns the first item in list L
$amplitude$	Amplitude of a channel
$Stop_freq$	End of the range of frequencies allocated to the network
$Start_freq$	Start of the range of frequencies allocated to the network
$neighbourOf(cluster)$	Function that returns the neighbors of the cluster passed in parameter
$UseFreq(amplitude, freq, neighbourFreqs, dest)$	Message that allows to communicate to a cluster the index of the channel on which to communicate ($freq$), as well as the indexes of the channels of its neighbors ($neighbourFreqs$). $amplitude$ denotes the amplitude of the channel, ($dest$) is the destination cluster.
$msg()$	Function that allows to build several types of messages
$Radio.tansmit(r_i, f_0, UseFreq)$	Function that allows the sink to broadcast a message (for example $UseFreq$) on a channel (for example, on the channel of index f_0) and within a radius (r_i)

Parameters	**Description**
$Radio.tansmit(message, freq)$	Function which allows a sensor to broadcast a message on a channel of indexe freq. The coverage radius is that of the sensor.
$UseFreq.dest$	Allows to extract the coordinates of the recipient of a message
$UseFreq.neighbourFreqs$	Denote frequency indexes used by neighboring clusters
$numberOfHops$	Number of hops
$Detect$	Message allowing different clusters to identify themselves
$EndTime$	End date of the empty cluster detection algorithm
AKC	Message acting as acknowledgment of receipt
$relay_cluster$	Cluster through which the CH sends messages to the sink
$relay(x, y, z)$	Table that allows the sink to have the coordinates of the relay cluster of the cluster of coordinates (x,y,z)
$h(x, y, z)$	Allows the sink to know whether the cluster of coordinates (x, y, z) is empty or not
$are_neighbours(c_1, c_2)$	Function which makes it possible to know if two clusters c_1 and c_2 are neighbors or not
τ	The duration of a slot
E_r	Residual energy of a sensor
E_s	Threshold energy of a sensor
$Detect_timerNotReceived$	Boolean that lets you know if a sensor has received a $Detect_timer$
$time()$	Function that returns the local time of a sensor
$Detect_timer$	Message that indicates to the sensors the date on which the CH will be elected
$fork_launch_CH_election$	Sub-process that effectively replaces an active wait that consumes enough energy. The latter schedules the launch of the CH election.
$fork_send_ACK$	Sub-process that effectively replaces an active wait that consumes enough energy. The latter schedules the sending of an acknowledgment of receipt.
$node(ID)$	Returns a sensor from its ID
$Head1$	Message used during the election of the CH to choose the sensor closest to the sink in terms of number of hops.
$Head2$	Message used during the election of the CH to choose the CH among the sensors closest to the sink in terms of number of hops and having the greatest residual energy (E_r).
$Head1MessageNotReceived$	Boolean which allows to know if a sensor has previously received a $Head1$ message.
$Head2MessageNotReceived$	Boolean which allows to know if a sensor has previously received a $Head2$ message.
$cluster_head$	Local reference of the sensor playing the role of CH.

Algorithm 1: Channel allocation to the clusters.

Input: A 2 dimensions table of clusters of size $c \times s$.
Output: $UseFreq$ messages giving the communication channel of each cluster.

Put the clusters of the table in a list L;
Sort L in descending order of $numberofneighbours$;
$NumberOfFreq \leftarrow \emptyset$ /* NumberOfFreq contains the number of necessary frequencies */
foreach $c \in L$ **do**
    $V_{c2} \leftarrow \emptyset$;
    $NeighbouringFreqOfC \leftarrow \emptyset$ /* frequencies' indexes of the clusters neighbouring to the cluster c */
    $V_{c2} \leftarrow neighbourOf(c) - \{c\}$ /* V_{c2} : neighbours of c, including those at two hops */
    $NeighbouringFreqOfC \leftarrow neighbouringFreqs(V_{c2})$;
    $UsableFreqForC \leftarrow NumberOfFreq - NeighbouringFreqOfC$;
    if $(UsableFreqForC = \emptyset)$ **then**
        $c.freq \leftarrow NumberOfFreq.size() + 1$;
        $NumberOfFreq \leftarrow NumberOfFreq \cup \{NumberOfFreq.size() + 1\}$;
    else
        $c.freq \leftarrow first(UsableFreqForC)$ /* first(L) returns the first element of the list L */
/* Distributing the frequencies to the clusters */
$amplitude \leftarrow \frac{Stop_freq - Start_freq}{NumberOfFreq.size()}$;
foreach $c \in L$ **do**
    $dest \leftarrow c$;
    $freq \leftarrow amplitude \times c.freq$;
    $neighbourFreqs \leftarrow neighbourFreqs(neighbourOf(dest))$;
    $UseFreq \leftarrow msg(amplitude, freq, neighbourFreqs, dest)$;
    $Radio.tansmit(r_i, f_0, UseFreq)$ /* Transmit the message M on the channels of the neighbouring clusters */

Algorithm 2: Reception of a message $UseFreq$.

Input: A message $UseFreq$.

/* If the message is for my cluster */
if $((i, j, k) = UseFreq.dest)$ **then**
    $freq \leftarrow UseFreq.amplitude \times UseFreq.freq$;
    $neighbourFreqs \leftarrow UseFreq.neighbourFreqs$;

Algorithm 3: Sink's algorithm.

Input: The duration τ of a slot.
Output: The tables $h(c, s)$ **and** $relay(c, s)$ filled.

```
1  sender ← (−1, −1, −1);
2  numberOfHops ← 0;
3  Detect ← msg(sender, numberOfHops);
4  NumFreq ← neigbouring clusters' frequencies list;
5  foreach (freq ∈ NumFreq) do
6      Radio.transmit(r_1, freq, Detect);
7  EndTime ← time() + 6 × τ;
8  while (time() < EndTime) do
9      if ((ACK ← radio.receive()) == true) then
10         EndTime ← time() + 2 × τ;
11         (x, y, z) ← ACK.sender;
12         if (h(x, y, z) == 0) then
13             h(x, y, z) ← 1;
14             relay(x, y, z) ← ACK.relay_cluster;
15 return the tables h(c, s) and relay(c, s) filled;
```

Algorithm 4: Reception of a $Detect$ message.

Input: A $Detect$ message.
Output: Another $Detect$ message.

if $(are_neighbours((i,j,k), Detect.sender)$ **and** $(E_r > E_s))$ **then**
  /* If i have not received a $Detect_timer$ message that means i am the first receiving the $Detect$ message. I build and broadcast a $Detect_timer$ */
  if $(Detect_timerNotReceived$ **and** $time() < End_Date)$ **then**
    $End_Date \leftarrow time() + (1 - \frac{1}{numberOfHops} + \frac{e^{(-ID)}}{\Lambda}) \times \tau$;
    $dest \leftarrow (i,j,k)$;
    $Detect2 \leftarrow msg((i,j,k), Detect.numberOfHops)$;
    foreach $(freq \in Frequences_voisines)$ **do**
      $radio.transmit(freq, Detect2)$;
    $Detect_timer \leftarrow msg(End_Date, dest, Detect.numberOfHops)$;
    $radio.transmit(dest.freq, Detect_timer)$;
    $numberOfHops \leftarrow Detect.numberOfHops + 1$;
    $relay_cluster \leftarrow Detect.sender$;
    $Detect_timerNotReceived \leftarrow false$;
    /* We shedule the starting of the cluster-heads election algorithm */
    $fork_launch_CH_election \leftarrow forkAt(End_Date, Election_CH)$;
    /* After the cluster-heads election, we send an ACK message to the sink */
    $fork_send_ACK \leftarrow forkAt(4 \times \tau + time(), Send_ACK)$;
  else
    if $((time() < End_Date)$ **and** $(Detect.numberOfHops + 1 < numberOfHops))$ **then**
      $relay_cluster \leftarrow Detect.sender$;
      $numberOfHops \leftarrow Detect.numberOfHops + 1$;
      /* We shedule the starting of the cluster-heads election algorithm */
      $fork_launch_CH_election \leftarrow forkAt(End_Date, Election_CH)$;
      /* After the cluster-heads election, we send an ACK message to the sink */
      $fork_send_ACK \leftarrow forkAt(4 \times \tau + time(), Send_ACK)$;

Algorithm 5: After the cluster-head election.

Output: An ACK message.

/* If i am the elected cluster-head, i send an ACK message */
if $(cluster_head = node(ID))$ **then**
  $sender \leftarrow (i,j,k)$;
  $dest \leftarrow (-1,-1,-1)$;
  $ACK \leftarrow msg(sender, dest, relay_cluster)$;
  $radio.transmit(relay_cluster.freq, ACK)$;

Algorithm 6: Distributed election of cluster-heads.

Input: Locale energy threshold E_s.

if $((numberOfHops! = \infty)$ **and** $(E_r > E_s))$ **then**
  $time \leftarrow (1 - \frac{1}{numberOfHops} + \frac{e^{(-ID)}}{\Lambda}) \times \tau$;
  $dest \leftarrow (i, j, k)$;
  $Head1 \leftarrow msg(numberOfHops, E_r, ID, relay_cluster, dest)$;
  $sleep(time)$;
  if $(Head1MessageNotReceived$ **and** $isFreqFree())$ **then**
    $radio.transmit(freq, Head1)$;
  $time \leftarrow (\frac{1}{E_r} + \frac{e^{(-ID)}}{\Lambda}) \times \tau$;
  $Head2 \leftarrow msg(numberOfHops, E_r, ID, relay_cluster, dest)$;
  $sleep(time)$;
  if
  $(Head2MessageNotReceived$ **and** $isFreqFree()$ **and** $numberOfHops \leq Head1.numberOfHops)$ **then**
    $radio.transmit(freq, Head2)$;
    $cluster_head \leftarrow node(ID)$;

Algorithm 7: Reception of a message Head1.

Input: A message $Head1$.

if $(Head1.dest = (i, j, k))$ **then**
  $cluster_head \leftarrow Nil$;
  $Head1MessageNotReceived \leftarrow false$;

Algorithm 8: Reception of a message Head2.

Input: A message $Head2$.

if $(Head2.dest = (i, j, k))$ **then**
  $Head2MessageNotReceived \leftarrow false$;
  $cluster_head \leftarrow node(ID)$;
  $numberOfHops \leftarrow Head2.numberOfHops$;
  $relay_cluster \leftarrow Head2.relay_cluster$;