Automatic and dynamic network establishment for linear WSNs

Moussa Déthié Sarr, François Delobel, Michel Misson, Ibrahima Niang

Automatic and Dynamic Network Establishment for Linear WSNs

Moussa Déthié Sarr[a,b], François Delobel[a], Michel Misson[a] and Ibrahima Niang[b]
[a]Clermont Université / LIMOS CNRS - Complexe scientifique des Cézeaux, 63172 Aubière cedex, France
[b] Laboratoire d'Informatique, Université Cheikh Anta Diop de Dakar (UCAD), B.P. 5005 Dakar-Fann, Sénégal
Emails: {sarr,delobel,misson}@sancy.clermont-universite.fr, iniang@ucad.sn

Abstract

The deployment of Wireless Sensor Networks (WSNs) in the mostly linear large structures, such as rivers, pipelines, etc, suffers from being the worst case for classical addressing schemes, e.g. Distributed Address Assignment Mechanism (DAAM) or stochastic addressing for Zigbee. Using DAAM for physical topologies composed of long lines of nodes connected together wastes addresses and generates orphan nodes.

We show in this paper the inherent limitations of classical (DAAM, stochastic) and specialized (usually cluster-orientated) addressing schemes for *Linear* WSNs. *DiscoProto*, is an addressing and routing scheme which builds a logical network corresponding to a corresponding physical linear network without any knowledge of physical topology. In this paper, we show thanks to a realistic simulation using Castalia (Omnet based simulator) that *DiscoProto* avoids waste of addresses and allows a high association ratio. We also propose a dynamic version of our protocol called *Dynamic DiscoProto* in the second part of the paper. *Dynamic DiscoProto* allows to add new nodes or new branches in an existing linear network.

Index terms— Linear Wireless Sensor Networks Addressing Routing Automatic Deployment Topology Discovery Clustering

1 Introduction

Wireless sensor networks (WSNs) are networks formed by a collection of sensor nodes with limited resources (power, bandwidth, memory, CPU and communication range) [1, 2]. Because sensor nodes used to perform these applications have limited resources, the main challenges in WSNs are scalability, self-organization and low energy consumption.

WSNs are used to monitor many applications fields (rivers, health monitoring [3], volcanoes monitoring [4], railways [5], bridge structure [6]).

Most applications are based on the logical topologies defined in WSNs standards (star, mesh, cluster-tree) [7, 8].

However, existing network formation protocols and nodes addressing assignment schemes [9, 2, 8] proposed in WSNs[1] are not suitable for applications monitoring pipelines, rivers, bridges, tunnels or railway structures because these protocols assume that the nodes can be easily organized in a mesh or cluster-tree logical topology. In fact, these application fields imply a mostly linear deployment [10, 11, 12] to monitor a mostly linear infrastructure.

Many classical WSNs addressing techniques are inefficient in this case because linear structures are often the worst case of their addressing techniques (by opposition to a well-balanced tree or a stochastic deployment). Moreover, in large cluster-trees or linear networks, it is difficult to provide an addressing mechanism covering all nodes because of the limitation due to the network parameters (Cm, Rm, Lm), the small (16-bits) size of logical addresses and the waste of address. Stochastic addressing has high probability to generate address conflicts on large networks and requires large routing tables.

To avoid those limitations, specialized protocols and mechanisms are proposed in [11, 5, 12], we showed in [13] the drawbacks of these protocols and mechanisms for large linear WSNs.

[1]Low-Power WAN such as LoRaWAN, Sigfox or Weightless could be a good solution to monitor such infrastructures if the nodes are in line-of-sight with the gateways, which is not the case for tunnels, rivers in canyons..

The goal of our paper is to propose a robust network formation mechanism not only for linear but also for partially linear wireless network topologies[2]. This mechanism provides:

- Linear network formation functionality without any prior knowledge of physical topology, without limitations of the logical topologies and without manual configuration of any kind,
- Efficient hierarchical routing and addressing based on exact size of subtrees,
- Dynamic node management after network formation phase (adding or removing nodes, or whole branches), preserving most of the benefits of hierarchical routing,
- Address block reallocation to ensure the reliability of LWSNs in harsh environment to keep the mechanism robust.

The paper is organized in the following manner. In section II, we present the related works in WSNs. In section III we discuss of the state of the art of addressing mechanisms suitable for linear networks in ZigBee cluster-tree. We introduce our automatic discovery of topology and addressing scheme for linear WSNs in Section IV. In section V, we present our proposition, a *Dynamic DiscoProto* mechanism. In section VI, we describe our implementation in Castalia simulator. Section VII presents the results of our simulation and section VII conclude this paper with a summary of our contribution and perspectives of future works.

2 State of the art

2.1 Cluster-tree network formation in ZigBee

IEEE 802.15.4 standard [7] introduces two different categories of nodes. A Full Function Device (FFD) which is a sensor node with router capability and a Reduce Function Device can only be used as end-device node

ZigBee [8] defines three roles of nodes: a ZigBee Coordinator, ZigBee Routers (ZR) that participates in routing messages for mesh and cluster-tree topologies and ZigBee End Devices (ZED) to provide sensing data capabilities. FFD devices can be used as ZC, ZR or ZED nodes. However, RFD devices can only be used as ZED nodes.

In ZigBee cluster-tree network formation [8, 14], the parent node, for every new child to associate, chooses an address and assigns it to its new child. A PAN coordinator uses parameters such as the maximum number of children per node (Cm), the maximum number of children routers (Rm), and the maximum tree depth (Lm) to perform a Distributed Address Assignment Mechanism (DAAM) and to guarantee a unique and hierarchical addressing. For this, each parent node uses the $Cskip$ function to calculate the size of address sub-block depending on its depth (d) in the network.

In this cluster-tree network, the network address block is calculated with the Equation (1)):

$$Amax = Cskip(0) \times Rm + Cm - Rm \tag{1}$$

Where:

$$Cskip(d) = \begin{cases} 1 + Cm \times (Lm - d - 1), \text{ if } Rm = 1 \\ \frac{1+Cm-Rm-Cm\times Rm^{Lm-d-1}}{1-Rm}, \text{ otherwise} \end{cases}$$

When a ZED node joins the network and becomes the n^{th} child end device of the parent node at the depth (d), its address is calculated with equation (2).

$$A_{child} = A_{parent} + Rm \times Cskip(d) + n \tag{2}$$

However, if it is a ZR node which joins the network and becomes the n^{th} child router of the parent node at depth (d), then its address is calculated with equation 3:

[2]In this article, we consider LWSNs as *mostly Linear Wireless Sensors Networks*, WSNs constituted of long branches of nodes connected together.

$$A_{child} = A_{parent} + 1, \qquad \text{if } n = 1 \tag{3}$$
$$A_{child} = A_{parent} + (n-1) \times Cskip(d), \quad \text{if } n > 1$$

Cm, Rm, Lm parameters have to be manually defined before the construction phase of the network according to its shape and strongly impact the network formation process. Those values can be optimally chosen only for perfectly balanced trees, with the same number of sons for each node. In fact, in real situation, it is difficult to predict good values for parameters Cm, or Lm. Poorly chosen values causes problems of non-associated nodes (orphan nodes) in large networks. In [15, 16], the authors propose a mechanism to reduce the number of non-associated nodes.

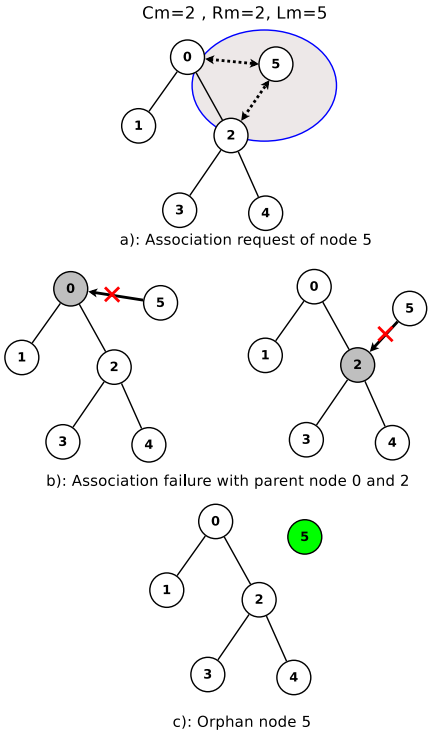


Figure 1: DAAM problem

On the other hand, with DAAM, it is difficult to expand network area particularly in Linear WSNs. Indeed, if we consider trees with high depth (close to linear), the addressing space of ZigBee ($2^{16} = 65535$), limits the possible configurations (Cm, Rm, Lm) to $(2, 2, 15)$, $(3, 3, 9)$,$(4, 4, 7)$. In linear WSNs, we consider topologies whith depths that can exceed 15 hops.

2.2 DIstributed Borrowing Addressing Scheme for ZigBee/IEEE 802.15.4 Wireless Sensor Networks

The authors in [15] propose the Distributed Borrowing Addressing (DIBA) mechanism to solve association failure in distributed address assignment mechanism used in ZigBee. All router nodes in this network broadcast beacon frames including the *Available Address Count* (AAC), which is the number of children that the routers can add.

A new node joining the network chooses the parent node which has the biggest AAC. However, if the parent node has no available address or if its tree depth is Lm, it uses the *address borrowing mechanism* to obtain an address for the new node : When the parent node P receives an *Association Request* from a new node and has reached the limit of Cm, it broadcasts an *Address Borrowing Request* to borrow addresses from its neighbor nodes. Some of them responds with an *Address Borrowing Response* including Available Address (AA) and AAC. The node P that sends the request chooses an AA with the biggest AAC. If multiple nodes with same AAC are found, P selects the node with

the highest AA. The neighbor node called LN (Lender Node) that has received a request message calculates the appropriate AA to lend by the following Equation (4), where num_BA is the number of borrowed address.

$$AA = \quad Addr_{LN} + (Rm - num\_BA) \times Cskip(d) + 1, \quad num\_BA < Rm \tag{4}$$

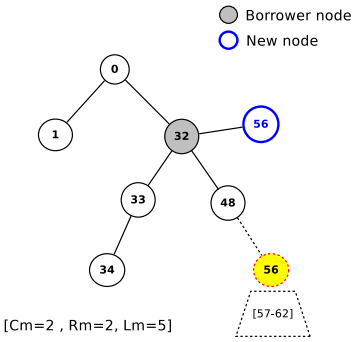


Figure 2: DIBA mechanism on cluster-tree WSNs

If the incoming node receives an available address for lending (called AAL), it also reserves the address block $[AAL, AAL + Cskip(d) - 1]$, where d is the depth of the node lending the address block (see Figure 2).

Routing table of node 32

	Address	Lender	Borrower
Parent	0	-	-
Child 1	33	-	-
Child 2	48	-	-
Child 3	56	48	-

Routing table of node 48

	Address	Lender	Borrower
Parent	32	-	-
Child 1	56	-	32

Table 1: Routing tables of some nodes shown in Figure 2

In dense networks, *Address Borrowing Response* messages cause many collisions and can result in association failures. In addition, when all available addresses have been borrowed in 1-hop neighbors, the candidate node can no longer be associated because multi-hop borrowing is not allowed in DIBA. Moreover, DIBA increases the complexity of routing in the case of numerous address borrowing because purely hierarchical routing becomes insufficient (see Table 1).

2.3 Address Assignment and Routing Schemes for ZigBee-Based Long-Thin WSNs

In [12], the authors consider a special case of ZigBee cluster-tree where each cluster is composed of nodes deployed on line segment. In addition, each node in the Long-Thin WSNs (LTWSNs) has router capabilities and two special nodes need to be defined (see Figure 3):

- The *cluster head* node is the ZigBee router node connected to the *cluster bridge* node of the parent cluster.

- A *cluster bridge* node is the ZigBee router node connected to the cluster head node of the child cluster.

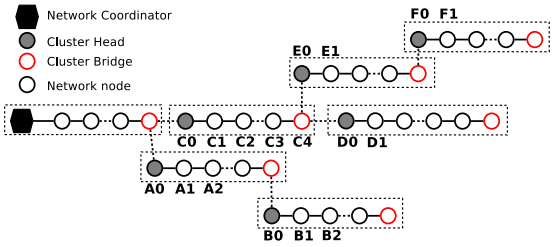


Figure 3: Role assignment in a LT-WSN

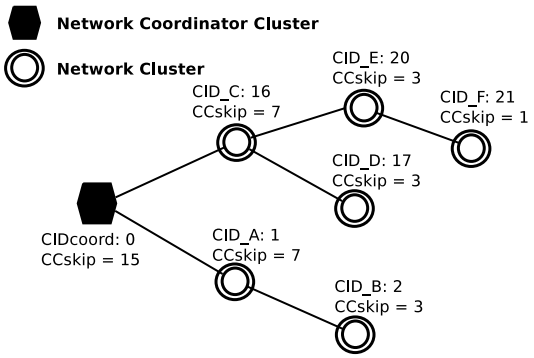


Figure 4: A Logical Network of LT WSNs in Figure 3

In order to deploy this mechanism, the head and bridge of each cluster have to be manually defined. Then, a logical network is formed (see Figure 4) where each cluster in a physical network is represented by a node in a logical topology. For the node address assignment, the authors divide the 16-bit network address in two parts: The first part of m bits is used for cluster addressing and the second part of $16 - m$ bits for node addressing in a cluster.

Then, as in ZigBee Cluster-Tree Addressing, two parameters are fixed: CLm (Cluster maximum Length) and CCm (Cluster maximum Children) to compute the $CCskip$ function defining the starting Cluster ID (CID) of each cluster in a logical network.

- if $CCm > 1$

$$CCskip(d) = \frac{1 - CCm^{CLm-d}}{1 - CCm}, \tag{5}$$

Each cluster at depth d in the logical network assigns Cluster ID to its n^{th} cluster child using Equation 6

$$CID_{Child} = \begin{array}{l} CID_{Parent} + (n-1) \times CCskip(d) + 1, \\ \text{if } CCm >= 2 \end{array} \tag{6}$$

- If $CCm = 1$, the logical network is a segment line with no branch and cluster addresses are assigned sequentially.

After the CID assignment, each node in the cluster starts a network setup sending *HELLO* messages with 16-bit network address where the CID part is its cluster ID and the $nodeID$ is 0. The network coordinator starts a nodeID address assignment broadcasting *beacon* with $nodeID$

set to 0. If a node u *in the neighborhood is not already associated, it has to respond with an Assoc_Request to the beacon sender having the strongest signal strength. Each beacon sender* v with an address $(CID_v, nodeID_v)$ sorts *Assoc_Request* senders by signal strength in a List **L**. Then it sets $nodeID = nodeID_v + 1$ and assigns sequentially $nodeID_u = nodeID_u + 1$ to *Assoc_Request* senders having a same CID. After assigning $nodeID$ for all received requests, the beacon sender either delegates this role of beacon sender to the bridge node or to the last node in **L**.

This approach suffers the same limitations as DAAM: The maximum number of clusters that can be addressed is 2^m and the maximum addressable node is $2^{(16-m)}$. In addition, the network depth CLm is limited by the chosen parameter m (see the example in Equation (7)).

$$\begin{gathered} \text{for } CCm = 2, \\ Amax \leq 2^m \Rightarrow 2 \times (2^{CLm} - 1) \leq 2^m \\ CLm \leq \frac{\ln(2^{m-1}+1)}{\ln 2} \end{gathered} \tag{7}$$

Finally, the head and bridge of each cluster have to be manually selected during the network deployment. Therefore, human intervention is also necessary to elect another node as cluster-head or bridge when one of this node fails. This manual selection method of the cluster-head or bridge can cause a network partitioning. In fact, all traffic of clusters is routed by a cluster-head to bridge node of its parent cluster.

3 Automatic Discovery of Topology and Addressing for Linear WSNs

In [13], the authors propose a discovery and a network formation mechanism adapted for linear network environment. We confirm this work with a realistic simulation and extend it to become dynamic.

3.1 Initial formation of network

[13] performs the network setup without knowledge of the topology parameters (maximum depth, maximum number of children by node, position of the node) and also provides a hierarchical addressing scheme to ensure efficient (and memory-saving) data routing.

In the discovery network phase, [13] propose a distributed algorithm called *DiscoProto* to generate a spanning tree minimizing several criteria. For this, let us consider in a graph a set of connected nodes CN and a set of unconnected nodes UN. Let $c \in CN$ and $u \in UN$. *DiscoProto* incrementally constructs a connected network with the best link (c, u) having the *local* best cost. Note that the cost being a *local* optimum[3], several nodes can be added at the same time, providing they are far enough not to interact. So the formation phase of the network is actually done in parallel.

The choice of the best association father/son depends on three criteria (cost) sorted by importance:

- First, the number of common neighbors with c and u, $common(c, u)$. This parameter represents the proximity of two nodes. Closest noded are associated first.
- Second, the number of sons that the connected node c has already associated, called $sons(c)$. If two connected nodes c_1 and c_2 have the same number of common neighbors, i.e $common(c_1, u) = common(c_2, u)$, u will be associated with the node with fewer sons between $c1$ and $c2$. This criterion ensures a good linearity of the network and keeps it as close as possible to the physical topology.
- Third, the sum of neighbors of c and u. If two nodes u_1 an u_2 have same two first criteria, the node with low connectivity will be associated first (*fail first* heuristic, as we will use a distributed greedy algorithm).

[3]The distributed algorithm is a relaxation of the non-distributed Prim-Jarnik algorithm and best costs are only compared in a portion of the graph.

Those values are aggregated together as a single *objective value* by the following equation:

$$\begin{aligned} objective(c,u) \quad &= \alpha \times \text{common}(c,u) - \text{sons}(c) \\ &\quad -\beta \times (\text{neighbors}(c) + \text{neighbors}(u)), \\ &\quad \text{with } \alpha = 10 \text{ and } \beta = \tfrac{1}{1000}. \end{aligned} \tag{8}$$

To discover a linear network and to associate all nodes, each node goes thru five different states.

3.1.1 Neighborhood Discovery State

All awaken nodes broadcast *Hello* messages. Nodes that receive these messages update their neighbor tables. This step is important because computing the *objective value* for an association requires the knowledge of the neighborhood.

3.1.2 Wait for Father State

After neighborhood discovery, all nodes stay in this state until receiving a *FatherOffer* message, and then go into next step.

3.1.3 Son Collection State

In this state, connected nodes, i.e. nodes that are already associated, start to broadcast *FatherOffer* messages in the neighborhood. This message contains information needed to compute the objective value of the association (number of neighbors and number of sons).

Each unassociated neighbor that received *FatherOffer* messages computes the association *objective value* with the given function (see Equation (8)). Then, it answers to the *FatherOffer* sender with unicast *SonOffer* message containing the objective value.

If no *SonOffer* messages are received from potential parents until the expiration of the timer for collect offers, connected nodes can retry sending *FatherOffer* message up to three times.

When the timer of a connected node sending *FatherOffer* expires, it selects the best *SonOffer* in its neighborhood and broadcasts its value using a *ChallengeOffer* message to the other connected nodes. This challenge is locally propagated (up to three hops) so that only the best father/son association in a neighborhood will be chosen. If a *ChallengeOffer* sender node receives an *ChallengeOffer* with the higher objective value from its neighborhood, then it cancels its potential father/son association and restarts a new *Son Collection*. At the expiration of *ChallengerOffer* timer, nodes that have not cancelled their potential father/son association start the association step.

After the challenge phase in *Sons Collection* state, the winner node sends an *AssociationAccept* message to the node selected as new son. This node answers with an *AssociationAck* and memorizes it as new father. When an *AssociationAccept* sender node does not receive an *AssociationAck* until expiration of the timer, it can retry up to three time before giving up the selected node and restarting a new *Son Collection* state. If a selected node receives an *AssociationAccept* when it is already connected, it sends an *AssociationFailed* to the sender node to cancel the useless association. Therefore, the latter gives up the selected node and restarts a new *Son Collection* state.

A new son node sets its potential father node as new father node and starts a new *Son Collection* state. When a *FatherOffer* sender did not receive *SonOffer* message after three repetitions, it stops collecting sons and goes to the *Wait for SonSize* state.

3.1.4 Wait for SonSize State

This state lasts until all the node's sons have provided the size of their subtree. Then the node computes the value $size = \sum_{i=1}^{n} SonSize(i) + 1$ and sends this value to its father node using *PropaSons* messages. When all nodes have left *Wait for SonSize State*, all the nodes in the network know the size of all their subtrees. A node leaving *Wait for SonSize State* enters the *Wait for Address* state.

3.1.5 Wait for Address State

After receiving all son sizes, the coordinator computes appropriate address block (AB) for the entire network using this following Equation 9.

$$AB = (\sum_{i=1}^{n} SonSize(i) + 1) \times Fskip \tag{9}$$

As the addresses can be smoothly distributed in the network, spare addresses can be uniformly distributed ($Fskip$ coefficient in Equation (9)).

Every node (including the sink) uses *PropaAddr* and *PropaAddrAck* messages to dispatch address blocks to its sons. Each node that receives the *PropaAddr* message containing the range $[a, b]$, chooses a as its own address. Then, it keeps addresses from $[a+1, a+Fskip]$ as available addresses for future association. $Fskip$ is the number of available addresses. The remaining interval $[a + Fskip + 1, b]$ is divided between its N sons proportionally to the size of their subtree.

Figure 5 shows an example of address distribution after the network formation.

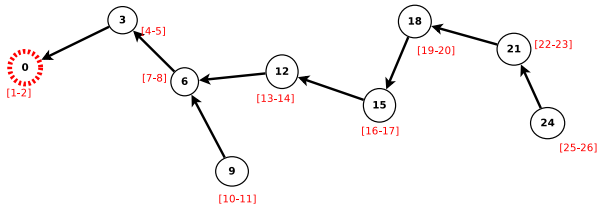


Figure 5: Example of address distribution

start_Block	end_Block	addr_Node	id_Son
9	11	9	4
12	26	12	5

Table 2: Son address table of node 6 in Figure 5

3.2 Routing algorithm

A routing algorithm based on our hierarchical addressing ensures efficient data routing. Each node in the network maintains a son address table containing for each son: its logical and physical addresses and the first and last logical addresses allocated in its subtree (see Table 3).

start_Block	end_Block	addr_Node	id_Son
Logical address	Logical address	Logical address	Physical address

Table 3: Son address table format

The memory used by the routing table is:

$$\begin{aligned} memory\_size \quad &= son(C) \times (3 \times network\_address\_size \\ &\quad + mac\_address\_size) \end{aligned} \tag{10}$$

The worst care complexity of routing is trying all the rules up to the fourth one, with a complexity of numbers of entry.

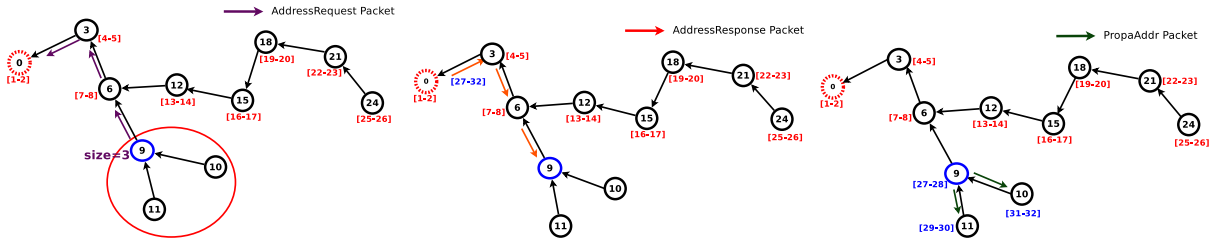
Suppose that the node n with an address $addr$ receives a data packet, it performs the following procedures:

1. If node n is the destination, the packet is accepted and transmitted to the application layer.
2. If not, the node verifies if the destination address belongs to one of the entries of the son address table. If it is found, then the packet is sent to the son node corresponding to the matching entry.
3. Otherwise, the packet is sent to the father node.

4 Dynamic DiscoProto

In this section, we introduce a new mechanism for *DiscoProto* to dynamically add sparse nodes or branches to an existing linear WSNs. We also present an address reallocation mechanism in case nodes have exhaust their spare addresses. We consider that the network is already formed and that all connected nodes have received their own address block.

4.1 Network join procedure



(a) New address block request (b) New address block response (c) Address block propagation

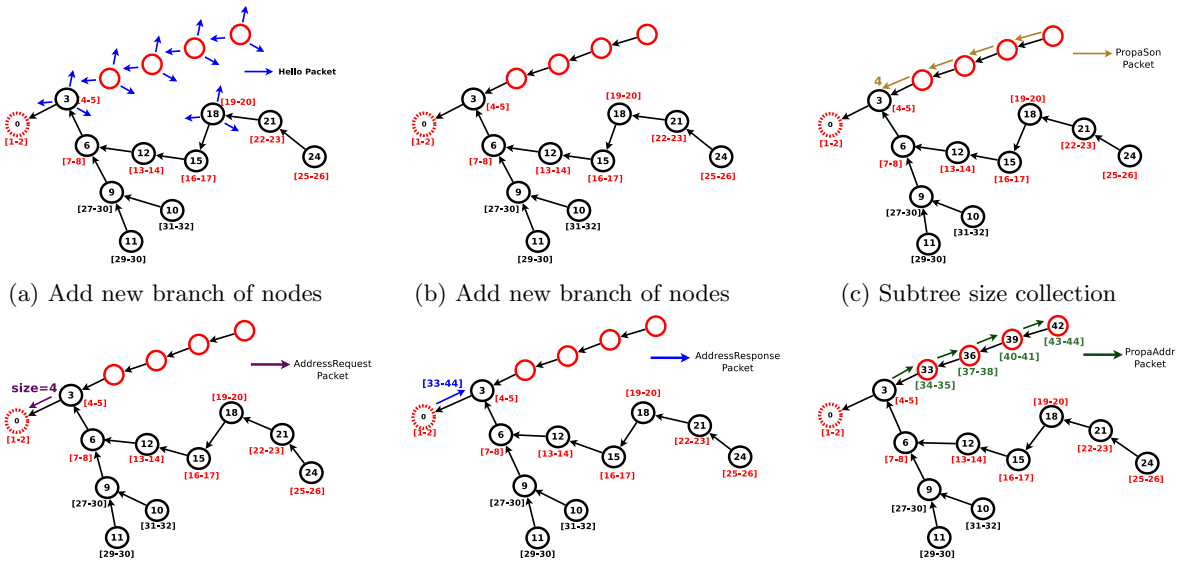
Figure 6: Address block reallocation procedure

A new node that needs to join the network starts by broadcasting *HELLO* messages until expiration of *hello timer* cycle. When an associated node i.e., a connected node with an address, receives a *HELLO* message, it updates its neighbor table and responds by broadcasting *HELLO* messages. At expiration of the *hello timer*, associated nodes which received *HELLO* messages from new node enter *Son Collection* state (described in section 3.1.3). After associating the new node, the selected father node (with logical address *myaddr*) waits until receiving the size N of the new son subtree node before entering *Addresses Propagation* state.

When the selected father receives the size of all its subtrees, it performs the following tasks:

1. If the father node has enough spare addresses (i.e., $N \leq R - nbrNewSon$) then it attributes the first available address $myaddr + nbrNewSon$ to the new son node, where $nbrNewSon$ is the number of new sons connected after the network setup.
2. Otherwise, it sends an *AddressRequest* message to the sink including $N + 1$ (see Figure 6). The sink answers with an *AddressResponse* message including the new address block $[a, b]$ where
   $a = lastEndBlock + 1$
   and $b = lastEndBlock + Fskip \times (N + 1) + 1$.
3. When the father receives the new address block range from the sink, it keeps the address range $[a + 1, a + Fskip + 1]$ for future associations and the remaining address range is dispatched between its newly added sons proportionally to their subtree size using *PropaAddr*.

The join procedure for new branches operates in the same manner as for new spare nodes (see Figure 7). However, the addressing mechanism only starts after the collection son procedure for all nodes in the branch. The first connected node of the branch waits to collect all son size subtree before sending it to its father to prevent useless messages and address block fragmentation. The father node of the new node performs the above procedures to distribute the address block to the new branch.



(a) Add new branch of nodes

(b) Add new branch of nodes

(c) Subtree size collection

(d) New address block request

(e) New address block reallocation

(f) Addresses propagation to newly added nodes

Figure 7: new branch join procedure

4.2 Address block reallocation

In our Linear WSNs, connected nodes can exhaust their available addresses by different manner:

- Connected nodes have distributed all available addresses to new nodes or branches,
- Connected nodes have not enough available addresses for new branches that need to join the network,
- New associated nodes have new candidate nodes or branches that need to join the network

For all these cases, connected nodes send an *AddressRequest* message to the sink and the sink answers with *AddressResponse*. In order to maintain addressing, nodes traversed by the *AddressResponse* add a new entry in their routing tables. This entry contains *start_Block*, *end_Block* and the destination node of the *AddressResponse* (see Table 4).

start_Block	end_Block	addr_Node	id_Son
9	11	9	4
12	26	12	5
27	32	9	4

Table 4: Son address table of node 6 in Figure 6c

start_Block	end_Block	addr_Node	id_Son
6	26	6	3
27	30	9	3
33	44	33	12

Table 5: Son address table of node 3 in Figure 7f

Note that requesting large blocks instead of multiple single addresses avoids bloating routing tables.

4.3 Routing Algorithm

The routing mechanism based on our hierarchical addressing mechanism and the information in the routing table is the same as described in subsection 3.2.

Son address table maintains 16 bits prefix, *stat_Block* and *end_Block* addresses and 64 bits MAC addresses. The total memory size of son addresses table can be calculated as in Equation (11). Let $son(C)$ be the sons number of node C and $newAddrBlock$ the number of times that new blocks are forwarded to the subtree of node C.

$$\begin{aligned} memory\_size &= (son(C) + newAddrBlock) \\ &\times (3 \times network\_address\_size \\ &+ mac\_address\_size) \end{aligned} \tag{11}$$

The Table 5 shows an example of entry of node 3.

5 Implementation in Castalia Simulator

In *Automatic Discovery of Topology and Addressing for Linear WSNs* [13], we validated the topology formation mechanism on a home-made simulator developed in Ruby, using a green thread per node. This environment had no realistic low layer (MAC and physical layer) and perfect propagation: Indeed, every message was received by all nodes located in range.

To validate the protocol from [13], realistic propagation conditions and collision have to be taken into account. Therefore, we propose an implementation of Automatic Discovery of Topology in the WSNs Castalia environment, which has a proper physical and MAC layer (S-MAC protocol [17]).

5.1 Castalia environment

Castalia [18, 19] is a simulator specially designed for WSNs based on OMNet++ platform. It implements advanced channel model and radio model, realistic modeling of RSSI and carrier sensing and it is designed for adaptation and expansion. Castalia provides many path loss models, including the log-normal shadowing model which is well adapted to WSNs [20]. It also has a useful energy model. Physical node parameters are based on real specifications of CC2410 (see Tables 6 and 7) motes.

Name	**normal**
Data rate	250 kbps
Modulation type	PSK
Bandwidth	20 MHz
Noise bandwidth	194 MHz
Noise floor	-100
Sensitivity	-95
Power consumption	62 mW

Table 6: CC2410 RX specifications implemented in Castalia nodes

Tx (dBm)	0	-1	-3	-5
Tx (mW)	52.42	55.18	50.69	46.2

Table 7: CC2410 TX level specifications implemented in Castalia nodes

5.2 Link cost computation

Castalia computes RSSI and LQI values during transmission between nodes in a network. We use this advantage to slightly improve the objective function in Equation (12) to compute link cost in network graph by adding RSSI measurements. If two potentials parents of the same node have the same number of common neighbors, the same number of associated sons, and the same number of neighbors, we select the potential parent with the better RSSI value.

$$\begin{aligned} objective(c,u) \quad &= \alpha \times \text{common}(c,u) - \text{sons}(c) \\ &- \beta \times (\text{neighbors}(c) + \text{neighbors}(u)) \\ &+ \epsilon \times \text{RSSI}(c) \\ &\text{with } \alpha = 100 \ , \ \beta = \tfrac{1}{1000}, \ \epsilon = \tfrac{1}{10000} \end{aligned} \tag{12}$$

Neighborhood discovery state is stopped when the timer expires or when the node receives a *FatherOffer* message from potential parents. During this state, all nodes collect RSSI of their neighbor nodes and put the nodes with the best value of RSSI in their neighbor table.

6 Evaluation and Results

In this section, we evaluate our protocol in Linear Wireless Sensors Networks (LWSNs) using Castalia Simulator.

6.1 Evaluation of initial network formation

The simulation scenarios use different random topologies with 50 to 300 nodes placed in areas of side $1km$ to $6km$. The characteristics of our topology generator are the following.

- Only connected topologies are built.
- The distance between two nodes is $20m \pm 0.2m$, so that each node can reach at least two other nodes in a direction (Physical 2-vertex-connectivity for better network resilience).
- During node placement, the probability to create a branch coming out a node is 10% and the deviation angle is between 40 and 140 degrees or -40 and -140 degrees to avoid parallel lines.
- Flickering effect is between -10 and 10 degrees so that the lines are not perfect.

Figure 8 shows an example of generate linear topology

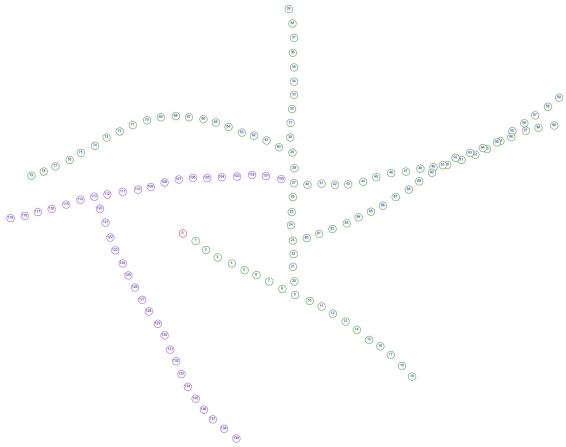


Figure 8: Example of generated linear topology

To evaluate our protocol in Castalia simulator, we run for each network enumerated above 10.000 experiences with simulation parameters in Table 8.

Simulation time (sec)	1800, 3600, 5200, 7200
Numbers of runs	10000
Propagation model	Shadowing (PE=2, 4, sigma=4, 0)
Tx output power	0 dBm

Table 8: Parameters of simulation

After checking the association rates, we will discuss influence of various factors on total association time (time for all the notes to be associated, including timeouts), either in the case of an initial network formation, or for the case of a new branch addition. Then, we will study the overhead linked to control messages and the distribution of the control messages.

6.1.1 Efficiency

The Figure 9 shows the comparison of association rates of our protocol *DiscoProto* and *DAAM ZigBee* for different network sizes. For *DAAM*, we use parameters $Cm = 2$, $Rm = 2$ and $Lm = 15$.

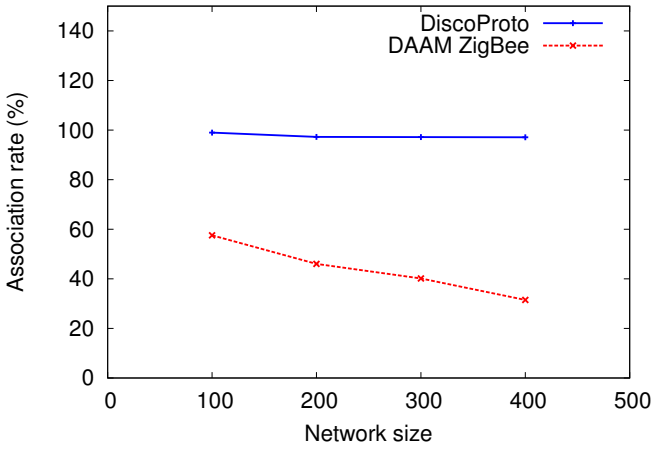


Figure 9: Average association rate by network size

Unlike DAAM, our mechanism associates and provides addresses for almost all nodes of the network node. With DAAM, a lot of orphan nodes appear because of maximum depth limited. Collision problems because of MAC layer in dense linear network prevent message delivery in son collection step and nodes stop sending *FatherOffer* after three attempts. Therefore, the association rate for the initial network is close to 100% but the few orphans nodes have other chances to get associated. But, even if a node misses the association phase, it will try to join network again during the *network join* procedure describes in Section 4.1.

6.1.2 Scalability

Figure 10 represents the average association time by network size. We note that the average time increases sublineary: our protocol has the capacity to perform node association in parallel manner which explains why it scales well.

6.1.3 Influence of number of disjunctions

In Figure 11, we show that the average association time decreases with number of disjunctions until a limit value depending on network size. We call *disjunction* a node in the logical topology with more than one son (usually because a *branch* has been created by the topology generator). At first, the increasing number of disjunctions allows DiscoProto to run in parallel. But, when the density raises,

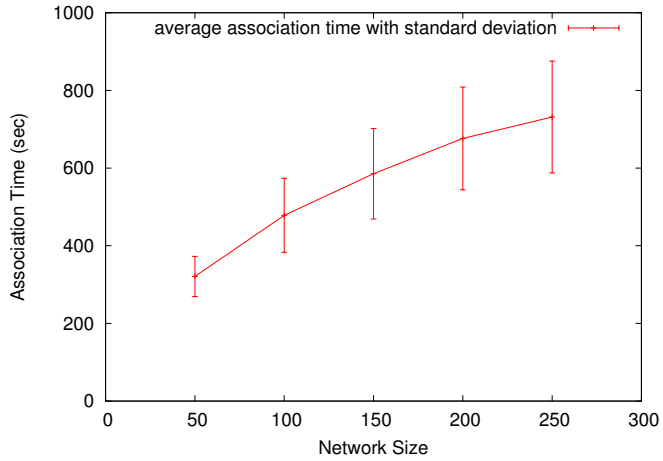
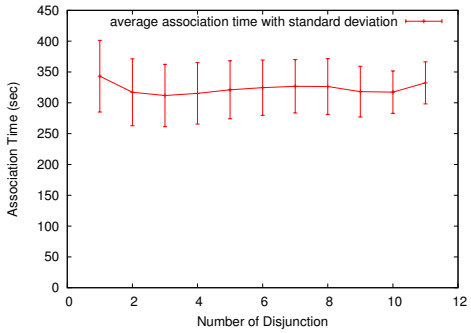
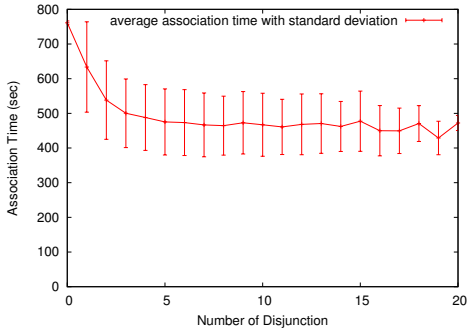


Figure 10: Average association time by network size

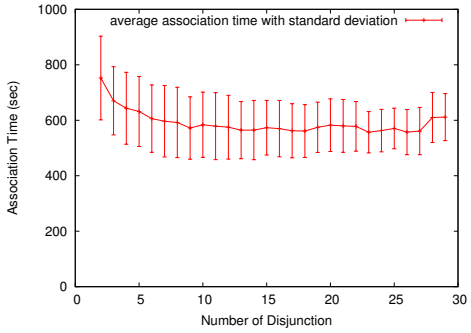
the high number of nodes in the same neighborhood that participate to challenges in each *Collection son* cycle increases the number of association father/son cancelled and slows down the association.



(a) Network of 50 nodes

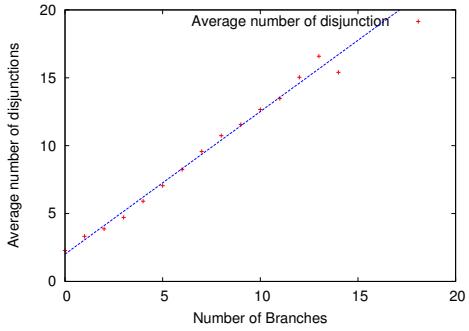


(b) Network of 100 nodes

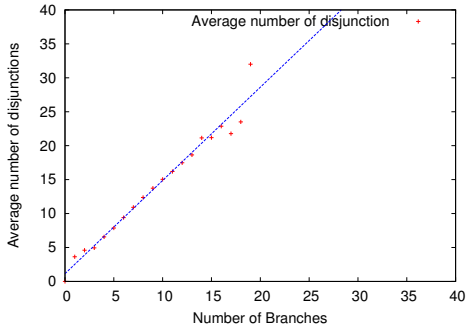


(c) Network of 150 nodes

Figure 11: Average association time by disjunction



(a) Network of 100 nodes



(b) Network of 150 nodes

Figure 12: Average number of disjunction by number of branches

6.1.4 Closeness to physical topology

Figure 12 shows the average number of disjunctions (in the network) with regards to the number of branches actually generated. These figures confirmed the results obtained in [13] for the procedure of discovery and topology generation. In a realistic WSN environment, our implemented protocol setups a linear network as well as in the home-made Ruby simulator with poor physical and MAC layer.

6.2 Evaluation of the cost of addition of new nodes

We study the cost of association of new nodes to an already formed network using *Dynamic DiscoProto*. The metrics we are studying are not only the association time but also the cost in terms of number of association messages. The efficiency of the association is measured in terms of association success (percentage of successfully associated nodes). We study initial networks of various sizes (100, 200, 300, 400, 500 nodes), submitted to associations of several new branches (5, 10, 15, 20 and 25 new branches). Each new branch is 20 nodes long. The nodes of the new branch start up at random time after the end of initial network formation.

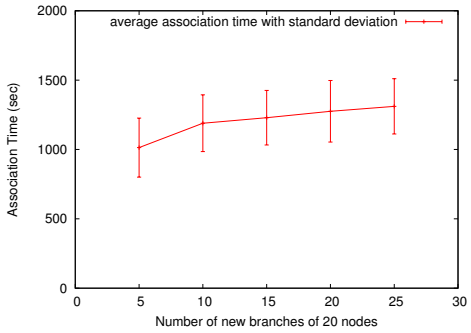


Figure 13: Average association delay of addition of new branches

Figure 13 shows that the number of new branches does not affect exponentially the association delay for the same network size.

6.2.1 Influence of initial network size on association time of new branches

Figure 14 shows that the association delay for new branches is not dependent on the network size.

Table 9 shows that the association rate before and after addition of new branches, with an initial network of 200 nodes. Dynamically adding branches does not lower the association rate (it is even better because orphans nodes of the network had time to dynamically associate afterward).

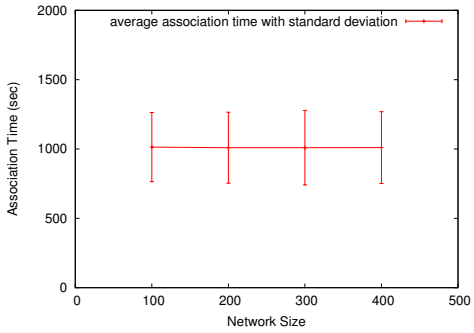


Figure 14: Average association time of 400 new nodes by network size

Nb branches to add	5	10	15	20
Before adding new branches	99.747	99.630	99.687	99.688
After adding new branches	99.955	99.896	99.884	99.860

Table 9: Average association rate vs added branches

6.3 Study of overhead and control message distribution

Figure 15 shows that the number of control messages sent during network building grows up linearly with the number of branches because each potential node responds to each *fatherOffer* message diffusing in the neighborhood. Association messages are *HELLO*, *FatherOffer*, *SonOffer*, *ChallengerOffer*, *AssociationAccept*.

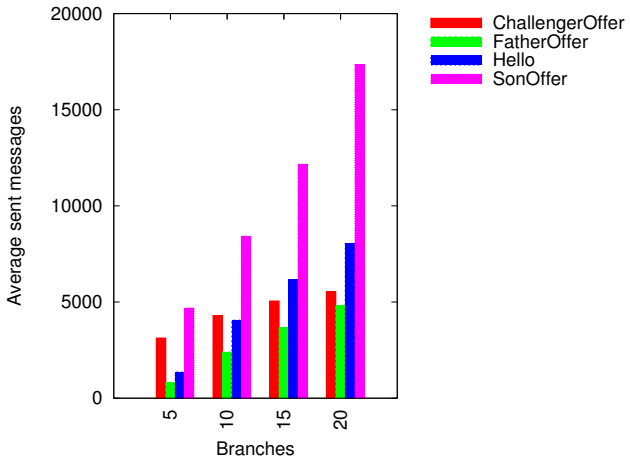


Figure 15: Average number of messages with varying number of branches for a network of 200 nodes

7 Conclusion

In this paper, we evaluate and improve *DiscoProto*, an efficient discovery mechanism for Linear WSNs topologies. In addition, we propose *Dynamic DiscoProto* to support addition of new nodes or new branches to an existing network while keeping the routing nearly hierarchical. The evaluation of our protocol in the Castalia simulator shows that it can be adapted for large Linear WSNs but can cause latency in dense WSNs with many overlapping branches. The implemented routing algorithm supports new address bloc reallocation and ensures efficient data routing for entire network.

The next step is to implement a mechanism to detect and repair failing links (both upward and downward). The idea is to define different procedures used by a node to adopt another father without many messages exchanged and to ensure a fast convergence of routing tables. Another aspect we plan to study is to find the optimal number of hops for broadcasting *ChallengerOffer* message in the neighborhood and, more generally, the impact of different constants (timers of different messages) used in the Sons Collection Step on the association rate and average association time. We also plan to evaluate our protocol with MAC protocol dedicated to Linear WSNs proposed by authors in [21].

For the research perspective in the future, we plan to study the deployment of our protocol in real environment (river, pipeline, bridge) using tinyOS nodes [22] or nodes with Lora [23] modulation and Mac and Network layers adapted for mesh networks. We could also propose that the pan-coordinator maintains a mapping of *DiscoProto* logical addresses to IPv6 addresses to use have our nodes as be a part of IoT [24].

References

[1] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communications magazine*, 40(8):102–114, 2002.

[2] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Comput. Netw.*, 52(12), August 2008.

[3] C. R. Baker, K. Armijo, S. Belka, M. Benhabib, V. Bhargava, N. Burkhart, A. Der Minassians, G. Dervisoglu, L. Gutnik, M. B Haick, et al. Wireless sensor networks for home health care. In *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, volume 2, page 832–837, 2007.

[4] Konrad Lorincz, Matt Welsh, Omar Marcillo, Jeff Johnson, Mario Ruiz, and Jonathan Lees. Deploying a wireless sensor network on an active volcano. In *IEEE Internet Computing*, pages 18–25, 2006.

[5] Marco Zimmerling, Waltenegus Dargie, and Johnathan M. Reason. Localized power-aware routing in linear wireless sensor networks. In *Proceedings of the 2nd ACM international conference on Context-awareness for self-managing systems*, CASEMANS '08. ACM, 2008.

[6] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steven Glaser, and Martin Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks.* ACM Press, 2007.

[7] I.E.E.E. Computer Society. *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wirelees Personal Area Networks (WPANs).* IEEE Computer Society, ieee std 802.15.4 2006 edition, 2006.

[8] ZigBee Alliance. *ZigBee Specification.* ZigBee Standard Organization, zigbee document 053474r17 edition, January 2008.

[9] Ameer Ahmed Abbasi and Mohamed Younis. A survey on clustering algorithms for wireless sensor networks. *Comput. Commun.*, 30, October 2007.

[10] Imad Jawhar, Nader Mohamed, and Dharma P. Agrawal. Linear wireless sensor networks: Classification and applications. *J. Netw. Comput. Appl.*, September 2011.

[11] Imad Jawhar, Nader Mohamed, and Khaled Shuaib. A framework for pipeline infrastructure monitoring using wireless sensor networks. In *Wireless Telecommunications Symposium, 2007. WTS 2007.* IEEE, 2007.

[12] Meng-Shiuan Pan, Hua-Wei Fang, Yung-Chih Liu, and Yu-Chee Tseng. Address assignment and routing schemes for zigbee-based long-thin wireless sensor networks. In *IEEE Vehicular Technology Conference '08, Spring 2008*, pages 173–177, 2008.

[13] Moussa Déthié Sarr, François Delobel, Michel Misson, and Ibrahima Niang. Automatic discovery of topologies and addressing for linear wireless sensors networks. In *Wireless Days (WD), 2012 IFIP*, pages 1–7. IEEE, 2012.

[14] André Cunha. On the use of ieee 802.15.4/zigbee as federating communication protocols for wireless sensor networks. Technical report, Polytechnic Institute of porto (ISEP-IPP), 2007.

[15] Sungjin Park, Eun Ju Lee, Jae Hong Ryu, Seong-Soon Joo, and Hyung Seok Kim. Distributed borrowing addressing scheme for zigbee/IEEE 802.15.4 wireless sensor networks. *ETRI journal*, 31, 2009.

[16] Meng-Shiuan Pan, Chia-Hung Tsai, and Yu-Chee Tseng. The orphan problem in zigbee wireless networks. *IEEE Transactions on Mobile Computing*, 8, 2009.

[17] Wei Ye, Jhon S. Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2002.

[18] Athanassios Boulis. Castalia : A simulator for wireless sensor networks. https://castalia.forge.nicta.com.au.

[19] Yuriy Tselishchev, Athanassios Boulis, and Libman Libman. Experiences and lessons from implementing a wireless sensor network mac protocol in the castalia simulator. In *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*. IEEE, April 2010.

[20] Marco Zuniga and Bhaskar Krishnamachari. Analyzing the transitional region in low power wireless links. In *Proceedings of the First IEEE International Conference on Sensor and Ad hoc Communications and Networks (SECON)*, October 2004.

[21] El Hadji Malick Ndoye, Frédérique Jacquet, Michel Misson, and Ibrahima Niang. Using a token approach for the MAC layer of linear sensor networks: Impact of the node position on the packet delivery. In *2014 IFIP Wireless Days, WD 2014, Rio de Janeiro, Brazil, November 12-14, 2014*, 2014.

[22] Muhammad Amjad, Muhammad Sharif, Muhammad Khalil Afzal, and Sung Won Kim. TinyOS-new trends, comparative views, and supported sensing applications: A review. *IEEE Sensors Journal*, 16(9):2865–2889, 2016.

[23] Aref Mouhamed and Sikora Axel. Free space range measurements with semtech lora technology. In *2014 2nd International Symposium on Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems*, pages 19–23, 2014.

[24] Han-Chuan Hsieh, Kai-Di Chang, Ling-Feng Wang, Jiann-Liang Chen, and Han-Chieh Chao. ScriptIoT: A script framework for and internet-of-things applications. *IEEE Internet of Things Journal*, 3(4):628–636, 2016.

[25] Amjad Muhamed, Akhtar Fayaz, Rehmani Mubashir Husain, Reisslein Martin, and Umer Tariq. Full-duplex communication in cognitive radio networks: A survey. *IEEE Communications Surveys Tutorials*, PP(99):1–1, 2017.