HAL
archives-ouvertes.fr

Automatic Discovery of Topologies and Addressing for Linear Wireless Sensors Networks

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

▶ **To cite this version:**

Moussa Déthié Sarr, François Delobel, Michel Misson, Ibrahima Niang. Automatic Discovery of Topologies and Addressing for Linear Wireless Sensors Networks. Wireless Days (WD), 2012 IFIP, Nov 2012, Dublin, Ireland. IEEE, pp.1-7, 2013, Wireless Days (WD), 2012 IFIP. <10.1109/WD.2012.6402801>. <hal-01661182>

HAL Id: hal-01661182

https://hal-clermont-univ.archives-ouvertes.fr/hal-01661182

Submitted on 11 Dec 2017

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.

Automatic Discovery of Topologies and Addressing for Linear Wireless Sensors Networks

Moussa Déthié Sarr[1,4], François Delobel[2,3], Michel Misson[2,3] and Ibrahima Niang[4]
[1]Clermont Université, Université Blaise Pascal, LIMOS,
BP 10448, F-63000 Clermont-Ferrand, France
[2]CNRS, UMR 6158, LIMOS,
F-63175, Aubière, France
[3]Clermont Université, Université d'Auvergne, LIMOS,
BP 10448, F-63000 Clermont-Ferrand, France
[4] Université Cheikh Anta Diop, LID,
BP 5005, Dakar, Sénégal
Emails: {sarr,delobel,misson}@sancy.clermont-universite.fr, iniang@ucad.sn

Abstract

Wireless sensor networks are a collection of sensor nodes that collaborate to sense a specific event in a given environment. When sensors monitor a structure organized linearly (e.g., pipelines, rivers, railways), they are organized in Linear WSN (LWSN, constituted by connected portions of lines), with different properties than a uniformly deployed WSN. The distributed address allocation in the ZigBee cluster-tree suffers from limitations in LWSN: the number of children is limited, as well as the maximum number of children routers, and the maximum tree depth. Stochastic address assignment, also available in ZigBee, has a high cost in exchanged messages and requires an expensive (in terms of messages and memory) routing process. This paper proposes an automatic discovery of topologies for linear wireless sensor networks combined with an efficient addressing mechanism. We show that our proposition avoids the waste of addresses while keeping the number of messages exchanged proportional to the number of nodes in the network.

Index terms— Linear Wireless Sensors Networks, LWSN, Addressing, Routing, Automatic Deployment, Topology Discovery, Clustering

1 Introduction

Wireless Sensor Networks (WSNs) allow data collection in many application fields (bridge structure [1], railways [2], volcanoes [3], rivers, animals [4], and health [5] monitoring). The physical topology of a WSN depends on the zone to be monitored. WSNs are organized using logical topologies (star, mesh, cluster tree) [6], [7]. The cluster-tree topology is the most used because it is easy to maintain and does not require a sophisticated routing protocol. Two kinds of nodes are usually used for those topologies, one having routing facilities, the other having a minimal implementation to support one or more sensor or actuator. The IEEE 802.15.4 standard named respectively each of them: Full Function Devices (FFDs) and Reduced Function Devices (RFDs). A FFD acts as a personal area network coordinator, coordinator or simple device. One PAN coordinator exists in WSN. A RFD is most often used as a data sensing node and is always associated with one and only one FFD.

A Linear WSN [8] is a special case of physical topology where nodes are deployed along lines. Linear WSNs can collect data in environments that cover linear zones such as rivers, bridges [1], water[9], gas or oil pipelines [10], production chains.

In WSNs, addressing and routing algorithms are usually designed for general topologies such as mesh or cluster tree topologies. In [7] and [9], solutions for addressing and routing are proposed for Linear WSN but present some drawbacks (see Section 2).

Therefore, we propose a solution for discovering topologies without prior knowledge, and use it to propose an efficient addressing mechanism for a hierarchical routing scheme without wasting too many addresses. In Section 2, we discuss the state of the art of address allocation. In Section 3, we present our topology discovery mechanism. In Section 4, we describe our addressing scheme. Section 5 presents our simulation results. Finally, Section 6 concludes the paper and presents some perspectives for future work.

2 Addressing Mechanism in Linear Wireless Sensor Networks

The, ZigBee standard [7] for WSNs provides the standard address assignment mechanisms. In the following, we describe the mechanisms, and an address assignment in long thin WSNs.

2.1 Stochastic Address Assignment Mechanism

ZigBee Stochastic Address Assignment Mechanism (SAAM) attributes addresses randomly. In large networks, this method has a high probability of generating address conflicts, so SAAM uses a reparation algorithm to solve these conflicts. The AODV [11] routing protocol is used to route packets in the network.

However, the address conflict resolution adds delay in the network establishment phase. The use of the AODV flooding algorithm has a high control overhead. The cost of this algorithm is high as there is a trade-off between memory for caching the routing tables and number of messages sent. Motes usually have low memory [12], so keeping a large route cache is usually not a good solution. Power capacity is also limited, and network speed is not comparable to the speed of a wired network commonly used with AODV, this control overhead (either during the addressing phase, or during the network usage) is a major issue. So SAAM and AODV routing are not efficient ideas for linear WSNs.

2.2 Distributed Address Assignment Mechanism

ZigBee Distributed Address Assignment Mechanism (DAAM) for the cluster-tree proposes a hierarchical addressing and routing mechanism based on the maximum number of children (Cm) per node, the maximum number of children routers (Rm) per node, and the maximum tree depth (Lm). This mechanism provides easy and fast allocation and routing.

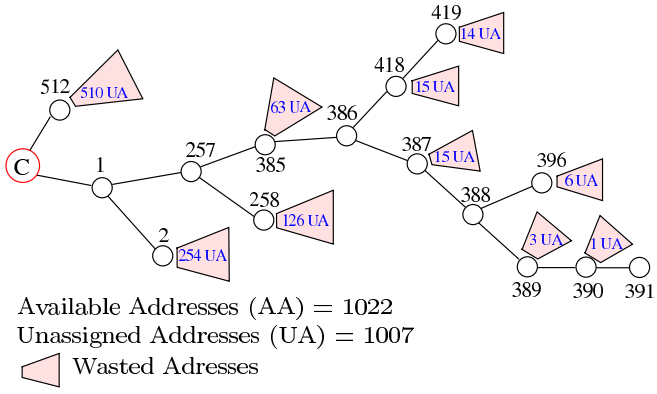

Figure 1: Waste of addresses on a ZigBee cluster-tree (RM=2, Cm=2, Lm=9).

In linear WSNs (LWSNs), DAAM is not appropriate. It can result in a huge waste of addresses on linear parts (see Figure 1) and orphan nodes [13]. In fact, with the characteristics of LWSNs, the maximum number of addresses that can be allocated within the limits of the address space wich is $2^{16} = 65536$ can be computed according to Equation 1 and is shown in Table 1. Finally in DAAM, the available addresses are not well balanced in the network (see Figure 1).

$$Amax = Cskip(0) \times Rm + Cm - Rm, \qquad (1)$$

where

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

Cluster-tree parameters			
Cm	Rm	Lm	available addresses
2	2	15	65534
3	3	9	29523
4	4	7	21844

Table 1: Available addresses with different cluster-tree parameters, in DAAM.

2.3 Addressing Mechanism in Long Thin Wireless Sensor Networks Based on ZigBee

In [9], the authors propose a long linear network topology called Long Thin Wireless Sensor Network (LTWSN) based on ZigBee. In this topology, all nodes are FFDs and are divided into clusters (as shown in Figure 2). In each cluster, two new roles are defined: a cluster head and a bridge. Those nodes have to be manually selected amongst all nodes.

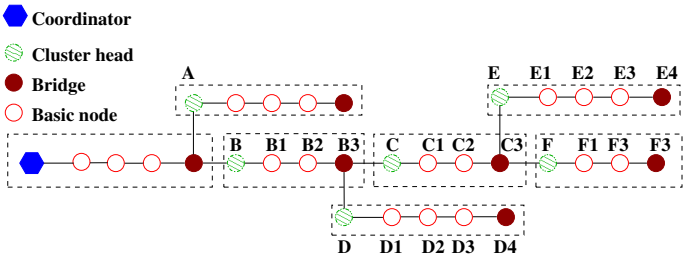

Figure 2: A Long Thin WSN as proposed in [9]

The authors propose to divide the 16-bit network address of the nodes in two parts: the first part is a cluster ID and the second part is a node ID. Then, the network administrator calculates the address blocks to allocate to the clusters from the logical network (GL) (shown in Figure 3), where each node in the logical network represents a cluster in Figure 2. With this logical network, the network administrator fixes two parameters: the maximum tree depth in GL (CLm) , and the maximum number of children per node in the logical network CCm

Then, the network administrator manually assigns the same cluster ID to each node of a cluster while the node ID are allocated sequentially. If $CCm = 1$, GL has no branch and the cluster IDs are attributed in a sequential manner. If $CCm \geq 2$, the cluster IDs are assigned in a recursive manner as in ZigBee. From CCm, and CLm, the administrator computes a parameter called $CCskip$ to derive the starting cluster ID of cluster children.

$$CCskip(d) = \frac{1 - CCm^{CLm-d}}{1 - CCm} \qquad (2)$$

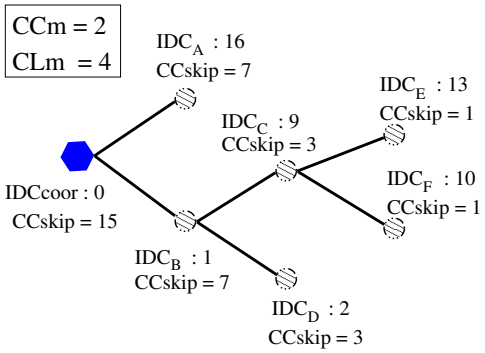

Figure 3: A Logical network of Figure 2

The cluster ID of the PAN coordinator cluster is 0. For each node at depth d in GL, if the current cluster ID is ID_C, then the $i-th$ child cluster is assigned the following cluster ID :

$$ID_C i = ID_C + (i-1) * CCskip(d) + 1. \quad (3)$$

After cluster ID attribution and node deployment, each node of the network periodically broadcasts HELLO packets including its address with its cluster ID to start the Node ID attribution phase.

However, in this topology, if a bridge or a cluster head fails, the routing of data becomes impossible because all inter cluster transmissions are routed through the bridge-cluster head links even if some other communication links exist between nodes of different clusters. Manual intervention is also necessary to elect a new cluster head or bridge because these nodes are defined manually by the network administrator at the startup of the network. Also, address assignment of clusters can cause a waste of addresses as in DAAM because the computation of the cluster ID addresses is based on the initial network parameters (CCm, CLm).

3 Discovering the Logical Topology

3.1 Centralized algorithm

Two main sources of information can be considered when trying to discovery a coherent logical topology without knowing the physical topology: either an estimation of the quality of a link (Received Signal Strength Indication (RSSI) or Link Quality Indication (LQI)), or the possibility to establish a stable link between two motes. RSSI (and LQI) can be considered as attractive, being more precise than the sole existence of a communication link, but results in [14][15] have shown that RSSI is not a good indicator for distance. So, we choose to focus on the valid links between motes.

We consider a graph where nodes are the motes, and where an edge is added between two nodes if and only if the two corresponding motes are neighbors (the radio links are supposed to be symmetrical). Solving the address problem is equivalent to finding a spanning tree with a minimum number of branching nodes. Minimizing the number of branches allows to identify the disjunctions in linear parts of the physical topology. This problem is different from a traditional minimum spanning tree problem because the cost function is global and dynamic, and is not a static weight associated to each edge.

The centralized version of our algorithm is a greedy algorithm, similar to the Prim's algorithm (Jarnik's Algorithm)[16][17] that extends a connected subset of a graph at each iteration, but with a custom objective function instead of the usual static weight of the edges. The objective function to maximize is dynamic (it varies during the algorithm). We use it while examining the edges from the connected part C of the graph (nodes which have already been added) to the remaining nodes belonging to R. Let $a \in C$ and $b \in R$ be two vertices and (a, b) be an edge, we are interested in the following criteria (classified by importance):

- the number of neighbors that a and b have in common common(a, b): this is the main factor, and it is strongly linked to the proximity of nodes a and b;
- the number of sons of a, sons(a): we try to avoid creating too many branches and want to keep this value equal to 1;
- The total number of neighbors of the two nodes, neighbors(a): we try to first connect nodes with low connectivity, because the algorithm has to connect them and would have less choices for connecting them later.

So, we try to maximize the objective function objective$(a, b) \stackrel{def}{=} \alpha \times \text{common}(a, b) - \text{sons}(a) - \beta \times (\text{neighbors}(a) + \text{neighbors}(b))$. α should be larger than the maximum number of sons of a node, and β should be smaller than $1/(\text{neighbors}(a) + \text{neighbors}(b))$. For our tests, we use $\alpha = 10$ and $\beta = \frac{1}{1000}$.

Such an algorithm would be inefficient when deploying on real case problem as it is centralized. Therefore, we propose a distributed version of the algorithm.

3.2 Distributed version

The centralized part of our algorithm is the part where the vertex with the smallest cost is chosen amongst all vertices. Nevertheless, as all nodes in C are already associated to the network, having messages exchanged through C could be implemented in a distributed manner, but would require sending many messages (flood the connected network). So, we designed a distributed version of the algorithm where the chosen edge is only a *local* maximum, therefore requiring much less communication: instead of choosing the best edge amongst all the edges, an edge is added to C if and only if no better edge can be found at a distance of $challenge - radius$ hops. Experiments showed that $challenge - radius = 3$ is a good compromise between cost and efficiency.

3.3 Protocol implementing the distributed algorithm

Every mote in the network uses the same protocol named DiscoProto. The coordinator node (the root of the tree) has to be manually selected and is in charge of starting a topology discovery process.

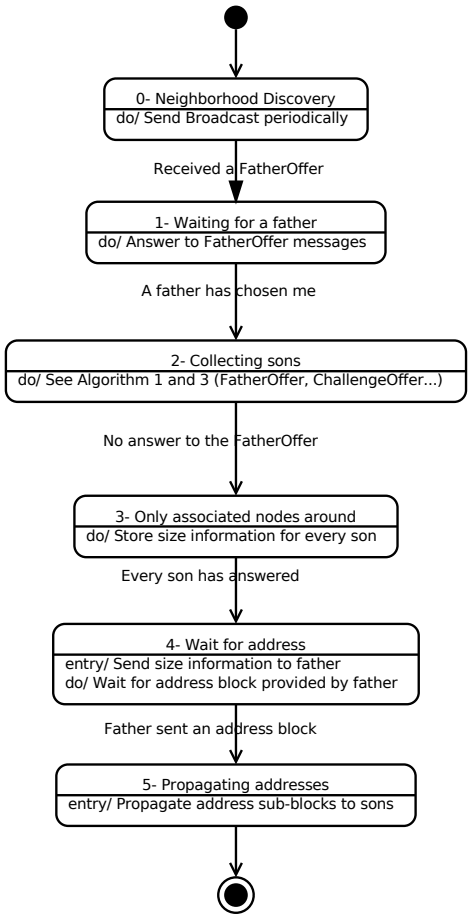

Figure 4: State diagram of Discoproto.

Figure 4 shows the six possible states of a mote. State 0 is the startup state in which a node periodically broadcasts HELLO messages so that every node can discover its neighbors (the period of broadcasts and the duration of a cycle has been empirically chosen and their precise definition is left as future work, using a more realistic simulator). This is required because the computation of the objective function depends on the knowledge of the neighborhood. As soon as a $FatherOffer$ message is received, the mote enters state 1.

State 1 lasts until the node gets associated. The only thing a mote does in State 1 is to answer $FatherOffer$ sent by an already-connected node by computing and sending the *objective* value of the vertex at that time (see Algorithm 2).

When a node was chosen as a son by a father mote, it enters State 2 (it is associated) and begins looking for possible sons. For that, it performs two operations: it runs Algorithm 1, and it diffuses and/or answers *ChallengeOffer* messages from other associated nodes (see Algorithm 4). When no node answers the fatherhood offer in a given time, the mote enters state 3.

Algorithm 1: Collecting Sons (action for State 2).

```
Result: Finds new sons, ends when no candidate is found.
Input: A set of neighbors
Input: MACAddr the MAC address of the node
Input: sonsTimeout the delay left for sons to answer a
       fatherOffer. (1s)
Input: challengeTimeout the delay left for radius-neighbors to
       answer a challengeOffer. (Defaults to 2s)
challenge-radius ← 3                          // maximum hop count
while state = 2 do
    offer ← ∅      // Memorize the best offer (mote, quality) received.
                   // offer is asynchronously modified by Algorithm 3
    fatherOffer ← (MACAddr, neighbors,sons)
    Broadcast(fatherOffer)     // Sons will answer as defined in Algorithm 2
    sleep(sonsTimeout)   // Leave some time for asynchronous answer from
                         // unassociated neighbors giving an offer (See Algorithms 2 and 3 ).
    if offer ≠ ∅ then
        // At least one son answered to the association request
        better ← false                  // better will be modified asynchronously
                                                          // in Algorithm 4
        challengeOffer ← (MACAddr, challenge-radius, offer)
        Broadcast (challengeOffer)          // Ask all associated
                                  // nodes around if they have a better offer
        sleep(challengeTimeout)          // Leave some time for radius-neighbors
                                         // to answer. See Algorithm 4
        if better = false then    // If no radius-neighbors can do better
            sons ← sons ∪ offer.node
    else
        // No neighbor has answered
        state ← 3
```

Algorithm 2: Answer to a FatherOffer (action for State 1).

```
Result: Reacts to the reception of a fatherOffer. Computes
        the value of the objective and answers to the potential
        father.
Input: A fatherOffer from an associated node, including the
       neighboring of the father.

if fatherAddr = ∅ then          // The node is not associated
    objective ← Objective (fatherOffer.sons,fatherOffer.neighbors ,
    sons, neighbors)
    sonOffer ← ( MACAddr, objective)
    Unicast (fatherOffer.MACAddr, sonOffer)
```

Algorithm 3: Answer to a SonOffer (action for State 2).

```
Result: Reacts to the reception of a sonOffer by memorizing
        the attribute offer if it is better than the current one.
Input: A sonOffer from a candidate node (in reply to a
       FatherOffer), including the objective value.

if offer = ∅ ∨ offer.objective < sonOffer.objective then
// Better solution
    offer ← sonOffer
```

State 3 is the beginning of the addressing process, which is detailed in the next section. In State 3, a node waits for all its sons to send information about the size of its subtree. When this is done, the mote enters state 4. When entering State 4, a node sends the size of its subtree to its father, and then waits for a range of addresses from its father, entering State 5. When a

Algorithm 4: Answer to a ChallengeOffer (action for State 2).

Result: Reacts to the reception of a ChallengeOffer. Sets attribute offer to newly received offer if better. Eventually propagates the ChallengeOffer to its neighbors. Changes the attribute better (used in Algorithm 1) to *true* if a better offer was received. Sends a better offer to the ChallengeOffer emitter if local offer is better.
Input: offer: The current best offer
Input: newOffer: the newly received offer

```
if newOffer.objective > offer.objective then
    offer ← newOffer
    better ← true
if newOffer.objective < offer.objective then
    nextStep ← newOffer.path.last
    destPath ← newOffer.path −{nextStep}
    Unicast (nextStep, offer, destPath)
if newOffer.challengeRadius ≥ 1 ∧ MACAddr ∉ newOffer.path
then
    newOffer.challengeRadius ← newOffer.radius−1
    newOffer.path ← newOffer.path ∪ MACAddr
    Broadcast (newOffer)
```

node receives a range of addresses, it dispatches them between its son, keeping a few for itself. The node is then fully associated, with an address.

4 Addressing

We propose a hierarchical addressing scheme based on the topology discovered by our algorithm (see previous section). When in State 3 (see Figure 4), a node is not accepting sons anymore. It waits until it knows the size sent by each of its sons, and memorizes it. When all the information is available, the size of each subtree is summed up, and this information is propagated to the father node. At a cost of one message per node, we can propagate the size value in the tree, and memorize it locally.

When the coordinator node has been informed of the size of the whole tree, address propagation begins. Addresses are allocated from the root of the tree and are propagated from father to sons. When a node m receives a block of address $[a_1, a_2]$ and an availability coefficient R, it chooses a_1 as its own address. Then, it keeps addresses from $[a_1 + 1, a_1 + R]$ as available addresses in case another node has to be adopted later. The remaining block $[a_1 + R + 1, a_2]$ has to be shared between the n_{sons} subtrees of m (denoted by son^i_m). It is done by splitting $[a_1 + R + 1, a_2]$ between the sons of m, proportionally to $size(son^i_m)$. Each son receives its own block of address, and the availability coefficient, and the whole process is repeated until the end of the tree. This address allocation is deterministic and generates one message per node in the network.

The total number of exchanged messages is easy to calculate: each node sends a message to its father when all its subtrees are associated. Each node receives a message from its father indicating a range of addresses to share. The total number of (unicast) messages required for the addressing is therefore equal to twice the number of nodes in the network.

5 Evaluation

We wrote a DiscoProto simulator in Ruby [18]. Each mote is implemented using its own thread, so they all run in parallel. Motes communicate through message exchanges, both unicast and multicast. All the messages are sent asynchronously and logged. We have not implemented any MAC layer and two nodes can communicate if the distance between them is below the maximum range. The code for the simulator is available at http://sancy.clermont-universite.fr/~delobel/DiscoProto.

We also designed a random physical topology generator for linear networks. This generator creates topologies depending on the maximum range allowing two motes to communicate, the average distance between two motes, the average frequency of branches creation, and the number of motes.

The complete set of data consists in 21380 simulated topologies, with either 50, 100, 200 or 500 motes. *Frequency* is set to 0.05. On the whole test, the average ratio of number of disjunctions (nodes with more than one son in the discovered logical topology) by number of generated branches (generated during the physical topology generation) is 1.03, with a standard deviation of 0.31. In many cases, short branches or branches close enough to another branch could be merged into a single discovered branch. In every case, all the nodes were connected as we generate only topologies where the distance between two nodes is below the reception distance. The topologies which were automatically generated are too large to be reproduced in this paper.

5.1 Closeness to the physical topology

Figure 5 presents the number of count disjunctions computed as a function of the number of branches which occurred during the generation. Both values are linearly correlated. On some examples, the computed topology has less disjunctions than the generated topology has branches. It is caused by short branches in the generated topology, or branches very close to another portion of the network which can be merged with another without any additional disjunction.

5.2 Control overhead to discover the logical topology

Figures 6 and 7 present the number of broadcast and unicast messages sent during the topology discovery phase. The average number of messages sent is proportional to the number of nodes in the network, but the standard deviation (95% confidence interval are shown on the Figure 6, and 7) is high as challenge offers are often denied when the physical topology is dense. It should be noted that the distribution of number of frames (unicasts and broadcasts) is not Gaussian, as show on Figures 9, 11, 8, and 10 (similar distribution occur for a topologies with 100 or 200 nodes).

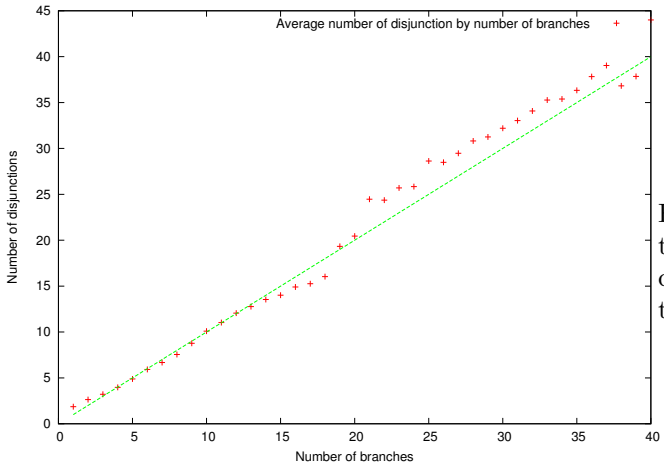

Figure 5: Number of disjunctions found vs number of generated branches.

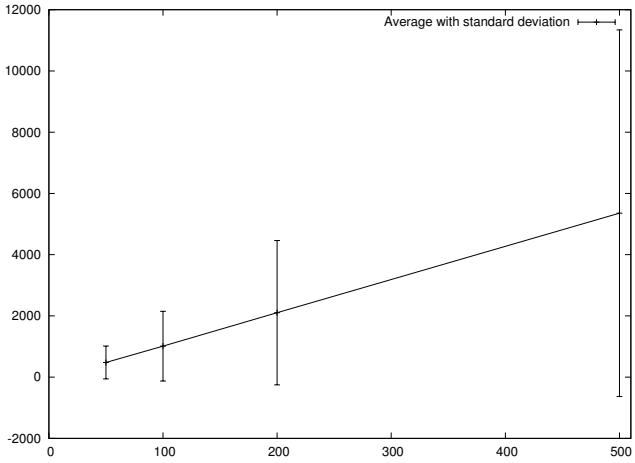

Figure 6: Number of broadcast messages sent to build the topology.

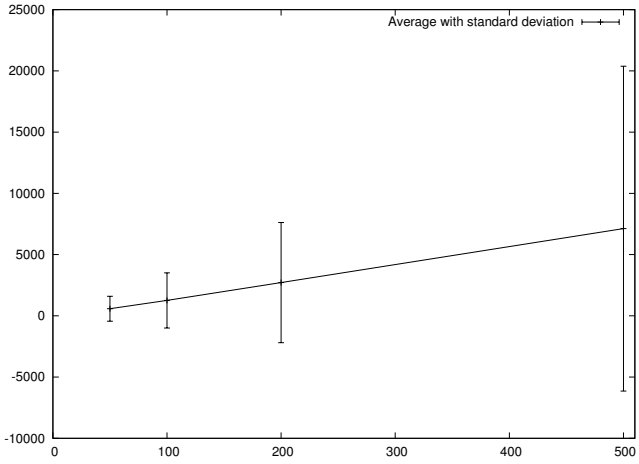

Figure 7: Number of unicast messages sent to build the topology.

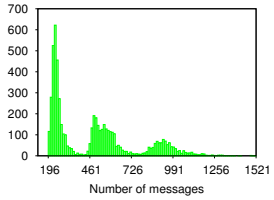

Figure 8: Distribution of total number of broadcasts for various 50 nodes topologies.

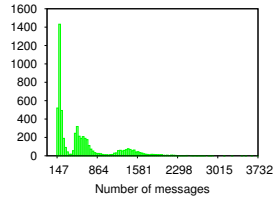

Figure 9: Distribution of total number of unicasts for various 50 nodes topologies.

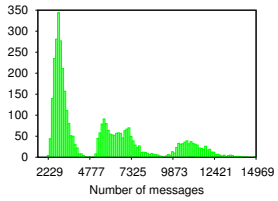

Figure 10: Distribution of total number of broadcasts for various 500 nodes topologies.

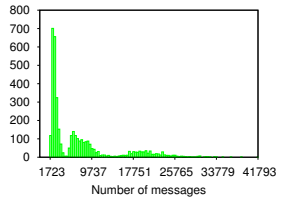

Figure 11: Distribution of total number of unicasts for various 500 nodes topologies.

5.3 Distribution of addresses in the network

Our main issue with hierarchical addressing is that in the particular case of linear networks, the available addresses are not well balanced in the network. Figure 12 presents the topology discovered by running the TopoProto simulator on the topology presented in Figure 3, with the addresses given by our addressing mechanism. We choose in this example $R = 2$, and observe that the available addresses are uniformly shared between all the nodes in the network. This property will ease the possibility to welcome dynamic nodes incoming (this part is left for further work). The number of disjunctions found is slightly inferior to the number of linear parts.

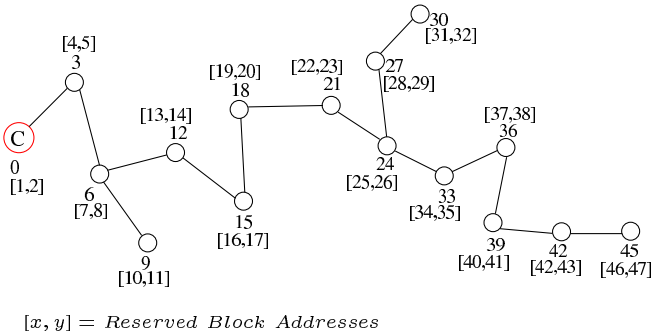

Figure 12: Simple topology discovered by TopoProto

More generally, in a topology with n nodes, we can use exactly n addresses (with no available address). If the parameter R is chosen, $n * R$ addresses will be allocated, and $R-1$ available addresses will be available for each node.

5.4 Generalization to a non linear network

Out TopoProto a protocol could run on non linear networks (e.g., a random network topology) but this would lead to poor results in term of both time and logical topology. The time for building the whole network would be longer as the number of challenges would be high because of the density. As the protocol tries to minimize the number of nodes with multiple sons, the logical topology built for random physical topologies would have uselessy long branches which would increase the duration of communications.

6 Conclusion

In this paper, we proposed a discovery mechanism for the topology of a linear WSN. In our protocol, only the PAN is chosen manually. We propose an addressing scheme which overcomes the ZigBee DAAM limitations while keeping the simplicity, low memory footprint and effectiveness of a hierarchical addressing and routing scheme.

The next step is to implement our protocol on the NS-2 simulator, using a proper IEEE 802.15.4 Medium Access Control layer, to measure the effect of link losses and collisions, and to compare our protocol with ZigBee DAAM protocol. We also plan to deal with dynamic topologies to recover from a node or from a link destruction and deal with new nodes incoming, without rebuilding the whole network or routing table (roughly, the idea is to implement permutations of nodes with consecutive blocks of addresses, and have nodes be back in a son discovery state). Another aspect we plan to study is to find solutions for balancing available addresses in the network when a shortage has been produced by a massive increase of nodes in a small part of the network.

We would like to thank Alexandre Guitton for careful reading and useful suggestions.

References

[1] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*. ACM Press, 2007, pp. 254–263.

[2] M. Zimmerling, W. Dargie, and J. 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*, ser. CASEMANS '08. New York, NY, USA: ACM, 2008, p. 24–33. [Online]. Available: http://doi.acm.org/10.1145/1367943.1367946

[3] K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees, "Deploying a wireless sensor network on an active volcano," in *IEEE Internet Computing*, 2006, pp. 18–25.

[4] E. S. Nadimi, H. T. Søgaard, T. Bak, and F. W. Oudshoorn, "ZigBee-based wireless sensor networks for monitoring animal presence and pasture time in a strip of new grass," *Computers and electronics in agriculture*, vol. 61, no. 2, p. 79–87, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169907002013

[5] 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*, vol. 2, 2007, p. 832–837. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4224209

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

[7] Z. Alliance, *ZigBee Specification*, zigbee document 053474r17 ed., ZigBee Standard Organization, January 2008.

[8] I. Jawhar, N. Mohamed, and D. P. Agrawal, "Linear wireless sensor networks: Classification and applications," *J. Netw. Comput. Appl.*, pp. 1671–1682, September 2011.

[9] Y.-C. L. Meng-Shiuan Pan, Hua-Wei Fang and Y.-C. Tseng, "Address assignment and routing schemes for zigbee-based long-thin wireless sensor networks," in *IEEE Vehicular Technology Conference '08, Spring 2008*, 2008, pp. 173–177.

[10] I. Stoianov, L. Nachman, S. Madden, and T. Tokmouline, "Pipenet: A Wireless Sensor Network for Pipeline Monitoring," in *ACM IPSN*, 2007. [Online]. Available: http://db.csail.mit.edu/pubs/ipsn278-nachman.pdf

[11] C. Perkins, E. Belding-Royer, and S. Das, *Ad hoc On-Demand Distance Vector (AODV) Routing*, ser. Request for Comments. IETF, Jul. 2003, no. 3561, published: RFC 3561 (Experimental). [Online]. Available: http://www.ietf.org/rfc/rfc3561.txt

[12] M. Inc, "Wireless modules specifications for LOTUS, IRIS, MICAZ, MICA2, TELEOSB." [Online]. Available: http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html

[13] M.-S. Pan, C.-H. Tsai, and Y.-C. Tseng, "The orphan problem in zigbee wireless networks," *IEEE Transactions on Mobile Computing*, vol. 8, pp. 1573–1584, 2009.

[14] F. V. Karel Heurtefeux, "Is RSSI a good choice for localization in wireless sensor network?" in *IEEE 26th International Conference on Advanced Information Networking and Application (Aina)*, 2012, pp. 732–739.

[15] J. Luo, X. Xu, and Q. Zhang, "Understanding link feature of wireless sensor networks in outdoor space: A measurement study," in *GLOBECOM*. IEEE, 2011, pp. 1–5.

[16] V. Jarník, " O Jistém Problému Minimálním (About a Certain Minimal Problem) (in Czech, German summary)," *Práce Mor. Prírodoved. Spol. v Brne VI*, vol. 4, pp. 57–63, 1930.

[17] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technology Journal*, vol. 36, pp. 1389–1401, 1957.

[18] Y. Matsumoto and K. Ishituka, *Ruby programming language*. Addison Wesley Publishing Company, 2002.