Handling Nodes Mobility and Failure During Bootstrapping in Randomly Deployed Ring-based Wireless Sensor Networks

http://dx.doi.org/10.3991/ijes.v1i1.29345

Ghofrane Fersi, Wassef Louati and Maher Ben Jemaa
National School of Engineers of Sfax, Sfax, Tunisia

***Abstract*—Distributed Hash Table (DHT)-based protocols are new approaches proposed to Wireless Sensor Networks (WSN). Their main advantage resides on the easy integration of DHT-based WSN into the Internet Of Things. However, these protocols face multiple challenges in their bootstrapping phase, especially at the case of randomly deployed WSN. We presented in a recent work a bootstrapping protocol for use in DHT-based protocols having the structure of ring in static WSN. However, in most cases, there are some nodes that may fail or move in the network. Bootstrapping should take into account such situations in order to achieve better performance.**

In this paper, we propose a distributed local recovery method for use in our bootstrapping protocol that allows it to handle efficiently nodes failure and mobility. Simulation results have shown that our proposed approach is able to ensure the local recovery in a timely and energy-efficient manner.

***Index Terms*—Bootstrapping, Distributed Hash Table-based protocol, Failure, Mobility, Wireless Sensor Networks.**

I. INTRODUCTION

The Internet Of Things (IOT) [17] is the future vision of Internet: Internet is no more related to the virtual world but it is extended to our real life. In the future Internet all the objects and area are connected to the internet. Each object is identified with a unique identifier and we can get information about it by sending a query with its identifier. This query will be received by this object even if it has changed its physical position. The communication nature in the Internet of Things is information centric which means that data are identified by keys. When given data need to be retrieved, a query with their corresponding key is sent.

A great part of IOT is made up of Wireless Sensor Networks [1] [2]. Hence, there is a need to conceive protocols that facilitate the integration of these networks into IOT. Distributed Hash Table (DHT)-based protocols [4] and more specifically the protocols having the virtual structure of a ring [5] [6] [10] [11] [12] are suitable solutions to facilitate such integration. Effectively, each node in these protocols is identified with a unique location-independent identifier. All the nodes are organized into a virtual ring with the increasing order of their identifiers. Such structure makes the integration of WSN into IOT, an easy and efficient task.

Also, there are some situation where the WSN cannot be deployed manually such the case of natural disasters and/or hostile supervision. In these cases, sensors are scattered randomly and they should form autonomously their own network without human intervention. It is clear that in such cases, we need the use of autonomous and location independent protocols. DHT-based protocols are autonomous and ensure their functionalities without the need to have any information from any central point in the network. Hence these protocols are well suited to randomly deployed sensors cases.

However, the bootstrapping phase of these DHT-based protocols is so challenging at the case of randomly deployed WSN since all the nodes are scattered simultaneously and start joining the network at the same time. They will then exchange lot of simultaneous messages which leads to interference problems. All these problems lead to multiple inconsistencies. Existing works [8] [9] in this field argue that bootstrapping should contain two phases: In the first phase, all nodes search their places simultaneously and form consequently multiple rings. In the second phase, messages are exchanged between nodes in order to heal all the rings into a global consistent one.

In a recent work [15], we have proposed a bootstrapping approach that avoids concurrent joining and that forms directly one global consistent ring in randomly deployed static WSN. This is ensured by organizing all the nodes into a tree and orchestrating the nodes joining from root to children nodes in a way that at a given time only one node joins the network.

Nodes in WSN are prone to failure. They may fail when scattered or fail due to their energy exhaustion. Also, in most WSN, there are some nodes that can move and change their place during the network bootstrapping. Effectively, even if the total bootstrapping duration is limited, they can be some few nodes that may change their location. For example, at the case of flooding, sensors when scattered in order to collect information about these flooding, are obviously static or move so slowly in the same speed which preserves the same physical neighbors for almost nodes. However, the presence of some physical obstacles, may change the speed of some few nodes leading to a modification in the physical neighbors set. There is hence a need to take into account such situations in order to avoid any inconsistency that could be caused by a node failure or mobility.

In this paper, we propose a distributed local recovery mechanism that can be integrated into our proposed bootstrapping protocol in order to handle efficiently nodes failure and/or mobility in the network.

Our paper is organized as follows. In section 2, we present a general overview of existing ring-based protocols. The related work is given in section 3. We

specify in section 4 a brief presentation of our bootstrapping protocol. Our proposed improvement to handle nodes failure and mobility is given in section 5. Section 6 specifies the cost of our proposed recovery approach in terms of sent messages number. Section 7 concludes the paper.

II. BACKGROUND

In order to be familiar with the ring-based protocols, we present in this section a short overview of two DHT ring-based protocols:

Virtual Ring Routing (VRR) [5] and Scalable Source Routing (SSR) [6].

A. Virtual Ring Routing

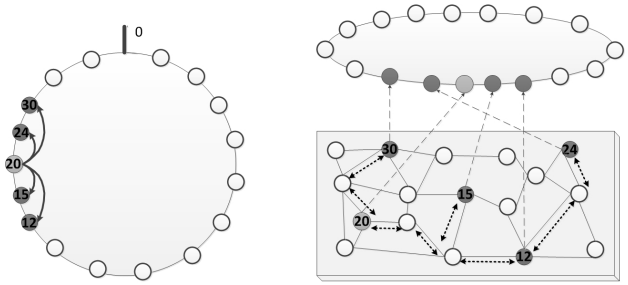

Figure 1. Virtual Ring Routing virtual and physical topology

Virtual Ring Routing (VRR) [5] combines DHT and routing functionalities without the need of any underlying routing protocol. VRR is directly integrated on the top of the link layer. VRR nodes identifiers are location independent.

As depicted in figure 1, all the nodes are organized into a virtual ring in an increasing order of their identifiers. Each VRR node maintains in its routing table the next physical hops towards the virtual paths that the current node participated in their setup. It contains also a physical neighbor set containing the identifiers of the physical neighbors in addition to the next physical hops towards the virtual neighbors of the current node. To route a message, the VRR node picks from its routing table the node having the closest identifier to the destination's identifier and forwards the message to its corresponding physical hop.

B. Scalable Source Routing

All the Scalable Source Routing (SSR) [6] nodes should be organized into a global virtual ring. Each node should have a source route to its virtual neighbors. SSR packets contain source address, destination address and source route. The former is not essentially the total route from the source to the destination. It can only be a sub route that ends at an intermediate node. This intermediate node checks its local cache in order to search an appending route to the destination. If this search fails, it appends the source route to the closest node to the destination.

III. RELATED WORK

There are some DHT-based protocols that support bootstrapping in randomly deployed WSN. Iterative Successor Pointer Rewiring Protocol (ISPR) [8] is a simple bootstrap protocol for use in ring-based protocols like SSR [6] or VRR [5].

Each node maintains exactly one and only one successor and predecessor by exchanging *Successor Solicitation* messages. At the end of this step, multiple consistent local rings are formed. To ensure global consistency, a specified node in each ring floods the network. VRR uses also a similar procedure to ISPR. Flooding the entire network consumes a lot of energy.

The linear method [9] shares with ISPR the same steps to reach local ring consistency. However, in order to ensure the global ring consistency, it does not use the flooding step. The

linear method assumes that the address space is linear and not a ring. The edges in the virtual graph are undirected. Total ordering of nodes addresses is used in order to distinguish right and left neighbors. To form a global ring, the leftmost node establishes an edge to the rightmost node.

The two mentioned approaches require two steps in order to arrive to the steady state: multiple consistent ring formation, then, rings fusion.

Our protocol does not need any ring merging since it ensures directly the formation of a global consistent ring.

IV. BOOTSTRAPPING

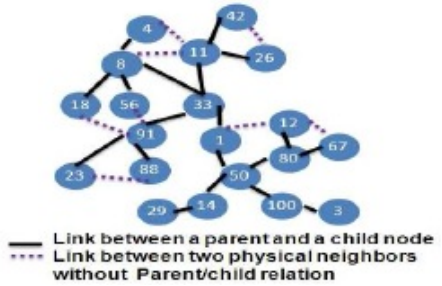

Figure 2. Example of randomly deployed network topology

In this section, we give a brief overview of our proposed bootstrapping approach [15]. All the nodes in our approach are organized into a tree. The root node is the first node that joins the network. When active, its child node having the smallest identifier joins the network and so on. When the current joining node has no more children, it sends a message to its parent that in turn chooses another child node to join the network. Recursively, all the nodes join the network.

Bootstrapping in our approach is divided into rounds. In each round i, there is one active node that broadcasts HELLO messages in order to discover its children and starts a hearing phase. In this step, neighboring nodes that received this HELLO message and that did not have received any HELLO messages from any other node

Algorithm 1 Recovery approach pseudo code: *src* is the identifier of the source node, *MyParent* is the identifier of the parent of the current node. *parentSrc* is the parent of the source address, *LevelSet* contains the identifiers of the nodes having the same parent as the current identifier. *PhysSet* is the set of the physical neighbors. *reschedule_TimerNeighbor* and *reschedule_TimerParentSearch* reschedule respectively the timers maintaining the physical neighbors responses and the eventual parent responses. *waitingSet* is the set containing the children identifiers that are waiting to join the network, *Moved* checks if the node has moved. *AdditionalWaitingSet* is set of parents requesting an additional permission from the first active node to join other nodes to the network. *me* is the identifier of the current node. *forward PermissionException* forwards the message using the employed routing protocol.

```
procedure RECEIVE(LIFE, src)
    if myParent=ParentSrc then
        add LevelSet(src)
    if src ∈ PhysSet then
        receivedLife=true
        reschedule TimerNeighbor (src)
    else
        add PhysSet(src)
        reschedule TimerNeighbor (src)
end procedure
procedure RECEIVE(ParentRequest, src)
    if (src ∈ LevelSet) And (sentReallocation = False) And (Moved = false) then
        send ParentOffer(me) to src
end procedure
procedure RECEIVE(ParentOffer, src)
    if sentAssociation = False then
        send Association(me) to src
        sentAssociation = True
end procedure
procedure RECEIVE(Association, src)
    add waitingSet(src)
    if sentReallocation then
        send PermissionException(src) to FirstActiveNode
end procedure
procedure EXPIRED TIMERNEIGHBOR(identifier)
    if receivedLife=true then
        receivedLIFE=false
        reschedule TimerNeighbor (src)
    else
        if nbNoResponse=2 then
            remove PhysSet(identifier)
            if identifier ∈ waiting_set then
                remove waitingSet(identifier)
            if identifier=myParent then
                send ParentRequest(me)
                reschedule TimerParentSearch
        else
            nbNoResponse=nbNoResponse+1
            reschedule TimerNeighbor (src)
end procedure
procedure EXPIRED TIMERPARENTSEARCH(identifier)
    if associated=False then
        Send ParentRequestException(me)
end procedure
procedure RECEIVE(ParentRequestException, src)
    if (src ∈ LevelSet) And (sentReallocation = True) And (Moved=false) then
        send ParentExceptionOffer(me) to src
end procedure
procedure RECEIVE(ParentOfferException, src)
    if sentAssociation = False then
        send Association(me) to src
        sentAssociation = True
end procedure
procedure RECEIVE(PermissionException, src)
    if FirstActiveNode=me then
        add_AdditionalWaitingSet(src)
        if nbWaitingSet=0 then
            send PermissionException (me) to src
    else
        forward PermissionException to FirstActiveNode
end procedure
```

during the i-1 rounds are considered as children nodes of this active node. This latter, in turn, is considered as their parent forming hence a tree. The children nodes send to their parent node a HELLO response message at a random time in order to avoid interference problems that can be caused by simultaneous message sending.

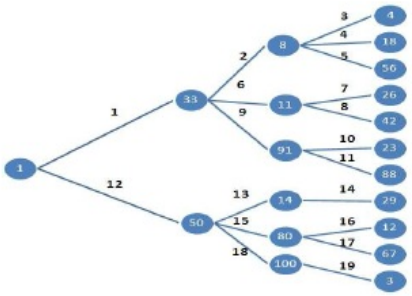

Figure 3. Bootstrapping tree

The active node of a given round stores all the identifiers of its waiting set (the set containing the identifiers of the children nodes) by the increasing order of their identifiers. Then, it sends to the node having the smallest identifier in its waiting set a Permission message. This message allows the destination node to start the joining process. For example at the case of VRR, when this message is received by the child node, it sends a setup request message to its corresponding active node. This latter picks from its routing table, the node having the closest identifier to this new node and forwards the message to its corresponding next physical hop and so on until reaching the node having the closest identifier to the current joining node identifier. This node adds the current joining node to its virtual neighbors set called vset. Then, it responds it by a setup message containing the set of its other virtual neighbors in order to help the joining node finding the other neighbors. Once this setup message is arrived to the destination, the source node is added to the vset of this new joining node then it sends setup request messages to the nodes in the vset field of the setup message. When the candidate nodes respond the new joining node, it becomes well stabilized into the virtual ring and becomes active. This recently active node will start the next round and so on. The first branch in the bootstrapping phase starts by an active node chosen at the

network startup. The first branch ends when the recently active node does not receive any HELLO response messages from any nodes during its entire hearing step (The first branch in figure 3 is made up of nodes 1, 33, 8 and 4). In such a case, this latter node sends to its parent node a Reallocation message. This message informs the parent node that its first child node has no more nodes to treat. When the parent node receives this message, it removes the source node from its waiting set and chooses from its waiting set, the next smallest identifier and sends to its corresponding node a Permission message. All the steps stated before are repeated until the active node receives a Reallocation message from all its waiting set. At this case, the active node sends to its parent node a Reallocation message in order to choose another child node to start the joining process. All the steps are repeated until the first active node (the root) received Reallocation messages from all the nodes in its waiting set.

When this happens, all nodes in the network are well stabilized into one global virtual ring and reach the steady state. Figure 3 illustrates an example of our proposed bootstrapping scheme for the network given in figure 1. The numbers in black color in the figure 3 depict the order of the nodes that join the network.

V. NODES MOBILITY AND FAILURE

A. Case of homogeneous WSN

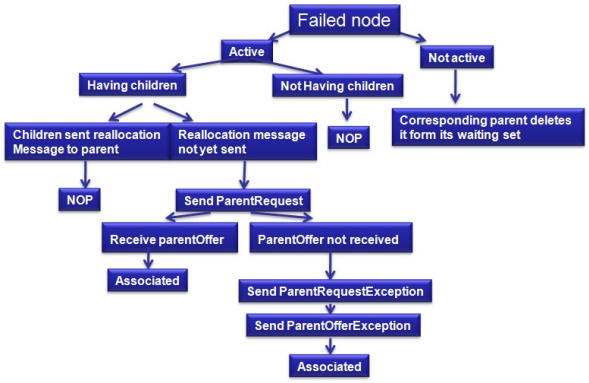

Figure 3. Tree reparation after node failure

In this section, we describe the behavior of our bootstrapping protocol at the case of nodes mobility and failure. The key idea is to detect the node departure by neighboring nodes and to try to heal the tree locally and to associate the orphan nodes of a leaving parent to other parents without the need to reconstruct the tree and without causing any inconsistency.

In order to detect that a node is no longer a physical neighbor, each node in the network that is already associated to a parent, broadcasts LIFE messages. In order to avoid interference problems, each node starts broadcasting these LIFE messages at a random time. Each node that is already associated and receives this LIFE message, sends an acknowledgement. If the source node does not receive any acknowledgement, it perceives that there is an interference problem.

When an interference is detected, the node rebroadcasts LIFE messages after another random time. This procedure is repeated until no interference is detected. From that moment, LIFE messages are broadcasted each T period from the successful broadcast and so on. The LIFE message contains the source node identifier and the identifier of the source node's parent. Each node that receives this message, adds the identifier of the source node in its physical neighbors set. Nodes that have received a LIFE message from a node whose parent is the same as their parent are considered in the same level. They add the identifier of the source node in their Level set. When the physical neighbors of a given node do not receive any LIFE message from it during two periods of T, they realize that this neighbor becomes unreachable. This can be due to two different reasons: the physical neighbor failed or moved and changed its place. The unreachable node can be not yet active and it can be also active and having children nodes.

In the first case, the corresponding parent of the unreachable node deletes it from its waiting set. In the second case, the children nodes of the moving parent can have multiple status:

1. Children nodes have already sent to their parent a Reallocation message: In such a case, the children nodes and their corresponding children have already joined the network, hence they do not need to be associated to another parent.
2. Children nodes have not sent a Reallocation message to their parent. Then, they need a parent to orchestrate their joining process if they have not yet joined the network and to orchestrate the joining process of their children otherwise.

In the second case, the orphan node broadcasts a ParentRequest message. This message contains the identifier of the old parent. We assume here that the network is sufficiently dense so that each node A is at least a physical neighbor of a node B having the same level as the parent of A. If there are nodes having received this message and belonging to the same level of the old node that do not have yet sent to their parent a Reallocation message, they will respond this orphan node with a ParentOffer message. When the orphan node receives the first ParentOffer message from a given node, it chooses the source node of this message as its new parent and sends it an Association message in order to become its child. After that, it ignores any other parent offer since it has been already associated to a parent.

When the orphan nodes do not receive any ParentOffer message from any nodes, they notice that the other nodes having the same level as their old parent, have already joined the network with their children. They will then broadcast a ParentRequestException message. When the nodes having the same level as the old node, receive these messages, they are aware that the orphan nodes did not find a new parent. They will then send ParentOfferException to these nodes.

These latter choose the source of the first received ParentOfferException message as new parent and respond it with an Association message. The parent node

that receives this message, adds the identifier of the source node in its waiting set and sends to the first active node in the network a PermissionException message. This message is routed using the used routing protocol (for example in our case, this message is routed to the destination using VRR). At the reception of this message, the first active node adds the identifier of the source node in its Additional Waiting set. When this active node receives Reallocation messages from all its waiting set, it sends a Permission message to the first node in its Additional Waiting Set using the appropriate routing protocol. At the reception of this message, the parent node sends a Permission message to the first node in
its waiting set and the procedure of bootstrapping already explained is repeated until the waiting set of the parent node becomes empty.
When this happens, the parent node sends a Reallocation message to the first active node which allows it to send a Permission message to the next node in its Additional Waiting Set. When the waiting set as well as the Additional waiting set of the first active node become empty, the network reaches the steady state.
The node that do not receive any LIFE messages from any node from its physical neighbors set from 2T period of time, realizes that it has been moved from its place. If a moving node is already active, it does not need to be reinserted in the bootstrapping tree. Since it does no longer belong to the tree, it does not respond to any Parent request or Parent Exception request message. The
moving node only needs to update the path between it and its virtual neighbors.
In most cases, this update can be achieved locally by replacing only the next hops towards its virtual neighbors. If this local repair fails, the path between the moved node and its virtual neighbors is reconstructed. If the moving node has not yet joined the network, it hears LIFE messages from its new physical neighbors nodes. There are different cases that can be presented:
1. There is a neighboring node that is active and that have not yet sent a Reallocation message to its parent. In this case, the mobile node sends a parentRequest message to this node and becomes associated to it.
2. All neighboring nodes are not yet active. The moved node does not receive in such case any LIFE messages. It should then wait for the activation of one of these neighboring nodes in order to be associated to it.
3. All neighboring nodes are active and have already sent Reallocation messages to their parents. Hence, the moved node sends a ParentRequestException message to the closest node to it. The latter node adds the moved node in its waiting set and sends to the first active node a PermissionRequestException.
Figure 4 gives an example of a node mobility. When the node 8 moves, its neighboring nodes do not receive from it any LIFE messages during a period T.

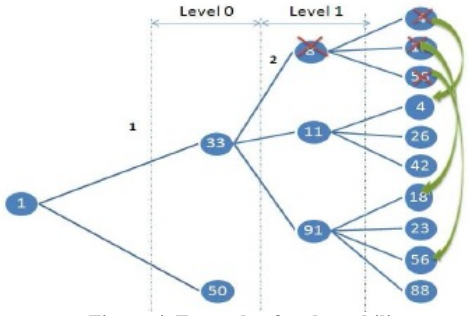

Figure 4. Example of node mobility treatment

Algorithm 1 presents the tree recovery process pseudocode and figure 3 summarizes the main tree recovery cases that can occur.

Its children nodes 4, 18 and 56 become orphans. They are not yet active (the joining process is still in step 2 as depicted in the figure). Hence, they will search a new parent. They start broadcasting ParentRequest messages containing the identifier of their old parent, the node 8. Nodes 11 and 91 find the node 8 in their level set. The node 11 sends to the node 4 a ParentOffer message. Similarly, the node 91 sends to the nodes 18 and 56 ParentOffer messages. Evidently, node 11 does not receive the ParentRequest messages that are sent by the nodes 18 and 56 because they are not its physical neighbors as shown in figure 1. When the node 4 receives the ParentOffer from the node 11, it sends it an Association message and becomes its new child. Nodes 18 and 56 become also the children of the node 91. The moving node 8 is in this case already active, hence it is aware that the nodes 1 and 33 are its virtual neighbors. In this case, the node 8 does not need to rejoin the tree. It only needs to update its virtual paths to its virtual neighbors depending on its new position.

B. Case of heterogeneous WSN

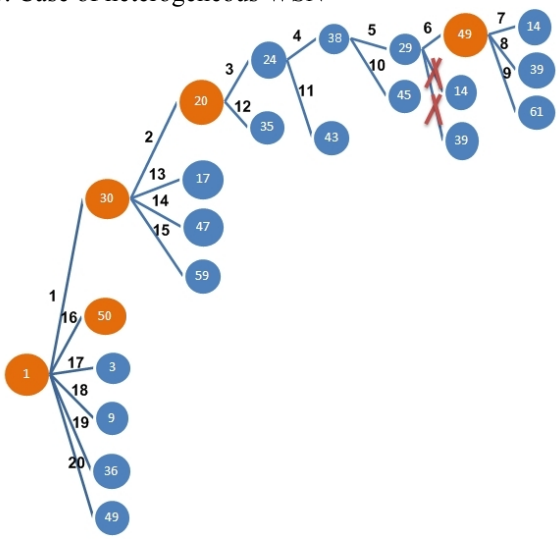

Figure 4. Bootstrapping in heterogeneous WSN

Most of WSN are made up of great number of sensors having different modalities and various characteristics. Taking into account such heterogeneity and especially energy variation, when conceiving a network protocol improves significantly network performance. We have proposed in [18], an energy-aware bootstrapping protocol for use in heterogeneous WSN. The main goal of this approach is to use at most energy powerful nodes in the bootstrapping phase in order to avoid the use of energy critical nodes and preserve their amount of energy. In this protocol, the priority of the joining process is given to energy-powerful nodes as depicted in figure 4. If a node that has not yet joined and is associated to a weak node receives HELLO message from an energy powerful node; it removes its association to the weak node and reassociates itself to the new strong node. This builds an powerful network backbone for use in the routing process.

In order to handle nodes failure and mobility in this bootstrapping protocol, we should also take into account nodes heterogeneity. Hence, the orphan node should not be associated directly to the first parent form which it has received a ParentOffer. Instead of that, it starts a timer of 1 second when it broadcast the ParentRequest message. During this time, if the orphan node has received a ParentOffer from an energy powerful node, it is associated to it. Otherwise, at the timer expiration, the orphan node is associated to a weak node since it has not found an energy-powerful parent.

VI. MOBILITY AND FAILURE DETECTION COST

The total bootstrapping phase lasts approximately 20 seconds in a 100-nodes network. Each node should broadcast a LIFE message each 1 second. Hence, each node needs to broadcast 20 messages during the bootstrapping phase. The nodes that detect the node departure need to broadcast ParentRequest messages. Since the bootstrapping period is very short, there is only so few nodes that can change their places during this phase. We assume that the maximal number of moving nodes in the bootstrapping phase is 4. There is on average 2 physical neighbors that are associated to a given parent. Hence, the maximal number of ParentRequest messages is 8 messages. There is also at most 8 ParentRequestException messages. Each orphan node has to be associated to a new parent by sending an Association message. Hence, there are on average 8 Association messages that are sent during the bootstrapping phase.

Nodes having the same level as the old parent send ParentOffer and Par entOfferException messages to the orphan nodes. There are on average 8 orphan nodes. There are hence on average 8 ParentOffer messages or 8 ParentOfferException messages (the nodes that send ParentOffer message to a node do not send it ParentOfferException and vice versa). When the ParentOfferException is accepted, the new parent sends PermissionException message. The maximal number of PermissionException messages is 8. The active node responds by at most 8 Permission messages. The total number of messages needed to detect a failure or a mobility and recover the bootstrapping tree is at most 2048 messages.

VII. SIMULATION

In order to investigate the feasibility of our proposed mobility support scheme in the bootstrapping phase, we have used simulations on NS2 [13]. We analyzed the behavior of the physical neighbors in order to recover the bootstrapping tree at the case of nodes mobility and/or failure. The basic configuration simulates 50 nodes distributed randomly over 100 m*200 m plane. We vary the number of simultaneous mobile nodes in the network and measure the time and the energy needed by the neighboring nodes to recover locally the tree without the need of whole reconstruction. It is clear that when a node has no children or has an empty waiting list, its neighboring nodes do not need to exchange any messages and need only to delete the moved node's identifier from the physical neighbors list. That's why we have chosen the time of the nodes mobility in a way that the mobile node has already some children in its waiting set in order to study the reassociation process of these children nodes to new parents.

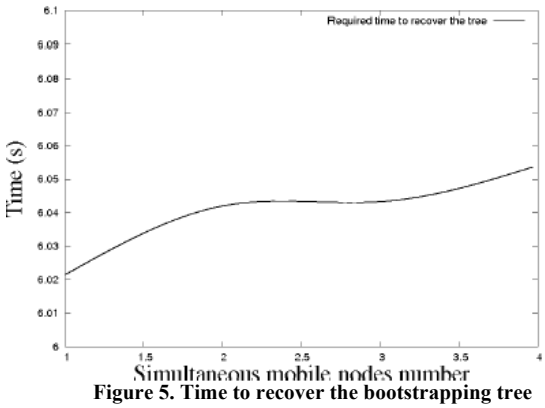

Figure 5. Time to recover the bootstrapping tree

Figure 5 clearly shows that neighboring nodes of moving ones are able to recover the tree locally and to find new parents to the orphan nodes in a very short time (6 seconds) even if the number of moving nodes increases.

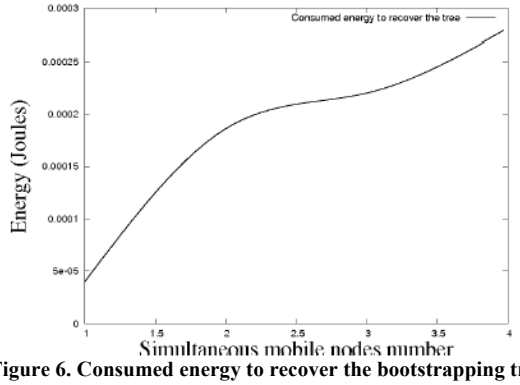

Figure 6. Consumed energy to recover the bootstrapping tree

Figure 6 measures the total amount of consumed energy at the recovery phase. Evidently, this amount increases slightly when the number of mobile nodes increases

because the number of exchanged messages to recover the tree increases as well. However, it is clear that the total consumed energy is so limited.

VIII. CONCLUSION

In this paper, we have improved our bootstrapping protocol in order to take into account nodes mobility and failure cases. Nodes detecting the failure or the movement of a node, try to heal the tree locally and in a totally distributed manner without the need to reconstruct all the tree. This ensures ring global consistency and avoids time and energy wastage of global tree reconstruction.

REFERENCES

[1] Akyildiz, I. F. , Su, W. , Sankarasubramaniam, Y. and Cayirci, E. , Wireless sensor networks: A survey Computer Networks,38(4),393-422,2002. http://dx.doi.org/10.1016/S1389-1286(01)00302-4

[2] Yick, J. , Mukherjee, B. and Ghosal, D. , Wireless sensor network survey Computer Networks, 52(12),2292-2330, 2008. http://dx.doi.org/10.1016/j.comnet.2008.04.002

[3] Lewis, F.L. ,Wireless sensor networks. Smart environments: Technologies, protocols, and applications. New York: Wiley, 2004.

[4] Fersi, G. , Louati, W. and Ben Jemaa, M. , Distributed Hash Table-based Routing and Data Management in Wireless Sensor Networks: a Survey: ACM/Springer Wireless Networks,19 (2), pp 219-236.

[5] Caesar,M.,Castro,M.,Nightingale,G.OShea,E. and Rowstron,A., Virtual ring routing: Network routing inspired by DHTs: In Proceedings of SIGCOMM, Italy, 2006.

[6] Fuhrmann,T. ,The use of scalable source routing for networked sensors:In Proceedings of the 2nd IEEE workshop on embedded networked sensors, Australia, 2005.

[7] Wehrle,K. , Gtz,S. and Rieche,S. , Distributed Hash tables: In R. Steinmetz, K. Wehrle (Eds.), Peer-to-Peer systems and applications (Chapter 7, pp. 79-93). Berlin, Heidelberg: Springer, 2005. http://dx.doi.org/10.1007/11530657_7

[8] Cramer, C. and Fuhrmann, T. ,Self-stabilizing ring networks on connected graphs: University of Karlsruhe (TH), Technical report 2005-5, 2005.

[9] Kutzner,K. and Fuhrmann,T. ,Using linearization for global consistency in SSR: In Proceedings of international IEEE workshop on hot topics in P2P systems,CA,2007.

[10] Almamo, A. and Labiod, H. , ScatterPastry: An overlay routing using a DHT over wireless sensor networks: In Proceedings of the international conference on intelligent pervasive computing, Korea, 2007.

[11] Almamou, A. , Schiller, J. , Labiod, H. and Mesut, G. , A Case for an overlay routing on top of MAC layer for WSN: In Proceedings of the second international conference on sensor technologies and applications, France, 2008.

[12] Fersi, G. , Louati, W. and Ben Jemaa, M. , Energy-aware virtual ring routing in wireless sensor networks, Journal of Network Protocols and Algorithms,2(4),16-29, 2010.

[13] NS2 website Available at http://www.isi.edu/nsnam/ns/

[14] Malkhi, D., Sen, S., Talwar, K., Werneck, R. and Wieder, U.,Virtual Ring Routing Trends, DISC 2009.

[15] Fersi, G. , Louati, W. and Ben Jemaa, M. , Consistent and Efficient Bootstrapping Ring-Based Protocol in Randomly Deployed Wireless Sensor Networks Intenational Conference on telecommunications (ICT), Maroc, 2013 (to appear).

[16] Bradonji, M. and Kong, J. S., Wireless Ad Hoc Networks with Tunable Topology, Forty-Fifth Annual Allerton Conference Allerton House, USA, 2007.

[17] Christin, D., Reinhardt, A. , Mogre, P. , Steinmetz, R. , Wireless Sensor Networks and the Internet of Things: Selected Challenges, Proceedings of the 8th GI/ITG KuVS Fachgesprach ”Drahtlose Sensornetze”, 2009.

[18] Fersi, G., LOUATI, W., BEN JEMAA, M., Energy-Aware Distributed Hash Table based Bootstrapping Protocol for Randomly Deployed Heterogeneous Wireless Sensor Networks. 28th International Symposium on Computer and Information Sciences (Iscis’2013), Springer, Paris, France, 2013.

AUTHORS

Ghofrane FERSI is a Ph.D student at the National School of Engineers of Sfax, Tunisia (e-mail: ghofrane.fersi@redcad.org).

Wassef LOUATI is an associate professor at the Higher Institute of Computer Science of Mahdia, Tunisia (e-mail: wassef.louati@redcad.org).

Maher BEN JEMAA is a keynote professor at the National School of Engineers of Sfax, Tunisia (e-mail: maher.benjemaa@enis.rnu.tn).

Submitted 22 June 2013. Published as re-submitted by the authors 23 July 2013.