A Dynamic and Distributed Addressing and Routing Protocol for Wireless Sensor Networks

Tomás Sánchez López[1] and Gonzalo Huerta-Canepa[2]

[1] EADS UK Innovation Works, Newport NP10 8FZ, UK
tomas.sanchezlopez@eads.com
[2] Korean Advanced Institute of Science and Technology, Daejeon 305-701, S. Korea
gonzalo@huerta.cl

Abstract. Wireless Sensor Networks (WSNs) have traditionally had static topologies that require little or no maintenance from their addressing mechanisms. New research directions have introduced additional scenarios where WSNs should adapt to the environment and network conditions. This shift in requirements also means that the addressing schemes should evolve dynamically, providing a distributed and lightweight way of assigning addresses to the sensor nodes. In addition, the routing protocols, which drive the information exchange within the network, must leverage the addressing properties in order to function efficiently. In this paper, we present Sequence Chain (SC) as a lightweight tree-based addressing mechanism for Wireless Sensor Networks. SC offers distributed and dynamic address allocation together with optimized tree routing. We compare our work with the de-facto industry standard, ZigBee, and show how SC improves the formation of addressing trees and yields up to three and a half times shorter multi-hop routes.

Keywords: Wireless Sensor Networks, Addressing, Smart Objects.

1 Introduction

Mobile and Ad-hoc Networks (MANETs) are mobile networks of computing devices that are able to communicate amongst each other without the use of an infrastructure. MANETs devices run typically on batteries and use wireless communication. Unlike wired networks, which lack strong power and infrastructure constraints, MANETs must optimize their operation and keep the connectivity even when unexpected topology changes occur. Wireless Sensor Networks are similar to MANETs in that they are formed by wireless devices which must communicate among themselves without the use of an infrastructure. However, traditional WSNs do not consider mobility among their sensor nodes but rather assume static topologies that remain after deployment for all the network lifetime. Furthermore, WSNs are also traditionally centralized in the sense that a Base Station collects all the sensor readings from the network members.

Dynamic address allocation is a common problem for MANETs where mobile nodes constantly join and leave the network. Most of the approaches are based on the Internet Protocol (IP). Node mobility is typically handled by the IP standards themselves, whose objective is to provide transparent routing allowing nodes to conserve their original address while moving around different subnetworks. Automatic address allocation, however, poses additional challenges. A number of approaches rely on centralized agents that assign new addresses according to a pool of available alternatives [5,13]. Other approaches allow decentralized address assignment but require Duplicate Address Detection (DAD) techniques in order to ensure unique addresses [12,9].

Recent developments on Wireless Sensor Networks [7,2] make the inclusion of dynamic address allocation a desirable feature. However, the dynamic addressing techniques used in MANETs are inappropriate for WSNs due to their specially scarce resources. Typical WSNs node limitations include limited RAM and Flash memories, slow processing, short radio ranges and short-lived batteries. IP addressing is generally unsuitable for WSNs due to the excessive overhead of the IP protocol stack. Furthermore, centralized sensor node agents are also inappropriate since the sensor nodes's energy is easily exhausted. Finally, approaches that use DAD should also be avoided because of the message overhead caused by the search for duplicate addresses throughout the network.

In this paper, we propose a lightweight addressing scheme that provides excellent dynamic properties while minimizing the overhead of the assignment process. Our addressing scheme, called Sequence Chain (SC), assigns addresses in a hierarchical manner, additionally enabling simple and efficient routing among the network nodes. In this paper, we prove that both the functionality and the performance of SC are superior to any other related work, including the de-facto industry standard for Wireless Personal Area Networks (WPAN) ZigBee [16].

The rest of this paper is organised as follows. Section 2 reviews the related work in the area. Section 3 and 4 introduces our proposed scheme. Section 5 compares SC with ZigBee as part of our evaluation. Finally section 6 concludes the paper.

2 Related Work

Address allocation in ad-hoc and wireless networks has been extensively discussed over the last years. Many protocols have been proposed that try to address the problems of this type of networks, such as mobility, node joining and separation, network merge, etc. Although the specific constraints of WSNs restrict even more the protocol design, very little work has been done in the area. Besides few exceptions, most existing WSNs architectures just provide static addresses or assume that generic protocols designed for ad-hoc wireless networks would fit into the sensor nodes. We have reviewed those existing approaches that are more relevant to our work, and we present a summary of them on table 1. Although due to spece restriction we can not provide a detailed discussion on the related work, the following list describes the criteria that we followed to comparing and reviewing them. As the table shows, with the support of mobility and the splitting and merging of networks, Sequence Chain is the only protocol that fully considers dynamic WSNs.

- *WSN:* If the scheme was developed specifically for WSNs. This characteristics indicates that more concern was put on the constraints of the nodes.
- *Mobility:* If mobility of nodes is specifically considered.
- *Split/Merge:* If network split or network merger are supported.
- *Unique:* If the result of the address assignment is a unique address, without the need of any additional protocol such as DAD.
- *Concatenation:* If the assigned addresses are a result of concatenating local identifiers. This is the technique followed by our proposal.
- *Distributed:* If the assignment process is distributed (no centralized agents).
- *Overhead:* The stimated overhead of the address assignment process, including duplicate address detection and address look-up costs when applicable.

Table 1. Comparison of relevant addressing protocols

Scheme	WSN	Mob.	S/M	Unique	Conc.	Dist.	Overhead
MANETconf [9]	No	No	Yes	No	No	Yes	$O(n)$
DACP [5]	No	Yes	Yes	No	No	Yes	$O(n)$
PACMAN [12]	No	No	No	No	No	Yes	$O(1) + F(P)$‡
DART [6]	No	Yes	Yes	Yes	No	Yes	$O(logn), O(n)$†
L^+ [3]	No	Yes	No	Yes	Yes	No	$O(1), O(logn)$†
Yao and Dressler [14]	Yes	No	No	No	No	No	$F(P) + O(n)$‡
Bhatti and Yue[1]	Yes	No	No	Yes*	No	Yes	$O(1) + O(n)$
ZigBee [16]	Yes	No	No	Yes	No	Yes	$O(1)$
Zheng & Lee [15]	Yes	No	No	Yes	No	Yes	$O(1)$
Sequence Chain	Yes	Yes	Yes	Yes	Yes	Yes	$O(1)$

† Address assignment overhead, Address look-up overhead
‡ $F(P)$ is a function of the probability that the selected address is not duplicated
dependent on the destination agent

3 The Sequence Chain Addressing Scheme

In our previous work [10][11], we proposed an architecture and a set of protocols that drive the formation of Smart Object (SO) networks. Wireless sensor nodes were attached to the objects to implement those protocols and to represent the objects in the architecture. As a requirement, SOs should be able to leave or join a network at any time, and so the SO network protocols need to support mobility tailored for the constrained WSNs.

Distributed addressing schemes such as ZigBee [16] or LEADS [8], fix their address space to a number of bits. Starting from this fixed address space, they assign addresses by, for example, allowing parents to distribute portions of the space to their children. This assures global unique addresses and fairness in the assignment. However, portions of the address space become easily exhausted, causing the rejection of new nodes while other portions of the space remain free. Furthermore, even if a low number of nodes are part of the network, their addresses still occupy the full number of bits, wasting memory in packets and routing tables and also processing time in address comparison and others.

In SC, rather than partitioning a fixed address space we construct addresses by attaching locally unique identifiers to build a global unique address. Each parent assigns its children a sequence number that distinguishes it locally from its siblings. This sequence number is then attached to the address of the parent to form the final child's address. This way, any node will be assigned a unique address in a simple, distributed, hierarchical way. The simplicity of the address assignment allows SC to provide advanced functionality with relatively low cost. The proposed addressing scheme has the following properties:

– *Hierarchical*: Nodes receive addresses organized in a tree structure. The node from the network that accepts a joining request of a new node becomes a parent. Hence, senders of joining requests become children.
– *Distributed and unique*: Each node is responsible for assigning addresses only to its children. The address that a child receives is derived from its parent address in a way that makes that address unique for the network.
– *Scalable*: If a node leaves a network, its address becomes automatically available for any other node joining with the same parent. The addresses increase in size as the network becomes bigger. Network merges reassign the addresses of the network with the smallest number of nodes.
– *Low overhead*: A parent only needs to know its immediate children to assign addresses in a unique manner. The addressing scheme provides routing along the tree with nearly no cost. A node can know how many hops it is away from any destination by just analyzing the destination address, and parents can route packets following the tree by just comparing their address with the packet's destination address. Additionally, parents can provide shortcuts to the destination by routing packets to neighbors that are closer to the destination than following the tree.

3.1 Node Join and Network Merge

As described in our previous work [10], Smart Objects periodically send request packets to discover other SOs and networks. One of the purposes of those discovery packets is to trigger an address allocation process. When the process is over, the SO that received the advertisement packet will become the *parent*, and the SO that sent the advertisement packet will become a new *child* of that parent. This parent/children relationship creates a hierarchical relationship among all the nodes of a network, resulting in a *tree* structure. Every parent but the root of the tree (the first node of a network) assigns local identifiers to its children in a sequentially increasing order, starting from `0`. The root of the tree assigns itself the address `0`, which is both a local and a global tree address. For this reason, a root starts assigning addresses to its children from `1`. To form a global unique address, a child attaches its local address to the address of its parent. As a result, the address of every node of a network contains all the addresses of its parents from its immediate parent to the tree's root. Since it is not possible to specify the separation of parent-child portion within the address, the number of bits used for each child address must be equal in order to ensure that addresses are unique. This number is a network-wide value and is called bits-per-level

(BPL). For example, two bits will provide four children per parent, three bits eight children, etc. Figure 1 shows an example with two BPL and node addresses displayed in binary format.

Sequence Chain does not fix the address space but allows addresses to grow in length as needed (i.e. as the number of the levels of the tree grows), effectively providing a dynamic address space. An address length (i.e number of bits) is determined by the level (i.e depth) of the node holding the address and the BPL variable ($length = depth * BPL$). Thus each node in the network can know the length of its own address and the length of the addresses of its parent and children. Additionally, given any address a node can know the address' node depth since the BPL variable is known network wide. Addresses can also be found in packets, as source and destination end-points, and in neighbour tables (see section 4). In order to be able to determine the length of arbitrary addresses and thus be able to extract and process them adequately, the level of the node to which each address belongs is included together with the addresses. Although the size of this *level* field would depend on the maximum desired depth for a network, 5 or 6 bits (32 and 64 levels respectively) would be enough to fulfil the needs of networks of millions of nodes while giving enough flexibility for unequal tree growth. For an evaluation on the SC address lengths please see section 5.2.

Algorithm 1. JoinAdvRequest reception

```
Input: JoinAdvRequest
Output: JoinAdvResponse
HAL = HighestSOnet#(ListOfRequests + Myself) ;
if (NetworkID ϵ HAL) then
    SOnet# = SOnet# + Sum(List of AssociationRequest.SOnet#) ;
    SOtree# = SOtree# + Sum(List of AssociationRequest.SOnet#) - 1;
    Store(List of AssociationRequest.SOtree#) ;
    send JoinAdvResponse(NetworkID, SOnet#, ComputeAddress(Address), Address,
    NoReverse) to ListOfRequests & children ;
end
```

Algorithm 2. JoinAdvResponse reception

```
Input: JoinAdvResponse(NewNetID,SumSOnet#,
NewAddress,ParentAdd,NoReverse)
Output: JoinAdvResponse
NetworkID=NewNetID ;
Parent_address=ParentAdd ;
Address=NewAddress;
SOnet# = SOnet# + SumSOnet# ;
if (Reverse) then
    MakeChildParent ;
end
if (HaveChildren) then
    JoinAdvResponse(NewNetID, SOnet#, ComputeAddress(NewAddress), Address,
    NoReverse) to children ;
end
if (HaveParent) then
    MakeParentChild ;
    send JoinAdvResponse(NewNetID, SOnet#, ComputeAddress(NewAddress), Address,
    Reverse) to parent ;
end
```

Network merging is supported by re-assigning the addresses of the network with the *least* number of nodes (i.e. the merged network). The number of nodes of each network is calculated in a distributed way and requires no global knowledge. The network merging process is very similar to joining individual nodes, and every new child's address is computed entirely by its parent according to its own address and the sequence of the child. The node from the merged network that serves as the bridge for the merge (i.e. the node that sent the request packet - JoinAdvRequest) will obtain its new address from its new parent, the same way as if it was a single node. Once its new address is computed, it will propagate a re-assignment command to its children, which in turn will propagate it to their own children. This process will continue until all the nodes of the merged network are assigned a new address. Join requests may be sent and received by nodes which are not the root of the tree. As a result, network merging often incurs in a parent-child misplacement, where links from the merged network must be reversed in order to keep the hierarchical structure of the tree.

Algorithms 1 and 2 detail the process involved in the joining and merging of networks. The same algorithm is utilized for both single node joining and network merge, since a single node is considered as a network which contains only one node. Sequence Chain also supports the joining of several networks simultaneously. The `ListofRequests` variable contains all the requests that were received by this node in a particular request period.

When a merge occurs, a new network ID must be elected in order to represent the new network. The election of a new network ID is influenced by the number of nodes in a network. As mentioned before, the number of nodes also dictates which one of the networks that are merging will change its addresses. The *SOnet#* and *SOtree#* variables store the total number of nodes of the network and the number of descendants of a given node respectively. The *SOtree#* will be used in section 3.2 in order to update the *SOnet#* variable when a subtree leaves the network. The algorithm generates a `JoinAdvResponse`, that is sent to the requesting nodes with the addresses computed as described earlier. Upon reception, the node receiving the `JoinAdvResponse` should first proceed with the changes indicated in the packet: update its Network ID information, its own address and the address of its parent. The *Reverse* and *NoReverse* arguments indicate if the child-parent relationship among the node that sends the `JoinAdvResponse` packet and the node that receives it should be reversed, as explained earlier. Finally, the changes reported by the `JoinAdvResponse` packet should also be transmitted to the rest of the nodes of the merged networks. The algorithm achieves that by sending `JoinAdvResponse` in a recursive manner until all the members are notified.

3.2 Node Leave and Network Split

When a network leave occurs, a tree link is broken, in which one of the ends was a parent and the other one was a child. We assume that both parent and child become aware of their broken bond in a reasonably low amount of time. Algorithm 3 depicts what is the procedure when either child or parent notices the broken link. Children that find their parent missing will attempt to find a new parent from the same network, choosing the parent with the shortest address. A parent that finds one of its children missing, will first wait for any *orphan* node to request to rejoin the network. The algorithm also elects a new network identifier when needed and updates the node count of both the network and each node's subtree. Note how all the procedure is distributed and does not depend on any node warning about its departure, but just on nodes noticing that their parent or children are not responding.

Algorithm 3. *Leaving* procedure

```
if (Parent missing) then
    AnswerList=SearchNewParent(NetworkID) ;
    if (AnswerList ≠ null) then
        NewParent=SelectShortestAddress(AnswerList);
        MakeParent(NewParent) ;
        rejoin=True ;
        SOnet#=(SOtree#+1)+NewParent.SOnet# ;
    else
        rejoin=False; Address=rootAddress; SOnet#=SOtree# + 1;
    end
    UpdateChildren(Address, SOnet#); CheckNetIDleft() ;
end
if (Child missing) then
    SOnet#=SOnet# - (Child.SOtree# + 1) ;
    SOtree#=SOtree# - (Child.SOtree# + 1) ;
    ListOfOrphans=wait(OrphansToJoin) ;
    if (ListOfOrphans = null) then
        CheckNetIDleft() ;
    else
        SOnet#=SOnet#+(Orphan.SOtree# + 1);
        SOtree#=SOtree#+(Orphan.SOtree# + 1) ;
    end
    UpdateTree(SOnet#,Difference(SOtree#)) ;
end
```

4 Sequence Chain Routing

The Sequence Chain addressing scheme provides tree routing by simple comparison of addresses. This is possible due to two simple properties:

- Any node's address contains the addresses of all its parents towards the root.
- The number of maximum children for each parent is fixed network-wide. This means that the number of address BPL is fixed.

Let `dest` be the destination address of a packet and `destL` the length of that address. In a similar way, let `router` be the address of the node that receives the packet and is asked to route it to the destination, and `routerL` the length of that address. Let `nbits` be the number of BPL and '$\times$' and '$*$' the concatenation and arithmetic multiplication operations respectively. Fundamental properties of Sequence Chain are:

1. $destL = nbits * n$, where $\mathtt{n} \in \aleph$. In a similar way, $routerL = nbits * m$, where $\mathtt{m} \in \aleph$.
2. if $dest = x \times k$ and $router = x \times j$, where $k \neq j$, then $\frac{length(k)+length(j)}{nbits}$ is the number of hops from the router node to the destination. This is also true if $length(x) = 0$.
3. if $dest = router \times y$, then the destination is one of the children in `router`'s subtree
4. if $dest = router \times y$, and $y = z \times v$, $router \times z$ is the next hop to $dest$, where $length(z) = nbits$ and z is the local address of one of the children of $router$.

In the previous properties, x, k, j, y, z and v are portions of addresses and their length also follow property 1.

The general tree routing operation is then as follows: When a node `router` receives a packet and is asked to route it to `dest`, it first checks if its address `router` is contained in the destination address `dest`. If so, it means that the destination is in its subtree, and the next hop is computed as in property 4. If its address is partially contained or not contained at all, the packet must be sent to its parent.

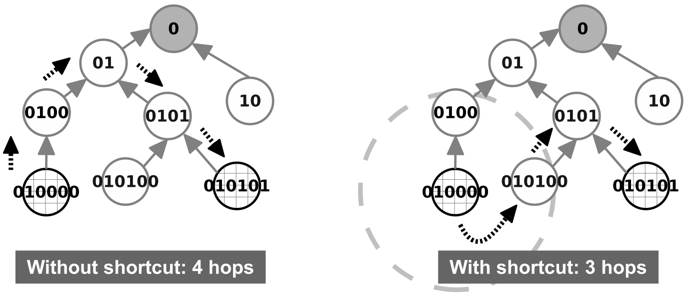


Fig. 1. Example of routing in Sequence Chain. BPL=2.

It is possible to provide shortcuts to the destination by analyzing the addresses of the neighbour nodes before proceeding with the general tree-routing. Neighbour addresses can easily be obtained by overhearing packets at node joining time, routing or link health checking, and storing them in a neighbour table. Figure 1 shows an simple example of how one routing hop can be saved by analysing the address of the neighbouring nodes of the origin node of a packet. Note that this technique can be applied at any point on a routing chain, every time a routing decision has to be made.

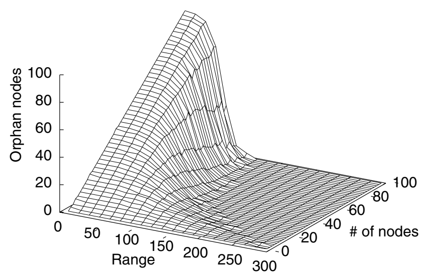
It also worth noting that networks built using the SC addressing scheme can also benefit from the addressing properties outlined earlier even if tree routing is not used. For example, in many mesh routing mechanisms, such as AODV or DSR [4], the initial effects of flooding for routing discovery can be mitigated by limiting the number of hops that the broadcast will be propagated. In SC networks, this number can be set to the number of hops between source and destination nodes if the SC tree routing was used.

5 Evaluation

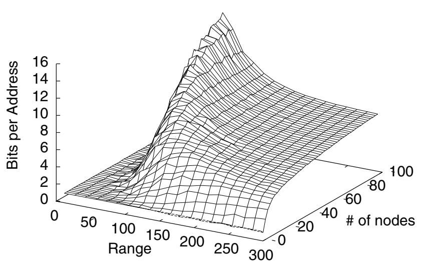
According to section 2, ZigBee is the most similar related work to Sequence Chain as well as being a widely used industrial standard for WSNs. Therefore, in this section we will analyse a number of metrics aiming to demonstrate that our proposed addressing mechanism is superior to ZigBee. The rationale behind each one of the metrics is detailed on each of the subsections below. The simulator introduced in [10] was used to obtain all the evaluation results presented in the following sections. Common parameters of all the simulation scenarios include:

– A square deployment area of 500x500 meters.
– A Sequence Chain BPL of two. This limits the maximum number of direct children per node to four.
– ZigBee parameters of *nwkMaxChildren*=4 and *nwkMaxRouters*=4. In this way, the maximum number of children is the same as SC, and all the children are capable of having other children (i.e. being *routers*) in the same way as SC. Since the *nwkMaxDepth* variable very much affects the results, it will be varied throughout our simulations
– For each simulation scenario, 100 repetitions with randomized node locations.

The method adopted for the simulations was the following: the programmed disconnected nodes were positioned randomly in the deployment area, and the node closest to the center of such area was identified. This node was assigned the address '0' and made part of the tree (root and initiator of the network). For each other node in an increasing distance from the initiator, a parent node was selected, which was 1) in the transmission range of the node 2) already part of the tree and 3) had an available child address. For all the potential parents meeting these requirements, the parent which was in the lowest level (least depth) in the tree was selected. After a parent was chosen for a node, the following node was selected and the process repeated until all the nodes in the deployment area had searched for a parent.



(a) Orphan nodes



(b) Bits per address

Fig. 2. Average orphan nodes and maximum depth resulting from simulations of SC

5.1 Orphan Nodes

Those nodes that are unable to find a parent are considered *orphan*. Orphan nodes may appear for two main reasons: no parent was found in the transmission range of the selected node, or parents were found but they were unable to accept more children. As a general rule, the smaller the number of orphans that appear in a random deployment scenario, the better the addresses are being assigned.

Graph 2(a) shows how the average orphan nodes from the SC simulations vary when both the number of nodes in the simulation and their transmission range varies. As the graph shows, when the number of nodes increases, the number of orphan nodes tends to decrease since the density of deployed nodes allows more nodes to find an available parent. At the same time, the number of orphan nodes decreases as the transmission range increases, since the chances of finding available parent nodes also increases. For ZigBee, instead of showing one graph for each value of *nwkMaxDepth*, table 2 shows how often the number of orphan nodes resulting from the SC simulations are higher than, lower than or equal to those of the ZigBee simulations for each simulated value of *nwkMaxDepth*. From Table 2, we see that SC obtains a lower number of orphan nodes for almost any value of *nwkMaxDepth*, and that the improvement of SC is lower in the middle values of *nwkMaxDepth*.

Table 2. Comparison percentages of orphan nodes between SC and ZigBee

nwkMaxDepth → ↓ Orphan nodes	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$SC > ZigBee$	0%	1%	2%	4%	7%	9%	11%	0%	1%	1%	1%	0%	0%	1%
$SC < ZigBee$	81%	75%	53%	23%	16%	10%	9%	13%	32%	40%	45%	49%	52%	55%
$SC = ZigBee$	19%	24%	45%	73%	77%	81%	80%	87%	67%	59%	54%	51%	48%	44%

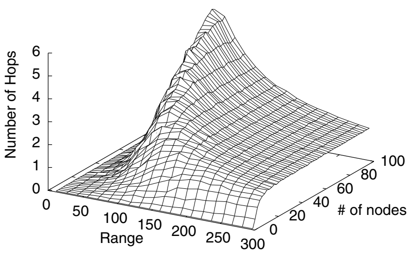
5.2 Bits Per Address

ZigBee uses fixed 16 bit network addresses and so can accommodate a maximum of 65536 nodes per network. The address assignment mechanism uses chunks of the 16 bit address space that are assigned by a parent to its children. As a consequence, children are allocated addresses according to the potential descendants that they may have. Previously it was shown that random joins tend to produce more orphan nodes in ZigBee, which fixes its addressing space for each child. Also, if the number of nodes per network is not very high, fixed large addresses waste valuable memory space both in the nodes themselves (e.g. in neighbour tables) and in the protocols that use the addresses for communication (e.g. message exchange). SC allows the increase of address lengths on demand when the number of nodes increases, as long as the number of BPL is respected. This strategy not only produces shorter addresses, but also provides more flexibility by not limiting the maximum number of nodes in any part of the addressing tree.

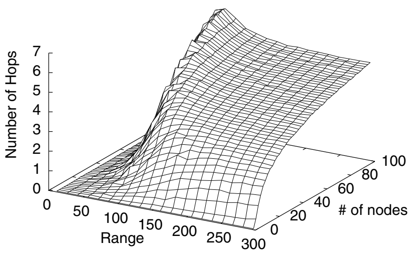
Graph 2(b) shows the variation of the number of address bits as both the number of nodes and their transmission range are varied. Results show that the Sequence Chain addressing scheme always produces shorter addresses than ZigBee (i.e under 16 bits). Addresses are longer for high number of nodes and relatively low ranges. These conditions coincide with the location of the deepest trees, and therefore with the longest routes (Graph 3(a)). Only 19% of the addresses generated by SC are longer than 8 bits, which would be beneficial even in case of memory restrictions (for example if a byte is the minimum addressable memory portion). Note that even if we consider the 5 or 6 bit `level` field needed to calculate an address length as part of its bit count (as mentioned in section 3.1), the resulting address lengths are still 25 to 30 per cent shorter than those of ZigBee.

5.3 Tree Depth and Routing

Nodes looking for parents to join the tree always choose the candidate which is located in the lowest level of the addressing hierarchy. However, depending on the addressing scheme, the list of candidates may differ and the trees depth might grow in different ways. Trees with larger maximum depths are potentially less efficient, since messages have to traverse longer distances from the origin to the destination if the addressing tree is followed for routing. We analyzed the performance of tree routing between Sequence Chain and ZigBee by measuring the number of hops between origin and destination nodes on randomized communication scenarios. For each deployment scenario, addressing trees were formed for both Sequence Chain and ZigBee addressing schemes. Finally, origin and destination were randomly chosen and communication simulated between them. This last step was repeated fifty times, and the average number of hops was recorded for those repetitions.



(a)



(b)

Fig. 3. Average number of hops with randomized routing scenarios using the SC with shortcut 3(a) and ZigBee 3(b)

Table 3. Distribution of simulated average route lengths for different tree depths

nwkMaxDepth → ↓ Average n°of hops	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$SCshortcut > ZigBee$	89%	36%	23%	15%	12%	9%	10%	4%	15%	23%	25%	26%	26%	27%
$SCshortcut < ZigBee$	8%	61%	74%	82%	85%	88%	87%	92%	82%	74%	72%	71%	71%	70%
$SCshortcut = ZigBee$	3%	3%	3%	3%	3%	3%	3%	4%	3%	3%	3%	3%	3%	3%

Figure 3(a) shows the results obtained by using the Sequence Chain addressing and routing with the shortcut mechanism. Figure 3(b) show the results using the ZigBee tree routing. In order to compare these differences with different values of *nwkMaxDepth*, Table 3 presents a percentage comparison among these 2 routing algorithms. Sequence Chain with the shortcut mechanism obtains significantly shorter routes as compared with ZigBee. We have recorded differences of up to 3.8 hops for some scenarios, which is 3.5 times shorter than ZigBee.

6 Conclusion

In this paper, we present the tree-based Sequence Chain addressing mechanism for Wireless Sensor Networks. SC proposes a simple and distributed way of assigning addresses to nodes inside a WSN, and supports all the features required by dynamic and mobile environments such as node join and leave or network split and merge, as well as mechanisms for avoiding address space exhaustion problems. By means of the hierarchical way in which addresses are assigned, SC can provide extremely simple and efficient routing along the tree. We demonstrate that SC provides notable benefits in tree formation and routing, and generally produces more stable networks than ZigBee, in which the number of orphan nodes and tree depths is balanced and does not depend on protocol variables or deployment scenarios. We conclude that the Sequence Chain addressing mechanism is adequate for dynamic and mobile WSNs, and that it provides excellent performance both in tree formation and in tree routing. Additional results and an overview of the conclusions on the context of a Smart Objects architecture can be found in [11].

References

1. Bhatti, G., Yue, G.: A structured addressing scheme for wireless multi-hop networks. Tech. rep., Mitsubishi Electric Research Labs Tech. Report (2002)
2. Butler, Z., Rus, D.: Event-based motion control for mobile-sensor networks. IEEE Pervasive Computing 2, 34–42 (2003)
3. Chen, B., Morris, R.: : Scalable landmark routing and address lookup for multi-hop wireless networksl^+. Tech. rep., MIT LCS Technical Report 837 (2002)
4. Johnson, D., Hu, Y., Maltz, D.: RFC 4728: The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. The IETF Trust (2007)
5. Elizabeth, Y.S., Sun, Y., Belding-Royer, E.M.: Dynamic address configuration in mobile ad hoc networks. Tech. rep., Computer Science, UCSB, Tech. Rep (2003)
6. Eriksson, J., Faloutsos, M., Krishnamurthy, S.V.: Dart: Dynamic address routing for scalable ad hoc and mesh networks. IEEE/ACM Transactions on Networking 15(1), 119–132 (2007)
7. Heo, N., Varshney, P.: A distributed self spreading algorithm for mobile wireless sensor networks. In: 2003 IEEE Wireless Communications and Networking, WCNC 2003, vol. 3, pp. 1597–1602 (2003)
8. Lu, J.L., Valois, F., Barthel, D., Dohler, M.: Low-energy address allocation scheme for wireless sensor networks. In: IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2007, pp. 1–5 (2007)

9. Mesargi, S., Prakash, R.: Manetconf: Configuration of hosts in a mobile ad hoc network. In: Proc. of INFOCOM 2002, New Yok, NA, pp. 1059–1068 (2002)
10. Sánchez López, T., Kim, D., Huerta Canepa, G., Koumadi, K.: Integrating wireless sensors and rfid tags into energy-efficient and dynamic context networks. The Computer Journal 52(2), 240–267 (2008)
11. Sánchez López, T., Ranasinghe, D., Harrison, M., McFarlane, D.: Adding sense to the internet of things. Personal and Ubiquitous Computing, 1–18 (2011)
12. Weniger, K.: Pacman: Passive autoconfiguration for mobile ad hoc networks. IEEE Journal on Selected Areas in Communications 23(3), 507–519 (2005)
13. Weniger, K., Zitterbart, M.: Ipv6 autoconfiguration in large scale mobile ad-hoc networks. In: Proceedings of European Wireless 2002, pp. 142–148 (2002)
14. Yao, Z., Dressler, F.: Dynamic address allocation for management and control in wireless sensor networks. In: Proc. of HICSS 2007, p. 292b (2007)
15. Zheng, J., Lee, M.J.: A resource-efficient and scalable wireless mesh routing protocol. Ad Hoc Networks 5, 704–718 (2007)
16. ZigBee Alliance: ZigBee Specification. ZigBee Alliance (2008)