Node Addressing Schemes for Scalable and Fault Tolerant Routing in Hierarchical WSNs

Anurag D
Indian Institute of Management Calcutta
anurag@email.iimcal.ac.in

Somprakash Bandyopadhyay
Indian Institute of Management Calcutta
somprakash@iimcal.ac.in

Abstract— Most wireless sensor network deployments are 2-tiered where sensors form the leaves of the network and do not participate in the routing. A plot of the best path from each of the leaves to the sink reveals the network topology to be hierarchical in nature. The AODV routing algorithm was designed for a mesh network with highly mobile nodes and is not directly suitable for a hierarchical sensor network where the sensors and relays are predominantly static. The hierarchical routing as implemented by ZigBee's C_{skip} does not support fault tolerance and has a restriction on the network depth. In this paper, we develop a node addressing methodology that merges the structure of a hierarchical tree with the flexibility of AODV. We show its completeness and develop three algorithms - deterministic, probabilistic and heuristic, based on our methodology. The performance of the algorithms against AODV is compared. Our simulation is made for two probability distributions of network formation – uniform and geometric.

Keywords-Hierarchical Addressing; 802.15.4; ZigBee;

I. INTRODUCTION & MOTIVATION

Wireless Sensor Networks (WSNs) have received immense attention of the technical community lately [2][3], largely due to the wide application areas and the unique constraints of the technology that provide scope for innovations. A WSN brings sensing, computing and communicational capabilities into a single node [2]. The nodes are generally deployed to collect environmental and other information in harsh and unconditioned areas where unpredictable events can reduce the effective operation of a WSN [1]. The nodes have limited communication range and the data is propagated to a central data repository (designated as a sink) in a multi hop fashion. The nodes compute in a distributed way, the best route for multi hop data dissemination. Thus, important aspects of a WSN are routing and the degree of robustness.

We study the case of a wireless sensor network with sensors being placed at particular points in the region to be monitored. The sensed parameters include temperature, vibration and (harmful) gases. The sensors are placed at pre-determined locations and the sensed information is transported to a centralized data repository located at a sink. The sensors do not participate in the packet forwarding and relays are placed at appropriate locations to transmit the data in a multi-hop fashion to the sink. Such networks are known as 2-tiered [6] and are the most common form of sensor network deployment. The data propagates from the sensors to the sink and vice-versa. There is no requirement for one sensor to communicate to another sensor, or one relay to communicate to another relay. Further, if such communication is desired, it generally accounts for a fraction of the overall data communication from the sensors to the sink and we can afford to have non-optimal routing for such cases. If we plot the topology of the best path (the metric can be hop counts, signal strength, etc) from the sensors to the sink, it would result in a hierarchical structure as shown in figure 1. The proof is analogous to that of the minimum spanning tree.

The AODV routing algorithm [4] builds the route tables by flooding the entire network with RREQ control packets. These control packets are costly since all devices are battery powered and packet transmissions are the biggest source of energy dissipation. In case of node failures, RRER control packets and subsequent RREQ packets are generated to establish the new route. The AODV algorithm was designed for a mesh network where nodes are predominantly mobile. Although the algorithm is scalable and is fault tolerant, it is expensive in terms of control packets and the memory requirement for route tables. The implementation of the AODV algorithm is non-trivial [7] especially on nodes with constrained resources.

The hierarchical network topology has interesting characteristics. Every node (except the root) has a single parent. More importantly, the address of each node can be set in a way that no routing tables are needed to forward packets [8][9]. The routing is done based on the addresses of the nodes. If we have complete knowledge of the topology, we can assign addresses and not have any route tables. However, in practical deployments, the network topology information is not known apriori and the algorithm must be able to support any arbitrary topology. For example, consider a particular topology with 'n' nodes has been formed and we have assigned node addresses to all these 'n' nodes such that the hierarchy is maintained and no route tables are needed for packet propagation. Now, if a new device wishes to join the network, at any point (i.e with any of the 'n' nodes), we must still be able to supply the new node an address without making the hierarchy in the rest of the topology break.

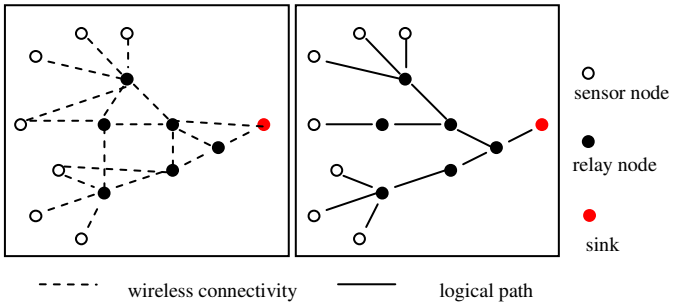

Figure 1. A mesh network reduces to a hierarchical tree when data communication is only between the nodes (sensors and relays) and the sink.

As an illustration, let us consider the C_{skip} hierarchical algorithm used by ZigBee. This algorithm earmarks each node to have a fixed number of children. This fixed allocation restricts the total depth (number of hops) that the network can support. For example, if every relay is expected to support upto 9 relays as its children, the maximum depth is 4. As shown in figure 2(a), a device at depth 5, although in wireless range, cannot join the network due to the restriction of the algorithm. The C_{skip} algorithm also does not support fault tolerance. As an illustration, consider node '2' of figure 2(a) is unable to communicate with node '1' (i.e. a fault has occurred). Node '2' now discovers that node '93' can be used to reach the sink. It thus associates with node '93' as its child. However, this means node '2' is rechristened as '104' (so that hierarchical property is not broken figure 2(b)). Similarly, nodes '3' and '13' of figure 2(a) are now given address of '105' and '106' (figure 2(b)). Note that the depth of nodes '104', '105' and '106' have increased by one when compared to their earlier topology in figure 2(a). Thus, nodes '14', '4', '5', '6' and '7' of figure 2(a), which were part of the network are no longer able to join the network due to a single fault.

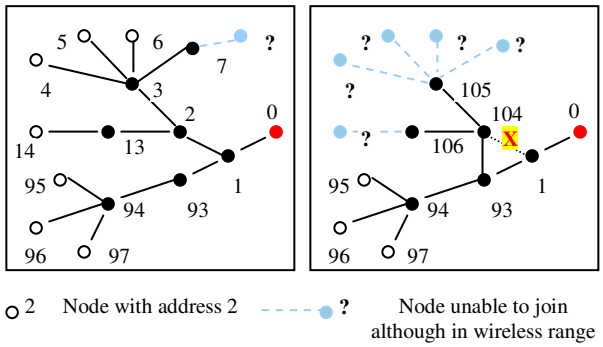

Figure 2(a). The nodes given addresses according to C_{skip} with each relay expected to support upto 9 children. In such a case, depth larger than 4 is not possible. **(b).** Break in link between 2 and 1 (shown as X) results in 3 devices being assigned a new address and 4 sensors no longer part of the network.

In this paper, we define scalability as the ability to support any number of devices within the maximum address space (2^{16} for IEEE 802.15.4) for any topology and in any order of devices joining the network. Fault tolerance is defined as the ability to change the routing topology around the fault without affecting the scalability. We provide in this paper, a methodology to achieve scalability and fault tolerance while following the hierarchy to a large extent. We develop three algorithms using this methodology and show its advantage over AODV in terms of memory requirements. We also provide the complexity analysis and the practicality of the algorithm for implementation.

II. Related Work

Routing protocols can be classified as reactive or proactive. Ad-Hoc On-demand Distance Vector (AODV) and Dynamic Source Routing (DSR) [5] are two well known protocols in the reactive class. Hierarchical routing is an example of proactive protocols and has been well studied in wired networks [12] and preliminary comparisons with AODV have been made recently in IEEE 802.15.4 based wireless networks [7]. The properties and fallouts of AODV and the hierarchical routing in ZigBee have been well explained in Section 1. Here we highlight important concepts and results from the existing literature.

Hierarchical routing is a labeling scheme that renames or labels the vertices of a tree network graph. The vertices of an 'n'- node tree of arbitrary dimension can be labeled using $log_2 n$ bits [12]. The header of the packet at the network layer is simply the label of the destination [12] and the routing decision is taken by comparing the destination address and the current node's address. AODV makes the routing decision by comparing the destination address against a route table. The size of the route table scales with $O(n\log(n))$ [11] and thus grows extremely large as the network grows in size. A related concept is that of compaction, where the route table size is made compact by omitting some details of the network topology. This can lead to a non-optimal route and is measured as stretch. Stretch is the ratio of the path produced by the compacting algorithm and the shortest path [11]. It has been shown that compaction can be achieved keeping the stretch a constant [12][11].

The labeling of vertices for a tree, the build-up of route tables and compaction assume a global knowledge of the network topology. The AODV algorithm develops this global view of the network through a series of control packets which grow exponentially with network size. In this paper, we are concerned with the distributed way of assigning addresses to nodes as they join the network (using IEEE 802.15.4 *MLME-ASSOCIATE.request* and *MLME-ASSOCIATE.response*). We focus on the completeness of our algorithm (i.e. it supports fault tolerance and scalability) and compare with AODV the memory requirement. In our future work, we look to analyze the control packet overhead, performance against compacting and modified algorithms of AODV.

III. Problem Formulation

Consider a network topology as shown in figure 3. The entire topology is divided into two sub-trees, 'A' and 'B' with number of nodes 'k' and 'm' respectively. All nodes of sub-tree 'A' and 'B' have been assigned an address and are completely functional. Now, a new node (shown in grey in figure 3) wishes to join the network. Our task is to assign an address to this new node without breaking the routing in the rest of the network.

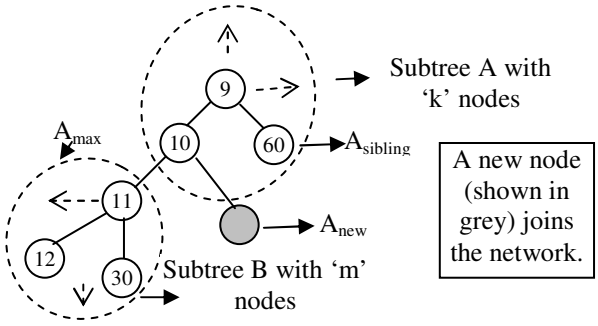

Figure 3: Assignment of an address to a node joining the network.

First, we define the strategy that would be used for routing in the hierarchical network. Let us represent the network in matrix notation (figure 4). The first row represents the parent

addresses and the rest are the children. Note that the order in which the matrix is filled is the order in which the nodes have joined the network. i.e. a node A_{ij} has joined the network before $A_{kj} \forall\, k > i$.

$$\begin{bmatrix} . & 9 & 10 & 11 & . \\ . & 10 & 11 & 12 & . \\ . & 60 & A_{new} & 30 & \\ . & & & & \end{bmatrix}$$

Figure 4: Representation of the network of figure 3.

We define the hierarchical routing strategy as:

$$\{A_{ij} > A_{km} where\ A_{1m} = A_{(i-q)j}\ \forall\, k > 0, m > j, q < i-1\}\ \ \forall\, i,j > 1 \quad (1)$$

and

$$A_{ij} < A_{(k+1)m}\ \forall\, i,j,k > 1\ , m < j\ where\ A_{km} = A_{1j} \quad (2)$$

To put in words, the routing strategy is simple and says: all child nodes have an address greater than it parent but less than the parent's immediate sibling and the sibling of the parent's parent, looping till the root. For example, in figure 3, node 30 > 11, but < 60. Routing would then work as follows: if node 9 receives a packet destined for 30, it checks the addresses of its child. 30 > 10, but 30 < 60. Therefore, node 9 forwards the packet to node 10. Similar forwards brings the packet to its intended destination. No routing table has been used and this is the essence of hierarchical routing.

Now, if a node wishes to join the network as in figure 3, 4 and A_{max} represents the maximum address in sub-tree 'B', then, for routing to be preserved, the following conditions are to be satisfied:

$$A_{new} > A_{max} \quad (3)$$

and

$$A_{new} < A_{sibling} \quad (4)$$

If (3) is not met, a packet sent to A_{max} from the sink will be wrongly routed to a different node. If (4) is not met, then a packet sent to A_{new}will be wrongly routed to a different node. Note that conditions (3) and (4) are already captured in (1) and (2). Now, with routing strategy defined, we develop our algorithms. Let the new node be given an address as:

$$A_{new} = A_{max} + \partial \quad (5)$$

We note that (3) is satisfied for all cases. This implies routing to all nodes in sub-tree 'B' is preserved. When $\partial \geq A_{sibling} - A_{max}$, the routing breaks at particular nodes in sub-tree 'A', but the only for those packets that are destined for the new node. All routing for existing nodes of sub-tree 'A' is preserved. In such a case, where the routing is broken in sub-tree 'A', we include *a route entry only for this new node at all the nodes where routing breaks*. We call this entry as an exception and we note that the route entry is similar to that generated by AODV but also includes the structure of the hierarchical tree. Our aim now, is to choose ∂ in such a fashion that the total number of exceptions is minimized.

Lemma 1: Defining the routing strategy as in (1), (2) and the addresses as in (5), newly assigned addresses cannot invalidate the already setup routing scheme.

Proof: A sample topology is shown in figure 5. Assume an existing node with an address "A". We define all the addresses of the nodes (parents) connecting "A" from the sink as "P_k", where k =1 to D, D= depth of A. Let a new node joining the network get an address "N". Let "Q_k" be the list of addresses of the nodes connecting "N" to the sink with k=1 to D, D = depth of N. Select the node common to P_k and Q_k that has the highest address. (There will be atleast one node common to P_k and Q_k). Let this node be denoted C. Let the node under C with the largest address be denoted as C_n. The routing to A will fail if N joins the network at C and N < A. However, N > C_n & C_n >= A by (5). Thus the routing to A is preserved. ■

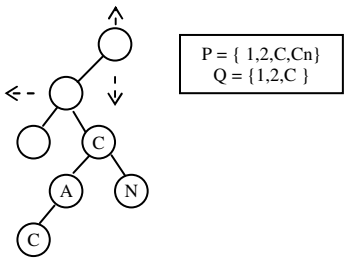

Figure 5: Sample Topology for Lemma I.

For a given topology, i.e. at the time a new node joins, the value of ∂ can be chosen such that the exceptions in sub-tree 'A' are minimized. For a given topology in the representation of figure 4, the minimization problem can then be stated as:

Objective function to be minimized:

$$\frac{1}{C} A_{new} + x_1 + x_2 + x_3 + \cdots + x_p$$

Constraint coefficients:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & -C & 0 & \cdots & 0 \\ 1 & 0 & -C & \cdots & 0 \\ \vdots & & & & \\ 1 & 0 & 0 & \cdots & -C \end{bmatrix}$$

Where the co-efficient C is a large value (e.g. 10^4).

Auxiliary variables:

$$\begin{bmatrix} A_{max} \\ A_{sibling_1} \\ A_{sibling_2} \\ \vdots \\ A_{sibling_p} \end{bmatrix}$$

Where,

$$\{A_{sibling_1}, A_{sibling_2}, \cdots, A_{sibling_p}\} \in \{A_{(k+1)m} : A_{1j} = A_{km} \forall i,j,k > 1, m < j\}$$

With bounds on variables:

$$0 < A_{new} < 2^{16}$$

$$0 \leq x_1 \leq 1$$

$$0 \le x_2 \le 1$$
$$\vdots$$
$$0 \le x_p \le 1$$
$$A_{max} \le A_{max} < 2^{16}$$
$$0 < A_{sibling_1} \le A_{sibling_1}$$
$$0 < A_{sibling_2} \le A_{sibling_2}$$
$$\vdots$$
$$0 < A_{sibling_p} \le A_{sibling_p}$$

With the additional constraint that all structural variables are integers:

$$\{A_{new}, x_1, x_2, x_3, \cdots, x_p\} \in I$$

Based on these constructs, *our first algorithm* is to choose A_{new} such that the number of exceptions in the current network is minimized. This algorithm ignores the possibility that any new devices could join the network and works to choose the best address given a particular topology. We, therefore, call this algorithm as a deterministic one. The algorithm does not earmark an address space for future devices that might join sub-tree 'B'.

Our *second algorithm* is designed with the idea to earmark an address space for future nodes that might join sub-tree 'B'. Thus, this algorithm is future looking and would be ready to generate exceptions in the sub-tree 'A' now, with the hope that future exceptions in sub-tree 'B' are avoided. Our effort is directed to choosing the average case or the expected ∂ in a given topology. We assume that the maximum address in a topology equals the expected number of devices that would join. Then, if 'n' is the number of devices that would join the entire network (sub-tree 'A' and sub-tree 'B'), the expected number of devices that would join under a single node (A_{max}) is given by $E(n)$. Referring to figure 3, if $p(x)$ refers to the probability of having 'x' nodes join under A_{max}, we have:

$$p(0) = \frac{k}{k+1} * \frac{k+1}{k+2} * \frac{k+2}{k+3} * \cdots * \frac{k+n-1}{k+n} = \frac{k}{k+n}$$

$$p(1) = \frac{1}{k+1} * \frac{k}{k+2} * \frac{k+1}{k+3} * \cdots * \frac{k+n-2}{k+n} + \frac{k}{k+1} * \frac{1}{k+2} * \frac{k+1}{k+3} * \cdots * \frac{k+n-2}{k+n}$$
$$\vdots$$
$$+ \frac{k}{k+1} * \frac{k+1}{k+2} * \frac{k+2}{k+3} * \cdots * \frac{1}{k+n} = \frac{n}{k+n-1} * p(0)$$

$$p(2) = \frac{1}{k+1} * \frac{2}{k+2} * \frac{k}{k+3} * \cdots * \frac{k+n-3}{k+n} + \frac{1}{k+1} * \frac{k}{k+2} * \frac{2}{k+3} * \cdots * \frac{k+n-3}{k+n}$$
$$\vdots$$
$$+ \frac{k}{k+1} * \frac{k+1}{k+2} * \frac{1}{k+3} * \cdots * \frac{2}{k+n} = {}^{n}C_2 * \frac{2}{k+n-2} * p(1)$$
$$\vdots$$

$$p(n) = {}^{n}C_n * \frac{n}{k} * p(n-1)$$

$$E(n) = \sum_{i=0}^{n} i * p(i)$$

Thus, we assign new node addresses as:

$$A_{new} = A_{max} + E(n) \quad (7)$$

We denote (7) as the probabilistic algorithm which keeps some address space for nodes that *might* join the network even at the cost of creating an exception now.

Our *third algorithm* is a simple heuristic. We set ∂ to an integer constant throughout the network buildup. There is no theory on choosing the value of ∂, so we start with a value of 5 and simulate the results till 55. We call this algorithm as the heuristic one.

$$A_{new} = A_{max} + 5 \quad (8)$$

IV. RESULTS

We first depict how the proposed scheme solves the issues of large depth and fault tolerance. Considering the example shown in figure 2(a), the new node must be given an address to join the network. However, no address will satisfy the hierarchical property. We therefore give it an address based on any of the three algorithms proposed and assume an address of '23' is assigned. The new address creates route table entry only at node '2'. A packet destined for node '23' is routed as follows. At node '1', the hierarchical strategy (eqn. (1) & (2)) implies the packet must be forwarded to node '2'. Similarly, the hierarchical property at node '2' implies the packet must be forwarded to node '13'. However, the route entry is present at node '2' directing the packet to node 3. The hierarchical property is satisfied at node '7' as well thus reaching node '23' through a single route entry.

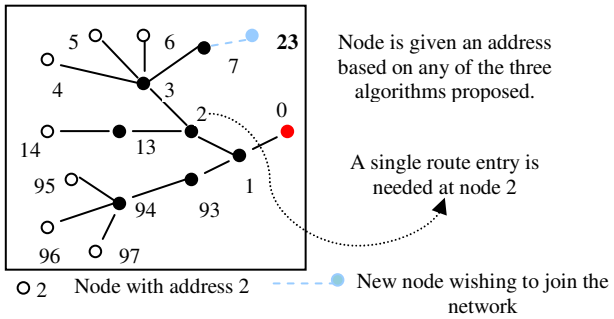

Figure 6. Support for large network depth

Our addressing schemes also have inherent capability to support fault tolerance. In the event of a node failure, a node selects the new parent that would enable it to reach the sink. It presents its address to the new parent. The new parent in turn,

checks for the hierarchical property and creates/updates route table entries if needed and forwards the packet to its parent. This can be considered similar to the AODV style RREQ (route request) and RREP (route reply) messages. An example is shown in figure 7. The link between nodes '2' and '1' breaks and node '2' joins node '93'. Since all of the addresses under node '2' are less than '93', a route table entry is required for each of the 7 devices at node '1'. However, no route table entries are needed at any other node since the hierarchical property is satisfied elsewhere.

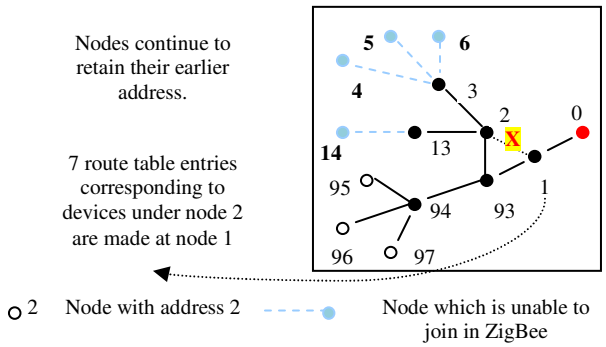

Figure 7. Support for fault tolerance

We compare the above address allocation algorithms with the memory requirement in AODV for storing the route tables. Intuitively, our scheme must show a significant improvement since the address allocation happens in an intelligent way where the hierarchical structure is maintained as much as possible. An additional overhead in our scheme is a single unit to store the maximum address under the device (A_{max}).

Table 1. Simulation Setup.

Simulation Environment	C++
Number or nodes in network	25 to 200, in steps of 25
Location of nodes (Sensitivity Runs)	pseudo random uniform distribution
	pseudo random geometric distribution with p=0.8
Number of runs	100 for each network size

The simulation was carried out in C++ where the node placement is studied under two settings. In one, a uniform probability distribution of node placement was done. Here, a new node would have equal probability to choose any of the previously deployed nodes as its parent. In the second setting, a geometric probability distribution was used. Here a new node has maximum probability of choosing the last deployed node as its parent and the probability of choosing an earlier deployed node falls proportionately. The geometric distribution is a more realistic representation of the practical scenario. Once a random network topology is generated, the memory needed for AODV and the three schemes is calculated for the same topology. We calculate the network average (denoted as AVG, = (sum of memory requirement of all nodes / number of nodes) and the node average for the node with the biggest route table (denoted as MAX, = max (memory requirement of node)).

The results, although largely expected, are quite interesting (figures 8, 9 & 10). The probabilistic algorithm outdoes all other others and on average requires 50% lower memory than AODV. The deterministic algorithm is the worst of the three proposed in this paper. It thus suggests, saving for the future even at the cost of certain non-optimality now, shows better results. Further, this saving for the future when made a function of the network lifetime (in terms of nodes expected to join) works best (Heuristic versus Probabilistic). In other words, during early part of the network life, we make a larger allocation for devices to join than later. Comparing figures 8 and 9, an interesting point is borne out. The average memory needed for probabilistic is higher than the deterministic or the heuristic algorithms while the maximum memory needed is much lower. This can be explained as follows: a node at a lower depth (closer to the root) requires a larger address space. When this space runs out, nodes that join at higher depth (farther from the root), need exceptions at the lower depth node for routing to be preserved. Thus the deterministic and heuristic algorithms are prone to creating more exceptions at the lower depth node with relatively lesser number of exceptions at the higher depth nodes. Thus the average case exceptions are lower. In the probabilistic algorithm, the lower depth nodes have a larger address space and thus a lower maximum memory requirement. It distributes the exceptions among the network and therefore has a higher average count.

Another interesting observation (figures 9 & 10) is the closer results of deterministic and heuristic schemes to the probabilistic for geometric topology distribution. This is explained as follows: our probabilistic scheme makes the calculation of $E(n)$ assuming the network topology to follow a uniform distribution. However, we see that when the distribution is different, the results although better, are not by the same margin as before. This suggests, studying an actual network deployment and estimating the random distribution would lead to better results.

Figures 11 & 12 shows the performance of the heuristic for varying values of ∂. We find the performance improves only initially, suggesting there is a limit to the performance and this limit is a function of the deployment distribution. We cannot readily explain this non-intuitive finding. We hope to theoretically derive this limit in our future works. However, for now, the results indicate that saving blindly for the future has a limit on its effectiveness.

Finally, we study the implementation considerations of the three algorithms. The heuristic algorithm is simplest to implement and has a O(1) complexity. The deterministic and the probabilistic algorithms need O(n) running time. However, all the three algorithms need the entire sub-tree 'A' information and the maximum value of sub-tree 'B' (A_{max}). The topology of 'A' is needed to ensure no duplicate addresses exist. Note that having A_{max} prevents the need to broadcast into sub-tree 'B' searching for duplicates. A comparison of the address duplication is shown in figure 13. AODV performs best as it gets to choose an address out of an address space of 2^{16}. All of the other schemes are significantly prone to address duplication. We also note that the duplication probability for

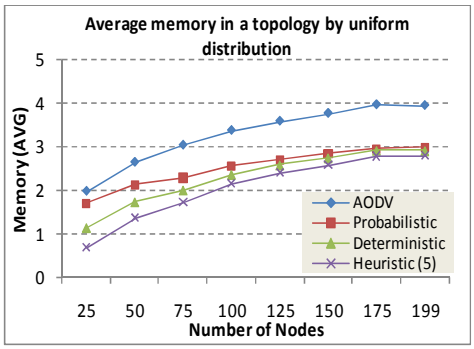

Figure 8

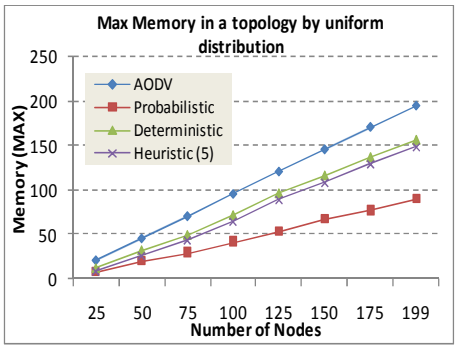

Figure 9

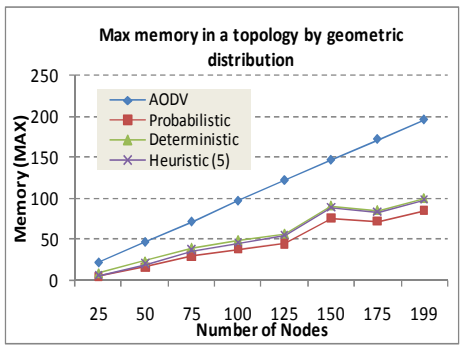

Figure 10

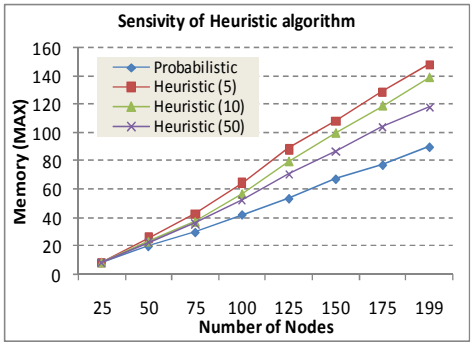

Figure 11

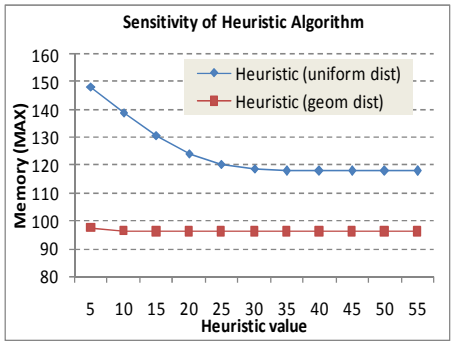

Figure 12

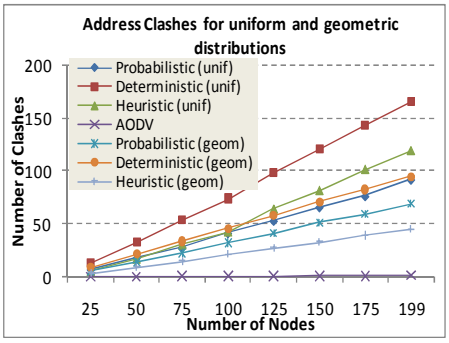

Figure 13

geometric distribution is lower than the uniform distribution for all three algorithms. The control packets needed for all the three algorithms are minimal. When a new node joins, the *join-request* packet is sent to the root, which determines the address (to avoid duplications) and is sent back. Every node along the return path, updates its A_{max} and creates a route entry if needed. Thus, the data needed for routing is piggy-backed on the data packet of the joining of a new node.

V. CONCLUSION

We have shown that a sensor network deployment is hierarchical in nature thus mitigating the use of AODV for routing. We explained problems with the current hierarchical algorithm and developed a methodology that guarantees the routing for existing nodes is preserved and follows the hierarchical principle as much as possible. Based on this, we presented three addressing schemes for scalable and fault tolerant routing. Through simulations we show over 50% improvement over AODV and develop insights into different schemes of saving an address space for the future (none for deterministic, constant for heuristic and diminishing for probabilistic algorithm).

Our future effort would be to make a comparison against modified AODV and compaction algorithms. We also look to develop a simpler probabilistic scheme, reduce the address collision probability and obtain theoretical results for the heuristic algorithm. The routing methodology developed is simple, has shown better performance over AODV and importantly, has low implementation complexity. These attributes give good impetus to adopt the methodology and the algorithms with a practical implementation on commercially available sensor motes.

REFERENCES

[1] A. Wheeler, "Commercial Applications of Wireless Sensor Networks using ZigBee", IEEE Communications Magazine, April 2007.

[2] Akyildiz, I.F., Xudong Wang: A Survey on Wireless Mesh Networks, IEEE Communications Magazine (September 2005)

[3] Bhaskaran Raman and Kameswari Chebrolu, "Censor Networks: A Critique of "Sensor Networks" from a Systems Perspective", ACM SIGCOMM Computer Communications Review, Volume 38, Number 3, July 2008.

[4] C. Perkins, E. Royer, "Ad hoc On-Demand Distance Vector Routing", IEEE Workshop on Mobile Computing Systems and Applications, 1999

[5] Charles E. Perkins, Elizabeth M. Royer, Samir R. Das and Mahesh K. Marina, "Performance Comparison of Two On-Demand Routing Protocols for Ad Hoc Networks", IEEE Personal Communications, February 2001.

[6] E.L. Lloyd, G. Xue, "Relay Node placement in wireless sensor networks", IEEE Transactions on Computers, vol 56, pp. 134-138, 2007.

[7] Francesca Cuomo, Sara Della Luna, Ugo Monaco and Tommaso Melodia, "Routing in ZigBee: benefits from exploiting the IEEE 802.15.4 association tree", ICC 2007.

[8] Gang Ding, Zafer Sahinoglum Philip Orlik & Bharat Bhargava, "Tree-Based Data Broadcast in IEEE 802.15.4 and ZigBee Networks", IEEE Transactions on Mobile Computing, Vol. 5, No. 11, November 2006.

[9] Jamal N. Al-Karaki & Ahmed E. Kamal, "Routing Techniques in Wireless Sensor Networks: A Survey", IEEE Wireless Communications, December 2004.

[10] K. Young-bae and N. Vaidya, "Location-aided Routing (LAR) in Mobile Ad Hoc Networks", ACM MobiCom, 1998.

[11] Mihaela Enachescu, Mei Wang & Ashish Goel, "Reducing Maximum Stretch in Compact Routing", IEEE InfoCom 2008.

[12] Mikkel Thorup and Uri Zwick, "Compact routing schemes", 13th ACM Symposium on Parallel algorithms and architechtures, 2001.

[13] ZigBee Alliance: www.zigbee.org