International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:7, No:4, 2013

Cooperative Energy Efficient Routing for Wireless Sensor Networks in Smart Grid Communications

Ghazi AL-Sukkar, Iyad Jafar, Khalid Darabkh, Raed Al-Zubi, and Mohammed Hawa

***Abstract*—**Smart Grids employ wireless sensor networks for their control and monitoring. Sensors are characterized by limitations in the processing power, energy supply and memory spaces, which require a particular attention on the design of routing and data management algorithms.

Since most routing algorithms for sensor networks, focus on finding energy efficient paths to prolong the lifetime of sensor networks, the power of sensors on efficient paths depletes quickly, and consequently sensor networks become incapable of monitoring events from some parts of their target areas. In consequence, the design of routing protocols should consider not only energy efficiency paths, but also energy efficient algorithms in general.

In this paper we propose an energy efficient routing protocol for wireless sensor networks without the support of any location information system. The reliability and the efficiency of this protocol have been demonstrated by simulation studies where we compare them to the legacy protocols. Our simulation results show that these algorithms scale well with network size and density.

***Keywords*—**Data-centric storage, Dynamic Address Allocation, Sensor networks, Smart Grid Communications.

I. Introduction

WIRELES Sensor networks are recently gaining popularity to realize smart grids for power systems to improve their resilience, efficiency, adaptability, and sustainability[1], [2]. They consist of a large number of tiny devices with sensing capabilities, able to perform simple data processing tasks, and to communicate wirelessly with other similar devices. These networks are deployed in an ad-hoc manner by densely scattering sensors across an area of interest.

For disseminating and storing the sensed data, three methods are investigated in literature[3], [4]: Local storage, External storage and Data-centric storage.

In data-centric storage, events are named, and sensors cooperate locally to detect named events. When a sensor detects a named event, it determines the sensor in the network that is responsible for that name, and then stores the data at that sensor.

Which sensor is responsible for storing a type of data is typically determined by taking a hash of the name, andmapping that hash onto a sensor in the network. When the base station wishes to query the network, it sends the query only to the sensor responsible for the data relevant to the query. In this approach, queries do not need to be flooded through the network, nor does data that the user does not ask about get sent to the base station. Additionally, the query may be partially processed at the sensors storing the data, allowing a small message consisting of aggregated data to be sent to the base station instead of all individual records relevant to the query.

Data-centric storage provides a *(key, value)* based associative memory, in a way similar to the distributed hash table (DHT) systems designed for the internet use, like Pastry [5], CAN [6], Chord [7], and Tapestry [8], where nodes communicate in an application level fashion, through the formation of an overlay network between them.

In data-centric storage, events are named with keys, and both the storage of an event and its retrieval are performed using these keys. Thus the two operations available in data-centric storage based sensor network are:

Put(k,v): which stores the observed data v according to the key k.

Get(k): retrieves whatever stored value associated with key k.

As shown by [4] data-centric storage is preferable in cases where (a) the sensor network size is large, (b) there are many detected events and not all event types are queried. In this work we concentrate on this method, since it seems to be the most efficient way of data dissemination and storage in sensor networks.

In this paper, we present two correlated algorithms for energy efficient routing in wireless sensor networks.They are completely distributed algorithms without any centralized control, which result in all sensors have identical responsibilities.

In the first algorithm a sensor is assigned a unique address according to its relative location in the network, the address assignment mechanism works in a distributed manner where address conflict is avoided without the need to flood the whole network. Sensors change their addresses as they move, thus their addresses have a topological meaning.

The second algorithm is the routing algorithm which is very simple and depends only on the node's first hop neighbors,

G. Al-Sukkar is with the Electrical Engineering Department, The University of Jordan, Amman, 11942 Jordan, (e-mail: ghazi.alsukkar@ju.edu.jo).

I. Jafar is with the Computer Engineering Department, The University of Jordan, Amman, 11942 Jordan, (e-mail: iyad.jafar@ju.edu.jo).

K. Darabkh is with the Computer Engineering Department, The University of Jordan, Amman, 11942 Jordan, (e-mail: k.darabkeh@ju.edu.jo).

R. Al-Zubi is with the Electrical Engineering Department, The University of Jordan, Amman, 11942 Jordan, (e-mail: r.alzub@ju.edu.jo).

M. Hawa is with the Electrical Engineering Department, The University of Jordan, Amman, 11942 Jordan, (e-mail: hawa @ju.edu.jo).

and the forwarding process resembles to some degree the one found in Pastry peer-to-peer protocol [5].

The rest of this paper is organized as follows. In section II we describe the related work. In section III we describe the address allocation procedure.Section IV describes the routing procedure.Data centric storage mechanism is investigated in section V. Topology dynamism and address reassignment areexplained in section VI.Simulation results are introduced in section VII. Finally we conclude with section VIII.

II. Related Work

Ratnasamy et al. [3] were the primaries who introduce the concept of data-centric storage in sensor networks, as well as they describe the simpler concepts of local and external storage. Their work continued in [4], where they describe GHT, a data-centric storage system for sensor networks built on top of GPSR [9]. Their approach depends on the use of geographical information, where they assumed that each node knows its location coordinates using some technologies (e.g. GPS), although this works well, however, location information is not always available. Current methods of determining the location information consume much energy and may not be possible in many sensor network scenarios. Also GPSR works best when geographic locality accurately represents network topology.

Taking this in consideration, a number of new routing protocols where invented, that try to estimate node coordinates in a relative way without the assistant of any positioning system, examples of these protocols NoGeo[10] and GEM [11]. NoGeoproposed an algorithm for performing node to node routing with only neighbor information, without geographic location information. Where a virtual coordinate system is built by having nodes on the perimeter of the network determine their positions relative to each other. They then use an iterative relaxation algorithm for other nodes to determine their coordinates.

GEM constructed a labeled graph that embedded in the original network topology in a distributed fashion. In that graph, each node is given a label that encodes its position in the original network topology.

To do that, they developed two algorithms, with the first one VPCS, they embedded a ringed tree into the network topology, and labeled the sensors in such a manner as to create a virtual polar coordinate space. They have also developed VPCR, a routing algorithm where each node keeps state only about its immediate neighbors, and requires no geographical information.

III. Address Allocation Algorithm

This algorithm enables the sensors to allocate addresses in a local way i.e., without the need to contact faraway nodes in the network or flooding the whole network, at any given time; each node manages a range of addresses including its own address.Node addresses are dynamically assigned depending on the node's current position in the network. More specifically, the addresses are organized as a tree. We call this the *address tree*, see Fig. 1.

To understand this addressing assignment mechanism, let us assume that the addresses are *d*digits with base 10 numbers, so addresses will be in this form $A_{d-1}, \dots, A_0$, where $A_i \in \{0,1,\dots,9\}$. As we will notice, the choice of the base **B** will determine the maximum number of children a node could have, in our example here the maximum number of children is 9.

The base station will be a logical choice to play the role of the first node which will form the network (since it is the most stable node), so it will take the all zeroes address 00. . .0, we call it the root node, as sensor nodes arrive in the neighborhood of the root (i.e., they are in its transmission range), they contact it to obtain an address (call these nodes level 1 nodes). The root node control the first digit (leftmost digit) of the address, where it give the first arriving node address 100…0, the second arriving node 200…0 and so on up to 900…0. These first level nodes control the second digit (from left) in the address, so when nodes connect to any of these nodes and ask for address, they fix the first digit as their address and change the second digit according to node arriving sequence.For example if a node arrive and it is in the neighborhood of the node with address 100…0 and ask this node for an address, then node 100…0 will give it the address 110…0, the second node ask 100…0 for an address will take 120…0 and so on (we call node 100…0 parent of nodes 110…0, 120…0,…,190…0 and thus they are its children).

These second level nodes take control of the third digit and so on. Fig. 1 show an example of an address tree with three digits addresses, for *d = 3*digits, the entire address space can berepresented by *xxx*, where *x*Є {0, 1,…, 9}, nodes in level *l*subtree are the children of the node in level *l-1*. We call the last level nodes in the tree *leaves*. These leaves do not take control of addresses since address space reaches its limit.

Address tree illustrates how addresses are allocated; it does not represent the actual network topology although address of a node depends on its current position in the network. Fig. 2 shows an example of a network topology which uses this algorithm.

When a new node *i* arrives in the network, it receives an address R_i (call it temporary address) which will be used for routing. A new node in the network receives the temporary address from one of its neighbors (we call this neighbor the *parent neighbor* P_i). We assume the existence of some bootstrap mechanism which allows new nodes to identify their neighbors in the network.

This mechanism results in a list containing information about all neighbors. Let $N_i = \{n_1, n_2, \dots, n_q\}$be the set of *q* nodes in the neighborhood of node *i*(in its transmission region). The neighborhood listL_i of node *i* is defined as
$L_i = \{(n_1, R_{n_1}, C_{n_1}), (n_2, R_{n_2}, C_{n_2}), \dots, (n_q, R_{n_q}, C_{n_q})\}$where$C_{n_j} = \{R_{c_1}, R_{c_2}, \dots, R_{c_n}\}$ is the *children list* managed by node n_j, $\forall n_j \in N_i$.

The neighborhood list is used to determine which existing node in the neighborhood will give a temporary address to the arriving node. Several factors must be taken into account.

We apply the following criteria to assign one temporary address to a new node. Using this criterion the joining node selects, among a set of candidate neighbors, the node which will be the parent neighbor of it. This node will be the one with the least level i.e., the nearer to the root. If two or more nodes have the same level then it chooses the node with the least number of children, if a gain two or more nodes satisfy this condition then it will choose the one with the least address.

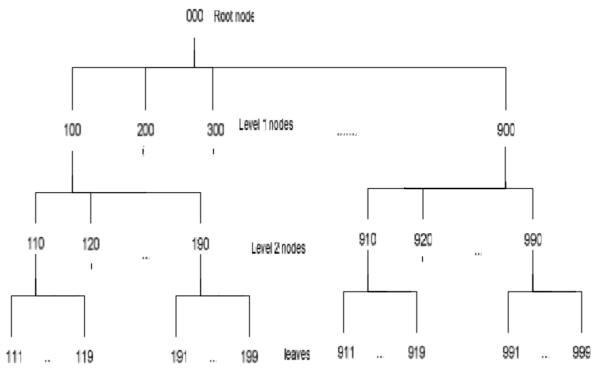

Fig. 1 Address tree with three digits decimal address space

After the new arriving node chooses the parent neighbor it asks that parent for a temporary address which will be assigned according to our address allocation algorithm, we said that an *association relationship* established between the two nodes. In Fig. 2 this association relationship is represented by continuous thick lines, where the dotted thin lines represent the *neighborhood relationship*.

IV. Routing Algorithm

The previously mention address allocation algorithm simplifies the routing procedure as we will see.Where routing is performed in a hop by hop basis.

Having obtained its temporary address, the new node *i* also learns the temporary addresses of its immediate neighbors. This neighborhood information will compose its routing table.

In this algorithm a node routes a message by simply forwarding to the neighbor whose address is the closest to the searched temporary address of the destination until the messages reaches the destination. This forwarding procedure resembles the forwarding procedure in Pastry [5]; where the message is forwarded to a node from the routing tablethat has a temporary address with longer shared prefix with the temporary address of the destination.

If the node cannot find in its routing table such a node that have a longer shared prefix matching, it simply forward the message to its parent and so on until the message reach its destination.

Fig. 2 shows an example of how the routing algorithm works, here node 7 with R_7 = 220 want to send message for the destination 14 with R_{14} = 311. Node 7 find in its routing table that node 16 has a temporary address that matches the destination temporary address in the first digit, so it forwards the message to this neighbor, in its turn node 16 forwards this message to node 15 which is its parent neighbor since it does not have in it routing table any node that has a longer prefix matching with the destination node's temporary address. Node 15 forward the message to node 11 which has a temporary address that matches the destination's temporary address in two digits. Finally, this node forwards the message to node 14 which is the destination node.

Also Fig. 2 illustrates another routing example, where the source is node 7 and the destination is node 4. As you can note from this example, the message forwarded back to the root node 0 which in its tern forward it to the destination.

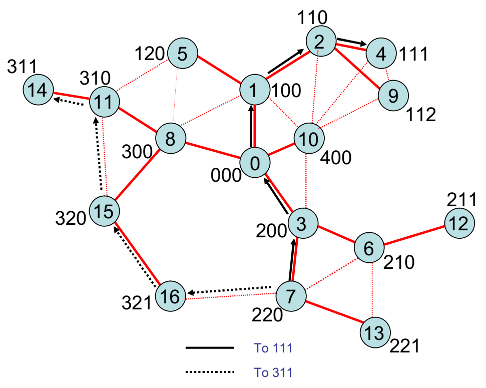

Fig. 2 A network with 17 nodes and three digits address space. Numbers in the circles are nodes identifiers, whereas numbers beside the circles are nodes addresses

The arrival of a new node affects only a limited number of existing nodes (nodes that are in its direct transmission region). The number of neighbors and, consequently, the signaling overhead, depend only on the node's transmission range and are independent of the total number of nodes in the system. Furthermore, a small amount of information suffices to implement this routing algorithm. Each node only stores information about itself and about its neighbors.

V. Data-Centric Storage Mechanism

As we will see, implement these algorithms in sensor networks will simplify applying data-centric storage in these kinds of networks. Here we will explain how this is done.

A. Event Storing Procedure Put (k,v)

This operation is used to identify the node which will be responsible for storing a sensed named event *v*. We will assume the existence of previously known naming system, which maps each defined event to a key *k*.

By using any well-known functions like SHA-1 [12], the sensor *i* which detect the event *v*, hashes the key of the sensed event, and obtains an m-bit number. This number is then translated using certain function into another number which falls in the temporary address space, this number R_r is used to find the sensor which will be responsible of storing the event data *v*, as follows.

Sensor *i* forwards a *registration* message using R_r as a destination address, by applying the routing procedure as in section IV.This request will be forwarded until it reaches the sensor having temporary address that has the longest prefix matching with R_r.

So this sensor is the one responsible for storing the sensed data event *v*.

B. *Event Lookup Procedure Get(k)*

The interested node *s* apply the same globally know hash function on the events key, so it well get a temporary address R_d , this temporary address is the one used to find the sensor which is responsible for storing this event value *v*.

To find this sensor, node *s* forwards a *lookup* message using R_d as a destination address, applying the routing algorithm in section IV, this message will be forwarded until it reaches the sensor with the longest prefix matching with R_d, this sensor is the final destination. So it is our target, which will respond with the value *v* corresponding to the key *k*.

VI. Topology Dynamism And Address Reassignment

Our algorithms have to deal with dynamic topology changes caused by sensors voluntarily join or leave the network due to sensors mobility, or by sensors failures due to energy depletion (though some may fall prey to a hostile environment). When a sensor*i*departure,the system must guarantee the stability of the routing protocol, also the persistency and the consistency requirements of the framework have to be guaranteed.

A. *Persistency*

To ensure the persistency of the system in case of dynamic topology changes, the sensed event data *v* has to be stored in multiple locations. These locations have to be chosen in an efficient way.

We suggest here to store *v* in different branches of the address tree, so in case of a complete distortion in the first level subtree, the data *v* is guaranteed to be available in another subtree.

To do that we have to modify the hash function in such a way to give use multiple numbers which will be used to store the data in multiple locations. One simple way to do that is to modify the left most digit in the number which result from applying the original hash function.

B. *Consistency*

To ensure consistency we have to deal with sensor movement and sensor sudden failure.

- **Dealing with Sensor Movement:**

We consider that before leaving its location, a sensor explicitly hands overits temporary address R_i,its neighborhood set N_i, neighborhood list L_i, its children list C_i, and the associated mapping information database to its parent neighbor[1].

In this situation we have to deal with one of the following two cases:

Case 1: The leaving sensor *i* is a leaf node, Fig. 3, shows an example, where node 9 leaves the network (or changes its position), in this case, the node mobility will cause no impact on the organization of the topology, the only process that will take place is the handover of the mapping information database, to the parent P_i, and the temporary address of the leaving node will be available again for its parent to be assigned to another node.

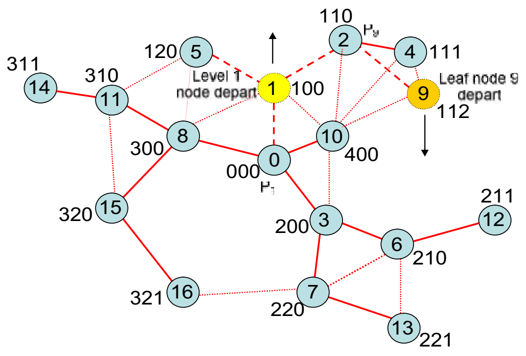

Fig. 3 Leaf node 9 and node 1 leave (or change their position)

Case 2: The leaving node is not a leaf node; it could be any node in any level of the address tree.So the system must guarantee the persistency and the consistency after a node departure. In Fig. 3, node 1 leaves the network, the parent neighbor is node 0, and it leaves behind four descendants; its two children, node 2 and 5, and the children of node 2; which are node 4 and 9.

Based on the received neighborhood list L_i of the depart node *i*, its parent neighbor P_i will face one of the following:

- The children of the leaved node *i* are also neighbors of the parent node P_i of *i* i.e., $C_i \subset N_{P_i}$, in this case the parent neighbor establishes an association relationship with these node, telling them that it now play the role of their previous parent *i* and no other operation will be required. Thus any messaged directed to (through) or from these children will be processed by the parent neighbor of the previously departed node. We call this a smooth reassignment.
- All or some children of the leaved node *i* are note neighbors of the parent node P_i of *i* i.e., $C_i \not\subset N_{P_i}$, in this case, the parent neighbor P_i try to find the set *S* of the children nodes that are also neighbors to itself, i.e., $S = C_i \cap N_{P_i}$, if it is not empty $S \neq \emptyset$, then the parent neighbor establishes an association relationship with these node as in the previous case, so these node will keep their temporary address.

[1]We assume the existence of a mechanism that allows a node to determine when it leaves its location.

For the rest of children $f = C_i - S$ that are note neighbors of P$_i$, after detecting the absence of their parent neighbor *i*, they will try to rejoin the network as they are a new arriving nodes, so they will ask another node with a higher level for a new temporary address and establish an association relationship with it, and inform all of its children nodes about the address change, so these nodes will change their addresses according to the new address of their parent, these children will do the same process, this process will repeated recursively until the change include all the nodes in their subtree.

If a node from set *f* could not find a node with higher level to establish an association relationship with it, this node will try to establish an association relationship with nodes in the same level as its current one or even with lower level.

This node cannot be one from the subtree of the leaved node, in case that they still hold the previous temporary address.

At the same time this node will send a message for all of its children telling them to rejoin the network, in this case each one of these children will try in its turn to apply the same previous mechanism which will recursively repeated. See Fig. 4.

Taking this case in consideration, each time a new node arrives in a location that has been previously occupied by another node, the parent neighbor verifies if the new node is appropriate to receive the previous handed over temporary address and the associated mapping information database.

For doing this, the parent neighbor compares the neighborhood set sent by the previous mobile node, before it's moving, with the one for the new arrived node. If the new node is also a neighbor of the children of the previously leaved node i.e. if $\subset N_{NEW}$, the parent neighbor assigns the temporary address and the mapping information database of the previously leaved node to the new node. However, if the new node cannot satisfy this condition a new temporary address will be attributed to it, according to the described joining procedure.

- **Dealing with Sensor Sudden Failure:**

The mechanism to deal with sensor sudden failure is same as the one used in sensor movement. The difference here is that in case of sensor sudden failure, its neighborhood set N$_i$, neighborhood list L$_i$, and its children list C$_i$ of the failed sensor *i* are not available for its parent neighbor. In this case its parent neighbor and its children will depend on the received hello messages and the routing table, to decide if they are still neighbors, so they could apply the same mechanism in the case of sensor movement. This will result just in a higher handover time than the case of sensor movement, since discovering sensor sudden failure, and the dependency on the hello messages, will take longer time.

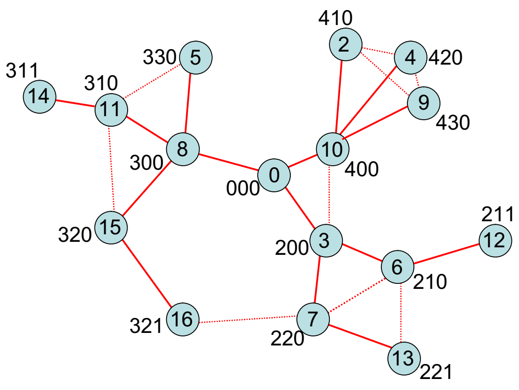

Fig. 4 This figure shows the network after node 1 leaved the network

VII. Simulation Results

In the following, we present the simulation results in which we compare the performance of our proposed protocol (we named it SenPARTY) with that of the well-known reactive routing protocol, ADOV [13], after applying a simplified version of a P2P service over it. We used AODV to compare with, because of the relatively low routing overhead in reaction to low traffic arrival rate, and due to the stability of NS code that implement of this protocol, Further details about other comparisons with other routing protocols is available in [15]. Simulations were performed using the Network Simulator NS2 [14]. Performance comparisons are studied in terms of overhead, throughput and energy consumption.

For all simulations, we used the standard values of NS2. The duration of all the NS2 simulations was set to 300 seconds; the first 20 seconds were free of data traffic, allowing the initial address allocation to take place and for the network to organize itself. In order to study the effect of network density on the overall performance, the size of the simulation area was chosen to keep average sensor degree at two values; the first one was 10 while the second one was set to 30. We used UDP/CBR flows with 5 seconds interval. In our simulation, the size of the address was set to 10 decimal digits. Table I shows most of the important simulation parameters used in all simulation scenarios.

For the mobility we used the random waypoint model with an average speed of 5 m/sec and 20 seconds as an average pause time. We used three mobility scenarios, in the first one we used a static topology where all sensors were not moving during the whole simulation run, in the second scenario 10% of the sensors were selected randomly to move, while in the last scenario 25% of the sensors were moving.

TABLE I
Simulation Parameters

Parameters	Setting
Hello message rate	1s
SenPARTY Signalling packet size	48 bytes
CBR packet size	625 bytes
CBR packet interval	5 s
Address request waiting time	200 ms
Senor velocity	5 m/s

A. Increasing Network Size with Various Densities

The first set of experiments compares the performance of SenPARTY and AODV protocol with increasing network size while keeping the traffic load constant at 100 CBR flows. The source and destination of each CBR flow were selected randomly, also in the mobility cases the mobile sensors were selected randomly. In order to study the scalability of these protocols with high number of sensors, we varied the number of sensors from 50 to 500. Furthermore, to study the effect of node density on the scalability of these protocols, we chose two different densities.

Fig. 5 and Fig. 6 show that SenPARTY achieves high delivery ratios almost above 70% in all mobility scenarios irrespective of node density. With increasing number of sensors, delivery ratios do not decrease much in case of SenPARTY, while for AODV with high density (i.e. 30) they decrease dramatically, this happen because in high density, AODV queue packets while it repair routes that fail due to congestion which is a consequence of high density. Additionally, packets spend more time in the interface queues because of collision avoidance and packet retransmissions at the MAC layer. Accordingly, the delivery ratio decreases because packets are dropped when interface queues fill up.

Increasing the network size aggravates the problem because the message overhead to repair routes grows and longer routes are more likely to fail. The large increase in per node overhead per second for AODV in Fig. 7 and Fig. 8 illustrates this problem.

AODV incurs a high overhead to repair a failed route because it uses flooding. SenPARTY has a low message overhead which is independent of the size of the network, because it uses a local unicast messages to repair a failed route.

For both protocols, when mobility increases, there are more route failures because nodes move, this result in more message overhead and lower delivery ratio. In all scenarios, SenPARTY achieves low overhead for all network sizes with good delivery ratios. It can do this for the reasons mentioned previously and because it can repair routes with lower overhead than the other protocols.

Fig. 9 and Fig. 10 show that the average consumed energy in SenPARTY is always lower than that of AODV. In high density, energy consumption increases for AODV with increasing network size, again, this could be explained by the increase in number of collisions due to flooding mechanism used in path establishment and repair. Therefore, as the mobility increase, the energy consumption will increase.

For AODV in low density and SenPARTY in both densities, the average consumed energy decreases slowly with increasing network size, this is due to the fact that we kept the overall traffic load constant as we increase the network size, accordingly the traffic overhead will distributed among more sensor nodes, besides, as illustrated in Fig. 5 and Fig. 6, the number of CBR packets delivered correctly decreases as network size increase.

B. Increasing Traffic Load with Various Densities

The second set of experiments evaluates the performance of these two protocols with increasing traffic load while keeping the size of the network constant at 200 nodes. We varied the total number of CBR flows between 10 and 250.

As illustrated in Fig. 11 and Fig. 12, SenPARTY achieves higher delivery ratio than AODV in all scenarios. As the number of flows increases, the delivery ratio drops. Also with mobility increase the delivery ratio decrease, this islogically, since with an elevation in the dynamism of the network more route failures will occur.

The variation in network density almost does not affect delivery ratio in SenPARTY while in AODV, high density results in more decay in the delivery ratio as shown in Fig 11.

Fig. 13 and Fig. 14 show a higher overhead for AODV than that for SenPARTY at more than 50 flows, this overhead increases dramatically with increase in traffic load, while it is almost stay at a constant value in SenPARTY, this value represent the average number of neighbors for each sensor. As mentioned before this is due to the fact that SenPARTY uses uncomplicated routing mechanism which depends on neighborhood forwarding, and exchanges a local unicast messages to repair a failed route. Meanwhile AODV depends on flooding the whole network, so with high density more collision will happen at the MAC layer, which causes more and more route failure and hence more overhead.

Obviously energy consumption in SenPARTY is lower than that in AODV as we can notice from Fig. 15 and Fig. 16. This consumption in energy increases with the increase in traffic load, node density, and mobility.

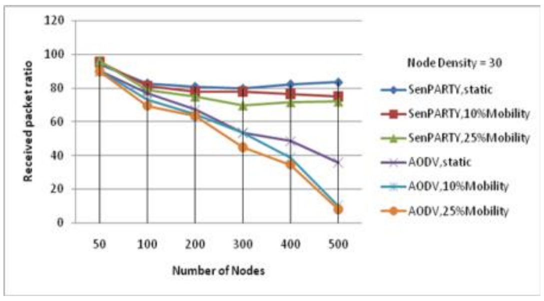

Fig. 5 Throughput vs. Network Size: 100 UDP/CBR flows, Density = 30

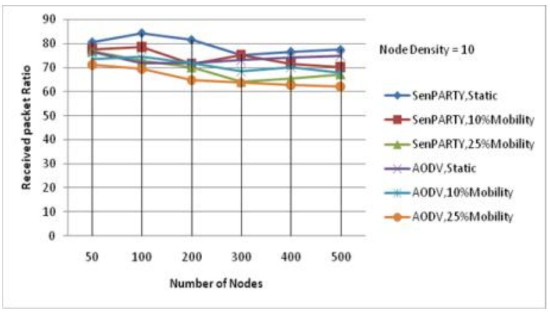

Fig. 6 Throughput vs. Network Size: 100 UDP/CBR flows, Density = 10

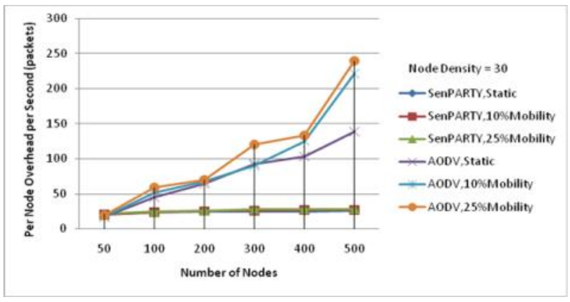

Fig. 7 Overhead vs. Network Size: 100 UDP/CBR flows, Density = 30

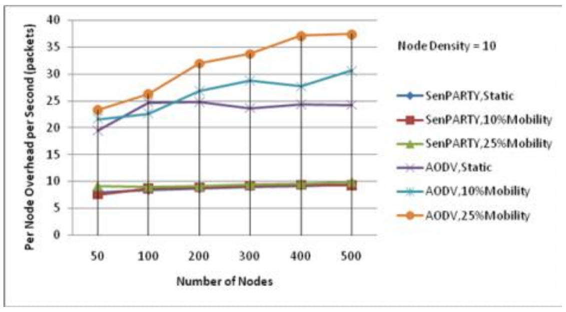

Fig. 8 Overhead vs. Network Size: 100 UDP/CBR flows, Density = 10

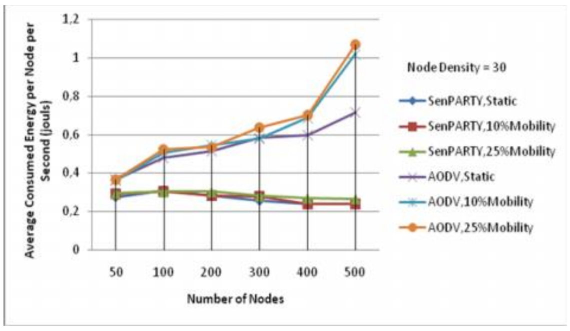

Fig. 9 Energy consumption vs. Network size, 100 UDB/CBR flows, Density = 30

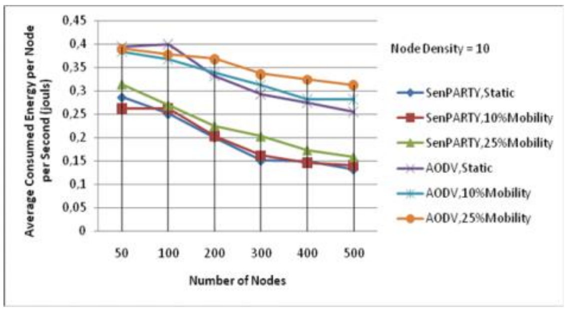

Fig. 10 Energy consumption vs. Network size, 100 UDB/CBR flows, Density = 10

C. Increasing Traffic Load with Various Densities

The second set of experiments evaluates the performance of these two protocols with increasing traffic load while keeping the size of the network constant at 200 nodes. We varied the total number of CBR flows between 10 and 250.

As illustrated in Fig. 11 and Fig. 12, SenPARTY achieves higher delivery ratio than AODV in all scenarios. As the number of flows increases, the delivery ratio drops. Also with mobility increase the delivery ratio decrease, this islogically, since with an elevation in the dynamism of the network more route failures will occur.

The variation in network density almost does not affect delivery ratio in SenPARTY while in AODV, high density results in more decay in the delivery ratio as shown in Fig 11.

Fig. 13 and Fig. 14 show a higher overhead for AODV than that for SenPARTY at more than 50 flows, this overhead increases dramatically with increase in traffic load, while it is almost stay at a constant value in SenPARTY, this value represent the average number of neighbors for each sensor. As mentioned before this is due to the fact that SenPARTY uses uncomplicated routing mechanism which depends on neighborhood forwarding, and exchanges a local unicast messages to repair a failed route. Meanwhile AODV depends on flooding the whole network, so with high density more collision will happen at the MAC layer, which causes more and more route failure and hence more overhead.

Obviously energy consumption in SenPARTY is lower than that in AODV as we can notice from Fig. 15 and Fig. 16. This consumption in energy increases with the increase in traffic load, node density, and mobility.

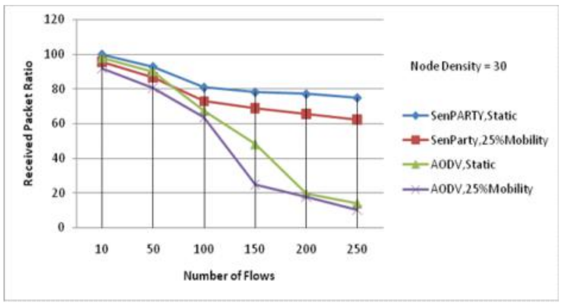

Fig. 11 Throughput vs. Flow count: 200 Sensor, Density = 30

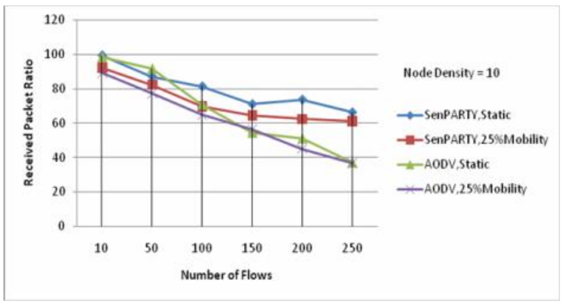

Fig. 12 Throughput vs. Flow count: 200 Sensor, Density = 10

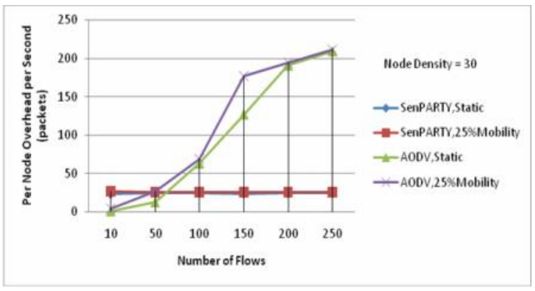

Fig. 13 Overhead vs. Flow count: 200 Sensor, Density = 30

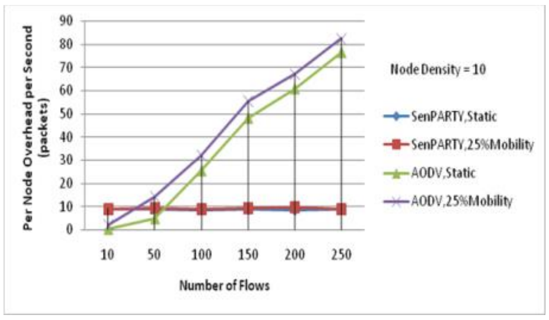

Fig. 14 Overhead vs. Flow count: 200 Sensor, Density = 10

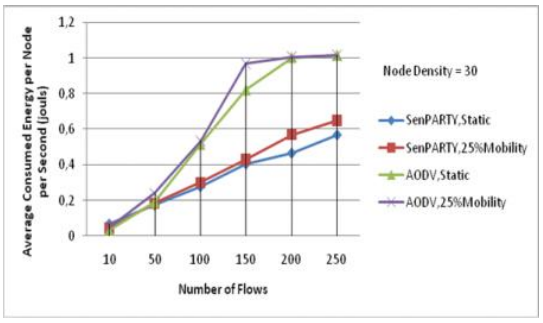

Fig. 15 Energy consumption vs. Flow count, 200 Sensors, Density = 30

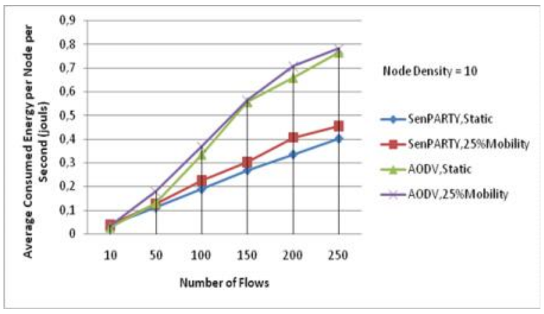

Fig. 16 Energy consumption vs. Flow count, 200 Sensors, Density = 10

VIII. Conclusion

Two algorithms were proposed for energy efficient routing in wireless sensor networks for Smart Grid communications, without the support of any location information system. Neighborhood information exchange are sufficient to implement the routing mechanism and maintain the routing path, consequently, low signaling overhead is generated, only local neighborhood communication.

We expect these algorithms to be applied in environments with large number of sensors where the scalability of the network has great issue such as Smart Grids.

We have presented simulation results where we compared our protocol with a P2P supported AODV protocol, since AODV is one of the most optimized protocols, and its overhead is relatively small for almost low data arrival rate. The results demonstrate that our protocol provides robust performance across a range of different environments and workloads.

References

[1] V.C. Gungor, “Multimedia Wireless Sensor Networks for Smart Grid Applications,” in *IEEE COMSOC MMTC E-Letter*, vol. 6, no. 12, pp.9-11, 2011.

[2] Emilio Ancillotti, Raffaele Bruno, and Marco Conti, “The Role of the RPL Routing Protocol for Smart Grid Communications,” *IEEE Communications Magazine*, Vol.51, no.1, January 2013, pp. 75–83.

[3] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. “Data-centric storage in sensornets,” *Proc. ACM SIGCOMM Workshop on Hot Topics In Networks,* 2002.

[4] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. “GHT: a geographic hash table for data-centric storage,” *Proceedings of the ACM Workshop on Sensor Networks and Applications*, pp. 78--87, Atlanta, Georgia, USA:ACM, September 2002.

[5] Rowstron and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” *in Proceedings of the Middleware*, 2001.

[6] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker. “A Scalable Content-Addressable Network,” In *Proceedings of the ACM SIGCOMM*, 2001.

[7] Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. “Chord: A Scalable Peer-topeer Lookup Service for Internet Applications,” *ACM SIGCOMM 2001*, San Diego, CA, August 2001.

[8] Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, “Tapestry: A resilient global-scale overlay for service deployment,” *IEEE Journal on Selected Areas in communications*, vol. 22, no. 1, pp. 41–53, January 2004.

[9] Karp and H. T. Kung. “GPSR: greedy perimeter stateless routing for wireless networks,” *Proceedings of the 6th annual international conference on Mobile computing and networking*, Boston, Massachusetts, United States, 2000, pages 243–254.

[10] Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, “Geographic routing without location information,” *in ACM MobiCom*, 2003.

[11] James Newsome and Dawn Song, “Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information,” in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, New York, NY, USA, 2003, pp. 76–88, ACM Press.

[12] “FIPS 180-1, Secure Hash Standard.” *U.S. Department of commerce/NIST, National Technical Information Service*, Springfield, Apr. 1995.

[13] C. Perkins and E. Royer,“Ad hoc on demand Distance Vector routing,” *in proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*: (WMCA’99), New Orleans, Louisiana, USA, February 1999.

[14] Network Simulator, http://www.isi.edu/nsnam/ns/.

[15] N. Sarkar, and W. Lol, “A study of MANET Routing Protocols: Joint Node Density, Packet Length and Mobility”, *in Proceedings of ISCC '10 The IEEE symposium on Computers and Communications*, Reccione, Italy, June 2010.