A Fast Distributed and Efficient Virtual Backbone Election in Large Scale MANETs

Wasim El-Hajj
College of Information Technology
UAE University
United Arab Emirates
Email: welhajj@uaeu.ac.ae

Mohsen Guizani
College of Information Technology
UAE University
United Arab Emirates
Email: mguizani@uaeu.ac.ae

Abstract— **Virtual backbone based routing is a promising approach for enhancing the routing efficiency in wireless ad hoc networks. To establish communication in the network, the virtual backbone nodes have to be connected. Connected dominating sets (CDS) are the earliest structures proposed as candidates for virtual backbones in ad hoc networks. In this paper, we propose a fast distributed and efficient algorithm to find a connected dominating set (DE-CDS) in wireless ad hoc networks. DE-CDS has a message and time complexity of $O(n)$ and $O(\Delta^2)$, where n is the number of nodes in the network and Δ is the maximum node degree. According to our knowledge, DE-CDS achieves the best message and time complexity combinations among the previously suggested approaches. Moreover, DE-CDS constructs a reliable virtual backbone that takes into account (1) node's limited energy, (2) node's mobility, and (3) node's traffic pattern.**

I. Introduction

Mobile wireless ad hoc networks appear in a wide variety of applications, including military battle field, disaster relief, surveillance, sensing and monitoring. An Ad-Hoc network is a collection of autonomous arbitrarily located wireless nodes, in which an infrastructure is absent. Two nodes can communicate directly with each other if they are within each others' range; otherwise, intermediate nodes have to relay messages for them.

It has been proven in [1], [2] that a flat network has poor scalability. In fact, the authors showed that the node throughput declines rapidly to zero as the number of nodes in the network increases. To overcome this problem, the network is designed in a hierarchical fashion, where some nodes are elected as leaders and other nodes connect to the leaders forming the clusters. We adapt such a technique and we call the leaders cluster heads. The cluster heads form the backbone of the network (also called virtual backbone). In order to establish communication between the various cluster heads, the backbone nodes have to be connected. The network can be modeled as a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges.

Connected dominating sets (CDS) are the earliest structures proposed as candidates for virtual backbones in ad hoc networks [3], [4], [5]. A dominating set (DS) is a set D of vertices of G such that every vertex of G is either in D or adjacent to a vertex in D. A CDS is a DS, where the elements of D are connected. A minimum connected dominating set (MCDS) is a CDS, where $|D|$ is minimum. In the context of ad hoc networks, a well studied problem is that of finding a MCDS in a Unit Disk Graph (UDG), a class of graphs used to model connectivity in ad hoc networks. Unfortunately, finding the CDS and the MCDS were proven to be NP-hard problems [6], [7], [8].

In this paper, we propose a fast distributed and efficient algorithm to find a connected dominating set (DE-CDS) in wireless ad hoc networks. DE-CDS has a message complexity of $O(n)$ and a time complexity of $O(\Delta^2)$, where n is the number of nodes in the network and Δ is the maximum node degree. According to our knowledge, such complexities are the best achieved among the previously proposed schemes. We approach the problem based on: fast convergence, energy efficiency, and reliability. The remainder of this paper is organized as follows. Section II includes some survey on the backbone construction. Section III describes the cluster head election scheme (virtual backbone construction). Section IV forms the clusters based on the elected cluster heads. Section V presents our contributions and section VI concludes the paper and discusses future work.

II. Related Work

Researchers have proposed several distributed algorithms to find a CDS for arbitrary undirected graphs and UDGs. These algorithms differ in their running time and message complexity. In the following section, we briefly survey some of the schemes for virtual backbones in wireless ad hoc networks. The message and time complexity of our approach ($O(n)$ and $O(\Delta^2)$ respectively) provide a substantial improvement over the suggested approaches.

Das et al. [3], [4] proposed an approach that contains three stages: approximating the minimum dominating set, constructing a spanning forest of stars, and expanding the spanning forest to a spanning tree. The major drawback of this approach is that it has a high message complexity of $O(n^2)$ and a high time complexity of $O(n^2)$.

Wu et al. [9] proposed a simple distributed algorithm that marks a node as a gateway if two of its neighbors are not directly connected. To route traffic from a source to a destination, the source sends the traffic to its gateway, the gateway routes the traffic to the destination gateway, and then from the destination gateway to the destination node. The proposed algorithm has $\Theta(m)$ message complexity and $O(\Delta^3)$ time complexity, where m is the number of edges in UDGs and Δ is the maximum node degree. Although such complexities are promising, this approach may produce poor results where the output CDS consists of all nodes in the network. Also, the CDS might be composed of nodes that have very low residual energy.

Stojmenovic et al. [10] proposed a scheme that is very similar to Wu's scheme except that it requires neighborhood topology which may be achieved by GPS or other location technique. Stojmenovic's approach has a very high message complexity of $O(n^2)$ and a time complexity of $\Omega(n)$. Both approaches do not consider message losses due to collisions in their model.

Cardei et al. [11] proposed a scheme that first finds a maximal independent set and then connects all vertices in the set using a steiner tree. The second step is based on the distributed depth-first search spanning tree algorithm. Cardei's scheme has message complexity of $O(n\Delta)$ and time complexity of $O(n)$. It also requires leader election before the algorithm starts which is not a favorable approach in ad hoc networks.

Parthasarathy et al. [12] proposed two algorithms for finding the virtual backbone. The first algorithm has message and time complexity of $O(n \log^2 n)$ and $O(\Delta \log^2 n)$ respectively. The second algorithm has message and time complexity of $O(n \log n)$ and $O(\log^2 n)$ respectively. The authors assume that each node knows (approximately) the number of its neighbors, the maximum degree, and the size of the network. There is really no fast way to know the number of nodes in the network or the maximum degree. Acquiring such information requires some kind of flooding which increases the complexity of the algorithm.

Several other distributed algorithms for finding the MCDS in UDG exist in [13], [14], [15], [16]. The message and time complexity of these algorithms are $O(n \log n)$ and $O(n)$ respectively. In [17], Alzoubi reduced the message complexity to $O(n)$. The following table summarizes some of the approaches discussed above.

measure	[9]	[10]	[11]	[12]	[17]
msg complexity	$\Theta(m)$	$O(n^2)$	$O(n\Delta)$	$O(n \log n)$	$O(n)$
time complexity	$O(\Delta^3)$	$\Omega(n)$	$O(n)$	$O(\log^2 n)$	$O(n)$

Our approach (DE-CDS) provides better message complexity ($O(n)$) and time complexity ($O(\Delta^2)$) combination than the approaches discussed above. Moreover, DE-CDS provides a reliable backbone suitable for mobile ad hoc network applications. DE-CDS takes into account: (1) the energy restriction imposed on the wireless nodes, (2) the mobility of the wireless nodes, and (3) the traffic pattern of the wireless node. In the next section, we describe our approach in details and we analyze the message and time complexity of each step.

III. Distributed and Efficient Connected Dominating Set (DE-CDS)

We assume that each node knows its own ID, residual energy (RE), mobility (M), and traffic load (T). DE-CDS is divided into four steps. The first step performs a simple neighbor discovery protocol and assigns a weight for each node. The second step elects an initial set of cluster heads. The third step connects the cluster heads together (those elected in the second step) forming a connected dominating set. The last step eliminates some redundant cluster heads. In our approach, we consider that message collisions are handled by the MAC layer. In the following subsections, each step is described and analyzed.

A. Step 1: Neighbor discovery and weight generation

Before DE-CDS is executed, each node needs to know its 1-hop information. To acquire the 1-hop information, a simple neighbor discovery protocol is performed by each node. Each node sends a message containing its ID, RE, mobility, and traffic (send $nodeInfo\{ID, RE, M, T\}$). Every node that receives the $nodeInfo$ message extracts the data and stores it in a special data structure (Vector).

After collecting the $nodeInfo$ messages, each node knows the ID, RE, M, and T of each of its 1-hop neighbors. Let RE_i, M_i, and T_i be the residual energy, the mobility, and the traffic load of node i respectively. Let RE_{max} be the maximum residual energy among the neighbors of i including i. Similarly, let M_{max} and T_{max} be the maximum mobility and the maximum traffic load among node i and its 1-hop neighbors. Node i normalizes its own values with respect to the maximum values i.e.

$$REn_i = \frac{RE_i}{RE_{max}}, \quad Mn_i = \frac{M_i}{M_{max}}, \quad Tn_i = \frac{T_i}{T_{max}}$$

In [18], we designed a fuzzy logic controller which is used to calculate the node's quality. REn_i, Mn_i, and Tn_i are fed to this controller and a single value (W_i) is returned. W_i represents the quality of node i. Note that the fuzzy logic controller combines the residual energy, mobility, and traffic according to certain rules keeping in mind the synergy between them. The fuzzy logic controller tends to give a high weight for nodes that have: (1) high residual energy, (2) low mobility, and (3) low traffic. When W_i is generated, node i sends W_i (send $nodeWeight\{i, W\}$) to its 1-hop neighbors. A neighbor receiving the message, records the weight of node i. After the completion of this phase, each node knows the ID's and weights of its neighbors.

Note that a node with a high weight is a cluster head candidate.

Step 1 requires each node to send 2 messages ($O(1)$ message complexity). The first message is used to send the node's initial information (ID, RE, M, T) and the second message is used to send the node weight (W). Let Δ be the maximum node degree (maximum number of neighbors). The time complexity of Step 1 is $O(\Delta)$ because each node searches its neighbors to find RE_{max}, M_{max} and T_{max}.

B. Step 2: Initial Cluster Head Election

1) Algorithm: This step presents one of our important contributions. In this step, nodes cooperate with each other in order to elect cluster heads. The cooperation is established when a node asks another node to become a cluster head. The node receiving the request should agree on becoming a cluster head. Algorithm 1 presents the pseudo code of this step. A node checks if its weight (W) is the maximum among its neighbors. If the node has the maximum weight, it sets itself as a cluster head. If it does not have the maximum weight, it asks the neighbor having the maximum weight to become a cluster head. Each node in the network executes algorithm 1.

Algorithm 1: Initial CH Election

```
  Data: $\Gamma$ = list of my neighbors
  Result: An election of one cluster head
1 begin
2     if myWeight is the maximum weight among the nodes in $\Gamma$ then
3         set myself as a CH
4         send a message to my neighbors informing them of my decision
5     else
6         Let $j$ be the neighbor that has the maximum weight
7         send a message to node $j$ asking it to become a CH
8 end
```

Figure 1 shows the elected cluster heads (nodes enclosed in grey rectangles). Node 4 has $W_4 = 57$ which is the maximum weight among its neighbors. So, node 4 sets itself as a cluster head. Same for nodes 10 and 15. Node 5 has $W_5 = 50$ and it does not have the maximum weight among its neighbors. But it was elected as a cluster head because node 11 sent a message to node 5 asking it to become a cluster head.

2) Analysis: This step requires each node to send one message ($O(1)$ message complexity). If the node has the maximum weight among it neighbors, it sends a message informing them that it declared itself as a cluster head. If the node does not have the maximum weight among its neighbors, it requests from the neighboring node having the maximum weight to become a cluster head. The time complexity of this step is $O(\Delta)$ because a node needs to search its 1-hop neighbors looking for the node having the maximum weight. If the 1-hop neighbors are sorted according to their weight, the time complexity of this step becomes $O(\Delta \log(\Delta))$.

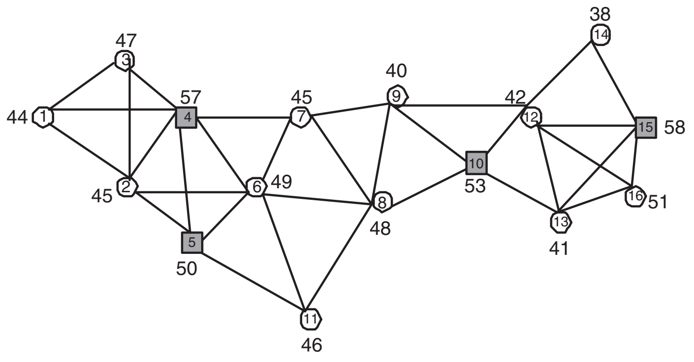

Fig. 1. Initial CH Election. The numbers next to the nodes represent the nodes' weights. Nodes 4, 5, 10, and 15 are elected as cluster heads.

Figure 1 shows that the elected cluster heads does not form a connected backbone. Let $d(u, v) = k$ represent the number of hops between nodes u and v. For example, cluster head 4 is 3-hops away from cluster head 10 i.e. $d(4, 10) = 3$.

Theorem 1. *$\forall$ normal node u, $\exists$ cluster head v such that $d(u, v) = 1$.*

Proof: The proof is extracted directly from the algorithm. Lines 6 and 7 in algorithm 1 indicate that if a node is not a cluster head, it asks one of its neighbors to become a cluster head. Therefore, every node in the network is either a cluster head or a neighbor of a cluster head.

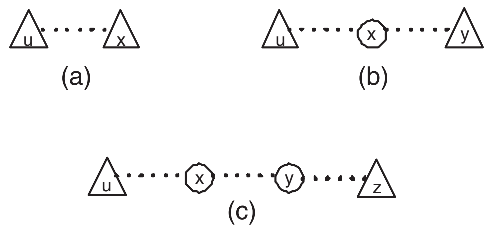

Fig. 2. For a given cluster head u, u can reach another cluster head in: (a) One hop: $d(u, x) = 1$, (b) two hops: $d(u, y) = 2$, (c) three hops: $d(u, z) = 3$

Theorem 2. *$\forall$ cluster head u, $\exists$ cluster head v such that $d(u, v) = 3$ or less, and the intermediate hops are normal nodes.*

Proof: Assume that u is a cluster head. Let $C(u) = \{x | x$ is a cluster head neighbor of $u\}$. $\forall$ $x \in C(u)$, $d(u, x) = 1$ (figure 2(a)). If $x \notin C(u)$, then x is a normal node (figure 2(b)). Let $C(x) = \{y | y$ is a cluster head neighbor of $x\}$. $\forall$ $y \in C(x)$ and $y \notin C(u)$, $d(u, y) = 2$. If $y \notin C(x)$, then y

is a normal node (figure 2(c)). But according to algorithm 1, y must have at least one cluster head neighbor. Let such a cluster head be z, then $d(u, z) = 3$. Therefore, any cluster head can reach another cluster head in at most three hops.

C. Step 3: Connect Cluster Heads

1) Algorithm: According to theorem 2, each cluster head is 1-hop, 2-hops, or 3-hops away from another cluster head. In figure 1, cluster head 15 is 2-hops away from cluster head 10. Nodes 12 and 13 can potentially connect both cluster heads. Cluster head 4 is 3-hops away from cluster head 10. In order to connect cluster heads 4 and 10, 2 normal nodes have be elected as cluster heads. In the following section, we present the algorithm that elects new cluster heads leading to a connected backbone.

The decision of electing new cluster heads is made by the cluster heads that were elected in step 2. Referring to figure 1, only cluster heads 4, 5, 10, and 15 execute the algorithm. The algorithm requires each cluster head to know its 2-hop neighbors. Each node sends its 1-hop information to its neighbors. A node receiving the 1-hop information, stores the data in its data structure. Figure 3 shows the data structure that node 1 uses. All other nodes use similar data structure.

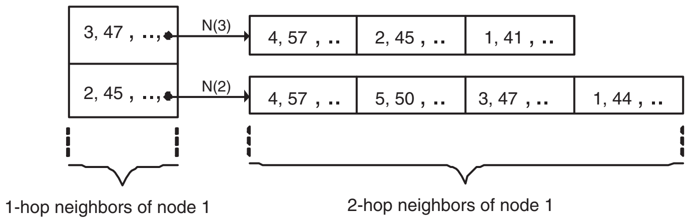

Fig. 3. Node 1 data structure. N(3) and N(2) represent the neighbors of nodes 3 and 2 respectively. The nodes are sorted according to their weights. Such a data structure enables node 1 to know its 1-hop and 2-hop neighbors.

Algorithm 2: Connecting the backbone

```
Data: 1-hop and 2-hop neighbors of cluster head $v$.
      Notation: $\Gamma_x$ = 1-hop neighbors of node $x$
Result: Cluster heads elected by $v$
1  begin
2      for each node $i$ in $\Gamma_v$ do
3          if $i$ is a cluster head then
4              $CHR_i$ = true
5              for each node $j$ in $\Gamma_i$ do
6                  if $j$ is a cluster head then
7                      $CHR_j = true$
8                  else
9                      $NR_j = true$
10         else
11             $NR_i = true$
12 end
13 begin
14     for each node $i$ in $\Gamma_v$ do
15         if $i$ is not a cluster head then
16             for each node $j$ in $\Gamma_i$ do
17                 if $j \neq i$ && $j$ is a cluster head &&
                   $CHR_j = false$ then
18                     ask $i$ to become a cluster head
19                 if $j$ is not a cluster head &&
                   $NR_j = false$ then
20                     ask $i$ to become a cluster head
21                     ask $j$ to become a cluster head
22 end
```

Algorithm 2 is used to generate a connected backbone. The Algorithm is divided into two parts. Assume that cluster head v is executing the algorithm. Lines 1 through 12 are responsible for finding the cluster heads and the normal nodes that can be reached by v using 2-hops. Line 2 checks every 1-hop neighbor (i) of v. Neighbors that are cluster heads are marked as $CHR_i = true$; meaning that cluster head v can reach cluster head i. Neighbors that are normal node are marked as $NR_i = true$; meaning that cluster head v can reach node i. Line 5 checks the neighbors of every 1-hop neighbor. Such neighbors are represented by j, where j is 2-hops away from v. If j is a cluster head that can be reached by v, it is marked as $CHR_j = true$. If j is a normal node that can be reached by v, it is marked as $NR_j = true$.

Lines 13 through 22 loop across the 1-hop and 2-hop neighbors and elect new cluster heads. If cluster head v is connected to a normal neighbor i and i is connected to a cluster head j, but cluster head v can not reach cluster head j, then elect i as a cluster head (lines 17-18). This process connects a cluster head to other cluster heads that are 2-hops away from it. In figure 1 cluster head 15 elects node 12 to be a new cluster head so that it can connect to cluster head 10. If cluster head v is connected to a normal neighbor i and i is connected to a normal node j, but cluster head v can not reach node j, then elect i and j as new cluster heads (lines 19-21). This process connects a cluster head to other cluster heads that are 3-hops away from it. This election process might elect redundant cluster heads. But, sometimes it is good to have more cluster heads because more cluster heads translate to a more reliable network. Also more paths exist between source and destination pairs. If the cluster head made its decision using 3-hop information (larger locality) rather than 2-hop information, less cluster heads would have been elected. In figure 1 cluster head 5 elects nodes 6 and 8 to be new cluster heads. Nodes having higher weight are chosen first to act as cluster heads.

2) Analysis: This step requires each node to send one message ($O(1)$ message complexity). Each normal node sends a message containing its 1-hop information to its neighbors. After executing the algorithm, each cluster head sends a maximum of 2 messages asking some nodes to become cluster heads. The time complexity of the algorithm is $O(\Delta^2)$, because a cluster head needs to loop across its 1-hop and 2-hop neighbors. Note that algorithm 2 is only executed by the cluster heads.

D. Step 4: Reduce Cluster Heads

1) Algorithm: Step 3 elected new cluster heads based on the 1-hop and the 2-hop information. Such a small locality fails to produce an optimal global result. So, some newly elected cluster heads might be redundant. Algorithm 3 presents a very simple algorithm to eliminate unnecessary cluster heads. If cluster head v and its 1-hop neighbors are fully covered by a neighboring cluster head, then cluster head v changes its status to become a normal node. In case of a tie (i.e. both neighboring cluster heads have the exact same neighbors), the cluster head having the lower weight switches to become a normal node. Figure 4 shows the resultant network after executing all the steps.

Algorithm 3: Cluster head reduction

```
Data: 1-hop and 2-hop neighbors of cluster head $v$.
      Notation: $\Gamma_x$ = 1-hop neighbors of node $x$; $N_x = \Gamma_x \cup x$
Result: Cluster head $v$ reduced or kept
1 begin
2     for each node $i$ in $\Gamma_v$ do
3         if $i$ is a cluster head then
4             if $N_v \subset N_i$ then
5                 cluster head $v$ becomes a normal node
6             if $N_v = N_i$ then
7                 if $W_v < W_i$ then
8                     cluster head $v$ becomes a normal node
9 end
```

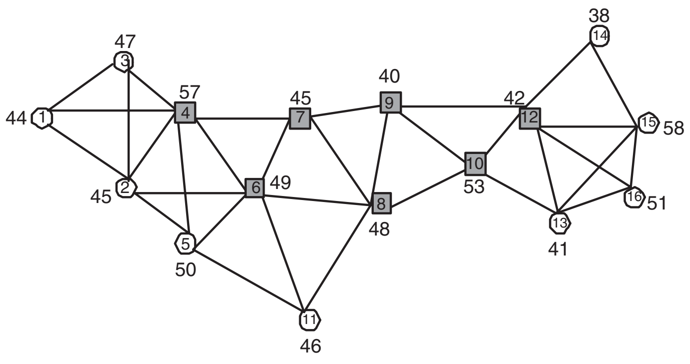

Fig. 4. The resultant network after removing some redundant backbone nodes

2) Analysis: Each cluster head that is reduced sends one message informing its neighbors that it seized being a cluster head ($O(1)$ message complexity). The time complexity of the algorithm is $O(\Delta^2)$ because a cluster head needs to loop across its neighboring cluster heads to check if anyone of them can entirely cover it along with its neighbors.

Theorem 3. *The elected cluster heads form a connected backbone (connected dominating set).*

Proof: Prior to executing algorithm 2, theorem 2 proved that any cluster head can reach another cluster head in at most 3 hops, where the intermediate hops are normal nodes. If we prove that algorithm 2 changes such intermediate nodes to cluster heads, then all the cluster heads should be connected. We split the proof into three parts. Given cluster heads u and v:

1) If $d(u, v) = 1$, then both cluster heads are already connected.
2) If $d(u, v) = 2$, then $\exists$ a normal node i than can connect u and v. Line 18 in algorithm 2 elects such a node as a cluster head.
3) If $d(u, v) = 3$, then $\exists$ two normal nodes i and j that can connect u and v. Lines 20-21 in algorithm 2 elects such nodes as cluster heads.

Therefore, any 2 cluster heads that are at most 3-hops away from each other can be joined together by executing algorithm 2. Thus all the elected cluster heads are connected. After executing the backbone reduction phase, the cluster heads are still connected because the cluster heads removed are already covered by one of their cluster head neighbors.

IV. Cluster formation

Cluster formation involves associating each normal node in the network with a unique cluster head. A normal node can not connect to more than one CH. All nodes connected to a single CH form a cluster. Each elected cluster head sorts its 1-hop neighbors according to their received signal strength (RSS). A cluster head elects members with higher RSS first and keeps track of the traffic load generated by these members. When the traffic load exceeds the capacity of the cluster head, it stops electing members. Nodes left without a cluster head join the cluster head closest to them. The time complexity of the cluster formation algorithm is $O(\Delta \log(\Delta))$ because the cluster head needs to sort its 1-hop neighbors according to RSS. The message complexity is $O(1)$ because the cluster head sends one message informing neighboring nodes of its decision. Figure 5 shows the final topology after cluster formation.

V. Our contribution

The algorithm we propose (DE-CDS) requires 4 steps. In each step, a constant number of messages is sent by each node. So, the overall message complexity of DE-CDS is $O(n)$. The time complexity of DE-CDS is $O(\Delta^2)$ and it is dominated by steps 3 and 4. Many approaches discussing the construction of connected dominating sets in Ad hoc networks have been

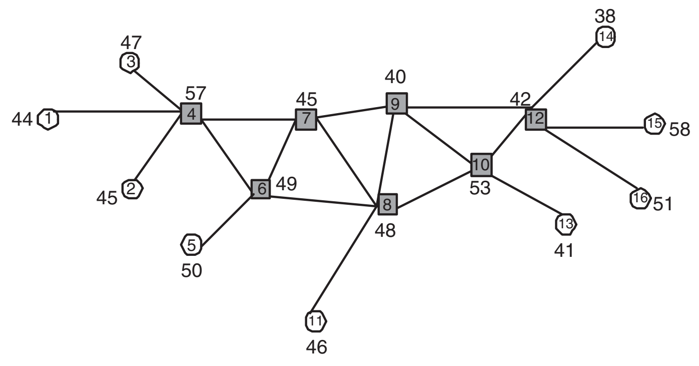

Fig. 5. Network after cluster formation

proposed in literature. According to our knowledge, DE-CDS achieves the best message and time complexity combinations among the previous suggested approaches. In addition to DE-CDS fast convergence, it has the following advantages:

1) DE-CDS takes into account the residual energy, the mobility, and the traffic load of each wireless node when constructing the backbone. These parameters were combined using a fuzzy logic controller [18]. The weight produced by the controller played a very important role in electing the backbone nodes. So, the backbone nodes are ensured to be high quality nodes which enables them to perform their duties (routing discovery, relaying, address assignment, etc.).
2) DE-CDS creates a backbone that provides multiple paths between source and destination pairs. Figure 4 shows an example of such a backbone

VI. Conclusion

In this paper, we proposed a fast distributed and efficient algorithm to find a connected dominating set (DE-CDS) in wireless mobile ad hoc networks. DE-CDS constructs a reliable and energy efficient virtual backbone using $O(n)$ message complexity and $O(\Delta^2)$ time complexity. These complexities are the best achieved complexities among the previously proposed schemes. In our future work, we will compare our approach to other approaches based on: performance, stretch, and degree.

References

[1] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, IT-46(2):388–404, 2000.
[2] P. Gupta, R. Gray, and P. R. Kumar. An experimental scaling law for ad hoc networks. May 2001.
[3] B. Das, R. Sivakumar, and V. Bharghavan. Routing in ad-hoc networks using a virtual backbone. In *6th International Conference on Computer Communications and Networks (IC3N'97)*, pages 1–20, 1997.
[4] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *ICC (1)*, pages 376–380, 1997.
[5] R. Sivakumar, B. Das, and V. Bharghavan. Spine routing in ad hoc networks. *ACM/Baltzer Cluster Computing Journal (special issue on Mobile Computing)*, 1998.
[6] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
[7] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86:165–177, 1990.
[8] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, april 1998.
[9] J. Wu and H. L. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *3rd ACM international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 7–14, 1999.
[10] I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination based broadcasting algorithms in wireless networks. In *IEEE Hawaii International Conference on System Sciences*, 2001.
[11] M. Cardei, X. Cheng, X. Cheng, and D.-Z. Du. Connected domination in multihop ad hoc wireless networks. In *6th International Conference on Computer Science and Informatics*, North Carolina, USA, 2002.
[12] S. Parthasarathy and R. Gandhi. Fast distributed well connected dominating sets for ad hoc networks. Technical Report CS-TR-4559, University of Maryland, 2004.
[13] K. M. Alzoubi, P.-J. Wan, and O. Frieder. Distributed heuristics for connected dominating sets in wireless ad hoc networks. *IEEE ComSoc/KICS Journal on Communication Networks*, 4:22–29, 2002.
[14] K. M. Alzoubi, P.-J. Wan, and O. Frieder. New distributed algorithm for connected dominating set in wireless ad hoc networks. In *IEEE HICSS35*, 2002.
[15] P.-J. Wan, K. Alzoubi, and O. Frieder. Distributed construction of connnected dominating set in wireless ad hoc networks. In *IEEE INFOCOM*, 2002.
[16] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.
[17] K. M. Alzoubi, P.-J. Wan, and O. Frieder. Message-optimal connected-dominating-set construction for routing in mobile ad hoc networks. In *the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2002.
[18] W. El-Hajj, D. Kountanis, A. Al-Fuqaha, and M. Guizani. A fuzzy-based hierarchical energy efficient routing protocol for large scale mobile ad hoc networks (feer). In *IEEE ICC 2006*, Istanbul, Turkey, 2006.