Available online at www.sciencedirect.com
ScienceDirect
Computer Communications 30 (2007) 2880–2891
computer communications
www.elsevier.com/locate/comcom

Fast distributed dominating set based routing in large scale MANETs

Wassim El-Hajj *, Zouheir Trabelsi, Dionysios Kountanis

College of Information Technology, UAE University, United Arab Emirates

Available online 15 June 2007

Abstract

Hierarchical routing techniques have long been known to increase network scalability by constructing a virtual backbone. Even though MANETs have no physical backbone, a virtual backbone can be constructed by finding a connected dominating set (CDS) in the network graph. Many centralized as well as distributed algorithms have been designed to find a CDS in a graph (network). Theoretically, any centralized algorithm can be implemented in a distributed fashion, with the tradeoff of higher protocol overhead. Because centralized approaches do not scale well and because distributed approaches are more practical especially in MANETs, we propose a fast distributed connected dominating set (FDDS) construction in MANETs. FDDS has 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, FDDS achieves the best message and time complexity combinations among the previously suggested approaches. Moreover, FDDS constructs a reliable virtual backbone that takes into account (1) node's limited energy, (2) node's mobility, and (3) node's traffic pattern. Our simulation study shows that FDDS achieves a very low network stretch. Also, when the network size is large, FDDS constructs a backbone with size smaller than other well known schemes found in the literature.

Keywords: MANETs; Dominating set; Virtual backbone; Energy efficiency

1. 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 actually proven [1,2] that a flat network has poor scalability. In [1], theoretical analysis show that the node throughput declines rapidly to zero as the number of nodes in the network increases. It was also shown, that even when the nodes are optimally placed, a network of size n cannot provide a per-node throughput of more than $\frac{c}{\sqrt{n}}$ bits/s, where c is a constant. The experimental results of the scaling law described in [1], employing IEEE 802.11 technologies, are reported in [2]. The results show that the decline in throughput is like $\frac{c}{n^{1.68}}$ bits/s, which is considerably worse than the theoretical results.

To overcome the scalability problem that exists in flat networks, we adopt a hierarchical network design approach. Most hierarchical design approaches for MANETs are based on the concept of cluster head [3–5]. We follow the same concept and we divide the network into a group of clusters. Each cluster is represented by a cluster head (CH). Nodes inside a cluster can only communicate with their associated cluster head and cluster heads can communicate with each other. The collection of cluster heads form the backbone network (also known as virtual backbone). The virtual backbone has to be connected at all times because all network traffic has to be routed through it. Fig. 1 shows an example of our hierarchical design.

We construct the virtual backbone by designing a fast distributed algorithm that finds a connected dominating set (CDS) in the network graph. CDS are the earliest

* Corresponding author.
E-mail addresses: welhajj@uaeu.ac.ae (W. El-Hajj), trabelsi@uaeu.ac.ae (Z. Trabelsi), kountan@cs.wmich.edu (D. Kountanis).

0140-3664/$ - see front matter © 2007 Published by Elsevier B.V.
doi:10.1016/j.comcom.2007.03.011

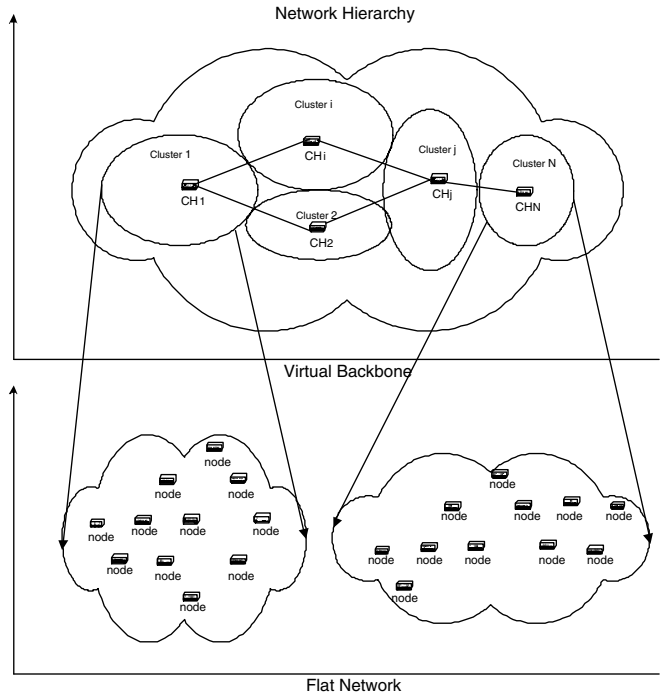


Fig. 1. Hierarchical network design.

structures proposed as candidates for virtual backbones in ad hoc networks [6–8]. 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 [9–11].

In this paper, we propose a fast distributed algorithm to find a connected dominating set (FDDS) in wireless ad hoc networks. FDDS 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 2 includes some survey on the backbone construction. Section 3 describes our proposed distributed approach. Section 4 forms the clusters based on the elected cluster heads. Section 6 discusses our simulation results. Section 7 presents our contributions and Section 8 concludes the paper and discusses future work.

2. 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 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. [7,8,12,13] proposed two algorithms based on the algorithms suggested by Guha and Khuller in [11]. The first algorithm requires each node to know if it has the maximum degree among all nodes in the network. It also requires the nodes chosen as part of the CDS to know which 1-hop and 2-hop neighbors are marked. These two requirements force the flooding of degree information in the whole network. The algorithm starts by growing the CDS starting from the node with the highest degree. In each step, a one-edged or two-edged path emanating from the current CDS is selected until all the network is covered. The algorithm has time complexity of $O(|C|(\Delta + |C|))$ and message complexity of $O(n|C|)$, where n is the total number of nodes, C is the set of nodes constituting the CDS, and Δ is the maximum node degree. The algorithm has an approximation ratio of $2H(\Delta)$, where H is a harmonic function such that $H(\Delta) = \sum_{i=1}^{\Delta} \frac{1}{i} \leqslant \ln(\Delta) + 1$.

The second algorithm contains three stages: approximating the minimum dominating set, constructing a spanning forest of stars, expanding the spanning forest to a spanning tree. In the first stage, each node is assigned a *weight* equal to the number of its unmarked neighbors. The dominating set (D) is initially empty. An unmarked node compares its weight with the weights of its 1-hop and 2-hop neighbors. The node with the maximum weight is included in D. The algorithm continues by iteratively adding the node with the maximum weight to D. This stage terminates when D becomes a dominating set. The stage just described is a translation of Chvàtals greedy algorithm [14] for Set Cover, and thus it guarantees an approximation factor of $H(\Delta)$. In the second and third stages, each edge is assigned a weight equal to the number of endpoints not in D. Then a distributed minimum spanning tree algorithm is used to connect all the nodes in D and thus forming a CDS. The algorithm has time complexity of $O(\Delta(n + |C|))$ and message complexity of $O(n|C| + m + n\log(n))$, where m is the cardinality of the edge set [15].

Alzoubi et al. [16–18] proposed a distributed CDS construction based on computing and then connecting a maximal independent set (MIS). The algorithm starts by electing a leader v in a distributed fashion [19]. The leader election algorithm requires $O(n)$ time complexity and $O(n\log(n))$ message complexity. A spanning tree T rooted at node v is then constructed in order to define the nodes' ranks as follows: v announces that it has level 0. When the children of v receive the announcement, they increase their parent's level by 1 and send an announcement. This process continues until the announcement reaches the leaf nodes. When the leaf nodes receive the announcement, they transmit a LEVEL-COMPLETE message which propagates up the tree until it reaches the root.

Each node sets its rank to be the ordered pair of its level and its ID. The labeling process begins from the root node and finishes at the leaves. The node with the lowest rank marks itself black and broadcasts a DOMINATOR message. The marking process then continues according to the following rules:

- If the first message that a node receives is a DOMINATOR message, it marks itself gray and broadcasts a DOMINATEE message.
- If a node received DOMINATEE messages from all its lower rank neighbors, it marks itself black and sends a DOMINATOR message.

The nodes marked as black constitute the MIS. In [16–18], Alzoubi et al. showed that the distance between any subset of the MIS and its complement is exactly two hops. The second phase constructs a tree spanning all the black nodes, thus creating the CDS. The phase starts by the root joining the CDS and broadcasting an invite message. When a black node joins the CDS, it sends an invitation to all black nodes that are two hops away and outside the current CDS to join the CDS. When a black node receives the message, it joins the CDS along with the gray node that relayed the message. The process terminates when all the black nodes join the CDS. The algorithm has $O(n)$ time complexity and $O(n\log(n))$ message complexity.

Wu et al. [20] 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.

Cheng et al. [21–23] proposed a distributed CDS construction algorithm that has some similarities with the algorithm described by Azoubi et al. [16–18]. The algorithm starts by finding a MIS and then it transforms the set into a CDS. Initially every node has white color. When the algorithm terminates, the MIS nodes become black and the rest of the nodes become gray. The algorithm requires a leader election phase [19] with $O(n)$ time complexity and $O(n\log(n))$ message complexity.

The first phase of the algorithm is similar to the first phase in [16–18], with the *level* changed to *effective degree* (number of white neighbors). It requires that the effective degree of a node be broadcasted as many times as its degree. In the second phase, an approximation of the Steiner tree is used to connect the nodes of the MIS. The approximation is based on the distributed depth-first search spanning tree algorithm. Gray nodes with maximum

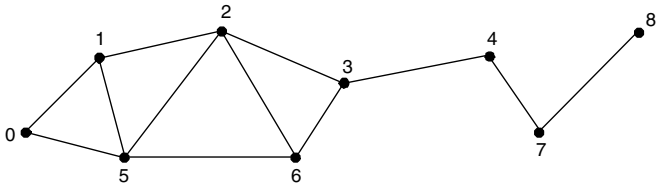


Fig. 2. An example of Unit Disk Graph G containing 9 vertices and 12 edges.

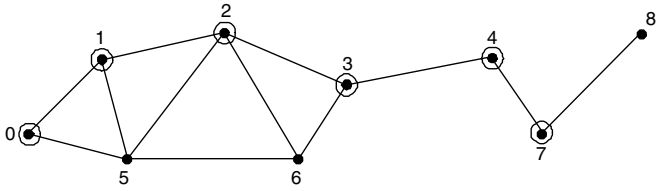


Fig. 3. The computed connected dominating set from Cheng's algorithm contains vertices $\{0,1,2,3,4,7\}$. The optimal solution contains $\{1,2,3,4,7\}$.

black degree are considered as interconnecting nodes. Figs. 2 and 3 are taken from [15] and they illustrate Cheng's algorithm. Fig. 2 presents a graph with 9 vertices and 12 edges. By applying Cheng's algorithm to Fig. 2, vertices $\{0,1,2,3,4,7\}$ are elected as CDS members (Fig. 3). The algorithm has $O(n)$ time complexity and $O(n\log(n))$ message complexity.

Stojmenovic et al. [24] 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.

Parthasarathy et al. [25] 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.

It is worth noting that our proposed distributed approach out performs the approaches discussed above by having a message complexity of $O(n)$ and a time complexity of $O(\Delta^2)$. In the next section, we give a detailed description of our fast distributed connected dominating set (FDDS) construction.

3. Fast distributed connected dominating set (FDDS)

We assume that each node knows its own ID, residual energy (RE), and traffic load (T). A node can calculate

its mobility (M) by measuring its own displacement with respect to its neighbors at different time periods. At time t_1, node X measures the average distance ($D1_{avg}$) between itself and its neighbors. X repeats the same calculation at time t_2 in order to obtain $D2_{avg}$. X can then estimate its mobility by calculating ($D2_{avg} - D1_{avg}$). Note that X estimates the distance to its neighbors by measuring their received signal strengths (RSS). **FDDS** 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.

3.1. Step 1: Neighbor discovery and weight generation

Before FDDS 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.,

$$REn_i = \frac{RE_i}{RE_{max}}, \quad Mn_i = \frac{M_i}{M_{max}}, \quad Tn_i = \frac{T_i}{T_{max}} \tag{1}$$

In [26], we designed a fuzzy logic controller which is used to calculate the node's quality. REn_i, Mn_i, and Tni 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. 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}.

3.2. Step 2: Initial cluster head election

3.2.1. Algorithm

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: Γ = list of my neighbors
Result: An election of one cluster head

```
1 begin
2   if myWeight is the maximum weight among the nodes in Γ 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
```

Fig. 4 shows the elected cluster heads (nodes enclosed in gray 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.

3.2.2. Analysis

This step requires each node to send one message ($O(1)$ message complexity). If the node has the maximum weight

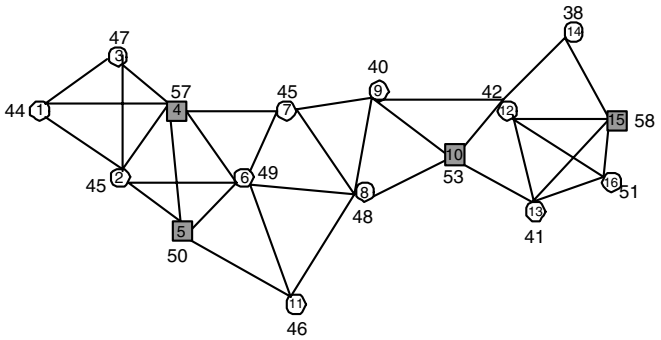


Fig. 4. Initial CH election. The numbers next to the nodes represent the nodes' weights. Nodes 4, 5, 10, and 15 are elected as cluster heads.

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. 4 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 1. 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. □

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 2. Assume that u is a cluster head. Let $C(u) = \{x | x \text{ is a cluster head neighbor of } u\}$. $\forall x \in C(u), d(u,x) = 1$ (Fig. 5(a)). If $x \notin C(u)$, then x is a normal node (Fig. 5(b)). Let $C(x) = \{y | y \text{ is a cluster head neighbor of } x\}$. $\forall y \in C(x)$ and $y \notin C(u)$, $d(u,y) = 2$. If $y \notin C(x)$, then y is a normal node (Fig. 5(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. □

3.3. Step 3: Connect cluster heads

3.3.1. Algorithm

According to Theorem 2, each cluster head is 1-hop, 2-hops, or 3-hops away from another cluster head. In Fig. 4, 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 Fig. 4, 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. Fig. 6 shows the data structure that node 1 uses. All other nodes use similar data structure.

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

Algorithm 2: Connecting the backbone

Data: 1-hop and 2-hop neighbors of cluster head v. Notation: Γ_x = 1-hop neighbors of node x

Result: Cluster heads elected by v

```
1  begin
2      for each node i in Γ_v do
3          if i is a cluster head then
4              CHR_i = true
5              for each node j in Γ_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 Γ_v do
15         if i is not a cluster head then
16             for each node j in Γ_i do
17                 if j ≠ 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
```

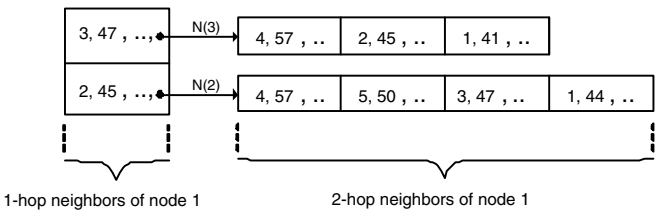


Fig. 6. Node 1 data structure. N(3) and N(2) represent the neighbors of nodes 3 and 2, respectively. The nodes are sorted according to their weights. Such a data structure enables node 1 to know its 1-hop and 2-hop neighbors.

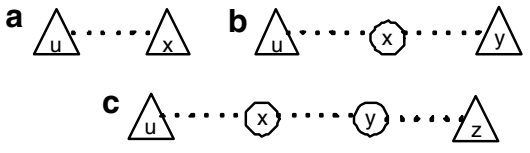


Fig. 5. For a given cluster head u, u can reach another cluster head in: (a) 1-hop: $d(u,x) = 1$, (b) 2-hops: $d(u,y) = 2$, (c) 3-hops: $d(u,z) = 3$.

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 cannot 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 Fig. 4, 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 cannot 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 Fig. 4, 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.

3.3.2. 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.

3.4. Step 4: Reduce cluster heads

3.4.1. 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 some 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. Fig. 7 shows the resultant network after executing all the steps.

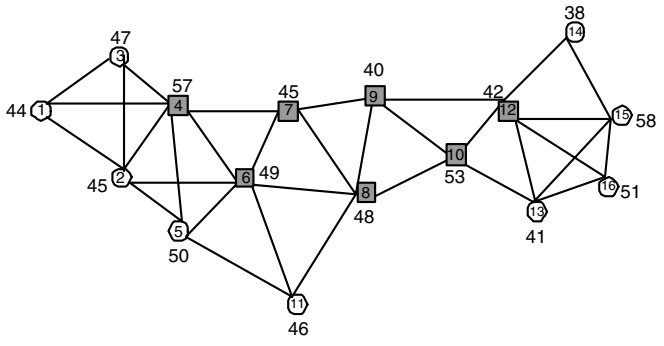


Fig. 7. The resultant network after removing some redundant backbone nodes.

Algorithm 3: Cluster head reduction

Data: 1-hop and 2-hop neighbors of cluster head v. Notation: Γ_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 Γ_v do
3     if i is a cluster head then
4       if N_v ⊂ 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
```

3.4.2. 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. Note that, the storage capacity is directly related to the time complexity and it is also $O(\Delta^2)$. The smaller the storage capacity, the smaller the time complexity and vise versa.

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

Proof 3. 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. □

4. Cluster formation

Cluster formation involves associating each normal node in the network with a unique cluster head. A normal node cannot 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. Fig. 8 shows the final topology after cluster formation.

5. Mobility and traffic models

In the following section, we discuss the mobility and the traffic models that we used in our simulation. Note that changing the models will definitely change the simulation results. For this reason, we selected models that are widely used and very realistic.

5.1. Mobility model

Evaluating the performance of a mobile ad hoc network highly depends on the mobility model used. The mobility model should dictate the movement of the mobile nodes in a realistic way. Two of the most mobility models used by researchers are the Random Walk Mobility Model [27] and the Random Waypoint Mobility Model [28,29].

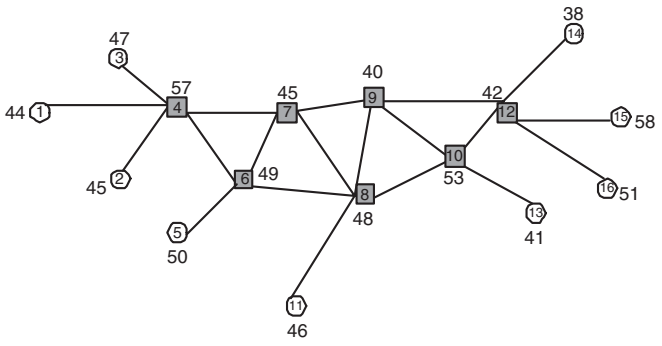


Fig. 8. Network after cluster formation.

Each one of these models generates unrealistic scenarios that make them inappropriate for mobile ad hoc network simulation. An alternative is to use the Gauss–Markov Mobility Model [30] that fixes the problem encountered by the previous two models.

The Random Walk Mobility Model was developed to mimic the erratic movement of entities in nature that move in unpredictable ways [27]. A mobile node moves from one location to another by choosing two random values corresponding to speed and direction. Speed and direction are chosen to be within predefined ranges, $[speedmin, speedmax]$ and $[0, 2\pi]$, respectively. After a certain time period t, or a distance d, new values for speed and direction are generated. No relation exists between the current and the past movements of the node. This might lead to unrealistic scenarios, where a node stops suddenly or makes sharp turns. Also, if the time period (t) or the distance (d) were small values, the node will be moving abruptly in a small region.

The Random Waypoint Mobility Model uses pause time between changes in speed and/or direction [28,29]. A mobile node starts by pausing for a certain time period. Then it moves from one location to another by choosing two random values corresponding to speed and destination. Speed is chosen to be uniformly distributed between $[minspeed, maxspeed]$. The mobile node travels towards the new destination at the selected speed. Upon arrival, it pauses for a certain period of time and then starts the process again. The Random Waypoint movement is similar to the Random Walk movement when the pause time is 0. Hence, the Random Waypoint Mobility Model suffers from the same problems that the Random Walk Mobility Model suffers from.

To eliminate the problems (sudden stops and sharp turns) encountered by the Random Walk and the Random Waypoint mobility models, we use the Gauss–Markov Mobility Model. The Gauss–Markov Mobility Model was originally proposed for the simulation of a personal communications service (PCS) [31]; however, this model has been used for the simulation of an ad hoc network protocol [30]. The main advantage of this model is allowing past velocities (and directions) to influence future velocities (and directions). A mobile node starts moving using a current speed and direction. At fixed intervals of times, n, new speed and direction values are assigned to the mobile node. These values are calculated based on the values used in the previous time interval and a random variable. The speed and direction at the nth instance are given by the following equations:

$$s_n = \alpha s_{n-1} + (1-\alpha)\bar{s} + \sqrt{(1-\alpha^2)s_{x_{n-1}}} \quad (2)$$

$$d_n = \alpha d_{n-1} + (1-\alpha)\bar{d} + \sqrt{(1-\alpha^2)d_{x_{n-1}}} \quad (3)$$

where s_n and d_n are the new speed and direction of the mobile node at time interval n; α, $0 \leq \alpha \leq 1$, is the tuning parameter used to vary the randomness; $\bar{s}$ and $\bar{d}$ are

constants representing the mean value of speed and direction as $n \to \infty$; $s_{x_{n-1}}$ and $d_{x_{n-1}}$ are random variables from a Gaussian distribution. Totally random values (or Brownian motion) are obtained by setting $\alpha = 0$ and linear motion is obtained by setting $\alpha = 1$. Intermediate levels of randomness are obtained by varying the value of α between 0 and 1.

At each time interval, the next location is calculated based on the current location, speed, and direction of movement. Specifically, at time interval n, the mobile node's position is given by the equations:

$$x_n = x_{n-1} + s_{n-1}\cos(d_{n-1}) \tag{4}$$

$$y_n = y_{n-1} + s_{n-1}\sin(d_{n-1}) \tag{5}$$

where (x_n, y_n) and (x_{n-1}, y_{n-1}) are the x and y coordinates of the mobile node's position at the nth and $(n-1)$st time intervals, respectively, and s_{n-1} and d_{n-1} are the speed and direction of the mobile node, respectively, at the $(n-1)$st time interval [32]. The Gauss–Markov Mobility Model can eliminate the sudden stops and sharp turns encountered by the models discussed above by allowing past velocities (and directions) to influence future velocities (and directions).

5.2. Traffic model

Traffic generation is simulated using the Poisson distribution. Traffic modeled after the Poisson distribution has exponential inter-arrival time and exponential holding time as shown in Fig. 9.

Exponential distributions are a class of continuous probability distribution. They are often used to model the time between events that happen at a constant average rate [33].

6. Simulation results

In this section, we present the performance evaluation of FDDS through simulation. We focus on evaluating the structural properties of FDDS including: (1) average number of hops between randomly generated (source, destination) pairs, (2) network stretch, and (3) backbone size. All data points were averaged over 10 simulation runs.

6.1. Average number of hops

In this experiment, a uniform random generator is used to generate networks of different sizes such that the resultant networks are connected. Networks with sizes $n = 10 \to 100$ are generated in a 500 square units space. For a given network A, n (source, destination) pairs are randomly generated. The shortest paths between all pairs are calculated and then averaged to obtain the average number of hops in network A. We then vary the transmission range (R) of every node and compare the average number of hops in flat network A with the average number of hops in the network generated using FDDS. Figs. 10–12 show this comparison for different network sizes when $R = \{150, 250, 350\}$.

All these figures show that the average number of hops in a flat network is less than that in FDDS. This is true because FDDS reorganizes the whole network by removing certain links. So, some paths that exists in a flat network might seize to exist in FDDS. The elimination of such links makes FDDS use slightly longer paths to connect a certain source to its destination. When the transmission range increases, the network becomes more dense. The more dense the network is, the lower the average number of hops is. This can be clearly seen by looking at Figs. 10–12. The most interesting observation that we can obtain from these figures is that the difference between the average number of hops in both networks (flat and FDSS) is very small even when the network size increases.

6.2. Network stretch

Let S_{ij} be the length of the shortest path between nodes i and j in flat network A. Let S'_{ij} be the length of the shortest path between nodes i and j in the network generated using FDDS. Network stretch is defined as:

$$\mathrm{NS} = \mathrm{Max}\left(\frac{S'_{ij}}{S_{ij}}\right) \quad \forall i, j \tag{6}$$

In general, a route is efficient if its total cost (or total hop number) is no more than a constant factor of the minimum total cost (or total hop number) needed to connect the source and the destination in the original (flat) communication graph. The constant is called cost (or hops) stretch factor. So, any hierarchical design is preferred to have low stretch. In this experiment, we use the same parameters as those in Section 6.1, i.e, network size = n, grid size = 500 square units, and n (source, destination) pairs are randomly generated. For a given network, we vary R and calculate the network stretch. R can take values of 150, 250, or 350.

Fig. 13 shows the hop stretch of networks with different sizes and different transmission ranges. As the transmission range increases (network is more dense), network stretch decreases. This is true because when the nodes' coverage increases, more shorter paths can be achieved between (source, destination) pairs. This can be also seen by looking

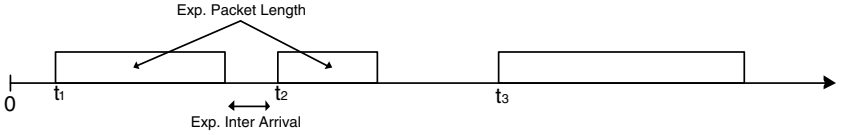


Fig. 9. Poisson Traffic Model: exponential inter-arrival time and exponential holding time.

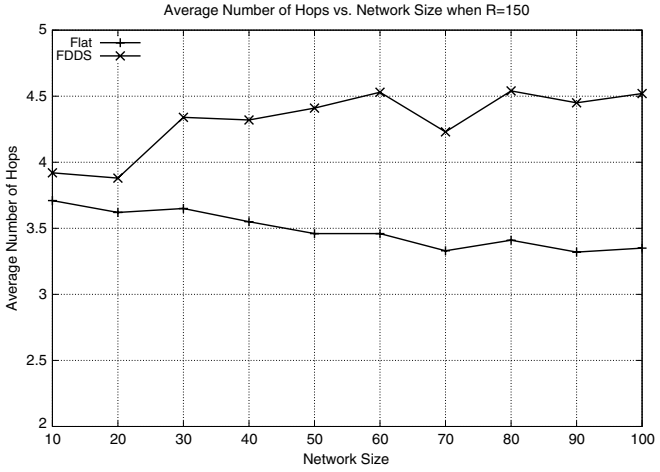


Fig. 10. Average number of hops in flat network vs. FDDS when $R = 150$.

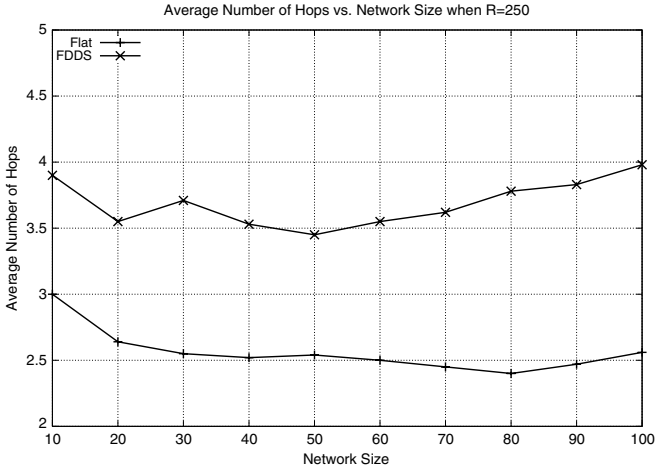


Fig. 11. Average number of hops in flat network vs. FDDS when $R = 250$.

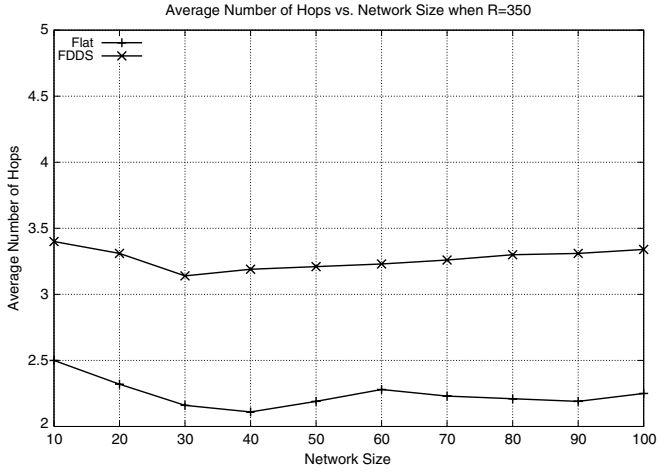


Fig. 12. Average number of hops in flat network vs. FDDS when $R = 350$.

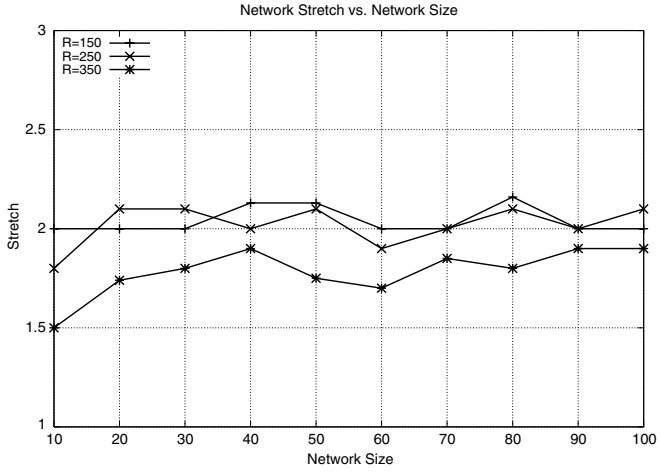


Fig. 13. Hop stretch of networks with different sizes when $R = \{150, 250, 350\}$.

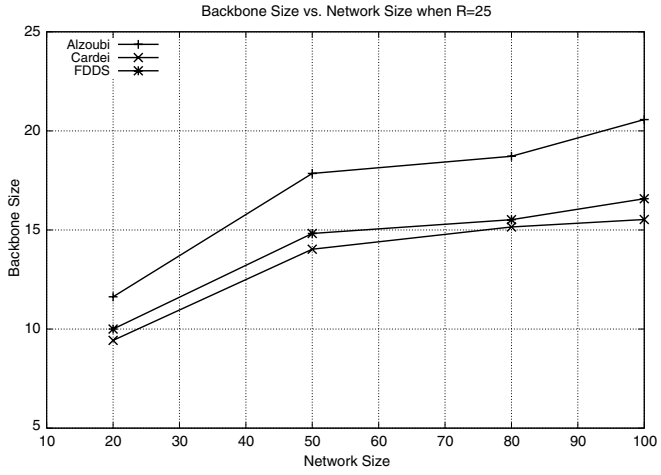
at Figs. 10–12 and noticing that the average number of hops decreases as the node transmission range increases. The very important observation that can made from Fig. 13 is that no matter what the node transmission range is and the network size is, the network stretch is always very close to 2. Having the network stretch to be a constant factor and a very small constant indeed, proves the efficiency of FDDS in terms of constructing the virtual backbone (connected dominating set). We cannot really generalize this observation and make it a theorem, but one of our future goals is to theoretically calculate the order of complexity of the network stretch.

6.3. Backbone size

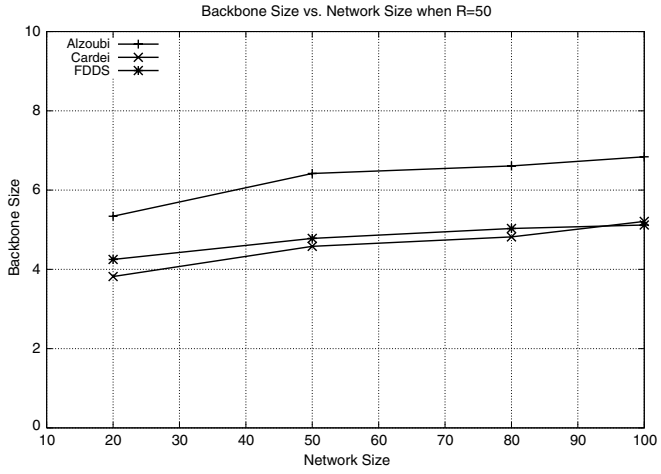
The virtual backbone is mainly used to route traffic between all (source, destination) pairs. Therefore, the virtual backbone nodes (CHs) have higher computation and communication burden than other nodes. Thus, one of the important parameters is the number of nodes in the virtual backbone after it is constructed. If the virtual backbone size is large, many nodes might be extensively used leading to their battery drainage. So, the smaller the size of the backbone is, the better the network performance is. But a smaller size backbone means that all the burden is only put on a small set of nodes. This burden might overwhelm the CHs and bring them to a halt. Therefore, sometimes it is better to have more backbone nodes with each one having less duties. This technique will distribute the load on many CHs and more importantly it will allow for multiple paths to exist between (source, destination) pairs.

In this experiment, we change the simulation parameters. We randomly generate networks of sizes 20, 50, 80, and 100 and place them in a 100 square units space. The transmission range of every node can be either 25 or 50 units. We vary the transmission range and compare the size of the backbone when networks are generated using different approaches.

We implemented the approaches suggested by Alzoubi [34] and Cardei [22]. Figs. 14(a) and (b) present the backbone sizes of different networks when the approaches mentioned above are used. When the node transmission range is small (Fig. 14(a)), our approach (FDDS) generates a backbone size that is less than that of Alzoubi's, but larger than that of Cardei's. When the node transmission range is large (Fig. 14(b)), our approach (FDDS) generates a backbone size than is less than that of Alzoubi's and very close to that of Cardei's. The following important observation can be made from Fig. 14(b): as the network size increases, our approach generates backbones with sizes that are very close (equal or smaller) to that of Cardei's. By looking at these figures, we can say that *FDDS performs better when the network is large and more dense* (*R is large*). Note that we could have easily reduced the backbone size when using FDDS by improving the reduction algorithm described in Section 3.4. But, we chose to keep the backbone intact in order to provide more paths between (source, destination) pairs. This is important when designing an energy efficient routing protocol. For example, paths that contain CHs with low energy are avoided.



(a) $R = 25$



(b) $R = 50$

Fig. 14. Backbone size when the following approaches are used: FDDS, Alzoubi's [34], and Cardei's [22].

7. Our contribution

The algorithm we proposed (FDDS) requires four steps. In each step, a constant number of messages is sent by each node. So, the overall message complexity of FDDS is $O(n)$. The time complexity of FDDS is $O(\Delta^2)$ and it is dominated by Steps 3 and 4 of the algorithm. Many approaches discussing the construction of connected dominating sets in ad hoc networks have been proposed in literature. According to our knowledge, FDDS achieves the best message and time complexity combinations among the previous suggested approaches. In addition to FDDS's fast convergence, it has the following advantages:

(1) FDDS 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 the fuzzy logic controller described in [26]. The *weights* 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) FDDS creates a backbone that provides multiple paths between source and destination pairs. Fig. 7 shows an example of such backbone.
(3) FDDS produces a hierarchical structure with very low network stretch. Such property is very important when designing hierarchical protocols.
(4) FDDS performs better when the network is large and more dense. Specifically, it constructs a backbone with small size in large scale and dense networks.

8. Conclusion and future work

In this paper, we proposed a fast distributed and efficient algorithm (FDDS) to find a connected dominating set in wireless mobile ad hoc networks. FDDS constructs a reliable and energy efficient virtual backbone using $O(n)$ message complexity and $O(\Delta^2)$ time complexity. To our knowledge, these complexities are the best achieved complexities among the previously proposed schemes. Some of our future goals are: (1) theoretically calculate the order of complexity of the network stretch, (2) design a network maintenance protocol that preserves the structure of the CDS, and (3) design an efficient routing protocol that uses the CDS structure.

References

[1] P. Gupta, P.R. Kumar, The capacity of wireless networks, IEEE Transactions on Information Theory IT-46 (2) (2000) 388–404.

[2] P. Gupta, R. Gray, P.R. Kumar, An experimental scaling law for ad hoc networks, May 2001.
[3] D.J. Baker, A. Ephremides, The architectural organization of a mobile radio network via a distributed algorithm, IEEE Transactions on Communications (1981) 1694–1701.
[4] D.J. Baker, J. Wieselthier, A. Ephremides, A distributed algorithm for scheduling the activation of links in a self-organizing, mobile, radio network, in: IEEE ICC'82, pp. 2F.6.1–2F.6.5.
[5] M. Gerla, J.T.-C. Tsai, Multicluster, mobile, multimedia radio network, ACM-Baltzer Journal of Wireless Networks 1 (3) (1995) 255–265.
[6] B. Das, R. Sivakumar, V. Bharghavan, Routing in ad-hoc networks using a virtual backbone, in: Sixth International Conference on Computer Communications and Networks (IC3N'97), 1997, pp. 1–20.
[7] B. Das, V. Bharghavan, Routing in ad-hoc networks using minimum connected dominating sets, in: ICC (1), 1997, pp. 376–380.
[8] R. Sivakumar, B. Das, V. Bharghavan, Spine routing in ad hoc networks, ACM/Baltzer Cluster Computing Journal (1998) (special issue on Mobile Computing).
[9] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
[10] B.N. Clark, C.J. Colbourn, D.S. Johnson, Unit disk graphs, Discrete Mathematics 86 (1990) 165–177.
[11] S. Guha, S. Khuller, Approximation algorithms for connected dominating sets, Algorithmica 20 (4) (1998) 374–387.
[12] Bevan Das, Raghupathy Sivakumar, Vaduvur Bharghavan, Routing in ad hoc networks using a spine, in: International Conference on Computers and Communications Networks, Las Vegas, NV, September 1997.
[13] R. Sivakumar, B. Das, V. Bharghavan, An improved spine-based infrastructure for routing in ad hoc networks, in: IEEE Symposium on Computers and Communications, Athens, Greece, June 1998.
[14] V. Chvaatal, A greedy heuristic for the set-covering problem, Mathematics of Operation Research 4 (3) (1979) 233–235.
[15] J. Blum, M. Ding, A. Thaeler, X. Cheng, Connected dominating set in sensor networks and manets, in: D.-Z. Du, P. Pardalos (Eds.), Handbook of Combinatorial Optimization, Kluwer Academic Publishers, 2004, pp. 329–369.
[16] K.M. Alzoubi, P.-J. Wan, O. Frieder, Distributed heuristics for connected dominating sets in wireless ad hoc networks, Journal of Communications and Networks 4 (1) (2002).
[17] K. Alzoubi, P.-J. Wan, O. Frieder, New distributed algorithm for connected dominating set in wireless ad hoc networks, in: Proceedings of the 35th Hawaii International Conference on System Sciences, Big Island, Hawaii, 2002.
[18] P.-J. Wan, K. Alzoubi, O. Frieder, Distributed construction of connected dominating set in wireless ad hoc networks, in: IEEE INFOCOM, 2002.
[19] I. Cidon, O. Mokryn, Propagation and leader election in multihop broadcast environment, in: Twelfth International Symposium on DIStributed Computing (DISC98), Greece, September 1998, pp. 104–119.
[20] J. Wu, H.L. Li, On calculating connected dominating set for efficient routing in ad hoc wireless networks, in: Third ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, 1999, pp. 7–14.
[21] S. Butenko, X. Cheng, D.-Z. Du, P.M. Pardalos, On the construction of virtual backbone for ad hoc wireless networks, in: S. Butenko, R. Murphey, P.M. Pardalos (Eds.), Cooperative Control: Models, Applications and Algorithms, Kluwer Academic Publishers, 2003, pp. 43–54.
[22] M. Cardei, X. Cheng, X. Cheng, D.-Z. Du, Connected domination in multihop ad hoc wireless networks, in: Sixth International Conference on Computer Science and Informatics, North Carolina, USA, 2002.
[23] X. Cheng, Routing Issues in Ad Hoc Wireless Networks, PhD thesis, Department of Computer Science, University of Minnesota, 2002.
[24] I. Stojmenovic, M. Seddigh, J. Zunic, Dominating sets and neighbor elimination based broadcasting algorithms in wireless networks, in: IEEE Hawaii International Conference on System Sciences, 2001.
[25] S. Parthasarathy, R. Gandhi, Fast distributed well connected dominating sets for ad hoc networks, Technical Report CS-TR-4559, University of Maryland, 2004.
[26] W. El-Hajj, D. Kountanis, A. Al-Fuqaha, 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.
[27] V. Davies, Evaluating mobility models within an ad hoc network, Master's thesis, Colorado School of Mines, 2000.
[28] J. Broch, D.A. Maltz, D.B. Johnson, Y.-C. Hu, J. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols, in: Proceedings of MobiCom, ACM International Conference on Mobile Computing and Networking, Dallas, USA, October 1998.
[29] D. Johnson, D. Maltz, Dynamic source routing in ad hoc wireless networks, in: T. Imelinsky, H. Korth (Eds.), Mobile Computing, Kluwer Academic Publishers, 1996, pp. 153–181.
[30] V. Tolety, Load reduction in ad hoc networks using mobile servers, Master's thesis, Colorado School of Mines, 1999.
[31] B. Liang, Z. Haas, Predictive distance-based mobility management for pcs networks, in: Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), March 1999.
[32] T. Camp, J. Boleng, V. Davies, A survey of mobility models for ad hoc network research. Communication & Mobile Computing (WCMC), Trends and Applications 2 (5) (2002) 483–502 (special issue on Mobile Ad Hoc Networking: Research).
[33] Wikipedia, Exponential distribution, 2002. Available from: <http://en.wikipedia.org/wiki/Exponentialdistribution>.
[34] K.M. Alzoubi, P.-J. Wan, O. Frieder, Message efficient distributed algorithms for connected dominating set in wireless ad hoc networks, manuscript, 2001.

Wassim El-Hajj is currently an Assistant Professor in UAE University at Al Ain, UAE. He got his Ph.D. in Computer Science from Western Michigan University. He received his masters in Computer Science from Western Michigan University and his Bachelor from the American University of Beirut. His research interests include artificial intelligence, QoS routing, performance analysis of telecommunication networks, Firewalls, IDSs, and ad hoc network security.

Zouheir Trabelsi received his Ph.D. from Tokyo University of Technology and Agriculture, Japan, in the field of computer science, March 1994. From April 1994 until December 1998, he was a computer science researcher at the Central Research Laboratory of Hitachi in Tokyo, Japan. From November 2001 until October 2002, he was a visiting Assistant Professor at Pace University, New York, USA. Currently, he is an Assistant Professor at the College of Information Technology, the United Arab Emirates University. His research areas are mainly networking security, Intrusion Detection and Prevention, Firewalls, TCP/IP Covert Channels, network protocol design, distributed algorithms, mobile ad hoc networks, QoS.

Dionysios Kountanis received his Ph.D. degree in Computer Science from the University of Pennsylvania in 1977. Currently, he is an Associate Professor at the Computer Science Department of Western Michigan University. Prior experience includes service on the faculties of Rutgers University and Temple University and as a systems programmer for Dun and Bradstreet. His research interests include: graph algorithms, computational geometry, computer architectures, artificial intelligence, mobile ad hoc networks, and telecommunication networks.