Minimizing Recovery State in Geographic Ad Hoc Routing

Noa Arad and Yuval Shavitt, *Senior Member*, *IEEE*

Abstract—Geographic ad hoc networks use position information for routing. They often utilize stateless greedy forwarding and require the use of recovery algorithms when the greedy approach fails. We propose a novel idea based on virtual repositioning of nodes that allows to increase the efficiency of greedy routing and significantly increase the success of the recovery algorithm based on local information alone. We explain the problem of predicting dead ends, which the greedy algorithm may reach and bypassing voids in the network, and introduce Node Elevation Ad hoc Routing (NEAR), a solution that incorporates both virtual positioning and routing algorithms that improve performance in ad hoc networks containing voids. We demonstrate by simulations the advantages of our algorithm over other geographic ad hoc routing solutions.

Index Terms—Ad hoc, routing, distributed, elevation, repositioning.

1 Introduction

1.1 Ad Hoc Networks

Ad hoc networks are infrastructureless networks, made up of mobile nodes, which are using their neighbors as a mean of communication with other nodes in the network. Ad hoc networks change their topology, expressed by the node connectivity, over time, as the nodes change their position in space. Routing schemes in mobile ad hoc networks can be crudely divided into two groups: topology-based routing, and position-based routing. Topology-based routing uses existing information in the network about links; it includes table driven protocols such as DSDV [1] and CGSR [2] and on demand protocols such as AODV [3], DSR [4], and more. Position-based routing, on the other hand, is based on the nodes position in space and their local neighboring node position.

1.2 Position-Based Routing

Geographic ad hoc networks using position-based routing are targeted to handle large networks containing many nodes. Such networks are unsuited to use topology-based algorithms as the amount of resources required would be enormous. The advantage in geographic networks is the ability to deliver a packet from its source to the destination based as much as possible on local information without keeping networkwide information [5]. While topology-based algorithms may be more efficient in delivering packets in terms of delivery success probability and route optimality, position-based routing has the advantage of modest memory requirement at the node and low control message overhead, which also translate to more efficient use of power resources [6]. While this is not a full comparison between the two groups, it emphasizes the will to center position-based routing algorithms as much as possible on local information.

Position-based routing algorithms can employ either single path, multipath, or flooding. Flooding protocols are usually restricted directional, such as DREAM [7] and LAR [8]; the flooding is done only in a section of the network, which is selected based on the source and destination node location. Multipath protocols such as c-GEDIR [9] attempt to forward the message along several routes toward its destination in order to increase the probability of finding a feasible path. Single path protocols, on the other hand, aim for a good resource consumption to throughput ratio. Most common among the single path protocols are those based on greedy algorithms. The greediness criteria can be distance, minimum number of hops, power (best usage of battery resources), etc.

Position-based routing algorithms typically use *Location Services* to obtain the destination's current position. Flooding [7], [8], quorum-based [10], [11], hierarchical [12] and flat [13], [14] hashing-based protocols are used for this purpose.

1.3 The Concavity Problem and Solutions

A major issue in greedy routing algorithms is how to proceed when a concave node is reached, i.e., a node that is closer than any of its neighbors to the destination [15] (see Section 2 for an exact definition). The simplest solution is to allow the routing algorithm to forward the packet to the best matching neighbor, excluding the sender itself. Such a solution can guarantee the packet delivery but can result in routing loops in algorithms that are otherwise loop free. Other solutions require switching to a recovery algorithm that guarantees packet delivery. They can be classified as memory based and memory free.

Definition 1.1. ***Recovery*** *is the routing mode taken when a concave node is reached through greedy routing. In the recovery stage, the message is transmitted from the concave node toward the destination according to a nongreedy algorithm. A node may exit recovery mode if it is not concave, so greedy routing can be resumed.*

- *N. Arad is with the School of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978, Israel. E-mail: noa@eng.tau.ac.il.*
- *Y. Shavitt is with the School of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978, Israel E-mail: shavitt@eng.tau.ac.il.*

Manuscript received 13 Mar. 2007; revised 1 Dec. 2007; accepted 13 May 2008; published online 2 June 2008.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2007-03-0078. Digital Object Identifier no. 10.1109/TMC.2008.86.

One routing algorithm that employs memory is the Greedy/Flooding algorithm [16]. This algorithm, once reaching a concave node floods the message toward the target. The algorithm stores a list of all neighbor nodes that declare their concavity, and avoid flooding to them. A different use of the memory is done in Terminodes [17], where the routing distinguishes between local and remote routing. While the local routing is topology-based, the remote routing is greedy. However, the remote routing has a forced list of anchoring nodes, which the path should loosely traverse.

An anchor is a point, described by geographical coordinates. It does not, in general, correspond to any node location [17].

The anchored node list is stored in memory and updated every time the packet passes in the vicinity of the next anchor node. Among other algorithms that incorporate memory, we can find the INF [18] and SAGF [19] algorithms. Intermediate node forwarding (INF) is a probabilistic solution for routing around voids using intermediate geographic locations, and SAGF is a spatial aware geographic forwarding solution suitable mainly for networks with preassigned routes.

Definition 1.2. ***Void*** *is an area free of nodes that cannot be routed through. A void diameter is larger than a node's transmission radius.*

Multilevel-clustering approaches such as Landmark [20], LANMAR [21], L+ [22], and Safari [23] elect certain nodes as cluster heads called Landmarks. These cluster heads in turn select higher level cluster heads, up to some desired level. The cluster heads are later used to forward the packet to its destination based on the destination address.

Recovery algorithms without memory often use planar graphs for routing (A planar graph is a graph that can be drawn on a plane, such that no two edges intersect). One of the first works in this area was "Compass Routing II" [24], also called "Face Routing," proved that Delaunay triangulations of point sets on the plane supports compass routing and guarantees delivery. This algorithm was the basis for further suggestions such as AFR [25] and GOAFR [26].

Bose et al. [27] introduced an improved face routing algorithm, FACE-2, which combined with the GEDIR [16] algorithm form the GFG algorithm.

An important algorithm similar to GFG is Greedy Perimeter Stateless Routing (GPSR) [28]. It, too, requires the network to be planar in order to accomplish successful routing. This property is achieved by creating a Gabriel Graph (GG) or a subset of it, Relative Neighborhood Graph (RNG). In GPSR, a packet is initially routed using a greedy algorithm until reaching a concave node. It then switches to perimeter mode, traversing the face of the planar graph using the right-hand rule, until it recovers from the local maxima, and the greedy routing can continue.

Definition 1.3. ***Perimeter Routing*** *is a recovery routing mode in which a packet traverses the face of a routing obstacle.*

The PAGER [29] algorithm represents a different approach to the recovery problem but in sensor networks where all messages are destined to a single node. It divides a sensor network graph into functional subgraphs and provides each node with message forwarding directions based on these subgraphs. PAGER defines concave nodes as "Shadow Nodes" and a group of shadow nodes as a "Shadow Area." It later routes messages through the graph using cost gradients of the shadow areas.

Fang et al. [30] also suggested an algorithm for sensor networks, called BOUNDHOLE. The algorithm builds routes around holes, which they define as connected regions of the network with boundaries consisting of all the stuck nodes. The routes can be found on demand or in a preprocessing phase, based on the TENT rule that checks for stuck angles [30].

One problem that recovery protocols do not prevent is that the packet always needs to reach a dead end before the recovery algorithm takes charge and delivers the packet to its destination. This is problematic when the algorithm enters a long cul-de-sac, as the retreat to a point where an alternative path can be found is long. This work proposes a novel scheme to deal with this problem by preventing the routing algorithm from entering concave areas. The scheme is comprised of three contributions:

- A novel algorithm that uses local information to identify concave areas, not necessary only a single node. The algorithm assigns virtual coordinates to nodes.
- A routing scheme that is based on the virtual coordinates.
- An obstacle bypass procedure.

2 The Concavity Problem

Definition 2.1. *A* ***Concave node*** *is a node that has no neighbor that can make a greedy progress toward some destination (for the greedy routing algorithm in use).*

Our definition is slightly more accurate than the common one [15]. Since position-based routing uses local information for forwarding decisions, a concave node cannot be predicted in advance, based on the position of its neighbor nodes. Even using the information of the 2-neighborhood cannot prevent reaching concave nodes, though can improve decisions made during the algorithm.

Assuming one uses a recovery algorithm that switches back to greedy mode once recovered from the concave situation, the number of backtracking packet transmissions required to switch back to greedy mode can vary between just a few hops to a very long retreat. Fig. 1 shows an example path to a concave node that is reached only after several hops, and only at this node that the recovery process (not shown in the figure) begins. In addition, [31] reviews additional deficiencies of perimeter-based recovery algorithms: network disconnection due to graph planarization, nodes mobility causing routing loops, and routing in the wrong direction causing error due to mobility or increasing the number of hops.

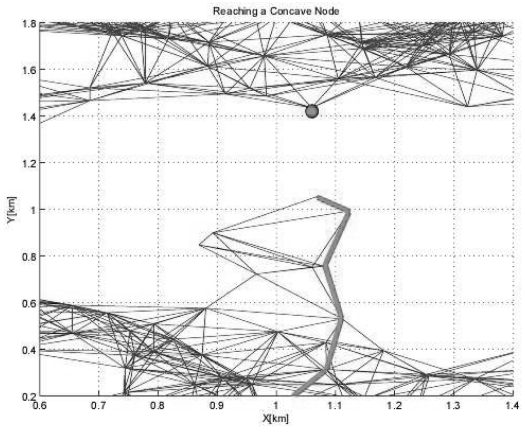

Fig. 1. A concave node requiring a recovery path. The picture shows an enlarged part of a network with a dominant void (like the one depicted in Fig. 5a).

We thus extend the definition of concavity:

Definition 2.2. *A node has **wide sense concavity** if some destination cannot be reached by any of the node's neighbors using only greedy progress.*

We say that a concave node (not in the wide sense) has a first degree concavity. A concave node in the wide sense is said to have a concavity of the nth degree, if the smallest concavity degree of its neighbors is $n-1$.

3 The NEAR Algorithm

Many of the problems of position-based routing originate from the fact that the shape of the network is unknown a priori, and it is dynamically changing due to nodes mobility. The lack of information prevents the network shape from being considered a substantial part of the routing process and does not allow educated routing decisions. GPSR [28], for example, switches from recovery mode back to greedy mode when the current node is closer to the destination than the node who switched to perimeter mode (Definition 1.3). However, there is no guarantee that this node, or the next one, will not be another concave node, a local maximum for the greedy algorithm on the perimeter face.

We suggest a way to virtually reposition nodes in the network, so that greedy routing decisions can be wisely taken and the recovery process can be significantly improved or avoided altogether. Node repositioning has several goals. The first is to identify and mark concave nodes. Identifying a concave node is simple, as every node can do so locally by analyzing its connectivity. If the angle between two adjacent node's neighbors exceeds 180 degrees, then the node is necessarily concave for routing in this direction (we will see later that even here one must be cautious in deciding about concavity). Our method marks a concave node by elevating it. In an N-dimension coordinate system, an $N+1$ dimension is added that indicates, if its coordinate is nonzero, the node is virtually repositioned. The rest of the N coordinates are updated as well to reflect the node's connectivity, as will be later described.

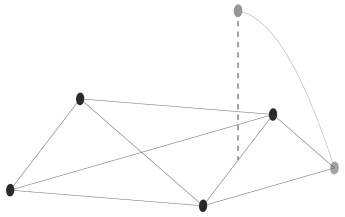

Fig. 2. A repositioning example.

Definition 3.1. ***Elevation** is the process of virtually repositioning nodes, at the end of which a node is assigned an $N+1$ dimension with a nonzero value.*

Definition 3.2. ***Virtual coordinates** are the coordinates assigned to a node by the elevation process, reflecting its position in $N+1$ dimensions.*

For simplicity, let us assume that nodes have two coordinates, describing their X and Y coordinates. In this case, a concave node will be assigned a positive virtual Z coordinate that reflects its distance from its neighbors. Fig. 2 demonstrates a simple repositioning of a node. The figure represents a part of a network, with four nonconcave nodes, on the left, and a fifth, right most, concave node. The virtual position of the node after applying NEAR is shown above the original nodes plane. Note that the projection of the node's virtual coordinates on the x-y plane lies between its two neighbors.

Definition 3.3. *A **Floating** node is a node, which was elevated. Every floating node is concave in the wide sense.*

A second goal of repositioning is to improve greedy routing. Our greedy algorithm avoids using the floating nodes and thus does not get stuck in a concave area. This way, we can avoid switching to recovery mode in many cases.

The third purpose, which is derived from the implementation of repositioning, is to improve the recovery process. Though NEAR improves greedy routing significantly, reaching a concave node is sometimes unavoidable. However, an immediate effect of the repositioning is that every peninsula in the network is elevated, and a smooth edge is surrounding the routing void. We use an additional distributed void identification algorithm to help us identify the voids. Once a void is identified, its smooth edge can be easily followed with a minimal number of hops and without the entanglement that plagues some of the other recovery algorithms.

Definition 3.4. *A perimeter of the void region, which is free from peninsulas and other obstacles that may require backtracking or extending the routing path is a **smooth edge**.*

We thus present as a solution, the Node Elevation Ad Hoc Routing (NEAR) algorithm, which is comprised of several algorithms that feed each other and are all distributed. At network start-up, we run the repositioning algorithm. This algorithm is distributed and local and is executed periodically at low cost due to its local nature. We also execute a void identification algorithm, which is performed around

the void. This algorithm is distributed as well, but it is executed by all the nodes at the void edge, i.e., the perimeter of the void and possibly their neighbors. We thus define it to be a regional algorithm, and its termination time depends on the number of nodes on the void perimeter.

Definition 3.5. ***Termination time*** *is the number of iterations required to complete a distributed algorithm in units of nonsimultaneous transmitted messages, without any external changes occurring in parallel.*

Once a void is identified, the maintenance of its identified perimeter is purely local. The output of these two algorithms is used by the routing algorithm, which is a variant of previously published greedy routing algorithm, and by an efficient recovery algorithm. In the next sections, we will describe our algorithms and their performance.

3.1 Repositioning Algorithm

The node reposition algorithm is executed periodically by every node. The repositioning calculation is done locally, based on the node's neighbor positions. If neighboring nodes remain static, no repositioning is required. Otherwise, the node checks its neighbor disposition to see whether there is any direction in which it is concave. The decision is based on a threshold angle, α, which is essentially larger than 180 degrees.

Definition 3.6. *Node v has two adjacent neighbors, u and w. Given a* ***threshold angle****, α: If $\angle uvw > \alpha$, then the sector covered by $\angle uvw$ is concave from the perspective of node v.*

If the above condition holds, the node recalculates its position and updates its $n+1\text{th}$ dimension location. In addition, the void update algorithm is executed, which may result in the removal of the node from the void edge and the reconnection of its neighbors.

3.1.1 Detailed Algorithm Description

The repositioning algorithm is called upon the arrival of a coordinate update from a neighboring node. These updates are periodically sent (e.g., through a Hello protocol) in an asynchronous manner by every node to its neighbors. The rate of the updates depends on the nature of the network (handheld devices, vehicular, etc.) and the transmission radius. When a coordinates update is received from a node w, the algorithm compares it with the previous known position of this node. Only if the change in the position is not negligible, the repositioning algorithm continues. The purpose of this condition is to save resources (mainly computations) and possibly reduce temporal effects; however, it is not mandatory. The repositioning algorithm is also initiated at the node due to a significant change in its own position or if a connection with a neighbor node is lost (e.g., using time-out from the last Hello message).

Definition 3.7. *The angle between a node v and its neighbor node w is determined as the arctangent of $(w_y - v_y)/(w_x - v_x)$, in the range of $-\pi$ to π radians.*

The algorithm maintains an array of its neighbors' position, sorted by angle. As a node w moves, its place in the array is being updated to reflect its new relative position. Next, the algorithm finds the maximal angle in the sorted array between two adjacent nodes and compare it to the threshold angle α. If it is greater than α, the node is concave in the direction of the segment defined by the angle. We later prove (Lemma 3.4.) that α must be greater than π for repositioning to work. However, setting $\alpha > \pi$ also means that the greedy algorithm will fail to select the next hop if a message is sent to a destination that falls within the sector defined by the angle. The reason is that any decision will take the message further from the destination than its current location (We disregard here a minority of greedy algorithms that allow such a decision). When looking for the maximal angle in the array, we consider only nodes with height (Z dimension) zero. Nodes whose height is nonzero are indicated to already be concave in the wide sense. If the node v finds itself to be concave, due to the calculation above or if it has less than two nonconcave neighbors, it elevates.

The elevation is being carried out by a repositioning calculation. If the elevation is due to the maximal angle between two adjacent nodes with zero height, then its new (x, y) position will be in the middle of the segment connecting these two, and its height will be one above them unless elevation is forbidden (e.g., the maximal node height, Z_{max}, is set to zero). If all the node's neighbors are floating, then the new node's position will be the average position of all nodes with minimal height, and its new height will be one unit above them or the maximal allowed node height. We refer in this case to all minimal height nodes, as the elevation process made the angle criterion irrelevant and as it has given best elevation results. The selection of a simple average of nodes position here leads to proper role up of concave areas and creates a smooth edge to the void. Other methods, such as choosing the projection on the axis between the two neighbors causes the concave area not to elevate to a smooth shape and causes a difficulty when the destination node has an elevation height of more than one. If the change between current and new recalculated position of the node is not negligible, it is stored. The result of this reposition calculation is that a node is always at most one unit above his minimal-height neighbor. An equivalent statement is that the nodes height is equal to its concavity degree (see Lemma 3.3).

Another reason for a node to change its virtual position is if it becomes nonconcave, meaning the maximal angle between two adjacent nonfloating neighbors of it is below the threshold angle. In such a case, the virtual position of the node will be set to be its real (physical) coordinates.

If the node changed its virtual position due to the reposition algorithm, it sends an update to all its neighbors with its new position. The reposition algorithm also calls for the *void_update* algorithm, which will be described in Section 3.2.

3.1.2 Formal Algorithm Description

Fig. 3 gives a formal description of the node reposition algorithm. We assume, for simplicity, that nodes have two real dimensions, X and Y, and one virtual dimension, Z. A node v maintains a sorted array, CV, of the coordinates of its neighbor nodes, $\mathcal{N}_v$. We denote by $V = (v_x, v_y, v_z)$ the stored virtual coordinates of node v and by P its stored real coordinates. A dotted variable, e.g., $\dot{P}$ indicates the up to

Algorithm Node Reposition

For a new vector $\dot{W}$ from node w
1. if $(|W - \dot{W}| > \varepsilon)$ or $(|P - \dot{P}| > \varepsilon)$:
2. $w \leftarrow \dot{w}$
3. $P \leftarrow \dot{P}$
4. $(s,t) \leftarrow maxangle(resort(CV))$
5. if $(t = NULL)||(\angle s - \angle t > \alpha)||(s_z > 0)$
6. $chg \leftarrow calc(s,t)$
7. else
8. if $(v_z == 0)$
9. $chg \leftarrow false$
10. else
11. $chg \leftarrow true$
12. $V \leftarrow P$
13. $void\_update(\mathcal{VN}_v, w)$
14. if chg
15. Transmit new vector $\dot{V}$ of v

For loosing connection to neighbor w
16. $\mathcal{N}_v \leftarrow \mathcal{N}_v \setminus \{w\}$
17. $(s,t) \leftarrow maxangle(CV)$
18. if $(t = NULL)||(\angle s - \angle t > \alpha)||(s_z > 0)$
19. $chg \leftarrow calc(s,t)$
20. else
21. $chg \leftarrow false$
22. $void\_update(\mathcal{VN}_v, w)$
23. if chg
24. Transmit new vector $\dot{V}$ of v

Fig. 3. Repositioning algorithm for node v.

date real coordinates of node v. In a similar manner, $\dot{W}$ indicates a value, which has just been accepted in a message (as opposed to the stored value W). The array CV is sorted by the angle between v and the neighbors. $\mathcal{VN}_v$ denotes the list of void ids that v is at most one hop away from.

Each node periodically receives coordinate updates (e.g., through a Hello protocol) and is aware of new neighbors or a breaking of a connection. It is also aware of significant changes in its own position or in the position of a current neighbor (in lines 1 and 15).

The node looks for a change in a neighbor disposition using $maxangle$ (lines 4 and 17), which returns the two adjacent nodes having the largest angle between them. The two nodes must be either nonfloating or below v_z. In case a concave sector is found, the node recalculates its new virtual coordinates by calling $calc$ (in lines 6 and 19) or if it is no longer concave (line 10) by setting its virtual coordinates to be the physical ones, and immediately updates its neighbors (lines 15 and 24). In addition, it maintains the void edge by calling *void_update* (lines 13 and 22, as will be explained later). Note that node repositioning may either elevate a node or pull it down.

Fig. 4 gives a formal description of the node repositioning calculation. When called, this algorithm calculates the new virtual position of a node. The algorithm receives pointers to two adjacent neighbor nodes, s and t. If t is not null, and the nodes are nonfloating (line 1), the node will get the average position of s and t (line 2) and v_z will be set to one (line 3). Otherwise, the node virtual position will be set to the average of all nodes with the same height as s (line 5). The height of the node will then be set to one above

Algorithm Reposition calculation

$calc(s,t)$
1. if $(s_z = 0)$ and $(t \neq NULL)$
2. $\dot{v}_{x,y} \leftarrow avg(s_{x,y}, t_{x,y})$
3. $\dot{v}_z \leftarrow s_z + 1$
4. else
5. $\dot{v}_{x,y} \leftarrow avg(\forall u_{x,y} : u_z = s_z)$
6. $\dot{v}_z \leftarrow min(s_z, Z_{max} - 1) + 1$
7. if $(|V - \dot{V}| < \varepsilon)$ and $(Z_{max} > 0)$
8. $V \leftarrow \dot{V}$
9. return(true)
10. else
11. return(false)

Fig. 4. Reposition calculation.

the height of its lowest neighbor unless reaching Z_{max} (line 6). If the change of the node's virtual position is not negligible, and elevation is not forbidden $(Z_{max} = 0)$ the new position will be stored (line 7).

3.1.3 Additional Repositioning Aspects

Lemma 3.1. ***Repositioning.*** *A node changes its virtual position only due to a change in its immediate neighbor virtual position or its own position.*

Proof. Elementary from the algorithm. □

Fact 3.2. ***Distributed.*** *The algorithm is distributed and local.*

Proof. The algorithm is distributed—elementary. The algorithm is local:

- Each node is aware only of its own position, and its neighbors positions.
- All the decisions taken are based on this information alone and do not require querying nor storing of more information.
- Any update of a node's position is sent only to its immediate neighbors and is not relayed further in the network.
- Any change in the network is reflected to the node only through its neighbors position. □

Since the positioning update is done periodically and depends on the virtual position of the neighbor nodes, the nodes adapt their position to changes in the network and, more importantly, reflect the position of other nodes as well. Assume there is a piece of the network shaped as a peninsula, with the destination placed opposite of the area across a void (see Fig. 5a). Though only a small number of the nodes in this area may have first degree concavity in the peninsula direction by Definition 2.1 (depending on placement and type of routing), a routed packet should still avoid entering the peninsula area in order to prevent the routing algorithm from entering recovery mode. Our node repositioning will cause the peninsula to role up in the $n+1$th dimension and create a smooth edge around the void. Assuming two real dimensions, the further the node is into the tongue of land, the higher it is. This obviously leads our routing algorithm to prefer ground-height nodes over floating ones and lower nodes over higher ones (within a margin of error).

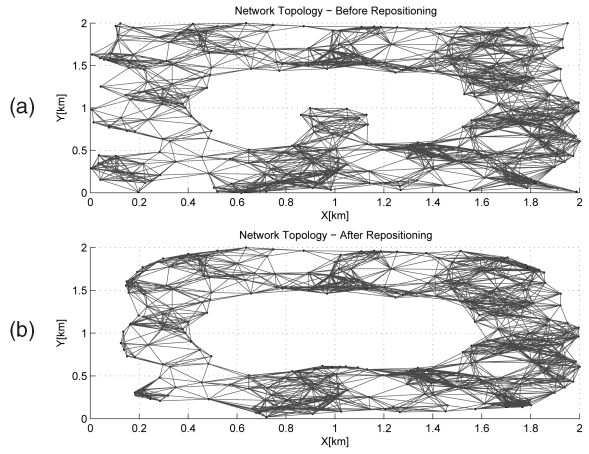

Fig. 5. An example network topology (a) before and (b) after repositioning.

Lemma 3.3. ***Elevation height.*** *The maximal elevation height of a node is N, where N is the number of hops to the closest node with concavity degree of zero (where $N < Z_{max}$).*

Proof. By simple induction, stemming from the fact that a node height is one above the direct neighbor with the lowest height. □

The "height" convention that we use here should not be confused with the "height" convention of the TORA algorithm [32], which refers to a completely different concept.

Fig. 5 demonstrates the repositioning effect for a sample network.

Lemma 3.4. ***Setting threshold angle.*** *The minimal threshold angle must be greater than π.*

Proof omitted due to space limitation and can be found in [33].

3.2 Routing Algorithm

Before the source emits a packet, NEAR uses a location service such as those described in the introduction to obtain the destination position.

The use of virtually repositioned nodes does not contradict the use of standard greedy routing algorithms. On the contrary, greedy routing is a basic element of NEAR, and the network virtual coordinates are intended to increase the efficiency of greedy routing. However, our routing algorithm cannot be purely greedy due to two reasons. First, as with previous work in this area, we may reach a concave node and may need to switch to recovery mode (though it is significantly infrequent in our case). In addition, if either the source node and/or the destination node are floating, we will not use greedy routing in the start phase or the final phase of the routing process, respectively. While in greedy routing, there is also one additional rule for the next node selection: one may use only nonfloating nodes. This way, we avoid the concave areas until we reach the destination or its vicinity.

Even though NEAR significantly reduces the need for recovery state, it cannot completely eliminate it. NEAR uses predefined paths, called Void bypass, as a mean to recover from dead ends.

Definition 3.8. ***Void bypass*** *is a path traversing the void edge, connecting every two nodes on the void's edge.*

3.2.1 Detailed Void Bypass Description

The void bypass discovery algorithm is called as part of the network start-up, after the node reposition algorithm was executed. The purpose of the discovery algorithm is to a priori identify routing obstacles and find recovery paths around them using perimeter routing. The knowledge obtained during the repositioning is used to identify voids: if a nonfloating node v has an angle between any two adjacent neighbor nodes (denoted by s and t), which is greater than β ($\pi \leq \beta < \alpha$, usually $\beta = \pi$), then it is concave in the direction of the sector defined by this angle. Though we know this node is concave, it is not elevated to avoid excessive node elevation (see Section 4). Next, the node selects a random id for the void (32-bit ids make the probability of misidentification negligible). The id is used to distinguish between different voids, as a single node may participate in two different void bypass routes. Node v transmits a void discovery message to one of the other neighbors on the void's edge, either s or t. We deterministically select the clockwise neighbor, say, t. Node t, in its turn, will forward the void discovery message to its neighbor node adjacent to v in the sorted neighbors array (and in the discovery routing direction) and so forth until the discovery message will traverse the entire face of the void and return to its origin, v. A simplified explanation of the next node selection is by a no-crossing heuristic using the right hand rule. Though this selection may succeed in over 99.5 percent of the times [34], we can use additional rules such as information about the 2-neighborhood to further improve the process success probability . Every node in the network maintains an array of the void bypasses in which it takes part. For every void bypass, it keeps its id, the next hop for routing in a clockwise direction, and the next hop for routing in a counterclockwise direction.

During the discovery process, several special scenarios may occur. Most common is when several nodes may initiate a void discovery process for the same void. This can be easily identified, and one of the void ids is selected. Due to space constrains the full discussion on this and some other rare problems is removed and can be found in [33].

3.2.2 Detailed Routing Description

The routing algorithm uses two routing modes: greedy routing and perimeter routing, where the latter is our enhanced equivalent to recovery routing. In the greedy routing mode, almost any type of position-based greedy algorithm can be used, with NEAR being oblivious to the greediness criteria. For perimeter routing, NEAR's predefined void bypass routes are used. A message routing starts in greedy routing mode. In this mode, the greedy algorithm uses the nodes' virtual coordinates, and NEAR does not intervene in the greedy algorithm decisions, except for preventing it from forwarding to elevated nodes. In addition, there are two additional exceptions. The first is the case of a floating message's source node, where we allow the greedy algorithm to route through floating nodes, as long as every hop is one

height unit lower than the previous one until a nonfloating node is reached. This scenario is called descending.

Definition 3.9. ***Descending (Climbing)*** *means transmitting a packet from node v to node w, where $v_z > w_z$ ($v_z < w_z$).*

In a second scenario, the destination node is floating. Here, the greedy algorithm must be allowed to route through floating nodes in order to reach the destination. We refer to this scenario as climbing to the destination. It should be noted that when we reach the very close vicinity of the destination, we sometimes allow the greedy algorithm to use the nodes' real coordinates, since floating neighbor nodes may share similar virtual coordinates.

In the rare cases where we reach a concave node, we abandon greedy routing for perimeter mode. In this mode, the node first searches in its void bypass array for a void that is in the same direction as the message's destination. This node may be on the edge of this void or the void may be one-hop away. If no matching void is found, a void discovery process is initiated and a void bypass is created. Our simulations have shown that this situation occurs on the average only once per every 62,000 messages through concave nodes, and their average traversal length was less than 10 hops, which is very short compared to the average void bypass length. In more than 80 percent of the simulated networks all the voids were detected during the initialization process. The success rate of void recognition during the initialization and maintenance process can grow higher if we are ready to use a better and more complicated discovery algorithm, but we feel that we struck the right balance.

The direction to traverse the face of the void, clockwise or counterclockwise, is selected when switching to perimeter mode based on the minimal angle between cw or ccw next hop and the destination. The message will then follow the void bypass route until it reaches a node that is closer (in the criterion of the greedy algorithm) to the destination than the node that switched to perimeter mode. From this point on, greedy routing will continue until another concave node or the destination is reached.

3.2.3 Formal Routing Description

A pseudocode of the routing algorithm is given in Fig. 6. The algorithm receives as input six parameters carried by the message: the destination node's virtual and real coordinates, D and D^p, respectively; the current routing mode (greedy or perimeter), $\mathcal{M}$; the starting point of current perimeter routing, V_{per}; the current void bypass direction (cw or ccw), dir; and the current void bypassed id, id. The last three parameters are nonnull only when appropriate. The node maintains several internal variables: P and V are the node real and virtual coordinates, respectively; Z_{max} is the maximum allowed height for routing; n stores the next node to route to (and N is its coordinates vector); and $\mathcal{VN}_v$ is the group of voids v participates in.

The routing algorithm first checks for two special cases described above, climbing to the destination and descending from the source. The condition for climbing to the destination is (line 1) that both our virtual and physical coordinates are close enough to the destination.

The virtual coordinates should be within less than the transmission radius, and the physical coordinates should be within the destination height times the transmission radius. These parameters ensure that the message is within the proximity of the destination.

Algorithm Message Routing $(D, D^p, \mathcal{M}, V_{per}, dir, id)$

For a new message M reaching node v

1. if $(|P - D^p| < \varepsilon_{phys})$ and $(|V - D| < \varepsilon_{virt})$
2. $\quad Z_{max} \leftarrow Z_D$
3. else if $(v_z > 0)$
4. $\quad Z_{max} \leftarrow v_z - 1$
5. else
6. $\quad Z_{max} \leftarrow 0$
7. if $(\mathcal{M} = greedy)$
8. $\quad n \leftarrow GreedyAlg(D, Z_{max})$
9. $\quad$ if $(n = NULL)$
10. $\quad\quad \mathcal{M} \leftarrow perimeter$
11. $\quad\quad$ if $\mathcal{VN}_v \neq \emptyset$
12. $\quad\quad\quad$ choose $id \in \mathcal{VN}_v$:
13. $\quad\quad\quad (\angle\mathcal{VN}_{id}.ccw \leq \angle D \leq \angle\mathcal{VN}_{id}.cw)$
14. $\quad\quad\quad dir \leftarrow minangle(dest, \mathcal{VN}_{id})$
15. $\quad\quad\quad V_{per} \leftarrow V$
16. $\quad\quad\quad n \leftarrow \mathcal{VN}_{id}.dir$
17. $\quad\quad$ else
18. $\quad\quad\quad n \leftarrow min_{w \in CV, W_z \leq Z_{max}}\{|\angle W - \angle D|\}$
19. $\quad\quad\quad V_{per} \leftarrow V$
20. $\quad\quad\quad id \leftarrow NULL$
21. $\quad\quad\quad dir \leftarrow minangle(dest, \mathcal{VN}_{id})$
22. else /* perimeter mode */
23. $\quad$ if $(id \neq NULL)$
24. $\quad\quad n \leftarrow \mathcal{VN}_{id}.dir$
25. $\quad$ else
26. $\quad\quad$ if $\mathcal{VN}_n \neq \emptyset$
27. $\quad\quad\quad$ choose $id \in \mathcal{VN}_n$:
28. $\quad\quad\quad (\angle\mathcal{VN}_{id}.ccw \leq \angle D \leq \angle\mathcal{VN}_{id}.cw)$
29. $\quad\quad$ else
30. $\quad\quad\quad id \leftarrow init\_void\_discovery(\angle D)$
31. $\quad\quad n \leftarrow \mathcal{VN}_{id}.dir$
32. $\quad$ if $|N - D| < |V_{per} - D|$
33. $\quad\quad \mathcal{M} \leftarrow greedy$
34. Send message $M(D, \mathcal{M}, V_{per}, dir, id)$ to n

Fig. 6. Message routing algorithm.

This is the only part of the algorithm that requires knowledge of the original physical coordinates, since in case more than a single tongue rolls to the same virtual location, we need to be able to identify, which is the one we would like to climb. By setting the maximal floating node height that may be accessed, Z_{max}, appropriately, we force the greedy routing to climb or descend. When the source node is floating (line 3), Z_{max} will always be set lower than the source node height in order to force descent toward ground level and will be updated at every hop until a nonfloating node is reached (line 5). When a message has to reach a floating destination, Z_{max} is set to the destination height (line 2).

If neither of the two conditions above is true we are routing the message only through nonfloating nodes, and we can be in either greedy or perimeter routing mode. Thus, the algorithm mostly spends time in line 7 where it selects the next greedy node toward the destination. If the greedy algorithm fails to find a next node, we switch to perimeter mode (line 10). Here, the node is either part of the void edge (line 11) or, since we are not using plannerization like other solutions [28], [27], have a neighbor who is part of the void

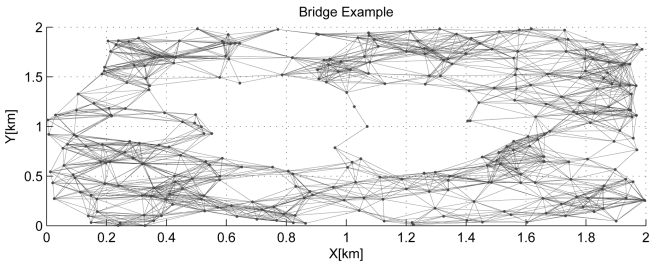

Fig. 7. A bridge example.

edge (line 17). In either case, we choose the best (based on the local angle) of the two directions around the void. While in perimeter mode, we check if we can return to greedy mode on every hop. The condition is simply that the current node is closer to the destination than the node where we entered perimeter mode (line 32).

Note that the void update algorithm maintains a cycle around a void, which guarantees that voids are bypassed successfully with a minimal number of hops. In rare cases, we reach a concave node where a void bypass cycle is not defined. In this case, we initiate a void discovery algorithm (line 30), which is identical to the one used at network start-up.

3.3 Bridging

An interesting special structure in the virtual network is the *bridge*.

Definition 3.10. *A **bridge** is a series of floating nodes connecting two sides of a void.*

An example of a bridge is shown in Fig. 7. Bridge creation is fairly rare, since the threshold angle we use is usually higher than 210 degrees, e.g., if the two bridge nodes in Fig. 7 would be closer in the X coordinate, they would not float, and we will have two adjacent voids without a bridge. Though NEAR usually forbids routing through floating nodes, the case of a bridge is different, as it specifically may affect not just the efficiency, but also the guarantee of delivery, e.g., when a river divides the network to two sections connected by real physical bridges. Unlike standard floating nodes, which do not take part in void traversing paths, bridge nodes are not only part of the void bypass, but also divide a single void to two parts, one on each side of the bridge, with different void ids. This means that the system is stable with respect to slight node repositioning since nodes in bridges act in the same manner as nonfloating nodes. Identifying a bridge is simple, since the floating nodes X-Y coordinates are not close to the coordinates of the neighboring nonfloating nodes. Thus, using the bridges NEAR efficiency is not damaged by repositioning.

Definition 3.11. *A **bridge head** is the highest repositioned node, which is part of a bridge, and is the one to initiate void traversing process. A bridge head has at least 2-neighbor nodes.*

A bridge head is identified by the following conditions. First, the distance between the real and virtual coordinates should be smaller than some threshold δ_{TH}. In addition, the node height should be at least 2 and not lower than the height of any of its neighbors. If two nodes are at the same height and obey the other conditions to become a bridge head, one is selected arbitrarily, based on the nodes' id.

Lemma 3.5. ***Bridge head.*** *In every bridge, a single bridge head is selected.*

The proof is omitted due to space limitations and can be found in [33].

4 The NEAR Algorithm Characteristics

As locality is a major issue in geographic ad hoc networks [15], [5], the NEAR solution maximizes the use of local information while still keeping a notion of the network topography and behavior thanks to the repositioning information. A node is aware only of its own location as well as its neighbors' but still know that in a certain direction, it is concave based on its neighbors height or the void routing record it keeps. Certain routing algorithms cache routing information of previous packets. While we can certainly do this, our algorithm benefits only marginally from route caching, since our routes are very good without it, and our routing algorithm does not pay overhead per packet.

The algorithm locality is expressed in several ways. The first one is messaging: there are two types of messages required by NEAR, update messages from neighbors and void maintenance. The update messages include the node's virtual and real position. In most cases, the virtual coordinates are sufficient, however, for routing between floating nodes for more than a single hop, knowledge of the real neighbor physical place is required.

The void establishment messages are sent only around voids, thus their overhead is linearly proportional with the number of nodes around voids. While several nodes may start the void discovery process for the same void by sending void discovery messages, a node that already received such a message once is able to detect the duplicity. If this happens, it can either drop the new message, while updating backwards the already existing void id or forward it to replace the previous id (similar to leader election algorithms on ring networks). The shape of the void may change over time and nodes that were part of the void bypassing route may be replaced by others. Every replacement is done locally by updating the records of the nodes that are placed in or out of the bypass route and their immediate neighbors on the route. Locality is also expressed in the amount of memory required for the algorithm implementation. A node is required to store its neighbors real and virtual coordinates. Nodes that participate in the void bypass routes also store their immediate neighbors on the route (and not the entire bypass route).

While NEAR can perform well in any area, it is somewhat sensitive to the threshold angle in networks that are rather sparse. To improve its performance in sparse networks we can mark certain nodes or locations as grounded station.

Definition 4.1. *A **grounded station** is a special node whose virtual coordinates are always the same as its physical coordinates.*

This is beneficial, for example, in network corners, say at the geographic border of the network coverage due to law,

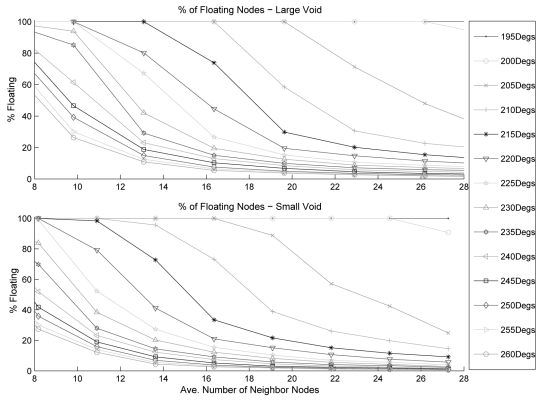

Fig. 8. The network density effect on repositioning.

regulation, or physical obstacle. The nonfloating nodes will prevent unnecessary flotation of nodes in their surrounding. Another solution is to adapt the angle threshold according to the network density: increase it in sparse networks and decrease it in dense ones. Failure to use the right threshold angle in a sparse network (without grounded stations) may cause the network to wrap-up.

Definition 4.2. *A **System wrap-up** happens if all the nodes in the system are floating.*

However, it is important to note that when a network becomes too sparse the risk of losing connectivity is high, thus we can expect ad hoc networks to be fairly dense, and pose no problem for the repositioning stability, as indicated in Section 5, Fig. 8.

A known problem in recovery mode is that traversing the graph based on the right-hand rule alone does not guarantee tracing the boundary of a closed polygon, a problem that is caused by crossing edges of the graph. A study of this problem was done by Karp for Greedy Perimeter Probing (GPP) [34]. GPP, which is based on the right-hand rule with the no-crossing heuristic rule, proved to succeed for over 99.5 percent of the routes but not to guarantee delivery. Our maintenance of void bypassing cycles annihilate the need for planarization. The fact that the regional cycle creation algorithm is executed only once per each void (afterwards, maintenance is local) permits the usage of algorithms with larger overhead, e.g., algorithms that are based on the node's 2-neighborhood, allow message backtracking, and combine void creation messages from several sources to a single void. A key idea in the void initialization algorithm is that the originating node that started the process can detect its failure when the packet fails to return from its counter side, thus initiating a bypass fixing process. On top of the above, it should be remembered that the repositioning process dictates a smooth shape of the void, therefore eliminating most of the obstacles that otherwise exist in other routing solutions.

A packet reaching an unmapped void can either wait for the void traversing process to complete, or it may be routed in the meanwhile using the right-hand rule with high chances of success. The new void bypass route discovery process will be executed independently.

NEAR does not guarantee the most efficient routing when bypassing voids. When a message reaches a node on the perimeter of a void and the routing switches from greedy to perimeter mode, the void traversal direction is determined by the angle of the current node to the destination: choosing the direction which seems to minimize it. This does not guarantee optimality, yet based on local information alone, a wiser decision may not be made. Another inefficiency stems from floating destinations. As NEAR forbids routing through floating nodes, a decision to climb through floating nodes toward the destination is taken by a proximity rule. The threshold proximity is based on the physical and virtual distance from the destination, and it does not necessarily become true at the optimal node, therefore sometimes increasing the route by several hops.

5 Simulation

5.1 Simulation Environment

The NEAR solution properties and characteristics have been simulated for the following aspects: Repositioning, Mobility Effect, Greedy Routing, and Recovery Routing. For this end, we randomly placed nodes with transmission radius of 250 m (as in IEEE 802.11 WaveLan) in a 2 Km $\times$ 2 Km square with variable network density. We discuss two simulation scenarios. The first scenario, termed **small void**, examines the algorithm with a randomly placed void, whose size does not exceed 10 percent of the network size. The second simulation, termed **dominant void**, examines the case of a void placed in the middle of the network and which covers approximately 25 percent of the network size. In addition, a tongue of land is entering the dominant void (see Fig. 5). Thus, The first simulation scenario is intended to prove the concept in simple conditions, while the second scenario stress tests the algorithm main goal of void bypass over most of the routing paths. GEDIR [16] is the greedy algorithm used in the simulations. Each result is an average of more than 20 different network distributions. Since our routing algorithm is based on a greedy routing algorithm for standard routing, comparing performance with a uniformly randomly selected node pairs obscures the focal point of the work, bypassing obstacles. Thus, in the routing section, we filter out all the paths that use only greedy routing.

5.2 Repositioning Simulations

5.2.1 Network Density and α Threshold

The repositioning depends on two important factors: The network density and the threshold angle, α. Fig. 8 shows the effect of the network density on the percentage of repositioned nodes, as a function of α. The graph shows the results of calculations performed on **dominant void** scenario (like the instance in Fig. 5) and on **small void** scenario. The first phenomenon that should be pointed from the graph is that the repositioning is effective in networks with 13 neighbors or more per node. When the network is sparser than that, the percentage of floating nodes is high. Below this density, the ad hoc network becomes less effective, regardless of the control algorithm in use, since a meaningful percentage of the nodes become disconnected from the largest connected component, as demonstrated in Fig. 9. Note that the density in our discussion always refers only to the populated node area, without voids, so the overall density is lower. The optimal density in a mobile

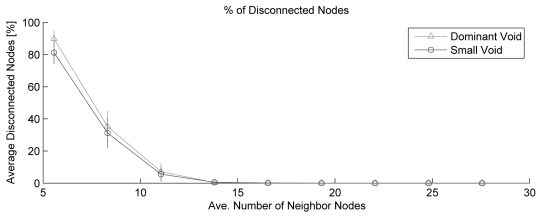

Fig. 9. The percentage of disconnected nodes versus nodes density.

ad hoc networks was found by previous papers [35], [12] to be 14 neighbors per node or more. Many studies in this area [28], [18] were also performed on random ad hoc networks which were at least this dense. We can allow sparser networks (and risk disconnections) by manually marking some areas or nodes as grounded (say, at the edge of the network coverage area), these nodes will halt the folding process without hurting routing performance.

The second factor is the threshold angle, α. A minimal angle of 180 degrees is simply too low, and almost all nodes will float. There is a clear threshold around $\alpha = 220$ degrees above which the system is stable, with relatively enough routable nodes. Though it seems that **large void** scenarios require threshold angle, which is 5 degrees-10 degrees higher than **small void** scenarios to get the same percentage of floating nodes, it should be recalled that a higher amount of nodes is expected to be elevated in the **large void** scenario by design. $\alpha = 225$-230 degrees was found to be best for the various scenarios. Using a wider angle threshold achieves less repositioned nodes, and completely misses the purpose of the repositioning algorithm: concave nodes might not be repositioned and the faces of voids might not be smooth enough to allow efficient routing.

5.2.2 Mobility Effect on Stability

Ad hoc networks are dynamic by nature, with node mobility being their main feature. Thus, it is important to verify that the NEAR solution can cope with the network dynamics and maintain system stability. Keeping the amount of elevated nodes at a constant level and quick positioning adaptation to node movement is an utmost concern.

We have conducted a simulation of node movement and checked the effect on nodes positioning, as well as other consequences. We used the simulation model described above and added movement elements quite similar to those described in [36] and [37]: every node in the system is assigned a random direction and speed to which it advances. Nodes are restricted from entering the forbidden zones that define the voids, as well as crossing the system boundaries. The velocity of a node is uniformly distributed (0, 25 m/s). This models vehicular mobile nodes moving at a speed of up to 90 Km/h with updating messages every second, or a person with a mobile device walking at 4.5 Km/h with messaging updates every 20 seconds.

The system stabilizes very quickly after repositioning. It requires less than 10 messaging cycles for the large void scenario and less than five for the small void scenario (Fig. 10a). For the majority of the nodes in the network, stabilization occurs much faster. This can be seen in Fig. 11, which shows the percentage of nodes that require i iterations to stabilize (i ranging from 0 to 10) according to the network's density. The tail of the distribution with

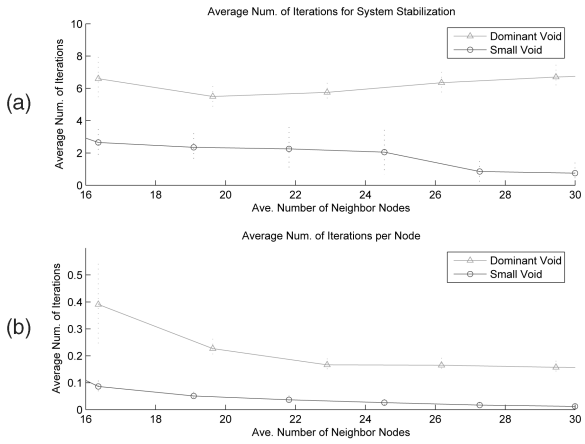

Fig. 10. Iterations required for the entire system stabilization and per single node.

nodes that require more than four iterations to converge contains less than 1 percent for the large void scenario and less than 0.1 percent for the small void scenario. The exception in very sparse networks with an average of 16 neighbors per node when the tail contains 2.25 percent and 1 percent for the large and small void scenarios, respectively. Fig. 10b gives us an indication of the low overhead of elevation messaging. The overhead is comprised of one hello message to every neighbor node and additional repositioning update messages for every local node iteration. As Fig. 10 shows, this means an average of less than 0.4 repositioning update messages per node in the worst case of a dominant void and a sparse network. It is expected that when a system has a dominant void, more repositioning update iterations will be required as the correction made by repositioning is greater. The contribution to the overall messaging in the network is therefore $n \cdot i_n \cdot N$, where n is the average number of neighbors per node, i_n the number of repositioning iterations per node, and N the number of nodes in the system. Based on our results, we can set a bound on the number of update messages in the network by $(1 + i_n) \cdot n \cdot N < 1.4 \cdot n \cdot N$.

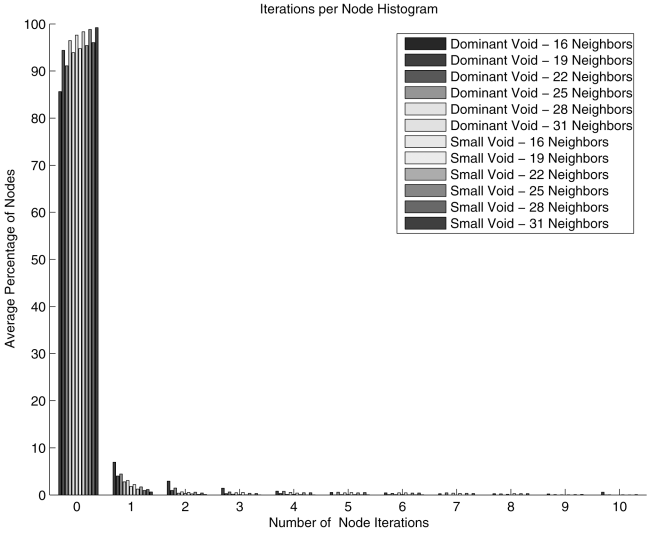

Fig. 11. Histogram of node's iterations by network density.

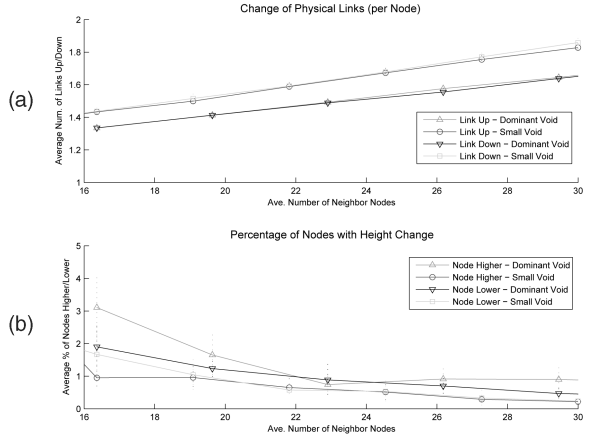

Fig. 12. Change of physical links per node and percentage of nodes with height change.

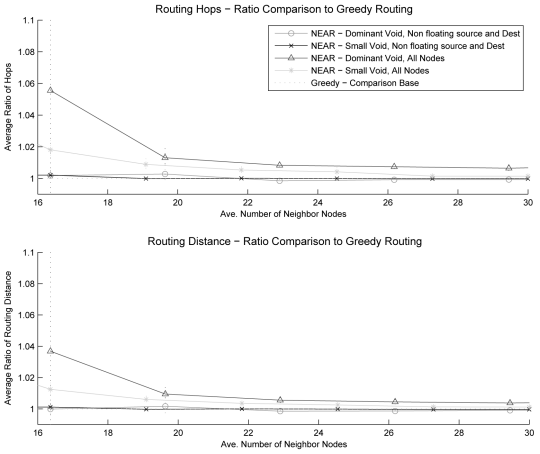

Fig. 14. Comparison to greedy routing by hops and distance ratio.

Fig. 12a shows the average number of physical links up or down between every update session per node. As the change is not large and the number of broken links is quite the same as the new connections made, the system's ability to stabilize grows. Fig. 12b depicts the percentage of nodes in the network that change their height as a result of the movement. Even in the most challenging scenario, a sparse network with a dominant void, less than 3.5 percent of the nodes are affected, and the number of nodes elevated up or pulled down is approximately the same (especially considering the confidence levels), the system ability to stabilize and quickly adjust to nodes movement using NEAR is evident.

In our movement simulation, we have also studied the case of losing a dominant neighbor or gaining a new one. The simulation results are consistent with results above, i.e., the system stabilizes quickly with minimal number of iterations per node.

5.3 Routing Simulations

5.3.1 NEAR Effect on Greedy Routing

The goal of NEAR is to improve the performance of greedy routing and minimize the usage of recovery algorithms. One way to achieve that is to reduce the number of concave nodes in the network. In Fig. 13, we plot the percentage of concave nodes that were eliminated from the routing because of the repositioning. When the network density is sufficiently high, 62 to 100 nodes per Km^2 (12-20 neighbors per node), up to 45 percent of the routes that previously reached concave nodes, now complete their routing procedure using only greedy routing.

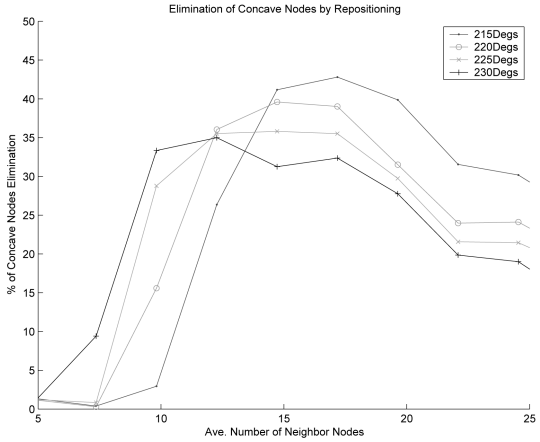

Fig. 13. Elimination of concave nodes by repositioning.

An additional concern is to make sure that nodes repositioning does not harm greedy routing.

Fig. 14 shows the effectiveness of NEAR compared to the original greedy algorithm path. When both source and destination nodes are not floating, the routes taken are almost identical—on the average, there is less than 0.5 percent difference in the number of hops and distance between pure greedy routing and routing using NEAR. This translates to no more than 2 hops difference in the worst case due to the change in the network topography. When floating source or destination nodes, placed in a concave areas, are being taken into account as well, there may be a variance of up to 5 percent hops (an average of half additional hop) and 3 percent distance in sparse networks, and less than 1 percent hops and length in more dense networks. The reason for this is the different routing rules applied in concave areas. In all the cases described above, routing success is guaranteed. In several cases, NEAR even outperformed the greedy algorithm, as the network topography was changed by the repositioning.

5.3.2 Comparison with the Shortest Path, GOAFR, and GPSR

A measure of the algorithm efficiency in routing and bypassing obstacles is obtained by a comparison to the shortest path routing results.

Fig. 15a shows a comparison of the ratio between the route length in hops of the two algorithms, NEAR and GPSR [28] routing algorithm, to the shortest path calculated by a central algorithm. We refer here to hops ratio and not to the number of hops, as the effectiveness of the routing should reflect the performance compared to the best path possible. Both the dominant and the small void scenarios were used for comparison. Since both NEAR and GPSR center on the recovery process and share greedy algorithms with the same performance for the rest of the time, we examine here only routes with concave nodes, where the greedy routing alone fails. We count the number of hops

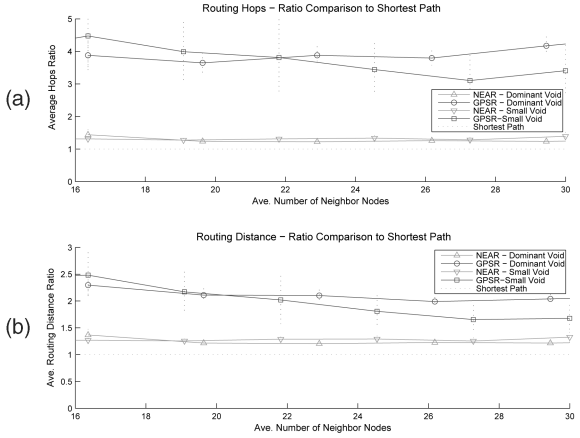

Fig. 15. Comparison between NEAR, GPSR, and the shortest path by distance and hops ratio.

from source to destination and not only from the concave node, as NEAR often does not reach a concave node at all thanks to the repositioning. With the shortest path being the optimal route, it out-performs NEAR by up to 40 percent at most, which translates to less than two hops on the average. GPSR average number of hops is 3 to 4.5 times the number achieved by shortest path, and 2.5 to 3.5 times more than NEAR. Fig. 15b compares between the algorithms, under the same scenarios, but with a criterion of physical routing length ratio. Here, the shortest path is better than NEAR by up to 35 percent at most and by 80 percent to 140 percent than GPSR. It should be noted that while many of the GPSR recovery routes are short almost as NEAR, some are very long—which is manifested in its large confidence intervals.

Fig. 16 shows the NEAR advantage over GPSR in yet another important factor, the failure rate, tested under the same scenarios as above. GPSR average delivery rate is 97 percent to 99 percent in the small void scenario and 94 percent to 96 percent in a dominant void scenario, whereas NEAR delivers over 99 percent of the packets in both scenarios, except for 94 percent delivery rate in a sparse network with a dominant void. The GPSR recovery algorithm suffers mostly in our simulation from routing toward the edges of the network. As Karp indicates in his dissertation [34], GPSR realizes a node is disconnected if a packet traverses the first edge it took on a certain face for the second time. Due to the topography of the network, this sometimes happen in nodes on the border of the simulated area even when they are not disconnected. As previous works [34], [28] averaged the GPSR performance overall scenarios, not looking at the problematic recovery process, this was not noticeable.

Simulation of GOAFR algorithm under the same scenarios resulted in 100 percent packet delivery;

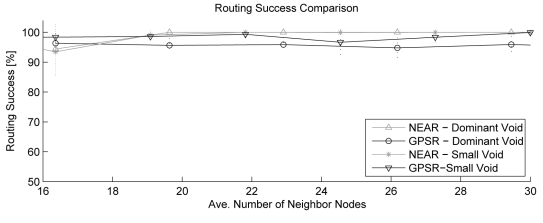

Fig. 16. Comparison between NEAR and GPSR success rate.

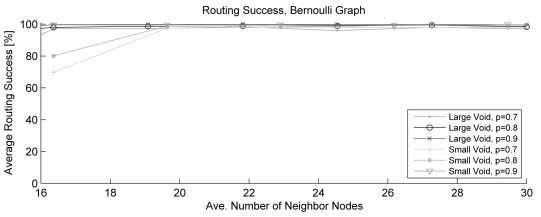

Fig. 17. Routing success, Bernoulli graph.

however, the performance of GOAFR was worse than GPSR's: 1.1 to 1.8 GPSR's distance in small void scenarios and 2.05 to 2.25 GPSR's distance in large void scenarios. For the clarity of the graphs this was omitted in Fig. 15.

5.3.3 Bernoulli Graphs

When connectivity between nodes is severely harmed by obstacles, close nodes may not be able to communicate, and there is a question whether geographical routing can work [38]. Thus, we conducted a set of simulations to check the performance of NEAR in such a scenario. Connectivity problems in ad hoc networks are frequently represented as Bernoulli graphs.

Definition 5.1. *A **Bernoulli graph** $B(n, p(n))$ is a graph consisting of n nodes, in which edges are chosen independently and with probability $p(n)$ [39].*

Here, an edge may exist between two nodes (v, w) if and only if they are within transmission range. $1 - p(n)$ represents the probability for link failure between two nodes within transmission range, and we assume no hidden terminals. Using the same simulation scenarios described in Section 5 and adapting it to the Bernoulli model, meaning removing links between neighbor nodes with probability p, we tested NEAR. Variable values of p were used to examine the effect of link failure on repositioning stability and further more routing success. While the link failure probability does affect the connectivity in the network, it does not affect the graph's repositioning stability as strongly. In all the cases that were simulated the repositioning succeeded and the elevation was not as strong as in a network that is p-dense. The reason is that while some nodes may have broken links between them, other neighbor nodes within the transmission range compensate for them in the repositioning process. The success rate of routing using a Bernoulli graph model is shown in Fig. 17. Except for the sparsest case, the success rate was over 97 percent. It should also be noted that the link failure rate commonly considered is less than 10 percent [40], [41], still NEAR performs adequately with higher failure rates as well.

5.3.4 Repositioning and Void Bypass Contribution to NEAR

The NEAR solution is constructed from two major parts: the repositioning part and the void bypass. Each of these parts contributes to NEAR performance, almost equally. We investigated the contribution of each part: to test the performance of repositioning alone, we ran the GPSR algorithm on scenarios where the repositioning algorithm was applied. Only slight modifications where made to the algorithm in order to be able to reach floating nodes. We

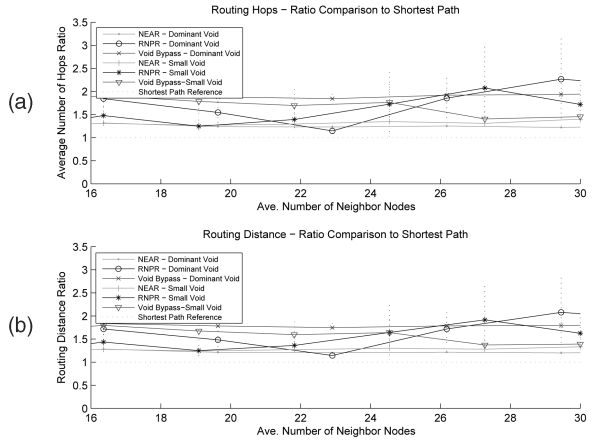

Fig. 18. Contribution of repositioning and void bypass to NEAR—by hops and distance ratio.

refer to this modified algorithm as "Repositioned Nodes Perimeter Routing—RNPR." To test the performance of void bypass alone, we ran NEAR without applying the repositioning algorithm. Fig. 18a shows RNPR and Void Bypass performance (by hops ratio) compared to NEAR and shortest path. Fig. 18b shows the distance ratio performance for the same scenarios. The results indicate that both the repositioning and the void bypass contribute to NEAR performance and that none of them alone performs as good as their combination. Like NEAR, the performance of Void Bypass does not vary significantly with network density. In a small void, it shows a performance that is a bit better for higher densities, presumably as the contribution of the greedy algorithm here is higher than the bypass itself. RNPR, on the other hand, is erratic—the performance varies as density rises. The simulation has shown that it depends on the network scenario topography and the adaptations made to the GPSR algorithm. The performance range from 20 percent to 140 percent worse than the shortest path. It is still always better than GPSR alone by hops and is about the same as GPSR in the distance for high network densities.

One outcome of these simulations is that none of the algorithms (RNPR, Void Bypass) performed as was expected in terms of message delivery success. We have already mentioned the modifications to the GPSR required to meet the repositioning effect, but Void Bypass is affected as well. The void bypass algorithm is based on several assumptions on the topography of the network after repositioning and the selection of void edge nodes is based on that. When no repositioning takes effect, the bypass discovery sometimes fails, and sometimes, it is just not good enough. RNPR and Void Bypass manage to deliver about 10 percent messages less than NEAR, which currently makes them unusable. We believe that with further work RNPR, the combination of repositioning and GPSR, can prove to be stable and still more efficient than GPSR, though not with the same performance as NEAR. Void bypass, on the other hand, is not recommended for use without repositioning.

6 Comparison with Existing Solutions

The NEAR solution belongs to the group of recovery algorithms, meant to handle routing through concave nodes, as mentioned in Section 1. An advantage of NEAR over most solutions is the decrease in the percentage of concave nodes in the virtual network, which increases the greedy routing efficiency and reduces the need for recovery. An exception in this case is the Terminodes [17] project, where concave nodes are avoided by using a combination of globally defined anchors and local table driven routing. However, Terminodes is significantly harder to implement compared to other solutions and is medially robust to a failure of a single node [5].

A group of recovery algorithms, including GPSR [28], GFG [27], GOAFR [26], and their variants have many similarities to NEAR. They, too, start in a greedy routing mode and switch to recovery mode only when a concave node is reached. The common denominator between these algorithms is that their recovery process is based on traversing the edge of a void using different techniques and are mostly based on planarization. The use of planar graphs requires the algorithms to maintain information about the planarized graph connectivity, based on local information. When a node's neighbor moves, the node has to check whether the planar connectivity has changed and if an edge should be removed or added to the graph. Practically, the resources required for planarization and NEAR are similar; however, our solution's performance is better in dense networks.

There are two main advantages to NEAR over planarization algorithms: Its recovery routes are shorter and smoother, and it uses guaranteed bypass routes. The "tangling" of the routing sometimes happened when planarization is employed is due to the rugged shape of the void edge. Whether the void is due to a lake, an area without reception, or due to nodes spreading, one cannot always expect to have a smooth edge. All the planarization-based algorithms mentioned above will follow the rugged edge and penetrate peninsulas as they occur (like in Fig. 1), increasing the route length significantly. NEAR, on the other hand, will reach the edge of this area and will not penetrate it because of the repositioning. The second advantage is our employment of guaranteed bypass routes: the void bypass paths of NEAR are known and guaranteed to bypass the void while being adaptive and sensitive to changes in the network. It does not require each packet to perform a face exploration of the graph, thus avoiding complex routing and simplifying the computations. Note that the repositioning algorithm can improve any of the other planarization-based algorithms, even without the void discovery process, as was shown in Section 5.

The second group of recovery algorithms refer to algorithms that incorporate memory. Greedy/flooding [16] and Terminodes were already introduced and differ greatly from NEAR. Two other solutions, closer to NEAR in their features are INF [18] and SAGF [19]. INF (intermediate node forwarding) is a probabilistic solution for routing around voids using intermediate geographic locations. When a concave node is reached, it randomly chooses an intermediate position through which to route the packet. If routing through the intermediate node fails, another intermediate node is chosen, and multiple intermediate nodes can be used as well for the routing. INF keeps a table of destination nodes and their intermediate nodes, which is periodically updated. Two disadvantages of this solution

are the fact that it requires a NAK message to start the INF process, and it is based on the random selection of the intermediate node, which is oblivious to network shape and does not guarantee routing success in the first attempt. SAGF is a spatially aware geographic forwarding solution suitable mainly for networks with preassigned routes, e.g., nodes mounted on cars driving along the highway system. It assumes that each node possesses the model information of the geographic space wherein it is located, keeping location information for intermediate nodes. Before a message is sent, a route is calculated based on the spatial information using algorithms such as Dijkstra. The algorithm's complexity here depends on the size of the model network and of the ad hoc network. As other solutions, SAGF also routes in greedy mode until a concave mode is reached, and then calculates its path according to the spatial model. The memory requirement here is expressed by the spatial awareness, though it is also suggested in SAGF to use external information sources, which then move the memory burden to extra messaging. In both INF and SAGF, the routing is not based on local information alone, and memory requirements scale up as the network grows; INF memory requirements scale with the number of nodes, and SAGF scales with the number of vertices in the spatial graph. NEAR memory requirements depend on the number of neighbors and do not grow with the network size, yet NEAR remains aware of the network topography.

Multilevel-clustering approaches (Landmark [20], LANMAR [21], L+ [22], and Safari [23]) are best suited to scenarios involving group mobility, as it heavily relies on the cluster heads. The approach is mostly table driven, hence, it employs memory, and as the above schemes have explicit cluster heads, and all addresses are therefore relative to these, they are likely to have to change if a cluster head moves away [42]. In addition, its memory requirements make it less scalable than NEAR.

An algorithm with some resemblance lines to NEAR is PAGER [29], a sensor networks algorithm that identifies "Shadow Areas" in the network that include concave nodes, and later applies cost gradients to avoid these areas. One can refer to the shadow areas and cost gradient as an analog to the repositioning in NEAR. However, since PAGER is designed to sensor networks, the destination node is always the base station. Clearly, Pager answers the recovery problem for a single destination, but its algorithm cannot be applied to a multiple destination dynamic system, as the ad hoc network in which NEAR operates.

NEAR void bypass algorithm has some similarity to the BOUNDHOLE [30] algorithm, as both algorithms discover voids and find bypass routes around them. While BOUNDHOLE also uses *stuck angle* as the basis for void discovery, the criterion for such an angle is different, and can be as low as 120 degrees (since repositioning implications do not apply). The main drawback of the BOUNDHOLE algorithm is the lack of packet delivery guarantee [43], as it fails when the destination node lies within the hole and requires restricted flooding to deliver the packet. Unlike NEAR BOUNDHOLE also caches the entire boundary of a void, which increases the memory requirements and reduces scalability [43]. In addition, as BOUNDHOLE targets sensor networks, it addresses node failure but may not be adequate for mobile networks. The algorithm periodically checks for node failure and rediscovers the entire boundary of the hole if a node on the boundary failed. NEAR, on the other hand, is immediately aware of such changes and rediscovers only the minimal section of the void bypass that was affected. This is especially important in networks with high mobility where the nodes on the edge of a void rapidly change. As a result, BOUNDHOLE tends to impose higher load on nodes near the hole boundaries [43].

To conclude, though the NEAR solution is no more complex than other suggested algorithms, it outperforms them in dense networks, while taking into consideration network resources such as memory, messaging, computations, and routing efficiency.

7 Conclusion

This work presented NEAR, a solution incorporating both positioning and routing aspects to improve performance, based on local information alone. It was shown that by simple virtual repositioning of nodes the shape of voids can be smoothed and concave nodes can be predicted by their added virtual height. The virtual repositioning simplifies void detection and allows discovering void bypass routes during the repositioning process. The bypass routes are then maintained based on local changes alone.

In the routing section, simulations results showed an improvement in greedy routing and a decrease in the number of concave nodes thanks to the use of virtual repositioning. The case of concave nodes and recovery was also explained by the use of guaranteed void traversing paths, which require nodes along the void to keep only the ids of their immediate neighbors in the bypass path.

We discussed the characteristics of NEAR in terms of localization, memory requirements, weaknesses and advantages compared to existing recovery algorithms, as well as future research and improvements.

NEAR can improve ad hoc networks' ability to deal with voids and concave nodes, by implementing a revolutionary view of repositioning that allows local nodes to sense part of the greater network without requiring extra resources.

Acknowledgments

An early version of this work appeared in the proceedings of the Seventh ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '06).

References

[1] C. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," *Proc. ACM SIGCOMM '94,* pp. 234-244, 1994.

[2] C. Chiang, H. Wu, W. Liu, and M. Gerla, "Routing in Clustered Multihop, Mobile Wireless Networks," *Proc. IEEE Singapore Int'l Conf. Networks (SICON '97),* pp. 197-211, Apr. 1997.

[3] C. Perkins and E. Royer, "Ad-Hoc On-Demand Distance Vector Routing," *Proc. Second IEEE Workshop Mobile Computing Systems and Applications (WMCSA '99),* pp. 90-100, Feb. 1999.

[4] D. Johnson and D. Maltz, *Mobile Computing.* Kluwer Academic Publishers, 1996.

[5] M. Mauve, J. Widmer, and H. Hartenstein, "A Survey on Position-Based Routing in Mobile Ad Hoc Networks," *IEEE Networks Magazine,* vol. 15, no. 6, pp. 30-39, 2001.

[6] D. Johnson, "Routing in Ad Hoc Networks of Mobile Hosts," *Proc. Workshop Mobile Computing Systems and Applications (WMCSA),* 1994.
[7] S. Basagni, I. Chlamatac, V. Syrotiuk, and B. Wood, "A Distance Routing Effect Algorithm for Mobility (DREAM)," *Proc. ACM MobiCom '98,* pp. 76-84, 1998.
[8] Y. Ko and N.H. Vaidya, "Location-Aided Routing (LAR) in Mobile Ad Hoc Networks," *Mobile Computing and Networking,* pp. 66-75, 1998.
[9] X. Lin, M. Lakshdisi, and I. Stojmenovic, "Location Based Localized Alternate, Disjoint, Multipath and Component Routing Schemes for Wireless Networks," *Proc. ACM MobiHoc '01,* pp. 287-290, Oct. 2001.
[10] Z. Haas and B. Liang, "Ad Hoc Mobility Management with Uniform Quorum Systems," *IEEE/ACM Trans. Networking,* vol. 7, no. 2, pp. 228-240, 1999.
[11] I. Stojmenovic, "A Routing Strategy and Quorum Based Location Update Scheme for Ad Hoc Wireless Networks," Technical Report TR-99-09, Computer Science Dept., Univ. of Ottawa, 1999.
[12] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris, "A Scalable Location Service for Geographic Ad-Hoc Routing," *Proc. ACM MobiCom '00,* pp. 120-130, Aug. 2000.
[13] S. Das, H. Pucha, and Y. Hu, "Performance Comparison of Scalable Location Services for Geographic Ad Hoc Routing," *Proc. IEEE INFOCOM,* 2005.
[14] S. Giordano and M. Hami, "Mobility Management: The Virtual Home Region," Technical Report SSC/037, EPFL, 1999.
[15] I. Stojmenovic, "Position Based Routing in Ad Hoc Networks," *IEEE Comm. Magazine,* vol. 40, no. 7, pp. 128-134, 2002.
[16] I. Stojmenovic and X. Lin, "Loop-Free Hybrid Single-Path/Flooding Routing Algorithms with Guaranteed Delivery for Wireless Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 10, pp. 1023-1032, Oct. 2001.
[17] L. Blazevic, L. Buttyan, S. Capkun, S. Giordano, J. Hubaux, and J.L. Boudec, "Self-Organization in Mobile Ad Hoc Networks: The Approach of Terminodes," *IEEE Comm. Magazine,* pp. 166-175, 2001.
[18] D.S.J. De Couto and R. Morris, "Location Proxies and Intermediate Node Forwarding for Practical Geographic Forwarding," Technical Report MIT-LCS-TR824, MIT Laboratory for Computer Science, June 2001.
[19] J. Tian, I. Stepanov, and K. Rothermel, "Spatial Aware Geographic Forwarding for Mobile Ad Hoc Networks," technical report, Stuttgart Univ., 2002.
[20] P.F. Tsuchiya, "The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks," *Proc. Symp. Comm. Architectures and Protocols (SIGCOMM '88),* pp. 35-42, 1988.
[21] G. Pei, M. Gerla, and X. Hong, "Lanmar: Landmark Routing for Large Scale Wireless Ad Hoc Networks with Group Mobility," *Proc. MobiHoc '00,* pp. 11-18, 2000.
[22] B. Chen and R. Morris, "L+: Scalable Landmark Routing and Address Lookup for Multi-Hop Wireless Networks," Technical Report 837, MIT Laboratory for Computer Science, 2002.
[23] S. Du, A. Khan, S. PalChaudhuri, A. Post, A.K. Saha, P. Druschel, D.B. Johnson, and R. Riedi, "Safari: A Self-Organizing Hierarchical Architecture for Scalable Ad Hoc Networking," *Ad Hoc Networks J.,* 2007.
[24] E. Kranakis, H. Singh, and J. Urrutia, "Compass Routing on Geometric Networks," *Proc. 11th Canadian Conf. Computational Geometry (CCCG '99),* pp. 51-54, Aug. 1999.
[25] F. Kuhn, R. Wattenhofer, and A. Zollinger, "Asymptotically Optimal Geometric Mobile Ad-Hoc Routing," *Proc. Workshop Discrete Algorithms and Methods for Mobile Computing and Comm. (Dial-M '02),* pp. 24-33, 2002.
[26] F. Kuhn, R. Wattenhofer, and A. Zollinger, "Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing," *Proc. ACM MobiHoc,* 2003.
[27] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with Guaranteed Delivery in Ad Hoc Wireless Networks," *Wireless Networks,* vol. 7, no. 6, pp. 609-616, 2001.
[28] B. Karp and H.T. Kung, "Greedy Perimeter Stateless Routing for Wireless Networks," *Proc. ACM MobiCom '00,* pp. 243-254, Aug. 2000.
[29] L. Zou, M. Lu, and Z. Xiong, "A Distributed Algorithm for the Dead End Problem of Location Based Routing in Sensor Networks," *IEEE Trans. Vehicular Technology,* vol. 54, no. 4, pp. 1509-1522, July 2005.
[30] Q. Fang, J. Gao, and L.J. Guibas, "Locating and Bypassing Holes in Sensor Networks," *Mobile Networks and Applications,* vol. 11, no. 2, pp. 187-200, 2006.
[31] C. Lochert, H. Hartenstein, J. Tian, H. Fussler, D. Hermann, and M. Mauve, "A Routing Strategy for Vehicular Ad Hoc Networks in City Environments," *Proc. IEEE Intelligent Vehicles Symp. (IVS '03),* June 2003.
[32] V.D. Park and M.S. Corson, "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks," *Proc. IEEE INFOCOM '97,* pp. 1405-1413, 1997.
[33] N. Arad, "Minimizing Recovery State in Geographic Ad-Hoc Routing," master's thesis, School of Electrical Eng., Tel Aviv Univ., May 2007.
[34] B. Karp, "Geographic Routing for Wireless Networks," PhD dissertation, Harvard Univ., Oct. 2000.
[35] P. Santi, "The Critical Transmitting Range for Connectivity in Mobile Ad Hoc Networks," *IEEE Trans. Mobile Computing,* vol. 4, no. 3, pp. 310-317, 2005.
[36] J. Broch, D.A. Maltz, D.B. Johnson, Y.C. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," *Proc. ACM MobiCom '98,* pp. 85-97, 1998.
[37] E. Royer, P. Melliar-Smith, and L. Moser, "An Analysis of the Optimum Node Density for Ad Hoc Mobile Networks," *Proc. IEEE Int'l Conf. Comm. (ICC),* 2001.
[38] Y.J. Kim, R. Govindan, B. Karp, and S. Shenker, "Practical and Robust Geographic Routing in Wireless Networks," *Proc. Second Int'l Conf. Embedded Networked Sensor Systems (SenSys '04),* pp. 295-296, 2004.
[39] B. Bollobas, *Random Graphs.* Cambridge Univ. Press, 1985.
[40] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for Wireless IP Communications," *IEEE J. Selected Areas in Comm.,* vol. 22, no. 4, pp. 747-756, 2004.
[41] R. Igarashi, K. Takahata, N. Uchida, and Y. Shibata, "Packet Loss Rate Control for Continuous Media over Heterogeneous Network," *Proc. 18th Int'l Conf. Advanced Information Networking and Application (AINA),* 2004.
[42] J. Eriksson, M. Faloutsos, and S. Krishnamurthy, "Scalable Ad Hoc Routing: The Case for Dynamic Addressing," *Proc. IEEE INFOCOM,* pp. 1108-1119, 2004.
[43] D. Chen and P.K. Varshney, "A Survey of Void Handling Techniques for Geographic Routing in Wireless Networks," *IEEE Comm. Surveys and Tutorials,* vol. 9, no. 1, pp. 50-67, 2007.

Noa Arad received the BSc (cum laude) and MSc (cum laude) degrees in electrical engineering from Tel-Aviv University, Israel, in 2003 and 2007, respectively. Since 1999, she has filled several system development, architecture, and leading positions in the telecommunications industry. She is currently a PhD candidate in the School of Electrical Engineering, Tel-Aviv University. Her research focuses on Internet measurement, mapping, and characterization and ad hoc networks.

Yuval Shavitt received the BSc degree (cum laude) in computer engineering, the MSc degree in electrical engineering, and the DSc degree from Technion, Israel Institute of Technology, Haifa, in 1986, 1992, and 1996, respectively. After graduation, he spent a year as a postdoctoral fellow in the Department of Computer Science at Johns Hopkins University, Baltimore, Maryland. Between 1997 and 2001, he was a member of technical stuff at the Networking Research Laboratory, Bell Labs., Lucent Technologies, Holmdel, New Jersey. Since October 2000, he has been a faculty member in the School of Electrical Engineering at Tel-Aviv University, Tel-Aviv, Israel. He served as a TPC member for INFOCOM 2000-2003 and 2005, IWQoS 2001 and 2002, ICNP 2001, IWAN 2002-2005, tridentcom 2005-2006, and more, and on the executive committee of INFOCOM 2000, 2002, and 2003. He was an editor of *Computer Networks* from 2003-2004 and has served as a guest editor for *JSAC* and *JWWW*. His recent research focuses on Internet measurement, mapping, and characterization, QoS in networks, and ad hoc routing. He is a senior member of the IEEE.