Minimizing Recovery State in Geographic Ad-Hoc Routing

Noa Arad
School of Electrical Engineering
Tel Aviv University
Email: noa@eng.tau.ac.il

Yuval Shavitt
School of Electrical Engineering
Tel Aviv University
Email: shavitt@eng.tau.ac.il

***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 NEAR, Node Elevation Ad-hoc Routing, 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.

I. Introduction

Ad-Hoc networks are infrastructure-less networks, made up from mobile nodes, which are using their neighbors as a means to communicate with other nodes in the network. Ad-hoc networks change their topology, which is expressed by the node connectivity, over time, as the nodes change their position in space. Routing schemes of 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], 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.

Geographic ad-hoc networks, using position based routing, are targeted to handle large networks, containing many nodes. Such networks are not suited to use topology based algorithm as the amount of resources required for that is 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 network-wide 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, multi-path, or flooding. Flooding protocols are usually restricted directional, such as DREAM [7] and LAR [8], namely, the flooding is done only in a section of the network, which is selected based on the source and destination nodes location. Multi-path protocols attempts to forward the message along several routes towards its destination in order to increase the probability of finding a feasible path. On the other hand, single path protocols aim at 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.

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 [9]. 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 to switch to a recovery algorithm which guarantee packet delivery. They can be classified into memory-based and memory-free.

One routing algorithm that employ memory is the Greedy/Flooding algorithm [10], that once reaching a concave node floods the message towards 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 [11], 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, that the path should loosely traverse. The anchored nodes list is memorized, and updated every time the packet passes in the vicinity of the next anchor node.

Recovery algorithms without memory often use planner graphs for routing (A planner graph is a graph that can be drawn on the plane such that no two edges intersect). One of the first works in this area was "Compass Routing II" [12], also called "Face Routing", proving that Delaunay triangulations of point sets on the plane support compass routing and guaranteeing delivery. This algorithm was the base to further suggestions, such as AFR [13] and GOFAR [14]. Another important algorithm is GPSR (Greedy Perimeter Stateless Routing) [15]. It too requires the network to be plannar in order to get successful routing. This property is achieved by creating a Gabriel Graph (GG) or a sub-set of it, Relative Neighborhood Graph (RNG). In GPSR, a packet is initially routed using a greedy algorithm, until reaching a concave node. Then it switches to perimeter mode and traverses the face of the planar graph using the right-hand rule, until it recovers from the local maxima, and the greedy routing can continue.

One problem that recovery protocols do not prevent is that the packet always needs to reach a concave node 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, there is a long retreat to reach a point where an alternative path can be found. We propose a novel scheme to deal with this problem by preventing the routing algorithm from entering concave areas. Our 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 which is based on the virtual coordinates.
- An obstacle bypass procedure.

II. Concave Nodes

A concave node is a node that has no neighbors closer to the destination than itself [9]. The term 'closer' is somewhat fuzzy, as different greedy algorithms have different closeness criteria. Since our ideas can be used with different greedy routing algorithms we define a concave node as a node that has no neighbor that can make a greedy progress towards the destination (for the routing algorithm in use). Since position-based routing is using local information for forwarding decisions, a concave node can not be anticipated in advance, based on neighbor nodes position. Indeed, using the 2-neighborhood information can improve the algorithm decisions, but cannot avoid reaching concave nodes.

Assuming one uses a recovery algorithm that switches back to greedy mode once recovered from the concave situation, the number of back-tracking packet transmissions required to switch back to greedy mode can vary between just a few hops to a very long retreat. Figure 1 shows an example how a concave node can be reached only after numerous hops before the recovery process begins. In addition, [16] 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 simply increasing the number of hops.

Thus, we extend the definition of concaveness. A concaveness in the wide sense means that none of the node's neighbors towards the destination can eventually lead to the destination. A first degree concaveness, is the same as the general definition for concaveness, a node that does not have a greedy next hop to the destination. A concaveness of the nth degree, is a concaveness where the smallest concaveness degree of all neighbors is $n-1$. By identifying regions of concaved nodes in the wide sense, our greedy algorithm can stop short of entering them, and thus avoid long retreats.

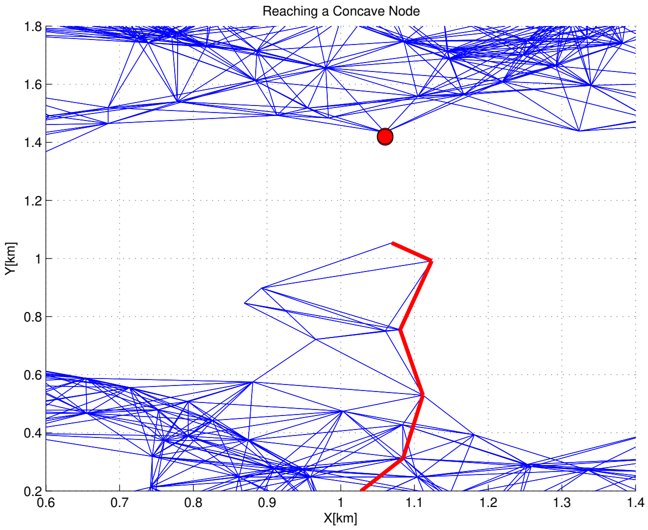

Fig. 1. Concave node with recovery path

III. Our Solution - The NEAR Algorithm

Many of the problems of position-based routing originate from the fact that the shape of the network is unknown ubiquitously, and it is dynamically changing due to node mobility. The lack of information remove the network shape from being a substantial part of the routing process, and does not allow educated routing decisions. GPSR [15], 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. However, there is no guarantee that this node, or the next one, will not be another concave node, a local maximum 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 recovery process can be either avoided or significantly improved. Node repositioning has several goals. The first one 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 for sure the node is concave for routing in this direction (we will see later that even here one must be cautious in deciding about concavity). Our method indicates a concave node by elevating it. In an N-dimension coordinate system, an $N+1$ dimension is added that indicates, if its coordinate is non-zero, that the node is virtually repositioned. The rest of the N dimension coordinates are updated as well to reflect the node's connectivity, as will be later described.

For simplicity lets assume that nodes has two coordinates describing their X and Y coordinates. In this case a concave

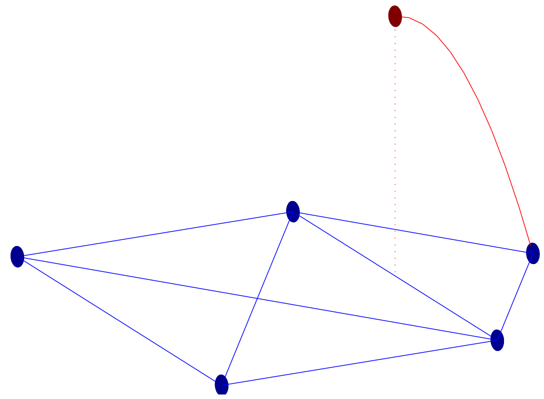

Fig. 2. A repositioning example

node will be assigned a positive virtual Z coordinate that reflects its distance from its neighbors. Figure 2 demonstrates a simple repositioning of a node. Note that the reflection of the node's virtual coordinates on the X-Y plane lays between its two neighbors. We shall term the virtually repositioned nodes *floating* nodes. Every floating node is concave in the wide sense.

A second goal of repositioning is to improve the 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 the repositioning, is to improve the recovery process. Though our greedy routing is improved, sometimes, reaching a concave node cannot be avoided. However, an immediate effect of the repositioning is that every peninsula in the shape of the network is elevated, and a smooth edge is surrounding the routing void. We use addition distributed void identification algorithm to help us identify the voids. Once a void is identified, its smooth edge be easily followed with minimal number of hops and without entanglement, which plague some of the other recovery algorithms.

Thus, we present as a solution, the NEAR (Node Elevation Ad-hoc Routing) algorithm, which is comprised of several algorithms that feed each other and are all distributed. At network start up, we run the repositioning algorithm which is distributed and local, this algorithm is executed periodically at low cost due to its local nature. At start up we also execute a void identification algorithm which is executed around the void. This algorithm is distributed as well but it is executed by all the nodes at the void edge and maybe their neighbors, thus we term it a regional algorithm and its termination time depends on the number of nodes in the void perimeter. Once a void is identified, the maintenance of its identification is purely local. The output of these algorithms is used by the routing algorithm which is a variant of previously published greedy routing algorithm and an efficient recovery algorithm. In the next sections we will describe our algorithms and their performance.

IV. 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 neighbtor 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°. If such a direction does exist, the node recalculates its position and updated its $n+1$th dimension location. In addition the void maintenance algorithm is executed, which can make either of two changes: insert the node to be part of the void edge, or remove it and reconnect its neighbors.

Figure 3 gives a formal description of the coordinate calculation 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 virtual coordinates of node v, and by P its real coordinates. a dotted variable, e.g., $\dot{W}$, indicates the value which has just been accepted (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 (e.g., through timeout from the last Hello message from this neighbor). The node looks for a change in a neighbor disposition using $maxangle$, which returns the two adjacent nodes with maximal angle between, which are either grounded or below v_z. In case of change, the node recalculates its new virtual coordinates by calling $calc$ (in lines 5 and 15), and immediately updates its neighbors. In addition it maintains the void surface by calling $voidupdate$ as will be explained later. Note that node repositioning may either elevate a node or pull it down.

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 important, 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 Figure 5). Though only very few of the nodes in this area may be concave in this direction by the narrow definition of concaveness (depending on placement and type of routing), a packet routing should still avoid entering the peninsula area in order to prevent the routing algorithm from entering recovery mode. Our nodes repositioning will cause the tongue of land to role up to create a smooth surface around the void, while gaining height in the $n+1$th dimension. 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 region of error). The "height" convention that we use here should not be confused with

Algorithm Node Reposition

For a new vector $\dot{W}$ from node w

1. if $|W - \dot{W}| > \varepsilon$:
2. $\quad w \leftarrow \dot{w}$
3. $\quad (s,t) \leftarrow maxangle(resort(CV))$
4. $\quad$ if $(t = NULL)||(\angle s - \angle t > \alpha)||(v_z > 1)$
5. $\quad\quad chg \leftarrow calc(s,t)$
6. $\quad$ else
7. $\quad\quad chg \leftarrow false$
8. $\quad\quad V \leftarrow P$
9. $\quad voidupdate(\mathcal{VN}_v, w)$
10. if chg
11. $\quad$ Transmit new vector $\dot{V}$ of v

For loosing connection to neighbor w

12. $\mathcal{N}_v \leftarrow \mathcal{N}_v \setminus \{w\}$
13. $(s,t) \leftarrow maxangle(CV)$
14. if $(t = NULL)||(\angle s - \angle t > \alpha)$
15. $\quad chg \leftarrow calc(s,t)$
16. else
17. $\quad chg \leftarrow false$
18. $voidupdate(\mathcal{VN}_v, w)$
19. if chg
20. $\quad$ Transmit new vector $\dot{V}$ of v

Fig. 3. Repositioning algorithm for node v.

Algorithm Reposition calculation

$calc(s,t)$

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

Fig. 4. Reposition calculation

the "height" convention of the TORA algorithm [17], which refers to a completely different concept. Figure 5 and Figure 6 demonstrate the repositioning effect for an example network.

The repositioning depends on two important factors: The network density and the threshold angle, α. Figure 7 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 static networks spread in a square area of $2Km \times 2Km$. Nodes with transmission radius of 250m (as in IEEE 802.11 WaveLan) were uniformly distributed in the area, where a horseshoe shaped area is disallowed for

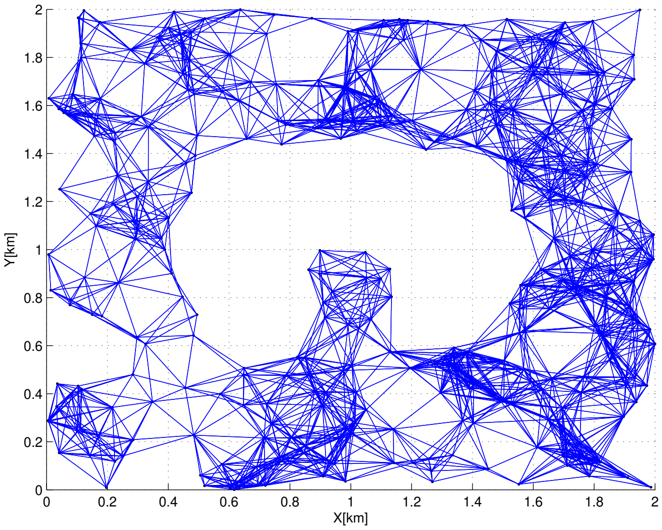

Fig. 5. An example network topology before repositioning

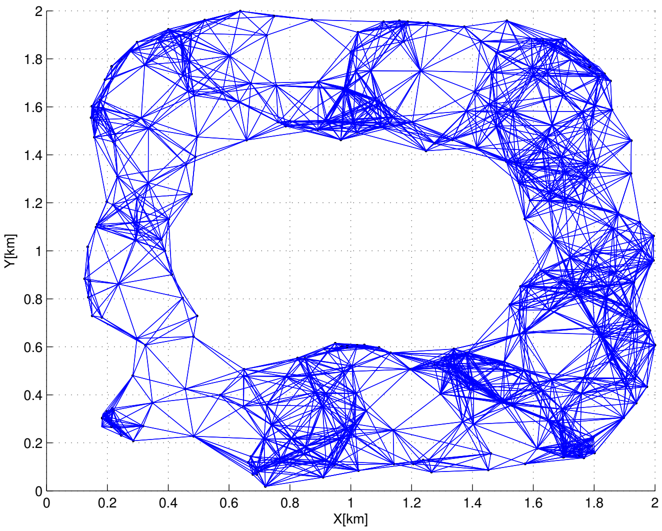

Fig. 6. AN example network topology after repositioning

positioning, thus creating a void (like the instance in Figure 5). 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 large connected component as demonstrated in Figure 8. Other studies in this area [15], [18], [14] were also using random ad-hoc networks which are, at least, this dense. One can allow sparser networks (and risk disconnections) by manually marking some areas or nodes as grounded, these nodes will halt the folding process without hurting routing performance.

The second factor is the threshold angle, α. A minimal angle of $180°$ is simply too low, and almost all nodes will float.

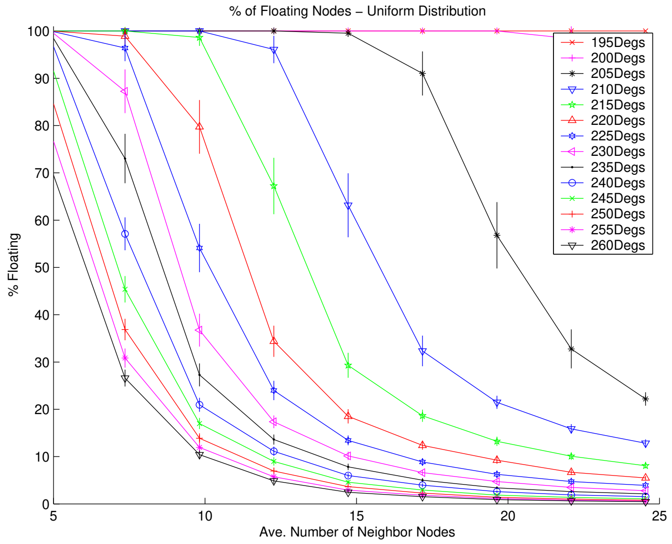

Fig. 7. The network density effect on repositioning

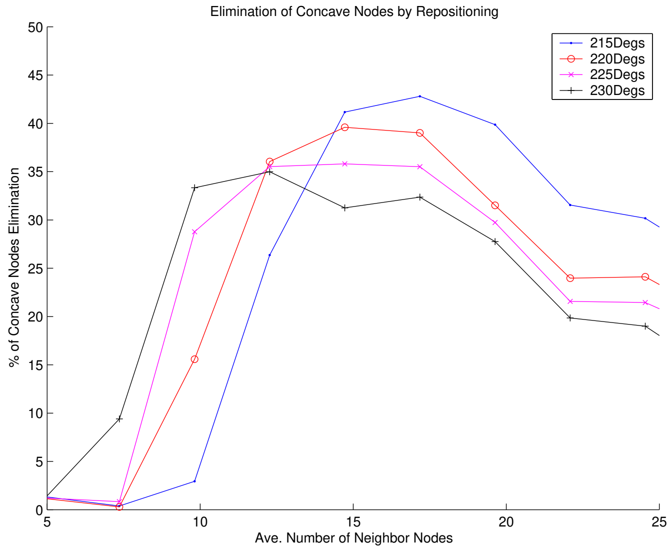

Fig. 9. Elimination of concave nodes by Repositioning

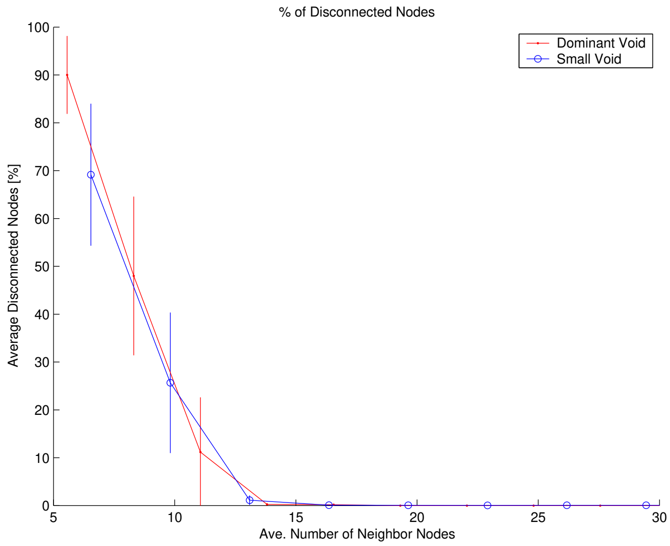

Fig. 8. The percentage of disconnected nodes vs. nodes density

There is a clear threshold around $\alpha = 200°$ above which the system is stable, with relatively enough routable nodes. $\alpha = 210 - 230°$ was found to be best for various scenarios. Using a wider angle threshold will achieve less repositioned nodes, and will miss the entire 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.

V. Routing Algorithm

The use of virtually repositioned nodes in network does not contradict the use of standard greedy routing algorithms. On the contrary, the use of greedy algorithms is in the base of the routing, and the network virtual coordinates are intended to increase the efficiency of greedy routing compared to the original geographic coordinates. The improvement in greedy routing is demonstrated in Figure 9. In the graph, we plot the percentage of concave nodes that were eliminated from the routing because of the repositioning. When the network's density is sufficiently high, 62 to 100 nodes per Km^2 (12 – 20 neighbors per node), up to 45% of the routes which previously reached concave nodes, now complete their routing procedure using only greedy routing.

However, our routing algorithm cannot be purely greedy due to three cases. First, as with previous work in this area, we may reach a concave node and when this happen (much less frequently in our case) we need to switch to recovery mode. In addition, if either the source node or (and) 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 one additional rule for the next node selection: one may use only non-floating nodes. This way we avoid the concave areas until we reach the destination or its vicinity.

A pseudo code of the routing algorithm is given in Figure 10. 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 non-null 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); $\mathcal{VN}_v$ is the group of voids v participates in;

The routing algorithm first checks for the 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 real physical coordinates are close enough to the destination. This is the only part of the algorithm that requires knowledge of the original physical coordinates, since in case more than a single tongue roles 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 Z_{max} will always be set lower than the source node height, in order to force descent towards ground level, and will be updated with every hop until a non-floating node is reached. When a message has to reach a floating destination, Z_{max} is set to the destination height.

If neither of the two conditions above is true we are routing the message only through non-floating nodes, and we can be in either greedy routing mode or perimeter routing mode, where the latter is our enhanced equivalent to recovery routing. Thus, the algorithm mostly spends time in line 7 where it select the next greedy node towards the destination. If the greedy algorithm fails to find a next node, we switch to perimeter mode (line 10). Here we are either part of the void edge (line 11) or, since we are not using a transformation to a planner graph like other solutions [15], [19], have a neighbor who is part of the void edge (line 17). In either case we choose the best (based on the local angle) of the two directions to travel along the void perimeter. While in perimeter mode we check for the 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.

Note that the void maintenance 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. Our simulations have shown that this situation occurs on the average only once per every 62,000 messages through concave nodes, and their average traversing length was less than ten hops, which is very short compared to average void bypass length. In more than 80% 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 better and more complicated discovery algorithm, but we feel that we stroke the right balance.

A pseudo code of the void discovery process is shown in Figure 11. Only non-floating nodes may take part in the void bypass cycle, besides one exception (bridge nodes) which will be discussed soon. A node may enter the void discovery algorithm in two ways: it can receive a $init\_void\_discovery$ message either from the routing algorithm or the repositioning algorithm. In the first case (line 4 in Figure 11) the initiation call passes the angle, α_D, which is a known concave direction; in the latter no parameter is passed and the node checks whether the maximal angle between any of its two neighbors is greater than β, which indicates concavity in that direction (line 6). The second way to enter the algorithm is by receiving a void message (VM) which is part of the void cycle creation

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. $\quad\quad\quad\quad (\angle\mathcal{VN}_{id}.ccw \leq \angle D \leq \angle\mathcal{VN}_{id}.cw)$
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. $\quad\quad\quad\quad (\angle\mathcal{VN}_{id}.ccw \leq \angle D \leq \angle\mathcal{VN}_{id}.cw)$
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. 10. Message routing algorithm

process. The next step is to create a record for the void bypass cycle, containing a randomly generated void id (32 bit ids makes the probability of misidentification negligible), and the next node in the void bypass circle. A void creation message is then sent to the next node (line 11) and v waits for the message to return on its other side. If the message fails to return within a timeout a void creation message is sent in the opposite direction. This is a rare event (which we did not encounter in our simulations) since it is most likely that other nodes will trigger the void discovery process concurrently.

A node that receives a void discovery message may be in one of the following states. The node may not be part of any processed void (line 28), in which case a new record is created for the void bypass cycle and a void discovery message is sent to a selected next node (line 30). A simplified explanation of the next node selection is by no-crossing heuristic using the right hand rule. Though this selection may succeed in over 99.5% of the time [20] we use additional rules, such as information about the 2-neighborhood, to improve further the process success probability (These rules are omitted from this text). If the process fails, it is easily detected, and an attempt in the other direction is made. A second node state in which a void creation message may be received, is when a void bypass cycle is complete (line 17), and the message returns to the initiating node. In the third state, the message is received by a node that is already part of the void, but is not the initiation node, which indicates a loop in the discovered void. This loop is corrected by backtracking (line 21). Finally, the void creation message may be received by a node that has already started the same void discovery process in parallel (as the algorithm is distributed). In this case, the two void bypass circles will be merged (line 26), and share a single void id (using some simple leader election like algorithm).

An interesting special structure in the virtual network is the *bridge*. A bridge is a series of floating nodes connecting two sides of a void. An example of such bridge is shown in Figure 12. Bridge creation is fairly rare, since the threshold angle we use is higher than 210°, e.g., if the two bridge nodes in Figure 12 would be closer in the X coordinate, they would not float, and we will have two adjacent voids without a bridge. Though the 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 to slight node repositioning since nodes in bridges act identical to non-floating nodes. Identifying a bridge is simple since the floating nodes X-Y coordinates are not close to the edge of the void, above non-floating nodes, but just have a floating height added with a small change of the X-Y coordinates (especially the bridge head(s)). The bridge head is the highest repositioned node on the bridge, and it is the one to initiate the void traversing process. Thus using the bridges the NEAR efficiency is not damaged by repositioning. In case of two bridge heads, one is elected to act as the head.

An estimation of the algorithm efficiency in routing and bypassing obstacles can be obtained by a comparison to shortest path routing results. 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, we discuss two simulation scenarios. The first simulation examines the algorithm with a randomly placed void, whose size does not exceed 10% of the network size. The second simulation examines the case of a void placed in the middle of the network and which covers approximately 25% of the network size. In addition, a tongue of land is entering the void (see Figure 5). 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. Chosen simulation area is $2Km \times 2Km$ in size, and has a variable network density. Each result is an average over twenty different network distributions. GEDIR [10] is the greedy algorithm used in the simulations.

Algorithm $init\_void\_discovery(\alpha_D)$

For calling $init\_void\_discovery(\alpha_D)$ in node v
1. if $(V_z > 0)\&\&(v \neq bridge\_head)$
2.   return()
3. if $(\alpha_D \neq NULL)||(angle(maxangle(CV))) > \beta)$
4.   if $(\alpha_D \neq NULL)$
5. $(s,t) \leftarrow \min_{w \in CV}(|\angle\alpha_D - \angle W|)$
6.   else
7. $(s,t) \leftarrow maxangle(CV)$
8. $id \leftarrow random()$
9. $\mathcal{VN}_{id}.cw \leftarrow t$
10. $\mathcal{VN}_{id}.ccw \leftarrow NULL$
11.   send message $VM(cw,id)$ to t
12.   start timer

For timeout
13. $\mathcal{VN}_{id}.cw \leftarrow NULL$
14. $\mathcal{VN}_{id}.ccw \leftarrow s$
15. send message $VM(ccw,id)$ to s

For message $VM(dir,id)$ received by v from w
16. if $(\exists\mathcal{VN}_{id})$
17.   if $(\mathcal{VN}_{id}.dir = NULL)$
18. $\mathcal{VN}_{id}.dir \leftarrow w$
19.   stop timer
20.   return() /* void discovery completed */
21.   else
22. $\mathcal{VN}_{id}.dir \leftarrow next(dir, w, \mathcal{VN}_{id}.dir)$
23. $delete\_loop(id, w)$
24.   send message VM to $\mathcal{VN}_{id}.dir(dir,id)$
25. else
26.   if $\exists\mathcal{VN}_i : \mathcal{VN}_i.\overline{dir} = w$
27. $merge\_voids(id, i)$
28.   else
29. $\mathcal{VN}_{id}.\overline{dir} \leftarrow w$
30. $\mathcal{VN}_{id}.dir \leftarrow next(dir, w, NULL)$
31.   send message $VM(dir,id)$ to $\mathcal{VN}_{id}.dir$

Fig. 11. The void discovery algorithm (simplified)

Figure 13 shows a comparison of the route length in hops between the NEAR routing, the GPSR [15] routing algorithm, and the shortest path calculated by a central algorithm. The compared routing areas are both the dominant and the small void scenarios. 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 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 10% at most, which translate to less than one

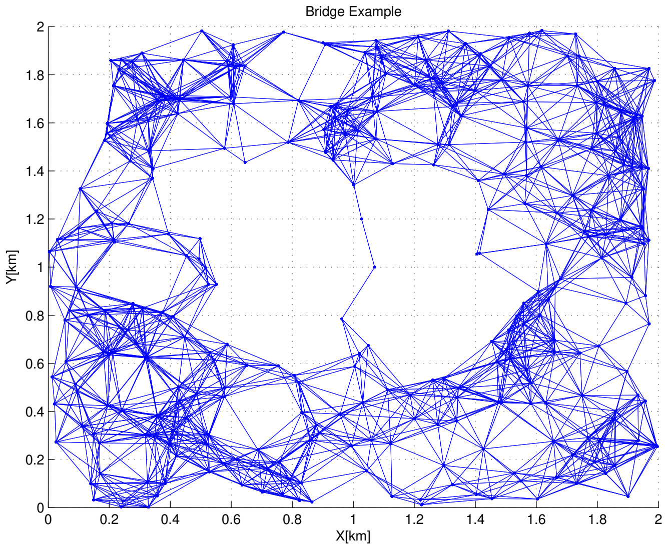

Fig. 12. A bridge example.

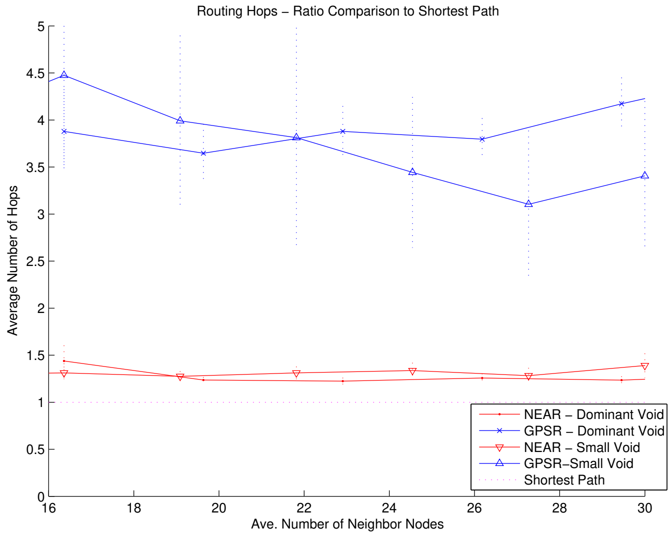

Fig. 13. Comparison between NEAR, GPSR, and shortest path by hops

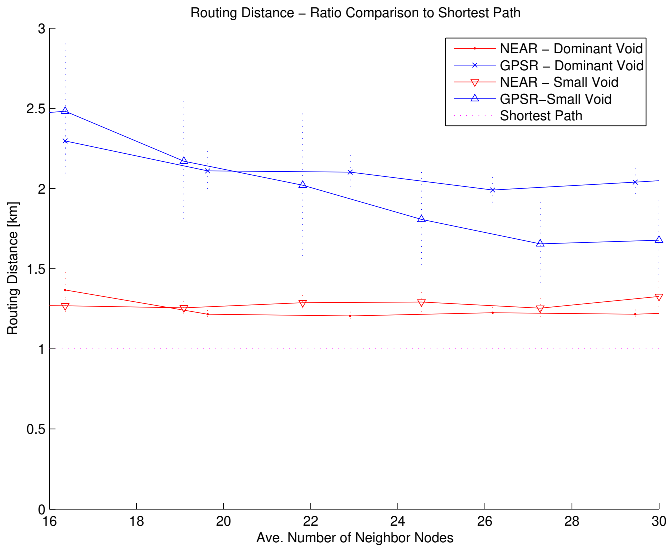

Fig. 14. comparison between NEAR, GPSR, and shortest path by distance

hop on the average. GPSR average number of hops is almost four times the number achieved by shortest path, and over three times more than NEAR. Figure 14 compares between the same algorithms, under the same scenarios, but with a criterion of physical routing length. Here too, the shortest path is better than NEAR by up to 10% at most, and by about 100% than GPSR. It should be noted that while many of the GPSR recovery routes are short almost as NEAR, some are very long - which manifests in its confidence intervals.

Figure 15 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% to 99% in the small void scenario, and 94% to 96% in a dominant void scenario, whereas NEAR delivers over 99% of the packets in both scenarios, except for 94% delivery rate in a sparse network with a dominant void. The GPSR algorithm recovery algorithm suffers mostly in our simulation from routing towards the edges of the network. As Karp indicates in his dissertation [20], the 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 [20], [15] averaged the GPSR performance overall scenarios, not looking at the problematic recovery process, this was not noticeable.

VI. Discussion

A. The NEAR Algorithm Characteristics

As locality is a major issue in geographic ad-hoc networks [9],[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 may be aware only of its own place 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 is only marginally benefiting from route caching, since our routes are very good in 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 maintenance messages are sent only around voids, thus their overhead is linearly

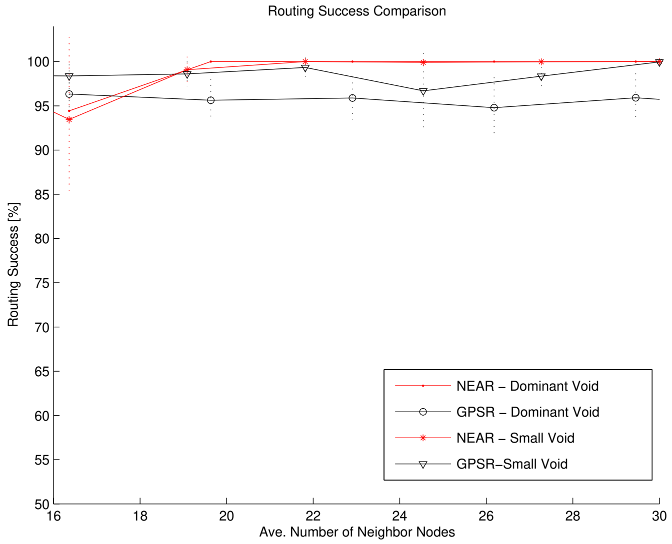

Fig. 15. comparison between NEAR and GPSR failure rate

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. In such occasion 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 algorithm on ring networks). During the network operation, the shape of the void may change 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 nodes is required to store its neighbors real and virtual coordinates. Nodes that participate in the void bypass routes store also 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 ground station, namely these stations cannot float. This is beneficial, for example, in network corners, say at the geographic border of the network coverage due to law, regulation, or physical obstacle. The non-floating 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. Failing to use the right threshold angle in a sparse network (without ground stations) may cause the network to wrap up. However, it is important to note that when a network becomes too sparse the risk of loosing 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 III, Figure 7.

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 GPP (Greedy Perimeter Probing) [20]. GPP, which is based on the right-hand rule with no-crossing heuristic rule, proved to succeed for over 99.5% 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 a single time per each void (afterwards maintenance is local) permits the usage of algorithms with larger overhead, e.g., algorithms that employ two neighbor lookahead, 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 all this, it should be remembered that the repositioning process dictates a smooth shape to the void, therefore eliminating most of the obstacles that otherwise exist in other routing solutions. A packet that reach an unmapped void can either wait for the void traversing process to complete, or it may be meanwhile routed 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 traversing direction is determined by the angle of the current node to the destination and choosing the direction which seems to minimize it. Thus, it may very well be that bypassing the void the other way around may be shorter by hops or distance, 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 towards 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.

B. *Mobility Effect on System Stability*

Ad-hoc networks are a 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 a top concern.

We have conducted a simulation of node movement, and checked the effect on nodes positioning as well other consequences. We used the simulation model described above, and added movement elements quite similar to those described in [21] and [22]: Every node in the system is assigned a random direction and speed to which it advances. Nodes are restricted

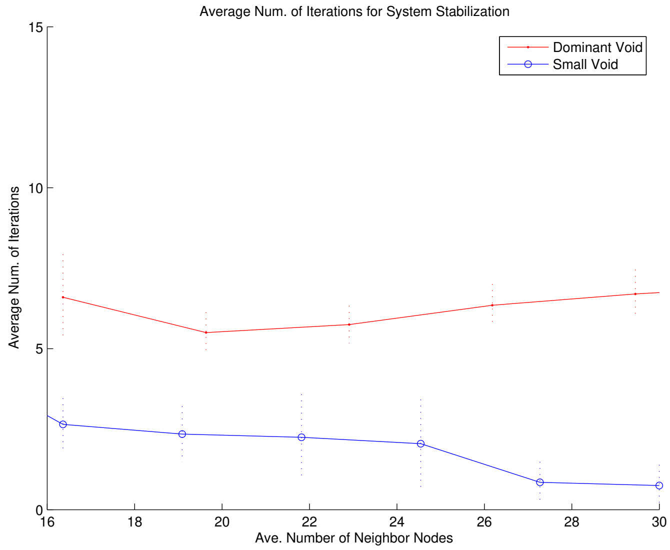

Fig. 16. Iterations Required for Entire System Stabilization

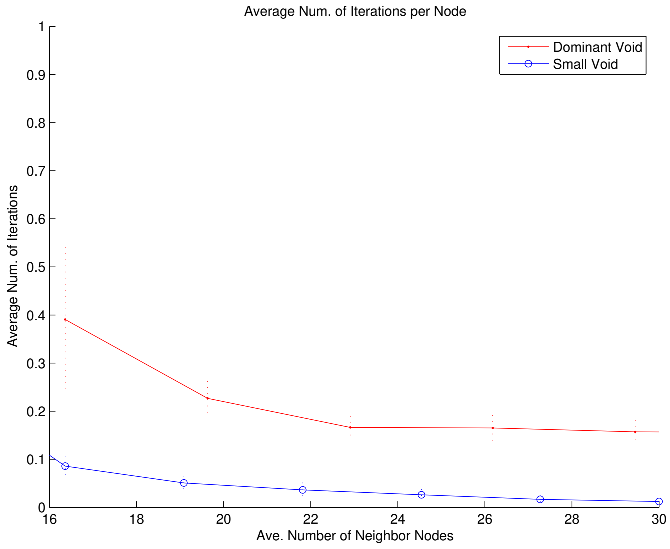

Fig. 18. Average Num. of Iterations per Node

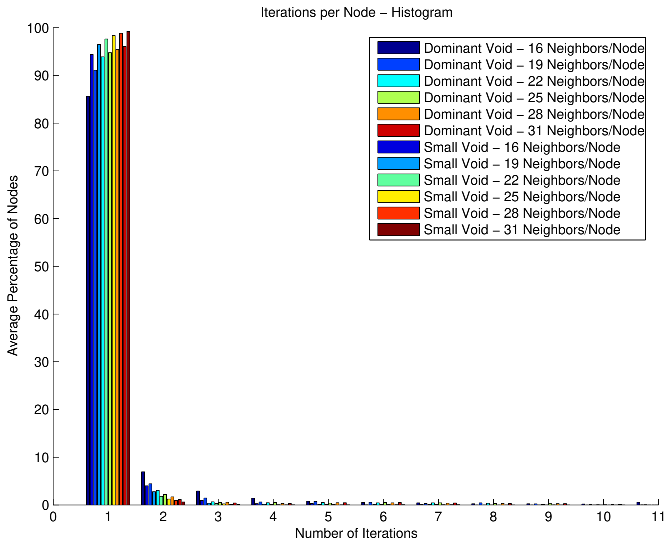

Fig. 17. Histogram of Node's Iterations by Network Density

from entering the forbidden zones that define the voids as well as crossing the system boundaries. The velocity of a node is uniformly distributed $(0, 25m/s]$. This models vehicular mobile nodes moving at a speed of up to $90Km/h$ with updating messages every second, or a person with a mobile device walking at $4.5Km/h$ with messaging updates every 20 seconds.

The system stabilizes very fast after repositioning. It requires less than 10 messaging cycles for the large void scenario and less than five for the small void scenario (Figure 16). For the majority of the nodes in the network, stabilization happens much faster, as can be seen by Figure 17, which show 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 nodes that require more than four iterations to converge contains less than 1% for the large void scenario and less than 0.1% for the small void scenario. The exception is the very sparse network with an average of 16 neighbors per node when the tail contains 2.25% and 1% for the large and small void scenarios, respectively. Figure 18 gives us an indication of the low overhead of elevation messaging. The overhead is comprised of one update message to every neighbor node and additional update messages for every local node iteration. As Figure 18 shows, this means less than 0.4 iterations 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 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 iterations per node, and N the number of nodes in the system. Based on our results we can bound the number of update messages in the network by $(1+i_n) \cdot n \cdot N < 1.4 \cdot n \cdot N$.

Figure 19 shows the average number of physical links up or down between every update session per node. As the change is not big and the number of broken links is quite the same as the new connections made, the system ability to stabilize grows. Figure 20 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% 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.

C. 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 I. An advantage of NEAR over

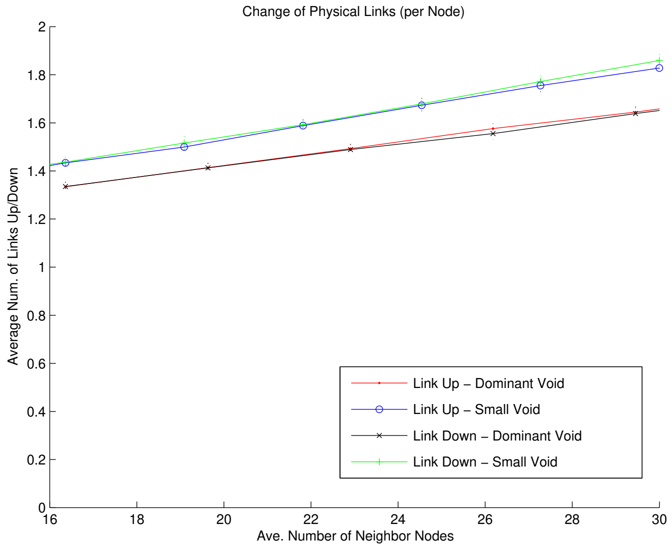

Fig. 19. Change of Physical Links (per Node)

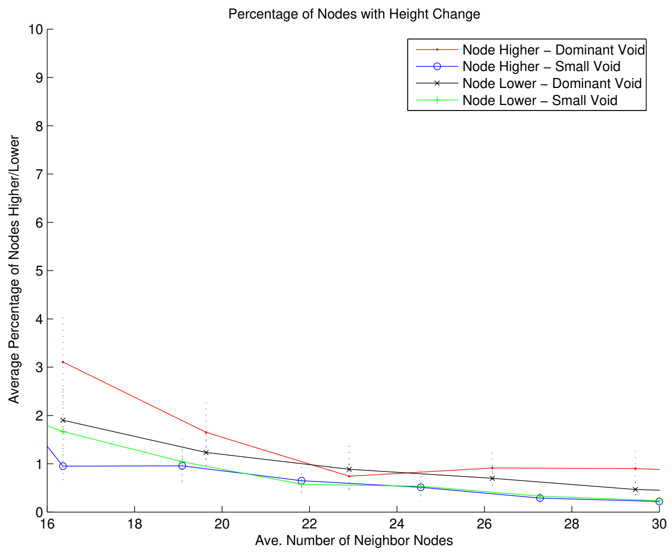

Fig. 20. Percentage of Nodes with Height Change

most solutions is the decrease in concave nodes percentage in the virtual network, which increase the greedy routing efficiency and reduce the need for recovery. An exception in this case is the Terminodes [11] 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 against the failure of single nodes [5].

A group of recovery algorithms, including GPSR [15], GFG [19], GOAFR [14], 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 reached void, using different techniques and are mostly based on planar graphs. The use of planar graphs require 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 a vertex should be removed or added to the graph. Practically, the resources required for planrization and NEAR are similar, however our solution performance is better.

There are two main advantages to NEAR over planarization algorithms: Its recovery routes are shorter and smoother and it uses guaranteed bypass routes. The sometimes "tangling" of the routing 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 node spreading one can not 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 Figure 1) which increases 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 traversing 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 for each packet to do 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 algorithm, even without the void discovery process.

The second group of recovery algorithms refer to algorithms which incorporate memory. Greedy/flooding [10] and Terminodes were already introduced, and differ greatly from NEAR. Two other solutions, closer to NEAR in their features are INF [18] and SAGF [23]. 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 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 spatial 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 possess the model information of the geographic space wherein it is located, keeping location information for intermediate nodes. When a message has to be sent, a route is calculated based on the spatial information, and using algorithms such as Dijkstra. The algorithm 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 way 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 based on more than 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 depends on the number of neighbors and does not grow with the network size, and yet remains aware to the network topography.

D. Future Work

The NEAR work has focused so far on use of constant radio range all across the network. As our work depends only on connection awareness between two neighbor nodes we expect no obstacles in adapting NEAR to work with different transmission radii, while the most probable element to be affected by the change is the thresholds.

We plan to extend further our dynamic scenarios, where nodes roam the network . In particular, we intend to study the effect of 'snapping', nodes quick change of virtual placement after losing a dominant neighbor, on the routing and repositioning algorithm.

Finally, we will study the trade offs in the design of the void discovery process. In particular, we need to increase the probability of a void to be discovered when it is created, while currently some void are discovered only when a crossing attempt is made.

REFERENCES

[1] C. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, 1994, pp. 234–244.

[2] C. Chiang, H. Wu, W. Liu, and M. Gerla, "Routing in clustered multihop, mobile wireless networks," in *IEEE SICON'97*, Apr. 1997, pp. 197–211.

[3] C. Perkins and E. Royer, "Ad-hoc on-demand distance vector routing," in *the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999, pp. 90–100.

[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 Mag.*, vol. 15, no. 6, pp. 30–39, 2001.

[6] D. Johnson, "Routing in ad hoc networks of mobile hosts," in *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, USA, 1994.

[7] S. Basagni, I. Chlamatac, V. Syrotiuk, and B. Wood, "A distance routing effect algorithm for mobility (DREAM)," in *the 4th Annual ACM/IEEE Int. Conf. on Mobile Computing and Networking (MOBICOM) '98*, Dallas, TX, USA, 1998, pp. 76–84.

[8] Y. Ko and N. H. Vaidya, "Location-aided routing (LAR) in mobile ad hoc networks," in *Mobile Computing and Networking*, 1998, pp. 66–75.

[9] I. Stojmenovic, "Position based routing in ad hoc networks," *IEEE Commun. Mag.*, vol. 40, no. 7, pp. 128–134, 2002.

[10] I. Stojmenovic and X. Lin, "Loop-free hybrid single-path/flooding routing algorithms with guaranteed delivery for wireless networks," *IEEE Trans. Parallel Dist. Sys.*, vol. 12, no. 10, pp. 1023–32, 2001.

[11] 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 Commun. Mag.*, pp. 166–175, 2001.

[12] E. Kranakis, H. Singh, and J. Urrutia, "Compass routing on geometric networks," in *11th Canadian Conference on Computational Geometry*, Vancouver, Canada, Aug. 1999, pp. 51–54.

[13] F. Kuhn, R. Wattenhofer, and A. Zollinger, "Asymptotically optimal geometric mobile ad-hoc routing," in *Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (Dial-M)*, 2002, pp. 24–33.

[14] ——, "Worst-case optimal and average-case efficient geometric ad-hoc routing," in *Proc. 4^{th} ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc)*, 2003.

[15] B. Karp and H. T. Kung, "Greedy perimeter stateless routing for wireless networks," in *the 6th Annual ACM/IEEE Int. Conf. on Mobile Computing and Networking (MobiCom 2000)*, Columbus, OH, USA, Aug. 2000, pp. 243–254.

[16] C. Lochert, H. Hartenstein, J. Tian, H. Fussler, D. Hermann, and M. Mauve, "A routing strategy for vehicular ad hoc networks in city environments," in *IEEE Intelligent Vehicles Symposium 2003*, Boston, MA, USA, June 2003.

[17] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *INFOCOM*, 1997, pp. 1405–1413.

[18] D. S. J. De Couto and R. Morris, "Location proxies and intermediate node forwarding for practical geographic forwarding," MIT Laboratory for Computer Science, Tech. Rep. MIT-LCS-TR824, June 2001.

[19] 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.

[20] B. Karp, "Geographic routing for wireless networks," Ph.D. dissertation, Harvard University, Cambridge, MA, USA, Oct. 2000.

[21] 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," in *the 4th Annual ACM/IEEE Int. Conf. on Mobile Computing and Networking (MOBICOM) '98*, Dallas, TX, USA, 1998, pp. 85–97.

[22] E. Royer, P. Melliar-Smith, and L. Moser, "An analysis of the optimum node density for ad hoc mobile networks," in *IEEE Int. Conf. on Communications*, Helsinki, Finland, 2001.

[23] J. Tian, I. Stepanov, and K. Rothermel, "Spatial aware geographic forwarding for mobile ad hoc networks," Stuttgart University, Tech. Rep., 2002.