Routing with uncertainty in the position of the destination

Evangelos Kranakis*, Danny Krizanc†, Lata Narayanan‡, Anup Patnaik§, Sunil Shende¶
*School of Computer Science
Carleton University, Ottawa, Canada
†Dept. of Mathematics & Computer Science
Wesleyan University, Connecticut, USA
‡Dept. of Computer Science & Software Engineering
Concordia University, Montreal, Canada
§Dept. of Computer Science & Software Engineering
Concordia University, Montreal, Canada
¶Dept. of Computer Science
Rutgers University, Camden, New Jersey, USA

***Abstract*— Position-based routing algorithms for mobile ad hoc networks utilize the position or location of the destination node to inform routing decisions. We consider the problem of routing in an ad hoc network where the source node knows the approximate position of the destination node, but is uncertain about its exact current location. We investigate two approaches to this problem: one, based on a traversal of the faces of a planar sub-graph of the graph representing the network, and the second, based on flooding a limited area of the graph that represents the region the destination is likely to be found. We propose several variants of both approaches, and do extensive simulations to analyze the performance of the algorithms. Our results indicate that a simple modification of the basic flooding approach yields the best trade-off for optimizing delivery rate, stretch factor, as well as transmission cost. If however, delivery is required to be guaranteed, then a variant of the face tree approach in [1] that we propose has the best performance.**

Keywords: Wireless networks, ad hoc networks, MANET, routing, geocasting, face traversal, flooding, greedy routing.

I. Introduction

Mobile ad-hoc networks (MANETs) have been the subject of intensive research in the last few years. A MANET is characterized by the lack of fixed network infrastructure. Hosts in the network can be mobile, and can communicate using wireless broadcasts with other hosts within their transmission range. Since all hosts may not be within the transmission range of each other, a protocol for multi-hop routing is required to ensure communication between any two hosts in a MANET. In general, the routing protocol cannot make any assumptions about the topology of the network, and so, mobile hosts have to build and update their routing tables automatically. If no assumption can be made about the location of the hosts in the network, routing protocols use a type of *flooding* of the network by control packets to obtain information about the network topology and to guarantee the delivery of messages [2], [3].

To reduce the amount of control traffic in MANETs, several authors have proposed the use of host *location information* [4], [5], [6], [7], [8]. We refer to the problem of routing from a source node s to a destination node d at position p by ROUTE(s, d, p). Algorithms that solve this problem are often called *position-based* routing algorithms. A survey of such algorithms can be found in [9]. In such algorithms, every node in the network is assumed to know the locations of itself and its neighbors. In addition, the position of the destination host is known to the source node, and is made available in the packet header. These algorithms limit the extent of flooding, but generally do not guarantee delivery. For instance, there are several scenarios where *greedy routing* [10], *compass routing* [11], or their combination are known to fail. One position-based algorithm that does guarantee delivery in a MANET is *face routing*, in which a planar sub-graph of the network is constructed locally, after which routing is performed using the right-hand rule to traverse those faces of the planar subgraph which are intersected by the line segment connecting the source to the destination [11]. Face routing can sometimes lead to very long paths in the graph. Bose *et al.* [5] propose a combination of greedy and face routing called *GFG* (also proposed by [6] as the GPSR routing protocol) that guarantees delivery of packets at the same time as reducing the length of paths.

All the routing algorithms above assume the use of exact location information about the destination node. This information can be obtained using a location service [12], or from messages previously received from the destination. But in both these cases, there can be inaccuracies in the position information. In the presence of such inaccuracies, the routing algorithm may have reduced rate of delivery. In particular, face routing can no longer guarantee delivery.

In this paper, we assume that the source node s knows the destination node d's position (x_0, y_0) at time t_0, but is interested in sending a packet to d at time t_1 where $t_1 > t_0$. If the maximum velocity of node d is v units per second, then the position of d at time t_1 is a point inside the circle with center (x_0, y_0) and radius $v(t_1 - t_0)$ units. This motivates the problem of routing with uncertainty in the position of the

1-4244-0495-9/06/$20.00 ©2006 IEEE

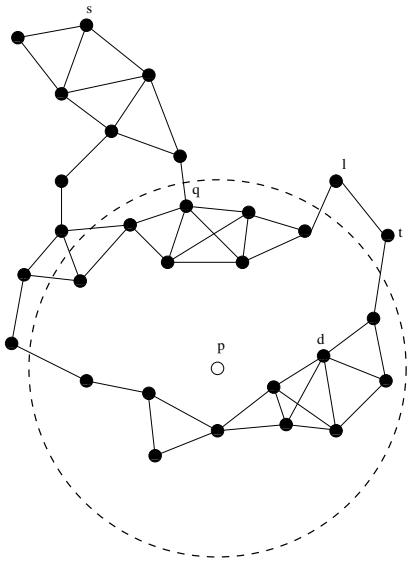

Fig. 1. An example where flooding within the uncertainty circle fails

destination. Instead of routing to a specific destination at a known position (x, y), we are interested in routing to a specific destination whose position is somewhere inside a circle with known center and radius. We refer to the problem of routing from a source node s to a destination node d contained in the circle of radius r centered at position p as ROUTE-U(s, d, p, r) and r as the uncertainty radius.

Related work

An obvious approach to solve ROUTE-U(s, d, p, r) is to use an algorithm to perform ROUTE(s, d, p), with an augmentation to stop if the destination node is found anywhere enroute. Since the destination may no longer be at position p, and may not be encountered enroute to the position p, the message may not be delivered. A second approach would be to ignore the imprecise location information and simply flood the entire network, but as stated above, this is resource-inefficient.

The problem of routing with uncertainty in the position of the destination is similar to, and is subsumed by, the problem of *geocasting*, where the objective is to route to *all* nodes in a given geographical region. Such an application may be especially relevant to sensor networks, where it may be necessary to broadcast a message to all nodes in a specific geographical area. In [13], the authors use a form of directed flooding, where packets are only forwarded in a rectangular zone defined as the smallest such region containing both the source and the geocast region. The authors of [14] give an algorithm which consists of reaching a node q contained in the circle with radius r centered at p, and then flooding all nodes contained in the circle. This approach also does not guarantee delivery, since it is possible that every path from q to d contains nodes outside the circle to which flooding is limited. See Figure 1 for an illustration of this phenomenon.

In [15], an algorithm that performs a traversal of a planar graph in $O(n^2)$ steps is given. The algorithm is based on constructing and traversing a tree of the faces of the planar graph. This was improved in [1], to an algorithm that can be used for geocasting with guaranteed delivery in a MANET in $O(n \log n)$ steps [5]. An important point to note is that these algorithms are *memoryless*, that is, no routing state needs to be stored at nodes, and no additional information needs to be carried in the packet. Recently, Stojmenovic [16] gives two additional algorithms for geocast that guarantee delivery. The first algorithm is essentially flooding inside the circle, augmented by face traversals initiated by *inside border* nodes; this approach was also outlined in [14]. In the second algorithm, the packet is sent simultaneously to a grid of points just outside the circle, which then initiate flooding inside the circle.

Our contributions

In this paper, we investigate the performance of flooding-based and face-tree traversal-based approaches to the problem of routing with uncertainty. To this end, we propose several variants of both basic flooding and the face tree traversal algorithm given in [1], [5]. Based on their memory requirements, we classify the algorithms into *memoryless*, *constant memory* and *non-constant memory*. The memoryless algorithms do not maintain any routing state at the nodes. The constant memory algorithms require constant amount of memory at every node for maintaining routing state. The non-constant memory algorithms require non-constant memory for the routing state at every node. In all algorithms, we assume that we are allowed to include $O(\log n)$ bits of information in the header of the packet for routing purposes.

We study the effect of the number of nodes and the size of the uncertainty radius on the *delivery rate*, *stretch factor*, and the *transmission cost* of the algorithms. The delivery rate is the percentage of packets that get transmitted successfully to the destination. The stretch factor is the number of hops taken by a packet compared to the minimum hop path available in the network, averaged over all successfully delivered packets. The transmission cost is the ratio of total number of times that copies of the packet get transmitted in the course of successful delivery of the packet to the number of transmissions in the minimum hop path, averaged over all successfully delivered packets. It is a measure of the energy costs of the algorithm.

While face tree traversal-based approaches are guaranteed to deliver the packets, flooding-based approaches may fail in some cases. On the other hand, flooding can be expected to have much better stretch factor. These general trends are confirmed by our experimental results. The surprising findings of our experiments are listed below:

- Flooding-based algorithms show an interesting behavior whereby the delivery rate first decreases and then increases as the uncertainty radius increases. We give an explanation for this in Section III.
- A simple augmentation of flooding, that we call EXTENDED SN FLOODING , achieves very high delivery rate, at the same time as achieving very low stretch factor, and a drastically reduced transmission cost.

- While the accepted wisdom is that flooding is very resource-inefficient, and would have a high transmission cost, our results show that some variations of the face tree approach, including the version given in [5] have a very high transmission cost as well. Indeed, there is considerable overlap between the transmission cost profiles of the two approaches. In particular, the cheapest algorithms among the ones studied are EXTENDED SN FLOODING and SN FLOODING , while the two most expensive algorithms are face-tree based algorithms.
- The difference between geocasting and ROUTE-U is highlighted by the fact that a technique that provably improves the performance for geocasting appears to *degrade* the performance for ROUTE-U.
- The original face tree based approach is memoryless as compared to the flooding-based approaches which require a constant amount of routing state at nodes. However, an obvious modification of FACE TREE that uses extra memory does not yield much benefit. In particular, its performance is still worse than the best flooding approach in our experiments.

Organization of the paper

In Section II, we present the algorithms we propose to study in this paper. Section III gives our experimental results, and we conclude with some discussion in Section IV.

II. THE ALGORITHMS

In this section, we describe several algorithms to solve the problem ROUTE-U(s, d, p, r). We assume that the network can be represented by a unit disk graph (UDG). For convenience, we refer to the circle of radius r centered at p as the *uncertainty circle* and r as the *uncertainty radius*. All our algorithms have two phases. In the first phase, we attempt to reach any node in the uncertainty circle. Any algorithm can be used for this; in our experiments, we use the GFG algorithm proposed in [5] to solve the problem ROUTE(s, d, p). We follow the path given by this algorithm until we reach a node, say q, inside the uncertainty circle. In the second phase, we attempt to find a path from q to d, using either a face tree traversal based approach or a flooding-based approach as described below. In the following, for each algorithm, the worst-case complexity is given as a function of n, the number of nodes in the network, and is an upper bound on the number of hops traversed during the *second phase* of the algorithm. Detailed pseudocode for all algorithms can be found in [17].

A. Face Tree Traversal Approaches

The geocasting algorithm given in [5] employs GFG routing on the problem ROUTE(s, d, p) until reaching a node inside the circle. At this point, it constructs a tree of all the faces intersecting and contained in the region, and traverses the tree in depth first order as described in [1]. The traversal of the face tree is followed until returning back to the start edge. This algorithm can clearly be used to solve the problem ROUTE-U(s, d, p, r).

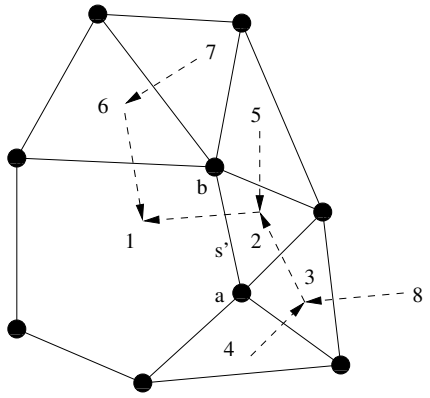

Fig. 2. An example showing the face tree for a planar graph with (a, b) as the start edge. The arrows moving from child to parent face, passing through the entry edges, represent the edges of the face tree.

Since some of the variants of this algorithm that we describe depend on further details of the face traversal algorithm, we give them here. We start with an arbitrary fixed point s' in the plane. Every face has a unique *entry edge*, which is the edge that minimizes a certain function f over all edges in the face. A first approximation of this function f is the distance from the edge to the point s'. In [1], Bose and Morin define a relationship on the faces of the planar graph using the entry edges. For any face f they define its parent f' as the other face that the entry edge for f belongs to. Based on this relationship they define a *face tree* as the spanning tree of all the faces of the graph. A traversal of the face tree then would result in visiting all the nodes in the graph. An example showing the face tree of a planar graph can be seen in Figure 2.

Bose and Morin also give an innovative doubling approach to determine if a given edge e is the entry edge. Essentially, starting with $d = 1$, we traverse d edges to the left of e, then $2d$ edges to its right, and so on, until an edge e' is encountered such that $f(e') < f(e)$ (which confirms that e is not the entry edge), or until we are sure that all edges of the face have been seen (which confirms that e *is* the entry edge).

We proceed to describe several variants of the above algorithm. All these algorithms utilize only the edges of planar subgraph of the unit disk graph representing the network. In our simulations we use the Gabriel Graph algorithm [18] for planarization. This algorithm computes the edges belonging to the planar subgraph incident on a given node in constant time using only local information.

1) DOUBLING FACE TREE *:* This is essentially the algorithm for geocasting given in [1], modified to stop as soon as the destination is encountered. To reduce the stretch factor, we make another slight change to the algorithm. We store the entry edge for the current face in the packet once it is found. Hence, the entry edge need not be computed again as long as we traverse the same face. For the example in Figure 2, the faces are traversed in the order 1, 2, 3, 4, 3, 8, 3, 2, 5, 2, 1, 6, 7, 6, 1. This algorithm is memoryless with a worst-case complexity of $O(n \log n)$.

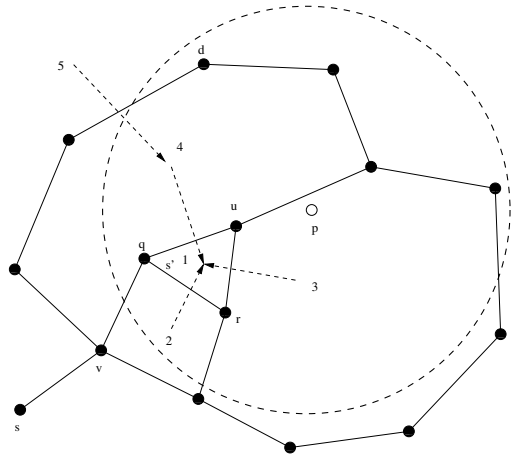

Fig. 3. An example where the path taken by DOUBLING FACE TREE is much longer than that taken by DFS FACE TREE .

2) DFS FACE TREE *:* In the DFS FACE TREE algorithm, the faces of the planar graph are traversed in the same order as in the DOUBLING FACE TREE algorithm. However, the entry edge computation is different. In the DFS FACE TREE algorithm, to determine if a given edge is the entry edge, we simply do a left-hand based traversal of the entire face as in [15]. As in the DOUBLING FACE TREE algorithms, we store the entry edge for the current face in the packet to avoid doing the same computation a second time. This algorithm is also memoryless and has a worst-case complexity of $O(n^2)$.

Interestingly, as our experimental results show, there are many situations where DFS FACE TREE has smaller stretch factor than the DOUBLING FACE TREE algorithm. The reason is that DFS FACE TREE can sometimes end up encountering the actual destination while simply doing its entry edge computation, which involves traversing the entire face. Note that when used for geocasting, DFS FACE TREE can never have a better performance than the DOUBLING FACE TREE algorithm.

An example of this phenomenon is shown in Figure 3. The start edge is (q, r). DFS FACE TREE finds the destination d while checking if (q, v) is an entry edge for face 4, while visiting face 2. On the other hand, DOUBLING FACE TREE quickly determines that (q, v) is not the entry edge for face 4, therefore visits face 2 then returns to face 1 then visits face 3 then returns to face 1 and now while checking if edge (u, q) is an entry edge for the opposite face it finds the destination d. For this example, the length of the path found by DOUBLING FACE TREE is 56 hops while that found by DFS FACE TREE is 14 hops.

3) MARK ENTRY EDGE FACE TREE *:* In this version of face tree traversal, faces are traversed in the same order as in the previous two algorithms, but the entry edge computation is different. In the MARK ENTRY EDGE FACE TREE algorithm, *first-time* entry edge computation is done in the same way as in the DFS FACE TREE algorithm, that is, by traversing the entire face. At this point, however, we *mark* the entry edge. This means the next time we enter the face, we do not need to do the entry edge computation, which potentially decreases the number of hops required. Hence, as shown in Figure 2, when the packet returns to face 3 from face 4, the entry edge for the face has already been computed and saved. This is unlike the DOUBLING FACE TREE and the DFS FACE TREE algorithms where the entry edge will need to be computed again. The marking of the entry edges requires storing of $O(d)$ routing state at the nodes, where d is the maximum degree of a node. The worst-case time complexity is $O(n)$.

4) BFS FACE TREE *:* In this algorithm, the face tree is traversed in breadth first order. For the example in Figure 2. the faces are traversed in the order 1, 2, 1, 6, 1, 2, 3, 2, 5, 2, 1, 6, 7, 6, 1, 2, 3, 4, 3, 8, 3, 2, 1. This algorithm is memoryless but requires a queue with the addresses of $O(n)$ nodes to be carried around in the packet. The worst-case complexity is $O(n^3)$.

5) $\|$-FACE TREE *:* This is a parallelized version of the Face Tree algorithm. While traversing a face, if a given edge is the entry edge for the opposite face, then another copy of the packet is spawned to explore that face. Each packet continues the traversal until either finding the destination or until completing the traversal of a face. For the example in Figure 2, first face 1 is traversed, then faces 2 and 6 in parallel, then faces 3 and 5 in parallel, and finally faces 7, 4 and 8 in parallel. The algorithm is memoryless and has a stretch factor of $O(n)$ in the worst case.

B. Flooding-based approaches

In all the flooding-based algorithms, we follow the same path as GFG routing on the problem ROUTE(s, d, p), until a node inside the region is reached. This node then broadcasts the packet to all its 1-hop neighbors. Any node receiving this packet broadcasts it if and only if it lies inside the region. Any further copies of the packet are ignored. Also, in flooding-based algorithms, unlike the face tree traversal-based algorithms, all the edges in the unit disk graph are traversed.

In all the flooding-based approaches, a unique identifier corresponding to the packet to be flooded must be stored at each node, to prevent re-transmission of the same packet *ad infinitum*. This requires a per packet constant memory of routing state to be maintained at the nodes. In the cases when the packet is delivered, the algorithm always finds the shortest path in the second phase.

1) AN FLOODING*:* We call the basic algorithm described above All-neighbor (AN) Flooding.

2) SN FLOODING *:* The difference between Subset Neighbor (SN) Flooding and AN Flooding is that nodes inside the region, instead of broadcasting the packet to all 1-hop neighbors, forward it to only a subset of 1-hop neighbors whose neighbors in turn include all 2-hop neighbors of the original node. Computing a minimal subset of such 1-hop neighbors is known to be an NP-complete problem for arbitrary graphs, but its complexity for unit disk graphs is unknown [19]. Hence we use a greedy algorithm to compute the subset: we iteratively select a 1-hop neighbor that covers the maximum number of

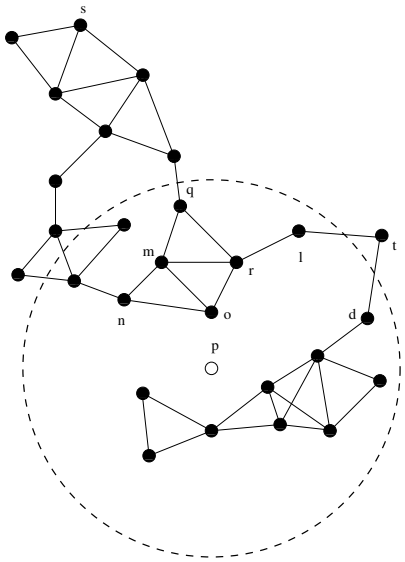

Fig. 4. An example where SN FLOODING within the uncertainty circle fails but AN FLOODING succeeds.

2-hop neighbors (in the uncertainty region) not yet covered, and stop when all 2-hop neighbors have been covered.

The SN FLOODING algorithm reduces the transmission cost by restricting flooding of packets, however, it fails to deliver the packet in some situations where AN FLOODING succeeds. As seen in Figure 4, the only path from s to d goes via the node t. However, the SN FLOODING algorithm would not send the packet to node t, which results in a failure of delivery. In contrast, in AN FLOODING all nodes inside the region flood to all their neighbors. Hence, the packet would be sent to t which in turn would deliver it to d, its 1-hop neighbor.

3) EXTENDED AN FLOODING*:* In order to discover paths to the destination, we extend the region of flooding by a constant value in this algorithm. Extended AN Flooding on the problem ROUTE-U(s, d, p, r) is identical to flooding on the problem ROUTE-U$(s, d, p, r + \lambda)$ where λ is the transmission radius of nodes. Therefore, in Figure 1, the packet will be sent to nodes l and t thereby reaching the destination d.

4) EXTENDED SN FLOODING *:* In this algorithm, we extend the uncertainty radius of SN FLOODING . That is, Extended SN Flooding on the problem ROUTE-U(s, d, p, r) is the same as SN Flooding on the problem ROUTE-U$(s, d, p, r+\lambda)$.

III. SIMULATION RESULTS

To evaluate the relative performance of these algorithms we will consider their packet delivery rates, stretch factors, and transmission costs. We first describe our simulation environment, and then describe and interpret our results, comparing our algorithms with each other.

A. Simulation Environment

In the simulation experiments, a set S of n points (where $n \in \{75, 100, 125\}$) is randomly generated on a rectangle of 800m by 700m. For the transmission range λ of nodes, we use 120m. After generating $G = UDG(S)$, a source and destination node is randomly chosen. If there is no path from s to d in $UDG(S)$, the graph is discarded; otherwise, all routing algorithms are applied on G. This process is repeated for 100 node-pairs on each graph, and then for 1000 graphs. The delivery rate is the percentage of packets that get transmitted successfully to the destination on a valid graph. The stretch factor is the number of hops taken by a packet compared to the minimum hop path available in the network, averaged over all successfully delivered packets. The transmission cost is the ratio of total number of times that copies of the packet get transmitted in the course of successful delivery of the packet to the number of transmissions in the minimum hop path, averaged over all successfully delivered packets.

B. Face tree traversal-based algorithms

The graphs in Figures 5, 6, and 7 present the results for face tree traversal-based algorithms for 75, 100, and 125 nodes, respectively.

The delivery rate is 1 for all face tree traversal-based algorithms and is therefore not shown here. The stretch factor of ‖-FACE TREE is best followed by BFS FACE TREE , DFS FACE TREE , MARK ENTRY EDGE FACE TREE , which all have similar performance followed by DOUBLING FACE TREE which has the worst performance. While all others are quite close, the doubling algorithm's performance is significantly worse.

The stretch factor of all algorithms increases with increasing uncertainty radius, especially the non-parallelized algorithms. This is because as the uncertainty radius increases, more and more of the graph is being searched using the face tree traversal method. In particular, there is a greater chance of the exterior face being traversed.

Somewhat surprisingly, using marked bits does not help in reducing the stretch factor for smaller values of uncertainty radius. At higher values of uncertainty radius, the MARK ENTRY EDGE FACE TREE implementation starts to outperform all face tree traversal-based algorithms except ‖-FACE TREE .

The transmission cost of BFS FACE TREE , MARK ENTRY EDGE FACE TREE , and DFS FACE TREE is much lower than that of ‖-FACE TREE or DOUBLING FACE TREE . For all algorithms, as the number of nodes increases, the stretch factor and transmission cost both increase. In particular, the transmission cost of ‖-FACE TREE becomes intolerable.

We conclude that the performance of DFS FACE TREE , BFS FACE TREE , and MARK ENTRY EDGE FACE TREE are all similar, and they use the least energy, while ‖-FACE TREE achieves the best stretch factor.

C. Flooding-based algorithms

The graphs in Figures 8, 9, and 10 present the delivery rate, stretch factor, and transmission cost for flooding-based algorithms respectively for 75 and 125 nodes. The results for 100 nodes are quite similar to those for 75 nodes and can be found in [17].

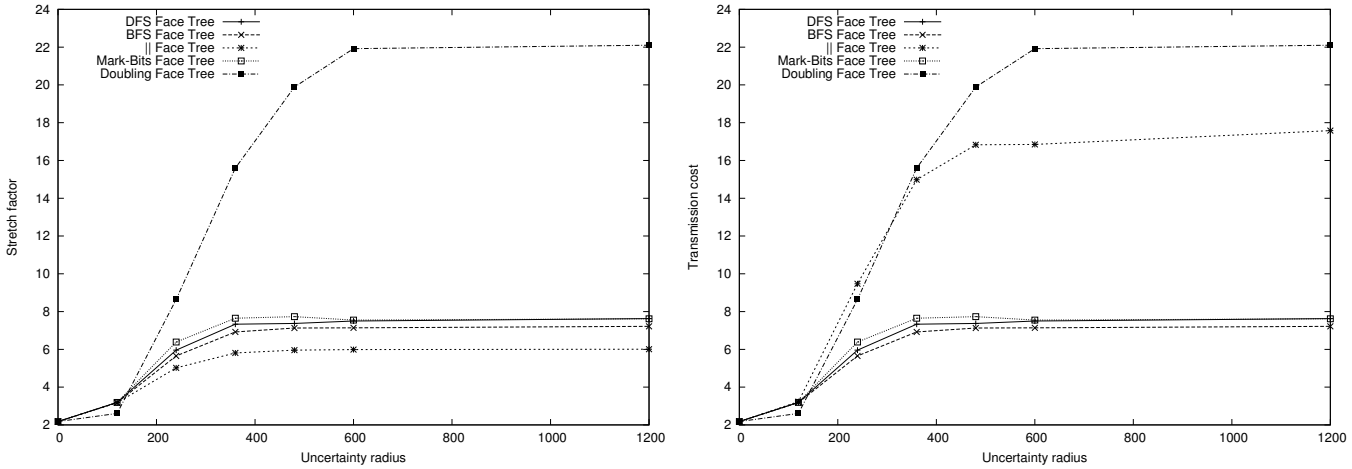

Fig. 5. Face tree traversal-based algorithms: Stretch factor and transmission cost for varying uncertainty radius. Number of nodes= 75.

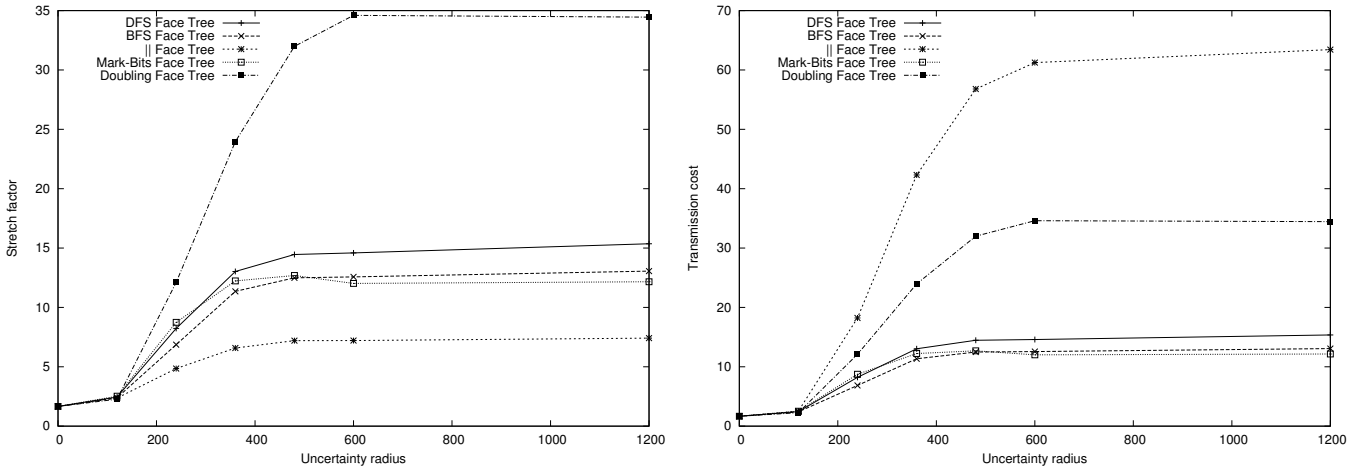

Fig. 6. Face tree traversal-based algorithms: Stretch factor and transmission cost for varying uncertainty radius. Number of nodes= 100.

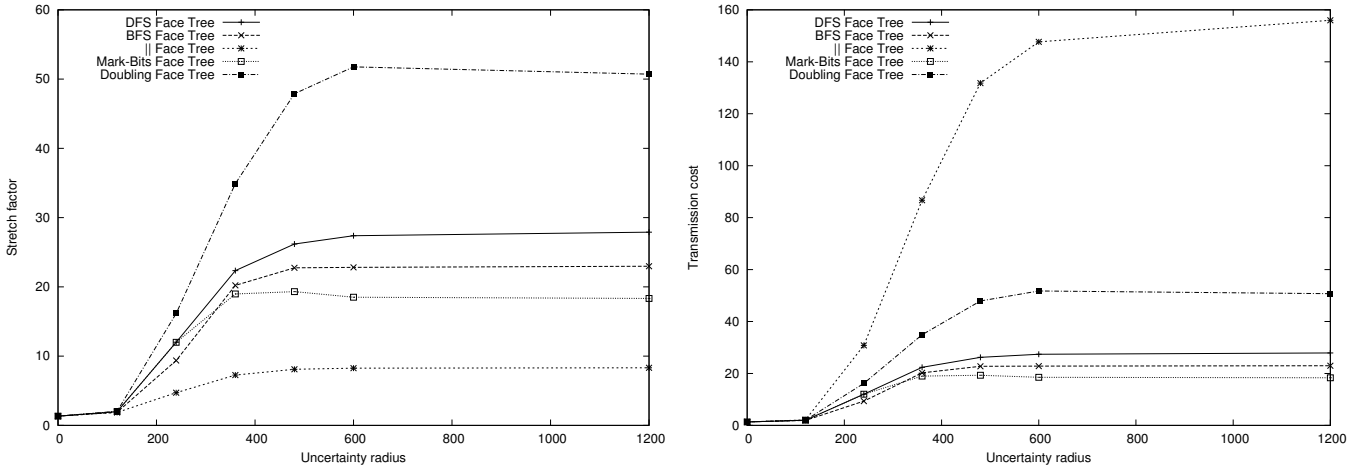

Fig. 7. Face tree traversal-based algorithms: Stretch factor and transmission cost for varying uncertainty radius. Number of nodes= 125.

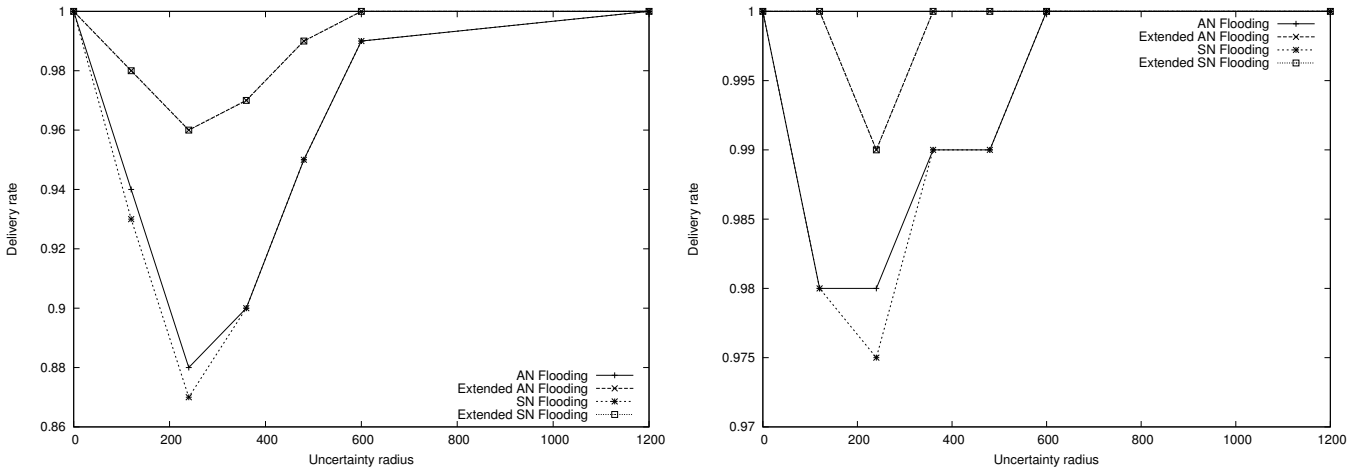

Fig. 8. Flooding-based algorithms: Delivery rate for varying uncertainty radius. Number of nodes = 75 (left) and 125 (right).

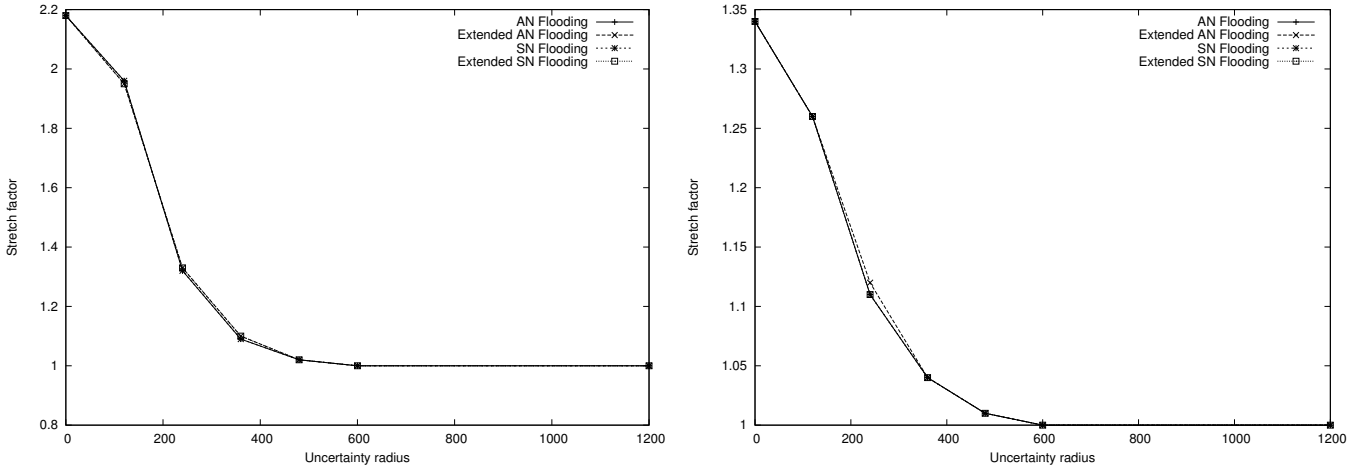

Fig. 9. Flooding-based algorithms: Stretch factor for varying uncertainty radius. Number of nodes = 75 (left) and 125 (right).

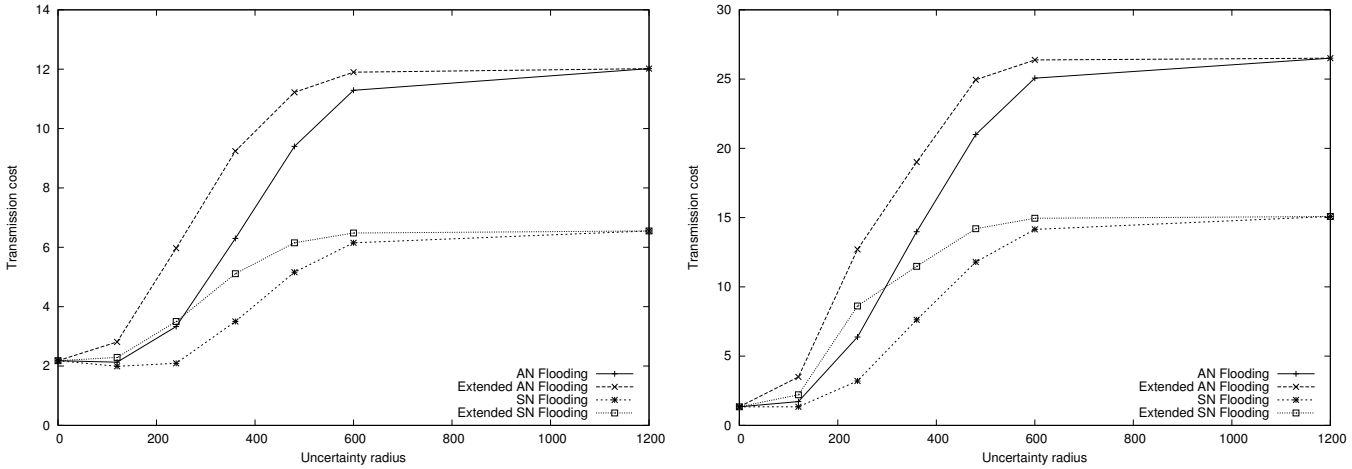

Fig. 10. Flooding-based algorithms: Transmission rate for varying uncertainty radius. Number of nodes = 75 (left) and 125 (right).

It is interesting to note that all flooding-based algorithms have a dip in the delivery rate at around $r = 2\lambda$ where λ is the transmission radius of nodes. This can be explained by the following observation. When $r \leq \lambda/2$, all nodes in the uncertainty circle are directly connected to each other, therefore flooding must succeed (this is confirmed by the experiments). Similarly, when the uncertainty radius is large enough that the entire field is contained in the uncertainty circle, flooding is guaranteed to succeed, since we only use connected graphs. However, as the uncertainty radius increases from $\lambda/2$, the chance of having two disconnected components inside the uncertainty circle first increases, and then again decreases as the number of nodes inside the circle increases.

EXTENDED AN FLOODING improves the delivery rate, but cannot guarantee delivery for all values of uncertainty radius, though always over 95% for all values studied.

The stretch factor is almost the same for all flooding-based approaches. The stretch factor is in general very good, always less than 1.25, and exactly 1 for uncertainty radius $>= 5\lambda$. As noted earlier, the flooding-based algorithms find the shortest path in the second phase when the packet is successfully delivered. Thus, an average stretch factor greater than one reflects the fact that the shortest path may not always be found in the first phase where GFG is being used to reach the uncertainty zone. As the uncertainty radius increases, the part of the path which is constructed in the first phase is proportionally smaller; this explains why the stretch factor decreases as the uncertainty radius increases as seen in Figure 9. Also, the stretch factor decreases as the number of nodes increases; this is in line with the well-known behavior of GFG.

The transmission cost of SN FLOODING and EXTENDED SN FLOODING are similar and much lower than other flooding-based approaches. The cost of AN FLOODING is higher, and the cost of EXTENDED AN FLOODING is even higher.

Extended SN flooding appears to achieve a good balance between delivery rate and transmission cost.

IV. DISCUSSION

It is clear from our simulations that the flooding-based approaches have much better stretch factor than the face-tree approaches. However, many of the face tree-based algorithms examined here are memoryless, and as such are not comparable to the flooding-based algorithms, which all require routing state to be maintained at nodes. One interesting finding is that storing routing state as in MARK ENTRY EDGE FACE TREE does not seem to improve the stretch factor except when the uncertainty is very high.

Flooding is known to be resource-inefficient. However, our experiments show that the transmission cost of AN FLOODING, the most expensive algorithm in the flooding-based class, while worse than most of the face tree based approaches, is not significantly higher, and indeed, is better than the $\|$-FACE TREE and DOUBLING FACE TREE algorithms. At the other end, the transmission costs of SN FLOODING and EXTENDED SN FLOODING are lower than the cheapest Face-tree algorithm. Thus, making minor adjustments to the basic flooding algorithm results in greatly reduced transmission cost while not sacrificing the delivery rate or the stretch factor. Meanwhile, the high stretch factor of the face-tree based approaches also translates to a high transmission cost.

In conclusion, if marked bits are not practical, or if guaranteed delivery is required, then DFS FACE TREE would seem to be best approach of the ones studied here, but otherwise, EXTENDED SN FLOODING would be the best choice.

REFERENCES

[1] P. Bose and P. Morin, "An improved algorithm for subdivision traversal without extra storage," *International Journal of Computational Geometry and Applications*, vol. 12, no. 4, pp. 297–308, 2002. Special issue of selected papers from the *11th Annual International Symposium on Algorithms and Computation (ISAAC 2000)*.

[2] V. Park and S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proceedings of INFOCOM'97*, pp. 1405–1413, 1997.

[3] J. Broch, D. Johnson, and D. Maltz, "The dynamic source routing protocol for mobile ad hoc networks," *Internet-draft, draft-ietf-manet-dsr-00.txt*, 1998.

[4] S. Basagni, I. Chlamtac, V. Syrotiuk, and B. Woodward, "A distance routing effect for mobility (dream)," in *4th ACM/IEEE Conference on Mobile Computing and Networking (MobiCom '98)*, pp. 76–84, 1998.

[5] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," *Wireless Networks*, vol. 7, pp. 609–616, 2001.

[6] B. Karp and H. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *6th ACM Conference on Mobile Computing and Networking (MobiCom '00)*, pp. 243–254, 2000.

[7] Y.-B. Ko and N. Vaidya, "Location-aided routing (LAR) in mobile ad hoc networks," in *4th ACM/IEEE Conference on Mobile Computing and Networking (MobiCom '98)*, pp. 66–75, 1998.

[8] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, "Geometric ad-hoc routing: Of theory and practice," in *Proc. of the 22nd ACM Symposium on the Principles of Distributed Computing (PODC)*, pp. 63–72, July 2003.

[9] S. Giordano and I. Stojmenovic, "Position based routing algorithms for ad hoc networks:a taxonomy," *Ad Hoc Wireless Networking, X. Cheng, X. Huang and D.Z. Du (eds.)*, pp. 103–136, 2004.

[10] X. Lin and I. Stojmenović, "Gedir: Loop-free location based routing in wireless networks," in *IASTED Int. Conf. on Parallel and Distributed Computing and Systems (PDCS '99)*, pp. 1025–1028, 1999.

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

[12] T. Camp, J. Boleng, and L. Wilcox, "Location information services in mobile ad hoc networks," in *Proceedings of the IEEE International Conference on Communications (ICC '02)*, pp. 3318–3324, 2002.

[13] Y.-B. Ko and N. H. Vaidya, "Geocasting in mobile ad hoc networks: Location-based multicast algorithms.," in *WMCSA*, pp. 101–110, 1999.

[14] K. Saeda and A. Helmy, "Efficient geocasting with perfect delivery in wireless networks," in *Proceedings of WCNC*, 2004.

[15] M. de Berg, R. van Oostrum, and M. Overmars, "Simple traversal of a subdivision without extra storage," in *SCG '96: Proceedings of the twelfth annual symposium on Computational geometry*, pp. 405–406, 1996.

[16] I. Stojmenovic, "Geocasting with guaranteed delivery in sensor networks," *IEEE Wireless Communications Magazine*, vol. 11, pp. 29–37, December 2004.

[17] A. Patnaik, "Routing protocols for ad hoc networks with uncertainty in the position of the destination," Master's thesis, Concordia University, 2006.

[18] K. Gabriel and R. Sokal, "A new statistical approach to geographic variation analysis," *Systematic Zoology*, vol. 18, pp. 259–278, 1969.

[19] G. Calinescu, I. Mandoiu, P. J. Wan, and A. Z. Zelikovsky, "Selecting forwarding neighbors in wireless ad hoc networks," *Mobile Networks and Applications*, vol. 9, no. 2, pp. 101–111, 2004.