Computer Science and Engineering 2013, 3(1): 1-7
DOI: 10.5923/j.computer.20130301.01

Wireless Routing in Ad-Hoc Networks

Tzvetalin S. Vassilev*, **Brian Eades**

Department of Computer Science and Mathematics, Nipissing University, North Bay, Ontario, P1B 8L7, Canada

Abstract Wireless networks present many challenges to standard routing algorithms. Ultimately, what we want from a routing algorithm is for it to be optimal in terms of robustness, scalability, power, and time; however, it will be shown that guaranteed delivery generally comes at the expense of any one of these desirables. This paper will exhibit a progression from routing in static networks to routing in unit distance wireless networks in order to illuminate the reality of the balance between what we want from wireless ad hoc routing algorithms and what we can expect from them. Much of the analysis presented will be on Dijkstra's algorithm, the Bellman-Ford algorithm, Compass Routing, Face Routing, as well as localized methods to extract planar subgraphs.

Keywords Wireless Routing, Unit Distance Wireless Networks, Dijkstra's algorithm, Bellman-Ford algorithm, Compass Routing, Face Routing

1. Introduction

The ongoing advancements in wireless technology have allowed for many different devices (old and new) to communicate in a wireless network at a declining cost. While some of these networks may consist purely of stationary devices, mobility is the primary advantage of wireless communication. Thus the underlying mechanisms that dictate how a packet is sent from one device to another should reflect not only our understanding of how wireless mobile devices operate, but also our desire for optimal performance. The representation of networks as geometrical graphs provides the foundation for ingenuity in routing protocols. Graphs provide insight into both the optimal aspects of a routing algorithm as well as its limitations. To this end, a thorough examination of several routing algorithms along with their inherent structures and optimality aspects will be presented, beginning with the classical approach in static networks and ending with methods used in homogenous wireless networks.

2. Classical Approaches

While it will be shown that the following static routing algorithms are not appropriate for wireless networks in general, the underlying methods and objectives provide the foundation to which several effective routing protocols are based on[1].

* Corresponding author:
tzvetalv@nipissingu.ca (Tzvetalin S. Vassilev)
Published online at http://journal.sapub.org/computer

2.1. Link-State Protocol: Dijkstra's Algorithm

The algorithm presented below produces the shortest path from a source s to all possible destinations v on a directed graph G with non-negative edge weights w. Its implementation in networking is known as a link-state algorithm[2, 3]. Let $|G|$ denote the number of nodes in the graph. Let $D(v)$ be the total weight of the shortest path from the source s to destination v. Let $c(u,v)$ be the cost of routing from u to v (i.e., the sum of all edge weights along the path from u to v. Finally, let N contain the vertices in G whose shortest paths have been determined.

```
Algorithm DIJKSTRA LS$(G,w,s)$
Input: a graph $G$ (represented by an adjacency list), non-negative edge weights $w$, and source $s$.
Output: weight of shortest path from source $s$ to all destinations $v$.
1. $N = \{s\}$
2. **for** all destinations $v \in G$
3. **if** $v$ is adjacent to $s$
4. **then** $D(v) = c(s,v)$
5. **else** $D(v) = \infty$
6. **for** $i := 1$ to $|G| - 1$
7. Amongst all nodes $n \notin N$ adjacent to any
8. $v \in N$, add $n$ to $N$ such that
9. $D(v) + c(v,n)$ is a minimum.
```

The structure of the above algorithm comes from[2], while a more detailed analysis of its functionality and correctness is outlined in[3]. Lines 6 through 9 are known as relaxing an edge. A total of $|G| - 1$ edges are relaxed, each requiring $O(|G|)$ computations, which results in an overall complexity of $O(|G|^2)$. The implementation of a binary heap for the priority queue improves the relaxation computation to $O(\log|G|)$and thus the overall complexity to $O(|G|\log|G|)$.

Regardless, the implementation of link-state protocols generally requires global state information from the graph. That is, the topology and link-costs are known to every node, which culminates in one exhaustive routing table that is maintained between all nodes. Hence a small change in the topology of the system can propagate into large errors in the routing table. Even with the available global state information, routing loops can still occur; for instance when two nodes a and b attempt to route through each other to get to a destination d. An example of this event will be provided in the next section, as the directed distance vector routing protocol is also sensitive to changes in topology. Thus the use of Dijkstra's algorithm in wireless ad hoc networks is ill suited due to its static nature, time-complexity, and the overhead requirements it imposes on a network by requiring global state information.

2.2. Directed Distance Protocol: The Bellman-Ford Algorithm

The directed distance-vector routing protocol is nearly identical to the Bellman-Ford algorithm – but with one modification: the relaxation phase uses an infinite while loop in order to achieve a quiescent state that responds to link-cost changes or updated distance vectors[2].

A major distinction between link-sate and directed distance-vector routing algorithms is the amount of information that is available to a given node in a network. The algorithm below uses localized information of a source to calculate the distance vector to each of its neighbours. Once each node has calculated its distance-vector, it is broadcasted to each of their respective neighbours. The result is a minimum weight-spanning tree.

```
Algorithm $\textsc{Bellman-Ford DDV}(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
Input: a graph $\boldsymbol{G}$ (represented by an adjacency list), edge weights $w$, and source $s$.
Output: weight of shortest path from source $s$ to all destinations $v$.
1. $D(s) = 0$
2. for all destinations $v \in G$
3.     if $v$ is adjacent to $s$
4.         then $D(v) = c(s,v)$
5.             send $D(v)$ to all adjacent nodes $v$
6.         else $D(v) = \infty$
7. while (there exists a link cost change or updated distance vector to some adjacent $v$)
9.     do: for all $n \in G$,
10.        $D(n) = \min\{D(n), D(v) + c(v,n)\}$
11.            send updated distance vector: $D(v) = [D(v): adjacent\ v \in G]$ to all adjacent nodes $v$
14.        if $D(s) > D(v) + c(v,s)$
15.            then report negative weight cycle
16.                break
```

Figure 1 illustrates how the Bellman-Ford algorithm produces the shortest path in the given graph with A as the source. Figure 2 exemplifies the distributive and asynchronous nature of the algorithm where each node maintains its own routing table and sends updated distance vectors to neighbouring nodes.

Note that the purpose of lines 14 to 16 is to check for negative weight cycles within the graph. This may be unnecessary if all edge weights are non-negative, but depending on what the routing costs are representing, negative weights may be unavoidable. For instance, they might signify the load carried or removed from a packet as it travels from a source to a destination. It is simply one of many preventative measures against routing loops. Unfortunately, the Bellman-Ford algorithm can converge very slowly, and despite its ability to accept changes in topology it is prone to routing loops. An example of how a link-cost change can result in a routing loop is shown in Figure 3.[2]

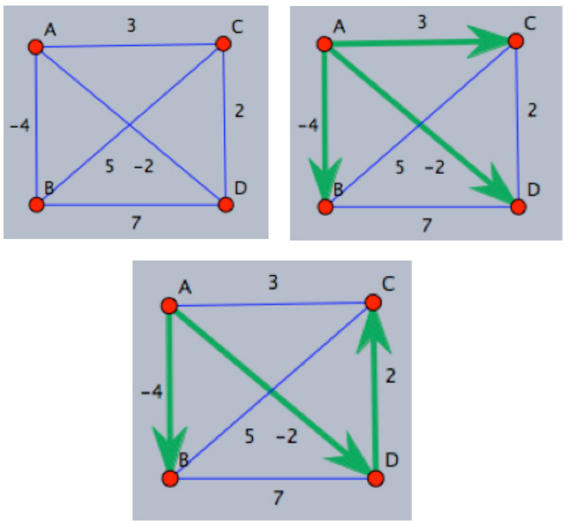
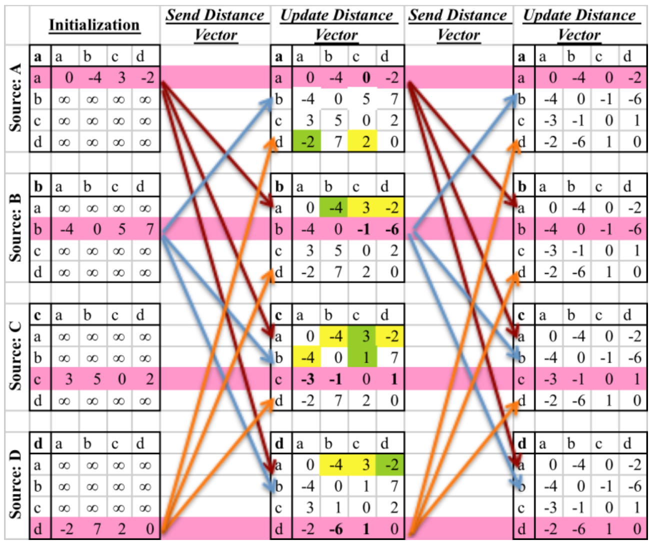


Figure 1. Bellman-Ford algorithm applied to the graph on nodes A, B, C, D with A as the source. The green arrows represent the shortest path from A to all other nodes. The change in shortest path reflects the iterative updates in distance vectors



	Initialization					Send Distance Vector	Update Distance Vector					Send Distance Vector	Update Distance Vector				
Source: A	**a**	a	b	c	d		**a**	a	b	c	d		**a**	a	b	c	d
	a	0	-4	3	-2		a	0	-4	**0**	-2		a	0	-4	0	-2
	b	∞	∞	∞	∞		b	-4	0	5	7		b	-4	0	-1	-6
	c	∞	∞	∞	∞		c	3	5	0	2		c	-3	-1	0	1
	d	∞	∞	∞	∞		d	-2	7	2	0		d	-2	-6	1	0
Source: B	**b**	a	b	c	d		**b**	a	b	c	d		**b**	a	b	c	d
	a	∞	∞	∞	∞		a	0	-4	3	-2		a	0	-4	0	-2
	b	-4	0	5	7		b	-4	0	**-1**	**-6**		b	-4	0	-1	-6
	c	∞	∞	∞	∞		c	3	5	0	2		c	-3	-1	0	1
	d	∞	∞	∞	∞		d	-2	7	2	0		d	-2	-6	1	0
Source: C	**c**	a	b	c	d		**c**	a	b	c	d		**c**	a	b	c	d
	a	∞	∞	∞	∞		a	0	-4	3	-2		a	0	-4	0	-2
	b	∞	∞	∞	∞		b	-4	0	1	7		b	-4	0	-1	-6
	c	3	5	0	2		c	**-3**	**-1**	0	**1**		c	-3	-1	0	1
	d	∞	∞	∞	∞		d	-2	7	2	0		d	-2	-6	1	0
Source: D	**d**	a	b	c	d		**d**	a	b	c	d		**d**	a	b	c	d
	a	∞	∞	∞	∞		a	0	-4	3	-2		a	0	-4	0	-2
	b	∞	∞	∞	∞		b	-4	0	1	7		b	-4	0	-1	-6
	c	∞	∞	∞	∞		c	3	1	0	2		c	-3	-1	0	1
	d	-2	7	2	0		d	-2	**-6**	**1**	0		d	-2	-6	1	0

Figure 2. Routing tables associated with the graph in Figure 1. Arrows indicate the forwarding of distance vectors to adjacent vertices. Highlighted cells indicate the detection of new shortest paths derived from recently updated distance vectors

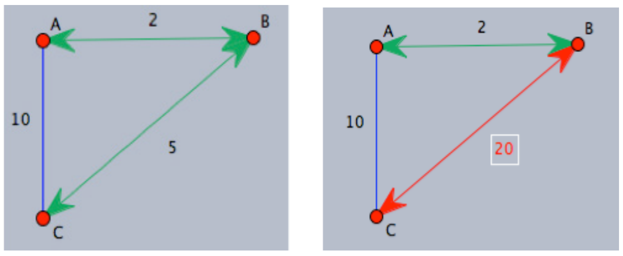


Figure 3. Example of how a link-cost change can result in a routing loop

In the first picture of Figure 3, the shortest path has already been determined between nodes, A, B and C. The problem arises when the link cost between nodes B and C changes from 5 to 20. At this moment, B detects the link-cost change and updates its distance vector accordingly to route through A to get to C. But A is not aware of the link-cost change and will continue to route through B to get to C which results in a routing loop between A and B. However, through continually updating their distance vectors, A will eventually route to C directly, terminating the routing loop. This phenomenon is known as the count-to-infinity problem[2]. Adding poisoned reverse is a technique designed to counteract this issue and can be applied to the example in Figure 3 in the following way: since the link-cost change results in B attempting to route through A to get to C, B will broadcast its cost of routing to C as $D(C) = \infty$. When A receives the updated distance vector from B, it will now believe that a direct path from B to C does not exist and will therefore route to C directly (as it would have all along if global state information was available).

While localized information certainly decreases the overhead costs of routing protocols, it enables errors to propagate from node to node. Despite the existence of preventative measures such as adding poisoned reverse, Dijkstra's algorithm and the Bellman-Ford algorithm are neither robust nor computationally efficient enough to handle the dynamic nature of wireless ad hoc networks. In order to formulate routing protocols that are appropriate for these purposes, desirable structures will be analysed next.

3. Structures of Wireless Ad Hoc Networks

The remainder of the algorithms to be presented are known to work in planar, connected, unit distance wireless graphs. First, a justification for modeling wireless ad-hoc networks in unit distance graphs will be outlined, followed by localized methods to extract planar sub graphs.

3.1. Unit Distance Wireless Graphs

Let P_n be a set of n points in the Euclidean plane. The unit distance wireless graph of P_n ($UDWG(P_n)$) contains all n points, and edges $e(u,v)$ such that the distance between u and v is within 1 unit (i.e., $d(u,v) \leq 1$)[4]. An example of a $UDWG$ on a set of 6 points is illustrated in Figure 4.

The reasoning behind using $UDWG$ to model wireless networks stems from the concept of broadcast ranges. That is, wireless devices are limited in their ability to communicate with other devices in accordance to their broadcast range. In the $UDWG$, all broadcast ranges are assumed to be uniform, hence u can send a packet directly to v if and only if v is contained within the circle of radius 1 centered at u.

Finally, it is entirely reasonable to assume that a wireless device has knowledge of its co-ordinates due to the wide use of Global Positioning Systems.

3.2. Gabriel Graphs

Consider a set of n points in the Euclidean plane $\mathrm{P_n}$. The Gabriel circle $\mathrm{C(p,q)}$ of two points $\mathrm{p,q}$ is defined as the circle that passes through both points and has the segment pq as a diameter, with the condition that no other point r from $\mathrm{P_n}$ lies within $\mathrm{C(p,q)}$.

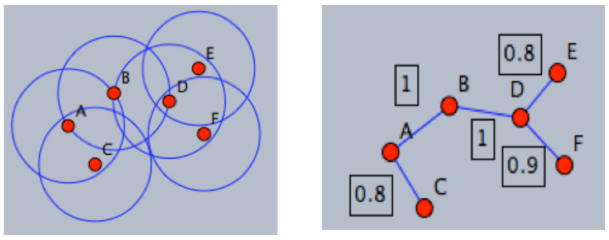


Figure 4. $UDWG$ of a set of 6 points: $\{A,B,C,D,E,F\}$. Each circle has a radius of 1 unit

The Gabriel graph of P_n, $GG(P_n)$, consists of all n points and edges $e(p,q)$ such that $C(p,q)$ exists[5]. Figure 5 demonstrates the construction of the Gabriel graph on 3 points.

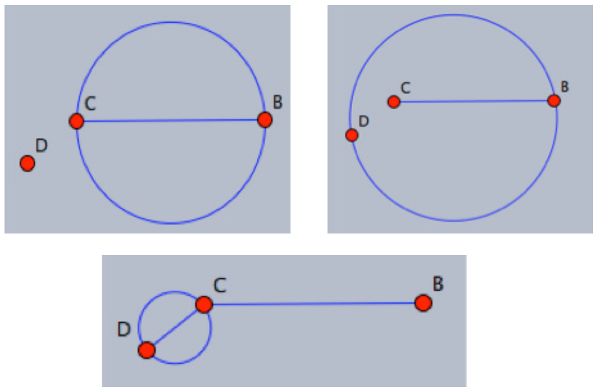


Figure 5. Gabriel Graph of a set of 3 points: $\{B,C,D\}$

To show that the Gabriel graph is planar we will first assume the opposite and then arrive at a contradiction.

Suppose that there exists a set of points $\mathcal{P}$ such that the Gabriel graph $GG(\mathcal{P})$ is non-planar. Then there exists at least one edge intersection between two edges $e(p,q)$ and $e(r,s)$. Since $e(p,q)$ is a Gabriel edge it follows that neither r nor s lies inside $C(p,q)$. Then regardless of the position of r or s outside of $C(p,q)$, the circle $C(r,s)$ will contain at least one of the points p or q. But then $e(r,s)$ cannot be a Gabriel edge, which contradicts the supposition. Therefore, the Gabriel graph on a set of points

$\mathcal{P}$ is always planar. Figure 6 illustrates the proof of the Gabriel graph's planarity.

Finally, given a connected $UDWG$ on a set of points P_n, the intersection of $UDWG(P_n)$ and $GG(P_n)$ results in a connected planar subgraph $UDWG'(P_n)$[5].

To prove the connectedness of $UDWG'(P_n)$, suppose that $UDWG(P_n)$ is connected and $UDWG'(P_n)$ is disconnected.

Then there exists points $u, v \in P_n$ such that they are adjacent in $UDWG(P_n)$ but not adjacent in $UDWG'(P_n)$. This necessitates the existence of a third point $t \in P_n$ such that t lies within the Gabriel circle $C(u, v)$. But then $d(t, u) < d(u, v)$ and $d(t, v) < d(u, v)$. Hence there are edges $e(t, u)$ and $e(t, v)$ resulting in a path from u to v in $UDWG'(P_n)$ (which contradicts the above supposition). Therefore if $UDWG(P_n)$ is connected, then $UDWG'(P_n)$ is connected as well[5].

It follows that a connected planar subgraph can be extracted from any connected $UDWG$ by having each node in the graph check for the existence of Gabriel circles between itself and its neighbours. Thus, there exists a localized method to extract planar subgraphs which can be accomplished for each point $p \in P_n$ in $O(k \ln k)$ time, where k is the number of neighbouring nodes to p[5].

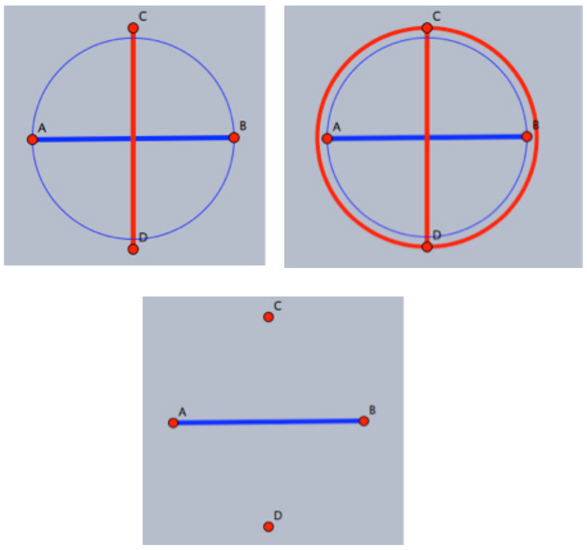


Figure 6. Planarity of the Gabriel graph on a set of 4 points: $\{A, B, C, D\}$

3.3. Delaunay Graphs

The Delaunay graph is an excellent structure to have within networks as it contains every Gabriel edge. Although the relative neighbourhood graph will not be discussed in this paper, it is worth mentioning that the Gabriel graph contains the relative neighbourhood graph as a subgraph, and the relative neighbourhood graph contains the minimum spanning tree as a subgraph. Hence the Delaunay graph contains many valuable structures as subgraphs. Finally the connectedness and planarity of the Delaunay graph and all its subgraphs can be derived from the idea of "forbidden zones" (e.g., the Gabriel circle between any two points is a forbidden zone for any other point) and using the relationships between the above-mentioned structures. However, this is beyond the scope of this paper and will be left for the interested reader as an exercise.

Before defining the Delaunay graph of a set of points P_n, we must define the convex hull. The convex hull of a set of points P_n is the smallest enclosing polygon $\mathcal{P}$ comprising of all points $p, q \in P_n$ with the condition that every line $l(p, q)$ is completely contained within $\mathcal{P}$[6].

The Delaunay graph $D(P_n)$ partitions the convex hull into disjoint triangles with the condition that the circumcircle of each triangle does not contain any other point in P_n.

$D(P_n)$ is uniquely defined if no 4 points are co-circular[5].

$D(P_n)$ is dual to the Voronoi diagram which divides the plane into n disjoint regions. Each region is defined by a point $p \in P_n$ such that anything within the region is closer to p than any other point $q \in P_n$. The Voronoi diagram can be computed in $O(n \log n)$ time; therefore, due to its duality, $D(P_n)$ can be calculated in the same time[6].

It has been suggested in[5] to force a wireless network to contain the Delaunay graph by either increasing the transmission rates of the wireless devices or by deploying more radio stations.

4. Routing in Wireless Ad-Hoc Networks

There are many approaches to routing in wireless ad-hoc networks. An absolute necessity in employable routing protocols is the guaranteed delivery of packets to their destination. Compass Routing and Face Routing can guarantee delivery in certain geometric structures as outlined below.

4.1. Compass Routing

Given a planar $UDWG$ on a set of points P_n in the Euclidean plane, suppose that a point u intends to send a packet to a destination v (Note if $UDWG(P_n)$ is non-planar, then compute $UDWG'(P_n)$). The forwarding packet must maintain the location of its: current position u, destination v, and all nodes n adjacent to u.

```
Algorithm COMPASS $(u, v, n)$
Input: the location of a packet's: current position $u$, destination $v$, and adjacent nodes $n$.
Output: arrival of packet at its destination
1. Forward packet to neighbour $w \in n$ such that the slope of $l(u, w)$ is closest to $l(u, v)$ amongst all $l(u, n)$
2. if $w = v$
3. then report arrival of packet at destination, break
4. else COMPASS $(w, v, n)$
```

Figure 7 illustrates how this recursive algorithm would proceed on a graph of 8 nodes. Nevertheless, Compass routing is known to work on Delaunay graphs that are uniquely defined, but is still prone to routing loops in graphs with low convexity, and can even fail in convex graphs[5].

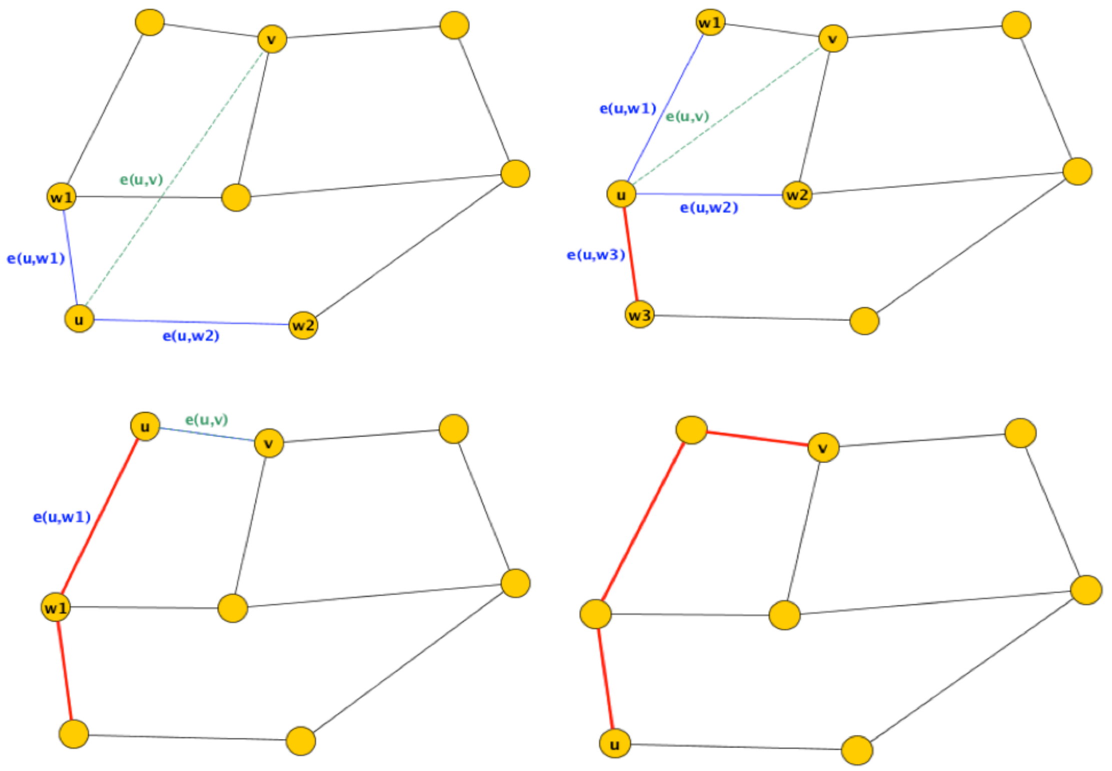


Figure 7. Forwarding of a packet from u to its destination v using Compass routing in the given graph of 8 nodes

Randomized compass routing guarantees delivery in convex subdivisions by modifying the above algorithm to forward packets randomly to adjacent nodes $w \in \{w_{cw}, w_{ccw}\} \subseteq n$ where w_{cw} and w_{ccw} are the neighbours of w that minimize the clockwise and counter-clockwise angle between all $l(u,n)$ and $l(u,v)$. The packet could theoretically take a very long time to arrive at its destination but in practice, randomized compass routing performs well[5].

4.2. Face Routing

Just as the name suggests, face routing considers the disjoint regions induced on the plane by the edges of the $UDWG$ on a set of points P_n, wherein some edges are considered to appear twice (namely the edge shared between two traversed faces). In addition to a distance d (initially set to 0), a packet must maintain the locations of: its origin u, an intermediate point s, the destination v, and the previous two nodes visited u_{p1} and u_{p2}. This last requirement helps prevent the occurrence of routing loops[4].

```
Algorithm FACE $(d, u, s, u_{p1}, u_{p2}, v)$
Input: distance $d$ (initially set to 0), and the locations of: its origin $u$, a point $s$ (initially set to $u$), the destination $v$, and the previous two nodes visited $u_{p1}$ and $u_{p2}$ (initially set to null)
Output: arrival of packet to its destination $v$
1. determine the face $F$, incident to $s$ that is intersected by the line $l(u,v)$
2. begin the traversal of $F$, updating $u_{p1}$ and $u_{p2}$ after each edge traversal
3. if packet arrives at $v$
4. then report arrival of packet at destination
6. else upon intersecting the line $l(u,v)$, calculate the distance $d'$ from u to this intersection point $s'$
7. if $d' > d$
8. then $d' = d$
9. upon arriving back at $s$ traverse $F$ until $s'$ has been reached
10. FACE $(d, u, s', u_{p1}, u_{p2}, v)$
```

Refer to Figure 8 for a simulation of face routing on the given graph of 17 nodes.

Out of all the algorithms presented, face routing is by far the most robust as it is known to work on all planar networks. Due to the constant amount of memory of the packet, the algorithm handles changes in topology very well. For instance if one of the nodes along the path from u to v stops transmitting its signal, the packet and its forwarding method is unaffected unless the packet resides at the failed node in question. If this is the case then the packet of will not reach its destination since only one copy of the packet is forwarded from node to node. Aside from this scenario, a packet is also subject to routing loops if a node along the path from u to v continually fails and recovers during a face traversal; but this is a highly degenerate case. Finally, each of the edges of a given face is traversed at most twice[4].

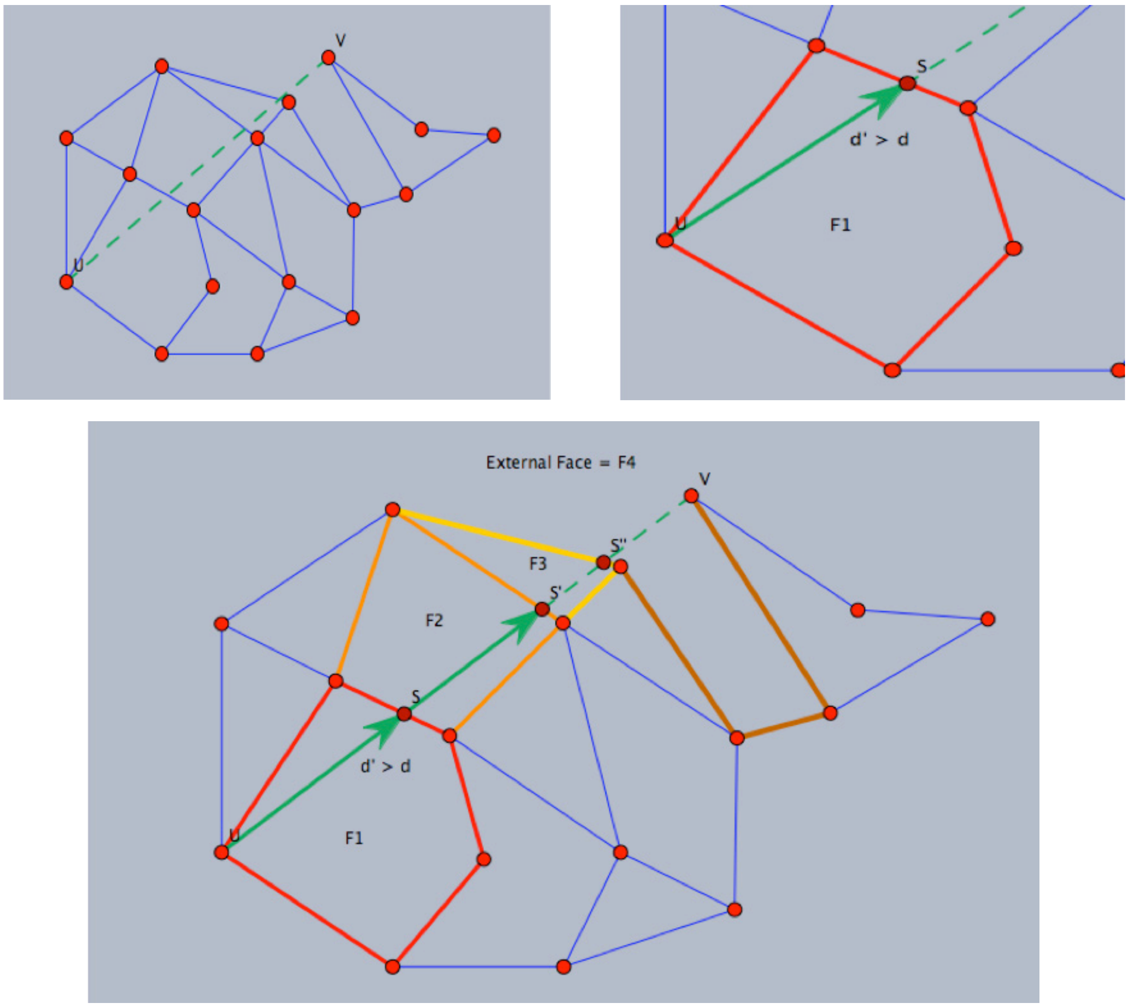


Figure 8. Face routing on a graph of 17 nodes. Non-blue edges indicate an edge traversal. In this scenario, 4 faces are traversed in total (including the external face) in order to deliver a packet from u to v

5. Conclusions

At first glance it would appear that the link-state and distance-vector routing protocols are ill suited for wireless ad-hoc networks, but the majority of heuristic protocols used are largely influenced by their methodologies[1]. What is highly undesirable of these methods is the overhead costs associated with maintaining routing table(s).

The unit distance wireless graph is a good representation of wireless ad hoc networks in which the wireless devices all have a uniform broadcast range. Of course, there are many cases in which broadcast ranges are not uniform, and there are often environmental factors that can inhibit a transmitter's signal strength. However this model provides the foundation to which compass routing and more importantly face routing can guarantee delivery.

It was suggested in[4] that since nodes within a unit distance wireless graph generally have relatively few adjacent nodes, the greedy quadratic time algorithms are better suited for determining incident edges, as their implementation is simpler. However, the robustness and scalability of face routing is undeniable and should at least be employed as an alternative routing protocol when routing loops persist within a system.

Many other on-demand routing protocols like face routing and compass routing require methods to extract planar subgraphs. This paper discussed only the use of the intersection of the Gabriel graph and the unit distance wireless graph ($UDWG$) as an alternative to the greedy approach. The relative neighbourhood graph (RNG) is another nice geometric structure that can be derived in the same amount of time and is equally useful in the formation of localized routing algorithms[2, 6].

ACKNOWLEDGEMENTS

Dr. Vassilev's research is supported by NSERC Discovery Grant.

REFERENCES

[1] Misra, P. (1999) Routing protocols for ad hoc mobile wireless networks.[Online]. Available at: http://www.cse.wustl.edu/~jain/cis788-99/ftp/ adhoc_routing/index.html

[2] Kurose, J., Ross, K. (2012). Computer networking: A top down approach. (6th ed.). Addison – Wesley, Boston, MA.

[3] Cormen, T., Leiserson, C., Rivest, R. (1999). Introduction to Algorithms. The MIT Press, Cambridge, MA.

[4] Urrutia, J. (2007). Local solutions for global problems in wireless networks. Journal of Discrete Algorithms, 5(3), 395-407.

[5] Urrutia, J. (2002). Routing with guaranteed delivery in geometric and wireless networks, in Handbook of Wireless Networks and Mobile Computing (ed I. Stojmenovic), John Wiley & Sons, Inc., New York, USA.

[6] de Berg, M., Cheong, O., van Kreveld, M., Overmars, M. (2008). Computational geometry: Algorithms and applications. (3rd ed.). Springer.