Face Routing in Wireless Ad-hoc Networks

by

Xiaoyang Guan

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Face Routing in Wireless Ad-hoc Networks

Xiaoyang Guan

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2009

Face routing is a simple method for routing in wireless ad-hoc networks. It only uses location information about nodes to do routing and it provably guarantees message delivery in static connected plane graphs. However, a static connected plane graph is often difficult to obtain in a real wireless network.

This thesis extends face routing to more realistic models of wireless ad-hoc networks. We present a new version of face routing that generalizes and simplifies previous face routing protocols and develop techniques to apply face routing directly on general, non-planar network graphs. We also develop techniques for face routing to deal with changes to the graph that occur during routing. Using these techniques, we create a collection of face routing protocols for a series of increasingly more general graph models and prove the correctness of these protocols.

Acknowledgements

I am deeply indebted to Faith Ellen and Peter Marbach, who I had the privilege to have as my supervisors throughout my graduate study. This thesis could not have been accomplished without their excellent guidance. They have given me invaluable advice on my study and on learning the skills to do good research. I also greatly appreciate their patience and support. Especially, Faith Ellen has put an incredible amount of time and effort helping me with the writing of this thesis and proofreading the algorithms and their proofs of correctness.

I would like to thank Eyal de Lara and Avner Magen for being members of my supervisory committee and their discussion and advice about my research.

I would also like to thank my fellow students, especially Stratis Ioannidis and Jingrui Zhang, for their friendship and encouragement.

Finally, I especially thank my wife. She has made many sacrifices to support my study and always been understanding and encouraging.

Contents

List of Algorithms and Figures

Glossary of Notation

UDG	Unit Disk Graph
QUDG	Quasi Unit Disk Graph
EDQUDG	Edge Dynamic Quasi Unit Disk Graph
MQUDG	Mobile Quasi Unit Disk Graph
$\angle xuy$	the clockwise angle around u from edge (u, x) to edge (u, y)
$\|uv\|$	the Euclidean distance between points u and v
u, v, w	real nodes
α, β, γ	virtual or real nodes
(α, β)	the directed edge from node α to node β
s	the source node
d	the destination node
pq	the straight line segment between points p and q
ε	distance within which nodes are neighbors in quasi unit disk graphs
$F(\pi)$	a function that maps a virtual path π to a sequence of real nodes

Chapter 1

Introduction

A wireless ad-hoc network consists of a collection of nodes that communicate with each other through wireless links without a pre-established networking infrastructure. It originated from battlefield communication applications, where infrastructured networks are often impossible. Due to its flexibility in deployment, there are many potential applications of a wireless ad-hoc network. For example, it may be used as a communication network for a rescue-team in an emergency caused by disasters, such as earthquakes or floods, where fixed infrastructures may have been damaged. It may also provide a communication system for pedestrians or vehicles in a city. Another example of a wireless ad-hoc network is a rooftop network [9, 12], which consists of a number of wireless nodes spread over an area to provide local networking service and access to wired networks, such as the Internet, for residents in the neighborhood. Another application of wireless ad-hoc networks is a sensor network, which consists of a large number of small computing devices deployed in a region that collect data and may send the information to a central server.

1.1 Routing in Wireless Ad-hoc Networks

In a data communication network, if two nodes are not connected directly by a communication link, their messages to each other need to be forwarded by intermediate nodes. Finding a path between two nodes on which to send messages in data communication networks is a fundamental problem, called *routing*. In a traditional computer network, there are nodes dedicated to the routing task, called *routers*. Applications on hosts communicate with servers and messages are forwarded by routers to their destinations. In contrast to traditional computer networks, wireless ad-hoc networks do not distinguish between hosts, servers, and routers. Wireless ad-hoc networks are also different from wireless networks with base stations, such as cellular phone systems, in which messages are relayed by the base stations. In wireless ad-hoc networks, nodes are not only application hosts, but also function as routers to forward messages for other nodes that are not within direct wireless transmission range of each other. The participating nodes form a self-organized network without any centralized administration or support [25]. Therefore, wireless ad-hoc networks are purely distributed systems.

The characteristics of wireless ad-hoc networks that are different from traditional networks pose two specific challenges in routing. First, since there are no dedicated routers nor persistent routing databases, wireless ad-hoc networks require fully distributed routing protocols. Second, the topology of a wireless ad-hoc network can change frequently and unpredictably. A routing protocol for a wireless ad-hoc network must be well adapted to the constant changes of topology. These characteristics make routing in wireless ad-hoc networks an interesting and challenging problem.

A large variety of ad-hoc routing protocols have been proposed, ranging from modifications and optimizations of traditional routing approaches for static networks to innovative methods for ad-hoc networks that utilize geographic location information about nodes. One approach to ad-hoc routing is to modify traditional routing algorithms by maintaining up-to-date topology information [52, 10, 49]. This is done by periodically

broadcasting updates to routing information throughout the network. However, this involves large communication overhead. To avoid periodically exchanging routing information, another approach is to establish a route only when it is needed by flooding a route request throughout the network [29, 53, 51, 23, 20]. Both approaches are efficient only in small and moderate sized networks [55, 8, 28, 11, 50, 60, 48].

Location-based routing, also known as geometric routing or geographic routing, has been proposed to address the scalability issue. Instead of using topological information, location-based routing protocols use geographical location information about nodes to route packets. These protocols assume that nodes know their own geographic locations (for example, from a Global Positioning System [30]) and the source node knows the location of the destination node.

A naive way to use location information is to broadcast route query messages within a restricted area to construct a route to the destination [6, 35]. More efficient location-based routing protocols transmit data packets directly without explicitly constructing a route in advance [14, 47, 36, 7, 31, 37]. In these protocols, nodes do not keep routing information, except for the locations of their neighbors, and a packet is not duplicated during routing. When a node receives a packet, the simplest way to route the packet is to forward it to the neighbor that is closest to the destination [14, 47, 36]. This is called *greedy routing*. Compared to other protocols, greedy routing has extremely low routing overhead and scales well to large networks. However, greedy routing may fail to deliver a packet, because the packet may reach a node whose neighbors are all farther away from the destination [36, 31].

A technique called *face routing* provably guarantees packet delivery in static connected plane graphs [36, 7, 31]. Face routing is applied on a plane graph, and the packet is forwarded along the boundaries of the faces that are intersected by the line segment between the source node and the destination node. The face routing protocols in the literature [7, 31, 38, 37] have the following two constraints: they need a separately constructed

spanning plane subgraph of the network for routing, and they assume that the plane subgraph remains static during the routing process. Extracting a connected spanning plane subgraph in a distributed manner may be difficult in real networks. Experimental results [33, 57] indicate that most existing algorithms have problems in real wireless networks due to radio range irregularities and imprecise location information. Problems with the resulting routing graph can lead to failure of face routing protocols. Moreover, experiments show that links in wireless ad-hoc networks, such as rooftop networks, are often unstable even when nodes are stationary [9, 1]. As a result, these protocols may not be practical in real wireless ad-hoc networks.

Other routing protocols that guarantee message delivery in static networks have been obtained by combining face routing with greedy routing [7, 31, 40, 37]. The protocols operate in the greedy routing mode until the packet reaches a node where greedy routing fails to proceed. Then, the protocols switch to the face routing mode as a recovery mechanism and switch back to greedy mode when possible.

This thesis extends face routing to more general and more realistic models of wireless ad-hoc networks with the goal of developing geometric routing protocols that guarantee message delivery in those models. We consider a series of graph models with increasing generality, which are defined in Section 1.2. We develop techniques that generalize and extend face routing, and design a collection of provably correct face routing protocols for these models.

1.2 Models

All graphs described in this thesis are considered to be drawings in the plane: vertices are represented by distinct points in the plane, and an edge is represented by the straight line segment between its endpoints. If no edges intersect except at their common endpoints, we say it is a *plane graph*. The edges of a plane graph partition the plane into disjoint

regions called *faces* [62]. A graph $H = (V', E')$ is a subgraph of graph $G = (V, E)$ if $V' \subseteq V$, $E' \subseteq E$, and each vertex in V' is placed at the same point in the plane as in G. If $V' = V$, then H is a spanning subgraph of G.

We assume that all nodes in a wireless ad-hoc network are located in a plane. For networks with stationary nodes, we assume that the nodes are in general position, i.e., no two nodes lie on the same point in the plane, and no three nodes lie on the same straight line. A network graph that represents a wireless ad-hoc network is drawn in the natural way: a vertex representing a network node is drawn at the point that represents the location of the node in the plane. For convenience, when we refer to a node or vertex, we do not distinguish between the network node, the vertex in the graph that represents the network node, and the point in the plane that represents the vertex. Similarly, when we refer to a link or edge, we do not distinguish between the wireless link, the edge in the graph that represents the link, and the line segment in the plane that represents the edge.

A simple graph model for wireless ad-hoc networks is the *unit disk graph* (UDG) model, in which each pair of vertices are connected directly by an edge if and only if they are at most distance 1 apart. A unit disk graph models a wireless ad-hoc network in which nodes have the same circular transmission range. Nodes are considered to be stationary during the routing process, so a unit disk graph is a static graph representing a snapshot of the network at a point in time. A connected unit disk graph contains a connected spanning plane subgraph, which can be used as the routing graph for face routing. Although the unit disk graph model is widely employed because of its simplicity, it is unrealistic since nodes in a wireless ad-hoc network often do not have circular transmission ranges due to obstacles.

The first extension we consider is the *quasi unit disk graph* (QUDG) model [5, 39], which allows the transmission region of nodes to be non-circular. In a quasi unit disk

graph, there is a constant ε, where $0 \leq \varepsilon \leq 1$, such that nodes at most distance ε apart are connected by an edge, nodes more than distance 1 apart are not connected by an edge, and nodes with distance in between may or may not be connected by an edge. It is a static graph model where no change of the network graph is allowed during the routing process. If $\varepsilon < 1$, a connected quasi unit disk graph may not have a connected spanning plane subgraph. For example, see Figure 1.1. Note that any graph can be viewed as a quasi unit disk graph with $\varepsilon = 0$ by taking the maximum length of any edge to be 1.

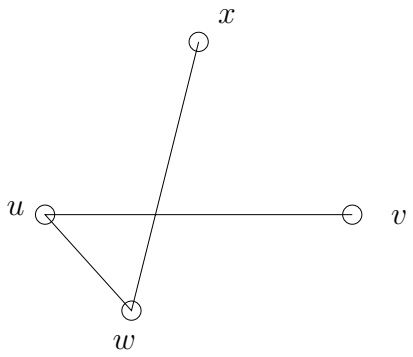


Figure 1.1: A quasi unit disk graph that does not have a connected spanning plane subgraph

Next, we extend our study to non-static graph models. An *edge dynamic graph* is a graph whose nodes remain stationary, but whose edges may change over time. An *edge dynamic quasi unit disk graph* (EDQUDG) is a quasi unit disk graph at all points in time: nodes that are at most distance ε apart are always connected by an edge, nodes that are more than distance 1 apart are never connected by an edge. The difference between an EDQUDG and a QUDG is that, in an EDQUDG, an edge whose length is between ε and 1 may repeatedly change between being active and inactive at arbitrary times. Edge dynamic quasi unit disk graphs represent networks in which the wireless connections between the nodes that are more than distance ε apart are unstable, or there exist moving obstacles that can interfere with connections between nodes during the routing process.

Finally, we extend the network graph model to graphs consisting of mobile nodes. A *mobile graph* is a graph each of whose vertices may move along a continuous trajectory, representing the movement of a mobile node in the network. A *mobile quasi unit disk graph* (MQUDG) is a mobile graph that is a quasi unit disk graph at all times. When two nodes are within distance ε of one another, they are connected by an edge. When their distance from one another is greater than 1, they are disconnected. When their distance from one another is greater than ε, but less than or equal to 1, they may or may not be connected.

The transmission time of a packet from a node to a neighbor is assumed to be bounded above by a constant. A transmission is successful as long as the connection exists for this length of time from the beginning of the transmission. Thus, in a static graph, no transmission failures occur. In edge dynamic and mobile graphs, it is possible that an edge becomes inactive when a packet is being transmitted along the link. We assume that such a transmission failure can be detected by the sender and, if this happens, it re-routes the packet.

1.3 Outline of Our Research and Contributions

The main contributions of our research are a better understanding of the capability and limitations of face routing and a collection of geometric routing protocols that guarantee message delivery in more realistic models of wireless ad-hoc networks. In the following, we give an outline of our research and contributions.

First, we present a new version of face routing that uses a more general rule to decide the face to be traversed. In static graphs, this version of face routing is similar to some protocols that combine greedy routing with face routing. Our new protocol is conceptually simpler. Moreover, in non-static graphs, our new protocol can be used to extend previous face routing protocols to obtain more general conditions under which

message delivery is guaranteed.

Then we study techniques to directly apply face routing on general non-planar network graphs, without extracting a plane subgraph. A (virtual) plane graph can be obtained from any graph by replacing each edge crossing with a virtual node. We show how to simulate face routing on the resulting plane graph without requiring real nodes to maintain extra information about virtual nodes. This extends face routing to more general graphs and simplifies geometric routing protocols that use face routing.

We have developed a collection of provably correct protocols that simulate face routing on unit disk graphs and quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$ using different assumptions about how much information is available at each node. The protocols that are based on more knowledge of the local neighborhood are straightforward, but those that require less information at each node are more complicated.

Next, we extend face routing to edge dynamic quasi unit disk graphs. In our earlier work, we presented a protocol for edge dynamic graphs, where we assumed that the graph is always a plane graph [21, 22]. However, edge dynamic quasi unit disk graphs are not necessarily plane graphs. Therefore, that protocol may not work for general edge dynamic graphs. We devise another protocol that combines our techniques for edge dynamic plane graphs with our techniques for applying face routing on quasi unit disk graphs. We prove that, under general conditions, this protocol works correctly in edge dynamic quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$.

Finally, we study face routing in mobile quasi unit disk graphs. It is challenging to do face routing in such graphs, because complicated changes to the network graph may happen during routing. We consider a restricted family of mobile quasi unit disk graphs in which each node may move only within a small circular region. We show that our protocol for edge dynamic quasi unit disk graphs can be applied to do routing in such graphs. We also consider a family of mobile quasi unit disk graphs in which the speed of the movement of nodes is limited. We present a variant of our protocol for routing to a

stationary destination node and give some evidence for its correctness.

1.4 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we give a survey of location-based routing protocols in the literature, in particular, the original face routing protocols and a variety of protocols that use face routing. In Chapter 3, we describe our new version of face routing that uses a more general face switching rule. In Chapter 4, we describe how to apply face routing on general non-planar network graphs directly. Then, in Chapters 5 to 7, we discuss the challenges of doing face routing in unit disk graphs, quasi unit disk graphs, and edge dynamic quasi unit disk graphs, and give routing protocols for each. Chapter 8 describes our results on some restricted versions of mobile quasi unit disk graphs. Finally, we highlight the main results in this thesis and discuss directions for future work in Chapter 9.

Chapter 2

Related Work

Location-based routing is done using physical location information about nodes. This is a very different approach than traditional routing, in which nodes need to maintain routing information [11, 42, 60]. Instead, location-based routing uses information about the geographic location of the neighbors of the current node to direct a packet to its destination.

In location-based routing protocols, it is assumed that nodes know their own locations and the source node knows the location of the destination node. A node equipped with a Global Positioning System (GPS) receiver can obtain its own geographic coordinates [30]. When GPS support is unavailable, there are other localization techniques that mobile nodes can use [24, 54, 56, 58]. In order to obtain the location of the destination node, a location service is needed from which a node can query about the locations of other nodes. Most location-based routing protocols assume that a location service exists as an external resource. Indeed, in the literature, location service has often been considered as an independent research topic. For research in this area, we refer the reader to [6, 19, 59, 32, 42].

According to how the location information is used, location-based routing protocols can be divided into three categories, *restricted directional flooding*, *greedy routing*, and

face routing. In the next two sections, we briefly describe protocols in the first two categories. Then, we describe face routing in more detail and discuss the variants of it that appear in the literature.

2.1 Restricted Directional Flooding

Restricted directional flooding protocols, including Location-Aided Routing (LAR) [35] and the Distance Routing Effect Algorithm for Mobility (DREAM) [6], are designed for mobile ad-hoc networks where the up-to-date location of the destination node is unknown. They use flooding to search for a path to the destination, but utilize location information so that packet broadcasts are confined to a subset of nodes instead of the entire network.

LAR assumes that the source has some knowledge about the movement of the destination node, for instance, its average or maximum speed. When the source node needs to send a message, it calculates a region, called the *expected zone*, in which it expects the destination node to be currently located. The size and shape of the expected zone depends on what information the source node knows about the destination. For example, if the source node knows the location of the destination node at some time in the past and its average speed, the expected zone is the circular region centered at that location with radius the distance it may have moved. If the source node does not know a previous location of the destination node, the entire region that may potentially be occupied by the ad-hoc network is the expected zone. Then, a *request zone* is determined that includes both the source node and the expected zone. The request zone is used to restrict the region of flooding in route discovery: only the nodes within the request zone broadcast the route request packet to their neighbors. Therefore, choosing the size of the request zone involves a trade-off between the routing communication overhead and the possibility that a path is found in it.

The DREAM protocol is used for sending data packets directly without route discov-

ery. Each node records the locations of all other nodes in the network and disseminates its current location to all other nodes with a frequency determined by its distances to them and its mobility rate. More specifically, the farther apart two nodes are, the less frequently they update their locations to each other; the faster a node moves, the more frequently it broadcasts its location. This approach reduces location update overhead without compromising the routing accuracy. When a node wants to send a message to another node, it obtains the direction of that node from its location information, and sends the message to all its neighbors in that direction. Each neighbor relays the message in the same way until the destination node is reached.

2.2 Greedy Routing

Greedy routing protocols use location information in a similar way, but they apply a greedy heuristic in path selection so that packet delivery is through point-to-point communication rather than through broadcast. A node forwards the packet to the neighbor that is closest to the destination in terms of distance [47] or direction [36], which is defined as the angle between the line segment from the node to the destination and the edge to a neighbor. Greedy routing is very simple and efficient since nodes do not need to maintain routing information, and packets are forwarded immediately without being duplicated. Compared to other protocols, greedy routing has extremely low routing overhead and it scales well to large wireless ad-hoc networks. However, greedy routing does not guarantee that a packet reaches its destination. If the distance to the destination is used to choose the next edge to send a packet, a local minimum may be reached. Even if the direction is used to choose the next edge, a loop may be formed [36].

2.3 Face Routing

Face routing, proposed in [36], was the first geometric routing algorithm that guaranteed message delivery without flooding. Several variants of face routing protocols [7, 31, 38, 40, 37, 41] were subsequently proposed. Face routing is applied on a plane subgraph of the network graph. A plane graph divides the plane into faces. The line segment between the source node and the destination node intersects some faces. In face routing, the packet is forwarded along the boundaries of these faces. A specific face routing protocol provides a set of rules for each node to decide where to send a packet using only the local information about its neighbors and the information in the packet header.

A typical face routing protocol works as follows [7]. When face routing starts, the packet is forwarded along the boundary of the first face intersected by the line segment from the starting point to the destination. The first edge of the traversal of a face is the first edge in clockwise order around the starting point from the line segment to the destination. After the traversal of an edge (u, v), the next edge of the face traversal is the first edge after (v, u) in clockwise order around v. In this way, the packet traverses the edges on the boundary of the face in the counterclockwise direction. The traversal in this way is called using the *right-hand rule*. When the traversal reaches an edge that intersects the line segment from the starting point to the destination at a point closer to the destination than the starting point is, that point becomes the new starting point and the traversal switches to the next face. This procedure repeats until the destination is reached.

Figure 2.1 shows an example of the route computed by such a face routing protocol. In this example, a packet is sent from node s to node d. It first travels along the boundary of face F_1, until it reaches p_1, the next intersection point of the boundary of F_1 and the line segment sd, where the traversal is switched to the next face, F_3. Subsequently, the packet travels along the boundaries of faces F_3 and F_5.

Although the basic idea of face routing is simple, some variants of face routing can fail

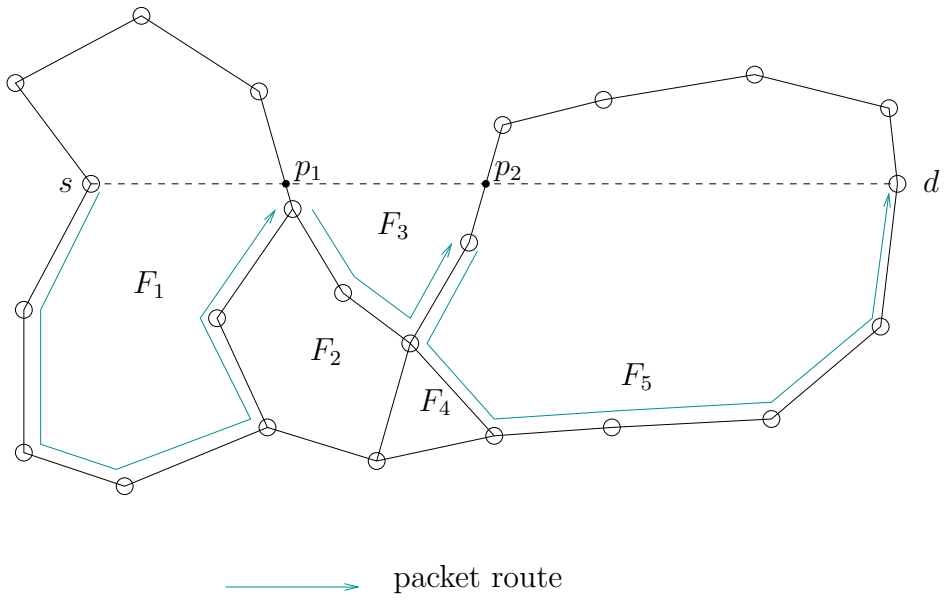


Figure 2.1: An example of the path followed by a packet using a face routing protocol

to deliver a packet in certain plane graphs [16]. For example, if a face traversal algorithm only uses nodes closer to the destination than the starting node as new starting points, it may not find a new starting point on some face, such as face F_1 in Figure 2.2 (a). In this example, node s is the source node, and node d is the destination node. The other two nodes v_1 and v_2 on the boundary of F_1 are farther away from node d than s. Another case when face routing can fail is if, when an edge intersecting the line segment between the starting point and the destination is found, a face traversal algorithm always uses the current node, instead of the intersection point, as the new starting point. In this case, a packet may loop along a sequence of faces without ever reaching the destination. This is illustrated in the example in Figure 2.2 (b), where this algorithm will forward the packet along (s_1, s_2), (s_2, s_3), and (s_3, s_1).

The Gabriel Graph [17], the Relative Neighborhood Graph [61], and several special subgraphs of the Delaunay triangulation [18, 43, 44, 45] are known to be plane graphs of a set of points in the plane. The intersection of any of them with a connected unit disk graph is a connected plane subgraph and can be extracted in a distributed manner,

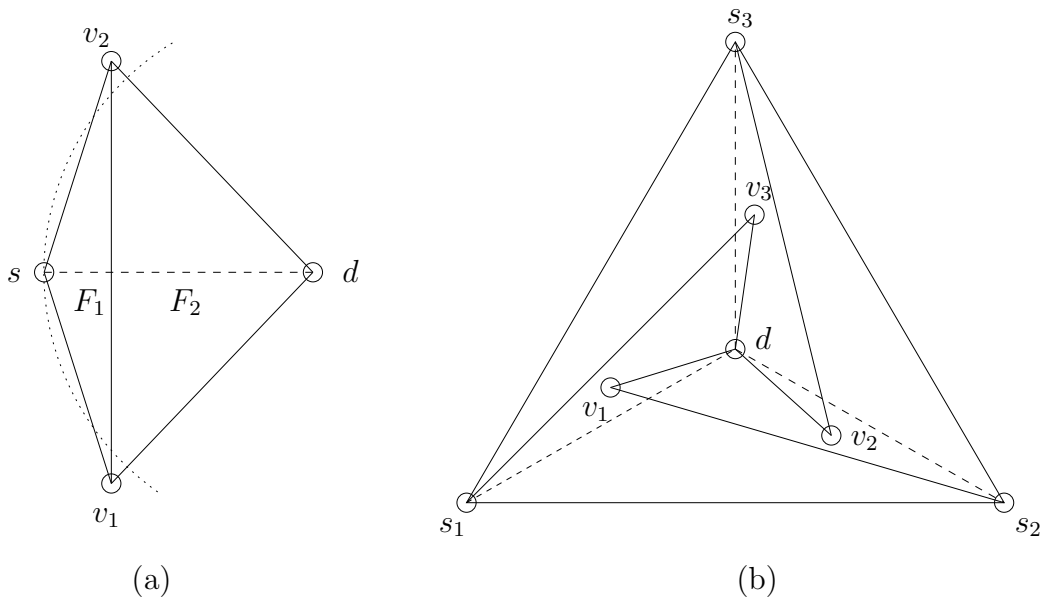


Figure 2.2: Examples of routing failure if a face routing algorithm only uses nodes as new starting points

i.e., using only local information about neighbors. These plane subgraphs can be used as routing graphs for face routing. For example, it is easy for a node in a unit disk graph to check whether each incident edge is in the Gabriel Graph of the same set of nodes, using only location information about its neighbors. If the unit disk graph is connected, the subgraph of the unit disk graph consisting of those edges is a connected plane subgraph.

It is known that face routing guarantees the delivery of a packet in static connected plane graphs [7, 21]. It has been combined with a greedy routing approach to make the overall routing protocol more efficient. One such protocol is Greedy Perimeter Stateless Routing (GPSR) [31]. The main routing strategy in GPSR is greedy routing. A node sends a packet to one of its neighbors in the network that is closest to the destination, provided that neighbor is closer to the destination than it is. However, when a node is closer to the destination than any of its neighbors, face routing is used as a recovery mechanism.

Like the GPSR protocol, the GOAFR+ protocol [37] is also a combination of greedy

routing and face routing. GOAFR+ uses much more complicated techniques to combine face routing with greedy routing so that the length of the route found by this protocol is always within a constant factor of the square of the length of the shortest path. Specifically, it defines an area that may be adaptively resized during the routing process. When the protocol is operating in face routing mode, the traversal is confined to this area. If the traversal along the boundary of the face in the counterclockwise direction will go out of this area, it turns around and traverses the face in the opposite direction. If traversing in both directions reaches the boundary of the confined area, this area is expanded. It also uses an involved set of rules to determine when the protocol should switch back to greedy mode from face traversal mode. It is shown in [38] that, for any geometric routing protocol, the length of the route found by the protocol is, in the worst case, at least a constant times the square of the length of the shortest path between the source and the destination. Thus, GOAFR+ is asymptotically optimal with respect to the length of the route. Their simulation results show that GOAFR+ is also efficient on random graphs.

There have been a few papers on geometric routing in quasi unit disk graphs. For connected quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$, Barrière et al. [5] propose a technique to construct a super-graph by adding extra edges to the network graph, which correspond to paths in the network graph. Their technique involves nodes exchanging information with neighbors so that they set up extra edges when needed. The extra edges guarantee that a connected spanning plane subgraph of the super-graph exists. Although the resulting super-graph is not a unit disk graph, the algorithm for extracting the intersection of the Gabriel Graph and a unit disk graph can be applied to extract a plane subgraph of the super-graph. They prove that if the network graph is connected, the resulting subgraph of the super-graph is a connected spanning plane graph of the network nodes.

Face routing can be performed on the resulting subgraph. Each node maintains a routing table for its incident extra edges, so that a message that is supposed to be routed along an extra edge is forwarded along the corresponding path in the network to the other

endpoint of the extra edge. For static quasi unit disk graphs, this approach may result in long routes. It is unclear whether this approach can be extended to edge dynamic quasi unit disk graphs. When there is a change in the network, the nodes whose neighborhood is changed need to re-compute the super-graph and the routing subgraph. This may have a ripple effect through the network: the changes at these nodes may cause changes to the extra edges at their neighbors, which in turn can affect more nodes. If there are frequent changes in the network during the routing process, their routing protocol may fail to deliver the packet.

For quasi unit disk graphs with $0 \leq \varepsilon \leq 1$, Kuhn et al. [39] give a distributed algorithm to construct a sparse spanner of the network graph. Then they apply Barrière et al.'s technique to obtain a sparse spanning plane graph for routing in quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$.

Lillis et al. [46] consider quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$. They use the idea of adding virtual nodes (where edges cross) on a sparse spanning subgraph of the network graph to obtain a plane graph for routing. This idea is also briefly mentioned by Kuhn et al. [39]. One of the endpoints of the edges crossing at each virtual node serves as a proxy for that virtual node, sending and receiving messages on its behalf, and maintaining its routing table. However, they do not explain how proxies are chosen, nor do they describe the detailed behavior of the proxies. Their approach requires extra storage space at nodes for keeping routing tables. To keep this storage expense small, the sparse subgraph is extracted from the network graph to limit the number of virtual nodes managed by each real node. In addition, they still assume the network graph to be static for routing.

Barrière et al. [5] showed that, for any constant k, there exists a quasi unit disk graph with $\varepsilon < \frac{1}{\sqrt{2}}$ that contains two crossing edges e and e', and any path connecting one endpoint of e and one endpoint of e' has length at least k hops. This means that the edge crossing cannot be detected at those endpoints with $(k-1)$-hop neighborhood information. Thus, a local routing algorithm based on face routing (without flooding)

may miss a crossing edge and fail to deliver a packet in a quasi unit disk graph with $\varepsilon < \frac{1}{\sqrt{2}}$.

The face routing protocols discussed above assume that the routing process is done instantaneously so that the protocols are executed on a static graph. This assumption is relaxed in my M.Sc. thesis [21] and my paper [22], where two new face routing protocols are proposed and proved to be correct in edge dynamic graphs that are always plane graphs. The Timestamp-Traversal protocol stores a time stamp in the packet header to record the departure time of the packet from its starting point. Links that become available after this time are ignored. The Tethered-Traversal protocol fully exploits all available edges, and takes advantage of new edges. To achieve this, it uses information about the path followed by a packet, carried in the packet header, to determine whether the next edge along the boundary of the current face should be traversed or not. As in the original face routing protocols, nodes do not keep any history information about links or about packets that they have received. Both protocols guarantee message delivery if, during the traversal of each face, the graph contains a stable connected spanning subgraph. Since edge dynamic quasi unit disk graphs are not necessarily plane graphs, Timestamp-Traversal and Tethered-Traversal do not necessarily guarantee message delivery in edge dynamic quasi unit disk graphs.

For mobile wireless ad-hoc networks, Ioannidou [27] addressed the routing problem in a different framework. She presented a simple model that can be simulated in a mobile ad-hoc network, given explicit bounds on the speed of nodes. The connections in this simplified model remain relatively stable for sufficiently long so that face routing can be applied to route a packet along the boundary of a stable face in this model.

Chapter 3

A New Version of Face Routing

In this chapter, we present a new version of face routing for that guarantees message delivery while naturally incorporating a greedy approach. Although we devised it to apply face routing to mobile graphs, it generalizes and is better than the original version described in Chapter 2. The main difference between our new version of face routing and the classical face routing protocols is the rule to choose the new starting point [7, 31, 37, 16]. Our rule for selecting a new starting point is more general. In our protocol, the new starting point can be, but is not necessarily, an intersection point on the line from the source to the destination.

Specifically, in our new version, starting from the starting point of a face, the packet is forwarded along the boundary of the face intersected by the line segment from the starting point to the destination, until it finds a point on the face that is closer to the destination than the starting point is. Then this point becomes the new starting point and the packet starts to traverse the face intersected by the line segment from the new starting point to the destination. This procedure repeats until the packet reaches the destination.

Figure 3.1 shows an example of the path followed by a packet using our new protocol. It is the same graph as in Figure 2.1 on page 14 where we explained the original version

of face routing. In this example, from node s to node d, the packet traverses faces F_1, F_2, F_4, and F_5 successively. Our traversal does not necessarily switch to another face when a new starting point is found: In our example, each node visited on face F_5 is closer to the destination than its predecessor. For comparison, notice that previous face routing protocols would switch from face F_1 to F_3 at node x, when it sees edge (x, y), which intersects sd, but our new protocol switches from F_1 to F_2 at node u.

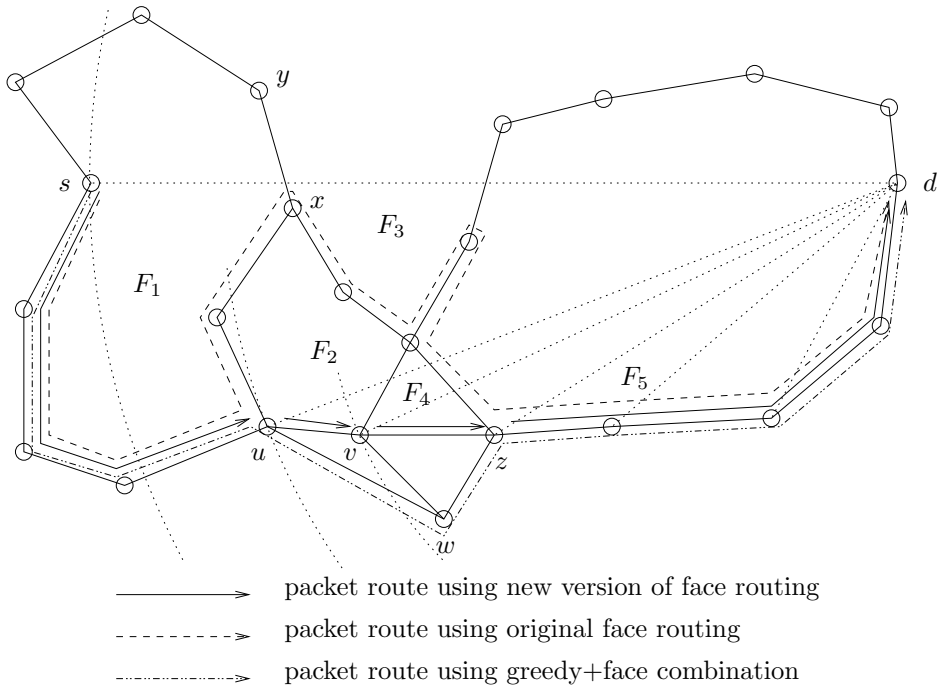


Figure 3.1: Different paths followed by a packet travelling from node s to node d

A new starting point can be any point on the boundary of the current face that is closer to the destination than the current starting point is. In our version of face routing, once we see an edge that contains such a point, we choose the point on that edge closest to the destination as the new starting point. In static graphs, such a point can always be found, as long as the source and the destination are in the same connected component of the graph. Note that, as in the original face routing protocols, a new starting point is not necessarily at a node.

In static graphs, the behavior of our new protocol is similar to some protocols that

combine greedy routing with face routing, such as GPSR [31]. During the traversal of a face, if the packet reaches a node, say, node u, that is closer to the destination than the starting point is, GPSR switches to greedy mode, whereas our protocol switches to the next face. Now, if node u does not have a neighbor closer to the destination than itself, GPSR will switch back to face routing and start to traverse the same face as our protocol. Otherwise, GPSR forwards the packet to the neighbor of u that is closest to the destination. This node may or may not be the same node to which our protocol forwards the packet. In the example in Figure 3.1, GPSR forwards the packet from node u to node w, but our protocol forwards the packet to node v. Both protocols subsequently forward the packet to node z, and the packet follows the same path in both protocols thereafter. A benefit of our protocol is that it unifies a greedy heuristic into face routing so that it is not necessary to switch between modes.

In non-static graphs, there are situations when the original face routing protocols fail, but our new protocol succeeds. One example is shown in Figure 3.2. The graph is the

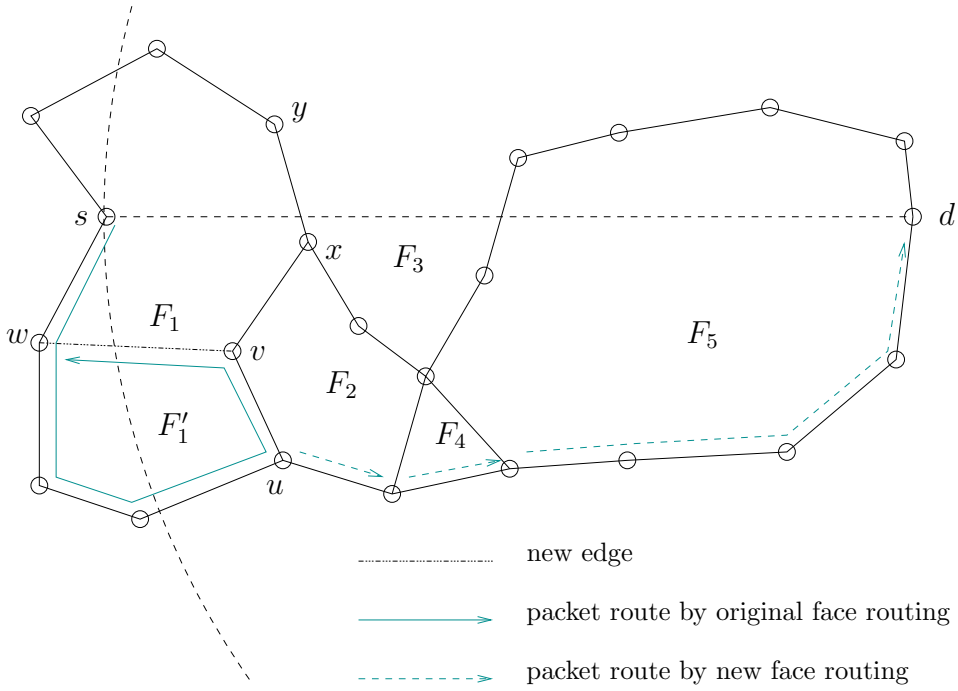


Figure 3.2: An example where the original face routing fails and the new version succeeds

same as in Figure 3.1, except that the edge between nodes v and w becomes available after the packet passes w, but before it reaches v. The original face routing protocols fail because the packet will be forwarded from v to w and will loop on the new face F_1'. Our new face routing is not affected by this change to the graph and routes the packet to the destination as before.

The Tethered-Traversal protocol in my M.Sc. thesis extends the original face routing protocols to deal with problems like the one illustrated in Figure 3.2. If the graph is an edge dynamic plane graph, Tethered-Traversal succeeds as long as there is a stable connected spanning subgraph during the traversal of each face, because a new intersection point will be found within the face of this stable subgraph that contains the face being traversed at the beginning of the traversal. However, in mobile graphs, Tethered-Traversal has some difficulties even when a stable connected spanning subgraph exists. One problem is when the face of the stable subgraph containing the current face moves so that it no longer intersects the line segment from the starting point to the destination. In this case, a new starting point cannot be found. In Section 8.2, we present a variant of Tethered-Traversal that uses this new version of face routing. Under a reasonable set of assumptions, we prove that the current face will always contain a point that is closer to the destination than the starting point was at the beginning of the traversal, and we conjecture that our new variant of Tethered-Traversal guarantees message delivery.

Chapter 4

Simulating Face Routing On Virtual Graphs

There are two problems with using face routing on a non-planar network graph: the network graph may not have a connected spanning plane subgraph, and, even if such a subgraph exists, extracting it may be difficult or complicated [33, 57, 5, 34]. In the following, we describe a general approach to apply face routing on non-planar graphs without extracting a plane subgraph.

A non-planar network graph can be viewed as a virtual plane graph. Conceptually, we add a virtual node at each point where two or more edges cross, and split the edges at these virtual nodes. Thus, we obtain a virtual plane graph that consists of the original network nodes and the virtual nodes. If the original graph is connected, so is the virtual plane graph, and if we apply face routing in this virtual plane graph, we would find a path to the destination. We call this path a *virtual path*, because it may contain virtual nodes. A virtual node cannot receive or send a packet. Kuhn et al. [39] and Lillis et al. [46] maintain routing tables at real nodes to enable messages to be sent to and from virtual nodes. In our approach, additional routing information does not need to be stored at real nodes. We simply compute a real path in the network graph that follows the virtual

path. We say that such a protocol *simulates* face routing in the virtual plane graph.

To formally define what it means for a real path to follow a virtual path, we first introduce some notation and definitions. We use Greek letters α, β, etc., to denote the nodes in a virtual path, which may be virtual or real, and use lowercase letters to denote real nodes. Each edge in the virtual graph is either an entire edge in the network graph or a part of it. We call an edge in the virtual graph a *virtual edge*. We use the *beginning node* or *point* and *ending node* or *point* to refer to the two endpoints of a real or virtual edge.

Given a virtual path whose last node is a real node, we define a function that maps this virtual path to a sequence of real nodes as follows.

Definition 4.1. Let $\pi = \nu_0, \nu_1, \ldots, \nu_k = u$, for $k \geq 1$, be a virtual path whose last node ν_k is a real node. Let $F(\pi) = u_0, u_1, \ldots, u_{k-1}, u$, where u_i is the beginning node of the real edge that contains the virtual edge (ν_i, ν_{i+1}) for $i = 0, \ldots, k-1$. We say that a real path P *follows* a virtual path π if $F(\pi)$ is a subsequence of P.

Proposition 4.2. *A protocol that simulates face routing in all virtual plane graphs guarantees message delivery in static connected graphs.*

Proof. The virtual plane graph of a connected graph is also connected, because adding virtual nodes and splitting an edge at a virtual node do not disconnect any path in the original graph. Because face routing guarantees message delivery in a static connected plane graph, the virtual path obtained by applying face routing in the virtual plane graph of a static connected graph leads to the destination node. By definition, a protocol that simulates face routing computes a real path that follows the virtual path and, hence, leads to the destination node. □

To simulate face routing in a virtual graph, there are two issues to be addressed. The first is to compute the virtual path, and the second is to find a real path in the network

that follows the virtual path. All the computation should be done in a distributed manner and use only local information available at each node, plus the information the protocol puts in the packet header.

The virtual path will be computed one edge at a time. A natural way to do this is to let the beginning node of the real edge containing the current virtual edge determine the next virtual edge. Given the current virtual edge (α, β), the next virtual edge (β, γ) is the first virtual edge after (β, α) in clockwise order around β, as illustrated in Figure 4.1. Suppose (α, β) is contained in the real edge (u, v), and (β, γ) is contained in the real edge

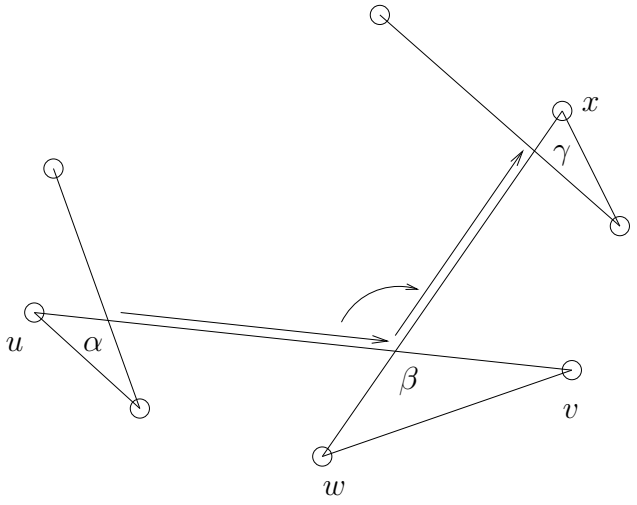


Figure 4.1: Computing the next virtual edge

(w, x), which intersects (u, v) at β. If node u knows its neighbors, which (real) edges intersect its incident (real) edges, and which (real) edges intersect them, then it is easy for u to compute (β, γ). However, if node u only knows its neighbors and the real edges that intersect its incident edges, node u cannot necessarily determine the ending point γ of the next virtual edge.

With this limited information, the virtual path can still be determined, if we distribute the computation in a different way. At each step of the traversal, given the beginning point α of the current virtual edge and the real edge (u, v) that contains it, node u computes the ending point β of the current virtual edge, which is also the beginning

point of the next virtual edge, and the real edge (w, x) that contains it. To find a real path that follows the virtual edge, node u computes a real path to w. Notice that, in this distributed computation of the edges in the virtual path, the two ends of a virtual edge may be determined at different real nodes: the beginning point of the virtual edge and its underlying real edge are determined in one step at a node, and its ending point is determined in the next step at a possibly different node.

Now we give an algorithm, EPND (Ending Point & Next Direction), which performs the computation in this procedure. The input parameters of EPND are (u, v) and α, where (u, v) is the real edge that contains the current virtual edge and α is the beginning point of the current virtual edge. So, α could be either node u or an interior point on edge (u, v). The output of EPND is β, the ending point of the current virtual edge, which is also the beginning point of the next virtual edge; (w, x), the edge that contains the next virtual edge; and π, a path from node u to node w, along which the packet can be forwarded to w to continue the traversal. Figure 4.2 illustrates the geometric relationships among the input and output parameters of EPND. The code for EPND is shown in Figure 4.3.

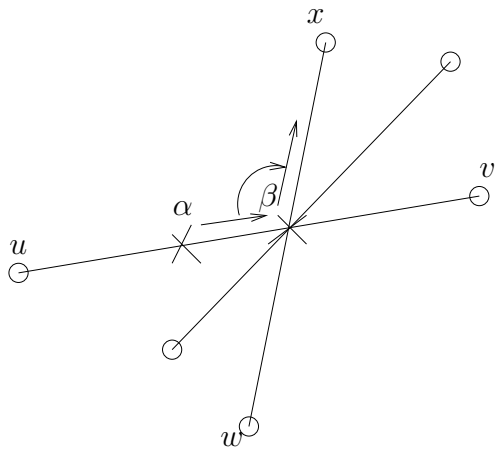


Figure 4.2: Illustration of the input and output parameters of Algorithm EPND

From the specifications of EPND and the definition of face routing, we get the following result.

Algorithm EPND$((u,v),\alpha,\beta,(w,x),\pi)$

▷ *Executed by node u.*

▷ *Input: edge (u,v) and a point α on (u,v);*

▷ *Output: β, the first node on (u,v) after α in the virtual graph; (w,x), the edge containing the first virtual edge after (β,u) in clockwise order around β; and π, a path from u to w in network graph.*

```
1  begin
2      if there is an edge that crosses (α, v) then   ▷ the next virtual edge is along the closest crossing edge
3          let β be the crossing point on (α, v) that is closest to α
4          let (w, x) be the edge that contains the next virtual edge after (β, α) in clockwise order
           around β
5          let π be a path from u to w in network graph
6      else                                   ▷ no edge crossing (α, v), the next edge begins at v
7          β ← v
8          (w, x) ← the next edge after (v, u) in clockwise order around v      ▷ note that w = v
9          π ← u, w
10 end
```

Figure 4.3: Algorithm EPND

Proposition 4.3. *If the input (u,v) contains the current virtual edge along the boundary of a virtual face and α is a point on that virtual edge, the output β of EPND is the ending point of the current virtual edge and (w,x) is the edge that contains the next virtual edge along the boundary of the same virtual face.*

Note that the if part in EPND specifies what the outputs should be without explicitly stating how to compute them, because it will depend on what information is available at node u and what is the network graph model. In Chapter 5 and Chapter 6, we will show that in unit disk graphs and quasi unit disk graphs, if node u knows the information about the real nodes within 2 hops and 3 hops from it, respectively, EPND can be implemented locally at node u. Furthermore, we will show that if less information is available at each node, the computation of EPND can be distributed among multiple nodes.

Once the ending point β of the current virtual edge is determined, node u checks whether (α, β) contains any point that is closer to the destination than the starting point of the current virtual face. If so, the traversal switches to the next virtual face; otherwise, node u forwards the packet to node w, and the traversal is continued.

Chapter 5

Routing in Unit Disk Graphs

One of our research goals is to apply face routing on a non-planar network graph directly without extracting a plane subgraph. If we can do this, nodes do not need to maintain information about the plane subgraph. More importantly, this will extend the application of face routing to graphs that do not contain a connected spanning plane subgraph.

In this chapter, we present two protocols that apply the ideas in Chapter 3 and 4 and prove their correctness. The first protocol is a simple implementation of the virtual face routing approach described in Chapter 4, which requires that each node has two hops neighbor information. The second one is a more clever version so that only one hop neighbor information is required at each node. We prove that both protocols simulate face routing in the virtual plane graph of connected unit disk graphs.

5.1 Geometric Properties of Unit Disk Graphs

We first study the properties of unit disk graphs to investigate what information is needed at each node to simulate face routing. The following lemma gives a geometric property of any two edges that cross each other in a unit disk graph.

Definition 5.1. For any edge (u, v), the *lens with chord* (u, v) is the intersection of the

two circles with unit radius that contain (u, v) as a chord.

The shaded area in Figure 5.1 is an example. Note that if a node is in the lens with chord (u, v), it is a neighbor of both u and v.

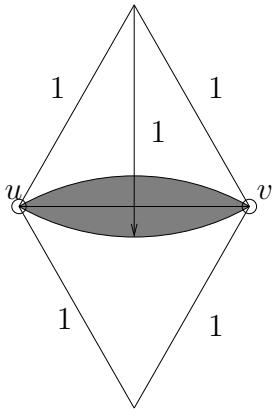


Figure 5.1: The lens with chord (u, v)

Lemma 5.2. *In a unit disk graph, if edge (x, y) intersects edge (u, v), and neither x nor y is in the lens with chord (u, v), then x and y are both neighbors of u or are both neighbors of v.*

Proof. In a unit disk graph, every edge has length at most 1. Consider Figure 5.2, in which the two straight lines are parallel to (u, v) and have distance 1 from (u, v), and the two circles are centered at u and v, respectively, with radius 1. Thus, the thick curve is the set of all points distance 1 from (u, v). Suppose (x, y) intersects (u, v) at point z. Then x is distance at most 1 from z and, hence, (u, v); and y is distance at most 1 from z and, hence, (u, v). Hence x and y are both located inside or on the thick curve.

Next, we will show that under the assumptions in the lemma, x is a neighbor of at least one of u and v. Suppose not. Then x is inside one of the two triangle-shaped areas that are not covered by the two circles, say, the one formed by the line segment between a and b, the arc between a and c, and the arc between b and c, in Figure 5.3.

By assumption, (x, y) intersects (u, v) and y is not in the lens with chord (u, v). Then, (x, y) must intersect the lower arc between u and v, say, at point y'. Also, (x, y) must

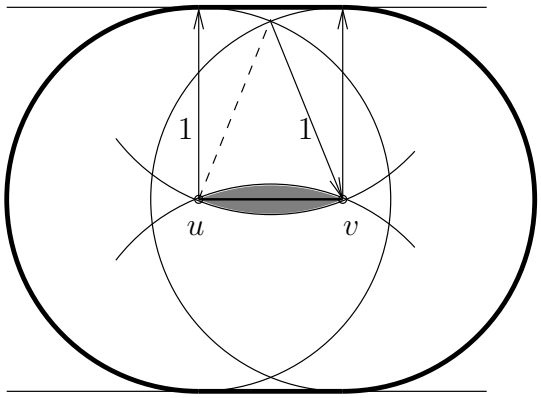


Figure 5.2: Proof of Lemma 5.2

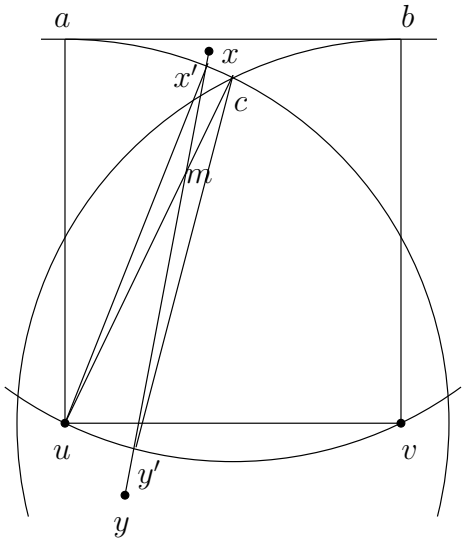


Figure 5.3: Proof of Lemma 5.2

intersect either the arc between a and c or the arc between b and c. Suppose (x, y) intersects the arc between a and c at point x', and (x, y) intersects the line segment between c and u at point m, as shown in Figure 5.3. From the triangle inequality, $|x'm| + |mu| \geq |ux'|$, and $|y'm| + |mc| \geq |cy'|$. We have that $|ux'| = |cy'| = 1$. So, $|x'm| + |mu| + |y'm| + |mc| = |x'y'| + |uc| \geq 2$. Since $|uc| = 1$, it follows that $|x'y'| \geq 1$. Because x is not a neighbor of u, $|xu| > 1$ so $|xx'| > 0$. Because y is not in the lens with chord (u, v), $|yy'| > 0$. Hence, $|xy| = |x'y'| + |xx'| + |y'y| > |x'y'| \geq 1$. This contradicts the fact that every edge in a unit disk graph has length at most 1. Therefore, x is a

neighbor of at least one of u and v. The same is true for y.

Finally, by contradiction, suppose neither u nor v is a neighbor of both x and y, say, x is a neighbor of u but not v, and y is a neighbor of v but not u, as illustrated in Figure 5.4.

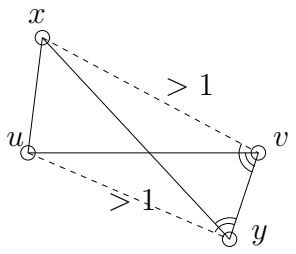


Figure 5.4: Proof of Lemma 5.2

Then we have $\angle yvu > \angle uyv$, because $|uy| > 1 \geq |uv|$. Also, because (x, y) intersects (u, v), $\angle xyv \leq \angle uyv$ and $\angle yvx \geq \angle yvu$. Thus, $\angle yvx > \angle xyv$, which yields $|xy| > |xv| > 1$. This contradicts the fact that $|xy| \leq 1$. Therefore, either u or v must be a neighbor of both x and y. □

Corollary 5.3. *In a unit disk graph, if edge (x, y) intersects edge (u, v), both x and y are at most two hops away from u and v.*

Proof. Case 1: Neither of x and y are in the lens with chord (u, v). From Lemma 5.2, either u or v is a neighbor of both x and y. Say, u is a neighbor of both x and y. Then x and y are one hop from u and at most two hops from v.

Case 2: At least one of x and y is in the lens with chord (u, v). Say, x is in the lens. Then, x is a neighbor of both u and v. Hence, y is at most two hops away from u and v. □

Corollary 5.3 implies that if each node has two-hop neighborhood information, it knows all the edges that intersect each of its incident edges. Thus, the EPND algorithm described in Chapter 4 can be implemented locally at each node.

5.2 Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info

Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info is a local routing protocol for unit disk graphs that illustrates the virtual face routing approach described in Chapter 4. It assumes that nodes have two hop neighbor information. It calls the EPND algorithm in Chapter 4 to compute the ending point of the current virtual edge and the direction of the next virtual edge. Face switching occurs when a point closer to the destination is found.

The overall idea of the Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info protocol is as follows. Starting at the source node s, compute the edge (s, v) that contains the first virtual edge. Then node s executes EPND$((s, v), s, \beta, (w, x), \pi)$ to compute the ending point β of the first virtual edge, which is the beginning point of the next virtual edge. It also computes the real edge (w, x) that contains the next virtual edge. Then, node s forwards the packet to node w, and w continues the traversal. This procedure continues until a point closer to the destination is found. Then a new starting point is determined and the traversal on the next virtual face starts from there.

Now we explain the components of Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info in more detail. We first describe the packet header fields the protocol uses to store information about the routing process. They include *packet.destination*, *packet.next_edge*, *packet.last_point*, and *packet.distance*. The destination node of the packet is stored in *packet.destination*. The *packet.next_edge* field stores the network edge that contains the next virtual edge along the boundary of the current virtual face. At each step, once *packet.next_edge* is computed, the packet is forwarded to the beginning node of *packet.next_edge*. The *packet.last_point* field stores the beginning point of the virtual edge contained in *packet.next_edge*. When the packet reaches the beginning node of *packet.next_edge*, that node uses the current value of *packet.next_edge* and

packet.last_point to call EPND and updates *packet.next_edge* and *packet.last_point* with the output from EPND. Finally, *packet.distance* stores the distance from the starting point of the current virtual face to the destination. This distance is used to check whether the traversal should switch to the next virtual face.

Next, we describe the algorithms called by the Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info protocol. At the source node or at an intermediate node where a new starting point is found, the node needs to determine the direction of the first virtual edge on the virtual face to be traversed. This is done in the algorithm INIT-UDG2HOP, shown in Figure 5.5. The input of INIT-UDG2HOP is the starting point of the virtual face. The first virtual edge is along the first edge from the line to the destination in clockwise order around the starting point. INIT-UDG2HOP assigns the edge that contains the first virtual edge to *packet.next_edge*, assigns the starting point to *packet.last_point*, and assigns the distance from the starting point to the destination to *packet.distance*.

Algorithm INIT-UDG2HOP(p)

▷ *Input: p, the starting point of the virtual face to be traversed*

▷ *Executed by a node that currently holds the packet and that has an incident edge containing p (which may be either endpoint of this edge).*

▷ *It computes the network edge that contains the first virtual edge along the boundary of the virtual face and sets up packet header.*

▷ *It requires that the node has two hop neighbor information.*

1 **begin**

2 $(v_1, v_2) \leftarrow$ the first edge in clockwise order around p starting from the line segment from p to *packet.destination*. ▷ *note that* $\angle(packet.destination)pv_2 < 180°$

3 *packet.next_edge* $\leftarrow (v_1, v_2)$

4 *packet.last_point* $\leftarrow p$

5 *packet.distance* $\leftarrow$ the distance from p to *packet.destination*

6 **end**

Figure 5.5: Algorithm INIT-UDG2HOP

From the definition of face routing and the right-hand rule, the first virtual edge along the boundary of the virtual face to be traversed is the first edge in clockwise order around the starting point starting from the line segment from the starting point to the destination node. Therefore we have the following result.

Proposition 5.4. *When INIT-UDG2HOP returns, packet.next_edge is the edge that contains the first virtual edge on the boundary of the virtual face to be traversed, and packet.last_point is the starting point on that virtual face.*

At the start of the routing process, the source node creates a packet containing the packet destination and executes INIT-UDG2HOP to initialize the rest of the packet header. It then executes Algorithm Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info, which is the main algorithm of the protocol, executed by each node during the routing process.

During the traversal along the boundary of the current virtual face, the beginning node of *packet.next_edge* calls EPND to compute the ending point of the current virtual edge and the edge that contains the next virtual edge. During the call of EPND, the current node u finds the edge crossing (u, v) that is closest to α, if one exists. From Corollary 5.3, when a node has two hop neighbor information, it can locally find all edges crossing its incident edges. Therefore, EPND can be performed locally.

In more detail, EPND can be implemented as follows. From Lemma 5.2, there are three (not necessarily disjoint) cases to consider for an edge crossing (u, v): (i) both endpoints of the edge are neighbors of node u; (ii) both endpoints of the edge are neighbors of node v; and (iii) one endpoint is a neighbor of both u and v and the other endpoint is neither a neighbor of u nor a neighbor of v. In the last case, one endpoint of the crossing edge is in the lens with chord (u, v), as shown in Figure 5.1. In any of these cases, an edge can be found locally by node u, because node u has two hop neighbor information. Therefore, node u checks all possible edges crossing (u, v) and finds the intersection point β that is closest to α, the edge (w, x) that contains the next virtual edge after (β, α) in

clockwise order around β, and a path π from u to w.

The path π returned by EPND has at most two hops, because both endpoints of an edge crossing (u, v) are at most two hops away from u by Corollary 5.3. If there is not any crossing edge, the beginning node of the edge containing the next virtual edge is v, which is a neighbor of u. Hence, in Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info, the path π returned by EPND is not stored in the packet header. Node u just checks whether the beginning node of *packet.next_edge* is its neighbor. If so, it forwards the packet to that node directly; otherwise, it forwards the packet to a node that is a neighbor of them both. When this intermediate node receives the packet, it will find that it is not the beginning node of *packet.next_edge*. In this case, it just relays the packet to that node.

Finally, we describe the procedure for face switching. Face switching occurs when the packet reaches a point on the boundary of the current virtual face that is closer to the destination than the current starting point is. Hence, after a node has computed the ending point of the virtual edge contained in *packet.next_edge*, it checks whether the distance from any point on this virtual edge to the destination node is less than the distance from the current starting point. If so, the closest point to the destination on this virtual edge will be the new starting point for the next virtual face to be traversed. The node calls INIT-UDG2HOP to find the edge containing the first virtual edge along the boundary of the next virtual face and update the packet header fields. Then the packet should be forwarded to the beginning node of the new *packet.next_edge* to start the traversal of the new virtual face. It is possible that the beginning node is just the current node. In this case, no packet forwarding is needed and the current node just executes Algorithm Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info again. If the beginning node of the new *packet.next_edge* is not the current node, it is at most two hops away, because both endpoints of the new *packet.next_edge* are within two hops of the current node. Hence, the current node forwards the packet to that node in the same way as we described before.

The complete code for Algorithm Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info is shown in Figure 5.6.

Theorem 5.5. *Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info simulates face routing in the virtual plane graph of connected unit disk graphs.*

Proof. Let π denote the virtual path obtained by applying face routing in the virtual plane graph. To prove the theorem, we need to show that the path computed by Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info follows the virtual path π, i.e., it contains the subsequence $F(\pi)$, defined in Chapter 4. Because the packet is always forwarded to the beginning node of the current *packet.next_edge* during the routing process, it suffices to prove that the sequence consisting of the beginning node of each successive value of *packet.next_edge* is $F(\pi)$.

At the source node or when a new starting point is found, INIT-UDG2HOP is called to compute the first virtual edge. From Proposition 5.4, when INIT-UDG2HOP returns, *packet.next_edge* is the edge that contains the first virtual edge on the boundary of the virtual face to be traversed. At the beginning node of each *packet.next_edge*, EPND is called to compute the next edge. From Proposition 4.3, the output (w, x) from EPND is the edge containing the next virtual edge along the boundary of the current virtual face. If no new starting point is found, (w, x) is assigned to *packet.next_edge*. Otherwise, the traversal switches to the next virtual face. Therefore, during the traversal, *packet.next_edge* is always the edge that contains each virtual edge in the virtual path π. Hence, the sequence of beginning nodes of *packet.next_edge* is $F(\pi)$. Thus Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info simulates face routing in the virtual plane graph. □

Algorithm Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info

▷ *Performed by node u, the node that currently holds the packet.*

▷ *The information stored in the packet header is:*

- *packet.destination, the destination node of the packet;*
- *packet.next_edge, the network edge containing the next virtual edge;*
- *packet.last_point, the ending point of the previous virtual edge or the starting point on the current virtual face if packet.next_edge contains the first virtual edge on that face; and*
- *packet.distance, the distance from the starting point of the current virtual face to the destination node.*

```
1  begin
2      if u is packet.destination then
3          release the packet; return
4      if packet.destination is a neighbor of u then
5          forward the packet to packet.destination; return
6      if u is not the beginning node of packet.next_edge then
7          forward the packet to the beginning node of packet.next_edge; return
       ▷ u is the beginning node of packet.next_edge
8      let v denote the ending node of packet.next_edge
9      EPND((u, v), packet.last_point, β, (w, x), π)
10     d ← shortest distance from a point on edge (packet.last_point, β) to packet.destination
11     if d < packet.distance then            ▷ switch faces
12         p ← the closest point to packet.destination on edge (packet.last_point, β)
13         INIT-UDG2HOP(p)
14         if u is the beginning node of packet.next_edge then
15             Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info
16             return
17     else
18         packet.next_edge ← (w, x)
19         packet.last_point ← β
       ▷ route the packet to the beginning node of packet.next_edge
20     let y denote the beginning node of packet.next_edge
21     if y is a neighbor of u then
22         forward the packet to y
23     else
24         find node z such that z is a neighbor of both u and y   ▷ z is either v or the ending node of
           packet.next_edge
25         forward the packet to z
26     return
27 end
```

Figure 5.6: Algorithm Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info

5.3 Virtual-Face-Traversal-For-UDG-With-One-Hop-Info

Virtual-Face-Traversal-For-UDG-With-One-Hop-Info is another distributed local routing protocol for unit disk graphs that applies the virtual face routing approach. It assumes that nodes have only one hop neighbor information. With only one hop neighbor information, a node cannot find all edges crossing its incident edges without exchanging information with its neighbors. One simple solution to this problem is that, when a node has a packet to be forwarded, it first sends a query to all its neighbors to collect the information about its two-hop neighbors. Then the node has sufficient information to compute all virtual nodes on its incident edges and, thus, to compute the boundary of the current virtual face.

Virtual-Face-Traversal-For-UDG-With-One-Hop-Info uses a different approach. Face routing repeatedly computes the next edge along the boundary of the current face. In a virtual plane graph, this becomes the computation of the underlying network edge that contains the next virtual edge along the boundary of the current virtual face. Recall that, in Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info, this is done by calling EPND at the beginning node of *packet.next_edge*. In Virtual-Face-Traversal-For-UDG-With-One-Hop-Info, this computation is carried out collectively by the beginning node of *packet.next_edge*, the nodes inside the lens with chord *packet.next_edge*, if any, and the ending node of *packet.next_edge*. This requires less communication than having a node query its neighbors about its two-hop neighbors.

To compute the next edge, we want to find the first virtual node on (α, v) after α, if one exists. As discussed in section 5.2, there are three cases for an edge crossing (u, v): (i) both endpoints of the edge are neighbors of node u; (ii) both endpoints of the edge are neighbors of node v; and (iii) one endpoint is a neighbor of both u and v and the other endpoint is neither a neighbor of u nor a neighbor of v. With one hop neighbor

information, node u can find all edges that belong to case (i), node v can find all edges that belong to case (ii), and nodes inside the lens with chord (u, v) can collectively find all edges that belong to case (iii). Therefore, the computation of finding the next edge is divided into three stages. A packet header field, *packet.mode*, is used to indicate the stage of the computation. In each stage, one case is checked and a candidate for the next edge is computed. When the packet reaches node v, the ending node of *packet.next_edge*, all edges crossing (α, v) have been considered and node v can determine the new value of *packet.next_edge*. To prove the correctness, we show that the new value of *packet.next_edge* is the same as computed by node u in Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info.

Now we describe the computation in detail. When the routing process starts at the source node, the source node creates a packet containing the packet destination and executes the INIT-UDG1HOP algorithm, shown in Figure 5.7, to compute the edge containing the first virtual edge along the boundary of the virtual face to be traversed and initialize the rest of the packet header. Note that in INIT-UDG2HOP, the initialization algorithm used in the previous section, the starting point can be any point on an edge incident to the current node, but it requires that the node has two hop neighbor information. The INIT-UDG1HOP algorithm can only be applied when the starting point is at the current node. Later, we will explain how to compute the edge containing the first virtual edge and update the packet header when the starting point is not at a node.

The computation of finding the next edge begins when the current node is the beginning node of *packet.next_edge* and *packet.mode* is "$traversing$". At this stage, the algorithm is concerned with crossing edges that belong to case (i). Let (u, v) denote edge *packet.next_edge*, and let α denote point *packet.last_point*, the beginning point of the current virtual edge, which may be either node u or an interior point on edge (u, v). Node u calls a subroutine, LCC (Local Closest Crossing), to find the edges crossing (α, v) both of whose endpoints are neighbors of u and to compute a candidate for the next edge.

Algorithm INIT-UDG1HOP

▷ *Executed by node u that currently holds the packet and is the starting point of the virtual face to be traversed.*

▷ *It computes the network edge that contains the first virtual edge along the boundary of the virtual face and sets up packet header.*

1 **begin**

2 $(u, v) \leftarrow$ the first edge in clockwise order around u starting from the line segment from u to *packet.destination*

3 *packet.next_edge* $\leftarrow (u, v)$

4 *packet.last_point* $\leftarrow u$

5 *packet.distance* $\leftarrow$ the distance from u to *packet.destination*

6 *packet.mode* $\leftarrow$ "*traversing*"

7 **end**

Figure 5.7: Algorithm INIT-UDG1HOP

A packet header field, *packet.closest_point*, is used to store the next node after α on (α, v) among those real and virtual nodes which are currently known by the node that has the packet. It is initialized to v by node u. In LCC, node u finds all the edges it sees that cross (α, v). If there exist such edges, it computes the intersection points of these edges with (α, v) and determines which is closest to α. This point is assigned to *packet.closest_point*. Another packet header field, *packet.crossing_edge*, is used to store the current candidate for the next edge. It is initialized to *null* by node u. Among the edges known to node u that cross (α, v) at *packet.closest_point*, LCC computes the edge that contains the next virtual edge after (*packet.closest_point*, α) in clockwise order around *packet.closest_point*. This edge is the current candidate for the next edge and is assigned to *packet.crossing_edge*. The code for the LCC algorithm is shown in Figure 5.8.

Next, node u sets *packet.mode* to "*on side path*" and stores the list of the nodes in the lens area, excluding itself but including the ending node v of *packet.next_edge*, in the packet header field *packet.side_path*. This list contains the nodes in nondecreasing order

Algorithm LCC (Local Closest Crossing)

▷ *Executed by node z that currently holds the packet, which can be the beginning node of packet.next_edge, the ending node of packet.next_edge, or a node in the lens with chord packet.next_edge.*

▷ *It computes the first real or virtual node on packet.next_edge after packet.last_point that is known to the current node. It also computes a candidate for the next edge and updates packet.closest_point and packet.crossing_edge.*

1 **begin**
2 **let** (u, v) denote *packet.next_edge* and **let** α denote *packet.last_point*
3 **if** $z = u$ or $z = v$ **then**
4 **let** $E = \{(v_1, v_2) \mid v_1$ and v_2 are neighbors of z and (v_1, v_2) is an edge that crosses $(\alpha, v)\}$
5 **else**
6 **let** $E = \{(z, v') \mid$
v' is a neighbor of z and not a neighbor of either u or v and (z, v') crosses $(\alpha, v)\}$
7 **if** E is not empty and *packet.crossing_edge* is not *null* **then**
8 $E \leftarrow E \cup \{$*packet.crossing_edge*$\}$
9 **if** E is not empty **then**
▷ *update packet.closest_point and packet.crossing_edge*
10 $\beta \leftarrow$ the closest point to α at which an edge in E intersects (α, v)
11 $(w, x) \leftarrow$ the edge in E intersecting (α, v) at β that contains the next virtual edge after (β, α) in clockwise order around β ▷ *note that* $\angle\alpha\beta x < 180°$
12 *packet.closest_point* $\leftarrow \beta$
13 *packet.crossing_edge* $\leftarrow (w, x)$
14 **end**

Figure 5.8: Algorithm LCC

of their distance to node u. It always ends with node v. Node u then forwards the packet to the first node in this list.

When a node in the lens area receives the packet with *packet.mode* = "*on side path*" and it is not node v, it calls LCC to recompute the values of *packet.closest_point* and *packet.crossing_edge*, taking account of possible edges crossing (α, v) that are seen by the current node but were not seen by the previous nodes. LCC finds every edge incident to the current node that crosses (α, v) and whose other endpoint is not a neighbor of either u or v. Such an edge belongs to case (iii). If there are such edges, LCC computes the intersection points of these edges with (α, v) and, among these points and the point stored in *packet.closest_point*, LCC finds the point closest to α and assigns it to *packet.closest_point*. This point is the virtual node on (α, v) currently known to be closest to α. Then, among the edges that intersect (α, v) at this point, which might include *packet.crossing_edge*, LCC finds the edge containing the next virtual edge after $(packet.closest\_point, \alpha)$ in clockwise order around *packet.closest_point* and assigns it to *packet.crossing_edge*. Then the packet is forwarded to the next node in *packet.side_path*.

When the packet reaches the ending node v of *packet.next_edge*, which is the last node in *packet.side_path*, all crossing edges that belong to case (i) or (iii) have been considered as the candidate for the next edge. Node v sets *packet.mode* to "$finding\ next\ edge$". Then it calls LCC to consider the crossing edges seen by node v that belong to case (ii) and update *packet.closest_point* and *packet.crossing_edge* if necessary.

When LCC returns at node v, all three cases for edges crossing the interior of (α, v) have been considered. Therefore, we have the following proposition about the LCC algorithm.

Proposition 5.6. *If there exist edges crossing the interior of* (α, v)*, then after node* v *performs LCC, packet.closest_point is* β*, the node on* (α, v) *that is closest but not equal to* α*, and packet.crossing_edge is the edge containing the next virtual edge after* (β, α) *in clockwise order around* β*.*

Proof. Let (w, x) be the edge that contains the next virtual edge after (β, α) in clockwise order around β. From Lemma 5.2, edge (w, x) belongs to one of the three cases: (i) both w and x are neighbors of node u; (ii) both w and x are neighbors of node v; and (iii) one of w and x is a neighbor of both u and v and the other is neither a neighbor of u nor a neighbor of v. Therefore, edge (w, x) belongs to the edge set E in LCC when LCC is performed at one or more nodes in the lens with chord (u, v), including u and v. At the first such node, LCC assigns β to *packet.closest_point* and (w, x) to *packet.crossing_edge*. In the subsequent calls to LCC, *packet.closest_point* and *packet.crossing_edge* do not change, because *packet.crossing_edge* is always considered in the computation. Therefore, after node v performs LCC, *packet.closest_point* is β and *packet.crossing_edge* is (w, x). □

If no edge crosses the interior of (α, v), *packet.closest_point* and *packet.crossing_edge* are not updated in LCC. Therefore, after LCC returns at node v, *packet.closest_point* is v and *packet.crossing_edge* is *null*, which are their initial values. In this case, node v is the node on (α, v) that is closest but not equal to α. Therefore, whether there are edges crossing the interior of (α, v) or not, after LCC returns at node v, *packet.closest_point* is the ending node β of the current virtual edge.

After performing LCC, node v checks whether there is a point on the virtual edge (α, β) that is closer to the destination than the starting point is. First, suppose there is no such point. Then the packet will continue to travel along the boundary of the current face. Node v assigns *packet.closest_point* to *packet.last_point*. If *packet.crossing_edge* is not *null*, node v assigns *packet.crossing_edge* to *packet.next_edge*. Otherwise, node v sets *packet.next_edge* to the next edge after (v, u) in clockwise order around v.

From the above description and the definition of face routing, we get the following result.

Proposition 5.7. *When no face switching occurs, the new value of packet.last_point is the ending node of the current virtual edge and the new value of packet.next_edge is the edge that contains the next virtual edge along the boundary of the same virtual face.*

If there exists a point on the virtual edge (α, β) that is closer to the destination than the starting point is, the point p on edge (α, β) that is closest to the destination will be the new starting point for the next virtual face to be traversed, and the traversal switches to the next virtual face. In this case, node v needs to find the network edge that contains the first virtual edge along the boundary of the next virtual face. We call this network edge the *first edge*.

There are three cases to consider: (i) p is the real node v (i.e., $p = \beta = v$), (ii) p is a virtual node (i.e., $p = \beta \neq v$), and (iii) p is not at a node (i.e., $p \neq \beta$). In the first case, the current node v is the starting point of the next virtual face. Then, node v calls INIT-UDG1HOP. Recall that INIT-UDG1HOP computes the first edge when the current node is the starting point and it also updates packet header fields.

The second case is when β is a virtual node and p is at β. Then, the first edge is the network edge that contains the first virtual edge in clockwise order around β starting from the line segment between β and *packet.destination*. Figure 5.9 illustrates an example of this case. In this example, the first edge is edge (x, w). However, node v might not know all the edges that cross (u, v) at β. To handle this problem, Virtual-Face-Traversal-For-UDG-With-One-Hop-Info uses *packet.first_edge* to keep track of a candidate for the first edge. Specifically, after a node calls LCC, if *packet.closest_point* is a virtual node and its distance to the destination is less than *packet.distance*, the node calls LFE (Local First Edge) to compute a value for *packet.first_edge*. LFE finds the edge that contains the first virtual edge in clockwise order around *packet.closest_point* starting from the line segment between *packet.closest_point* and *packet.destination* among those edges the current node knows about. This edge is stored as the new value of *packet.first_edge*. The code for LFE is shown in Figure 5.10. Initially, *packet.first_edge* = *null*. After node v performs LCC and has determined that $p = \beta \neq v$, it also calls LFE. The following proposition shows that when LFE returns at node v, *packet.first_edge* is the first edge. Therefore, node v assigns *packet.first_edge* to *packet.next_edge*.

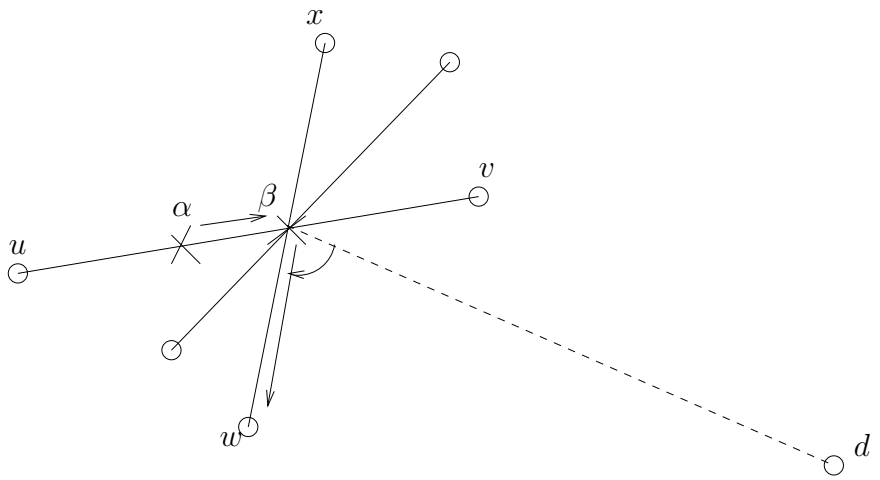


Figure 5.9: Example of the first edge when more than one edge cross (u, v) at β

Algorithm LFE (Local First Edge)

▷ *Executed by node z that currently holds the packet.*

1 **begin**

2 **let** β denote *packet.closest_point*

3 **let** E be the set of the (directed) network edges containing β that are known to z

4 **if** *packet.first_edge* is not *null* **then**

5 $E \leftarrow E \cup \{$*packet.first_edge*$\}$

6 *packet.first_edge* ← the edge in E that contains the first virtual edge in clockwise order around β starting from the line segment between β and *packet.destination*

7 **end**

Figure 5.10: Algorithm LFE

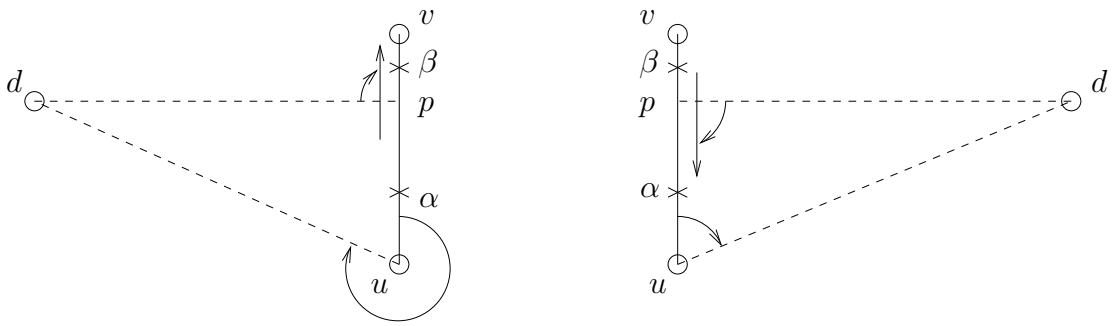
Proposition 5.8. *If $p = \beta \neq v$, after node v performs LFE, packet.first_edge is the edge containing the first virtual edge in clockwise order around β starting from the line segment between β and packet.destination.*

Proof. Let (v_1, v_2) be the edge containing the first virtual edge in clockwise order around β starting from the line segment between β and *packet.destination*. Edge (v_1, v_2) may be (u, v), (v, u), or an edge crossing (α, v) at β. Any edge crossing (α, v) is known to one or more nodes in the lens with chord (u, v), including u and v. Therefore, β and (v_1, v_2)

are known to at least one of these nodes.

After the first node z that knows about β and (v_1, v_2) calls LCC, *packet.closest_point* is β. Because *packet.closest_point* is a virtual node and its distance to the destination is less than *packet.distance*, node z calls LFE and, after LFE returns, *packet.first_edge* is (v_1, v_2). In subsequent calls to LFE, *packet.first_edge* does not change, by definition. Therefore, after node v performs LFE, *packet.first_edge* is the edge containing the first virtual edge in clockwise order around β starting from the line segment between β and *packet.destination*. □

The third case of the new starting point is when p is an interior point on edge (α, β). Then the first edge is either (u, v) or (v, u) – the direction depends on the location of the destination node with respect to (u, v). As illustrated in Figure 5.11, if the destination node is on the left side of (u, v) (i.e. $\angle vud > 180°$), the first edge is (u, v); if the destination node is on the right side of (u, v) (i.e. $\angle vud < 180°$), the first edge is (v, u). Since the current value of *packet.next_edge* is (u, v), *packet.next_edge* is updated to (v, u) in the latter case.



(a) destination d is on the left side of (u, v) (b) destination d is on the right side of (u, v)

Figure 5.11: The direction of the first edge when p is an interior point on edge (α, β)

Thus, in every case, when a new starting point is found, node v can find the first edge and update *packet.next_edge*. It also updates *packet.last_point* and *packet.distance* with the new starting point and its distance to the destination. From the definition of face

routing and the right-hand rule, we have the following result.

Proposition 5.9. *When a new starting point p is found, the new value of packet.next_edge is the edge that contains the first virtual edge on the boundary of the virtual face to be traversed, and the new value of packet.last_point is the starting point on that virtual face.*

Whether or not a new starting point is found, *packet.next_edge* is now the network edge containing the next virtual edge along the virtual path. Node v then changes *packet.mode* back to "*traversing*".

If the beginning node of *packet.next_edge* is now node v, the computation of the next edge is complete. However, the beginning node could also be a neighbor of v or a node two hops away from v. In these cases, the packet is routed to the beginning node of *packet.next_edge* either directly or via an intermediate node. If an intermediate node receives the packet, it knows to forward the packet to the beginning node of *packet.next_edge*, since *packet.mode* is "*traversing*".

The stages of the computation are illustrated in Figure 5.12. In the diagram, a solid arrow line indicates that the packet is forwarded to another node, and a dashed arrow line indicates that the packet stays at the same node.

The code for Algorithm Virtual-Face-Traversal-For-UDG-With-One-Hop-Info, which is the main algorithm of the protocol, executed by each node during the routing process, is shown in Figure 5.13, 5.14, and 5.15. Note that the algorithm can be optimized to use fewer packet header fields. For example, the stage of the computation can be inferred from *packet.side_path*, therefore it is not necessary to have the *packet.mode* field in the packet header. Furthermore, the candidate for the next edge can be stored directly in *packet.next_edge*, which will have the correct value at the end of the computation, so *packet.crossing_edge* is not necessary. To make the code clearer and easier to prove correct, we do not make these optimizations.

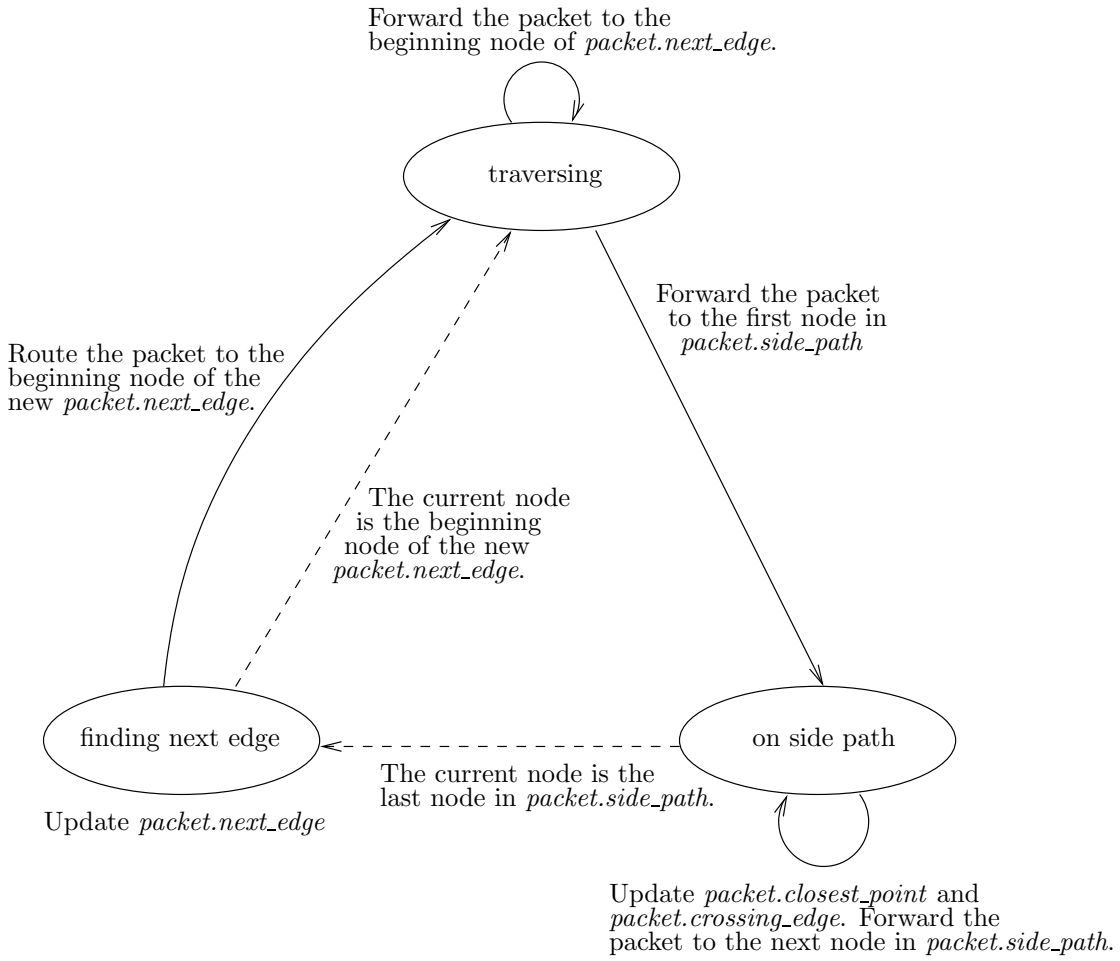


Figure 5.12: Stage transition of the computation in Virtual-Face-Traversal-For-UDG-With-One-Hop-Info

Algorithm Virtual-Face-Traversal-For-UDG-With-One-Hop-Info

▷ *Performed by node z, the node that currently holds the packet.*

▷ *The information stored in the packet header is as follows:*

- *packet.destination, the destination node of the packet;*
- *packet.next_edge, the network edge containing the next virtual edge;*
- *packet.last_point, the ending point of the previous virtual edge or the starting point on the current virtual face if packet.next_edge contains the first virtual edge on that face;*
- *packet.distance, the distance from the starting point of the current virtual face to the destination node;*
- *packet.mode, the stage of the computation;*
- *packet.side_path, the list of the nodes in the lens with chord packet.next_edge including the ending node of packet.next_edge, in non-decreasing order of their distance to the beginning node of packet.next_edge;*
- *packet.closest_point, the closest virtual node after packet.last_point that is known to the current node;*
- *packet.crossing_edge, the candidate for the next edge; and*
- *packet.first_edge, the network edge containing the first virtual edge in clockwise order around packet.closest_point starting from the line segment from packet.closest_point to packet.destination.*

```
1  begin
2      if z is packet.destination then
3          release the packet; return
4      if packet.destination is a neighbor of z then
5          forward the packet to packet.destination; return
6      let (u, v) denote the edge packet.next_edge
7      if packet.mode = "on side path" then
8          if z is not the last node in packet.side_path then
9              LCC
10             if packet.closest_point ≠ v and
               |(packet.closest_point)(packet.destination)| < packet.distance then
11                 LFE
12             forward the packet to the next node in packet.side_path after z; return
13         else
14             packet.mode ← "finding next edge"
```

(Continued on the next page)

Figure 5.13: Algorithm Virtual-Face-Traversal-For-UDG-With-One-Hop-Info

Algorithm Virtual-Face-Traversal-For-UDG-With-One-Hop-Info (Con't)

```
15      if packet.mode = "finding next edge" then
            ▷ z = v, the ending node of packet.next_edge
16          LCC
            ▷ packet.closest_point is the ending point of the current virtual edge contained in packet.next_edge
17          d ← shortest distance from a point on (packet.last_point, packet.closest_point) to
            packet.destination
18          if d < packet.distance then                          ▷ switch faces
19              p ← the closest point to packet.destination on (packet.last_point, packet.closest_point)
20              if p is v then
21                  INIT-UDG1HOP
22                  Virtual-Face-Traversal-For-UDG-With-One-Hop-Info; return
23              else if p is packet.closest_point then           ▷ packet.closest_point is a virtual node
24                  LFE
25                  packet.next_edge ← packet.first_edge
26              else                ▷ p is a point on (packet.last_point, packet.closest_point)
27                  if ∠vu(packet.destination) < 180° then       ▷ packet.destination is at the right side of (v, u)
28                      packet.next_edge ← (v, u)
29              packet.last_point ← p
30              packet.distance ← d
31          else                    ▷ next virtual edge is along the boundary of the current virtual face
32              if packet.crossing_edge is null then             ▷ no crossing edge
33                  packet.next_edge ← the next edge after (v, u) in clockwise order around v
34                  packet.last_point ← v
35              else
36                  packet.next_edge ← packet.crossing_edge
37                  packet.last_point ← packet.closest_point
            ▷ route the packet to the beginning node of packet.next_edge
38          packet.mode ← "traversing"
39          if z is not the beginning node of packet.next_edge then
40              let u′ denote the beginning node of packet.next_edge
41              if u′ is a neighbor of z then
42                  forward the packet to u′; return
43              else
44                  find node y such that y is a neighbor of both z and u′
45                  forward the packet to y; return
```

(Continued on the next page)

Figure 5.14: Algorithm Virtual-Face-Traversal-For-UDG-With-One-Hop-Info (con't)

Algorithm Virtual-Face-Traversal-For-UDG-With-One-Hop-Info (Con't)

```
46    if packet.mode = "traversing" then
47        if z is not the beginning node of packet.next_edge then        ▷ in transit
48            forward the packet to the beginning node of packet.next_edge; return
          ▷ z = u, the beginning node of packet.next_edge
49        packet.closest_point ← v
50        packet.crossing_edge ← null
51        LCC
52        packet.first_edge ← null
53        if packet.closest_point ≠ v and |(packet.closest_point)(packet.destination)| < packet.distance
          then
54            LFE
55        let v_1, ··· , v_m be the nodes inside the lens with chord packet.next_edge in non-decreasing
          order of their distance to u
56        packet.mode ← "on side path"
57        packet.side_path ← v_1, ··· , v_m, v_{m+1} = v
58        forward the packet to v_1; return
59  end
```

Figure 5.15: Algorithm Virtual-Face-Traversal-For-UDG-With-One-Hop-Info (con't)

From Proposition 4.2, the following result suffices to prove that Virtual-Face-Traversal-For-UDG-With-One-Hop-Info guarantees message delivery in connected unit disk graphs.

Theorem 5.10. *Virtual-Face-Traversal-For-UDG-With-One-Hop-Info simulates face routing in the virtual plane graph of connected unit disk graphs.*

Proof. By Theorem 5.5, it suffices to prove that the sequence consisting of the successive values of *packet.next_edge* is the same in both Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info and Virtual-Face-Traversal-For-UDG-With-One-Hop-Info algorithms.

At the source node s, *packet.next_edge* is initialized by INIT-UDG2HOP(s) in Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info and by INIT-UDG1HOP in Virtual-Face-Traversal-For-UDG-With-One-Hop-Info. In both cases, *packet.next_edge* is set to the first edge in clockwise order around s starting from the line segment from s to *packet.destination*. Furthermore, *packet.last_point* is initialized to the source node s in

both algorithms.

In Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info, *packet.next_edge* and *packet.last_point* are updated at node u. If no face switching occurs, they are updated with the edge (w, x) and the node β returned by EPND. Thus, from Proposition 4.3 and Proposition 5.7, the new values of *packet.next_edge* and *packet.last_point* are the same in both algorithms. If face switching occurs, it follows from Proposition 5.4 and Proposition 5.9 that the new values of *packet.next_edge* and *packet.last_point* are the same in both algorithms.

Therefore, the sequence consisting of the successive values of *packet.next_edge* is the same in both algorithms. □

Chapter 6

Routing in Quasi Unit Disk Graphs

In this chapter, we will present two routing protocols for quasi unit disk graphs that apply face routing on the network graph directly. Similar to the two protocols presented in Chapter 5, in the first protocol, each node maintains enough information so that it can compute the next edge on which to route a packet; in the second protocol, this computation is distributed to multiple nodes so as to reduce the amount of information required at each node. We take advantage of some geometric properties of quasi unit disk graphs to make the second protocol quite simple and efficient. Before describing the protocols, we first show these geometric properties of quasi unit disk graphs in section 6.1.

6.1 Geometric Properties of Quasi Unit Disk Graphs

Our study of routing protocols for the quasi unit disk graph model and all the subsequent models that contain it as a special case focuses on graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$. The design of our routing protocols is based on a geometric property of intersecting edges in quasi unit disk graphs, whose proof, in turn, uses a property of chords of a circle.

Proposition 6.1. *Let AB and CD be two chords and let d_{AB} and d_{CD} denote their distance to the center, respectively. If $d_{AB} \leq d_{CD}$, then $|AB| \geq |CD|$.*

Proof. The perpendicular bisector of a chord of a circle passes through the center of the circle. From this property and the Pythagorean theorem, we have $(\frac{1}{2}|AB|)^2 + d_{AB}^2 = (\frac{1}{2}|CD|)^2 + d_{CD}^2 = (\text{radius of the circle})^2$. Therefore, if $d_{AB} \leq d_{CD}$, then $|AB| \geq |CD|$. □

Lemma 6.2. *In a quasi unit disk graph with $\varepsilon \geq \frac{1}{\sqrt{2}}$, if edge (x, y) intersects edge (u, v) at a point q between u and the midpoint m of (u, v), then at least one of x and y is a neighbor of u.*

Proof. To obtain a contradiction, suppose both x and y are not a neighbor of u. From the definition of quasi unit disk graphs, both x and y are outside of the circle centered at u with radius ε. Let x' and y' be the intersections of (x, y) with that circle, so $|xy| > |x'y'|$. See Figure 6.1.

Because the length of any edge in quasi unit disk graphs is at most 1, $|um| \leq 1/2$. Consider the circle centered at u with radius ε. Let ab be the chord of the circle that is perpendicular to (u, v) and intersects (u, v) at m. Then m is the midpoint of ab, $|um|^2 + |am|^2 = \varepsilon^2$, $\varepsilon \geq \frac{1}{\sqrt{2}}$, and $|um| \leq 1/2$. Thus $|am| \geq 1/2$ and $|ab| \geq 1$.

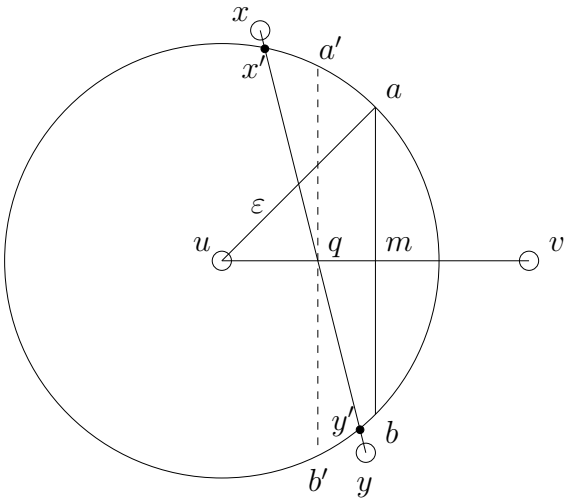


Figure 6.1: Proof of Lemma 6.2

Let $a'b'$ be the chord of the circle centered at u with radius ε that is perpendicular to (u, v) and intersects (u, v) at q. The distance from u to $a'b'$ is $|uq|$. Because $x'y'$

intersects (u, v) at q, the distance from u to chord $x'y'$ is less than or equal to $|uq|$. From Proposition 6.1, $|x'y'| \geq |a'b'|$. Furthermore, since q is between u and m, $|uq| \leq |um|$. From Proposition 6.1, $|a'b'| \geq |ab|$.

Therefore, we get $|xy| > |x'y'| \geq |ab| \geq 1$. This contradicts the fact that (x, y) is an edge in a quasi unit disk graph. Hence, at least one of x and y is a neighbor of u. □

Corollary 6.3. *In a quasi unit disk graph with $\varepsilon \geq \frac{1}{\sqrt{2}}$, if edge (x, y) intersects edge (u, v), then at least one endpoint of (x, y) is at most two hops away from u and v.*

Proof. Without loss of generality, suppose edge (x, y) intersects edge (u, v) at a point between u and the midpoint of (u, v). From Lemma 6.2, at least one of x and y is a neighbor of u and thus one hop away from u and at most two hops away from v. □

From Corollary 6.3, we know that if a node has three-hop neighborhood information, i.e., it knows the set of nodes that are at most three-hops away and the locations of these nodes, it can determine all the edges that intersect each of its incident edges. Thus, each node has sufficient information to locally perform the computation in the EPND algorithm and to simulate face routing in the virtual plane graph of a quasi unit disk graph with $\varepsilon \geq 1/\sqrt{2}$.

6.2 Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info

Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info is a simple virtual face routing protocol for quasi unit disk graphs with $\varepsilon \geq 1/\sqrt{2}$. It assumes that nodes have three-hop neighbor information, so that computing the next virtual edge along the boundary of the current virtual face is done locally at a single node. The main idea of Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info is the same as for the Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info protocol. At each step during the routing process, the

packet is initially at the beginning node of the network edge that contains the current virtual edge. The node calls the EPND algorithm to compute the ending point of the current virtual edge and the network edge containing the next virtual edge along the boundary of the current virtual face. It then checks whether there is a point on the current virtual edge that is closer to the destination than the current starting point is. If so, the traversal switches to the next virtual face. Otherwise, the packet continues to travel along the boundary of the current virtual face. In both cases, the packet is forwarded to the beginning node of the network edge containing the next virtual edge to be traversed.

Most parts of the Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info protocol are the same as the Virtual-Face-Traversal-For-UDG-With-Two-Hop-Info protocol. The main differences are in the implementation of the initialization algorithm and the EPND algorithm.

At the source node or when a new starting point is found, INIT-QUDG3HOP is called to compute the network edge that contains the first virtual edge on the boundary of the virtual face to be traversed and to set up the packet header. The high-level code of INIT-QUDG3HOP is shown in Figure 6.2, which is almost the same as INIT-UDG2HOP. The implementation of line 2 of INIT-QUDG3HOP is slightly different. To find the network edge that contains the first virtual edge in clockwise order around p starting from the line segment between p and *packet.destination*, the current node considers all the edges that contain point p. When the starting point p is a virtual node, the current node needs to know all the edges intersecting p. From Corollary 6.3, this requires that the node knows three-hop neighbor information. Therefore, the current node considers the edges in its three-hop neighborhood that intersect p. The other difference is that there is one more field, *packet.transit_path*, in the packet header. This field is used to route the packet from the current node to the beginning node of *packet.next_edge*.

From Corollary 6.3 and the definition of face routing, we have the following result

Algorithm INIT-QUDG3HOP(p)

▷ *Input: p, the starting point of the virtual face to be traversed*

▷ *Executed by a node that currently holds the packet and that has an incident edge containing p (which may be any point on this edge including both endpoints).*

▷ *It computes the network edge that contains the first virtual edge along the boundary of the virtual face and sets up packet header.*

▷ *It requires that the node has three-hop neighbor information.*

1 **begin**

2 $(v_1, v_2) \leftarrow$ the network edge containing the first edge in clockwise order around p starting from the line segment between p and *packet.destination*.

3 *packet.next_edge* $\leftarrow (v_1, v_2)$

4 *packet.last_point* $\leftarrow p$

5 *packet.distance* $\leftarrow$ the distance from p to *packet.destination*

6 *packet.transit_path* $\leftarrow$ a path from the current node to v_1 ▷ *at most 3 hops*

7 **end**

Figure 6.2: Algorithm INIT-QUDG3HOP

similar to Proposition 5.4.

Proposition 6.4. *When INIT-QUDG3HOP returns, packet.next_edge is the edge that contains the first virtual edge on the boundary of the virtual face to be traversed, and packet.last_point is the starting point on that virtual face.*

Similarly, to implement EPND, node u considers the edges in its three-hop neighborhood to find the virtual or real node on (α, v) that is closest to α. From Corollary 6.3, one endpoint of each edge crossing (α, v) is within two hops of node u. Therefore, EPND can be performed locally at node u.

The path π returned by EPND is used to route the packet to the beginning node of *packet.next_edge*. It contains at most three hops, and it is assigned to *packet.transit_path* after EPND returns. The current node u removes the first node in *packet.transit_path*, which is u, and forwards the packet to the new first node in *packet.transit_path*. If that node is not the beginning node of *packet.next_edge*, it continues to forward the packet along the path in *packet.transit_path*.

Alternatively, we can avoid using the extra packet header field *packet.transit_path* in this algorithm. Instead, the nodes on the path from the current node to the beginning node of *packet.next_edge* may compute the path on the fly. Each node has sufficient information to carry out this computation locally. Therefore, it is a tradeoff between communication overhead and computation time.

The code for Algorithm Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info is shown in Figure 6.3, which is the main algorithm of the protocol.

Theorem 6.5. *Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info simulates face routing in the virtual plane graph of connected quasi unit disk graphs with* $\varepsilon \geq \frac{1}{\sqrt{2}}$.

Proof. As in the proof of Theorem 5.5, it suffices to prove that the sequence consisting of the beginning node of each successive value of *packet.next_edge* is $F(\pi)$, where π denotes

Algorithm Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info

▷ *Performed by node u, the node that currently holds the packet.*

▷ *The information stored in the packet header is:*

- *packet.destination, the destination node of the packet;*
- *packet.next_edge, the network edge containing the next virtual edge;*
- *packet.last_point, the ending point of the previous virtual edge or the starting point on the current virtual face if packet.next_edge contains the first virtual edge on that face;*
- *packet.transit_path: a path from the current node to the beginning node of packet.next_edge; and*
- *packet.distance, the distance from the starting point of the current virtual face to the destination node.*

```
1  begin
2      if u is packet.destination then
3          release the packet; return
4      if packet.destination is a neighbor of u then
5          forward the packet to packet.destination; return
6      if u is not the beginning node of packet.next_edge then
           ▷ the last node in packet.transit_path is the beginning node of packet.next_edge
7          update packet.transit_path by removing its first node
8          forward the packet to the first node in packet.transit_path; return
9      let v denote the ending node of packet.next_edge      ▷ u is the beginning node of packet.next_edge
10     EPND((u, v), packet.last_point, β, (w, x), π)
11     d ← shortest distance from a point on edge (packet.last_point, β) to packet.destination
12     if d < packet.distance then          ▷ switch faces
13         p ← the closest point to packet.destination on edge (packet.last_point, β)
14         INIT-QUDG3HOP(p)
15         if u is the beginning node of packet.next_edge then
16             Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info
17             return
18     else
19         packet.next_edge ← (w, x)
20         packet.last_point ← β
21         packet.transit_path ← π      ▷ the last node in π is the beginning node of packet.next_edge
       ▷ route the packet to the beginning node of packet.next_edge
22     update packet.transit_path by removing its first node
23     forward the packet to the first node in packet.transit_path
24     return
25 end
```

Figure 6.3: Algorithm Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info

the virtual path obtained by applying face routing in the virtual plane graph and $F(\pi)$ is defined in Chapter 4.

At the source node or when a new starting point is found, INIT-QUDG3HOP is called to compute the first virtual edge. From Corollary 6.3 and the definition of face routing, when INIT-QUDG3HOP returns, *packet.next_edge* is the edge that contains the first virtual edge on the boundary of the virtual face to be traversed. At the beginning node of each successive value of *packet.next_edge*, EPND is called to compute the next edge. From Proposition 4.3, the output (w, x) from EPND is the edge containing the next virtual edge along the boundary of the current virtual face. If no new starting point is found, (w, x) is assigned to *packet.next_edge*. Otherwise, the traversal switches to the next virtual face. Therefore, during the traversal, *packet.next_edge* is always the edge that contains each virtual edge in the virtual path π. Hence, the sequence of beginning nodes of *packet.next_edge* is $F(\pi)$. Thus Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info simulates face routing in the virtual plane graph of connected quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$. □

6.3 Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info

Analogous to the second protocol for unit disk graphs, Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info is another routing protocol for quasi unit disk graphs that applies the virtual face routing approach. It assumes that nodes have only two-hop neighbor information. From our analysis of the geometric properties of quasi unit disk graphs, with only two-hop neighbor information, a node cannot find all the edges crossing its incident edges. Thus, the beginning node of *packet.next_edge* cannot perform EPND locally as in the Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info protocol.

However, we will show that to compute the ending point of the current virtual edge

and the edge containing the next virtual edge, the current node does not need to know all the crossing edges. Let edge (u, v) denote *packet.next_edge*, and let α denote the beginning point of the current virtual edge. If the closest point to α at which an edge crosses (α, v) is between α and the midpoint of (u, v), then, from Lemma 6.2, node u knows all the edges crossing (α, v) at that point. In this case, node u can complete the computation with its local information.

On the other hand, if node u does not see any edge crossing (α, v) at a point between α and the midpoint m of (u, v), no such an edge exists. The closest point to α at which an edge crosses (α, v) must be between m and v. By Lemma 6.2, all edges crossing (α, v) between m and v are known to node v. Therefore, in this case, node v can complete the computation with its local information. Moreover, if node v does not see any edge crossing (α, v), no such an edge exists. In this case, the ending point of the current virtual edge is v and the edge containing the next virtual edge is the next edge after (v, u) in clockwise order around v. Thus, in both cases, node v can complete the computation with its local information.

Therefore, in Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info, the computation of finding the network edge that contains the next virtual edge is carried out either entirely by the beginning node of *packet.next_edge* or by both the beginning node and the ending node of *packet.next_edge*, and both nodes perform the computation with only their local information.

Now we describe this protocol in more detail. At the beginning of the routing process, the source node s creates a packet containing the packet destination and calls INIT-QUDG2HOP(s) to initialize the rest of the packet header. The INIT-QUDG2HOP algorithm computes the edge containing the first virtual edge along the boundary of the virtual face to be traversed. This algorithm is also executed at an intermediate node when a new starting point is found. The input of INIT-QUDG2HOP is the starting point of the virtual face to be traversed. The code for the INIT-QUDG2HOP algorithm

is shown in Figure 6.4.

Algorithm INIT-QUDG2HOP(p)

▷ *Input: p, the starting point of the virtual face to be traversed*

▷ *Executed by a node that currently holds the packet and that has an incident edge containing p.*

▷ *It computes the network edge that contains the first virtual edge along the boundary of the virtual face and sets up packet header.*

▷ *It requires that the node has two-hop neighbor information.*

1 **begin**

2 $(v_1, v_2) \leftarrow$ the network edge containing the first edge in clockwise order around p starting from the line segment between p and *packet.destination*.

3 *packet.next_edge* $\leftarrow (v_1, v_2)$

4 *packet.last_point* $\leftarrow p$

5 *packet.distance* $\leftarrow$ the distance from p to *packet.destination*

6 **end**

Figure 6.4: Algorithm INIT-QUDG2HOP

The computation of the network edge that contains the next virtual edge along the boundary of the current virtual face begins when the node currently holding the packet is the beginning node of *packet.next_edge*. Let (u, v) be the edge *packet.next_edge*, and let α be the point *packet.last_point*. Node u checks whether it sees any edge that crosses (α, v) at a point between α and the midpoint of (u, v). If so, it assigns to β the closest point to α at which an edge crosses (α, v) and assigns to (w, x) the network edge that contains the next virtual edge after (β, α) in clockwise order around β. Otherwise, it just forwards the packet to node v.

If node v receives the packet, it checks whether it sees any edge crossing (α, v). If so, it assigns to β the closest point to α at which an edge crosses (α, v) and assigns to (w, x) the network edge that contains the next virtual edge after (β, α) in clockwise order around β. Otherwise, it assigns v to β and assigns to (w, x) the next edge after (v, u) in clockwise order around v.

After β is determined, either at node u or node v, the current node determines whether the traversal should switch to the next virtual face or not. The current node computes the shortest distance from points on (α, β) to the destination. If it is less than *packet.distance*, the distance from the starting point of the current virtual face to the destination, face switching occurs. The new starting point p is the closest point to the destination on (α, β). Note that $p \neq \alpha$, because α is either the current starting point, in which case its distance to the destination is equal to *packet.distance*, or the ending node of the previous virtual edge, in which case if α is closer to the destination, face switching would have occurred when the packet traversed the previous virtual edge. The current node calls INIT-QUDG2HOP(p) to set up packet header. If the distance from (α, β) to the destination is not less than *packet.distance*, the packet will continue to travel along the boundary of the current virtual face. In this case, the current node assigns β and (w, x) to *packet.last_point* and *packet.next_edge*, respectively.

From the above description and the definition of face routing, we get the following result.

Proposition 6.6. *When no face switching occurs, the new value of packet.last_point is the ending node of the current virtual edge and the new value of packet.next_edge is the edge that contains the next virtual edge along the boundary of the same virtual face.*

Finally, after the new value of *packet.next_edge* has been determined and this field is updated, the current node forwards the packet to the beginning node of (the updated) *packet.next_edge*. It is possible that the beginning node of *packet.next_edge* is the current node. In this case, no forwarding is needed: the current node just executes Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info again. If the beginning node of *packet.next_edge* is not the current node, it is at most two hops away, because at least one of the endpoints of *packet.next_edge* is a neighbor of the current node. The packet is forwarded to the beginning node of *packet.next_edge* either directly or via the ending node of *packet.next_edge*.

Notice that, as described above, in Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info, whenever a packet is sent to node z, z is either the beginning node or the ending node of *packet.next_edge*. If z is the beginning node of *packet.next_edge*, it performs the computation of finding the next edge. There are two circumstances when z is the ending node of *packet.next_edge*. First, the packet is sent from the beginning node of *packet.next_edge*. In this case, z performs the computation of finding the next edge as previously described. Second, the packet is sent from a node that is not an endpoint of *packet.next_edge*. This happens when that node has completed the computation of finding the next edge and has updated *packet.next_edge*. In this case, z just relays the packet to the beginning node of *packet.next_edge*. Therefore, when the ending node of *packet.next_edge* receives a packet, it must determine which action to perform. This can be determined by checking whether the packet is sent from the beginning node of *packet.next_edge* or not.

The complete code for Algorithm Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info is shown in Figure 6.5, which is the main algorithm of the protocol.

In the Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info protocol, whenever INIT-QUDG2HOP is called, its computation is performed based on the local information at the node that calls it. Therefore, it is critical whether that node has sufficient information for the computation, specifically, whether that node knows all the edges that contain the starting point p. We say that a node knows certain information if that information can be inferred from the node's local information.

In the following, we first show that the nodes that call INIT-QUDG2HOP do have sufficient information for the computation. Thus, when INIT-QUDG2HOP returns, the packet header fields are set correctly for the traversal on the current virtual face. Namely, *packet.next_edge* is the edge that contains the first virtual edge on the boundary of the virtual face to be traversed, and *packet.last_point* is the starting point on that virtual face. Then, we will prove that Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info

Algorithm Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info

▷ *Performed by node z, the node that currently holds the packet.*
▷ *The information stored in the packet header is as follows:*

- *packet.destination, the destination node of the packet;*
- *packet.next_edge, the network edge containing the next virtual edge;*
- *packet.last_point, the ending point of the previous virtual edge or the starting point of the current virtual face if packet.next_edge contains the first virtual edge on that face; and*
- *packet.distance, the distance from the starting point of the current virtual face to the destination node.*

```
1  begin
2      if z is packet.destination then
3          release the packet; return
4      if packet.destination is a neighbor of z then
5          forward the packet to packet.destination; return
6      let (u, v) denote packet.next_edge and let α denote packet.last_point
7      if z is the beginning node of packet.next_edge then
8          let m denote the midpoint of edge (u, v)
9          if α is between u and m and there exist edges crossing (α, v) at a point between α and m
           then
10             β ← the closest point to α at which an edge crosses (α, v)
11             (w, x) ← the edge crossing (α, v) at β that contains the next virtual edge after (β, α)
               in clockwise order around β        ▷ note that ∠αβx < 180°
12         else
13             forward the packet to the ending node of packet.next_edge; return
14     else          ▷ z is the ending node of packet.next_edge
15         if the packet was not sent from the beginning node of packet.next_edge then
16             forward the packet to the beginning node of packet.next_edge; return
17         if there exist edges crossing (α, v) then
18             β ← the closest point to α at which an edge crosses (α, v)
19             (w, x) ← the edge crossing (α, v) at β that contains the next virtual edge after (β, α)
               in clockwise order around β        ▷ note that ∠αβx < 180°
20         else
21             β ← v
22             (w, x) ← the next edge after (v, u) in clockwise order around v
23     d ← shortest distance from a point on (α, β) to packet.destination
24     if d < packet.distance then                ▷ switch faces
25         p ← the closest point to packet.destination on (α, β)
26         INIT-QUDG2HOP(p)
27     else
28         packet.next_edge ← (w, x)
29         packet.last_point ← β
30     if z is the beginning node of packet.next_edge then
31         Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info
32         return
33     else          ▷ route the packet to the beginning node of packet.next_edge
34         let (u′, v′) be the value of packet.next_edge
35         if u′ is a neighbor of z then          ▷ at least one of u′ and v′ is a neighbor of the current node z
36             forward the packet to u′; return
37         else
38             forward the packet to v′; return
39 end
```

Figure 6.5: Algorithm Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info

simulates face routing in the virtual plane graph of connected quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$, and therefore, it guarantees message delivery.

Proposition 6.7. *In Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info, whenever INIT-QUDG2HOP is called, the node that calls it knows all the edges that contain the starting point p.*

Proof. At the source node s, INIT-QUDG2HOP is called with s as the input. Node s knows all the edges with s as one endpoint and, thus, has sufficient information to perform the computation in INIT-QUDG2HOP.

At an intermediate node, INIT-QUDG2HOP is called when a new starting point p is found. Let (u, v) be the value of *packet.next_edge* before INIT-QUDG2HOP is called, and let (α, β) be the current virtual edge contained in (u, v). The new starting point p is on edge (α, β) and $p \neq \alpha$. There are the following two cases: (i) β is between α and the midpoint of (u, v); (ii) β is between the midpoint of (u, v) and v.

In the first case, since node u will find β and complete the computation of finding the new value of *packet.next_edge*, INIT-QUDG2HOP is called at node u. Node u knows all the edges that contain point p, because p is between u and the midpoint of (u, v).

In the second case, since no edges crossing (α, v) at a point between α and the midpoint of (u, v), node u will forward the packet to node v, which determines β and p and then calls INIT-QUDG2HOP. If p is between the midpoint of (u, v) and v, then node v knows all the edges that contain p. Otherwise p is strictly between α and the midpoint of (u, v). In this case, the only edges that contain p are edge (u, v) and edge (v, u), because there are no edges that cross (u, v) strictly between α and β. Node v knows both edges.

Therefore, in all cases, the statement of this proposition is true. □

Theorem 6.8. *Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info simulates face routing in the virtual plane graph of connected quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$.*

Proof. By Theorem 6.5, it suffices to prove that the sequence consisting of the successive values of *packet.next_edge* is the same in both Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info and Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info algorithms.

At the source node s, *packet.next_edge* is initialized by INIT-QUDG3HOP(s) in Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info and by INIT-QUDG2HOP(s) in Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info. In both cases, *packet.next_edge* is set to the first edge in clockwise order around s starting from the line segment between s and *packet.destination*. Furthermore, *packet.last_point* is initialized to the source node s in both algorithms.

In Virtual-Face-Traversal-For-QUDG-With-Three-Hop-Info, *packet.next_edge* and *packet.last_point* are updated at node u. If no face switching occurs, they are updated with the edge (w, x) and the node β returned by EPND. Thus, from Proposition 4.3 and Proposition 6.6, the new values of *packet.next_edge* and *packet.last_point* are the same in both algorithms. If face switching occurs, it follows from Proposition 6.4 and Proposition 6.7 that the new values of *packet.next_edge* and *packet.last_point* are the same in both algorithms.

Therefore, the sequence consisting of the successive values of *packet.next_edge* is the same in both algorithms. □

Chapter 7

Routing in Edge Dynamic Quasi Unit Disk Graphs

In this chapter, we describe our protocol that applies the virtual face routing approach for edge dynamic quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$. Recall that our Tethered-Traversal protocol [21, 22] works for edge dynamic graphs that are always plane graphs. It uses information about the packet path to deal with the changes of edges during routing. An edge dynamic quasi unit disk graph is not, in general, a plane graph. Moreover, as in static quasi unit disk graphs, extracting a connected spanning plane subgraph in an edge dynamic quasi unit disk graph is not always possible, because a connected quasi unit disk graph may not have a connected spanning plane subgraph. So, Tethered-Traversal does not work in an arbitrary edge dynamic quasi unit disk graph.

In the following, we present a protocol, Virtual-Face-Traversal-With-Tether, for edge dynamic quasi unit disk graphs, which combines the techniques in Tethered-Traversal with the virtual face routing approach described in Chapter 4.

7.1 Virtual-Face-Traversal-With-Tether

Since an edge dynamic quasi unit disk graph is a quasi unit disk graph at each point in time, we can apply a virtual face routing protocol, such as Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info. In an edge dynamic graph, a new edge may split the virtual face currently being traversed into two. If the packet is travelling on a newly formed virtual face that does not contain a point that is closer to the destination, the packet could be trapped on that virtual face. Virtual-Face-Traversal-With-Tether extends Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info to edge dynamic quasi unit disk graphs with $\varepsilon \geq 1/\sqrt{2}$, using techniques in Tethered-Traversal. The idea is to store information about the current virtual face in a field of the packet header, called the *tether*, and then use this information to avoid such problems. If following the next edge would cause the packet to loop back, that edge is ignored by the current node. Thus, the use of the tether will keep the traversal on the edges that existed when the packet starts to traverse the current virtual face and the new edges that do not cause problems.

Specifically, the tether stores the path followed by the packet on the current virtual face. Each time the starting point of a new virtual face is determined, the tether is initialized. As the traversal proceeds, additional nodes are appended to the tether. If the network edge that contains the next virtual edge on the current virtual face intersects the tether and forms a cycle whose direction is counterclockwise, the current node ignores this edge and recomputes the next edge of the traversal. (Since the right-hand rule is used to select the next edge, clockwise cycles are not a problem, as discussed in [22].) Otherwise, the packet header is updated and the packet is forwarded to the beginning node of the next edge. Figure 7.1 shows an example of a counterclockwise cycle formed by a candidate next edge with the tether during the traversal.

Now we describe this protocol in detail. At the beginning of the routing process, the source node s creates a packet containing the packet destination and calls INIT-EDQUDG(s) to initialize the rest of the packet header. INIT-EDQUDG performs the

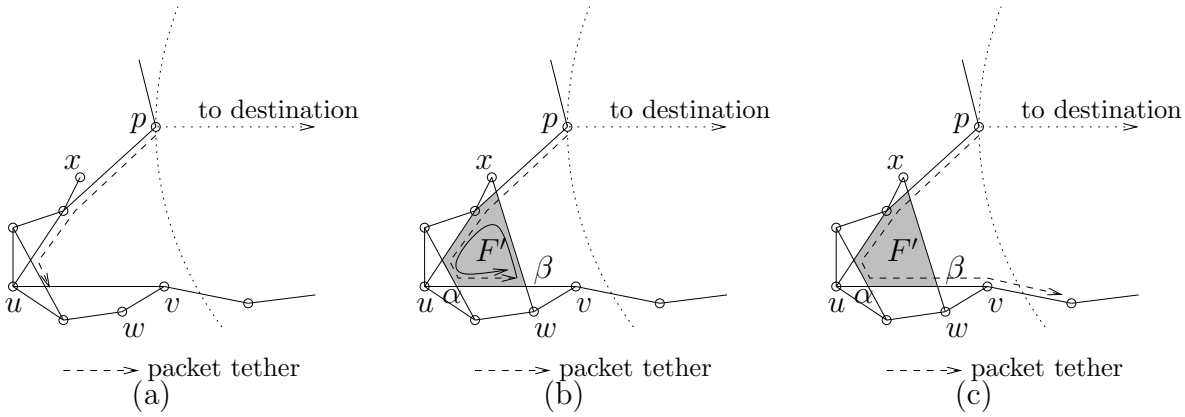


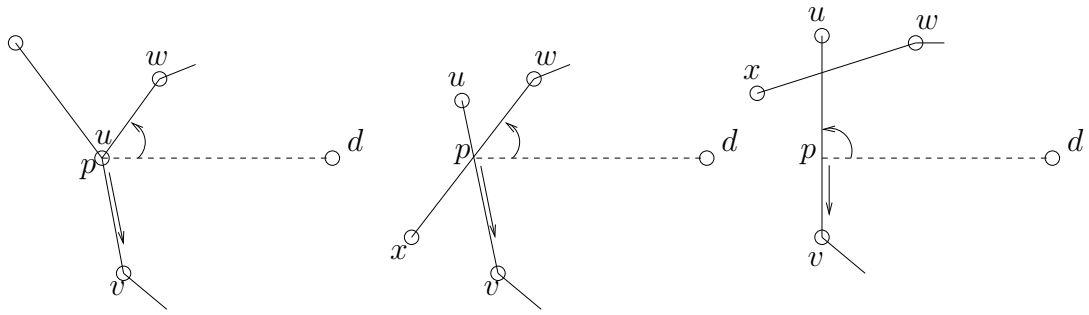
Figure 7.1: (a) Traversal of the current virtual face. (b) New edge (w, x) forms a counterclockwise cycle with the tether, and packet is on the new virtual face F'. (c) Traversal continues on the virtual face intersected by the line to destination by ignoring edge (w, x).

same initialization as INIT-QUDG2HOP, computing the edge that contains the first virtual edge along the boundary of the virtual face to be traversed and initializing *packet.next_edge*, *packet.last_point*, and *packet.distance*. In addition, INIT-EDQUDG also initializes *packet.tether*, the field that stores the tether.

The tether is a sequence of real or virtual nodes, with the possible exception of the starting point on the virtual face. The starting point on the virtual face is the input parameter of INIT-EDQUDG, denoted by p, which may be a real or virtual node, or an interior point on an edge. Conceptually, the traversal on the current virtual face starts at the starting point p. As the tether grows, a cycle can be formed at any point on the path, in particular, at the starting point p. If a cycle is formed at the starting point p, in order to determine the direction of the cycle, we need to know the edge that contains the first virtual edge in *counterclockwise* order around p starting from the line segment between p and the destination at the starting of the traversal. We could use a separate field in the packet header to store this edge, but for simplicity of the algorithm and its proof of correctness, we make this edge part of the tether and store it in *packet.tether*. The

details of how to determine the direction of a cycle using *packet.tether* will be described later in this section.

Thus, the first node of the tether is the ending node of the edge that contains the first virtual edge in *counterclockwise* order around p starting from the line segment between p and *packet.destination*. The second node of the tether is always the starting point p. This is illustrated in Figure 7.2 where d = *packet.destination*. The examples in this figure indicate the initial value of *packet.tether* when the starting point p is a real node, a virtual node, or an interior point point on an edge, respectively. The code for the INIT-EDQUDG algorithm is shown in Figure 7.3.



(a) *packet.tether* = w, p (b) *packet.tether* = w, p (c) *packet.tether* = u, p

Figure 7.2: Initialization of *packet.destination*

After INIT-EDQUDG returns, the current node executes Algorithm Virtual-Face-Traversal-With-Tether to route the packet. A large part of the computation is the same as that in the Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info algorithm. Specifically, the beginning node of *packet.next_edge*, say, edge (u, v), tries to find β, the ending node of the current virtual edge, and to find (w, x), the edge that contains the next virtual edge along the boundary of the current virtual face. This is done by checking its two-hop neighborhood for edges crossing (u, v) at a point between *packet.last_point* and the midpoint of (u, v). If it does not see any such edges, it forwards the packet to node v. Then node v can determine β and (w, x) using its two-hop neighbor information. Once β is identified, the current node checks whether there is a new starting point on

Algorithm INIT-EDQUDG(p)

▷ *Input: p, the starting point of the virtual face to be traversed*

▷ *Executed by a node that currently holds the packet and that has an incident edge containing p.*

▷ *It computes the network edge that contains the first virtual edge along the boundary of the virtual face and sets up packet header.*

▷ *It requires that the node has two-hop neighbor information.*

1 **begin**

2 *packet.next_edge* ← the network edge containing the first edge in clockwise order around p starting from (and including) the line segment between p and *packet.destination*

3 *packet.last_point* ← p

4 *packet.distance* ← the distance from p to *packet.destination*

5 **let** (v_2, v_1) be the network edge containing the first edge in counterclockwise order around p after the line segment between p and *packet.destination*

6 *packet.tether* ← v_1, p

7 **end**

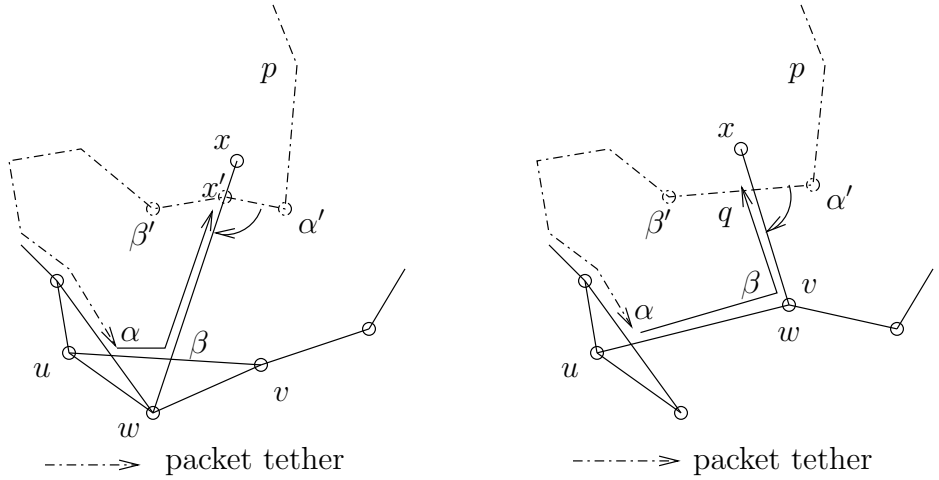
Figure 7.3: Algorithm INIT-EDQUDG

(α, β). If a new starting point is found, the current node calls INIT-EDQUDG and the traversal proceeds to the next virtual face. So far the computation is the same as that in Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info.

However, if a new starting point is not found, an additional procedure from the tethered-traversal technique is performed. We say that edge (w, x) forms a cycle with the tether if it intersects the tether excluding the first edge in the tether but including p, the starting point. As illustrated in Figure 7.4, there are two cases depending on the geometric relationship between (w, x) and the tether. In the first case, edge (w, x) intersects the tether at a real or virtual node in the tether excluding the first node of the tether. Let x' be that node in the tether, let α' be the previous node before x' in the tether, and let β' be the next node after x' in the tether. We say that this cycle is counterclockwise if edge (x', w) is between the clockwise angle from (x', α') to (x', β') around x', i.e., the angle from (x', α') to (x', w) is less than the angle from (x', α') to

(x', β') in the clockwise direction around x'.

In the second case, edge (w, x) intersects an edge in the tether excluding its first edge. Say, (w, x) intersects edge (α', β'). Let q be the intersection point. Analogously, we say that this cycle is counterclockwise if the angle from (q, α') to (q, w) in the clockwise direction around q is less than 180°.



(a) (w, x) intersects a node in the tether (b) (w, x) crosses an edge in the tether

Figure 7.4: Counterclockwise cycles formed by edge (w, x) with the tether

If edge (w, x) forms a counterclockwise cycle with the packet tether, the current node updates its two-hop neighborhood information by removing the connection between w and x. Then, it executes Virtual-Face-Traversal-With-Tether again to re-compute the next edge. If edge (w, x) does not form a counterclockwise cycle with the packet tether, the current node assigns (w, x) and β to *packet.next_edge* and *packet.last_node*, respectively. Moreover, it updates *packet.tether* by adding node β to the end of its current list. Then, as in Virtual-Face-Traversal-For-QUDG-With-Two-Hop-Info, the current node routes the packet to the beginning node of *packet.next_edge* if it is not that node. The beginning node of *packet.next_edge* continues the computation of finding the next edge for the traversal.

The code for Algorithm Virtual-Face-Traversal-With-Tether is shown in Figure 7.5 and 7.6, which is the main algorithm of the protocol.

Algorithm Virtual-Face-Traversal-With-Tether

▷ *Performed by node z, the node that currently holds the packet.*

▷ *The information stored in the packet header is as follows:*

- *packet.destination, the destination node of the packet;*
- *packet.next_edge, the network edge containing the next virtual edge;*
- *packet.last_point, the ending point of the previous virtual edge or the starting point of the current virtual face if packet.next_edge contains the first virtual edge on that face;*
- *packet.distance, the distance from the starting point of the current virtual face to the destination node; and*
- *packet.tether, a sequence of nodes that tracks the path followed by the packet along the boundary of the current virtual face.*

```
1  begin
2      if z is packet.destination then
3          release the packet; return
4      if packet.destination is a neighbor of z then
5          forward the packet to packet.destination; return
6      let (u, v) denote packet.next_edge and let α denote packet.last_point
7      if z is the beginning node of packet.next_edge then
8          let m denote the midpoint of edge (u, v)
9          if α is between u and m and there exist edges crossing (α, v) at a point between α and m
           then
10             β ← the closest point to α at which an edge crosses (α, v)
11             (w, x) ← the edge crossing (α, v) at β that contains the next virtual edge after (β, α)
               in clockwise order around β        ▷ note that ∠αβx < 180°
12         else
13             forward the packet to the ending node of packet.next_edge; return
14     else          ▷ z is the ending node of packet.next_edge
15         if the packet is not sent from the beginning node of packet.next_edge then
16             forward the packet to the beginning node of packet.next_edge; return
17         if there exist edges crossing (α, v) then
18             β ← the closest point to α at which an edge crosses (α, v)
19             (w, x) ← the edge crossing (α, v) at β that contains the next virtual edge after (β, α)
               in clockwise order around β        ▷ note that ∠αβx < 180°
20         else
21             β ← v
22             (w, x) ← the next edge after (v, u) in clockwise order around v
```

(Continued on the next page)

Figure 7.5: Algorithm Virtual-Face-Traversal-With-Tether

Algorithm Virtual-Face-Traversal-With-Tether (Con't)

```
23      d ← shortest distance from a point on (α, β) to packet.destination
24      if d < packet.distance then                ▷ switch faces
25          p ← the closest point to packet.destination on (α, β)
26          INIT-EDQUDG(p)
27      else
28          if (β, x) forms a counterclockwise cycle with packet.tether then
29              remove (w, x) and (x, w) from the two-hop neighborhood of node z
30              Virtual-Face-Traversal-With-Tether
31              return
32          else
33              packet.next_edge ← (w, x)
34              packet.last_point ← β
35              packet.tether ← packet.tether, β
36      if z is the beginning node of packet.next_edge then
37          Virtual-Face-Traversal-With-Tether
38          return
39      else          ▷ route the packet to the beginning node of packet.next_edge
40          let (u′, v′) be the value of packet.next_edge
            ▷ at least one of u′ and v′ is a neighbor of the current node z
41          if u′ is a neighbor of z then
42              forward the packet to u′; return
43          else
44              forward the packet to v′; return
45  end
```

Figure 7.6: Algorithm Virtual-Face-Traversal-With-Tether (Con't)

7.2 Conditions for Guaranteeing Message Delivery

Now we discuss the conditions under which Virtual-Face-Traversal-With-Tether works correctly. First, the delivery guarantee condition for Tethered-Traversal requires that a stable connected spanning subgraph exists during the entire traversal of each face [22]. This is also needed for Virtual-Face-Traversal-With-Tether. Second, for the virtual face traversal approach to work correctly, when an edge is assigned to *packet.next_edge*, it must remain connected until *packet.next_edge* is updated. We can bound this time period as follows. Once *packet.next_edge* is determined, the current node sends the packet to the beginning node of *packet.next_edge*, either directly or via the ending node of *packet.next_edge*, and during the computation of the next edge, the packet may be forwarded along *packet.next_edge*. Hence, the time between two successive updates of *packet.next_edge* is at most the time needed for three transmissions plus a small amount of the computation.

Therefore, we have the following conditions for message delivery guarantee using Virtual-Face-Traversal-With-Tether.

Condition 7.1. A stable connected spanning subgraph of the network graph exists during the entire traversal of each virtual face.

Condition 7.2. If an edge is assigned to *packet.next_edge*, it remains connected for at least the time needed for three transmissions plus a small amount of computation.

7.3 Proof of Correctness of Virtual-Face-Traversal-With-Tether

Now we will prove that Virtual-Face-Traversal-With-Tether guarantees message delivery in edge dynamic quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$ under Condition 7.1 and 7.2. We begin by introducing some definitions. Suppose that, at time t, a packet starts traversing

a virtual face F in the edge dynamic quasi unit disk graph. By Condition 7.1, a stable connected spanning subgraph G' exists during the entire traversal of F. F is contained within a virtual face F' of G'. The main idea of the proof is to show that the packet travels roughly along the boundary of F'.

Let $W_{F'}$ denote the traversal of the boundary of F' using the right-hand rule. This consists of a sequence of directed edges. We observe the following geometric properties regarding the edges in $W_{F'}$.

Proposition 7.1. *Let (α, β) and (β, γ) be two consecutive edges in $W_{F'}$, then, any edge that intersects both β and the interior of the clockwise angle around β from (β, α) to (β, γ) intersects the interior of F'.*

Proposition 7.2. *If edge (α, β) intersects an edge in $W_{F'}$ at β, the interior of (α, β) is in the interior of F', and e is the first edge in $W_{F'}$ in clockwise order around β starting from (β, α) then, any edge that intersects both β and the interior of the clockwise angle around β from (β, α) to e intersects the interior of F'.*

Furthermore, notice that since F' is a virtual face of a connected spanning subgraph of the network graph, there cannot exist any real nodes inside F'. Thus, we have the following observations.

Proposition 7.3. *If (α, v) intersects the interior of F' and v is a real node, then (α, v) must intersect $W_{F'}$ at a point that is not α.*

Proposition 7.4. *For any virtual or real edge (α, β) whose interior is in the interior of F', the network edge (u, v) that contains (α, β) must intersect $W_{F'}$ at least twice, and at least one intersection point is between u and α, and at least one intersection point is between v and β.*

Let $(\beta^{(k-1)}, \beta^{(k)})$ denote the kth edge of *packet.tether*, and let *packet.next_edge*$^{(k)}$ denote the kth value of *packet.next_edge*, for $k \geq 1$, if they exist. Note that $(\beta^{(k)}, \beta^{(k+1)})$

is contained in the real edge $packet.next\_edge^{(k)}$ and intersects $packet.next\_edge^{(k+1)}$ at $\beta^{(k+1)}$. From observation of the algorithm, $(\beta^{(0)}, \beta^{(1)})$ and $packet.next\_edge^{(1)}$ are set in INIT-EDQUDG. The algorithm computes $\beta^{(k+1)}$ and $packet.next\_edge^{(k+1)}$ together.

We will prove the following properties of *packet.tether* and *packet.next_edge*.

Lemma 7.5. *Suppose that a packet starts traversing a virtual face F with starting point p and let F' be the stable virtual face containing F. Let $k \geq 1$. Before the traversal switches to another virtual face, if $(\beta^{(k-1)}, \beta^{(k)})$ and $packet.next\_edge^{(k)}$ exist, the following properties hold:*

(a) *Any edge in the stable subgraph G' that intersects $\beta^{(k)}$ does not intersect the interior of the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $packet.next\_edge^{(k)}$;*

(b) *If $k \geq 2$, then either $(\beta^{(k-1)}, \beta^{(k)})$ is contained in an edge in $W_{F'}$, or the interior of $(\beta^{(k-1)}, \beta^{(k)})$ is in the interior of F'.*

Proof. We will prove the lemma by induction on k.

Base case From line 2 of INIT-EDQUDG, $packet.next\_edge^{(1)}$ is the network edge that contains the first edge in clockwise order around the starting point p starting from (and including) the line segment between p and *packet.destination*, as shown in Figure 7.7. From line 5-6 of INIT-EDQUDG, $\beta^{(1)} = p$ and $(\beta^{(0)}, \beta^{(1)})$ contains the first edge in counterclockwise order around p after the line segment between p and *packet.destination*. Therefore, if an edge intersects p and is a stable edge in G', it does not intersect the interior of the clockwise angle around p from $(\beta^{(0)}, \beta^{(1)})$ to $packet.next\_edge^{(1)}$. Hence, (a) is true for $k = 1$.

Let (u, v) denote $packet.next\_edge^{(1)}$, and let β be the point closest to p at which some edge intersects (p, v). Thus, (p, β) is on the boundary of F. Since F is contained within F', (p, β) is within F'. Furthermore, since β is the closest intersection point to p, no nodes on the boundary of F' are on the interior of (p, β). Thus, either (p, β) is

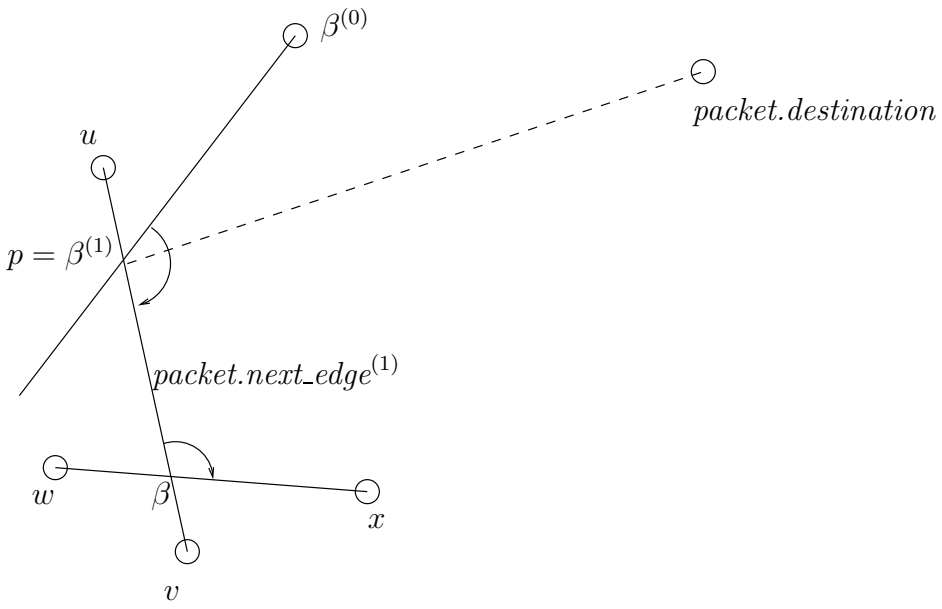


Figure 7.7: Base case of the proof of Lemma 7.5

contained in an edge in $W_{F'}$ or the interior of (p, β) is in the interior of F'. Let (w, x) be the network edge that contains the first edge in clockwise order around β starting from (β, p). Thus, any stable edge in G' that intersects β does not intersect the interior of the clockwise angle around β from (β, p) to (w, x).

Consider the computation to determine the ending node $\beta^{(2)}$ of the second edge of the tether and *packet.next_edge*$^{(2)}$. From the algorithm, if there is some point on (p, β) that is closer to *packet.destination* than p is, the traversal will be switched to another virtual face. Otherwise, (w, x) will be a candidate for *packet.next_edge*$^{(2)}$, and it is *packet.next_edge*$^{(2)}$ if and only if it does not form a counterclockwise cycle with the tether (excluding its first edge). From the definition of counterclockwise cycles, if a candidate edge forms a counterclockwise cycle with the tether, either it intersects a node in the tether excluding its first and last nodes, or it intersects an edge in the tether excluding its first edge. Since the tether currently contains only one edge, the tether excluding its first edge is empty. Thus, $\beta^{(2)} = \beta$ and *packet.next_edge*$^{(2)} = (w, x)$. Also notice that $\beta^{(1)} = p$, so $(\beta^{(1)}, \beta^{(2)}) = (p, \beta)$. Therefore, properties 1 and 2 are true for $k = 2$.

Induction step Assume that the lemma holds for all $1 \leq i \leq k$, where $k \geq 2$. By the

induction hypothesis, either $(\beta^{(k-1)}, \beta^{(k)})$ is contained in an edge in $W_{F'}$, or the interior of $(\beta^{(k-1)}, \beta^{(k)})$ is in the interior of F'. So $\beta^{(k)}$ is either on the boundary of F' or is inside F'. Let (u, v) denote $packet.next\_edge^{(k)}$, which intersects $(\beta^{(k-1)}, \beta^{(k)})$ at $\beta^{(k)}$.

Suppose $\beta^{(k)}$ is on the boundary of F' and the rest of $(\beta^{(k)}, v)$ is outside F'. We consider two cases: (a) $(\beta^{(k-1)}, \beta^{(k)})$ is contained in an edge in $W_{F'}$, and (b) the interior of $(\beta^{(k-1)}, \beta^{(k)})$ is in the interior of F'. These are illustrated in Figure 7.8.

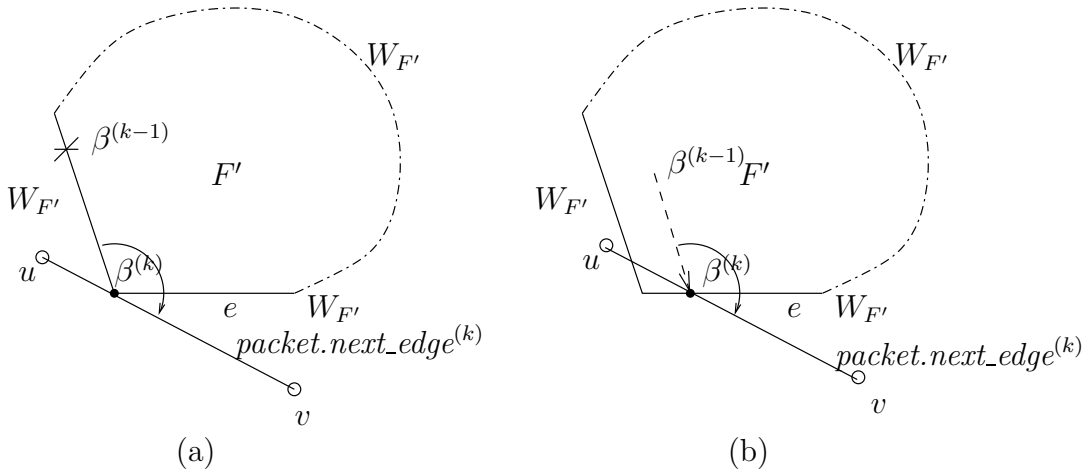


Figure 7.8: Edge $(\beta^{(k)}, v)$ is outside F'

Let e be the first edge in $W_{F'}$ in clockwise order around $\beta^{(k)}$ starting from $(\beta^{(k)}, \beta^{(k-1)})$. From Proposition 7.1 and 7.2, any edge that intersects $\beta^{(k)}$ and the interior of the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to e intersects the interior of F'. Therefore, $(\beta^{(k)}, v)$ must be outside the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to e. Then, edge e intersects $\beta^{(k)}$ and also intersects the interior of the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $(\beta^{(k)}, v)$. Since every edge in $W_{F'}$ is contained in a stable edge in G', there is a stable edge in G' (containing e) that intersects $\beta^{(k)}$ and intersects the interior of the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $(\beta^{(k)}, v)$, which is contained in $packet.next\_edge^{(k)}$. This contradicts the induction hypothesis **(a)**. Therefore, $(\beta^{(k)}, v)$ cannot be outside F'. Thus, either (i) $(\beta^{(k)}, v)$ intersects the interior of F', or (ii) $(\beta^{(k)}, v)$ overlaps an edge in $W_{F'}$.

Figure 7.9 gives some examples of the geometric relationship between $(\beta^{(k-1)}, \beta^{(k)})$ and *packet.next_edge*$^{(k)}$. In examples (a) and (b), $(\beta^{(k-1)}, \beta^{(k)})$ is contained in an edge in $W_{F'}$, and in (c) and (d), the interior of $(\beta^{(k-1)}, \beta^{(k)})$ is in the interior of F'. Examples (a) and (c) correspond to case (i) where $(\beta^{(k)}, v)$ intersects the interior of F', and examples (b) and (d) correspond to case (ii) where $(\beta^{(k)}, v)$ overlaps an edge in $W_{F'}$.

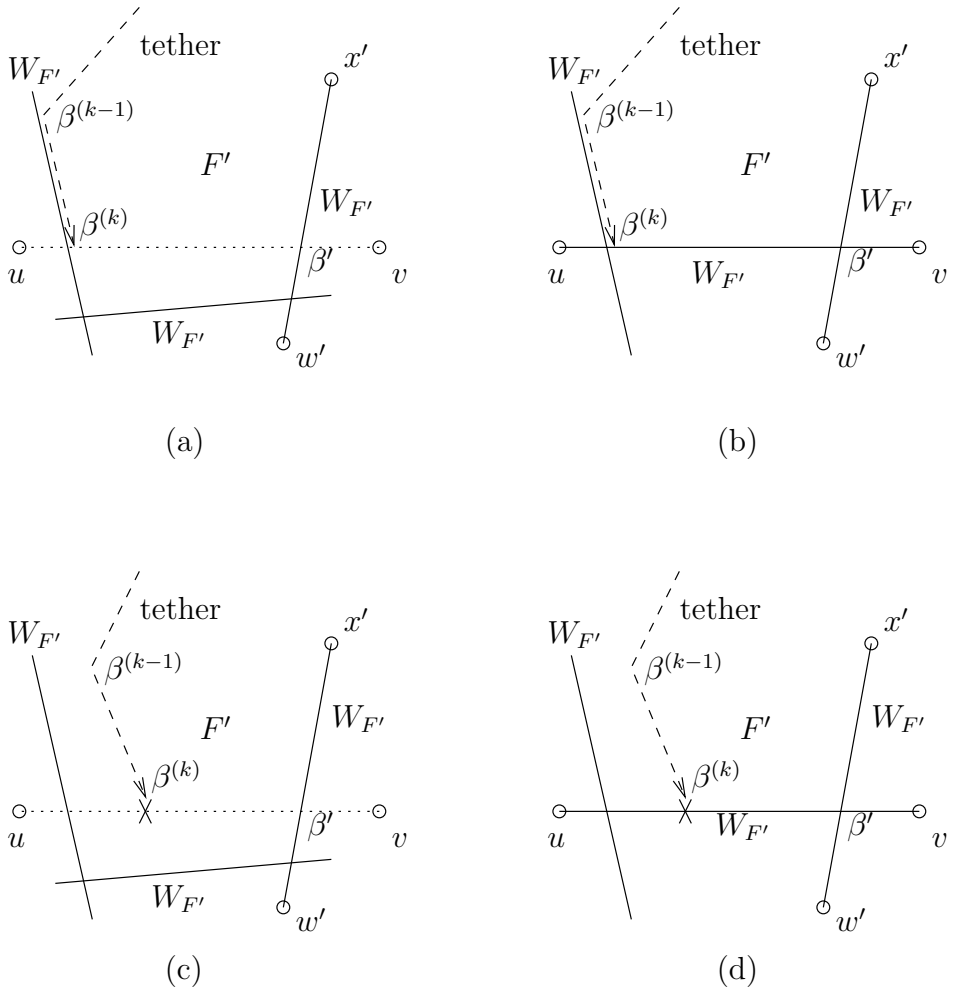


Figure 7.9: Induction step of the proof of Lemma 7.5

If $(\beta^{(k)}, v)$ belongs to case (i), by Proposition 7.3, $(\beta^{(k)}, v)$ must intersect $W_{F'}$ at a point that is not $\beta^{(k)}$. Let β' be the intersection point of $(\beta^{(k)}, v)$ and $W_{F'}$ that is closest to $\beta^{(k)}$. If $(\beta^{(k)}, v)$ belongs to case (ii), let β' be the next node in $W_{F'}$ after $\beta^{(k)}$. Hence, either the interior of $(\beta^{(k)}, \beta')$ is in the interior of F', or $(\beta^{(k)}, \beta')$ is contained in an edge

in $W_{F'}$. In both cases, let (w', x') be the network edge containing the first edge in $W_{F'}$ in clockwise order around β' starting from $(\beta', \beta^{(k)})$.

We claim that (w', x') does not form a counterclockwise cycle with the tether. To obtain a contradiction, suppose (w', x') forms a counterclockwise cycle with the tether. Then, (w', x') either intersects a node $\beta^{(i)}$ in the tether, where $1 \leq i < k$, or intersects the interior of an edge $(\beta^{(i-1)}, \beta^{(i)})$ in the tether, where $1 < i \leq k$. In the former case, by the definition of counterclockwise cycles, (w', x') also intersects the interior of the clockwise angle around $\beta^{(i)}$ from $(\beta^{(i)}, \beta^{(i-1)})$ to $(\beta^{(i)}, \beta^{(i+1)})$. However, this contradicts the induction hypothesis (a) since (w', x') is a stable edge in G'. In the latter case, from the induction hypothesis (b), either edge $(\beta^{(i-1)}, \beta^{(i)})$ is contained in an edge in $W_{F'}$, or the interior of $(\beta^{(i-1)}, \beta^{(i)})$ is in the interior of F'. Thus, the stable edge (w', x') either intersects the interior of an edge in $W_{F'}$ or intersects the interior of F'. This contradicts the definition of virtual faces. Therefore, (w', x') does not form a counterclockwise cycle with the tether.

Now, consider the computation to determine $\beta^{(k+1)}$ and $\mathit{packet.next\_edge}^{(k+1)}$. Notice that β' is the closest point to $\beta^{(k)}$ at which a stable edge intersects $(\beta^{(k)}, v)$, and (w', x') contains the first stable edge in clockwise order around β' starting from $(\beta', \beta^{(k)})$. So, either $\mathit{packet.next\_edge}^{(k+1)} = (w', x')$, or it is a nonstable edge that intersects the interior of $(\beta^{(k)}, \beta')$, or it is a nonstable edge that intersects β' and is between $(\beta', \beta^{(k)})$ and (w', x') in the clockwise direction around β'.

In all cases, $(\beta^{(k)}, \beta^{(k+1)})$ is contained in $(\beta^{(k)}, \beta')$. Therefore, either the interior of $(\beta^{(k)}, \beta^{(k+1)})$ is in the interior of F', or $(\beta^{(k)}, \beta^{(k+1)})$ is contained in an edge in $W_{F'}$. Furthermore, any stable edge that intersects $\beta^{(k+1)}$ does not intersects the interior of the clockwise angle around $\beta^{(k+1)}$ from $(\beta^{(k)}, \beta^{(k+1)})$ to $\mathit{packet.next\_edge}^{(k+1)}$. Therefore, the lemma holds for $k + 1$. □

Lemma 7.5 shows that before the traversal switches to another virtual face, the packet tether excluding its first edge is within F' and never crosses the boundary of F'. Next,

we will prove that the traversal eventually has to switch to another virtual face until the destination is reached. The proof is established through a series of propositions and lemmas.

Proposition 7.6. *For $k \geq 2$, if the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $(\beta^{(k)}, \beta^{(k+1)})$ is at least 180°, then $\beta^{(k)}$ is a real node.*

Proof. Proof by contradiction. Recall that every node in the packet tether is either a real or virtual node, with the only possible exception of $\beta^{(1)}$ that is the starting point p, which may be a real node, a virtual node, or an interior point on an edge, as shown in Figure 7.2.

Suppose $\beta^{(k)}$ is a virtual node, and the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $(\beta^{(k)}, \beta^{(k+1)})$ is at least 180°. Let (u, v) and (w, x) be the network edges that contain $(\beta^{(k-1)}, \beta^{(k)})$ and $(\beta^{(k)}, \beta^{(k+1)})$, respectively. Then, (u, v) and (w, x) intersect at $\beta^{(k)}$, which is an interior point on both edges. This is illustrated in Figure 7.10.

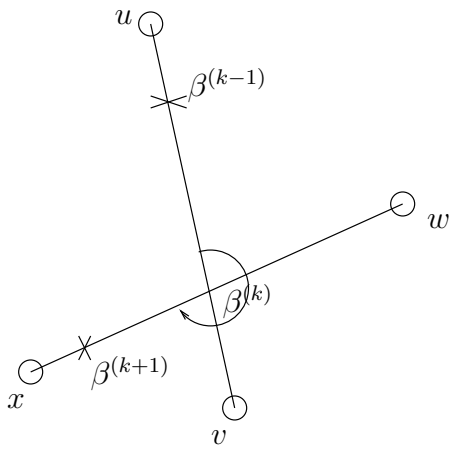


Figure 7.10: Proof of Proposition 7.6

Consider the computation of *packet.next_edge*$^{(k)}$. Starting from $(\beta^{(k)}, \beta^{(k-1)})$ and proceeding clockwise around $\beta^{(k)}$, we hit $(\beta^{(k)}, w)$ before $(\beta^{(k)}, x)$. If edge $(\beta^{(k)}, w)$ does not form a counterclockwise cycle with the packet tether, (x, w) will be *packet.next_edge*$^{(k)}$ and $(\beta^{(k)}, \beta^{(k+1)})$ should be contained in $(\beta^{(k)}, w)$. Then, the clockwise angle around

$\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $(\beta^{(k)}, \beta^{(k+1)})$ is less than 180°. This contradicts our supposition. If edge $(\beta^{(k)}, w)$ does form a counterclockwise cycle with the packet tether, from lines 28-29 of Virtual-Face-Traversal-With-Tether, edges (x, w) and (w, x) are ignored in the subsequent computation of $packet.next\_edge^{(k)}$, which contains $(\beta^{(k)}, \beta^{(k+1)})$. Thus, $(\beta^{(k)}, \beta^{(k+1)})$ cannot be contained in (w, x). This also contradicts our supposition.

Therefore, we have shown that if $\beta^{(k)}$ is not a real node, the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $(\beta^{(k)}, \beta^{(k+1)})$ is less than 180°. Hence the proposition is true. □

Next, we define a point $\beta'^{(k)}$ on the boundary of F', and we will show that the first k edges of the tether followed by $\beta'^{(k)}$ followed by part of the boundary of F' followed by part of the line between p and the destination form a region inside F'. More precisely, for $k \geq 2$, if $\beta^{(k)}$ exists, then from Lemma 7.5 (b), either $(\beta^{(k-1)}, \beta^{(k)})$ is contained in an edge in $W_{F'}$, or the interior of $(\beta^{(k-1)}, \beta^{(k)})$ is in the interior of F'. In the former case, let $\beta'^{(k)} = \beta^{(k)}$. In the latter case, let (u, v) denote the underlying network edge containing $(\beta^{(k-1)}, \beta^{(k)})$. From Proposition 7.4, there is at least one intersection point of (u, v) and $W_{F'}$ between $\beta^{(k)}$ and v. Let $\beta'^{(k)}$ be the intersection point between $\beta^{(k)}$ and v that is closest to $\beta^{(k)}$ (including $\beta^{(k)}$).

From the definition of $\beta'^{(k)}$, we can get the following property.

Proposition 7.7. *For $k \geq 2$, if $\beta'^{(k)} \neq \beta^{(k)}$, then $\beta^{(k)}$ is a virtual node.*

Proof. Suppose $\beta^{(k)}$ is a real node. From Lemma 7.5 (b), any node in the tether except the first one is either on the boundary of F' or inside F'. Thus, $\beta^{(k)}$ is on the boundary of F' since no real node is inside F'. Furthermore, either $(\beta^{(k-1)}, \beta^{(k)})$ is contained in an edge in $W_{F'}$, or the interior of $(\beta^{(k-1)}, \beta^{(k)})$ is in the interior of F'. If $(\beta^{(k-1)}, \beta^{(k)})$ is contained in an edge in $W_{F'}$, then $\beta'^{(k)} = \beta^{(k)}$ by the definition of $\beta'^{(k)}$. If the interior of $(\beta^{(k-1)}, \beta^{(k)})$ is in the interior of F', then $\beta^{(k)} = v$ where (u, v) denote the underlying network edge containing $(\beta^{(k-1)}, \beta^{(k)})$. Thus, from the definition of $\beta'^{(k)}$, $\beta'^{(k)} = \beta^{(k)} = v$. Therefore, the proposition holds. □

Recall that F' is intersected by the line segment from the starting point p to *packet.destination*, and F' is a virtual face in the stable connected spanning subgraph G'. Therefore, there is at least one point other than p at which the line segment from p to *packet.destination* intersects $W_{F'}$. Let p' be such an intersection point that is closest to p. Notice that in our notation, letters with a prime, e.g. $\beta'^{(k)}$ and p', denote points on the boundary of F'.

Proposition 7.8. *If the tether has length at least* 2*, then the line segment from* p *to* p' *is not on the boundary of* F'*.*

Proof. Suppose the tether has length at least 2 and the line segment from p to p' is on the boundary of F'. Let (u, v) denote the underlying network edge on the boundary of F', where p' is between p and node v. Then, all points on (p, v) excluding p are closer to the destination than p is.

From line 2 of INIT-EDQUDG, $packet.next\_edge^{(1)}$ is the network edge that contains the first edge in clockwise order around the starting point $p = \beta^{(1)}$ starting from (and including) the line segment between p and *packet.destination*. Thus, $packet.next\_edge^{(1)} = (u, v)$, and $\beta^{(2)}$ is a virtual or real node on (p, v) after p. So $\beta^{(2)}$ is closer to the destination than p is. This is a contradiction, because if $\beta^{(2)}$ exists, it cannot be closer to the destination than p is. Hence, the proposition holds. □

Proposition 7.9. *If the tether has length at least* 2*, then* $(\beta^{(2)}, \beta'^{(2)})$ *does not intersect the interior of the line segment between* p *and* p'*.*

Proof. Suppose the tether has length at least 2, i.e., $\beta^{(2)}$ exists, and $(\beta^{(2)}, \beta'^{(2)})$ intersects the interior of the line segment between p and p'. By the definition of $\beta'^{(k)}$, $\beta^{(1)}$, $\beta^{(2)}$, and $\beta'^{(2)}$ are collinear, and $\beta^{(2)}$ is between $\beta^{(1)} = p$ and $\beta'^{(2)}$. Thus, $\beta^{(2)}$ is on the line segment between p and p'. Since $\beta^{(2)} \neq p$, $\beta^{(2)}$ is closer to the destination than p is. This is a contradiction, because if $\beta^{(2)}$ exists, it cannot be closer to the destination than p is. Hence, the proposition holds. □

Proposition 7.10. *If the tether has length at least 2, then the part of the tether from p to $\beta^{(2)}$, $(\beta^{(2)}, \beta'^{(2)})$, the part of $W_{F'}$ from $\beta'^{(2)}$ to p', and the line segment from p' to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F'.*

Proof. Suppose the tether has length at least 2. From Lemma 7.5 (b), either $(\beta^{(1)}, \beta^{(2)})$ is contained in an edge in $W_{F'}$ or the interior of $(\beta^{(1)}, \beta^{(2)})$ is in the interior of F'.

In the former case, as illustrated in Figure 7.11, the line segment between p and p' divides F' into two regions, because it intersects F' and is not on the boundary of F' by Proposition 7.8. The part of the tether from p to $\beta^{(2)}$, $(\beta^{(2)}, \beta'^{(2)})$, the part of $W_{F'}$ from $\beta'^{(2)}$ to p', and the line segment from p' to p is the boundary of one of these two regions, shown as the darker-shaded area in Figure 7.11.

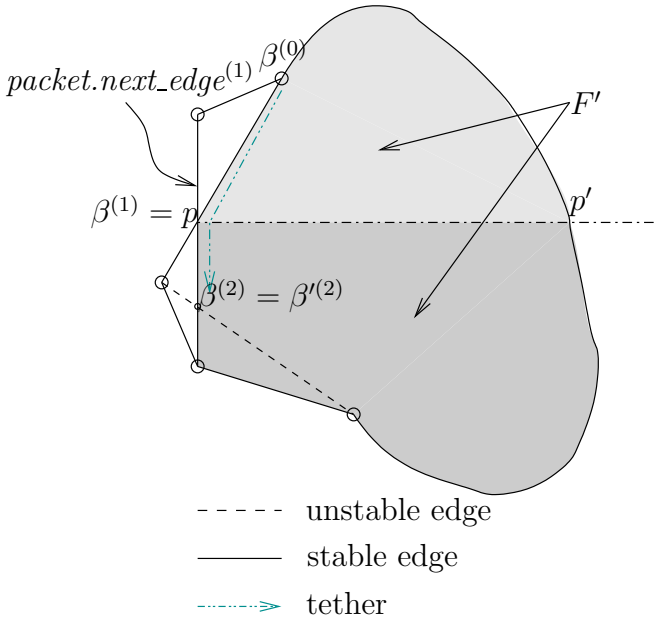


Figure 7.11: First case of the proof of Proposition 7.10

In the latter case, as illustrated in Figure 7.12, *packet.next_edge*$^{(1)}$ divides F' into two regions, and one of them is intersected by the line segment from p to p', which further divides that region into two subregions. The part of the tether from p to $\beta^{(2)}$, $(\beta^{(2)}, \beta'^{(2)})$, the part of $W_{F'}$ from $\beta'^{(2)}$ to p', and the line segment from p' to p is the boundary of one

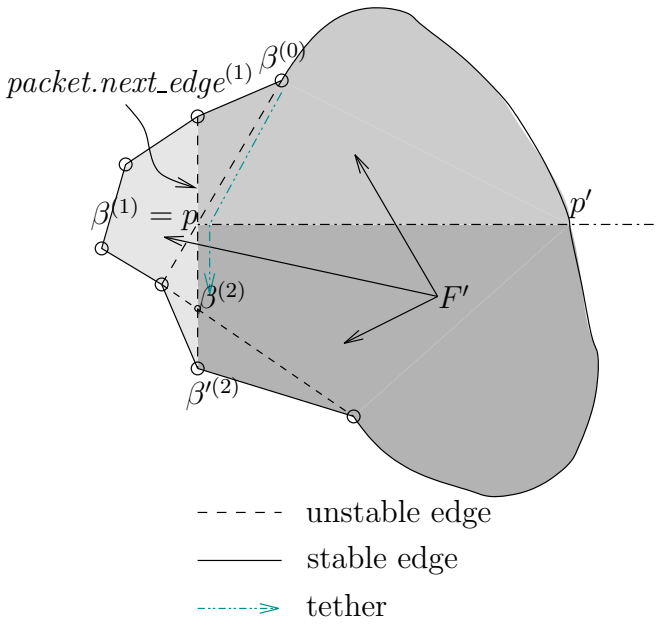


Figure 7.12: Second case of the proof of Proposition 7.10

of the resulting subregions. See the darkest area in Figure 7.12. □

Proposition 7.11. *Let $k \geq 2$. If the tether has length at least $k+1$, if $(\beta^{(i)}, \beta'^{(i)})$ does not intersect the interior of the line segment between p and p' for all $2 \leq i \leq k+1$, and if the part of the tether from p to $\beta^{(k)}$, $(\beta^{(k)}, \beta'^{(k)})$, the part of $W_{F'}$ from $\beta'^{(k)}$ to p', and the line segment from p' to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F', then $\beta'^{(k+1)} \neq \beta'^{(k)}$, $\beta'^{(k+1)}$ is on the part of $W_{F'}$ from $\beta'^{(k)}$ to p', and the part of the tether from p to $\beta^{(k+1)}$, $(\beta^{(k+1)}, \beta'^{(k+1)})$, the part of $W_{F'}$ from $\beta'^{(k+1)}$ to p', and the line segment from p' to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F'.*

Proof. Assume all the conditions in the proposition hold. Let R_k denote the region whose boundary consists of the part of the tether from p to $\beta^{(k)}$, $(\beta^{(k)}, \beta'^{(k)})$, the part of $W_{F'}$ from $\beta'^{(k)}$ to p', and the line segment from p' to p. See Figure 7.13 for an example where F' is an interior virtual face of the stable subgraph G', and see Figure 7.14 for an example where F' is the outer virtual face of G'.

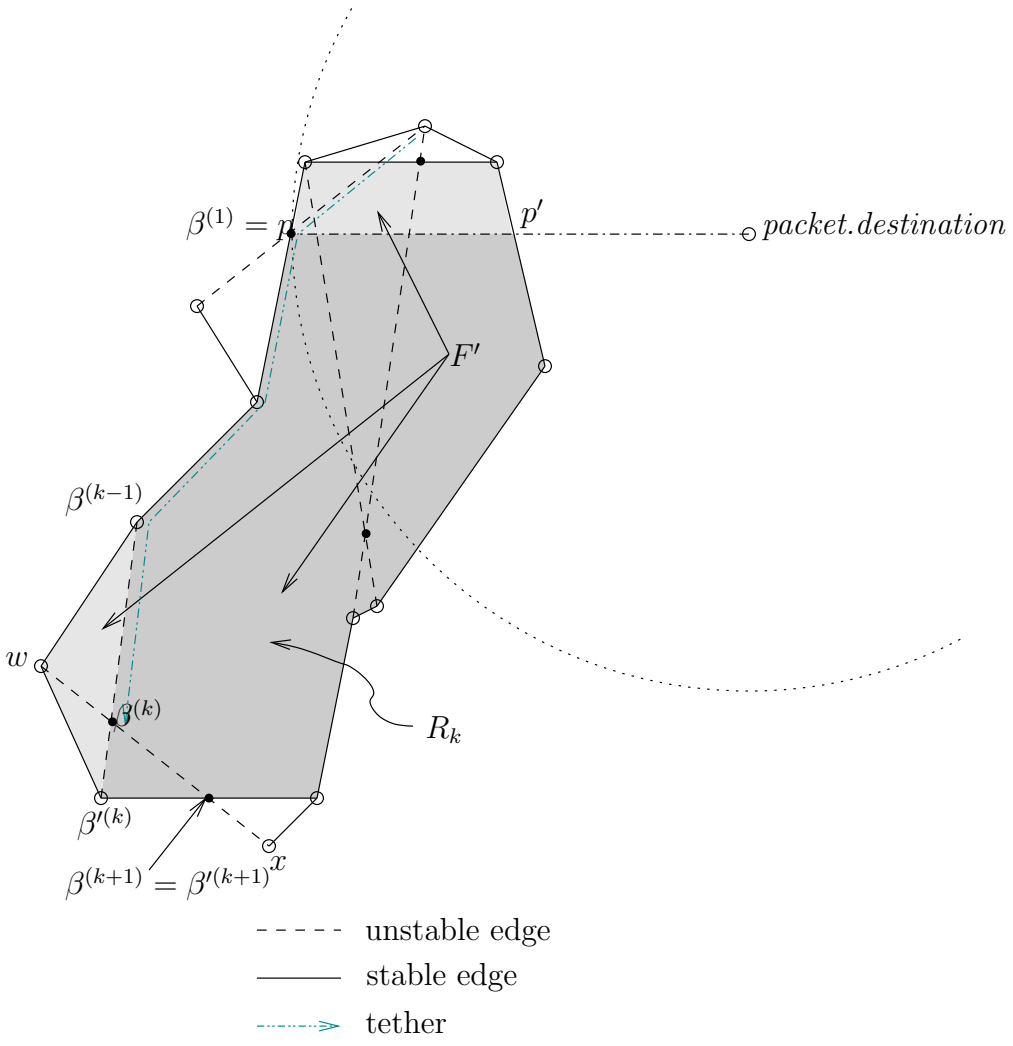


Figure 7.13: An example where F' is an interior virtual face of G'

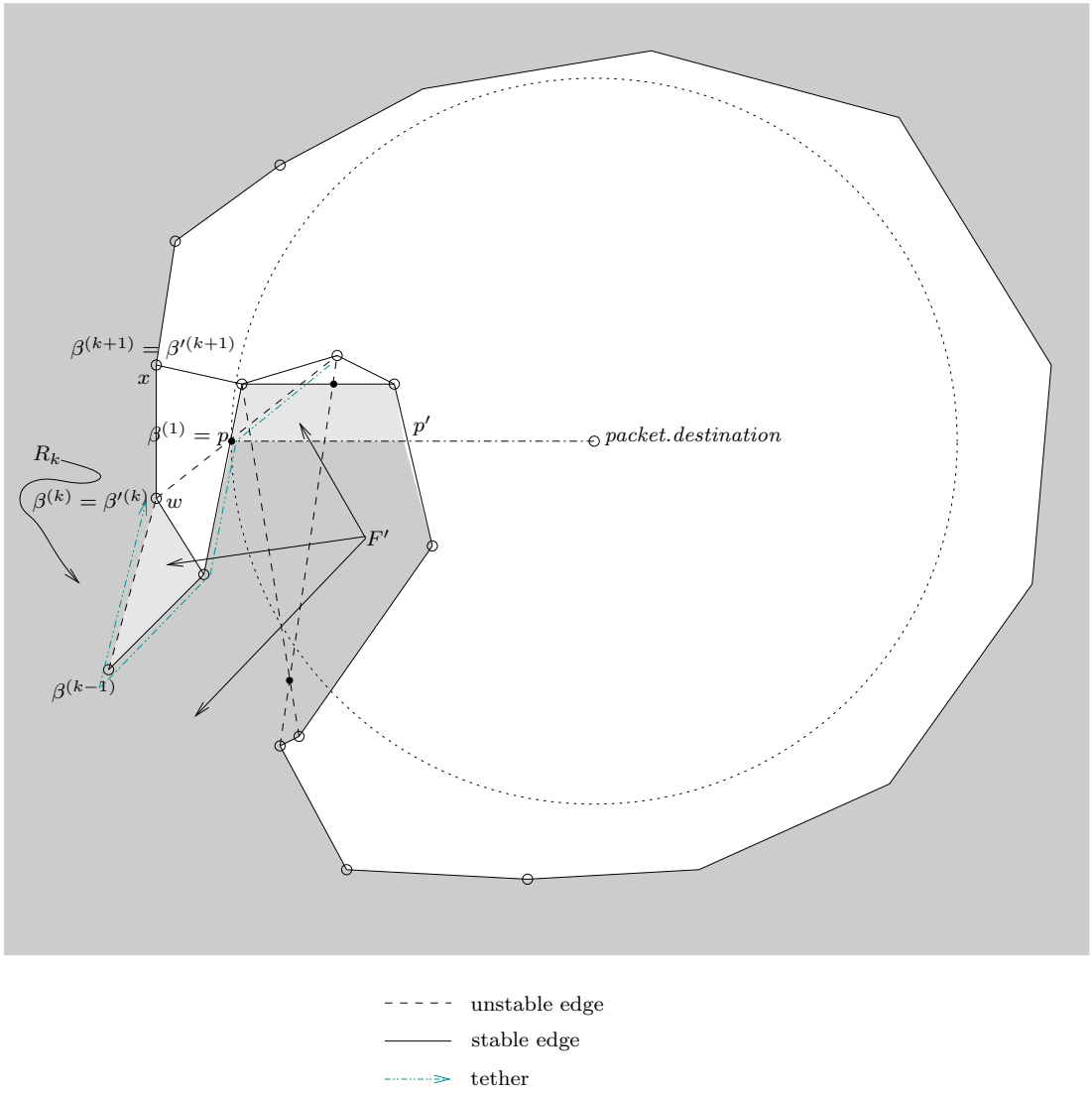


Figure 7.14: An example where F' is the outer virtual face of G'

Let (w, x) denote the network edge containing $(\beta^{(k)}, \beta^{(k+1)})$. First suppose $\beta'^{(k)} = \beta^{(k)}$. See Figure 7.15 for an example in this case where $k = 2$. Recall that $\beta'^{(k)}$ is a point

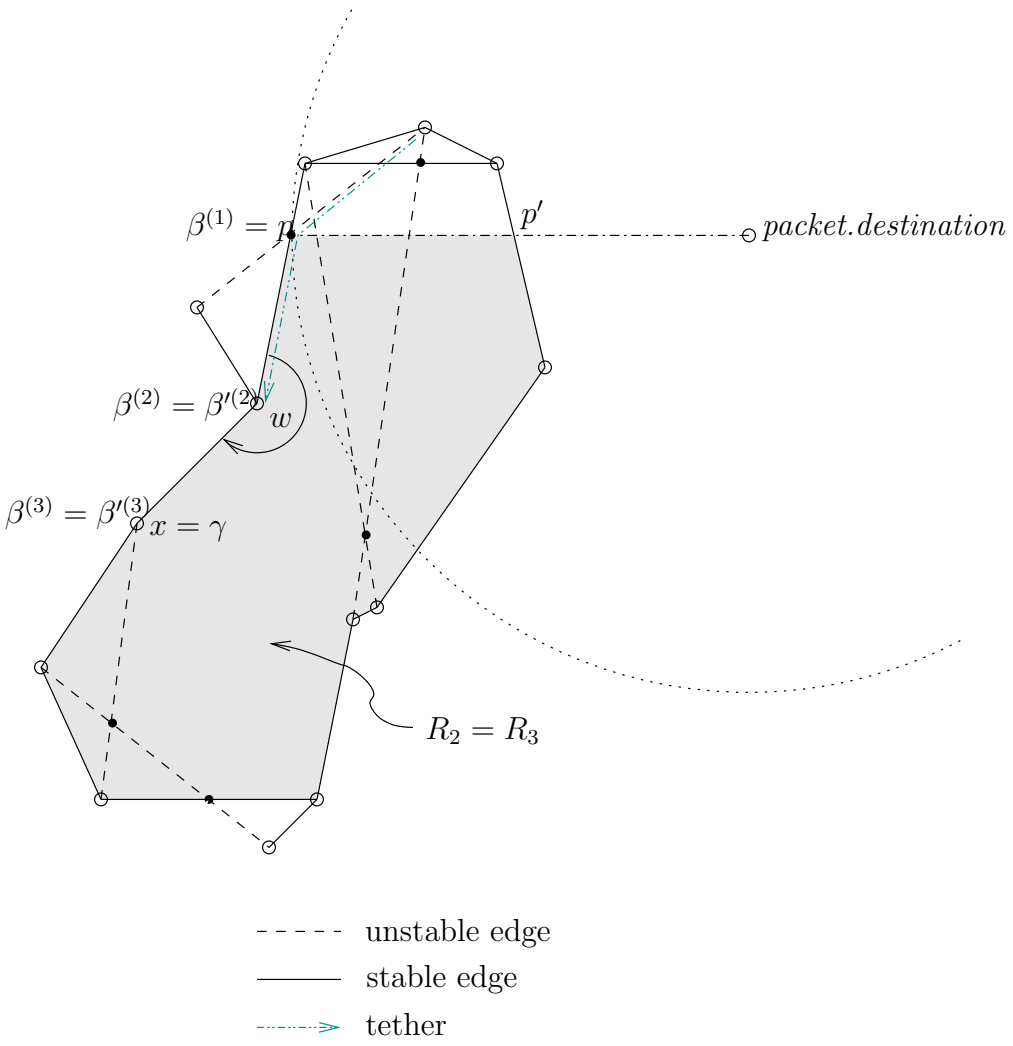


Figure 7.15: Example for the proof of Proposition 7.11 where $k = 2$

on $W_{F'}$. Let $(\beta^{(k)}, \gamma)$ be the first edge in $W_{F'}$ in clockwise order around $\beta^{(k)}$ after $(\beta^{(k)}, \beta^{(k-1)})$. Since the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $(\beta^{(k)}, \gamma)$ is in R_k and the boundary of R_k contains $(\beta^{(k-1)}, \beta^{(k)})$ followed by the part of $W_{F'}$ from $\beta'^{(k)}$ to p', $(\beta^{(k)}, \gamma)$ is on the part of $W_{F'}$ from $\beta'^{(k)}$ to p'. From Lemma 7.5 (a), $(\beta^{(k)}, \gamma)$ does not intersect the interior of the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $(\beta^{(k)}, \beta^{(k+1)})$. Therefore, $(\beta^{(k)}, \beta^{(k+1)})$ is either contained in $(\beta^{(k)}, \gamma)$ or is strictly between $(\beta^{(k)}, \beta^{(k-1)})$ and $(\beta^{(k)}, \gamma)$ in clockwise order around $\beta^{(k)}$.

If $(\beta^{(k)}, \beta^{(k+1)})$ is contained in $(\beta^{(k)}, \gamma)$, then $\beta'^{(k+1)} = \beta^{(k+1)}$ by definition. Figure 7.15 is an example of this case for $k = 2$. In this case, $(\beta^{(k)}, \beta^{(k+1)})$ is on the boundary of R_k. Thus, $R_{k+1} = R_k$ and the proposition holds. More formally, $\beta'^{(k+1)} \neq \beta'^{(k)}$ since $\beta'^{(k+1)} = \beta^{(k+1)}$, $\beta'^{(k)} = \beta^{(k)}$, and $\beta^{(k+1)} \neq \beta^{(k)}$. Furthermore, $\beta'^{(k+1)}$ is on the part of $W_{F'}$ from $\beta'^{(k)}$ to p', and the part of the tether from p to $\beta^{(k+1)}$ is the same as the part of the tether from p to $\beta^{(k)}$, $(\beta^{(k)}, \beta'^{(k)})$, plus the part of $W_{F'}$ from $\beta'^{(k)}$ to $\beta^{(k+1)} = \beta'^{(k+1)}$. Therefore, the part of the tether from p to $\beta^{(k+1)}$, $(\beta^{(k+1)}, \beta'^{(k+1)})$, the part of $W_{F'}$ from $\beta'^{(k+1)}$ to p', and the line segment from p' to p is just the boundary of R_k.

Therefore, we may assume that either (i) $\beta'^{(k)} = \beta^{(k)}$ and $(\beta^{(k)}, \beta^{(k+1)})$ is strictly between $(\beta^{(k)}, \beta^{(k-1)})$ and $(\beta^{(k)}, \gamma)$ in clockwise order around $\beta^{(k)}$; or (ii) $\beta'^{(k)} \neq \beta^{(k)}$. In both cases, we show that the first point after $\beta^{(k)}$ on (w, x) at which (w, x) intersects the boundary of R_k is not on $(\beta^{(k)}, \beta'^{(k)})$.

In case (i), the interior of $(\beta^{(k)}, \beta^{(k+1)})$ is in the interior of F'. Edge (w, x), which contains $(\beta^{(k)}, \beta^{(k+1)})$, intersects the interior of R_k and, thus, it intersects the boundary of R_k in at least one point other than $\beta^{(k)}$. Let λ be the first point after $\beta^{(k)}$ on (w, x) at which (w, x) intersects the boundary of R_k. Since $\beta^{(k)} = \beta'^{(k)}$, λ is not on $(\beta^{(k)}, \beta'^{(k)})$. See $k = 3$ in Figure 7.16 for an example of this case, where $\beta'^{(3)} = \beta^{(3)}$ and the network edge (w, x) containing $(\beta^{(3)}, \beta^{(4)})$ intersects the boundary of R_3 at another point $\beta'^{(4)}$.

An example of case (ii) is $k = 4$ in Figure 7.17. From Proposition 7.7, $\beta^{(k)}$ is a virtual node. From Proposition 7.6, the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $(\beta^{(k)}, \beta^{(k+1)})$ is less than 180°. Thus, edge (w, x) intersects the network edge containing $(\beta^{(k-1)}, \beta^{(k)})$ at $\beta^{(k)}$ and is not parallel to that edge. By the definition of $\beta'^{(k)}$, $(\beta^{(k)}, \beta'^{(k)})$ is contained in that network edge. Therefore, (w, x) does not intersect any point on $(\beta^{(k)}, \beta'^{(k)})$ except $\beta^{(k)}$. Furthermore, since $\beta^{(k)} \neq \beta'^{(k)}$, $\beta^{(k)}$ is not on $W_{F'}$. Then, by Lemma 7.5 (b), the interior of $(\beta^{(k)}, \beta^{(k+1)})$ is in the interior of F'. So (w, x) intersects the interior of R_k and, thus, it intersects the boundary of R_k in at least one point other than $\beta^{(k)}$. Let λ be the first point after $\beta^{(k)}$ on (w, x) at which (w, x) intersects the boundary

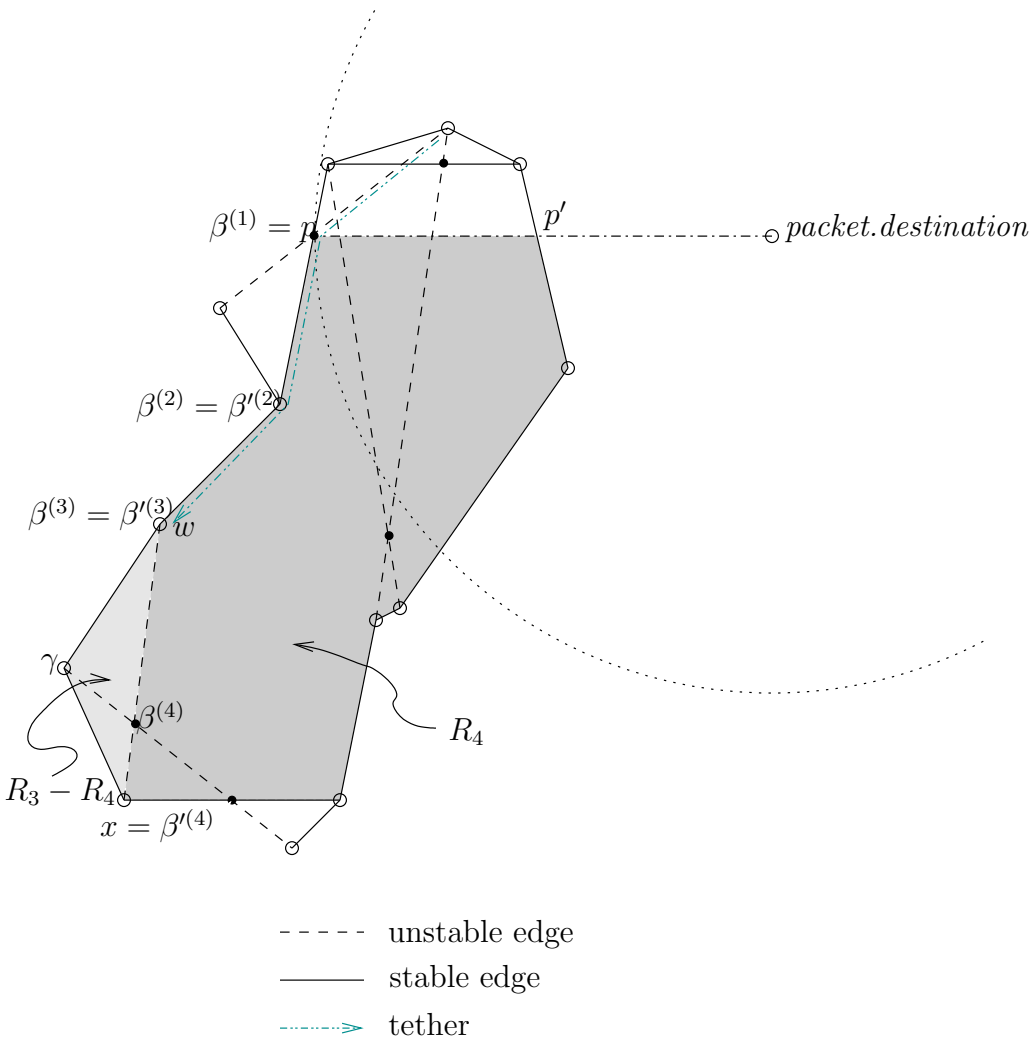


Figure 7.16: Example for the proof of Proposition 7.11 where $k = 3$

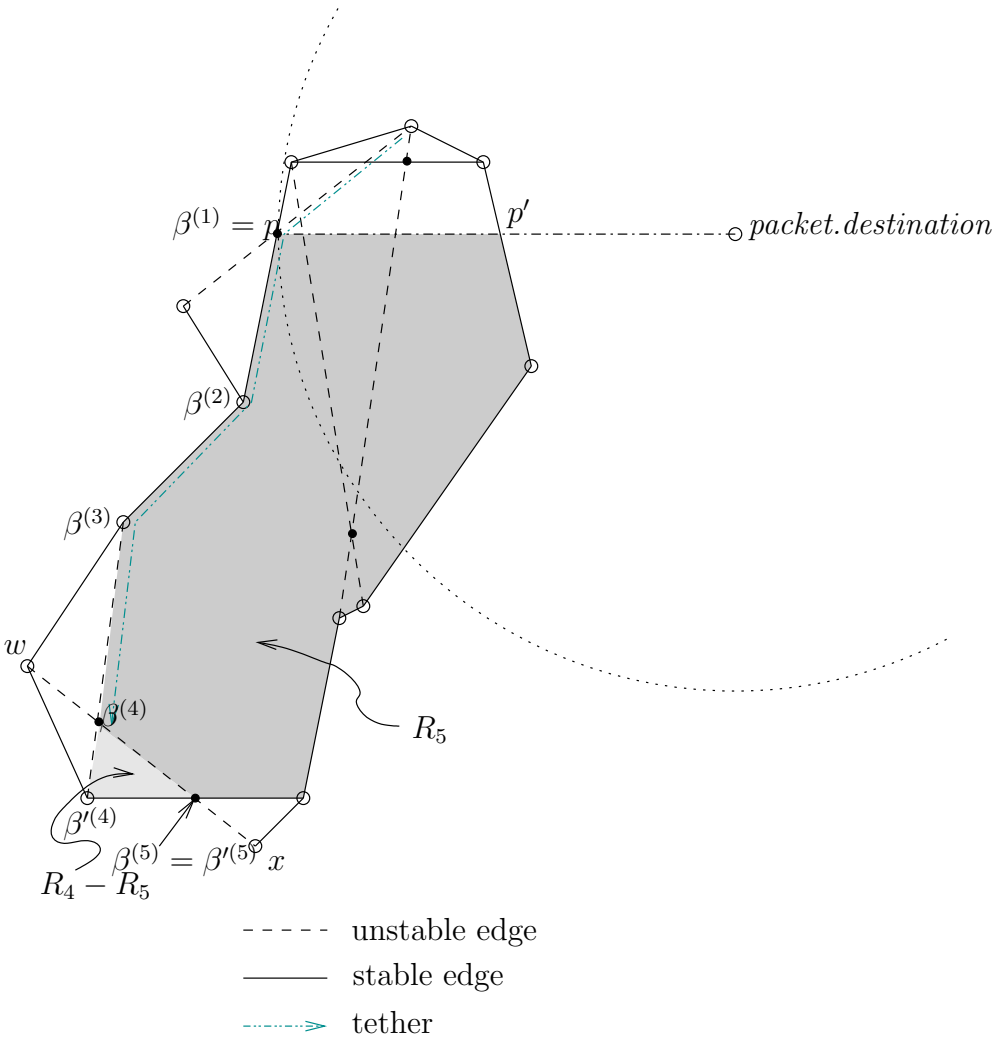


Figure 7.17: Example for the proof of Proposition 7.11 where $k = 4$

of R_k. Since (w, x) and $(\beta^{(k)}, \beta'^{(k)})$ intersect only at $\beta^{(k)}$, λ is not on $(\beta^{(k)}, \beta'^{(k)})$.

Thus, in both cases, λ is not on $(\beta^{(k)}, \beta'^{(k)})$ and the interior of $(\beta^{(k)}, \lambda)$ is in the interior of R_k. Thus, $(\beta^{(k)}, \lambda)$ divides R_k into two subregions. The boundary of R_k consists of the part of tether from p to $\beta^{(k)}$, $(\beta^{(k)}, \beta'^{(k)})$, the part of $W_{F'}$ from $\beta'^{(k)}$ to p', and the line segment from p' to p. We next show that λ can only be on the part of $W_{F'}$ after $\beta'^{(k)}$ up to and including p'.

λ cannot be on the part of the tether from p to $\beta^{(k)}$. Otherwise, $(\beta^{(k)}, \lambda)$ forms a counterclockwise cycle with the tether and (w, x) would have been ignored by the algorithm. This contradicts the fact that $(\beta^{(k)}, \beta^{(k+1)})$ is contained in (w, x).

Suppose λ is on the interior of the line segment from p' to p. Because the interior of $(\beta^{(k)}, \lambda)$ is in the interior of R_k and $\beta'^{(k+1)}$ is on $W_{F'}$, λ is between $\beta^{(k)}$ and $\beta'^{(k+1)}$ on edge (w, x). Since λ is closer to the destination than p is, and, by definition, no point on $(\beta^{(k)}, \beta^{(k+1)})$ is such a point, λ is not on $(\beta^{(k)}, \beta^{(k+1)})$. Since $(\beta^{(k+1)}, \beta'^{(k+1)})$ does not intersect the interior of the line segment between p and p', λ is not on $(\beta^{(k+1)}, \beta'^{(k+1)})$. This is a contradiction.

Therefore, λ can only be on the part of $W_{F'}$ after $\beta'^{(k)}$ up to and including p'. Since the interior of $(\beta^{(k)}, \beta^{(k+1)})$ is in the interior of F', $\beta^{(k+1)}$ is between $\beta^{(k)}$ and λ. Thus, by the definition of $\beta'^{(k+1)}$, $\beta'^{(k+1)} = \lambda$. Hence, $\beta'^{(k+1)} \neq \beta'^{(k)}$, $\beta'^{(k+1)}$ is on the part of $W_{F'}$ from $\beta'^{(k)}$ to p', and $(\beta^{(k)}, \beta'^{(k+1)})$ divides R_k into two subregions. The part of the tether from p to $\beta^{(k+1)}$, $(\beta^{(k+1)}, \beta'^{(k+1)})$, the part of $W_{F'}$ from $\beta'^{(k+1)}$ to p', and the line segment from p' to p is the boundary of one subregion of R_k divided by $(\beta^{(k)}, \beta'^{(k+1)})$. Therefore, the proposition is true. □

Proposition 7.12. *Let $k \geq 2$. If the tether has length at least k, and if $(\beta^{(i)}, \beta'^{(i)})$ does not intersect the interior of the line segment between p and p', for all $2 \leq i \leq k$, then the part of the tether from p to $\beta^{(k)}$, $(\beta^{(k)}, \beta'^{(k)})$, the part of $W_{F'}$ from $\beta'^{(k)}$ to p', and the line segment from p' to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F'.*

Proof. By induction on k using Proposition 7.10 and 7.11. □

Lemma 7.13. *Let $k \geq 2$. If the tether has length at least $k+1$, and if $(\beta^{(i)}, \beta'^{(i)})$ does not intersect the interior of the line segment between p and p', for all $2 \leq i \leq k+1$, then $\beta'^{(k+1)} \neq \beta'^{(k)}$ and $\beta'^{(k+1)}$ is on the part of $W_{F'}$ from $\beta'^{(k)}$ to p'.*

Proof. By Proposition 7.12, the part of the tether from p to $\beta^{(k)}$, $(\beta^{(k)}, \beta'^{(k)})$, the part of $W_{F'}$ from $\beta'^{(k)}$ to p', and the line segment from p' to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F'. Then, the lemma follows from Proposition 7.11. □

In the following proposition, we show a property of the tether when $(\beta^{(k)}, \beta'^{(k)})$ intersects the interior of the line segment between p and p' for the first time as k increases.

Proposition 7.14. *If the tether has length at least k_0, and $\beta^{(k_0)}$ is the first node in the tether such that $(\beta^{(k_0)}, \beta'^{(k_0)})$ intersects the interior of the line segment between p and p', then the part of the tether from p to $\beta^{(k_0-1)}$, $(\beta^{(k_0-1)}, \lambda)$, and the line segment from λ to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F', where λ is the intersection point of $(\beta^{(k_0)}, \beta'^{(k_0)})$ and the line segment between p and p'.*

Proof. Suppose the tether has length at least k_0, and $\beta^{(k_0)}$ is the first node in the tether such that $(\beta^{(k_0)}, \beta'^{(k_0)})$ intersects the interior of the line segment between p and p'. Figure 7.18 is an example for $k_0 = 8$.

From Proposition 7.9, if $\beta^{(2)}$ exists, $(\beta^{(2)}, \beta'^{(2)})$ does not intersect the interior of the line segment between p and p'. Therefore, $k_0 \geq 3$. Thus, from Proposition 7.12 with $k = k_0 - 1$, the part of the tether from p to $\beta^{(k_0-1)}$, $(\beta^{(k_0-1)}, \beta'^{(k_0-1)})$, the part of $W_{F'}$ from $\beta'^{(k_0-1)}$ to p', and the line segment from p' to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F'. Let R_{k_0-1} denote this region.

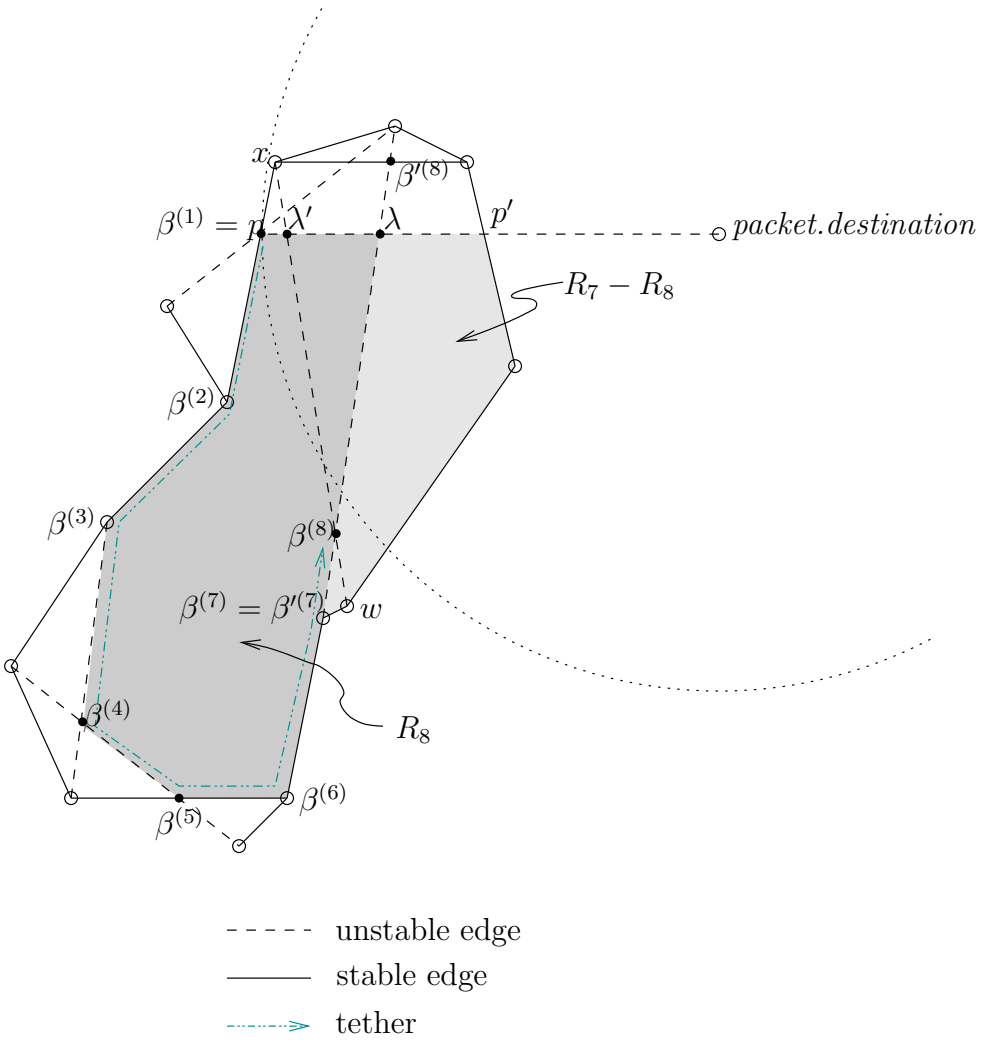


Figure 7.18: Example for the proof of Proposition 7.14

Since $(\beta^{(k_0)}, \beta'^{(k_0)})$ is contained in $(\beta^{(k_0-1)}, \beta'^{(k_0)})$, and $(\beta^{(k_0)}, \beta'^{(k_0)})$ intersects the interior of the line segment between p and p' at λ, $(\beta^{(k_0-1)}, \lambda)$ divides R_{k_0-1} into two subregions. The part of the tether from p to $\beta^{(k_0-1)}$, $(\beta^{(k_0-1)}, \lambda)$, and the line segment from λ to p is the boundary of one subregion of R_{k-1}. Hence, the proposition holds. □

The next proposition shows the properties of the next edge in the tether if the preceding edge $(\beta^{(k)}, \beta'^{(k)})$ intersects the interior of the line segment between p and p'. Notice that in the assumptions in this proposition, $(\beta^{(k)}, \beta'^{(k)})$ may not be the first edge that intersects the interior of the line segment between p and p', which is different from the assumption in Proposition 7.14.

Proposition 7.15. *If the tether has length at least $k+1$, if $(\beta^{(k)}, \beta'^{(k)})$ intersects the interior of the line segment between p and p' at λ, and if the part of the tether from p to $\beta^{(k-1)}$, $(\beta^{(k-1)}, \lambda)$, and the line segment from λ to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F', then $(\beta^{(k+1)}, \beta'^{(k+1)})$ intersects the interior of the line segment between p and λ at some point λ', and the part of the tether from p to $\beta^{(k)}$, $(\beta^{(k)}, \lambda')$, and the line segment from λ' to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F'.*

Proof. Let R_k denote the region whose boundary is the part of the tether from p to $\beta^{(k-1)}$, $(\beta^{(k-1)}, \lambda)$, and the line segment from λ to p.

Since $\beta^{(k)}$ exists, all points on $(\beta^{(k-1)}, \beta^{(k)})$ are farther from *packet.destination* than p is. Furthermore, since all point on the interior of the line segment between p and p' are closer to *packet.destination* than p is, $(\beta^{(k-1)}, \beta^{(k)})$ does not intersect the interior of the line segment between p and p'. If $\beta'^{(k)} = \beta^{(k)}$ then $(\beta^{(k)}, \beta'^{(k)})$ does not intersect the interior of the line segment between p and p', contrary to the assumption. Thus, $\beta'^{(k)} \neq \beta^{(k)}$, so by Proposition 7.7, $\beta^{(k)}$ is a virtual node.

Let (w, x) denote *packet.next_edge*$^{(k)}$. Edge (w, x) intersects $(\beta^{(k-1)}, \lambda)$ at $\beta^{(k)}$ and

$\beta^{(k+1)}$ lies on $(\beta^{(k)}, x)$. From Proposition 7.6, the clockwise angle around $\beta^{(k)}$ from $(\beta^{(k)}, \beta^{(k-1)})$ to $(\beta^{(k)}, \beta^{(k+1)})$ is less than 180°. Therefore, (w, x) intersects the interior of R_k.

The boundary of R_k consists of the part of the tether from p to $\beta^{(k-1)}$, $(\beta^{(k-1)}, \lambda)$, and the line segment from λ to p. Edge (w, x) intersects $(\beta^{(k-1)}, \lambda)$ at $\beta^{(k)}$, so it must intersect the boundary of R_k at another point that is not on $(\beta^{(k-1)}, \lambda)$. Edge (w, x) cannot intersect the part of the tether from p to $\beta^{(k-1)}$, because otherwise it forms a counterclockwise cycle with the tether and would have been ignored by the algorithm. Thus, (w, x) intersects the interior of the line segment from λ to p. Let λ' be the point at which they intersect. $(\beta^{(k)}, \lambda')$ divides R_k into two subregions. The part of the tether from p to $\beta^{(k)}$, $(\beta^{(k)}, \lambda')$, and the line segment from λ' to p is the boundary of one subregion of R_k.

Since $\beta^{(k+1)}$ exists, all points on $(\beta^{(k)}, \beta^{(k+1)})$ must be farther from *packet.destination* than p is. Thus, as above, $\beta'^{(k+1)} \neq \beta^{(k+1)}$ and $(\beta^{(k+1)}, \beta'^{(k+1)})$ intersects the interior of the line segment between p and λ at λ'. Therefore, the proposition holds. □

Using Proposition 7.14 and 7.15, we can prove that if $\beta^{(k_0)}$ is the first node in the tether such that $(\beta^{(k_0)}, \beta'^{(k_0)})$ intersects the interior of the line segment between p and p', then $(\beta^{(k)}, \beta'^{(k)})$ intersects the interior of the line segment between p and p', for all $k \geq k_0$ if $\beta^{(k)}$ exists.

Proposition 7.16. *If the tether has length at least $k \geq k_0$ and $\beta^{(k_0)}$ is the first node in the tether such that $(\beta^{(k_0)}, \beta'^{(k_0)})$ intersects the interior of the line segment between p and p', then $(\beta^{(k)}, \beta'^{(k)})$ intersects the interior of the line segment between p and p' at some point λ, and the part of the tether from p to $\beta^{(k-1)}$, $(\beta^{(k-1)}, \lambda)$, and the line segment from λ to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F'.*

Proof. By induction on $k \geq k_0$.

Base case $k = k_0$: follows from Proposition 7.14.

Induction step Assume the proposition is true for some $k \geq k_0$ and consider the case for $k+1$. Assume the tether has length at least $k+1$. By the induction hypothesis, $(\beta^{(k)}, \beta'^{(k)})$ intersects the interior of the line segment between p and p' at some point λ, and the part of the tether from p to $\beta^{(k-1)}$, $(\beta^{(k-1)}, \lambda)$, and the line segment from λ to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F'. Then, it follows from Proposition 7.15 that $(\beta^{(k+1)}, \beta'^{(k+1)})$ intersects the interior of the line segment between p and λ at some point λ', and the part of the tether from p to $\beta^{(k)}$, $(\beta^{(k)}, \lambda')$, and the line segment from λ' to p form a closed curve that is the boundary of a region whose interior is entirely contained in the interior of F'. Notice that, since λ is on the interior of the line segment between p and p', and λ' is on the interior of the line segment between p and λ, it follows that λ' is on the interior of the line segment between p and p'. Therefore the proposition holds for $k+1$. □

Combining Proposition 7.16 and 7.15 gives the following result.

Corollary 7.17. *If the tether has length at least $k+1 > k_0$ and $\beta^{(k_0)}$ is the first node in the tether such that $(\beta^{(k_0)}, \beta'^{(k_0)})$ intersects the interior of the line segment between p and p', then $(\beta^{(k)}, \beta'^{(k)})$ intersects the interior of the line segment between p and p' at some point λ and $(\beta^{(k+1)}, \beta'^{(k+1)})$ intersects the interior of the line segment between p and λ.*

Lemma 7.13 implies that as the tether grows, if $(\beta^{(k)}, \beta'^{(k)})$ does not intersect the interior of the line segment between p and p', then the length of the part of $W_{F'}$ from $\beta'^{(k)}$ to p' decreases. Note that $\beta'^{(k)}$ is a real or virtual node. Since there are a finite number of real and virtual nodes on $W_{F'}$, eventually, for some k, either the destination is reached, $(\beta^{(k-1)}, \beta^{(k)})$ contains a point closer to the destination (and thus the traversal switches to the next virtual face), or $\beta^{(k)}$ exists and $(\beta^{(k)}, \beta'^{(k)})$ intersects the interior of the line segment between p and p'.

If $\beta^{(k_0)}$ is the first node in the tether such that $(\beta^{(k_0)}, \beta'^{(k_0)})$ intersects the interior

of the line segment between p and p', and the tether has length at least $k+1 > k_0$, then Corollary 7.17 implies that $(\beta^{(k+1)}, \beta'^{(k+1)})$ intersects the interior of the line segment between p and λ_k, where λ_k is the point at which $(\beta^{(k)}, \beta'^{(k)})$ and the line segment between p and p' intersect. Since there are only a finite number of network edges intersecting the line segment between p and λ_{k_0}, there exists $k' \geq k_0$ such that either the destination is reached or $(\beta^{(k')}, \beta^{(k'+1)})$ contains a point closer to the destination (and thus the traversal switches to the next virtual face). This proves that message delivery is guaranteed.

Theorem 7.18. *Under Condition 7.1 and 7.2, Virtual-Face-Traversal-With-Tether guarantees message delivery in edge dynamic quasi unit disk graphs with* $\varepsilon \geq \frac{1}{\sqrt{2}}$.

Chapter 8

Some Results on Restricted Mobile Quasi Unit Disk Graphs

In this chapter, we present some results from our study on restricted mobile quasi unit disk graphs. First, we consider a special mobile quasi unit disk graph in which nodes do not move far away from their central locations and describe how to apply the Virtual-Face-Traversal-With-Tether protocol for routing in such a graph. Then, we consider more general mobile quasi unit disk graphs and discuss the difficulties in applying Virtual-Face-Traversal-With-Tether in such graphs. We study what kind of changes in the mobile graphs may cause problems for face routing. Finally, under the assumptions that the destination node is stationary and the speed of the movement of nodes is limited, we prove several properties of the graphs and conjecture that a slight variant of Virtual-Face-Traversal-With-Tether works correctly under these assumptions.

8.1 Routing in a Restricted Mobile Quasi Unit Disk Graph

In this section, we discuss an application of the Virtual-Face-Traversal-With-Tether protocol in a restricted mobile wireless ad-hoc network where each node may move only

within a small region around a central location.

Let G be a mobile quasi unit disk graph with parameter ε whose nodes each have a central location and may move only within a disk of radius δ around its central location. In this discussion, we consider the case where δ is small compared to the transmission range. Recall that the connectivity model of a mobile quasi unit disk graph is the same as for quasi unit disk graphs. If the distance between the central locations of two nodes is greater than $1+2\delta$, then the closest distance between them is greater than 1, so they are never connected by an edge. If the distance between the central locations of two nodes is less than or equal to $\varepsilon - 2\delta$, then the maximum distance between them is less than or equal to ε, so they are always connected by an edge. If the distance between the central locations of two nodes is greater than $\varepsilon - 2\delta$ and at most $1 + 2\delta$, these two nodes may or may not be connected by an edge as they move around.

Given such a mobile quasi unit disk graph G, we construct an edge dynamic quasi unit disk graph G' with parameter $(\varepsilon-2\delta)/(1+2\delta)$ as follows: For each node in G with central location (x, y), there is a corresponding node in G' at location $(x/(1 + 2\delta), y/(1 + 2\delta))$ and, at each point in time, there is an edge between two nodes in G' if and only if there is an edge between the corresponding nodes in G.

If the distance between two nodes in G' is greater than 1, then the distance between the central locations of the corresponding nodes in G is greater than $1 + 2\delta$, so they are never neighbors; if their distance in G' is at most $(\varepsilon - 2\delta)/(1 + 2\delta)$, then the distance between their central locations in G is at most $\varepsilon-2\delta$, and thus they are always neighbors.

Therefore, if $(\varepsilon - 2\delta)/(1 + 2\delta) \geq 1/\sqrt{2}$ or, equivalently, $\delta \leq (\sqrt{2}\varepsilon - 1)/(2(\sqrt{2} + 1))$, we can apply Virtual-Face-Traversal-With-Tether on G' to do routing in G. Note that to apply Virtual-Face-Traversal-With-Tether on G', nodes in G should use the scaled coordinates of their central locations instead of their current locations in the computation. This may be done by having nodes broadcasting their scaled central locations rather than current locations to their neighbors. Because the edge sets of G' and G are the same, a

route found in G' works in G as well.

Figure 8.1 shows the ranges of δ and ε for which Virtual-Face-Traversal-With-Tether guarantees message delivery in G. For example, if $\varepsilon = 1$, then $\delta \leq (\sqrt{2}-1)/(2(\sqrt{2}+1)) \approx 0.086$, i.e., about 8.6% of the transmission range. According to the IEEE 802.11 standards, a typical transmission range of current wireless devices is about 100 meters. In this case, $\delta \approx 0.086$ means that the mobile nodes may move only within a disk of radius about $8.6 \approx 100\delta$ meters. As ε decreases, the maximum value of δ also decreases. So, using this approach to guarantee message delivery, the mobile wireless network is quite restricted. One possible example of such a network is a collection of users in a residential area where each user has a laptop or handheld wireless device and may move within its own house. Another example is a wireless sensor network for a smart home, where sensors are embedded into furniture and appliances [2].

There are techniques using long-range antennas to increase the wireless transmission range from 100 meters to several kilometers. If such techniques are applied, a more realistic application could be a sensor network for tracking the locations of patients and doctors in a hospital.

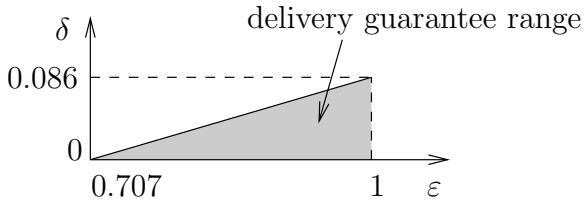


Figure 8.1: Values of δ and ε at which message delivery can be guaranteed

8.2 Towards More General Mobile Quasi Unit Disk Graphs

Now we consider more general cases of mobile quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$ where nodes are not constrained to remain near their central locations. We first identify some problems that can arise when the Virtual-Face-Traversal-With-Tether protocol is applied directly on such graphs. For routing to a stationary destination node, we prove that these problems cannot arise if we limit the speed at which nodes travel and use a slight variant of Virtual-Face-Traversal-With-Tether. We conjecture that this variant of Virtual-Face-Traversal-With-Tether works correctly in this restricted setting.

8.2.1 Problems caused by node movements

Node movements in mobile quasi unit disk graphs can cause changes to the graph that are much more complicated than that in edge dynamic quasi unit disk graphs. These changes cause difficulties for a routing protocol. First of all, in a general mobile quasi unit disk graph, the destination node may move during the routing process. Nodes forwarding a packet need to obtain the up-to-date location of the destination node if the destination node moves to a different location. How to keep track of the location of the destination node is a challenging problem.

Even if the destination node is stationary, problems can arise if other nodes move around during the routing process. In the following, we describe some problems that may arise when we apply the Virtual-Face-Traversal-With-Tether protocol to route a packet to a stationary destination node in mobile quasi unit disk graphs.

A problem may arise if, when a packet traverses the boundary of a face, the whole face moves farther from the destination. If all the points on the boundary of the face become farther away from the destination than the starting point is at the beginning of the traversal, then, Virtual-Face-Traversal-With-Tether will not switch to the next face,

and the packet returns to the starting point of the current face.

A face may change when a packet is travelling along its boundary. This happens if a new edge cuts through the face, an edge on the boundary of the face disappears, or a node moves across the boundary of the face. New edges and disappearing edges also occur in edge dynamic quasi unit disk graphs and these changes can be handled by the techniques in Virtual-Face-Traversal-With-Tether.

Problems can arise when nodes move across the boundary of the face. Consider the example in Figure 8.2. A packet is sent from node s to node d at time t_0. Due

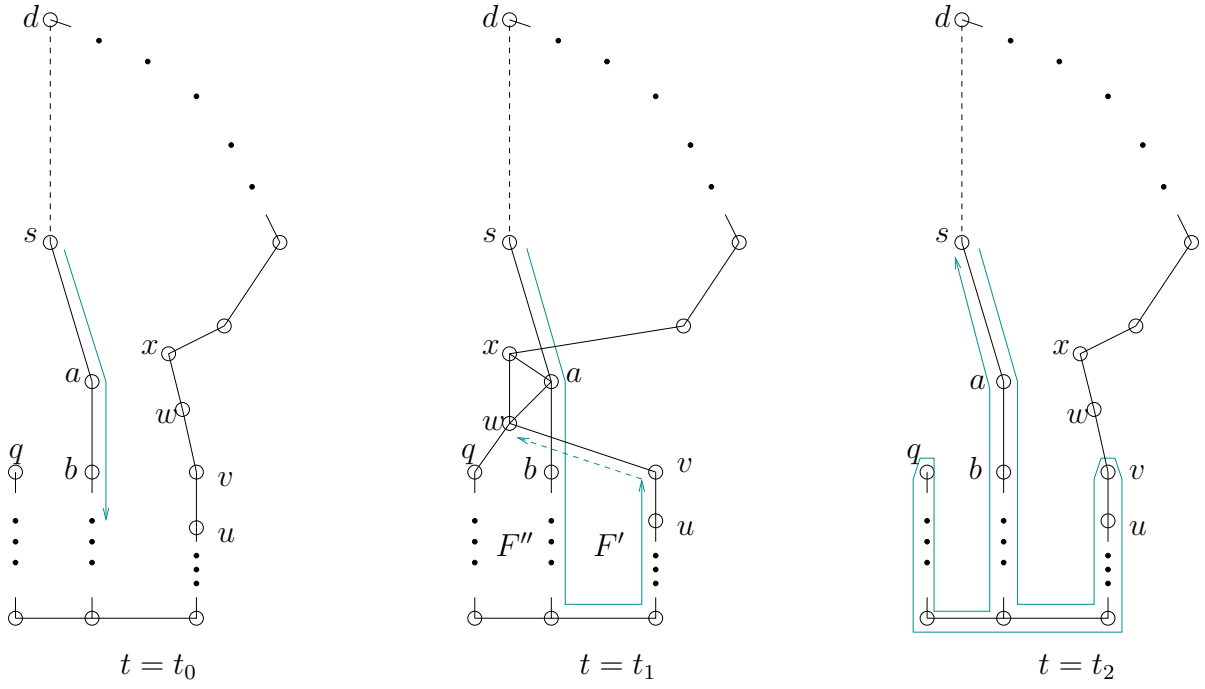


Figure 8.2: Example when Virtual-Face-Traversal-With-Tether cannot make progress

to the movements of nodes w and x, the graph has changed by the time t_1 at which the packet arrives at node v. In Virtual-Face-Traversal-With-Tether, the packet keeps track of the path it has followed. Edge (v, w) crosses an edge, (a, b), in that path. In Virtual-Face-Traversal-With-Tether, node v will send the packet back to node u, since a counterclockwise cycle is formed. However, as the packet continues to travel, the graph is restored to its original configuration. At time t_2, the packet returns to node s. In

this example, a is the only neighbor of s, so the same path is followed repeatedly. Thus, an adversary can prevent Virtual-Face-Traversal-With-Tether from making progress by repeatedly moving nodes w and x.

Notice that for the packet to be delivered to d, it must eventually be forwarded to node w. Even if a protocol did forward the packet to node w at time t_1, the right-hand rule implies that the packet would be forwarded to node q, beginning of the traversal of face F''. If the packet reaches b at time t_2, then it will, again, continue back to node s.

8.2.2 Limiting the speed of nodes

In the following, we restrict to the case where the destination node is located at a fixed location. For example, in many applications of wireless sensor networks, there is a stationary sink node in the network that acts as a data center, which collects information generated at sensor nodes that may move around [2, 63, 4].

Assumption 8.1. The destination node is stationary.

In the example in Figure 8.2, two edges (a, b) and (v, w), that were far away from each other at time t_0 intersect at time t_1. These edges are far away from each other in the sense that no endpoint of (a, b) can communicate directly with either endpoint of (v, w). The changes in Figure 8.2 are caused by nodes w and x moving a long distance during the traversal of the face. If nodes cannot move too far during the traversal, such changes will not occur.

If nodes do not move too fast, then during the traversal of a face, the graph does not change dramatically. In this case, we conjecture that the Virtual-Face-Traversal-With-Tether protocol guarantees message delivery. Specifically, we bound the distance nodes can travel during the traversal of one face.

Assumption 8.2. Nodes may move at most distance $\mu = \frac{1}{2}\sqrt{\varepsilon^2 - \frac{1}{4}}$ during the traversal of one face.

Note that $1/4 \leq \mu \leq \sqrt{3}/4$ for $1/\sqrt{2} \leq \varepsilon \leq 1$. With current technology, this is a reasonable assumption for a wireless mobile network. According to the IEEE 802.11 standards, at a typical transmission rate of 1 Mbit/sec, the link delay of a packet of length 512 bytes, typical for 802.11 packets, is 4 milliseconds. The transmission range of current wireless devices is about 100 meters. If the speed of mobile nodes is at most, say, 120 km/hour, typical for vehicles on a highway, the time for a node to move 1/4 of the transmission range is about 0.75 second. This is long enough for a packet to travel more than 150 hops.

All of the applications mentioned at the end of Section 8.1 could be implemented in this setting. Another example is a wireless sensor network deployed on vehicles to monitor traffic. Sensors embedded in cars can interact with each other and send information to fixed roadside data centers, where the data is processed to provide services such as traffic control. Yet another possible example is a wireless sensor network for tracking the movements of animals [2, 3].

Under Assumption 8.2, we have the following lemma.

Lemma 8.1. *During the traversal of a face, if node w moves across edge (u, v), then node w is a neighbor of either u or v at the start of the traversal.*

Proof. Suppose node w is not a neighbor of u nor a neighbor of v at the start of the traversal of a face. As illustrated in Figure 8.3, at the start of the traversal, node w is located outside the two circles with radius ε centered at u and v. Let a be one of the two intersections of the two circles, and let l, $l \leq 1$, be the length of edge (u, v). The distance from a to edge (u, v) is $\sqrt{\varepsilon^2 - \frac{l^2}{4}} \geq \sqrt{\varepsilon^2 - \frac{1}{4}} = 2\mu$.

During the traversal of the face, nodes u and v can move at most distance μ, so all points on edge (u, v) are within the shaded area in Figure 8.3. The distance from a to the shaded area is $\sqrt{\varepsilon^2 - \frac{l^2}{4}} - \mu \geq \mu$. Since all points, including w, that are outside both circles are farther from the shaded area than a, and w can move at most distance μ during

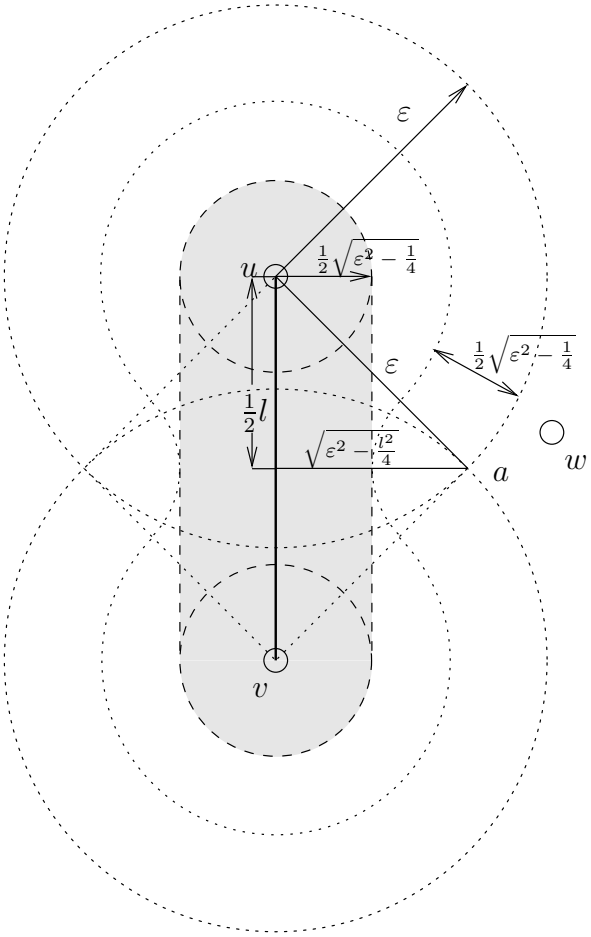


Figure 8.3: Proof of Lemma 8.1

the traversal, it cannot reach the shaded area and hence cannot cross edge (u, v). □

Thus, if edge (v, w) does not intersect edge (a, b), and neither a nor b is directly connected to v or w, then edges (v, w) and (a, b) cannot intersect each other during the traversal of one face. For example, if edges (v, w) and (a, b) do not intersect at the beginning of the traversal of a face, the subgraph induced by $\{a, b, v, w\}$ is depicted in Figure 8.4 (a) or Figure 8.4 (b). During the traversal of the face, the induced graph may change from (a) to (b), or from (b) to (c), but not from (a) to (c). Hence the changes in Figure 8.2 cannot happen under Assumption 8.2.

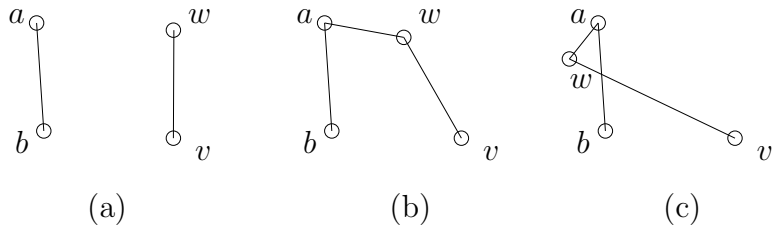


Figure 8.4: Changes of two edges (a, b) and (v, w)

Assumption 8.2 also provides a bound on the distance a face can move while it is being traversed. Using the following lemma, we show that if the starting point p of a face is the closest point to the destination among all paths known by a nearby node, then the distance from the face to the destination throughout the traversal of the face is less than the distance between p and the destination at the beginning of the traversal.

Lemma 8.2. *Let p be the starting point of the traversal of a virtual face F of a connected graph, and let edge (u, v) be a real edge that contains point p. Suppose that, at the start of the traversal, $|pv| \leq |pu|$, l is the distance from p to the destination node d, and $|p'd| \geq l$ for all points p' on paths of at most two hops starting from v. Then, throughout the traversal of F, the distance from F to d is less than l.*

Proof. Consider the location of nodes at the start of the traversal. Since $|p'd| \geq l$ for all points p' on paths of at most two hops starting from v, it follows that p is the closest point to node d on edge (u, v) and d is not a neighbor of v. Thus, in our model, $|vd| > \varepsilon$.

Let $C_v^{2\mu}$ be the circle centered at node v with radius $2\mu = \sqrt{\varepsilon^2 - \frac{1}{4}}$. Since $\varepsilon \geq 1/\sqrt{2}$, $2\mu \geq 1/2$. Let m be the midpoint of (u, v). Then, $|pv| \leq |mv| \leq 1/2 \leq 2\mu$, because $|pv| \leq |pu|$ and $|uv| \leq 1$. Thus, p is inside or on the boundary of the circle $C_v^{2\mu}$. Since $|vd| > \varepsilon > 2\mu$, it follows that d is outside the circle $C_v^{2\mu}$. Let $C_d^{|pd|}$ be the circle centered at node d with radius $|pd|$. Since p is the closest point to node d on edge (u, v) and $|pv| \leq |pu|$, either $p = v$ or edge (u, v) is perpendicular to pd, the line segment between p and d.

Figure 8.5 shows an example of the special case where $p = v$, Figure 8.6 illustrates the geometry of a general case where p is between v and m, and Figure 8.7 is a special case where $\varepsilon = 1/\sqrt{2}$, $2\mu = 1/2 = |pv|$, and $|vd| = 1/\sqrt{2} + \xi$ for some $\xi << 1$.

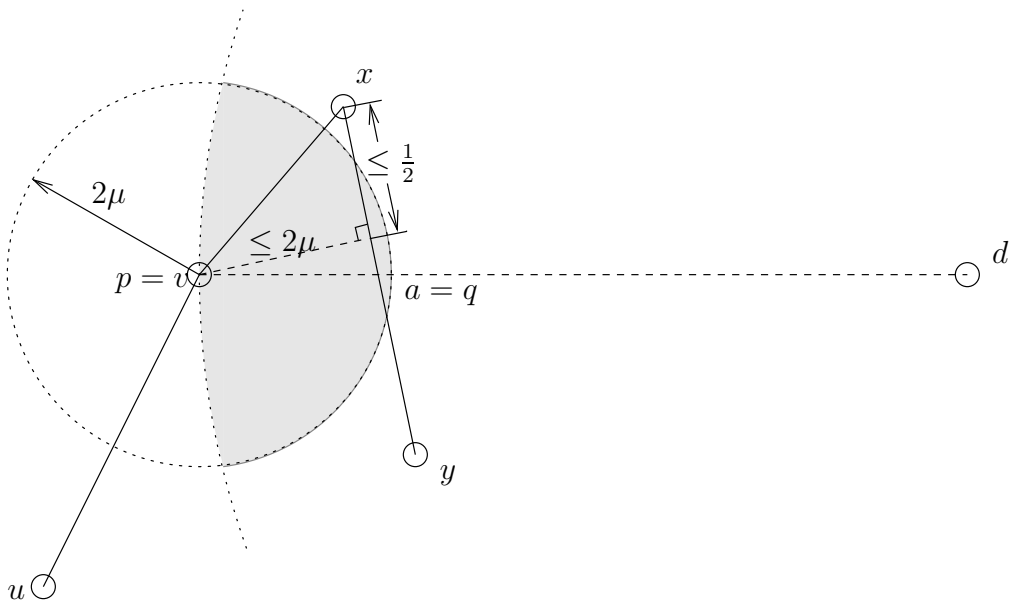


Figure 8.5: A special case where the starting point p is node v

Let A be the area in the intersection of $C_v^{2\mu}$ and $C_d^{|pd|}$, including the arc on $C_v^{2\mu}$ and excluding the arc on $C_d^{|pd|}$ and their intersection points, shown as the shaded area in the figures. We prove that A and F have a non-empty intersection by considering two cases: $|pv| < 2\mu$ and $|pv| = 2\mu$.

If $|pv| < 2\mu$, then p is inside $C_v^{2\mu}$ and, thus, A is not empty. Figure 8.5 and Figure 8.6

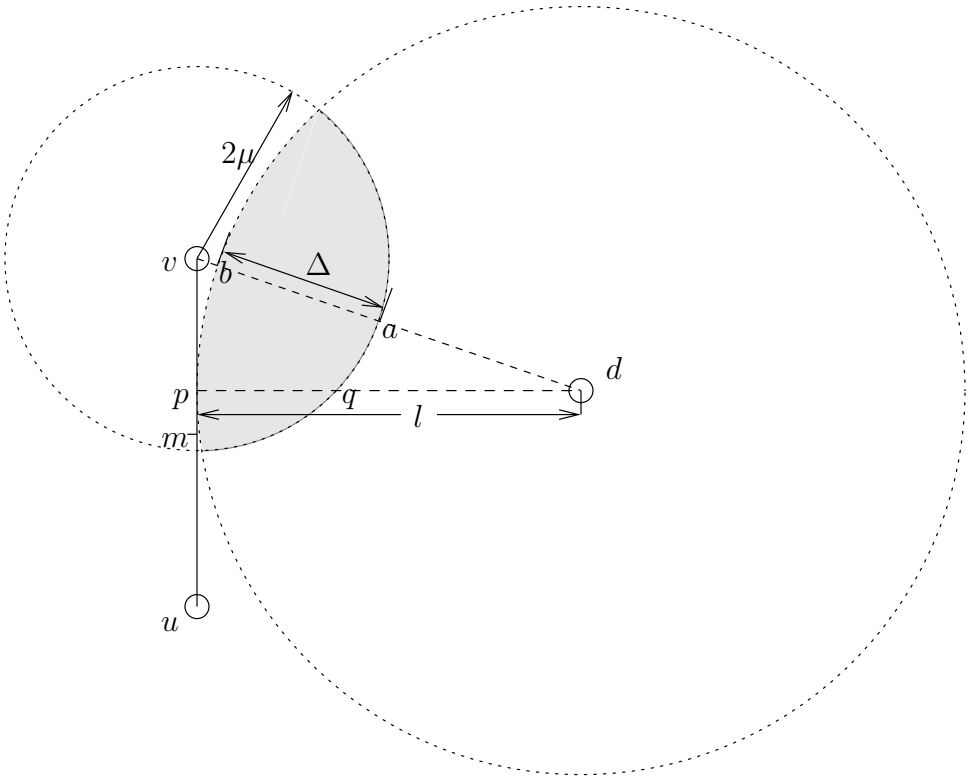


Figure 8.6: A general case of the starting point p and edge (u, v)

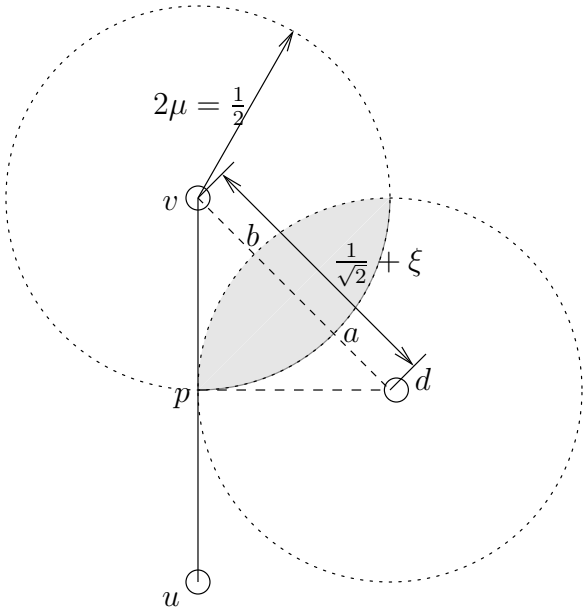


Figure 8.7: A special case where $|pv| = 2\mu = 1/2$ and $|vd| = \frac{1}{\sqrt{2}} + \xi$, where $\xi << 1$

are examples of this case. Let q be the intersection point of the line segment pd and $C_v^{2\mu}$. Since p is inside $C_v^{2\mu}$ and q is on the boundary of $C_v^{2\mu}$, the interior of pq is inside $C_v^{2\mu}$. Also, since pq is part of a radius of $C_d^{|pd|}$, the interior of pq is inside $C_d^{|pd|}$. Therefore, the interior of pq is inside A. From the definition of face routing, F is intersected by an initial part of the line segment pd with non-zero length. Thus, the interior of pq intersects F. Therefore, A intersects F.

Otherwise, $|pv| = 2\mu$. In this case, p is on the boundary of $C_v^{2\mu}$. Since $2\mu \geq 1/2 \geq |pv|$, it follows that $2\mu = 1/2 = |pv|$ and $\varepsilon = 1/\sqrt{2}$. Since $|uv| \leq 1$ and $|pv| \leq |pu|$, it follows that $|uv| = 1$ and p is the midpoint of (u, v). Figure 8.7 is an example of this case.

Since pd is perpendicular to (u, v), it follows by the Pythagorean theorem that $|vd|^2 = |vp|^2 + |pd|^2$. Since $|vd| > \varepsilon = 1/\sqrt{2} > 1/2 = |pv|$, we have $|pd| > 0$. Thus, $(2\mu + |pd|)^2 = (|vp| + |pd|)^2 > |vp|^2 + |pd|^2 = |vd|^2$. Hence $2\mu + |pd| > |vd|$ and the interior of the intersection A of the circles $C_v^{2\mu}$ and $C_d^{|pd|}$ is non-empty.

Let e' be the first virtual edge in the counterclockwise direction around p starting from pd, and let e be the real edge that contains e'. From the definition of face routing, e' is on the boundary of F. If e' overlaps pd, then p is the closest point to v on edge e and the distance from e to v is $1/2$. At least one of the endpoints of e is at most distance ε from v: otherwise the length of e is greater than $2\sqrt{\varepsilon^2 - (\frac{1}{2})^2} = 1$, which is not true. Therefore, at least one endpoint of e is a neighbor of v, and e is on a path of two hops starting from v. Hence $|p'd| \geq l$ for all points p' on edge e. Because e' overlaps pd, it contains a point p' that is closer to d than p is, i.e. $|p'd| < l$. This is a contradiction. Therefore, e' does not overlap pd.

The counterclockwise angle from pd to e' around p is greater than 0. Since pd is tangent to $C_v^{2\mu}$ at p, a segment of the arc on $C_v^{2\mu}$ starting at p that is on the boundary of A is contained in the interior of that angle. Therefore, A intersects F.

In both cases, A intersects F. If some point on the boundary of F belongs to A, let (x, y) be a real edge that contains such a point where $|xv| \leq |yv|$. See Figure 8.5 for an

example. From the definition of A, the distance from every point in A to v is at most 2μ, and the distance from every point in A to d is smaller than l. Therefore, the distance from edge (x, y) to v is at most 2μ, and the distance from edge (x, y) to d is less than l.

Let z be the point on edge (x, y) that is closest to v. If $z = x$, then $|xv| \leq 2\mu < \varepsilon$. Otherwise, vz is perpendicular to (x, y). Since $|xv| \leq |yv|$ and $|xy| \leq 1$, $|xz| \leq 1/2$. Hence $|xv| = \sqrt{|vz|^2 + |xz|^2} \leq \sqrt{(2\mu)^2 + (\frac{1}{2})^2} = \varepsilon$. In both cases, node x is a neighbor of v.

Thus, edge (x, y) is on a two-hop path from v, and $|p'd| \geq l$ for all points p' on edge (x, y). This contradicts the fact that the distance from edge (x, y) to d is less than l. Hence, A is entirely contained in the interior of F.

Since the graph is connected, node d cannot be in the interior of F. Thus, the boundary of F contains a point that is closer to d than all points in A. Since nodes can move at most distance μ during the traversal of F, and the destination node d is stationary, the distance from F to d can increase by at most μ during the traversal. Thus, it suffices to show that A contains a point whose distance to d is at most $l - \mu$.

Let a and b be the intersection point of the line segment vd with $C_v^{2\mu}$ and $C_d^{|pd|}$, respectively. Point a belongs to A. We will show that the distance $|ad|$ from a to d is less than $l - \mu$.

Let $\Delta = l - |ad|$. Then, $\Delta = l - (|vd| - |va|) = |pd| - |vd| + |va| = \sqrt{|vd|^2 - |pv|^2} - |vd| + 2\mu$, since $p = v$ or pd is perpendicular to pv. As $|pv|$ decreases, Δ increases, and as $|vd|$ increases, Δ does not decrease, because

$$\frac{\partial(\sqrt{|vd|^2 - |pv|^2} - |vd| + 2\mu)}{\partial |vd|} = \frac{|vd|}{\sqrt{|vd|^2 - |pv|^2}} - 1 \geq 0.$$

Since $|vd| > \varepsilon$ and $|pv| \leq 1/2$, we obtain $\Delta \geq \sqrt{\varepsilon^2 - (\frac{1}{2})^2} - \varepsilon + 2\mu = 2\sqrt{\varepsilon^2 - \frac{1}{4}} - \varepsilon$. Since $\varepsilon \geq 1/\sqrt{2}$, $\varepsilon^2 \geq \frac{1}{2} > \frac{9}{20}$, so $\varepsilon^2 - \frac{1}{4} > (\frac{2\varepsilon}{3})^2$. Hence $\frac{3}{2}\sqrt{\varepsilon^2 - \frac{1}{4}} > \varepsilon$, and thus

$$l - |ad| = \Delta \geq 2\sqrt{\varepsilon^2 - \frac{1}{4}} - \varepsilon > \frac{1}{2}\sqrt{\varepsilon^2 - \frac{1}{4}} = \mu.$$

Therefore, $|ad| < l - \mu$. □

If the conditions in Lemma 8.2 are satisfied, then during the traversal, the boundary of the next virtual face cannot move farther from the destination than the starting point was at the start of the traversal. Therefore, during the traversal, we can always find a new starting point. A slight modification of Virtual-Face-Traversal-With-Tether tries to ensure that the conditions are satisfied before starting to traverse the next virtual face: After a new starting point is determined, let the real node that is closer to the starting point check whether it sees any point on a two-hop path from it that is closer to the destination than the starting point is. If so, forward the packet towards that point and use that point as a new starting point. Otherwise, the packet starts to traverse the next virtual face. In this case, from Lemma 8.2, we know that the boundary of the current virtual face contains a point that remains closer to the destination than the starting point was at the start of the traversal.

We conjecture that this variant of Virtual-Face-Traversal-With-Tether works correctly in mobile quasi unit disk graphs with $\varepsilon \geq \frac{1}{\sqrt{2}}$ under conditions 7.1, 7.2, 8.1, and 8.2. To prove this, we need to show that this algorithm works correctly for all possible changes to the graph during the routing process. The challenges are to characterize the possible changes to the graph and to find some invariants that are useful for the proof of correctness.

Chapter 9

Conclusions and Future Work

In this thesis, we presented our research on extending face routing to more general models of wireless ad-hoc networks. In order to investigate the extendibility of face routing, we developed a series of models with increasing generality. One important part of our research is to find appropriate models through which we can better understand the difficulties of routing problems in more realistic network graphs than unit disk graphs. The graph models we considered gradually incorporate aspects of real networks, which enabled us to identify different types of problems, to gain an understanding of the capability and limitations of face routing, and to find techniques to extend the original face routing protocols.

We developed techniques that extend and generalize the face routing approach so that it can be applied directly on general non-planar network graphs, without separately constructing a plane routing subgraph. Our techniques also extend face routing to network graphs with unstable links that may change during the routing process. Using these techniques, we developed face routing protocols without the constraints suffered by the face routing protocols in the literature.

Face routing does not always work in a general non-static graph. For example, in my M.Sc. thesis, examples were given where face routing cannot deliver a packet, even

if the nodes are stationary. We are interested in identifying conditions under which face routing can guarantee message delivery. Therefore, in our research, we focus on designing deterministic algorithms and providing theoretical proofs for guaranteed message delivery in each graph model. These results are important for applications where guaranteeing message delivery is critical.

There are a few interesting problems we would like to investigate further. First of all, we would like to prove that the variant of Virtual-Face-Traversal-With-Tether discussed at the end of Chapter 8 guarantees message delivery under conditions 7.1, 7.2, 8.1, and 8.2. Alternatively, if this is not the case, we would like to find additional constraints under which message delivery is guaranteed, or to modify the protocol further to handle the counterexamples.

Our results in Section 8.2 assumed that the destination node is stationary. With a mobile destination node, the routing problem becomes even more difficult. It is possible that face routing could be combined with an efficient location update scheme to guarantee message delivery in such networks. One example is the Last Encounter Routing scheme analyzed in [26]: a source node forwards a packet to the last known location of the destination node, which was its location at some time in the past. During routing, if a more up-to-date location of the destination is learned, then the packet is diverted to this new destination.

Another direction for future work is to investigate face routing in graphs embedded in other kinds of surfaces or in high-dimensional spaces. For example, Fraser [15] studies face routing algorithms for static graphs embedded in the torus. Durocher et al. [13] study the problem for static graphs embedded in 3-dimensional Euclidean space by projecting the graph onto a 2-dimensional plane. They consider static graphs embedded in 3-dimensional Euclidean space where two nodes are neighbors if and only if their distance is at most 1. They show that if nodes are contained within a slab of thickness $1/\sqrt{2}$, routing can be done by projecting the graph onto a plane parallel to the slab and applying

a face routing protocol for quasi unit disk graphs on the resulting graph, which is a quasi unit disk graph with $\varepsilon = 1/\sqrt{2}$.

Durocher et al. also consider a restricted class of routing algorithms, called *k-local* algorithms, in which routing decisions are based only on the source and destination nodes, the k-hop neighborhood, and the previous node on the path. Notice that face routing algorithms are not k-local, because they need information about the starting point of the current face traversal. They prove that, for every $\varepsilon < \frac{1}{\sqrt{2}}$ and every $k \geq 1$, no k-local routing algorithm guarantees message delivery in all quasi unit disk graphs with parameter ε. It remains open whether this is also true for face routing.

Although face routing protocols guarantee message delivery in static networks, the length of a path computed by them could be significantly longer than the shortest path between the source and the destination in the network. Since the face routing protocols in the literature use a subgraph of the network as the routing graph, the length of the shortest path between the source and the destination may increase in the routing graph. Experiments have been performed to evaluate the performance of a variety of face routing protocols and the protocols that combine face routing with greedy routing [7, 31, 40, 37]. The length of the paths computed by those protocols are compared with the length of the shortest path between the source and the destination in the original network graphs. Our virtual face routing protocols are applied directly on the network graphs. It would also be interesting to perform the same experiments for our virtual face routing protocols to evaluate their performance.

In this thesis, we did not try to minimize the length of the route. The techniques used by GOAFR+ [37] to do this can also be applied to our protocols. In addition, because a virtual plane graph may contain many small faces, we may be able to exploit the two-hop or three-hop neighbor information to skip some small faces or to switch to a new face earlier.

For edge dynamic and mobile quasi unit disk graphs, our protocols guarantee mes-

sage delivery assuming the existence of a stable spanning subgraph during the traversal of each face. Experimental study may help us understand how the parameters of the networks, such as the frequency of link failure, the speed of movement, and the speed of communication, affect the performance of our protocols and the likelihood that the conditions for guaranteeing message delivery are satisfied.

Bibliography

[1] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurements from an 802.11b mesh network. *SIGCOMM Comput. Commun. Rev.*, 34(4):121–132, 2004.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Comput. Netw.*, 38(4):393–422, 2002.

[3] Ian F. Akyildiz, Tommaso Melodia, and Kaushik R. Chowdhury. A survey on wireless multimedia sensor networks. *Comput. Netw.*, 51(4):921–960, 2007.

[4] J.N. Al-Karaki and A.E. Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE*, 11(6):6–28, Dec. 2004.

[5] Lali Barrière, Pierre Fraigniaud, Lata Narayanan, and Jaroslav Opatrny. Robust position-based routing in wireless ad hoc networks with irregular transmission ranges. *Wireless Communications and Mobile Computing*, 3(2):141–153, 2003.

[6] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A distance routing effect algorithm for mobility (dream). In *Proceedings of the fourth annual ACM/IEEE international conference on Mobile computing and networking*, pages 76–84. ACM Press, 1998.

[7] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[8] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the fourth annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97. ACM Press, 1998.

[9] Benjamin A. Chambers. The grid roofnet: a rooftop ad hoc wireless network. Master's thesis, Department of Electrical Engineering and Computer Science, MIT, June 2002.

[10] C. Chiang, H. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks with fading channel. In *The IEEE Singapore International Conference on Networks*, pages 197–211, April 1997.

[11] Samir R. Das, Robert Castañeda, and Jiangtao Yan. Simulation-based performance evaluation of routing protocols for mobile ad hoc networks. *Mobile Networks and Applications*, 5(3):179–189, 2000.

[12] Douglas S. J. De Couto, Daniel Aguayo, Benjamin A. Chambers, and Robert Morris. Performance of multihop wireless networks: Shortest path is not enough. In *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, New Jersey, October 2002. ACM SIGCOMM.

[13] Stephane Durocher, David G. Kirkpatrick, and Lata Narayanan. On routing with guaranteed delivery in three-dimensional ad hoc wireless networks. In Shrisha Rao, Mainak Chatterjee, Prasad Jayanti, C. Siva Ram Murthy, and Sanjoy Kumar Saha, editors, *ICDCN*, volume 4904 of *Lecture Notes in Computer Science*, pages 546–557. Springer, 2008.

[14] Gregory G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. Technical Report ISI/RR-87-180, ISI, March 1987.

[15] Maia Fraser. Local routing on tori. In Evangelos Kranakis and Jaroslav Opatrny, editors, *ADHOC-NOW*, volume 4686 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2007.

[16] Hannes Frey and Ivan Stojmenovic. On delivery guarantees of face and combined greedy-face routing in ad hoc and sensor networks. In *MobiCom '06: Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 390–401, New York, NY, USA, 2006. ACM Press.

[17] K. Ruben Gabriel and Robert R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, Vol. 18(No. 3):259–278, Sep. 1969. Stable URL: http://links.jstor.org/sici?sici=0039-79893E2.0.CO

[18] Jie Gao, Leonidas J. Guibas, John Hershberger, Li Zhang, and An Zhu. Geometric spanner for routing in mobile networks. In *The 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'01)*, pages 45–55, 2001.

[19] S. Giordano and M. Hamdi. Mobility management: The virtual home region. Technical Report SSC/1999/037, EPFL, Lausanne, Switzerland, October 1999.

[20] Tom Goff, Nael B. Abu-Ghazaleh, Dhananjay S. Phatak, and Ridvan Kahvecioglu. Preemptive routing in ad hoc networks. In *Proceedings of the seventh annual international conference on Mobile computing and networking*, pages 43–52. ACM Press, 2001.

[21] Xiaoyang Guan. Routing in ad hoc networks using location information. Master's thesis, Department of Computer Science, University of Toronto, 2003.

[22] Xiaoyang Guan. Face traversal routing on edge dynamic graphs. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 12*, page 244.2, Washington, DC, USA, 2005. IEEE Computer Society.

[23] Zygmunt J. Haas and Marc R. Pearlman. The performance of query control schemes for the zone routing protocol. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 167–177. ACM Press, 1998.

[24] Lingxuan Hu and David Evans. Localization for mobile sensor networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 45–57. ACM Press, 2004.

[25] J. P. Hubaux, Th. Gross, J. Y. Le Boudec, and M. Vetterli. Towards self-organized mobile ad hoc networks: the Terminodes project. *IEEE Communications Magazine*, 31(1):118–124, 2001.

[26] Efstratios Ioannidis. Towards an understanding of last encounter routing in ad hoc networks. Master's thesis, University of Toronto, 2004.

[27] Kleoni E. Ioannidou. *Dynamic quorum systems in mobile ad-hoc networks*. PhD thesis, University of Toronto, 2006.

[28] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based performance analysis of routing protocols for mobile ad-hoc networks. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 195–206. ACM Press, 1999.

[29] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[30] Elliott D. Kaplan. *Understanding GPS: principles and applications*. Artech House, 1996.

[31] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, 2000.

[32] Wolfgang Kieß, Holger Füßler, Jörg Widmer, and Martin Mauve. Hierarchical location service for mobile ad-hoc networks. *Mobile Computing and Communications Review*, 8(4), October 2004.

[33] Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. On the pitfalls of geographic face routing. In *DIALM-POMC '05: Proceedings of the 2005 joint workshop on Foundations of mobile computing*, pages 34–43, New York, NY, USA, 2005. ACM Press.

[34] Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. Lazy cross-link removal for geographic routing. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 112–124, New York, NY, USA, 2006. ACM Press.

[35] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. In *Proceedings of the fourth annual ACM/IEEE international conference on Mobile computing and networking*, pages 66–75. ACM Press, 1998.

[36] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *Proc. 11th Canadian Conference on Computational Geometry*, pages 51–54, Vancouver, August 1999.

[37] Fabian Kuhn, Roger Wattenhofer, Yan Zhang, and Aaron Zollinger. Geometric ad-hoc routing: of theory and practice. In *Proceedings of the Twenty-second Annual*

Symposium on Principles of Distributed Computing (PODC'03), pages 63–72. ACM Press, 2003.

[38] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *DIALM '02: Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 24–33, New York, NY, USA, 2002. ACM Press.

[39] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Ad-hoc networks beyond unit disk graphs. In *DIALM-POMC '03: Proceedings of the 2003 joint workshop on Foundations of mobile computing*, pages 69–78. ACM Press, 2003.

[40] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Worst-case optimal and average-case efficient geometric ad-hoc routing. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 267–278, New York, NY, USA, 2003. ACM Press.

[41] Ben Leong, Sayan Mitra, and Barbara Liskov. Path vector face routing: Geographic routing with local face information. In *ICNP '05: Proceedings of the 13TH IEEE International Conference on Network Protocols (ICNP'05)*, pages 147–158, Washington, DC, USA, 2005. IEEE Computer Society.

[42] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the sixth annual international conference on Mobile computing and networking*, pages 120–130. ACM Press, 2000.

[43] Xiang-Yang Li, Gruia Calinescu, and Peng-Jun Wan. Distributed construction of planar spanner and routing for ad hoc wireless networks. In *IEEE INFOCOM*, pages 1268–1277, 2002.

[44] Xiang-Yang Li, Gruia Calinescu, Peng-Jun Wan, and Yu Wang. Localized delaunay triangulation with application in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(10):1035–1047, 2003.

[45] Xiang-Yang Li, Ivan Stojmenovic, and Yu Wang. Partial delaunay triangulation and degree limited localized bluetooth scatternet formation. *IEEE Transactions on Parallel and Distributed Systems*, 15(4):350–361, 2004.

[46] Kevin M. Lillis, Sriram V. Pemmaraju, and Imrana. Pirwani. Topology control and geographic routing in realistic wireless networks. In *Evangelos Kranakis and Jaroslav Opatrny, editors, ADHOC-NOW, volume 4686 of Lecture Notes in Computer Science*, pages 15–31. Springer, 2007.

[47] X. Lin and I. Stojmenović. Geographic distance routing in ad hoc wireless networks. Technical Report TR-98-10, SITE, University of Ottawa, December 1998.

[48] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network Magazine*, 15(6):30–39, 2001.

[49] Shree Murthy and J. J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mobile Networks and Applications*, 1(2):183–197, 1996.

[50] Asis Nasipuri, Robert Castañeda, and Samir R. Das. Performance of multipath routing for on-demand protocols in mobile ad hoc networks. *Mobile Networks and Applications*, 6(4):339–349, 2001.

[51] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of INFOCOM'97*, pages 1405–1413, April 1997.

[52] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proceedings of the conference*

on Communications architectures, protocols and applications, pages 234–244. ACM Press, 1994.

[53] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of WMCSA '99, Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[54] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 32–43. ACM Press, 2000.

[55] Elizabeth M. Royer and Chai-Keong Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, 6(2):46–55, 1999.

[56] Andreas Savvides, Mani Srivastava, Lewis Girod, and Deborah Estrin. Localization in sensor networks. *Wireless sensor networks*, pages 327–349, 2004.

[57] Karim Seada, Ahmed Helmy, and Ramesh Govindan. On the effect of localization errors on geographic face routing in sensor networks. In *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 71–80, New York, NY, USA, 2004. ACM.

[58] Adam Smith, Hari Balakrishnan, Michel Goraczko, and Nissanka Priyantha. Tracking moving devices with the cricket location system. In *MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 190–202. ACM Press, 2004.

[59] I. Stojmenović. Home agent based location update and destination search schemes in ad hoc wireless networks. Technical Report TR-99-10, Computer Science, SITE, University of Ottawa, September 1999.

[60] Ivan Stojmenović. Position based routing in ad hoc networks. *IEEE Commmunications Magazine*, 40(7):128–134, July 2002.

[61] Godfried T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.

[62] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, second edition, 2001.

[63] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Comput. Netw.*, 52(12):2292–2330, 2008.

Index