PROGRESS ROUTING: REMOVING BEACON PACKETS IN GEOGRAPHIC ROUTING PROTOCOLS

by

Ed Krohne

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematical and Computer Sciences).

Golden, Colorado

Date _____

Signed: _____
Ed Krohne

Approved: _____
Dr. Tracy Camp
Professor of Mathematical and Computer Sciences
Thesis Advisor

Golden, Colorado

Date _____

_____
Dr. Dinesh Mehta
Professor and Interim Head
Department of Mathematical and Computer Sciences

ABSTRACT

When location information is present in a mobile ad hoc network (MANET), it is possible to use *geographic routing protocols*; these protocols have the benefit of being scalable and efficient. Geographic routing protocols require a next hop neighbor to be selected at each hop based on the relative locations of the nodes in the area. Traditionally, to accomplish the selection, these protocols use proactive topology information exchange via periodic "beacon" or "hello" packets, resulting in unnecessary overhead and stale location information. Other proposals include reactive neighbor topology information exchange at each hop and contention strategies based on heuristic timers. These techniques can still lead to extra overhead and, in some cases, broadcast storms.

This thesis presents Progress Routing, a general technique capable of emulating the decisions of a variety of geographic routing protocols by encapsulating information about those decisions into a progress indicator function and a heuristic timing function. This division allows us to eliminate overhead required to maintain complete neighbor information, overcome the problems of stale neighbor knowledge, and prevent broadcast storms in these protocols. We illustrate the effectiveness of this concept by developing two pairs of progress indicator functions and heuristic timing functions that approximate the well-known Greedy Perimeter Stateless Routing (GPSR) protocol, and then comparing the performance of GPSR against that of Progress Routing.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

A **mobile ad hoc network (MANET)** is a set of wireless mobile devices that cooperatively form a network without established infrastructure such as wireless access points or cables. Sometimes a location system such as the Global Positioning System (GPS) allows nodes to know where they are located relative to each other; this location information is extremely useful for routing decisions in MANETs. Also, location information makes it meaningful to route packets towards a location – either the known location of a node or a specific geographic area for which the packet is relevant.

We consider a subset of MANET routing protocols that we refer to as **geographic routing protocols**. Well-known geographic routing protocols include **FACE-2** [1], **Greedy-Face-Greedy (GFG)** [3], **Greedy Perimeter Stateless Routing (GPSR)** [10], and **Greedy Other Adaptive Face Routing Plus (GOAFR+)** [13, 15]. In these protocols, each node specifically selects one (or a small number) of its neighbors to forward each packet it receives. A variety of criteria are used for this approach; most protocols combine some variant of greedy forwarding (where each node selects the closest neighbor node to the destination to forward a packet) and face traversal methods. An example of a face traversal method is the **right hand rule** strategy that can be used to exit a maze (or network) without wandering in circles: one places one's right hand on a wall and follows that wall to the exit.

One general shortcoming of this next forwarder selection process is that each node must constantly know the locations of all of its neighbors. This information is typically provided by periodic beacons that every node transmits proactively. Thus, geographic routing protocols may use stale information and cause network traffic even

in areas of the network that are not sending data. Both of these factors degrade performance. Alternatively, it is possible for each node to request some subset of its neighbors to reply with their locations at every hop, allowing the node to base its forwarding choice upon current information. Unfortunately, this alternative requires a significant number of transmissions at every hop. Protocols that utilize this strategy include **Contention-Based Forwarding (CBF)** [4], **Beaconless Forwarder Planarization (BFP)** [8], **Geographic Random Forwarding (GeRaF)** [25], **Implicit Geographic Forwarding (IGF)** [5], and **Beacon-less On Demand Strategy for Geographic Routing in Wireless Sensor Networks (BOSS)** [19].

Several variations on a technique that uses **heuristic timing functions** have been proposed in the past several years; the goal of this technique is to reduce overhead by allowing the neighbors at each hop to decide among themselves which is the best forwarder. Examples of such protocols include **Beacon-Less Routing (BLR)** [6] and **Blind Geographic Routing (BGR)** [23]. Generally, each candidate neighbor at each hop sets a timer to transmit. Each timer is set based on the candidate's location relative to the transmitting node; a "better" node should set a shorter timer than a "worse" node, and "worse" nodes cancel their transmissions when they overhear "better" nodes. This heuristic timer technique can be extremely efficient theoretically; however, in simulation, such protocols often duplicate packets in situations where one neighbor fails to receive another neighbor's transmissions [19].

To correct this problem, we present the concept of **progress indicator functions**. Progress indicator functions allow nodes to compare different copies of a packet and determine which packets have made or failed to make progress, greatly reducing the impact of duplications. Furthermore, a progress indicator function and a heuristic timing function together can encapsulate *all* of the behavior of a particular geographic routing algorithm such as GPSR. Representing a geographic routing algorithm as a heuristic timing function and a progress indicator function allows us to consider the problem of which nodes should transmit (the progress indicator func-

tion/heuristic timing function) separately from the problem of how to get the right nodes to transmit (the progress routing algorithm), and to make improvements to each solution independently. By substituting a new progress indicator function and heuristic timing function, the progress routing algorithm can be made to emulate the decisions made by a different geographic routing protocol. Existing beacon-less geographic protocols lack this clear division of layers that makes Progress Routing easy to extend.

Progress Routing also yields an additional benefit, which we mention because of the attention that this problem has received in the literature. Geographic routing protocols that rely on face traversal methods, such as FACE-2, GFG, GPSR, and GOAFR+, "guarantee" delivery only when the network can be modeled by a **unit disk graph**[1]. That is, each node in the network must have a perfect circular transmission area of unit radius. In reality, as discussed in [20, 21], delivery can fail due to obstacles, errors in a node's measured location, and non-uniform transmission ranges. Numerous approaches have been proposed to deal with realistic networks, such as the **Cross Link-Detection Protocol (CLDP)** [11], a technique based on hyperbolic space [12], **Greedy Echo Routing (GEcho)** [14], and **Greedy Distributed Spanning Tree Routing (GDSTR)** [17]. Progress Routing gives rise to a natural "local flooding" property (see Chapter 6) that makes face traversal methods unnecessary; thus Progress Routing is suitable for realistic networks.

We organize the rest of this thesis as follows. Chapter 2 defines the problem and establishes some basic terminology and notational conventions. One of the most well-known geographic routing protocols, GPSR, which is used in our simulations and in our examples, is presented in detail in Chapter 3. Chapter 4 presents and defines progress indicator functions as well as **designated paths**, a theoretical abstraction used to arrive at progress indicator functions. Chapter 5 then describes the progress routing algorithm, which uses progress indicator functions. Chapter 6 gives two

[1] Of course, these "delivery guarantees" also assume that packets are never dropped; the question of what to do if a packet *is* dropped is rarely addressed in the descriptions of these protocols.

example progress indicator functions that we developed to demonstrate the usefulness of our technique. In Chapter 7, we present simulation results comparing Progress Routing and our progress indicator functions with the geographic routing protocol our sample progress indicator functions most resemble (i.e., GPSR). Finally, Chapter 8 concludes this thesis.

Chapter 2

PROBLEM DEFINITION AND CONVENTIONS

Throughout this work, we examine the problem where some device in a MANET, called the source node, desires to route a single packet to another device (it does not matter which device) within a specific, known destination region (referred to as a geocast region in some papers). We assume that each packet transmitted contains identifying information about the destination region. Once the destination region has been reached, a broadcast within the whole destination region (or some other action dependent on the specific application) is taken. We consider the action after the packet has been delivered to the destination region to be application-specific and outside the scope of this work.

One example of an application for this problem is in a vehicular network designed to relay information about highway congestion and accidents; for example a device in a vehicle on a backed up stretch of highway may need to forward a message about the road conditions back to the vicinity of a previous highway entrance. In another application, autonomous robotic vehicles with low-power radios may need to relay data back to a central, fixed device with a known location. In this case, the destination region consists of a single point.

We use the term **routing scenario** to refer to a set of circumstances under which a node needs to route a single packet. A routing scenario consists of a source node $\vec{s}$ in an undirected graph N and a destination region $\mathbf{d}$. Nodes in N represent the devices in the network at the time node $\vec{s}$ transmits a packet, and two nodes are adjacent in N if they can hear each other directly using their radios. A node adjacent to some node in N is called a **neighbor** of that node. The set of all neighbors of some node $\vec{n}$ is denoted $\operatorname{nbr}\vec{n}$; likewise, the set of neighbors that $\vec{n}$ is *aware of* is denoted

knbr $\vec{n}$. It may be the case that knbr $\vec{n} \neq$ nbr $\vec{n}$ if node $\vec{n}$'s neighbor information is not up-to-date. Since a successful packet reaches the destination in milliseconds at most, we assume N to be approximately static during the lifetime of any one packet; obviously N can change in between packet transmissions.

In this work, we refer to nodes using lowercase letters with vector arrows above them, e.g., $\vec{a}$, $\vec{n}$, etc. We use the term **path** to mean a finite-length sequence of not necessarily unique nodes in the network. One path is a **subpath** of another if it appears within the other, in order and consecutively. For example, path $(\vec{a}, \vec{b})$ and path $(\vec{a}, \vec{b}, \vec{c})$ are both subpaths of path $(\vec{a}, \vec{b}, \vec{c})$, but path $(\vec{a}, \vec{c})$ is not. We represent paths with capital letters, usually P or D. Each path is indexable starting at 1, therefore P_1 is the first node in P, etc. We also use the notation such as P_{end} or P_{end-1} to refer to, e.g., the end node of a path or the second-to-last node in a path respectively. The binary plus (as in $A + B$) symbol denotes path concatenation. We represent sets of paths with bold capital letters (e.g., $\mathbf{D}$) and regions with bold lowercase letters (e.g., $\mathbf{d}$).

In some routing algorithms, a packet can sometimes be forwarded by more than one node at a particular hop. Therefore, we say that there can be multiple **copies of the packet** in the network at a time. Two different copies of the same packet are differentiated by which paths they have taken. Correspondingly, each copy of a packet is a copy of some **unique** or **original packet**; we use these two terms interchangeably. All copies of a unique packet share the same unique identifier and payload.

Chapter 3

GPSR

In GPSR (Greedy Perimeter Stateless Routing) [10], each node that receives a packet uses the relative locations of its neighbors to select a specific neighbor to forward the packet next. Each node knows the relative locations of its neighbors through a proactive, periodic beaconing system. Since only one node forwards a packet at any hop, GPSR uses very few transmissions to route a packet from its source to its destination. The main overhead in GPSR is its beaconing system.

GPSR has two modes: greedy and perimeter. Neither mode is sufficient, by itself, to route packets reliably, but the combination of the two modes is sufficient. In greedy mode, each node forwards a packet to its neighbor that is closest to the destination. For example, in Figure 3.1, node $\vec{s}$ has a packet that it wishes to send to region **d**. Nodes $\vec{s}$ and $\vec{n}_1$ each forward the packet directly to their closest neighbor to the destination, i.e., node $\vec{n}_1$ and node $\vec{n}_2$ respectively.

Unfortunately, greedy mode can fail at a node that is closer to the destination than any of its neighbors, such as node $\vec{n}_2$ in Figure 3.1. A node that is closer to the destination than any of its neighbors is called a **local minimum** node. Each local minimum node that a packet visits switches the packet to perimeter mode. In perimeter mode, nodes forward using the right hand rule; each node selects the next neighbor along the "right hand wall", shown in Figure 3.1. When the packet reaches a node that is closer to the destination than the packet was when it entered perimeter mode (e.g., node $\vec{n}_8$ is closer to $\bar{\mathbf{d}}$ than node $\vec{n}_2$ in Figure 3.1), then the obstacle is assumed to be routed around. In this way, every time a packet is transmitted in greedy mode, it is closer to the destination than the last time it was transmitted in greedy mode; eventually, if a path exists from the source $\vec{s}$ to the destination **d**, the

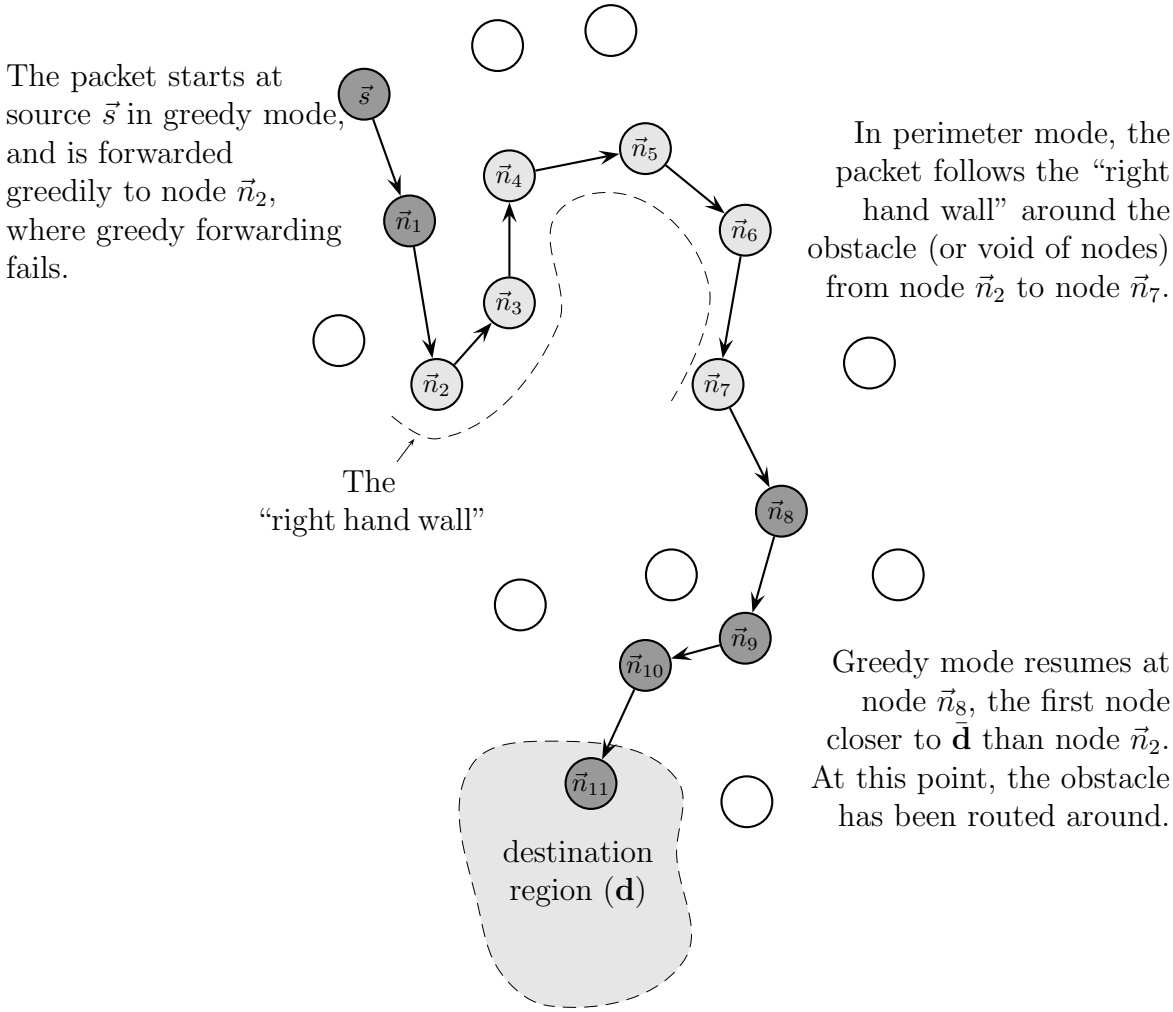


Figure 3.1. A full example of the operation of GPSR. In greedy mode, each node sends the packet to its closest neighbor to the destination. In perimeter mode, each node chooses the neighbor that minimizes the angle measured counterclockwise from either the destination or the previous hop, whichever is appropriate. Throughout this thesis, dark shaded colored nodes are nodes that transmit in greedy mode, light shaded nodes are nodes that transmit in perimeter mode, and unshaded nodes are nodes that do not transmit.

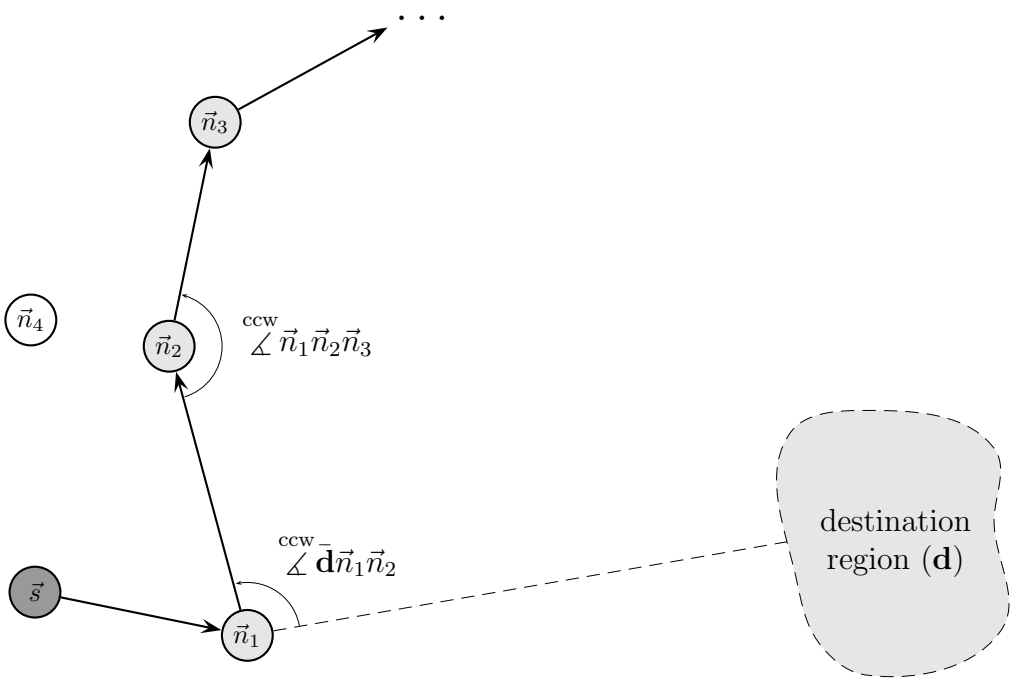


Figure 3.2. How the right hand rule is followed. The first node to transmit in perimeter mode (node $\vec{n}_1$) "finds" the right hand wall by selecting the neighbor that minimizes the angle measured counterclockwise from the center of the destination region. In this case, node $\vec{n}_1$ chooses node $\vec{n}_2$ with angle $\overset{\text{ccw}}{\measuredangle}\bar{\mathbf{d}}\vec{n}_1\vec{n}_2$. The remaining nodes that transmit in perimeter mode each "follow" the right hand wall by selecting a neighbor that minimizes the angle measured counterclockwise from the previous hop. For node $\vec{n}_2$, this minimum angle node is $\vec{n}_3$ with angle $\overset{\text{ccw}}{\measuredangle}\vec{n}_1\vec{n}_2\vec{n}_3$.

packet will reach $\mathbf{d}$.

To apply the right hand rule to MANETs, consider the links in the MANET as hallways in a maze. For example, in Figure 3.2, a packet arrives at node $\vec{n}_1$ from node $\vec{s}$. Node $\vec{n}_1$, being a local minimum node, must initiate perimeter mode. To initiate perimeter mode, it selects a next hop that minimizes the angle measured counterclockwise from the center of the destination. That is, it selects a neighbor $\vec{n'}$ that minimizes $\overset{\text{ccw}}{\measuredangle}\bar{\mathbf{d}}\vec{n}_1\vec{n'}$ where $\overset{\text{ccw}}{\measuredangle}$ stands for **counterclockwise angle**. Selecting a next hop this way ensures that the next hop is the next node along the right hand wall.

All remaining perimeter nodes follow this established right hand wall by selecting next hop neighbors that minimize the counterclockwise angle from the *previous hop* instead of the destination. That is, each node $\vec{n}_i$ along the right hand wall selects a neighbor $\vec{n'}$ that minimizes $\overset{\text{ccw}}{\measuredangle}\vec{n}_{i-1}\vec{n}_i\vec{n'}$. In the case of node $\vec{n}_2$ in Figure 3.2, this minimizing neighbor is node $\vec{n}_3$. In the selection of the next node along the right hand wall, the previous hop node is chosen only if no other neighbor is available.

Like greedy mode, perimeter mode can also fail given certain arrangements of nodes where links cross. To avoid these situations, GPSR, like many geographic routing protocols, uses a **graph planarization system** to remove crossing links. The original graph planarization system GPSR used, and which many other algorithms share, assumes an idealized network modeled by a unit disk graph. Other techniques such as CLDP [11] can improve the quality of the planarization at the expense of greater overhead. Our implementation of Progress Routing does not attempt to planarize the network (partially because Progress Routing achieves excellent delivery without it); therefore, we do not explain graph planarization here. We do, however, include the original graph planarization system in our simulation of GPSR (see Chapter 7). See [10] and [11] for details on graph planarization.

Chapter 4

PROGRESS INDICATOR FUNCTIONS

In Progress Routing, a given geographic routing algorithm is divided into a progress indicator function and a heuristic timing function. The progress indicator function establishes which nodes should transmit in a given scenario, and the heuristic timing function helps reduce duplicate packets. To use with these two functions, we developed the progress routing algorithm that ensures the right nodes always transmit.

Each node decides whether to forward a particular packet by determining whether the packet would make progress towards the destination if it did. Because a node does not necessarily have information about its neighbors, the node makes this determination by comparing progress indicators, which are small "hints" included in packet headers that sometimes indicate whether a different copy of the same packet has already made progress.

Progress indicators are based on an abstraction of geographic routing algorithms called a set of designated paths. A set of designated paths represents an ideal, efficient pattern of transmissions for a particular routing scenario if each node had perfect knowledge of its neighbors (e.g., "transmit greedily if possible, or follow the right hand wall if not possible" for GPSR). With full neighbor information, it is easy to make sure that a packet follows the designated paths to the destination. Without full neighbor information, however, it is not always possible to know what the designated paths are; thus, progress indicators are needed.

In order to make Progress Routing emulate a particular geographic routing algorithm, one would first reduce the geographic routing algorithm to a set of designated paths for each routing scenario. That is, one would determine what the pattern of

transmissions would be if each node had perfect neighbor knowledge. Then, one would determine a progress indicator function that matches the designated paths as closely as possible. Finally, one would develop a heuristic timing function that favors nodes more likely to continue the designated paths. Chapter 6 walks through the process of making Progress Routing emulate GPSR. Since most geographic routing protocols have, like GPSR, a greedy and/or perimeter component, the progress indicator function and heuristic timing function presented in Chapter 6 can be readily modified to match most other geographic routing protocols. Finally, one would execute the progress routing algorithm presented in Chapter 5 using the functions that have been determined.

The rest of this chapter is divided into two sections. Section 4.1 motivates and defines designated paths. Section 4.2 then illustrates how nodes can use progress indicators to ensure that packets follow the designated paths without full neighbor information.

4.1 Designated Paths

Before we discuss progress indicators, we first need to answer the question, "What is progress?" Specifically, "How can a packet make more progress, and when has one copy of a packet made more progress than another?" This question is important in all geographic routing; in fact, we could consider each geographic routing protocol to be a different answer to this question.

We answer this question using the designated paths; this concept is formally defined in Definition 1. The name "designated paths" is based on the observation that in traditional (fully-neighbor-aware) geographic routing algorithms, each node that receives a packet *designates* one or more of its (known) neighbors to forward the packet next. Geographic routing algorithms are distinguished primarily by which nodes are designated, and when they are designated. Definition 1 allows us to rephrase this distinction as "Which paths are designated, and when does one designated path

grant more progress than another designated path?" In this formulation, progress emerges as a strict weak ordering (called the progress relation) on the designated paths.

We consider each fully-neighbor-aware geographic routing protocol to be a mapping from routing scenarios to sets of designated paths; each set of designated paths represents the behavior of the routing algorithm in that scenario. Specifically, each designated path for a particular routing scenario is a path that a copy of the packet can take to get to a node it can visit in that scenario. Furthermore, if we can intuitively say that a copy of a packet that travels over one designated path makes more progress than a copy of a packet that travels over a different designated path, then the first designated path gives more progress than the second. This "gives more progress than" concept is the progress relation. Definition 1 states the general properties that a set of designated paths must have.

Definition 1 *Given a routing scenario consisting of a network N, a source node $\vec{s} \in N$, and a destination region* $\mathbf{d}$*, a set of paths* $\mathbf{D}$ *with a strict weak order $\rhd$ is called a* ***set of designated paths*** *for this scenario, and the strict weak order is called a* ***progress relation*** *for* $\mathbf{D}$ *if, and only if:*

1. *each path in* $\mathbf{D}$ *starts at the source node and is a possible path for a packet to take in N (i.e., each pair of consecutive nodes in the path is a pair of neighbors in N),*
2. *the path consisting of just the source node is in* $\mathbf{D}$*,*
3. *for each path*[1] *D in* $\mathbf{D}$ *that does not reach the destination, at least one next node exists in the network which, when appended to D, yields a path D' in* $\mathbf{D}$ *where $D' \rhd D$, and*
4. *there is a path in* $\mathbf{D}$ *that reaches the destination.*

Properties 1 and 2 of Definition 1 reflect the fact that in any geographic routing algorithm (indeed, any routing at all), a packet starts at the source node and only

[1]We follow the convention of naming paths that are definitely designated paths as D, D', etc. and paths that may or may not be designated paths as P, P', etc.

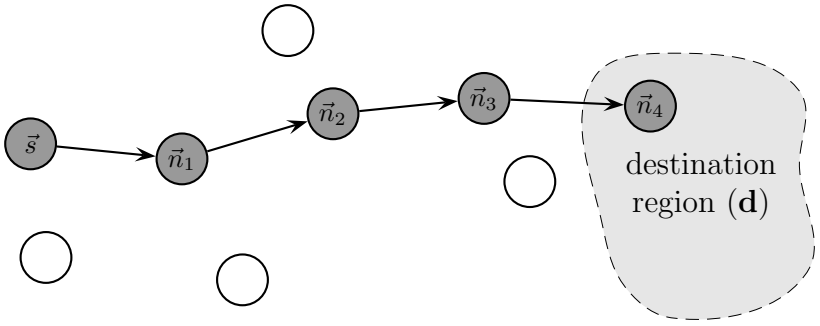


Figure 4.1. An example of designated paths. GPSR's designated paths for this scenario are $(\vec{s})$, $(\vec{s}, \vec{n}_1)$, $(\vec{s}, \vec{n}_1, \vec{n}_2)$, etc.

travels along links in the network. Property 3 captures the fact that each node that receives the packet always has a neighbor to forward the packet to next. When a node forwards the packet to this next hop neighbor, progress has been made according to the routing protocol, hence the relation $D' \rhd D$ in Property 3. Finally, because all designated paths must be finite-length in order to be considered a path (see Chapter 2) and thus there can be only a finite number of designated paths in any network, Property 4 requires that any geographic routing algorithm using these designated paths will *eventually* find the destination as long as the packet continues to make progress.

Figure 4.1 helps illustrate the relationship between routing protocols, designated paths, and progress relations. In the network shown, GPSR would forward the packet along the path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_3, \vec{n}_4)$. The designated paths are those paths that a copy of the packet may take to nodes it can visit toward the destination and are, in this scenario, therefore $(\vec{s})$, $(\vec{s}, \vec{n}_1)$, $(\vec{s}, \vec{n}_1, \vec{n}_2)$, etc. (for a total of five designated paths.). It is clear that a packet that is at $\vec{n}_2$ and has traveled along path $(\vec{s}, \vec{n}_1, \vec{n}_2)$ has made more progress than a packet that is at $\vec{n}_1$ and has traveled along path $(\vec{s}, \vec{n}_1)$. Thus, $(\vec{s}, \vec{n}_1, \vec{n}_2) \rhd (\vec{s}, \vec{n}_1)$.

It is tempting to think of "progress" as being a property of a node, i.e., to say

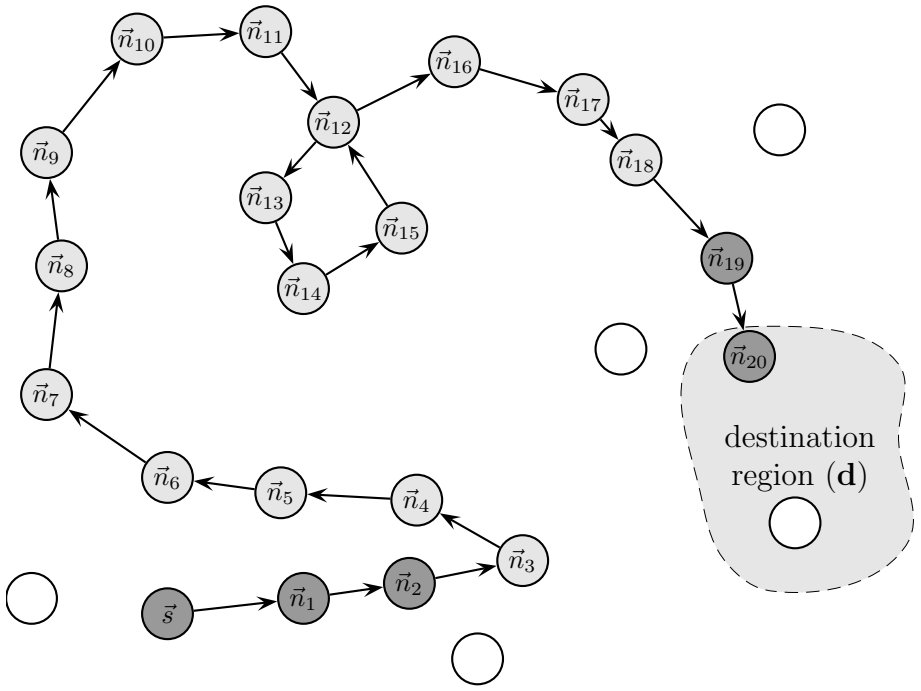


Figure 4.2. An example in GPSR illustrating the case where a node transmits more than once. In this case, when the packet is at node $\vec{n}_{12}$ and it has been around the loop consisting of nodes $\vec{n}_{13}$, $\vec{n}_{14}$, and $\vec{n}_{15}$, the packet has made more progress than when it was at $\vec{n}_{12}$ and had not been around the loop..

that a packet has made progress because it has reached a particular node; however, the progress a packet has made is determined not only by the node a packet is at, but by the path it has taken. For example, in Figure 4.2, GPSR calls for node $\vec{n}_{12}$ to transmit twice, once in reply to node $\vec{n}_{11}$ and once in reply to $\vec{n}_{15}$. The second time node $\vec{n}_{12}$ transmits the copy of the packet, the copy has made more progress, even though it is in the same location as before (i.e., at node $\vec{n}_{12}$). This extra progress exists because the packet has explored more of the right hand wall. Correspondingly, $\vec{n}_{12}$ forwards the packet to a different node than before (i.e., node $\vec{n}_{16}$ instead of node $\vec{n}_{13}$).

4.2 Progress Indicators

The fundamental challenge of geographic routing without beacon packets is to get packets to follow the designated paths defined by a given routing protocol, and to ensure that progress increases with time; this is tricky because nodes may not know where the designated paths are. For example, given the network shown in Figure 4.3, assume that node $\vec{n}_2$ does not know that node $\vec{n}_3$ exists (since node $\vec{n}_2$ and node $\vec{n}_3$ do not transmit beacons), and therefore forwards the packet to node $\vec{n}_7$. In this case, the packet has not made progress according to GPSR because the packet has not followed the right hand rule; in fact, the packet is no longer on a designated path. Continuing to route the packet from that point forward results in a routing loop, because the packet's concept of the "right hand wall" has erroneously changed.

While it is impossible to store information in the packet that answers the question, "how much progress has the copy of the packet made?" (as we see from Figure 4.3), it *is* possible to store information in the packet that sometimes answers the question, "is this copy of the packet erroneous or redundant?" The piece of information that sometimes answers the latter question is a progress indicator, named as such because it can indicate that progress has been (or will be) made. Progress indicators lend themselves to a simple routing strategy: whenever a node receives a copy of the

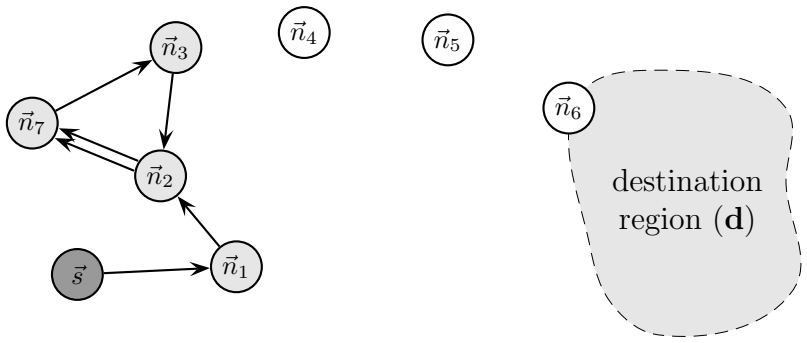


Figure 4.3. An example showing that ignorance of neighbors can prevent nodes from knowing whether a packet has made progress. In this scenario, we assume that when node $\vec{n}_2$ receives the packet, it is ignorant of node $\vec{n}_3$. Thus, it forwards the packet to node $\vec{n}_7$, causing the packet to take a non-designated path and not make progress. Furthermore, a routing loop results.

packet, if the node can conclude (using progress indicators) that either it received the packet in error, or another copy of the packet has made or will make more progress than the copy it just received, then the node does not transmit. If the node can not make such a conclusion, it rebroadcasts the packet to all of its neighbors, thus ensuring that if the packet *was not* received erroneously and more progress *was not and will not* already be made, then the node progresses the packet itself.

By including one or more progress indicators in each packet header, nodes can disseminate information about the network in an abstracted, lightweight form, and thus make intelligent decisions about when to transmit and when not to transmit. Indeed, this information can be powerful enough that very few redundant nodes need to transmit in order to ensure progress is made, even though nodes do not always know if a packet has taken a designated path.

We extract progress indicators from the paths that each packet has taken or could take. That is, progress indicator is a "summary" of a path and contains just enough information about that path to usually indicate when progress has been made. This summarizing is done by a function called a progress indicator function. The specific

quality that enables progress indicators to indicate progress is given by Property 1 of Definition 2.

Definition 2 *A function $\mathcal{F}$ that maps paths (typically labeled D or P) and destination regions (typically labeled $\mathbf{d}$) into some totally ordered set is called a* ***progress indicator function*** *and the totally ordered set is called a set of* ***progress indicators*** *if, and only if, for every routing scenario, there exists a set of designated paths $\mathbf{D}$ which has the following properties:*

1. *for every pair of paths, P and P', that start at the same node and end at the same node, if $\mathcal{F}(P, \mathbf{d}) < \mathcal{F}(P', \mathbf{d})$, then either*
   (a) *P is* not *a designated path (thus a copy of a packet that took P is erroneous), or*
   (b) *P is a designated path but P' has a designated subpath D with progress at least as high as P (thus, another copy of the packet has already made more progress than the copy of the packet that took P);*
2. *if $\mathcal{F}(D, \mathbf{d})$ for some designated path D is ∞, then D reaches the destination.*

Property 1 covers most of the behavior of the progress indicator function, while Property 2 simply reserves a special progress indicator for packets that reach the destination. For Property 1, assume a node $\vec{n}$ receives two copies of a packet, the first along some path P' and the second along some path P, and thus $P_{end} = P'_{end} = \vec{n}$. Furthermore, assume that node $\vec{n}$ has available the corresponding progress indicators, $\mathcal{F}(P, \mathbf{d})$ and $\mathcal{F}(P', \mathbf{d})$, and node $\vec{n}$ determines that $\mathcal{F}(P, \mathbf{d}) < \mathcal{F}(P', \mathbf{d})$.

In this situation, node $\vec{n}$ knows progress would not be made if it forwarded the copy of the packet it received along P. Because P and P' start and end at the same locations (namely, source $\vec{s}$ and node $\vec{n}$ respectively) and $\mathcal{F}(P, \mathbf{d}) < \mathcal{F}(P', \mathbf{d})$, then node $\vec{n}$ knows that either the copy of the packet received along path P is erroneous (because P is not a designated path) or another packet (specifically, the copy of

the packet that took path P') has made more progress. In either case, the fact that $\mathcal{F}(P, \mathbf{d}) < \mathcal{F}(P', \mathbf{d})$ means that there is no reason to retransmit the copy of the packet that took path P.

The stipulation in Property 1 that paths P and P' start and end at the same nodes serve to make it easier for us to find progress indicator functions without sacrificing the usefulness of progress indicators. The path along which some node $\vec{n}$ receives any copy of the packet will always start at the source node $\vec{s}$ and end at node $\vec{n}$ itself, thus any one node will only ever compare progress indicators of paths that start at the same node and end at the same node. Ensuring Property 1 for pairs of paths that do not start or end at the same nodes is, therefore, unnecessary and, for designated paths based on face traversal methods (such as following the right hand wall), prohibitively difficult and perhaps impossible.

To see how Property 1 can be used in a concrete example, consider Figure 4.4, which has the same arrangement of nodes as Figure 4.3. Instead of selecting node $\vec{n}_7$ to forward the packet, as in Figure 4.3, node $\vec{n}_2$ *suggests* (since node $\vec{n}_2$'s neighbor knowledge may not be perfect) that node $\vec{n}_7$ forward the packet. Unlike in traditional geographic routing protocols where node $\vec{n}_3$ would have to rely on node $\vec{n}_2$'s neighbor knowledge, node $\vec{n}_3$ can use progress indicators to decide for itself whether to forward the packet it received from node $\vec{n}_2$ or let node $\vec{n}_7$ transmit next.

Let D be the path that does not include node $\vec{n}_7$, i.e.,

$$D = (\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_3). \tag{4.1}$$

Furthermore, let P be the path that includes node $\vec{n}_7$, i.e.,

$$P = (\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_7, \vec{n}_3). \tag{4.2}$$

Assume that the copy of the packet included enough information for node $\vec{n}_3$ to compute a progress indicator for both path D and path P, i.e., node $\vec{n}_3$ has $\mathcal{F}(D, \mathbf{d})$

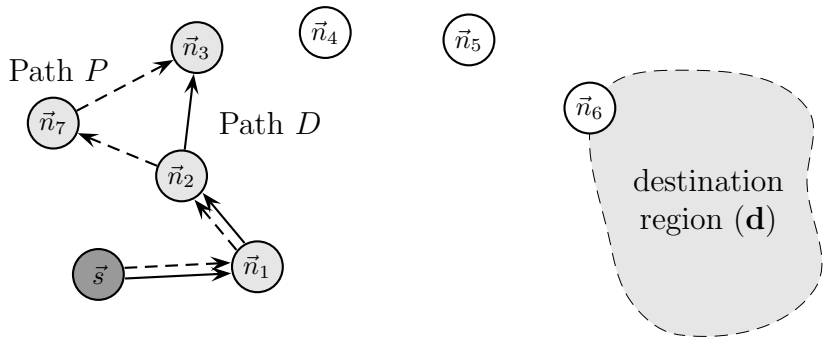


Figure 4.4. An example showing how progress indicators can overcome the problem of incomplete neighbor knowledge. Path P is shown with a dotted line, and path D is shown with a solid line.

and $\mathcal{F}(P, \mathbf{d})$ where $\mathcal{F}$ is a progress indicator function. We see that

1. because D is the path GPSR would have used (given complete neighbor knowledge), D is a designated path and P is not,

2. D and P each have the same start and end nodes, namely, node $\vec{s}$ and node $\vec{n}_3$, and

3. P has no designated subpath with more progress than D.

From these three observations and Property 1 of Definition 2, $\mathcal{F}(P, \mathbf{d})$ *can not* exceed $\mathcal{F}(D, \mathbf{d})$. In Progress Routing, any node transmits unless it knows not to. Thus, with any progress indicator function that matches GPSR (more than one exist), node $\vec{n}_3$ will always transmit the copy of the packet it received directly from node $\vec{n}_2$.

Alternatively, while it is not possible that $\mathcal{F}(D, \mathbf{d}) < \mathcal{F}(P, \mathbf{d})$ as previously stated, it *is* possible (depending on the definition of $\mathcal{F}$) that $\mathcal{F}(D, \mathbf{d}) > \mathcal{F}(P, \mathbf{d})$, because P is not a designated path. Assume it is the case that $\mathcal{F}(D, \mathbf{d}) > \mathcal{F}(P, \mathbf{d})$. In this case, node $\vec{n}_3$ knows that if node $\vec{n}_7$ transmitted next, then the resulting copy of the packet would be either erroneous or make less progress than if node $\vec{n}_3$ transmitted next. Thus, node $\vec{n}_3$ knows it should not wait for the copy of the packet from node $\vec{n}_7$, thus averting the routing loop.

Progress indicators can also be used to express ambiguity. That is, if two copies of a packet each take a different path to a node $\vec{n}$ where the paths have equal progress indicators, then $\vec{n}$ will forward *both* packets. This property is used in our greedy-only progress indicator presented in Section 6.2.

Of course, ambiguous progress indicators result in extra transmissions. Progress indicator functions become increasingly efficient as they become increasingly specific, or in other words, as they indicate more about their sets of designated paths by returning different progress indicators for different paths. A maximally efficient progress indicator function would be one that returned different progress indicators for as many pairs of paths as possible. One way to approach a maximally efficient progress indicator function, for example, would be to include full information about the path a packet has taken in the progress indicator. That is, define a progress indicator function $\mathcal{F}(P, \mathbf{d}) = P$. The ordering of any two progress indicators would then be based on the way the two paths diverged; however, such a bulky progress indicator would be overkill for most geographic routing algorithms. Instead, the progress indicator function we present in Chapter 6 (loosely corresponding to GPSR's designated paths) is constant-size regardless of the associated path length; furthermore, as shown in Chapter 7, few redundant transmissions are necessary with our proposed progress indicator function.

Chapter 5

THE PROGRESS ROUTING ALGORITHM

To demonstrate how progress indicator functions can be used, we have developed and implemented the progress routing algorithm. This algorithm uses progress indicators and heuristic timing functions as the basis for all of its forwarding decisions; we assume nothing about the pattern of transmissions desired.

To supply this information, we assume that the algorithm has two function calls available. The first function call updates a progress indicator in a packet with a new node in a path the packet has taken or could take. Specifically, given a progress indicator $\mathcal{F}(P, \mathbf{d})$ for some path P and a new node $\vec{n}$, the algorithm can compute the progress indicator for the concatenation path $P + (\vec{n})$, i.e., it can compute $\mathcal{F}(P + (\vec{n}), \mathbf{d})$.

The second function call is the heuristic timing function; it returns a delay value indicating how long the node should wait to transmit. This delay is calculated using the location information for the packet's last several hops. The delay should be smaller for nodes that should transmit than for nodes that should not transmit. For example, for greedy forwarding, nodes close to the destination should transmit before nodes farther away from the destination.

The progress routing algorithm is structured to maximize the probability that, at every hop, one neighbor transmits and all other neighbors do not transmit. In other words, ideally, exactly one neighbor receives the packet with a progress indicator higher than any it has seen before, and thus concludes it must transmit to continue making progress. All other neighbors are made aware that progress has been made or will be made without their help.

To achieve a number of forwarding nodes at each hop as close to one as possible,

the Progress Routing packet header contains either one or two progress indicators. The first progress indicator, which is always present, is called the **direct progress indicator (DPI)**. The direct progress indicator indicates how much progress the copy of the packet has actually made. Thus, the direct progress indicator allows a listening node to determine if the copy of the packet was sent in error, or, alternatively, if its own idea of how much progress the packet has made is out of date. The second progress indicator, which may not be present, is the **suggestion progress indicator (SPI)**. The suggestion progress indicator corresponds to a path the packet *might* take, as far as the transmitting node is aware. A high suggestion progress indicator allows a node that would otherwise forward the copy of the packet know that it does not need to, because a better node (the suggestion) exists. Thus, a node is only allowed to reply to a packet if that node would give the packet at least as much progress as the suggestion node. In order to make suggestions, each node keeps track of the most recent known locations of nodes it directly hears transmit.

Additionally, the progress routing algorithm uses two different packet types: `DATA` packets and `KILL` packets. `DATA` packets are regular packets and contain the data payload. When a node transmits a `DATA` packet, the node is claiming that it could be at the end of a designated path, and is requesting a neighbor to forward the packet. `KILL` packets, on the other hand, cannot be forwarded. `KILL` packets are used to give neighbors better progress indicators and to tell them, "do not retransmit." When a node transmits a `KILL` packet, the node is telling all of its neighbors that progress has already been guaranteed, and no transmission is necessary. Since `KILL` packets cannot be forwarded, they do not contain the data payload.

A `KILL` packet behaves very similarly to a `DATA` packet that has a **time to live (`TTL`)** of one. The only difference between these two packet types is that a `DATA` packet with $\texttt{TTL} = 1$ contains a data payload, and a `KILL` packet does not. The rationale behind this difference is that `KILL` packets are intended as "control" packets to be sent in tandem with other `DATA` packets; thus, it is not necessary to transmit the

payload in this packet type.

In an attempt to avoid collisions, each node maintains two separate transmit timers for each unique packet: one timer for `DATA` packets and one timer for `KILL` packets. Whenever a node decides to transmit, it sets the appropriate timer for a duration determined by the heuristic timing function. This way, not every neighbor transmits at once, and a neighbor can cancel its own planned transmission if another node has transmitted and made higher progress. If the node ever decides to forward a different copy of the packet when it has an existing timer set, then the node simply reschedules the appropriate (`DATA` or `KILL`) timer to the soonest of the new and existing timeouts.

In order to evaluate the progress made by a copy of a packet it receives, each node keeps track of the highest direct progress indicator the node has received for each unique packet (i.e., a table of progress indicators indexed by unique packet id). This progress indicator is called the **recorded progress indicator (RPI)** for that packet, which is $-\infty$ if the node has not previously received a copy of the packet. A node's recorded progress indicator serves as the node's idea of how much progress the unique packet has made overall. Every time the node transmits a packet, the node includes its recorded progress indicator in the packet as the packet's direct progress indicator; the direct progress indicator is then disseminated to as many nodes as possible.

As stated in Chapter 2, the goal of Progress Routing is to deliver a single packet into the destination region, but the specific action to be taken once this delivery happens is unspecified. For evaluation purposes in this work, when a packet is delivered to the destination region, one or more nodes in the destination region transmits a `KILL` packet with infinite progress; this `KILL` packet informs other nodes that delivery was successful and no more packets need to be sent.

A flowchart for the progress routing algorithm is given in Figure 5.1. When a node has a new, original packet it needs to route to the destination, (a situation

corresponding to Step 1), it initializes its own recorded progress indicator to be the progress indicator for the path consisting of just itself. This initialization is represented by Step 2. Then, in Step 3, the node schedules an immediate data transmit event.

Now, assume some node $\vec{n}_t$ has a `DATA` packet to transmit. This situation corresponds to Step 12 and can occur because the node is originating a packet (Steps 1-4) or because a node is forwarding a packet it has already received (Steps 5-7). Let P denote the path that node $\vec{n}_t$'s recorded progress indicator corresponds to; thus, node $\vec{n}_t$ is at the end of P.

In Step 13, node $\vec{n}_t$ includes its recorded progress indicator (i.e., $\mathcal{F}(P, \mathbf{d})$) in the packet, which becomes the packet's direct progress indicator. Then, in Step 14, the node checks to see if the packet it is about to send has already been to the destination, i.e., the node checks to see if the packet's direct progress indicator is ∞ (see Property 2 of Definition 2). If the packet has reached the destination, then in Step 15, node $\vec{n}_t$ turns the packet into a `KILL` packet. This step helps to prevent further future attempts to deliver the already-delivered packet. Note that in this circumstance, node $\vec{n}_t$ does not use a `KILL` timer because node $\vec{n}_t$ has already waited a delay to transmit this packet.

If the packet has not reached the destination, then node $\vec{n}_t$ attempts to include from among its known neighbors (i.e., $\text{knbr}\,\vec{n}_t$) a node $\vec{n}_s$ called the **suggested next node**, as well as the progress indicator that node $\vec{n}_s$ would use if it responded (Step 16). The progress indicator for the suggestion, i.e., $\mathcal{F}(P + (\vec{n}_s), \mathbf{d})$, becomes the packet's suggestion progress indicator. Of course, node $\vec{n}_s$ is merely a suggestion; it may be the case that node $\vec{n}_s$ is no longer present, or there exists a different, and better, neighbor that node $\vec{n}_t$ does not know exists. To ensure node $\vec{n}_s$ is a good suggestion (in the sense that it is likely to be at the end of a designated path), node $\vec{n}_t$ picks a node with the following two properties[1]: (1) the path to any other neighbor

[1]The simpler condition $\forall \vec{n'} \in \text{knbr}\,\vec{n}_t$, $\mathcal{F}(P + (\vec{n}_s), \mathbf{d}) \geq \mathcal{F}(P + (\vec{n'}), \mathbf{d})$ might appear to be a more straightforward way to ensure node $\vec{n}_s$ is a good suggestion, but the paths $P + (\vec{n}_s)$ and $P + (\vec{n'})$ do

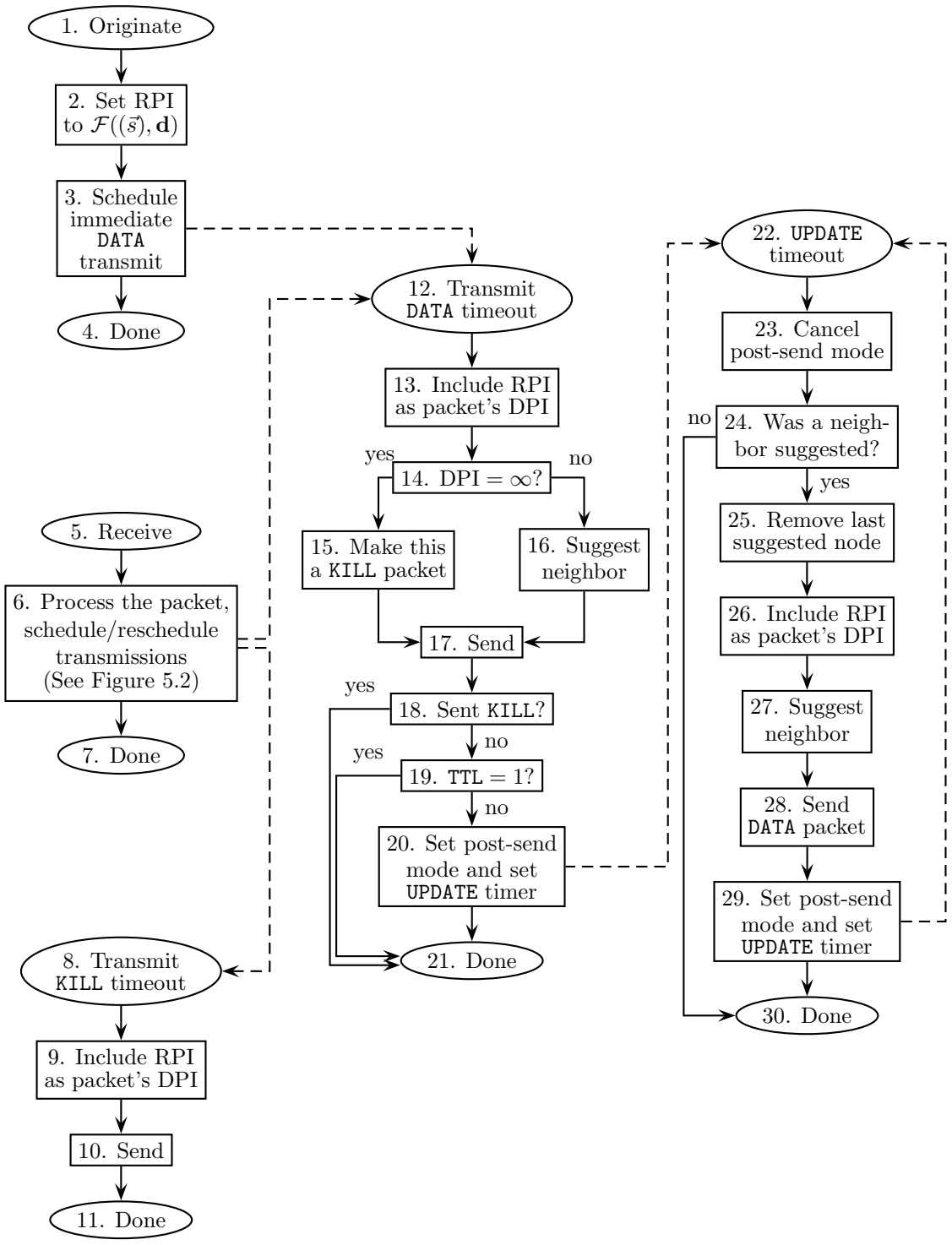


Figure 5.1. The progress routing algorithm. Circles represent events, boxes represent actions and decisions, solid arrows show flow control, and dashed arrows indicate timers that trigger later events.

with the highest progress indicator includes node $\vec{n}_s$, i.e.,

$$\forall \vec{n'} \in \text{knbr}\, \vec{n}_t, \quad \mathcal{F}(P + (\vec{n}_s, \vec{n'}), \mathbf{d}) \geq \mathcal{F}(P + (\vec{n'}), \mathbf{d}), \tag{5.1}$$

and (2) the highest progress indicator of all paths to node $\vec{n}_s$ is that of the path straight to node $\vec{n}_s$, i.e.,

$$\forall \vec{n'} \in \text{knbr}\, \vec{n}_t, \quad \mathcal{F}(P + (\vec{n}_s), \mathbf{d}) \geq \mathcal{F}(P + (\vec{n'}, \vec{n}_s), \mathbf{d}). \tag{5.2}$$

Next, in Step 17, node $\vec{n}_t$ sends the packet to the broadcast MAC address, allowing all of its neighbors to process it. After node $\vec{n}_t$ transmits, if it is expecting the copy of the packet it sent to be forwarded (i.e., it sent a `DATA` packet with a `TTL` > 1), the node enters **post-send mode** and sets a corresponding `UPDATE` timer. Post-send mode is needed in case node $\vec{n}_s$ does not exist or otherwise fails to reply, or if some node better than $\vec{n}_s$ transmits the packet before node $\vec{n}_s$ does. The check is performed in Steps 18 and 19, and the node enters post-send mode in Step 20.

Node $\vec{n}_t$ will be in post-send mode if and only if it has an `UPDATE` timer set. During the duration of the `UPDATE` timer, node $\vec{n}_t$ listens for its neighbors to transmit the packet and make progress. While $\vec{n}_t$ is in post-send mode, if a different neighbor from node $\vec{n}_s$ transmits with a better progress indicator than node $\vec{n}_t$ suggested, node $\vec{n}_t$ will inform its neighbors (neighbors who may not have heard this transmission) by immediately sending a `KILL` packet (see Figure 5.2 Step 8). If the `UPDATE` timer times out, represented by Step 22 in Figure 5.1, then node $\vec{n}_t$ cancels post-send mode in Step 23. Since the `UPDATE` timer timed out, then neither the suggested node nor any node with a higher progress path has forwarded the packet. In response to this fact, in Step 24, node $\vec{n}_t$ checks whether it suggested a node in the `DATA` packet previously sent. If node $\vec{n}_t$ did not suggest a node, then it concludes that no neighbor exists to progress the packet and gives up. If it did suggest a node, then it concludes that the

not end at the same node. Thus, comparing progress indicators for these two paths is not meaningful.

node it suggested, node $\vec{n}_s$, is no longer available and no node better than node $\vec{n}_s$ exists. Thus, in Step 25, node $\vec{n}_t$ removes node $\vec{n}_s$ from its neighbor table. Then, in Steps 26-29, node $\vec{n}_t$ tries to send the `DATA` packet again with a new suggestion.

To complete the discussion of Figure 5.1, Steps 8-11 show node $\vec{n}_t$'s actions when its `KILL` timer times out on a packet. These actions are similar to the `DATA` timeout actions in Steps 12-21, except that it is unnecessary to check to see if the packet has reached the destination, suggest a neighbor, or ensure the `KILL` packet gets forwarded by one of node $\vec{n}_t$'s neighbors.

Let us assume a node $\vec{n}_f$ hears a `DATA` or `KILL` packet transmitted by node $\vec{n}_t$. The steps node $\vec{n}_f$ takes are shown in Figure 5.2. In order to decide whether to forward the packet and, if so, when, node $\vec{n}_f$ must first perform three initial update actions, corresponding to Step 2 of Figure 5.2. First, node $\vec{n}_f$ adds node $\vec{n}_t$'s location to its neighbor table.

Second, if this is the first time $\vec{n}_f$ has received a copy of this unique packet, it records its current location for use in any subsequent calculations dealing with this unique packet. In our simulations we find we obtain higher delivery rates if each node pretends it is in the *exact* same location during the lifetime of a packet. We stated previously that we assume the network is approximately static during the lifetime of any one packet. This assumption is basically correct, as it is very unlikely for nodes to move in or out of range of each other during the lifetime of a single packet. The nodes, however, may move slightly during the lifetime of a packet, and progress indicators can be sensitive to small, unimportant changes in location.

Finally, node $\vec{n}_f$ updates the progress indicators in the packet it received by executing the progress indicator update function with its own location. This way, the progress indicators reflect node $\vec{n}_f$'s position relative to the other nodes in each path. To illustrate this process, Figure 5.3 shows the progress indicators available to node $\vec{n}_f$ before Step 2; node $\vec{n}_f$ has its own recorded progress indicator for the packet (which may be $-\infty$ if node $\vec{n}_f$ has not received a copy of this packet before), as well

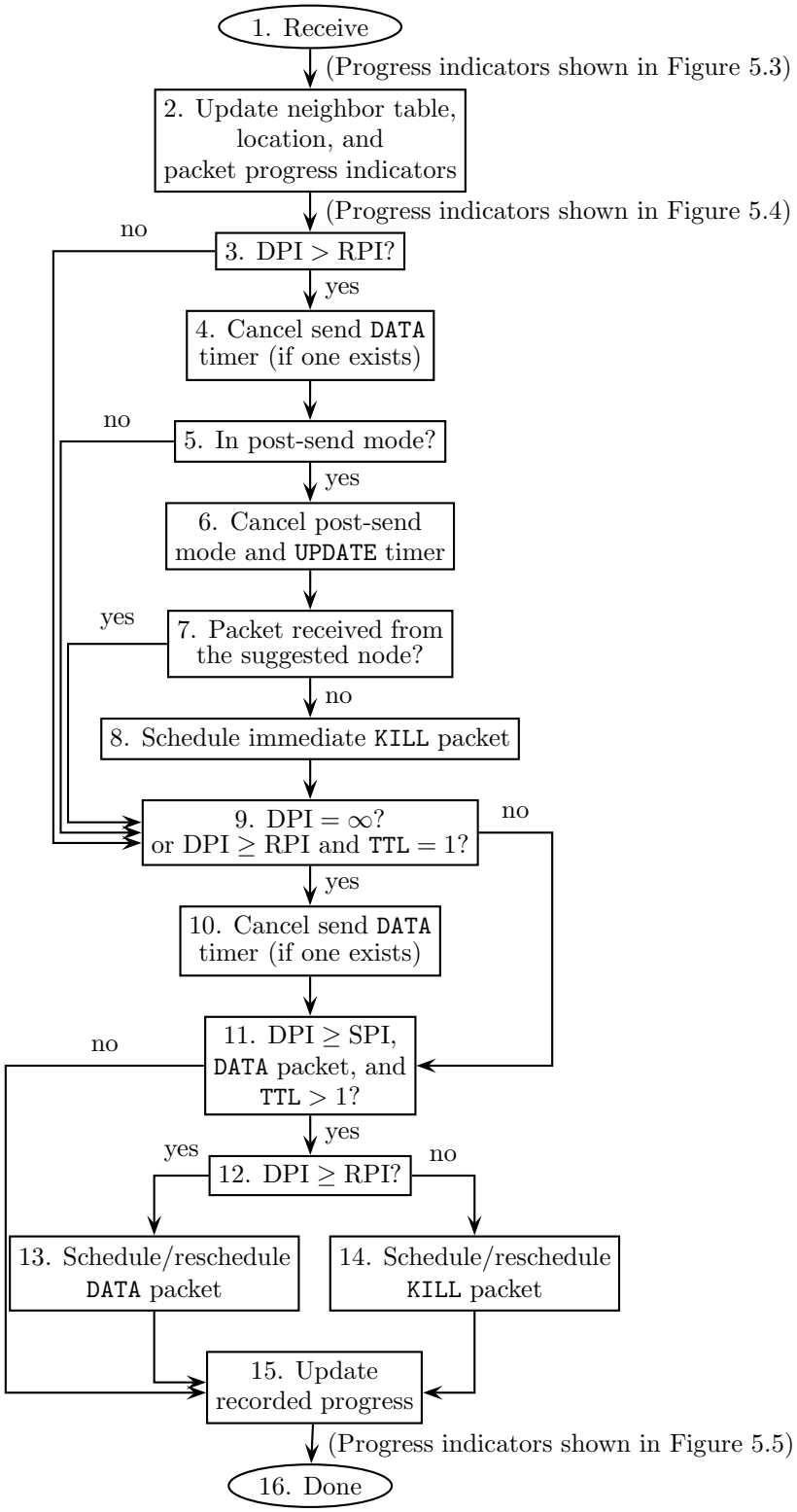


Figure 5.2. The progress routing algorithm "receive" event in detail. Circles represent events, boxes represent actions and decisions, and arrows show flow control.

Node $\vec{n}_f$'s Stored Progress Indicator

Recorded prog. ind.	$\mathcal{F}(R, \mathbf{d})$	$R = (\vec{s} \ldots \vec{n}_f)$

Packet Progress Indicators

Direct prog. ind.	$\mathcal{F}(P, \mathbf{d})$	$P = (\vec{s} \ldots \vec{n}_t)$
Suggestion prog. ind.	$\mathcal{F}(P + (\vec{n}_s), \mathbf{d})$	$P + (\vec{n}_s) = (\vec{s} \ldots \vec{n}_t, \vec{n}_s)$

Figure 5.3. A receiving node's progress indicators and the progress indicators in the packet that the node receives, before it performs any updates. In this figure and Figures 5.4 and 5.5, node $\vec{n}_f$ is the receiving node, and node $\vec{n}_t$ is the transmitting node.

Node $\vec{n}_f$'s Stored Progress Indicator

Recorded prog. ind.	$\mathcal{F}(R, \mathbf{d})$	$R = (\vec{s} \ldots \vec{n}_f)$

Packet Progress Indicators

Direct prog. ind.	$\mathcal{F}(P + (\vec{n}_f), \mathbf{d})$	$P + (\vec{n}_f) = (\vec{s} \ldots \vec{n}_t, \vec{n}_f)$
Suggestion prog. ind.	$\mathcal{F}(P + (\vec{n}_s, \vec{n}_f), \mathbf{d})$	$P + (\vec{n}_s, \vec{n}_f) = (\vec{s} \ldots \vec{n}_t, \vec{n}_s, \vec{n}_f)$

Figure 5.4. A receiving node's progress indicators and the progress indicators in the packet that the node receives, after it updates the packet. Node $\vec{n}_f$ updates the packet's progress indicators with its own location; thus, they reflect node $\vec{n}_f$'s location in relation to the paths taken. Note that the paths for all progress indicators now end with node $\vec{n}_f$.

as the direct progress indicator in the packet and (possibly) the suggestion progress indicator[2] in the packet. For convenience, in the rest of this chapter, we treat an absent suggestion progress indicator as $-\infty$.

The progress indicators after this update are shown in Figure 5.4. After the update, all progress indicators that node $\vec{n}_f$ has available (and, in fact, will ever use) are for paths that start at source $\vec{s}$ and end at node $\vec{n}_f$. Thus, the update step fulfills the requirement that paths must have the same end node in order for the corresponding progress indicators to be meaningfully compared (See Property 1 of Definition 2 in Section 4.2).

After updating the packet and its own internal state, node $\vec{n}_f$ checks whether the

[2]Note that the suggestion progress indicator is never recorded by node $\vec{n}_f$; the recorded progress indicator is always the highest *direct* progress indicator received.

direct progress indicator exceeds its own recorded progress indicator for this unique packet (Step 3 in Figure 5.2). If it does, then any copy of this packet that node $\vec{n}_f$ is planning to send is erroneous or, because more progress has already been made, redundant. Thus, node $\vec{n}_f$ can safely cancel the `DATA` timer associated with this packet (Step 4). Note that any `KILL` copy of this packet that node $\vec{n}_f$ is planning to send may still be necessary and node $\vec{n}_f$ does *not* cancel its `KILL` timer. If node $\vec{n}_f$ is in post-send mode for this packet (i.e., node $\vec{n}_f$ had recently sent the packet), then node $\vec{n}_f$ cancels the `UPDATE` timer and post-send mode for this packet (Step 6). If node $\vec{n}_f$ cancels its update timer, then it checks whether the packet came from the node it suggested. If not (which could easily be the case if node $\vec{n}_f$ did not suggest a node when it transmitted the packet), then it is likely that node $\vec{n}_f$ has neighbors with `DATA` timers set that are unaware that progress has been made. To inform these nodes of the progress that has been made, node $\vec{n}_f$ schedules a `KILL` packet to be transmitted immediately (Step 8).

Next, in Step 9, node $\vec{n}_f$ checks to see if either the direct progress indicator is infinity, or the `TTL` of this new packet is 1 and the direct progress indicator is not less than the recorded progress indicator. In the former case, the packet has already reached the destination. In the latter case, the packet has reached the end of its `TTL`; thus, the packet should not be forwarded further. In either case, if a `DATA` timer is set for this packet, node $\vec{n}_f$ cancels the timer (Step 10).

At this point, if node $\vec{n}_f$ cannot determine that node $\vec{n}_t$'s packet is erroneous or redundant, or that another of node $\vec{n}_t$'s neighbors will progress the packet, then $\vec{n}_f$ must transmit the packet itself. If, on the other hand, node $\vec{n}_f$ *can* conclude that node $\vec{n}_t$'s transmission was erroneous or redundant, then node $\vec{n}_f$ may need to send a `KILL` packet to stop node $\vec{n}_t$ from trying to route its packets. To reduce the number of redundant transmissions, if more than one of node $\vec{n}_t$'s neighbors reply, node $\vec{n}_f$ only sometimes replies with a `DATA` or `KILL` packet.

To see whether node $\vec{n}_f$ should transmit, node $\vec{n}_f$ checks to see if the direct

Node $\vec{n}_f$'s Stored Progress Indicators

Recorded prog. ind.	Highest of $\mathcal{F}(R, \mathbf{d})$ and $\mathcal{F}(P + (\vec{n}_f), \mathbf{d})$

Packet Progress Indicator

Direct prog. ind.	$\mathcal{F}(P + (\vec{n}_f), \mathbf{d})$
Suggestion prog. ind.	$\mathcal{F}(P + (\vec{n}_s, \vec{n}_f), \mathbf{d})$

$$P + (\vec{n}_f) = (\vec{s} \ldots \vec{n}_t, \vec{n}_f)$$
$$P + (\vec{n}_s, \vec{n}_f) = (\vec{s} \ldots \vec{n}_t, \vec{n}_s, \vec{n}_f)$$

Figure 5.5. A receiving node's progress indicators and the progress indicators of a packet that the node receives, after it has decided whether or not to transmit. The node's final action in handling this packet is to update its recorded progress indicator for this unique packet id.

progress indicator is greater than or equal to the suggested progress indicator (i.e., is DPI $\geq$ SPI?), the packet is a `DATA` packet, and the `TTL` is greater than one (Step 11). If node $\vec{n}_f$ should transmit, then it decides whether it should forward the `DATA` packet or send a `KILL` packet by comparing the packet's direct progress indicator to node $\vec{n}_f$'s own recorded progress indicator (Step 12). If the packet's direct progress indicator is greater than or equal to node $\vec{n}_f$'s recorded progress indicator (i.e., DPI $\geq$ RPI), node $\vec{n}_f$ concludes that node $\vec{n}_t$'s packet should be forwarded and sets a `DATA` timer to forward the packet (Step 13). Otherwise (i.e., DPI $\ngeq$ RPI), node $\vec{n}_f$ concludes that node $\vec{n}_t$'s packet is erroneous or redundant and sets a `KILL` timer to transmit a `KILL` packet (Step 14).

Regardless of whether node $\vec{n}_f$ transmits a packet, node $\vec{n}_f$ updates its recorded progress indicator by taking the highest of the recorded and direct progress indicators (Step 15). The progress indicators available to node $\vec{n}_f$ after the reception has been fully processed are shown in Figure 5.5.

Chapter 6

EXAMPLE FUNCTIONS

In order to demonstrate the effectiveness of progress indicator functions and our progress routing algorithm, we considered a specific geographic routing protocol (GPSR), reduced it to a set of designated paths for each routing scenario, derived two different progress indicator functions for it, each with a heuristic timing function, and implemented the progress indicator functions in Progress Routing. We call the resulting protocols **Greedy-Only Progress Routing** and **Greedy-Perimeter Progress Routing**. This chapter illustrates the general process to make Progress Routing emulate a given geographic routing protocol, thus eliminating beacon packets for the protocol. We present the designated paths we developed in Section 6.1 and the progress indicator functions for these designated paths in Sections 6.2 and 6.3. Finally, we present our heuristic timing functions for the two progress indicator functions in Section 6.4.

6.1 The Designated Paths

Because GPSR can use two different sets of criteria for selecting a next hop (i.e., greedy and perimeter), we divide our set of designated paths into two subsets: **greedy paths** and **perimeter paths**. When some node $\vec{n}_i$ in GPSR transmits a packet in greedy mode and selects node $\vec{n}_{i+1}$ to forward the packet, the full path from the source $\vec{s}$ to node $\vec{n}_i$ *including* the selected forwarding node $\vec{n}_{i+1}$ is a greedy path. In other words, in our work, a greedy path includes all perimeter and greedy hops that the packet has taken; the fact that the path ends with a greedy hop makes it a greedy path. Likewise, when a node in GPSR transmits a packet in perimeter mode,

we consider the full path from the source to the next hop node as a perimeter path.

In GPSR, a node transmits in greedy mode if it can forward the packet to a neighbor closer to the destination than any other node that has transmitted the packet. Thus, our set of greedy paths is the set of all paths where the final node (i.e., the chosen forwarding neighbor) in the path is closer to the destination than any other node in the path. In any particular routing scenario, we call this set of greedy designated paths $\mathbf{D}_g$.

In Chapter 4, we define a set of designated paths as a representation of the behavior of a geographic routing algorithm in a specific routing scenario, given full neighbor knowledge. Specifically, the set of designated paths for a routing algorithm in a specific routing scenario is the set of paths that the algorithm will allow a copy of the packet to take. We note that our set of designated paths, $\mathbf{D}_g$, contains paths that GPSR would never use. For example, in the routing scenario in Figure 6.1, GPSR would never use path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_3, \vec{n}_4)$ because node $\vec{n}_2$ is closer to the destination than $\vec{n}_1$ and node $\vec{n}_4$ is closer to the destination than node $\vec{n}_3$. We use a superset of GPSR's designated paths to emulate GPSR without beacons; this fact allows our progress indicator function to indicate high progress for a packet that has been close to the destination, even if that packet has taken a strange path to get that close.

As discussed previously, there may not always be a next node for a path in $\mathbf{D}_g$ that makes progress. For example, consider the example network shown in Figure 6.2. We can see that paths $(\vec{s})$, $(\vec{s}, \vec{n}_1)$, and $(\vec{s}, \vec{n}_1, \vec{n}_2)$ are in $\mathbf{D}_g$, but path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_3)$ is not. Similar to GPSR, we rectify this problem by adding perimeter paths that follow the right hand rule. Each of these perimeter paths is composed of a sequence of nodes that GPSR would have selected in perimeter mode, starting with a greedy path that ends at a local minimum node. These perimeter paths are split into two sets, $\mathbf{D}_{gp}$ and $\mathbf{D}_p$, reflecting GPSR's different criteria for the first perimeter-mode transmission and successive perimeter-mode transmissions (see Chapter 3).

The first step in using the right hand rule is to have the packet place its "right

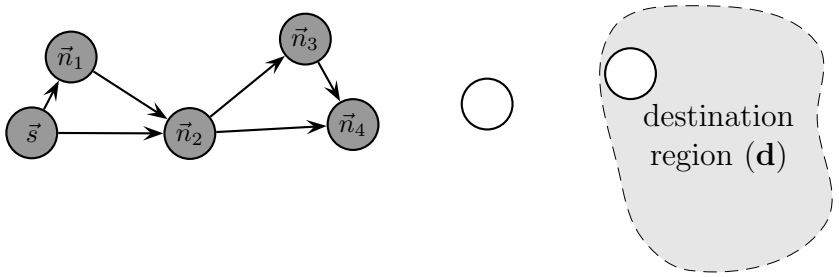


Figure 6.1. An example showing the difference between GPSR's designated paths and our designated paths when we emulate GPSR. In this scenario, GPSR would never designate node $\vec{n}_1$ or node $\vec{n}_3$ because of node $\vec{n}_2$ and node $\vec{n}_4$. Thus, to avoid violating Property 1 of Definition 2 (see Chapter 4), a progress indicator function strictly for GPSR's greedy paths must give a low progress indicator for a path that includes node $\vec{n}_1$ or $\vec{n}_3$, even though nothing is gained by such a restriction.

hand" on the "wall in front of it". This act is equivalent to finding the neighbor (e.g., node $\vec{n}_3$ in Figure 6.2) of the local minimum node (e.g. node $\vec{n}_2$ in Figure 6.2) that minimizes the angle measured counterclockwise from the direction of the center of the destination. See Figure 3.2 in Chapter 3 for further explanation of this process. These paths are the transition paths between greedy and perimeter modes, denoted $\mathbf{D}_{gp}$. A path D is in $\mathbf{D}_{gp}$ if, and only if, it has the following properties:

1. the second-to-last node, D_{end-1}, is a local minimum node (this property keeps $\mathbf{D}_{gp}$ disjoint from $\mathbf{D}_g$),

2. the subpath excluding the last node, D_{end}, is a greedy path, i.e., path $D_{1\ldots end-1}$ is in $\mathbf{D}_g$, and

3. the last node, D_{end}, minimizes the counterclockwise angle measured from the center of the destination, $\bar{\mathbf{d}}$, about D_{end-1}, i.e.,

$$D_{end} = \underset{\vec{n}\in \mathrm{nbr}\, D_{end-1}}{\operatorname{argmin}} \overset{\mathrm{ccw}}{\measuredangle}\, \bar{\mathbf{d}} D_{end-1} \vec{n}. \tag{6.1}$$

For example, in Figure 6.2, node $\vec{n}_3$ has the smallest angle measured counterclockwise

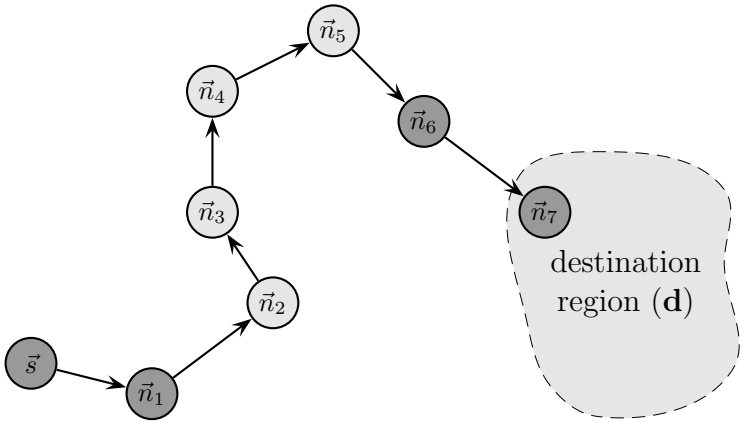


Figure 6.2. An example where greedy paths do not, by themselves, reach the destination. For these situations, we use paths that follow the right hand rule.

from $\bar{\mathbf{d}}$ among neighbors of node $\vec{n}_2$; therefore path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_3)$ is in the set $\mathbf{D}_{gp}$.

Each remaining node along the right hand wall is the neighbor that minimizes the angle measured counterclockwise about its previous hop from its *second-to-last hop*, instead of the destination. For example, in Figure 6.2, node $\vec{n}_4$ is the neighbor of node $\vec{n}_3$ that minimizes the angle measured counterclockwise from node $\vec{n}_2$, therefore node $\vec{n}_4$ is the next node along the right hand wall. The set $\mathbf{D}_p$ consists of these remaining designated perimeter paths. That is, $\mathbf{D}_p$ consists of each path D that has the following properties:

1. the second-to-last node, D_{end-1}, has no neighbors closer to the center of the destination than the closest node to the destination in D (this property ensures the path is not ready to return to greedy mode and keeps $\mathbf{D}_p$ disjoint from $\mathbf{D}_g$),

2. the subpath excluding the last node, D_{end}, is a perimeter designated path, i.e, $D_{1\ldots end-1}$ is in $\mathbf{D}_{gp}$ or $\mathbf{D}_p$, and

3. the last node, D_{end}, minimizes the counterclockwise angle measured from the second-to-last node, D_{end-2}, about D_{end-1}, i.e.,

$$D_{end} = \underset{\vec{n} \in \text{nbr } D_{end-1}}{\text{argmin}} \overset{\text{ccw}}{\measuredangle} D_{end-2} D_{end-1} \vec{n}. \tag{6.2}$$

The progress relation $\rhd$, like $\mathbf{D}$, has two parts: "$\rhd_g$", or "progress in the greedy sense", and "$\rhd_p$", or "progress in the perimeter sense". Progress in the greedy sense $\rhd_g$ is straightforward: if one path contains a node closer to the destination than another does, then it has higher progress than the other.

For progress in the perimeter sense $\rhd_p$, one path has made more progress than another if it has traveled a greater number of hops along the same right hand wall. Combining these two ideas of progress, one path has more progress than another if it has more progress in either the greedy sense ($\rhd_g$) or the perimeter sense ($\rhd_p$).

It is important to note that this set of designated paths does not guarantee delivery, even if the network is connected. Scenarios exist, e.g., when links along the right hand wall cross each other, where the set of successive perimeter paths (paths in $\mathbf{D}_p$) do not lead to a greedy path. In fact, GPSR's graph planarization system (which is not reflected by our progress indicator functions) is designed to address this problem, but even GPSR's graph planarization system is not perfect for all networks [20, 21]. Nonetheless, as shown in our simulations in Chapter 7, using a progress indicator function that reflects the designated paths presented in this section results in excellent success rates; in other words, our delivery rates are high even though we ignore the scenarios where our designated paths do not lead to the destination.

6.2 The Greedy-Only Progress Indicator

We now present our **greedy-only progress indicator function** for the designated paths presented in Section 6.1. Progress Routing with the greedy-only progress indicator function and the corresponding timing function from Section 6.4 yields Greedy-Only Progress Routing.

Remember that the progress relation $\rhd_g$ for the greedy designated paths only depends upon the minimum distance of each path to the destination; one path gives a packet more progress than another if and only if the path passes closer to the des-

tination than the other path. Thus, a function on paths and destination regions that satisfies Property 1 of Definition 2 for greedy-sense progress must have the following property: for two paths P and P' that start and end at the same location, if $\mathcal{F}(P, \mathbf{d}) < \mathcal{F}(P', \mathbf{d})$, then either P' has a node at least as close to the destination as any node in P, or P does not end at its own closest node to the destination.

This property is easy to ensure. The function

$$\mathcal{F}_g(P, \mathbf{d}) = -\min_{\vec{n} \in P} |\vec{n} - \mathbf{d}| \tag{6.3}$$

actually ensures that if $\mathcal{F}_g(P, \mathbf{d}) < \mathcal{F}_g(P', \mathbf{d})$, then P' has a node *strictly* closer to the destination than any node in P, whether P ends at P's own closest node to the destination or not. We use this function $\mathcal{F}_g$ as the greedy-only progress indicator function, with the addition that we set the progress indicator of any path that includes a node in the destination region $\mathbf{d}$ to ∞.

Unlike most greedy-only forwarding strategies, this progress indicator can actually make progress despite encountering local minimum nodes. Suppose a copy of a packet using this progress indicator encounters a local minimum node of distance x from the destination, then the progress indicator of all subsequent paths (until a node closer to the destination is reached) will be $-x$. Thus, until a node closer to the destination is reached, all paths will have equal progress indicators; in other words, all nodes will forward each packet received. This is effectively a flood with the exception that a node may transmit a given packet more than once. Eventually a node closer to the destination than the local minimum node will receive a copy of the packet; this node will then begin a series of `KILL` packets that will propagate to end the flood.

6.3 The Greedy-Perimeter Progress Indicator

The progress indicator function from the previous section, $\mathcal{F}_g$, only indicates progress in the greedy sense (i.e., $\rhd_g$). Essentially, it results in greedy transmissions

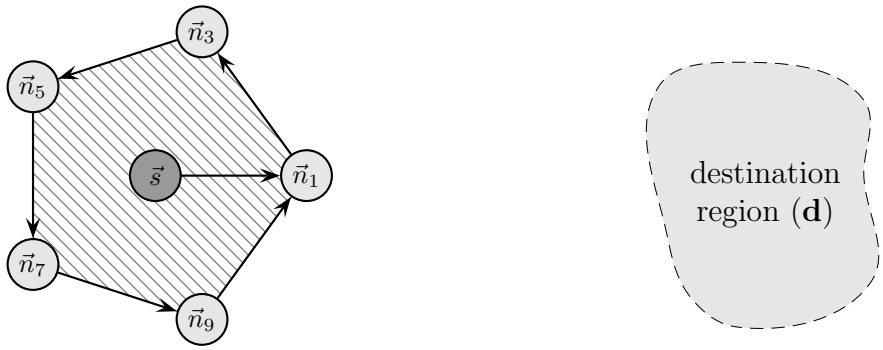
until a local minimum node is reached, and then causes a localized flood at the local minimum node. We can restrict these redundant retransmissions by *augmenting* the greedy-only progress indicator with a progress indicator that includes progress in the perimeter sense (i.e., $\rhd_p$) *and* progress in the greedy sense. This progress indicator function, the **greedy-perimeter progress indicator function**, is used by Greedy-Perimeter Progress Routing.

To construct the progress indicator, we need a function, $\mathcal{F}_p$, on paths and destination regions that has the following property: for two paths P and P' that start and end at the same location and have in common their closest node to the destination, if $\mathcal{F}_p(P, \mathbf{d}) < \mathcal{F}_p(P', \mathbf{d})$, then either P does not follow the right hand wall, or P' has a subpath D that follows the right hand wall at least as far as P.

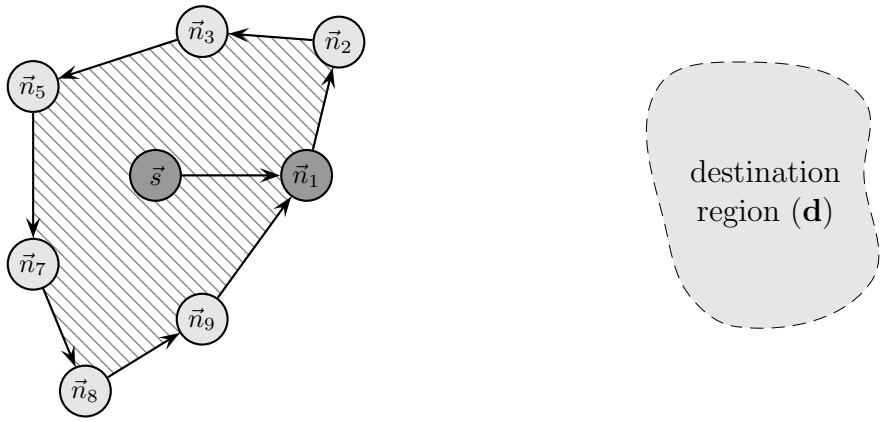
To find such a function, we use the relationship between *area* and the right hand rule. For example, Figure 6.3 shows how GPSR's perimeter mode can change behavior as more nodes are added to a network. As more nodes are added, the right hand wall moves outward, which means that the area enclosed by the path increases. In Figure 6.3, the path that follows the right hand wall is always the path that maximizes the area enclosed by the curve.

Alternatively, sometimes *minimum* area corresponds to the path that follows the right hand wall instead of *maximum* area. For example, in Figure 6.4, the extra nodes added in part (b) shrink the right hand wall and make the area encompassed by the path smaller. We note, however, that there exists a fundamental difference between the situations in Figures 6.3 and 6.4; specifically, in Figure 6.3, the packet follows the right hand wall *counterclockwise* (because the destination region is "outside of the wall") and in Figure 6.4, the packet follows the right hand wall *clockwise* (because the destination region is "inside the wall"). If we count "counterclockwise enclosed area" as positive and "clockwise enclosed area" as negative, then the right hand wall path will maximize area.

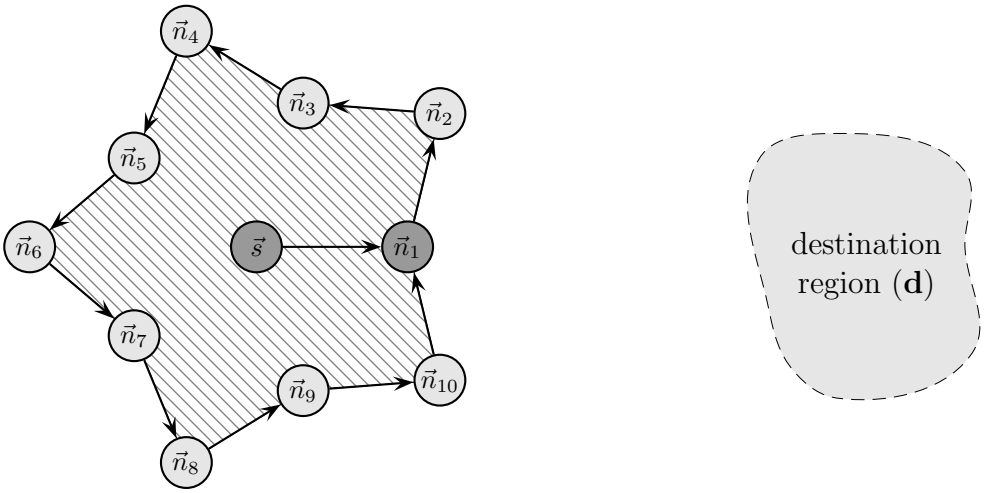
We can capture this relationship using a sum of trapezoids to represent the area



(a) A network with a small right hand wall.

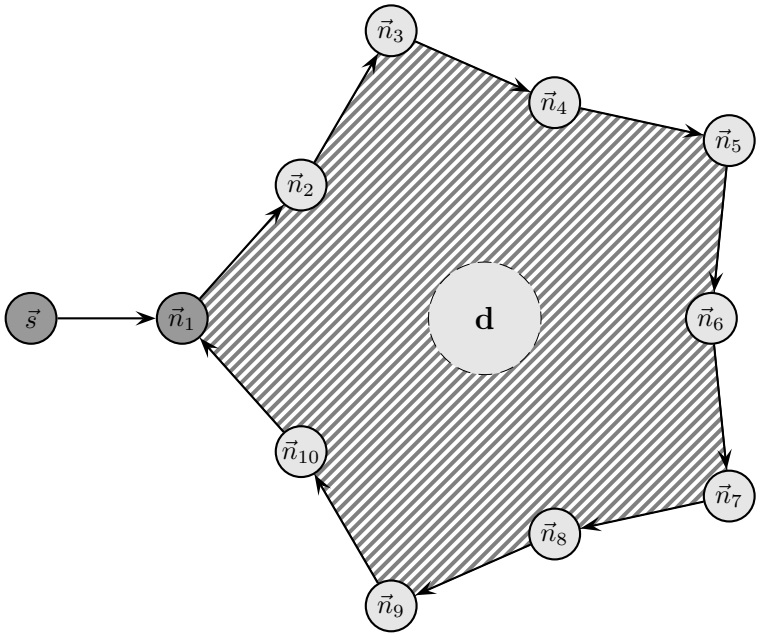


(b) A network with an expanded right hand wall.

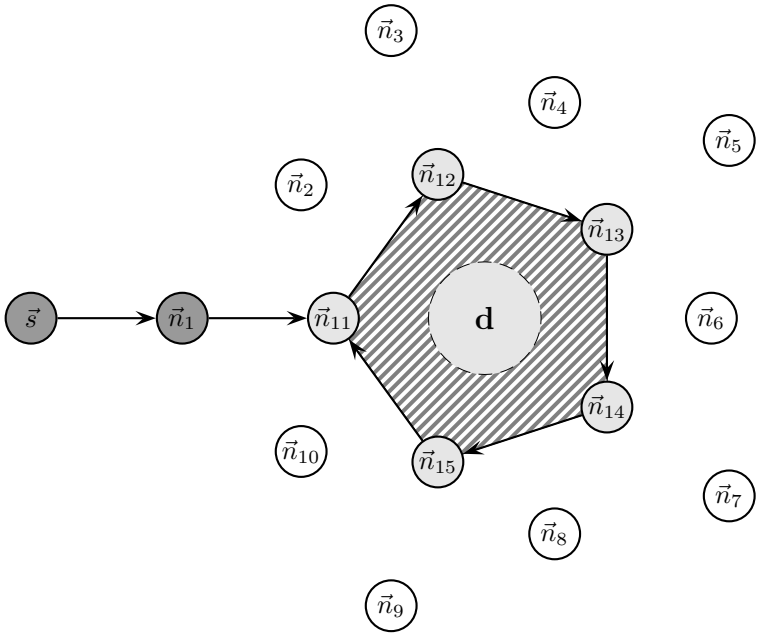


(c) A network with an even more expanded right hand wall.

Figure 6.3. A series of networks showing a situation where the right hand rule path always encompasses the maximum possible area. Thus, area can be used to indicate progress.



(a) A network with a wide wall around the destination.



(b) A network with a smaller wall around the destination.

Figure 6.4. A series of networks showing a situation where the right hand rule path always encompasses the minimum possible area.. The difference between this network and the network in Figure 6.3 is that in this network, the packet moves clockwise, whereas in Figure 6.3, the packet moves counterclockwise.

under the curve. Each **greedy-perimeter progress indicator** contains the greedy-only progress indicator as well as the perimeter component $\mathcal{F}_p$, which is initialized to zero when the packet is originated. Whenever a packet reaches a node that is closer to the destination than that packet has ever been (i.e., whenever the greedy-only progress indicator $\mathcal{F}_g$ is updated), the packet resets its perimeter progress indicator to zero; resetting $\mathcal{F}_p$ to zero ensures the perimeter progress indicator is ready whenever the current node is a local minimum node. Each subsequent node that receives the packet but is not closer to the destination than the minimum-distance node updates the perimeter progress indicator with the "area under the link". That is, the area of the trapezoid under the link just traversed, extending vertically up or down to the x-axis of the coordinate system. Specifically, if $\vec{n}_i$ is the current hop and $\vec{n}_{i-1}$ is the previous hop, we let $w = (\vec{n}_{i-1})_x - (\vec{n}_i)_x$ be the width of the trapezoid, $h_1 = (\vec{n}_{i-1})_y$ be the height of one leg of the trapezoid, and $h_2 = (\vec{n}_i)_y$ be the height of the other leg of the trapezoid. All three lengths may or may not be negative; this is intentional. Thus, each node $\vec{n}_i$ performs the following operation[1] on the packet's perimeter progress indicator $\mathcal{F}_p$:

$$\mathcal{F}_p \leftarrow \mathcal{F}_p + w(h_1 + h_2) \tag{6.4}$$

or, in terms of the coordinates of the nodes involved,

$$\mathcal{F}_p \leftarrow \mathcal{F}_p + \big((\vec{n}_{i-1})_x - (\vec{n}_i)_x\big)\big((\vec{n}_{i-1})_y + (\vec{n}_i)_y\big). \tag{6.5}$$

This process of updating perimeter progress indicators is illustrated in Figures 6.5-6.7. In these figures, which all show the same network scenario, a packet starts at source $\vec{s}$ and begins moving through the network. As discussed, in Progress Routing, it is theoretically possible for the packet to take any path through the network. Assume a copy of the packet travels to the local minimum node $\vec{n}_1$, and then travels through the network to node $\vec{n}_8$, along the path shown in Figures 6.5-6.7.

[1] We drop the constant factor $\frac{1}{2}$ in our formula for area of a trapezoid, $\frac{1}{2}w(h_1 + h_2)$, since all we are interested in is comparisons, and positive constant factors do not affect those comparisons.

Each node along the way, except the node closest to the destination (i.e., $\vec{n}_1$), adds the area of the trapezoid under the link just traversed (see Figure 6.5). Continuing in this way gives the "area under the path". Since the packet moves to the right on its way from node $\vec{n}_1$ to $\vec{n}_8$, each node it visits is to the right of its predecessor. Thus, at each hop, $(\vec{n}_{i-1})_x < (\vec{n}_i)_x$ or $w = (\vec{n}_{i-1})_x - (\vec{n}_i)_x < 0$, and the area under the curve is counted negatively. Counting the area as negative is appropriate in this situation; if extra nodes were added to the network to bring the right hand wall closer to the destination, then the area under the path following the new right hand wall would be reduced, and the resulting progress indicator would be higher (i.e., less negative).

Figure 6.6 shows the situation when the packet has reached node $\vec{n}_{10}$. To get to node $\vec{n}_{10}$, the packet moved to the *left* instead of the *right*; since $(\vec{n}_{i-1})_x > (\vec{n}_i)_x$, the trapezoids' areas are added instead of subtracted. The fact that the area is now added cancels out some of the area previously included in the progress indicator, leaving the shaded area in Figure 6.6.

The shaded area in Figure 6.7 is the perimeter progress indicator stored in the packet once the packet has reached node $\vec{n}_{13}$. Nodes $\vec{n}_{10}$ through $\vec{n}_{13}$ are again arranged left-to-right, therefore the trapezoids' areas are subtracted (as in Figure 6.5).

For an example of how $\mathcal{F}_p$ is used by the progress routing algorithm, see Figure 6.8. Assume, in this figure, that the first copy of a particular unique packet took path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4)$. The corresponding perimeter progress indicator is shown in Figure 6.8(a).

Suppose node $\vec{n}_4$ receives a copy of the packet that has a higher progress indicator than what node $\vec{n}_4$ has stored. In this case, there are two possibilities. The first of these possibilities is that the higher progress indicator corresponds to a path such as path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_3, \vec{n}_4)$ shown in Figure 6.8(b). The progress indicator received is higher than the progress indicator stored because path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4)$ does not actually follow the right hand wall, and therefore is not a designated path. In Figure 6.8(b), the real right hand wall includes node $\vec{n}_3$, which the other nodes may not have been

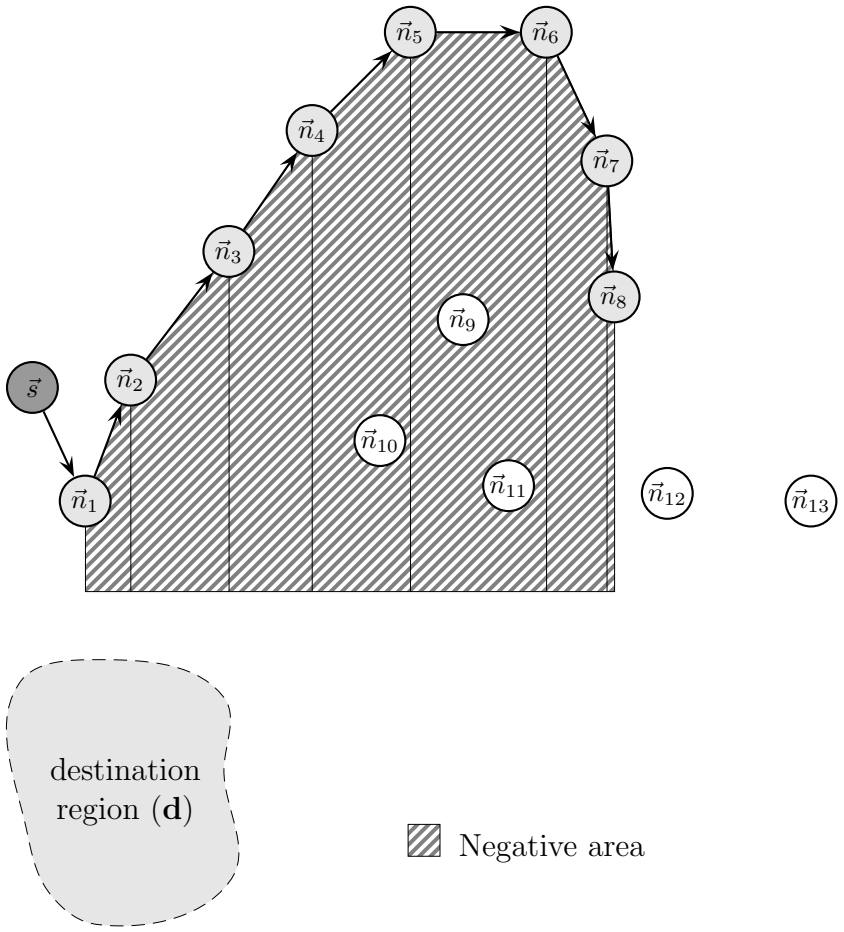


Figure 6.5. An example showing the accumulation of area for a perimeter progress indicator. Each node from node $\vec{n}_1$ to node $\vec{n}_8$ adds the area of the trapezoid under the link just traversed. Since each node that transmits is to the right of its predecessor, the area added is negative.

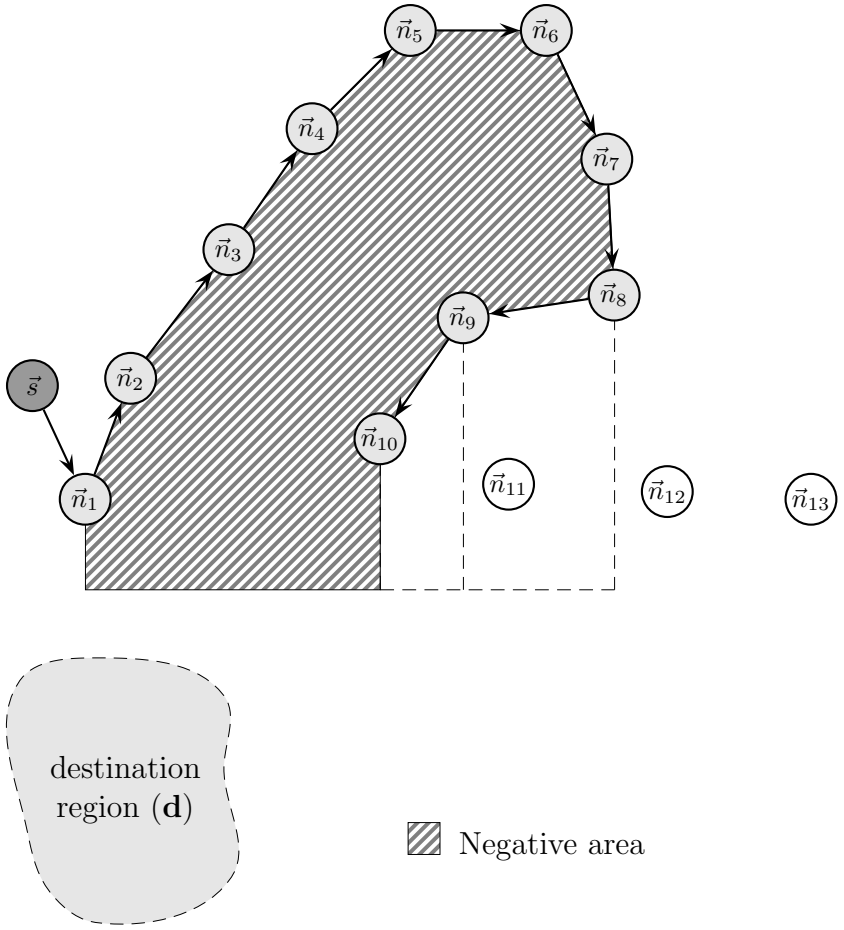


Figure 6.6. A continuation of the example in Figure 6.5. Since nodes $\vec{n}_9$ and $\vec{n}_{10}$ are to the left of their respective predecessors, they *add* area to the progress indicator and therefore cancel out some of the negative area.

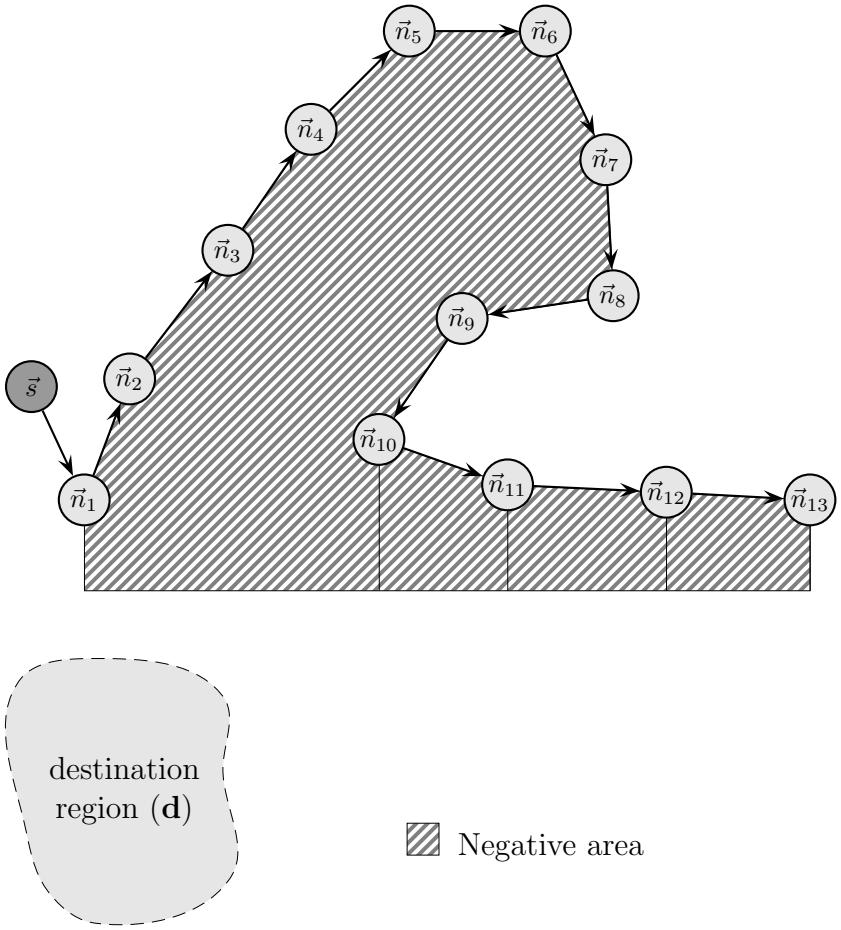
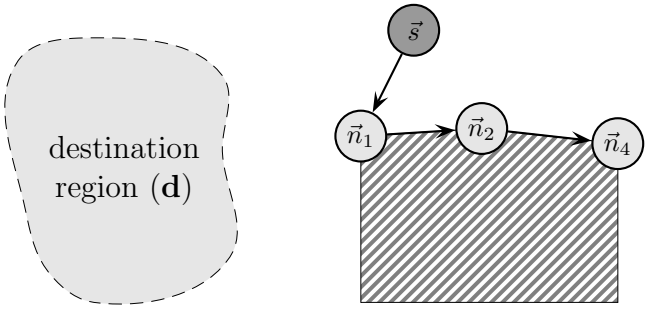
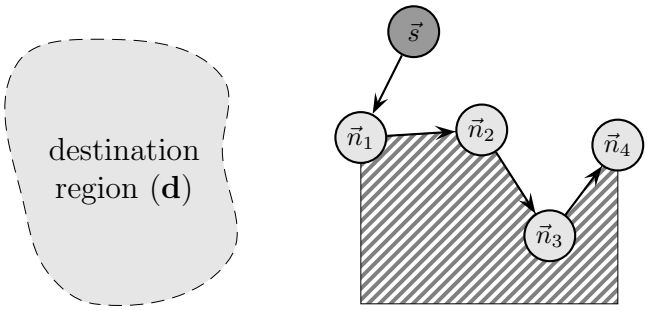


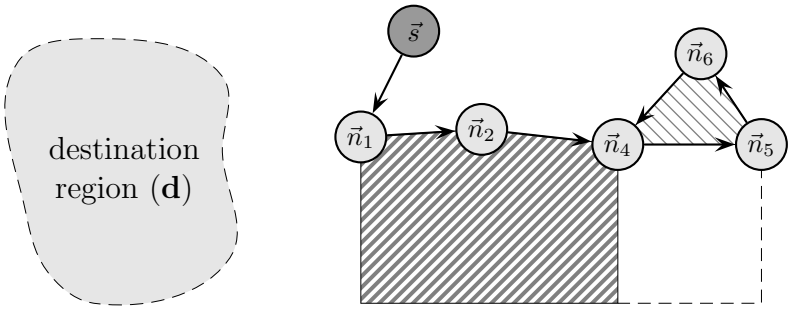
Figure 6.7. The conclusion of the examples shown in Figures 6.5 and 6.6. Each node is again to the right of its predecessor; thus area is subtracted as the packet moves from node $\vec{n}_{10}$ to $\vec{n}_{13}$.



(a) The area (or progress indicator) for a simple path that is taken by a copy of the packet.



(b) If node $\vec{n}_3$ exists, then the path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4)$ that the packet took in (a) is erroneous.



(c) A copy of the packet that has been to nodes $\vec{n}_5$ and $\vec{n}_6$ has made more progress than the copy that took path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4)$ in (a).

Positive area
Negative area

Figure 6.8. An example showing two different reasons a progress indicator for a path can be less than another. If node $\vec{n}_3$ exists as in (b), then path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4)$ in (a) is not a designated path. If the packet has already been farther along the right hand wall as in (c), then it has made more progress than the copy that took path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4)$ in (a).

aware of when the packet traversed path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4)$. The difference between the two progress indicators, i.e., the area of the triangle $\triangle \vec{n}_2\vec{n}_3\vec{n}_4$, represents the portion of the path in Figure 6.8(a) that failed to follow the right hand wall.

The second possibility is that the higher progress indicator received corresponds to a path such as path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4, \vec{n}_5, \vec{n}_6, \vec{n}_4)$ shown in Figure 6.8(c). That is, path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4)$ may be a designated path; however, the higher progress indicator received is higher than the progress indicator stored because its corresponding path includes a higher-progress designated subpath (namely, path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4, \vec{n}_5, \vec{n}_6)$). Even if path $(\vec{s}, \vec{n}_1, \vec{n}_2, \vec{n}_4, \vec{n}_5, \vec{n}_6, \vec{n}_4)$ is not designated, (i.e., a node exists such that $\vec{n}_5$ is *not* on the right hand wall), a packet can still take it to get to $\vec{n}_4$ through some combination of redundant transmissions or control packets discussed in Chapter 5.

These two possibilities for the higher progress indicator correspond exactly to the two possible conditions in Property 1 of Definition 2, the definition of a progress indicator function. That is, if one progress indicator is greater than a second, then either the path for the second is erroneous (as in Figure 6.8(b)), or the first indicates the existence of a higher-progress path (as in Figure 6.8(c)).

To integrate $\mathcal{F}_p$ and $\mathcal{F}_g$, we track both simultaneously. If the greedy-only portions differ for two progress indicators, then we only examine the greedy-only portions. If the greedy-only portions for two progress indicators are the same, then we compare the perimeter progress portions.

One drawback of our perimeter portion of the greedy-perimeter progress indicator is that the relationship exploited by $\mathcal{F}_p$ only holds when the two paths in question start and end at the same nodes, and do not cross themselves or each other. We note, however, that face traversal methods (e.g., as included in GPSR) also fail when links cross each other.

6.4 The Heuristic Timing Function

As mentioned in Chapter 5, a node should transmit quickly if it is likely to make the most progress. Specifically, a node at the end of a greedy path should transmit quickly if it is close to the destination region, and a node at the end of a perimeter path should transmit quickly if the relevant counterclockwise angle is small. In this section, we assume some node $\vec{n}_i$ is scheduling a transmission and node $\vec{n}_{i-1}$ was the previous hop node.

Let us assume that Greedy-Only Progress Routing is being used. Let Δ be node $\vec{n}_i$'s improvement in distance to the destination compared to $\vec{n}_{i-1}$, i.e., $\Delta = |\vec{n}_{i-1} - \bar{\mathbf{d}}| - |\vec{n}_i - \bar{\mathbf{d}}|$. Also, let Δ_{max} be the maximum expected transmission radius[2]. An example of these values is shown in Figure 6.9. If t_{max} is a protocol configuration parameter representing the maximum allowed delay, then the time to delay the transmission being scheduled, t, is given by:

$$t = \frac{t_{max}}{2} \times \frac{\Delta_{max} - \Delta}{\Delta_{max}}. \tag{6.6}$$

When $\Delta = \Delta_{max}$, then node $\vec{n}_i$ is as close to the destination as any node can be and still hear node $\vec{n}_{i-1}$; in this case, $t = 0$. If $\Delta = -\Delta_{max}$, then node $\vec{n}_i$ is as far from the destination as possible and still hear node $\vec{n}_{i-1}$; in this case, $t = t_{max}$. In summary, each neighbor of node $\vec{n}_{i-1}$ that decides to forward the packet received from node $\vec{n}_{i-1}$ will delay the transmission an amount of time between 0 and t_{max}.

Now, let us assume the Greedy-Perimeter Progress Routing is being used. In this case, node $\vec{n}_i$ may be at the end of a greedy path or a perimeter path. If node $\vec{n}_i$ is at the end of a greedy path, then it uses the delay for Greedy-Only Progress Routing, i.e., the delay given in Equation (6.6). If node $\vec{n}_i$ is at the end of a perimeter path, node $\vec{n}_{i-1}$ might have other neighbors that are at the end of a greedy path, and these "greedy neighbors" should have priority. To simplify our discussion momentarily,

[2]To account for nonuniform transmission radii and location errors such as errors due to movement, Δ_{max} should be configured slightly larger than the actual maximum transmission radius.

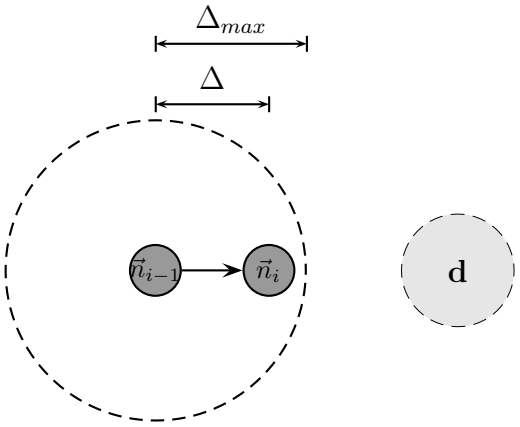


Figure 6.9. An illustration of the variables used for the greedy-only heuristic timing function. In this situation, node $\vec{n}_i$ is Δ closer to the destination than $\vec{n}_{i-1}$. The radius of the dashed circle is the maximum expected transmission range, or Δ_{max}.

let us assume that no neighbor of $\vec{n}_{i-1}$ is a "greedy neighbor". An example of this situation is shown in Figure 6.10. In this case, given the counterclockwise angle $\theta = \overset{\text{ccw}}{\measuredangle} \vec{n}_{i-2}\vec{n}_{i-1}\vec{n}_i$, node $\vec{n}_i$ delays its transmission of the packet by

$$t = \frac{t_{max}}{2} \times \frac{\theta}{2\pi}, \tag{6.7}$$

where the factor $\frac{1}{2}$ is used to ensure that, when we add in the additional delay to accommodate a potential node at the end of a greedy path, we will not produce a delay above the maximum delay, t_{max}. We note two special cases for Equation (6.7). First, a return to the second-to-last hop, i.e., $\vec{n}_i = \vec{n}_{i-2}$, should have the highest delay. Thus, in the case that $\vec{n}_i = \vec{n}_{i-2}$, we set $\theta = 2\pi$. Second, if the previous hop node, node $\vec{n}_{i-1}$, is at the end of a greedy path, then the counterclockwise angle should be measured from the destination instead of from the second-to-last node. In other words, we set $\vec{n}_{i-2} = \bar{\mathbf{d}}$ if node $\vec{n}_{i-1}$ is at the end of a greedy path.

When node $\vec{n}_{i-1}$ is close to the destination, it may have neighbors at the end of a greedy path. If node $\vec{n}_i$ is a neighbor at the end of a greedy path, then node $\vec{n}_i$ will delay the packet's transmission as much time as it would for Greedy-Only Progress

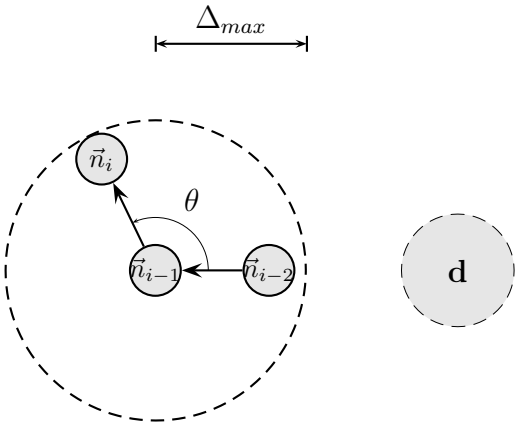


Figure 6.10. An illustration of the variables used for the greedy-perimeter heuristic timing function in the special case where no neighbor is at the end of a greedy designated path. The angle measured counterclockwise from node $\vec{n}_{i-2}$ around node $\vec{n}_{i-1}$ to node $\vec{n}_i$ is θ.

Routing, i.e., an amount of time based on its difference Δ in Equation (6.6). If node $\vec{n}_i$ is at the end of a perimeter path, it should wait the appropriate perimeter delay, i.e., based on the angle θ in Equation (6.7), *plus* the longest possible delay for a node at the end of a greedy path.

An example of this situation is shown in Figure 6.11. Let us assume that node $\vec{n}_{i-2}$ is closer to the destination than any other node in the path so far, node $\vec{n}_{i-1}$ made the last transmission, and node $\vec{n}_i$ is deciding how long to wait to transmit node $\vec{n}_{i-1}$'s packet. Since node $\vec{n}_i$ is not closer to the destination than any other node in the path, node $\vec{n}_i$ is at the end of a perimeter path; thus, node $\vec{n}_i$ must determine the longest possible delay a node at the end of a greedy path will use.

Recall that any node at the end of a greedy path *must* be closer to the destination than the closest node to the destination in the path taken; in the example in Figure 6.11, this minimum-distance node is node $\vec{n}_{i-2}$. Thus, any node at the end of a greedy path must wait a delay less than node $\vec{n}_{i-2}$ would wait. Now, let us consider the maximmum delay, denoted t_g, that a node at the end of a greedy path would wait to transmit. Let Δ_g represent the improvement in distance of the node on the

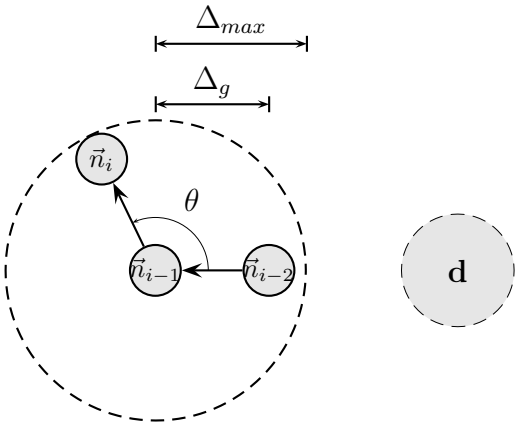


Figure 6.11. An illustration of the variables used for the greedy-perimeter heuristic timing function. The angle measured counterclockwise from node $\vec{n}_{i-2}$ around node $\vec{n}_{i-1}$ to node $\vec{n}_i$ is θ. The farthest node from the destination that can still use a greedy-based timer is Δ_g away from node $\vec{n}_i$. In this figure, node $\vec{n}_{i-2}$ is the closest node to the destination along the path taken by the copy of the packet that node $\vec{n}_i$ is scheduling to transmit.

path that is closest to the destination over node $\vec{n}_{i-1}$. Since this node at the end of a greedy path is closer to the destination than node $\vec{n}_{i-1}$, it will schedule its packet to transmit with delay

$$t_g = \frac{t_{max}}{2} \times \frac{\Delta_{max} - \Delta_g}{\Delta_{max}}. \tag{6.8}$$

If the delay t_g is negative (i.e., because Δ_g is large, indicating that the node on the path closest to the destination is far closer to the destination than node $\vec{n}_{i-1}$), then node $\vec{n}_i$ concludes that no nodes at the end of a greedy path are in range of node $\vec{n}_{i-1}$, and uses $t_g = 0$.

Now that node $\vec{n}_i$ has t_g, it waits a delay of

$$t = \frac{t_{max}}{2} \times \frac{\theta}{2\pi} + t_g \tag{6.9}$$

when scheduling the packet to transmit. We note that t_g will never exceed $\frac{t_{max}}{2}$, thus t can never exceed t_{max}.

Chapter 7

SIMULATION

In order to evaluate the performance of our progress indicator functions, heuristic timing functions, and the progress routing algorithm, we implemented them in the simulator ns-2 [22]. Also, we compared our progress indicator functions with GPSR, the exact details of which are available in [10]. To show relative performance, we present three metrics for each protocol:

Success rate: The number of unique data packets received by a node in the destination region, divided by the number of unique packets in the simulation. If the goal is geocasting, each protocol would be accompanied by a broadcast in the destination region; we ignore the cost associated with this broadcast because this cost is identical for GPSR and Progress Routing.

Transmissions per hop: The average transmissions necessary per hop to the destination; this includes data transmissions, routing protocol control packets transmissions, beacon packets, MAC control packet transmissions, ARP packet transmissions, etc. This metric is computed as the total number of transmissions of any kind times the success rate and divided by the total number of hops used by any successful packet[1].

End-to-end delay: The average time between transmission of the first copy of a data packet and its first reception by a node in the destination region. Only successful packets are calculated in the end-to-end delay result.

[1]Simulation results in the literature typically present transmissions per successful, unique packet or total transmissions overall as an overhead metric. We present packets per hop to illustrate one of the strengths of Progress Routing; specifically, overhead in Progress Routing is often very close to the theoretical minimum of one packet per hop.

We present the results of these metrics as we vary the following parameters: node density, mobility, congestion and a combination of the three metrics called severity.

The rest of this chapter is organized as follows. Section 7.1 details our simulation setup. Sections 7.2, 7.3, 7.4, and 7.5 present our results as node density, mobility, congestion, and overall network severity vary respectively. Finally, Section 7.6 summarizes our simulation results.

7.1 Setup

Our simulation setup differs from that presented in Karp *et al.* [10]. Karp *et al.* used a simulation setup similar to the well-known MANET routing protocol comparison by Broch *et al.* [2] in order to compare GPSR with Dynamic Source Routing (DSR) [7]. Our simulation setup differs from [10] because we executed both Progress Routing and GPSR in geocast mode. In other words, source nodes in our simulations transmitted packets to a destination region, not a destination node.

We implemented GPSR and two versions of Progress Routing (i.e., Greedy-Only Progress Routing and Greedy-Perimeter Progress Routing) in ns-2 version 2.31 [22], running on Ubuntu Linux machines with the 2.6.20 kernel. We used the Akaroa2/NS2 Interface [24] random number generator with a different random seed for each simulation. Each simulation was initialized with a different node arrangement according to the steady-state random waypoint distribution [18]. Each of these scenarios was repeatedly regenerated until, at time 0, the network scenario was not partitioned, at least one node was located in the destination region, and the source node was *not* located within the destination region. For the mobile scenarios, we did not ensure that these conditions remained true throughout the simulation.

For each data point in each graph, we ran 100 independent simulations with identical parameters and then took the *median* of the results, computing 95% confidence intervals for each median. To make the confidence intervals clearer, each line of data is slightly offset (horizontally) relative to the others on each graph. Also,

while many papers in the literature present averages, the distributions for most of our data are extremely long-tailed; thus averages do not give a good representation of the typical case of Progress Routing's performance. For example, some network scenarios give rise to overhead that is not present in most other scenarios. We believe, given the strong simulation results we present in this chapter, that Progress Routing can be optimized for these pathological scenarios (see our plans for future work in Chapter 8). In summary, to avoid the effect from extremely long-tailed distributions, we plot the median instead of the average in our results.

All of the scenarios generated were 300m by 600m. Each node had a transmission radius of 100m, resulting in a node's coverage area of 31,400m^2, or 17.4% of the total simulation area. Since each node had a uniform transmission radius, our networks approximated unit disk graphs. Our scenarios might deviate from the unit disk graph model because packets might be dropped, and nodes may have inaccurate neighbor location information. Because GPSR's graph planarization system assumes unit disk graphs and Greedy-Perimeter Progress Routing can fail if the network graph is non-planar, we expect the uniformity of the transmission radii to favor GPSR's performance over Greedy-Perimeter Progress Routing.

Except during the density study, we had 50 nodes in the simulations, and each node had an average of 7.8 neighbors. In each trial, a single node sent data packets with 64-byte payloads at even intervals to the destination region. The destination region was a 150m x 150m rectangle in a corner of the simulation, touching the edge of the network on two sides. A screenshot of one of these scenarios from iNSpect 3.5 [16] is shown in Figure 7.1.

We graphed the average number of hops required to reach the destination for both protocols (not shown), and found that Progress Routing required approximately between three and four hops to reach the destination, and GPSR required approximately between three and five hops to reach the destination. GPSR required more hops because of its graph planarization system, which causes GPSR to ignore certain

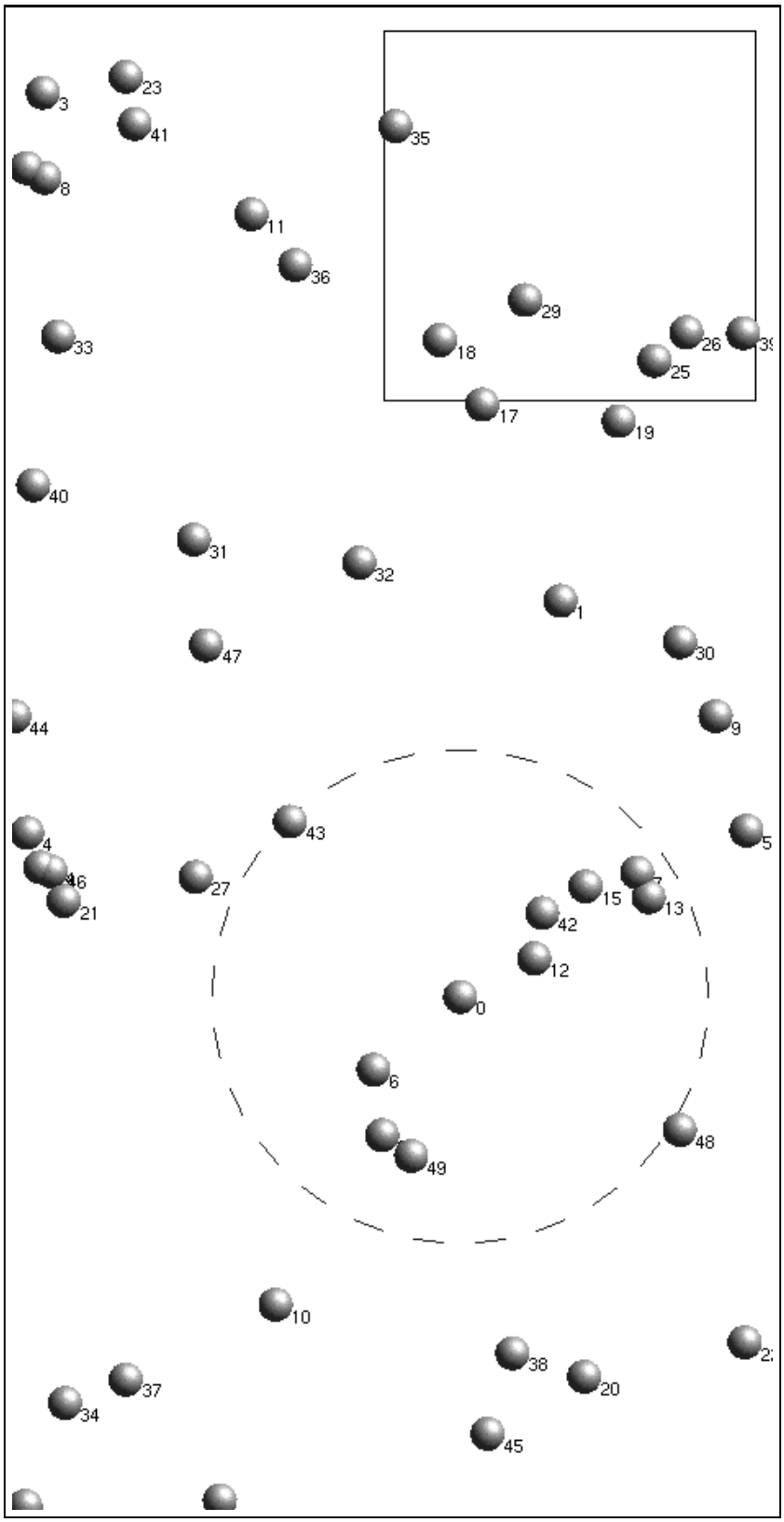


Figure 7.1. A screenshot of a sample network scenario viewed in iNSpect 3.5. The square region is the destination region, and the dashed circle is the transmission radius of node 0, the source node.

links to prevent routing errors.

Each simulation was 102 seconds in duration. For GPSR, nodes began beaconing immediately (at time 0) to allow them to build their neighbor knowledge before data packets were sent. For both protocols, constant bit rate (CBR) data traffic began at 1 second and continued for 100 seconds. In order to allow stragglers to reach the destination region, the actual simulation ended and results were computed at 102 seconds. Nodes in GPSR simulations continued to transmit control packets during this extra second.

We initialized the time to live (`TTL`) of a packet to allow the majority of the packets enough hops to get to the destination. We found through trial and error that a `TTL` of 20 hops was sufficient for both protocols. For Progress Routing, we set the parameter `MAX_DELAY` (see the equation in Section 6.4) to 30ms, which we also arrived at by trial and error.

We used the GPSR implementation written by Brad Karp *et al.*, which we downloaded from `http://www.cs.cmu.edu/~bkarp/gpsr/gpsr.html` [9]. We made as few changes to the GPSR code as possible; the only changes concerned the addition of geocasting support. Our modified GPSR implementation is packaged with our Progress Routing implementation; see `http://toilers.mines.edu` for details on obtaining the code.

The parameters used in our simulations are summarized in Tables 7.1-7.3. Except those parameters varied as part of the following studies, all simulations use the parameters in the tables.

One difference between our GPSR parameters and our Progress Routing parameters is that for Progress Routing, we did not use a packet queue. We found experimentally that, in Progress Routing, if a node is unable to transmit a packet when it needs to because the channel is busy, then there is little point in sending the packet later. That is, another node will probably handle the responsibility first, and a later transmission because of a queued packet will just degrade performance by

Table 7.1. Global parameters for all of our simulations. Number of nodes, mobility, and transmission rate are each studied in their respective sections and are, therefore, different from the values presented in this table.

Simulator	ns-2 version 2.31
Random Number Generator	Akaroa2 [24]
Operating System	Ubuntu Linux
Kernel Version	2.6.20
MAC Layer	802.11b
Propagation Model	Two Ray Ground
Interface Queue Length	20
Network Size	300m × 600m
Network Area	180000m^2
Transmission Radius	100m
Node Footprint	17.4%
Begin Control Packets	0s
Begin Data Packets	1s
End Data Packets	101s
Compute Results	102s
Number of Nodes	50
Average Neighbor Count	7.8 neighbors
Mobility	Static nodes
Initial Distribution	Steady-state random waypoint[18]
Traffic type	CBR
Number of Senders	1
Transmission Rate	20 pkts/second
Packet Payload	64 bytes

consuming bandwidth. On the other hand, GPSR performs worse without its packet queue, so we use a standard 20 packet queue for GPSR.

The GPSR-specific parameters (see Table 7.2) are chosen to correspond to those used in [10]. In particular, nodes wait a random time value for each beacon transmission between 0.25s and 0.75s. In GPSR, a node removes an entry from its neighbor table when it does not hear a beacon from that neighbor for more than three maximum beacon intervals, or 2.25s. We expect that a higher beacon interval would result in less overhead for GPSR, but also a lower success rate in the mobility study. We used standard features for GPSR such as Perimeter Mode, Graph Planarization, and the MAC Feedback Optimization, all of which are described in detail in [10].

Table 7.2. Parameters specific to GPSR.

MAC Interface Queue Length	20
Beacon Interval	0.50 ± 0.25s
Neighbor Expiration	2.25s
Implicit Beaconing	Yes
Use Perimeter Mode	Yes
Graph Planarization	Yes
MAC Feedback Optimization	Yes

Table 7.3. Parameters specific to Progress Routing.

MAC Interface Queue Length	None
Maximum Delay (t_{max})	30ms
Maximum Transmit Range (Δ_{max})	110m

7.2 Density

The first study we performed to compare GPSR and Progress Routing was node density. Geographic routing protocols are sensitive to node density; at lower densities, local minimum nodes become more common, and local minimum nodes cause greedy forwarding to fail. We simulated both protocols at densities of 40, 50, 75, 100, and 125 nodes, and kept all other simulation parameters set to those shown in Table 7.1. 40 nodes makes a reasonable lower bound on our evaluation of network density; we discovered that the majority of networks with density significantly less than 40 nodes were partitioned, indicating that an unpartitioned network with a density significantly less than 40 nodes is unrealistic in a 300x600m area. Networks are well-connected with 125 nodes, and results do not vary significantly with additional nodes. Using scripts, we calculated that each node had an average of 6.4, 7.8, 11.4, 14.8, or 18.2 neighbors as the density of nodes increased.

In Figure 7.2, success rate is plotted against the number of nodes in the network. As shown, the median success rate for both versions of Progress Routing is 100% regardless of the density; GPSR, however, can suffer slightly in lower densities. GPSR's failure in low densities is due to the fact that GPSR's control packets are sometimes

dropped, causing neighbor table entries to expire. For low densities, these erroneous expirations can cause nodes to switch packets to perimeter mode, resulting in routing loops for the reasons shown in Figure 4.3 in Chapter 4.2. Progress Routing, on the other hand, is designed to accommodate incomplete neighbor knowledge and does not suffer from this problem.

Figure 7.3 shows the packet overhead required by the two protocols at different densities. Progress Routing requires significantly fewer packets than GPSR, especially as the network becomes more dense. The savings in overhead is due to the fact that Progress Routing does not need to use beacon packets. In fact, generally, Progress Routing requires approximately 1.5 packets per hop, which is only slightly more than the theoretical minimum of 1 packet per hop. Also, Greedy-Only Progress Routing performs almost identically to Greedy-Perimeter Progress Routing. Thus, the inherent flexibility in choosing nodes on the fly together with Greedy-Only Progress Routing's local flooding property achieves excellent success rates and low overheads even though face-traversal methods are not used; this version of Progress Routing is therefore immune to the problems regarding face-traversal methods.

As shown in Figure 7.4, the cost for using Progress Routing instead of GPSR is a small amount of additional latency. This delay is due to the retransmission timers that cause nodes to wait at each hop, and are generally lower when there are more nodes picking different timers. We consider the added delay a small price to pay for the benefits of reliability and overhead that are obtained in Progress Routing, when compared to GPSR.

7.3 Mobility

Our second study concerned node mobility. Both GPSR and Progress Routing were designed to function well in highly-mobile situations, e.g., the two protocols store only a small amount of state that might be rendered stale as nodes move.

For this study, we used the steady-state random waypoint mobility model [18],

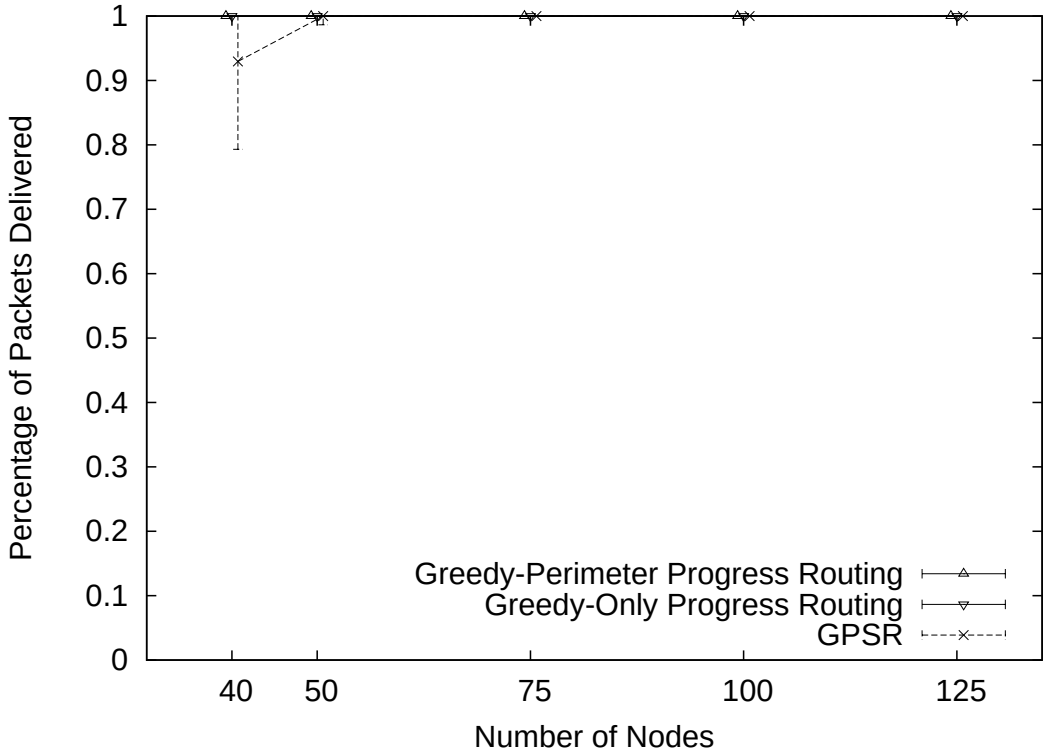


Figure 7.2. Packet delivery rates with varying density.

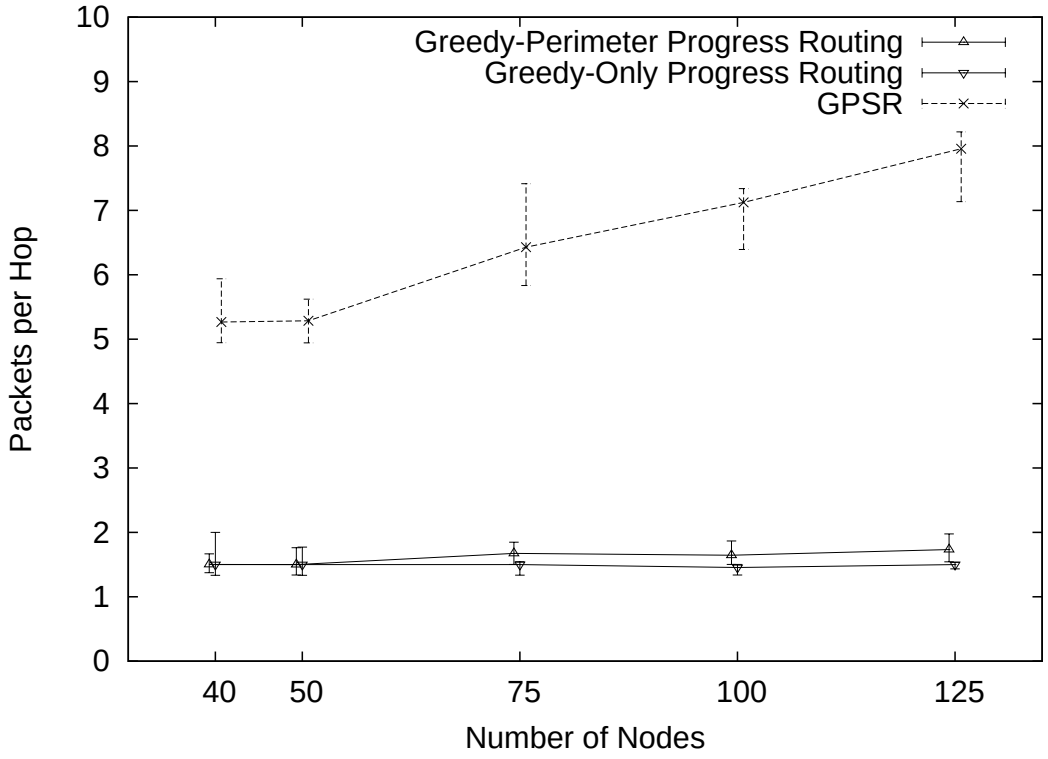


Figure 7.3. Packet overhead with varying density.

with a pause time of 10 ± 1s. We simulated average node speeds of 0, 1, 2, 3, 5, 10, and 15m/s, $\pm 10\%$. We note that 15m/s is approximately 33mph, well outside human walking speed and reasonable for an automobile driving within city limits. As mentioned previously, we ensured that the initial network was not partitioned, there was at least one node in the destination region, and the source node was *not* in the destination region at time zero; we did not ensure these conditions remained true throughout the entire simulation.

Figure 7.5 shows the success rate at the different mobility levels. In the figure, the delivery rates for both versions of Progress Routing exceed GPSR's delivery rate, with Greedy-Only Progress Routing doing slightly better than Greedy-Perimeter Progress Routing. We conclude that GPSR has a harder time than Progress Routing to adapt as neighbor information becomes stale due to movement.

Figure 7.6 shows that, similar to our density study, Greedy-Perimeter Progress Routing uses much less overhead than GPSR; approximately 2.5 packets per hop in the median. Unlike in the density study, however, Greedy-Only Progress Routing uses more packet overhead than GPSR. This fact is because the mobile scenarios are allowed to become temporarily partitioned. Because Greedy-Only Progress Routing executes local floods at local minimum nodes, this version of Progress Routing results in large amounts of uncontrolled flooding in partitioned networks. Given the success of Greedy-Only Progress Routing when the network is connected in the density study, we plan to investigate ways to restrict this uncontrolled flooding in future work.

Figure 7.7 shows that Progress Routing's end-to-end delays are higher than GPSR's end-to-end delays. Because of factors such as local flooding (which happens to a certain extent with both versions of Progress Routing) and transmit delays, sometimes packets can reach the destination despite the network being temporarily partitioned; of course, these packets reach the destination long after they are sent. Additionally, the congestion caused by partitioned networks will also delay packets.

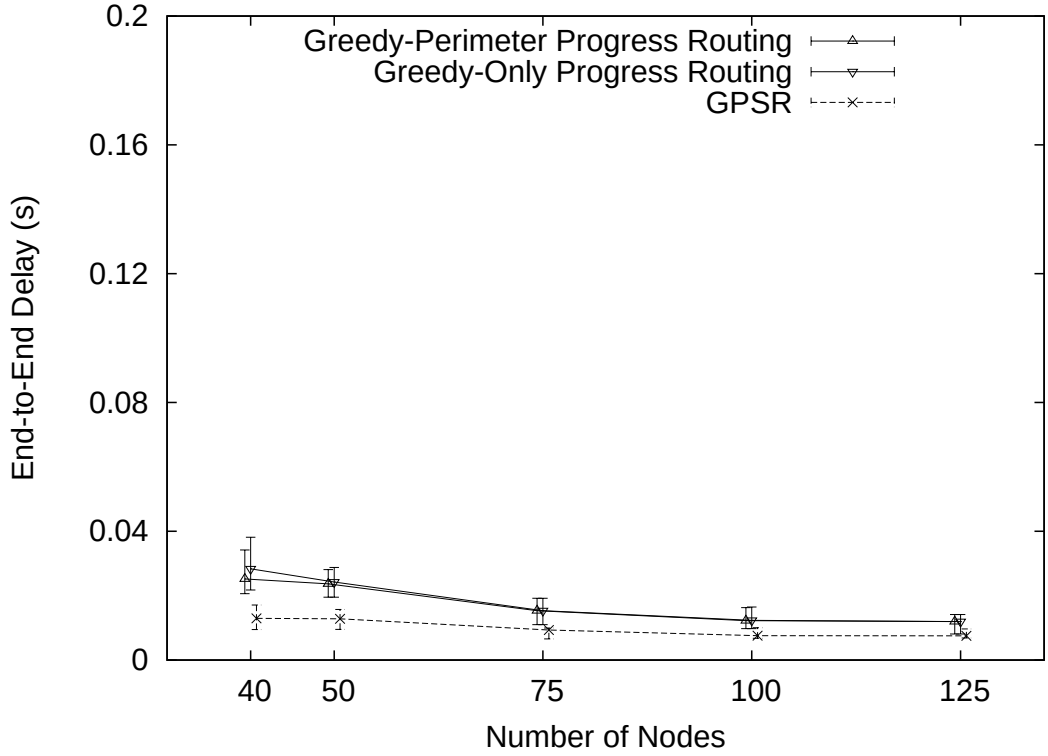


Figure 7.4. End-to-end delays with varying density.

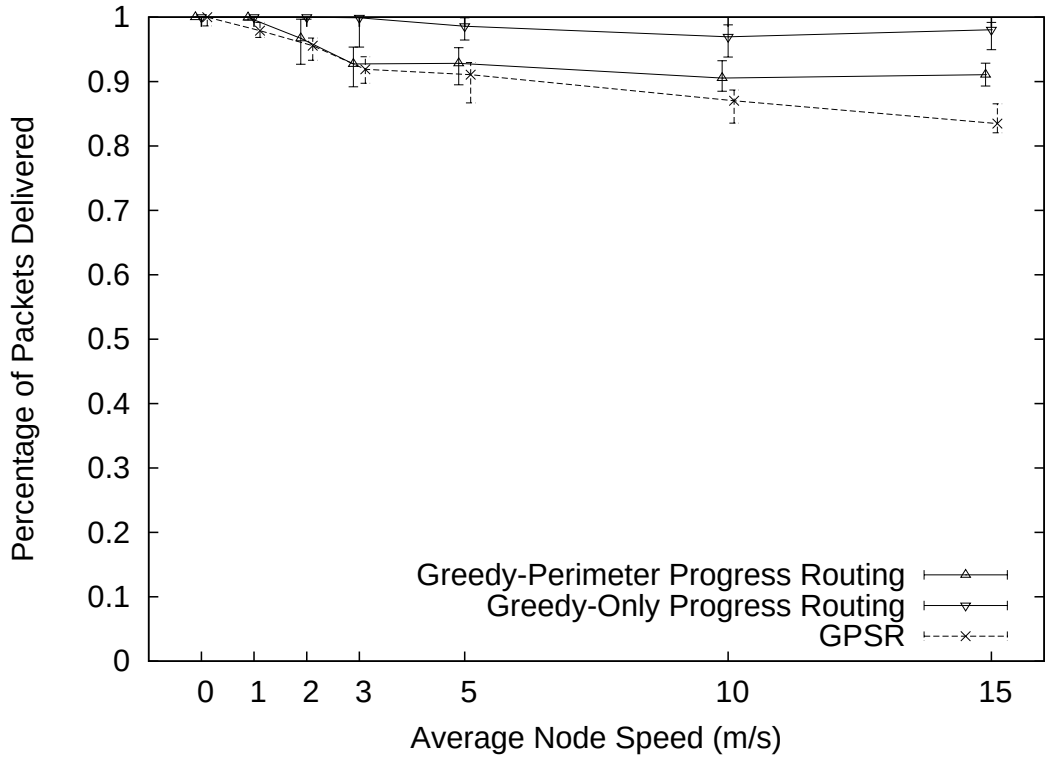


Figure 7.5. Packet delivery rates with varying mobility.

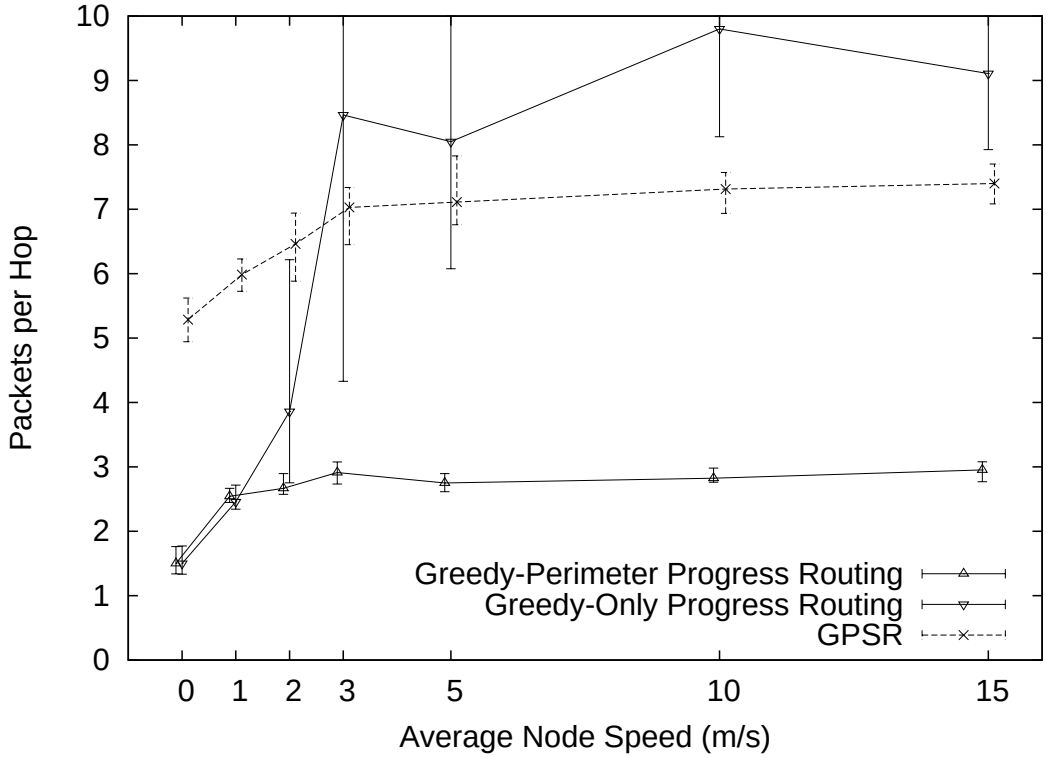


Figure 7.6. Packet overhead with varying mobility.

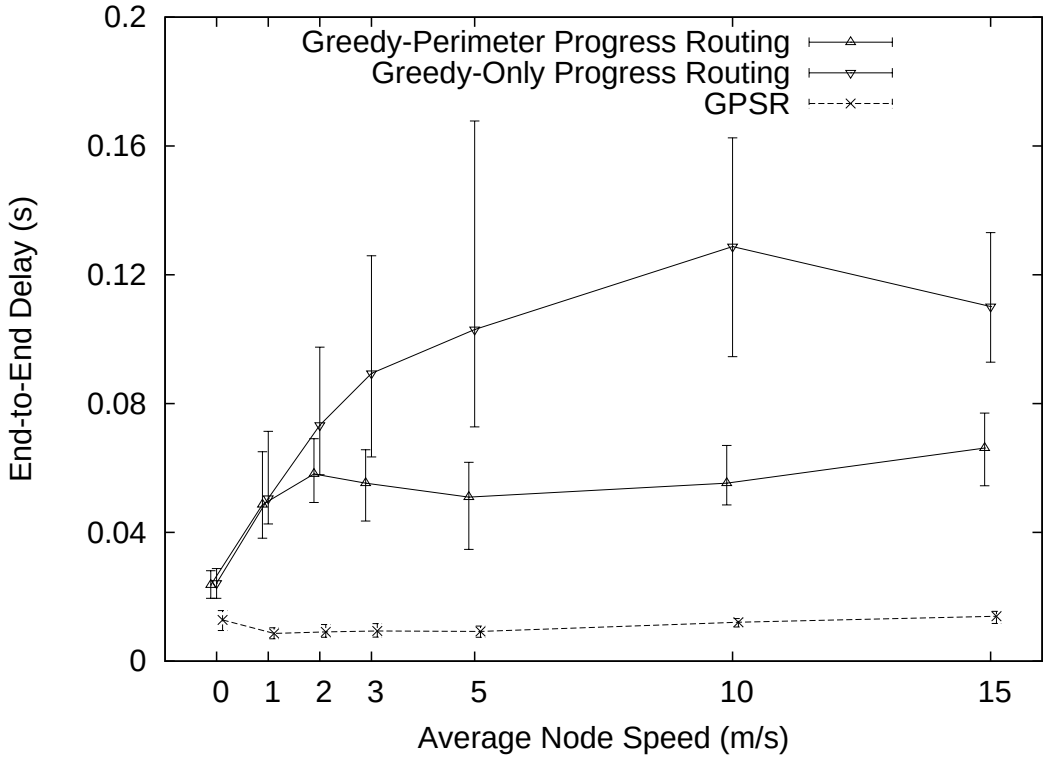


Figure 7.7. End-to-end delays with varying mobility.

7.4 Congestion

Our third study concerned congestion. For this study, we simulated the protocols at 25, 50, 75, 100, 125, and 150 packet originations per second. Later in this section we show that, at 25 packets per second, the end-to-end delay for all protocols is approximately .03s or lower. In other words, there was rarely more than one packet in the network at a time in simulations with 25 packet originations per second. At 150 packet originations per second, end-to-end delays were on the order of hundreds of milliseconds, meaning tens of packets in the network at a time. In summary, we vary packet originations per second to cover a large range of congestion.

As shown in Figure 7.8, all protocols show a steady decline in success rate as congestion increases; Progress Routing, especially Greedy-Perimeter Progress Routing, performs substantially better than GPSR. In particular, for 100 packets per second or less, Greedy-Perimeter Progress Routing obtains higher success rates than GPSR. Thus, not only does Progress Routing use less overhead and obtain better success rates when compared to GPSR, but it supports a higher effective bandwidth of data through the network. We note that GPSR uses only a single path through the network for a given packet and assumes that transmissions are never dropped. Therefore, in GPSR, a queue overflow means a packet will not be delivered. On the other hand, in Progress Routing, a dropped packet will appear to be the result of stale neighbor information; surrounding nodes will compensate, leading to higher delivery rates than GPSR.

Figure 7.9 shows that overhead decreases for GPSR per hop as congestion increases. This phenomenon is due to the fact that GPSR uses a set number of beacon transmissions per second, regardless of the number of original packets sent. Since the number of original packets increases, the ratio of total transmissions (data transmissions plus beacon transmissions) to original packets decreases for GPSR. By contrast, in Progress Routing, increased congestion results in nodes being unable to transmit at the appropriate times, resulting in extra packets. Nonetheless, Progress Routing

Table 7.4. Specific parameters for each of the four severity levels.

Severity Level	Density	Mobility	Congestion
1	100 nodes	1 m/s	25 pkts/s
2	75 nodes	3 m/s	50 pkts/s
3	50 nodes	5 m/s	75 pkts/s
4	40 nodes	10 m/s	100 pkts/s

outperforms GPSR in terms of overhead for all but the most congested networks.

Lastly, as shown in Figure 7.10, GPSR's end-to-end delay quickly becomes unpredictable as congestion increases, whereas Progress Routing's stays stable until approximately 100 packets/second. This further supports our conclusion that Progress Routing can support a higher effective bandwidth than GPSR.

7.5 Severity

Our fourth and final study concerned overall network severity. That is, we tested GPSR and both versions of Progress Routing as the network conditions became more and more difficult for communications, i.e., fewer nodes (thus more local minimum nodes), more mobility (thus more stale information), and higher congestion. We selected four levels of severity; the parameters for the four levels are shown in Table 7.4.

Figure 7.11 shows the success rate of the protocols plotted against severity. As shown in the figure, Greedy-Perimeter Progress Routing has better overall delivery than GPSR as the network conditions become more severe. Of course, as the network severity increases, the delivery rate decreases for all protocols evaluated.

Figure 7.12 shows that Greedy-Perimeter Progress Routing uses less packets per hop when compared to GPSR in all cases except the most severe network studied. Greedy-Only Progress Routing does not perform as well as Greedy-Perimeter Progress Routing, which is similar to the corresponding performance shown in our mobility study. If the greedy-only progress indicator can be modified as discussed in

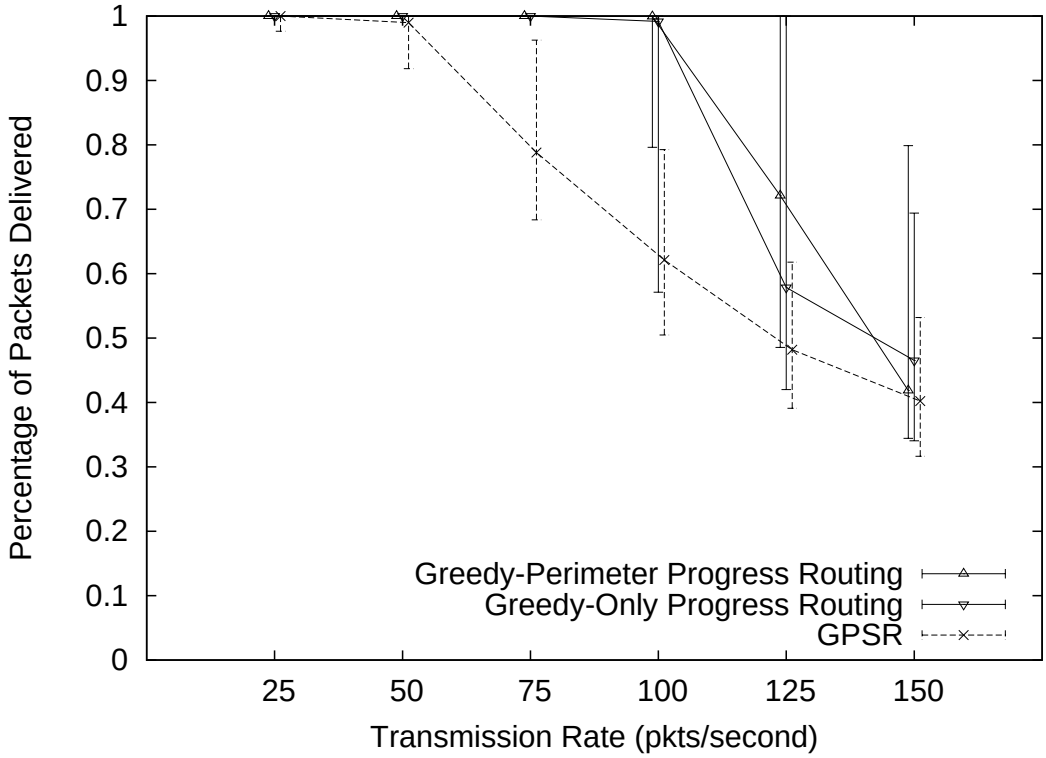


Figure 7.8. Packet delivery rates with varying congestion.

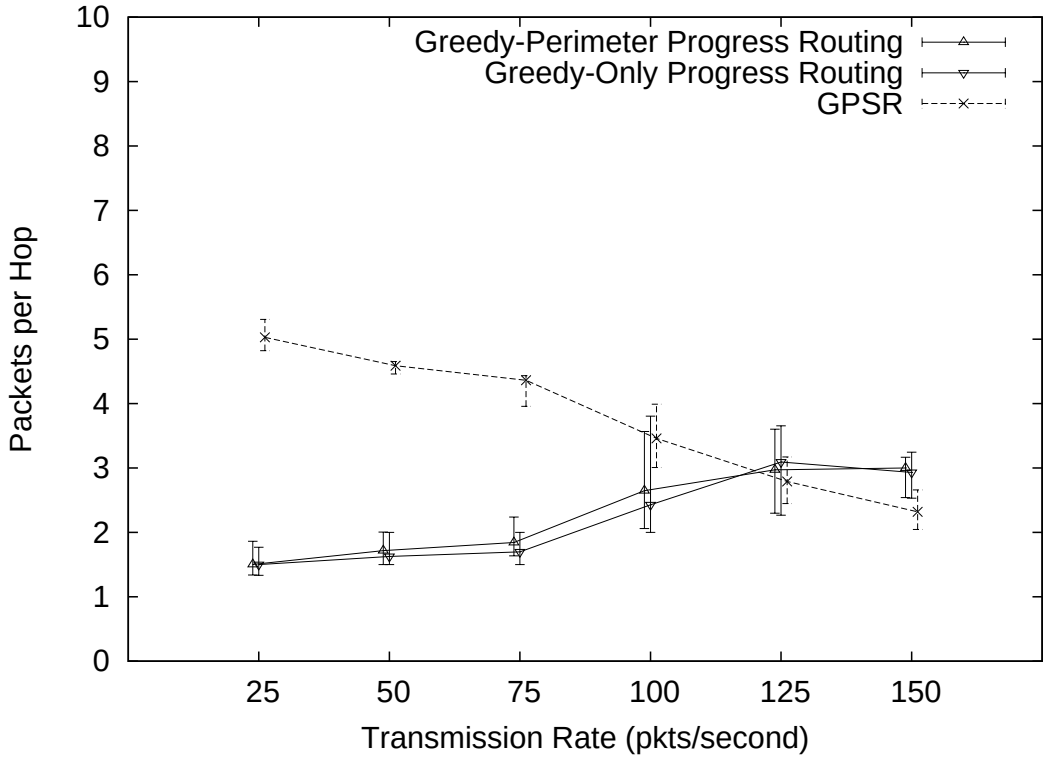


Figure 7.9. Packet overhead with varying congestion.

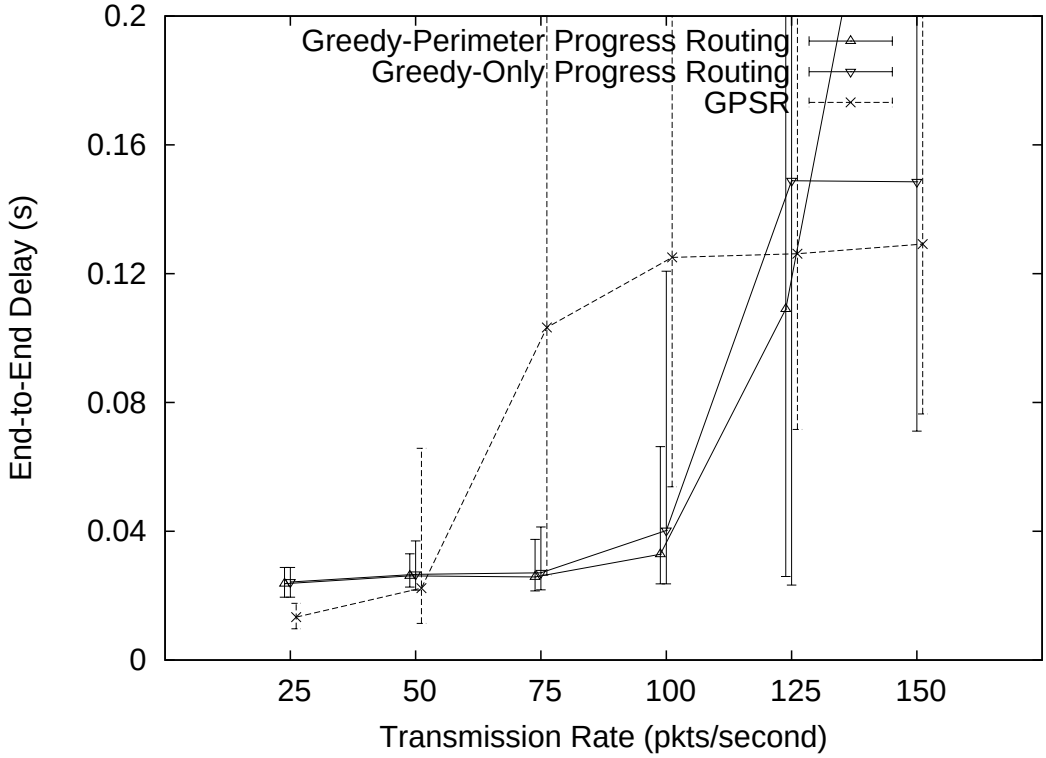


Figure 7.10. End-to-end delays with varying congestion.

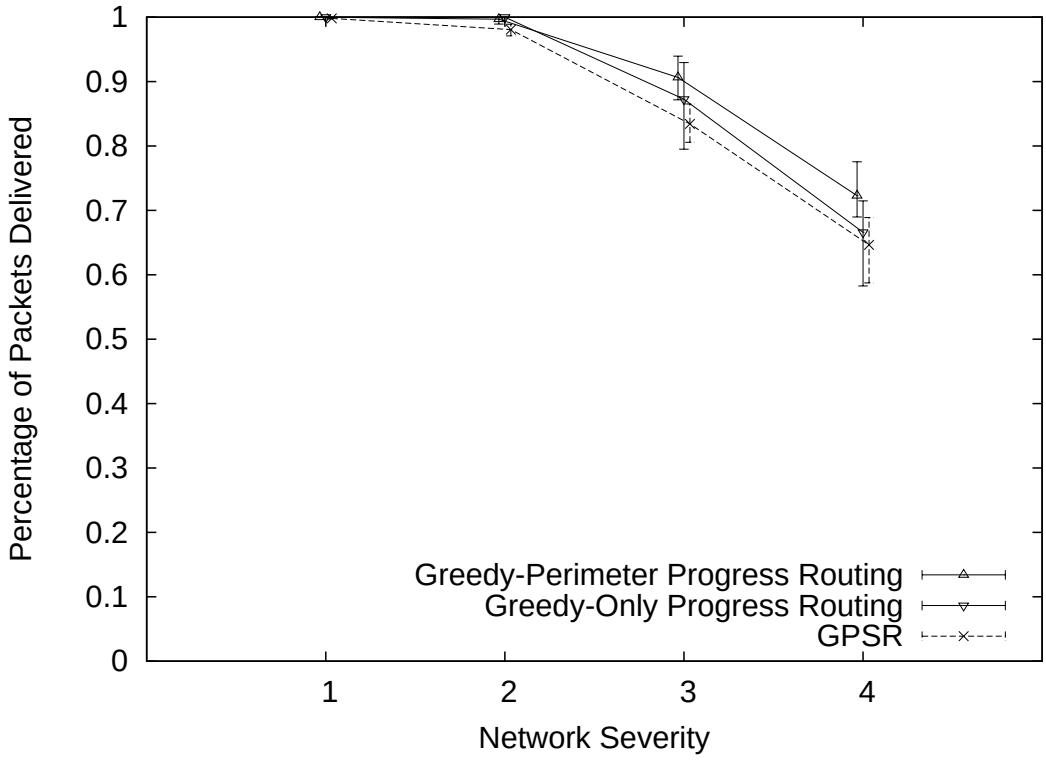


Figure 7.11. Packet delivery rates with varying severity..

Section 7.3, we expect the overhead to decrease in our network severity scenarios as well.

Lastly, as shown in Figure 7.13 the overall cost of Progress Routing continues to be higher latency. The end-to-end delay of both Progress Routing and GPSR remain stable at low severities, and then increase sharply and become less predictable as the network severity level increases.

7.6 Summary of our Simulation Results

In the scenarios we studied, Progress Routing typically delivers packets more often than GPSR and almost always accomplishes this higher performance with less packet overhead. Also, Progress Routing supports a higher effective bandwidth than GPSR. Interestingly, we also found that Progress Routing might be a solution to the problems of face-traversal methods; Greedy-Only Progress Routing performed very well in certain situations, and we believe there are ways to improve it for the situations where it was not as successful.

These results can be extended to other geographic routing protocols. Many of the other geographic routing protocols in the literature are similar to GPSR, i.e., they use a small number of paths, assume no packets dropped, use beaconing, etc. Progress Routing can be configured to emulate these other protocols; we expect to improve their performance as well, because Progress Routing avoids beacons.

Finally, during the execution of our simulations, we learned that while Progress Routing does well in the typical case, there were network scenarios that cause much higher overhead. For example, approximately 8.3% of all Progress Routing simulations had a packets per hop result above 100. These results are primarily due to the fact that Progress Routing does not have any guaranteed maximum number of times a node can transmit a packet. Despite this problem, we still have a solid algorithm that typically achieves excellent results; we expect that it will be possible to separately address the scenarios where Progress Routing does poorly.

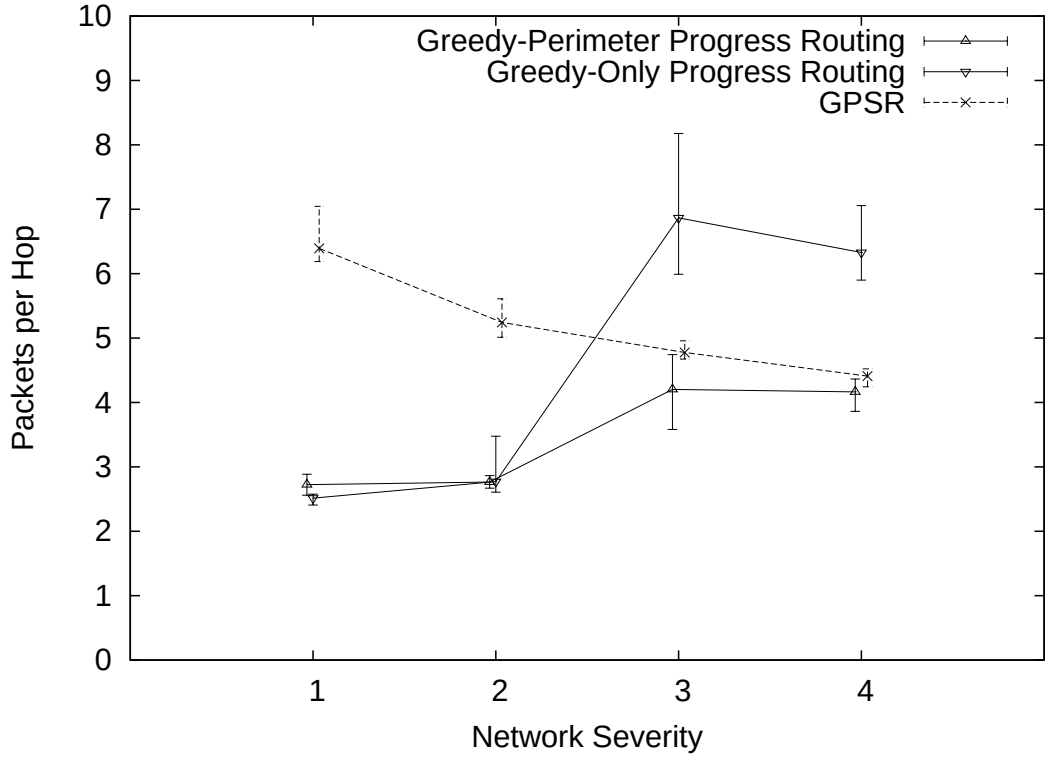


Figure 7.12. Packet overhead with varying severity.

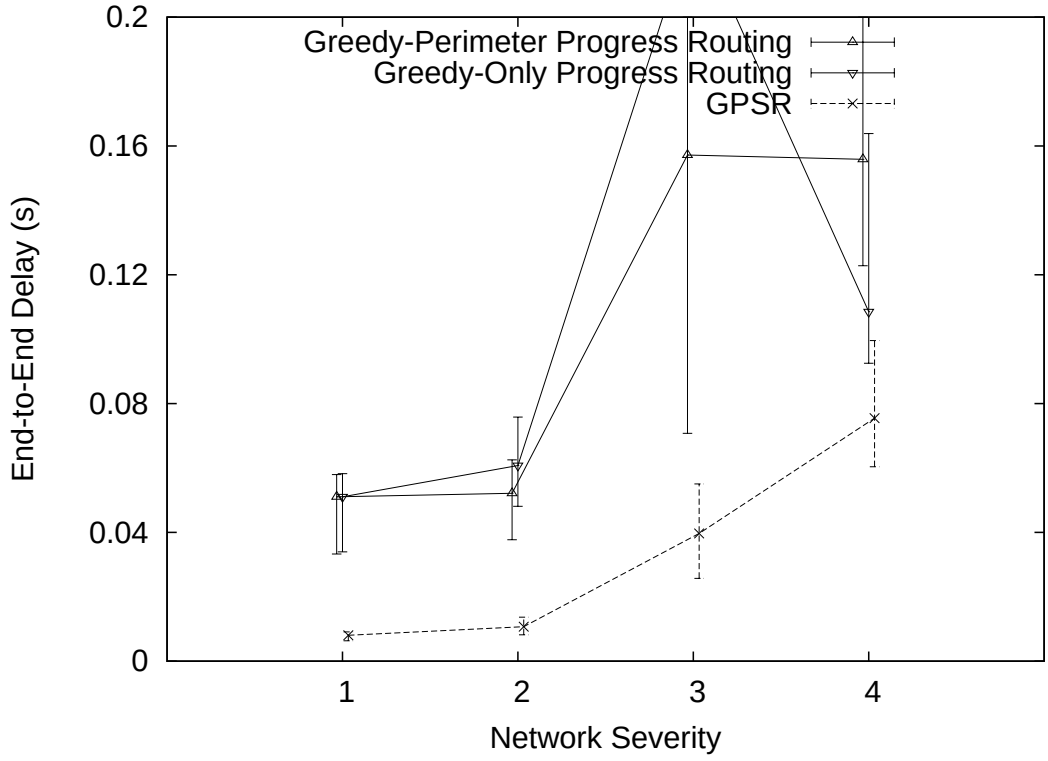


Figure 7.13. End-to-end delays with varying severity..

Chapter 8

CONCLUSION AND FUTURE WORK

This thesis presents a novel method for dividing geographic routing protocols into two independent layers: the progress indicator function/heuristic timing function and the progress routing algorithm. It provides a definition of a progress indicator function in general, which serves to define the interface between these two layers. This division makes it possible to consider the geometry of forwarding decisions separately from the details of ensuring those forwarding decisions are made in an efficient manner. To show the effectiveness of these ideas, we present an example where we derived two progress indicator functions and two heuristic timing functions corresponding roughly to the well-known GPSR protocol. Finally, we show through simulations that our progress indicator functions out-performed GPSR.

As future work, we plan to derive more progress indicator functions and improve upon our progress routing algorithm. For example, we plan to derive progress indicator functions to further limit redundant transmissions and improve Progress Routing's performance in the pathological scenarios where it performs poorly, as well as progress indicator functions to assist propagation in the destination region or assist routing in ideosyncratic networks such as vehicular networks. Also, to aid researchers in incorporating progress indicator functions into their research, we plan to investigate general properties of progress indicator functions such as what classes of designated paths can be reduced to progress indicator functions and which kinds of progress indicator functions lead to more efficient routing. In particular, we plan to determine if we can mitigate the problems that we discovered with the greedy-only progress indicator. We also intend to simulate Progress Routing in a wider variety of networks, including non-unit disk graph networks and networks where nodes are

not all confined to the same plane, and alongside a wider variety of proactive and beaconless geographic routing protocols.

REFERENCES

[1] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *ACM Wireless Networks*, 7(6):609–616, November 2001.

[2] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 85–97, 1998.

[3] S. Datta, I. Stojmenović, and J. Wu. Internal node and shortcut based routing with guaranteed delivery in wireless networks. *Cluster Computing*, 5(2):169–178, April 2002.

[4] H. Füßler, J. Widmer, M. Käsemann, M. Mauve, and H. Hartenstein. Contention-based forwarding for mobile ad hoc networks. *Ad Hoc Networks*, 1(4):351–369, November 2003.

[5] T. He, B. Blum, Q. Cao, J. Stankovic, S. Son, and T. Abdelzaher. Robust and timely communication over highly dynamic sensor networks. *Real-Time Systems*, 37(3):261–289, 2007.

[6] M. Heissenbüttel and T. Braun. A novel position-based and beacon-less routing algorithm for mobile ad-hoc networks. *Proceedings of the IEEE Workshop on Applications and Services in Wireless Networks (ASWN)*, pages 197–210, 2003.

[7] D. Johnson, Y.-C. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for mobile ad hoc networks for IPv4. RFC 4728, February 2007.

[8] H. Kalosha, A. Nayak, S. Rührup, and I. Stojmenović. Select-and-protest-based beaconless georouting with guaranteed delivery in wireless sensor networks. *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 346–350, 2008.

[9] B. Karp. Greedy Perimeter Stateless Routing, GPSR. `http://www.cs.cmu.edu/~bkarp/gpsr/gpsr.html`. Page accessed on July 14th, 2008.

[10] B. Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.

[11] Y.-J. Kim, R. Govindan, B. Karp, and S. Schenker. Geographic routing made practical. *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, pages 217–230, 2005.

[12] R. Kleinberg. Geographic routing using hyperbolic space. *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 1902–1909, 2007.

[13] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: of theory and practice. *Proceedings of the ACM Symposium on Principles of Distributed Computing (POMC)*, pages 63–72, 2003.

[14] F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad-hoc networks beyond unit disk graphs. *Proceedings of the ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 69–78, 2003.

[15] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-case optimal and average-case efficient geometric ad-hoc routing. *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 267–278, 2003.

[16] S. Kurkowski, T. Camp, and M. Colagrosso. A visualization and analysis tool for wireless simulations: iNSpect. *ACM's Mobile Computing and Communications Review*, to appear 2008.

[17] B. Leong, B. Liskov, and R. Morris. Geographic routing without planarization. *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.

[18] W. Navidi and T. Camp. Stationary distributions for the random waypoint mobility model. *IEEE Transactions on Mobile Computing*, 3(1):99–108, January-March 2004.

[19] J. Sánchez, R. Marín-Pérez, and P. Ruiz. Beacon-less geographic routing in real wireless sensor networks. *Journal of Computer Science and Technology*, 23(3):438–450, May 2008.

[20] K. Seada, A. Helmy, and R. Govindan. On the effect of localization errors in geographic face routing in sensor networks. *Proceedings of the ACM International Conference on Information Processing in Sensor Networks (IPSN)*, pages 71–84, 2004.

[21] K. Seada, A. Helmy, and R. Govindan. Modeling and analyzing the correctness of geographic face routing under realistic conditions. *Ad Hoc Networks*, 5(6):855–871, August 2007.

[22] The VINT Project. The network simulator - ns-2. `http://www.isi.edu/nsnam/ns/`. Page accessed on July 14th, 2008.

[23] M. Witt and V. Turau. BGR: blind geographic routing for sensor networks. *Proceedings of the IEEE International Workshop on Intelligent Solutions in Embedded Systems (WISES)*, pages 51–61, 2005.

[24] H. Woesner, H. Gillich, and A. Köpke. The Akaroa2/NS2 Interface. `http://www.tkn.tu-berlin.de/research/ns-2_akaroa-2/ns.html`. Page accessed on July 14th, 2008.

[25] M. Zorzi and R. Rao. Geographic random forwarding (GeRaF) for ad hoc and sensor networks: Energy and latency performance. *IEEE Transactions on Mobile Computing*, 2(4):349–365, October-December 2003.