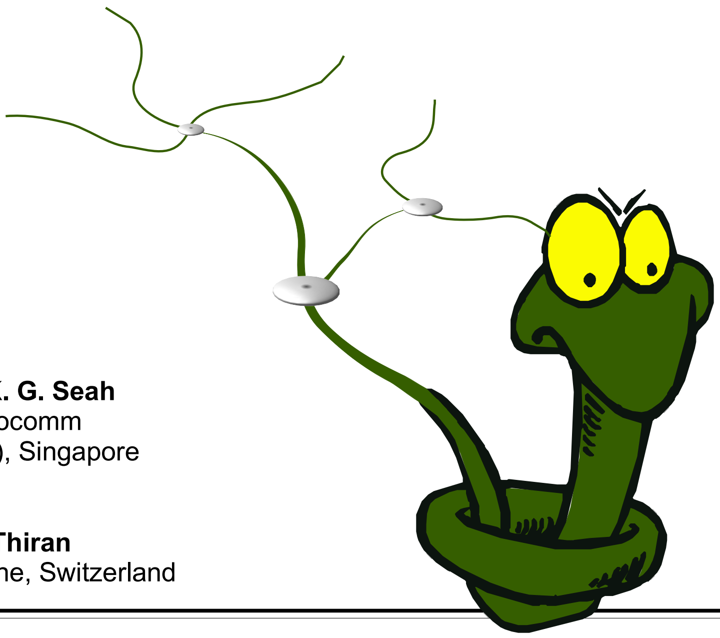
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Master Thesis

Probabilistic Path Discovery with Snakes in Ad Hoc Networks



Supervisor:
Dr. Winston K. G. Seah
Institute for Infocomm
Research (I^2R), Singapore

Examiner:
Prof. Patrick Thiran
EPFL, Lausanne, Switzerland

Thomas Lochmatter
thomas.lochmatter@epfl.ch

Communication Systems
EPFL, Lausanne, Switzerland

September 5, 2005

Abstract

Many routing protocols for wireless ad hoc networks proposed in the literature use flooding to discover paths between the source and the destination node. Despite various broadcast optimization techniques, flooding remains expensive in terms of bandwidth and energy consumption. In general, $\mathrm{O}(N)$ nodes are involved to discover a path.

In this thesis, we prove through a theoretical model that probabilistic path discovery is possible by involving $\mathrm{O}(\sqrt{N})$ nodes only. The constant factor depends on the desired path discovery probability.

Using a novel network primitive that we call *snakes*, we introduce practical and cheap probabilistic path discovery algorithms. These algorithms rely on the same network model and assumptions as its flooding counterparts, i. e., that the network is unstructured and that nodes only know their immediate (one-hop) neighbors. Numerical simulations in a static network show that these algorithms achieve path discovery probabilities close to the theoretical optimum.

We further present a snake-based algorithm for mobile ad hoc networks and several techniques to enhance the performance in some specific networks.

This master thesis was written during an internship from 2005-03-07 to 2005-09-01 at the Institute for Infocomm Research ($\mathrm{I}^2\mathrm{R}$) in Singapore.

Zusammenfassung auf Deutsch:
Probabilistische Pfadsuchalgorithmen mit Schlangen in Ad-Hoc-Netzwerken

Viele der vorgeschlagenen Routing-Protokolle für Ad-hoc-Netzwerke verwenden sogenanntes Flooding (Überschwemmen des Netzwerkes) um einen Pfad zwischen einem Quell- und einem Zielknoten zu finden. Obwohl in der wissenschaftlichen Literatur verschiedene Optimierungen erforscht wurden verschlingt Flooding viele Netzwerkressourcen (insbesondere Übertragungsbandbreite und Energie), da normalerweise $O(N)$ Knoten kontaktiert werden müssen um einen Pfad zu finden.

In dieser Arbeit zeigen wir anhand eines theoretischen Modells, dass Pfade probabilistisch durch Kontaktieren von nur $O(\sqrt{N})$ Knoten gefunden werden können. Der konstante Faktor hängt dabei von der gewünschten Wahrscheinlichkeit ab.

Mit Hilfe einer neuen Netzwerk-Funktion, die wir *Snake* (Schlange) nennen, stellen wir einfach implementierbare, probabilistische Pfadsuchalgorithmen vor. Diese Algorithmen basieren auf denselben Hypothesen, die auch für Flooding-Algorithmen notwendig sind. Insbesondere heisst dies, dass das Netzwerk unstrukturiert ist und die Knoten nur mit ihren direkten Nachbarn kommunizieren können. Wie numerische Simulationen in einem statischen Netzwerk zeigen, finden diese Algorithmen Pfade mit nahezu optimaler Wahrscheinlichkeit.

Im Weiteren präsentieren wir einen probabilistischen Pfadsuchalgorithmus für mobile Ad-hoc-Netzwerke sowie mehrere Optimierungsvarianten.

Resumé en français:
Algorithmes "serpents" de recherche de routes dans les réseaux ad hoc

De nombreux protocoles de routage des réseaux ad hoc utilisent le flooding (noyer le reseau) pour découvrir des chemins entre deux noeuds. Malgré plusieurs optimisations existantes dans la littérature, le flooding reste un moyen coûteux, notamment parce qu'il demande une grande bande passante et beaucoup d'énergie. En général, $O(N)$ noeuds doivent être contactés pour chercher une route.

Dans cette thèse, nous prouvons que la recherche probabiliste de routes nécessite de contacter $O(\sqrt{N})$ noeuds seulement. La constante dépend de la probabilité souhaitée.

En utilisant une nouvelle fonction de réseau que nous appelons *snake* (serpent), nous introduisons des algorithmes peu coûteux pour découvrir des routes de manière probabiliste. Ces algorithmes sont basés sur le même modèle que les algorithmes de flooding, c'est-à-dire, les réseaux sont considŕś comme non structurés et chaque noeud connaît uniquement ces voisins directs. Nos simulations numériques dans un réseau statique montrent que ces algorithmes atteignent une probabilité de découverte de routes proche de la limite théorique.

De plus, nous présentons des algorithmes pour des réseaux ad hoc mobiles ainsi que plusieurs techniques d'optimisation.

Contents

Chapter 1 Introduction

Wireless networks are an integral part of our daily life. Probably the most visible witness is the mobile phone which has become popular almost everywhere in the world. But wireless networks have found far more applications: TV programs are broadcast wirelessly from satellites or terrestrial antennas, modern PDAs are equipped with Bluetooth [1] and WLAN 802.11 [2] to synchronize data or to be connected to the Internet, pilots communicate with the tower staff over a wireless link and weather balloons transmit temperature recordings back to the earth. Wireless networks are used in a vast variety of fields.

Most wireless networks nowadays are infrastructure networks. This means that some infrastructure (e. g. base station for mobile phones, WLAN access point) is required for the network be operational. Less common, but also available, are so-called *ad hoc networks* [3] [4] that operate without any infrastructure. In these networks, two devices start communicating as soon as they are within each others reach. They are therefore also said to be self-organizing, since there is no central device which organizes the network. The WLAN 802.11 protocol [2], for example, contains an ad hoc mode which allows two laptops to directly exchange data without being connected to a WLAN access point.

The most interesting, but also the most challenging type of ad hoc networks are *multi-hop* ad hoc networks. In such networks, each device – called *node* – acts as a router and forwards messages (packets) on behalf of other nodes. Figure 1.1 shows a sample network where node **A** sends a packet to node **D** via the intermediate hops **B** and **C**. For this to work, all nodes must have previously agreed upon a forwarding strategy, a so-called *routing protocol*.

Ad hoc networks gained interest because of their usefulness in disaster recovery (when the infrastructure of other networks has been destroyed) or military operations (when no infrastructure is available). The potential deployment range spans much wider, though. Beyond (ad hoc) sensor networks which are widely used already, multi-hop ad hoc networks could in the future extend the range of base stations for mobile telephony [5].

Over the past 10 years, there has been a lot of ongoing research in the field of ad hoc networking. Even though they were initially thought of as a simple enhancement of wired networks or wireless infrastructure networks, ad hoc networks have shown to be much more challenging.

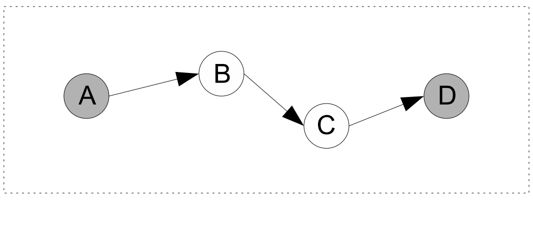


Figure 1.1: *Communication in a multi-hop ad hoc network. The source node **A** send its message for **D** to **B**. **B** forwards the message to **C** which forwards it to the intended destination node **D**.*

Many related problems and questions are still not solved.

One of the problems in large scale ad hoc networks with thousands or millions of nodes is the scalability of the routing protocol. Imagine you want to find one specific person in a crowd of 10000 people. The only thing you know about this person is its (unique) name. And the only thing you can do in this crowd is to talk to your neighbors. Transmitting a message to that specific person without bothering too many people is indeed a challenge – a challenge that we face in this master thesis.

1.1 Routing Protocols and Path Discovery

Routing protocols for multi-hop ad hoc networks have gained a lot of attention by researchers. Many protocols and variants have been defined and analyzed. We do not intend to present or classify them here, as there are good overview papers [6] [7] [8] and websites [9] [10] available.

Almost all routing protocols can be split up in 4 parts:

Path discovery consists of finding a path from *any* source node to *any* destination node. (In wireless sensor networks often: from *any* source node to *one or several predefined* destination nodes.)

Path setup is required to make the path usable. This step is closely linked with the path discovery phase and is sometimes performed in the same step. AODV [11], for example, sets up the path when the route reply (RREP) packet is sent back to the source.

Routing of data packets. This describes how each node forwards data packets. In AODV, for example, the packet is forwarded to the next hop found in the local routing table. In DSR [12], the next hop is found in the data packet itself.

Path maintenance is the way a routing protocol deals with broken or suboptimal paths. Paths may break or become suboptimal due to link failure (mobility, RF propagation changes, ...) or node failure (run out of energy, power off, software/hardware failure, ...).

This master thesis focuses on the path discovery phase of routing protocols. Path discovery consists of all necessary steps to find a path between two nodes. This includes the proactive exchange of routing information (if any) as well as the algorithm (if any) that is applied reactively when a path is needed.

The algorithm we present in this work may also be used for the path maintenance part, as this part usually consists of rediscovering a path or a part of it. We however do not explicitly address this issue in our work since the application of our algorithms in the path maintenance phase is straightforward.

1.2 Related Work

Many routing protocols make use of broadcasting techniques (flooding) to discover and maintain multi-hop paths between nodes. In the well studied AODV [11] routing protocol, for example, the source node broadcasts a route request packet (RREQ) to all nodes in the network during the path setup phase. Similarly, DSR [12] floods the network if no valid route to a desired destination is in the route cache.

Flooding is expensive from two perspectives. First, it involves all or a substantial part of the nodes in the network to receive, process and possibly send a packet. This consumes valuable energy. Second, the large number of transmitted packets eat up a lot of network capacity. This reduces the data throughput.

Many attempts have been made to alleviate this problem.

- ▷ Several **optimized broadcasting techniques** [13] have been proposed. Most of them try to reduce the number of packets transmitted to broadcast routing information to relevant nodes in the network. The OLSR [14] routing protocol, for example, uses multi-point relays (MPR) to lower the number of packets.
- ▷ **Expanding Ring Search (ERS)** [15] adds a TTL (time-to-live) value to the broadcast packet to limit the flood to a certain number of hops. For path discovery, a node first broadcasts a packet with a low TTL value, limiting the broadcast to the close neighborhood only. If there is no reply from the destination node, a new packet with a slightly higher TTL value is broadcast and the TTL is increased until the destination node is found or a maximum TTL value is reached. ERS therefore only involves nodes that are closer (hop count) or not much further away than the destination node. Although this scheme greatly reduces the spread of the flood if the two nodes are close to each other, it is more expensive than simple flooding for distant nodes.
- ▷ **Multi-step flooding** [16] uses the fact that multiple small floods (restricted to a small area) are cheaper than one big flood. ERS is used to look for a node that recently encountered the destination node. Once such a node is found, it launches ERS again to look for a fresher encounter. This procedure is repeated until the destination node is found. Multi-step flooding therefore guarantees a path to be found if the source and the destination node are connected, even though the discovered path may be suboptimal. If the encounters are well distributed over the network, this algorithm substantially reduces the number of nodes involved in path discovery. In the worst case, however, this scheme is more expensive than simple flooding.
- ▷ LAR [17], a geographic routing protocol, **reduces the flooding area** by directing the broadcast towards the destination node. The amount of involved nodes and sent packets during path discovery can be lowered significantly. However, the protocol relies on the assumption that position information is available and shared among the nodes.
- ▷ In [18] and [19], the authors propose scalable **geographic node location** schemes. Each node sends its position to a couple of well-chosen nodes (location servers) in the network. To find a path, a node first contacts a location server of the destination node to obtain the current position of the destination. Then, geographic forwarding is used to discover a path. All proposed schemes require the node positions to be known.
- ▷ In **proactive routing protocols** such as DSDV [20], nodes send topology updates to their direct neighbors only instead of flooding the whole network. Beyond the classical problem that these routing protocols become very expensive in the presence of node mobility, they require large routing tables to be stored on every node. Each node basically has one routing entry for every other node, thus requiring memory in $O(N)$, where N denotes the number of nodes in the network.
- ▷ Finally, **cluster or zone based methods** have been proposed. In the Zone Routing Protocol (ZRP) [21], for example, each node has a zone within which it uses an intrazone routing protocol (IARP). To route between zones, the interzone routing protocol (IERP) is employed. Thus, only a subset of the nodes must be flooded for path discovery. Since the zones size

is almost constant, however, the number of involved nodes is only a constant factor smaller than the total number of nodes. Other routing protocols that belong to this class are the Core Extraction Distributed Ad Hoc Routing Protocol (CEDAR) [22] and the Zone-based Hierarchical Link State Routing Protocol (ZHLS) [23].

Furthermore, there are strategies to reduce the number of times flooding is needed:

- ▷ **Local repair.** The authors of [24] propose to exploit path locality and node locality to repair a route.
- ▷ **Path caching.** DSR [12] caches additional paths that are found during the path discovery phase and tries these alternative paths when the main path breaks.
- ▷ AntHocNet [25], exploits **stigmergic information** in the network to find paths to destination nodes that have recently been searched by other nodes in the network.

1.3 Our Contribution

In this report, we propose and analyze a class of probabilistic path discovery algorithms that do not require flooding. These algorithms use *snakes* (Chapter 5) and are based on the RANDOM-ENCOUNTER concept (Chapter 2). In contrast to most flooding-based algorithms, our algorithms are probabilistic, i. e., path discovery may fail with some small (tuneable) probability. Moreover, the discovered paths are suboptimal. However, we show that our algorithms allow for much cheaper path discovery and are therefore more scalable. In particular, we prove in Chapter 3 that path discovery is possible by involving $O(\sqrt{N})$ nodes only.

1.4 Structure of this Thesis

This remainder of this thesis report is structured as follows: In Chapter 2, we introduce the RANDOMENCOUNTER concept which forms the basis of our work. In Chapter 3, we present a theoretical algorithm implementing this concept and analyze it analytically. Chapter 4 gives an alternative analysis that is slightly more complicated, but comes up with the same results. In Chapter 5, we introduce a network primitive called *snakes*. These snakes are the main building blocks for the algorithms presented subsequently. In Chapter 6 and Chapter 7, we present and discuss snake based algorithms for static and mobile networks. Chapter 8 gives an overview of three possible enhancements for snake algorithms. In Chapter 9, we tackle the problem from another point of view using a geometric approach. In Chapter 10, we finally discuss some possible application scenarios where snake algorithms could be useful. Chapter 11 concludes the report.

Our main contribution is stated in Chapter 2, Chapter 3, Chapter 5 and Chapter 6. Chapter 7 and Chapter 8 remain at the level of a preliminary study and would require more work.

Chapter 2

The RANDOMENCOUNTER Concept

All algorithms presented in this work are based upon one common concept which we refer to as the RANDOMENCOUNTER concept. It may be expressed as follows: Each node in the network randomly advertises itself at some other nodes. An advertisement contains enough information to establish a path from the node that receives the advertisement to the advertising node. A node **S** trying to establish a connection to another node **D** randomly contacts some nodes in the network to look for an advertisement of **D**. Upon encounter, i. e., if an advertisement is found, a path has been discovered.

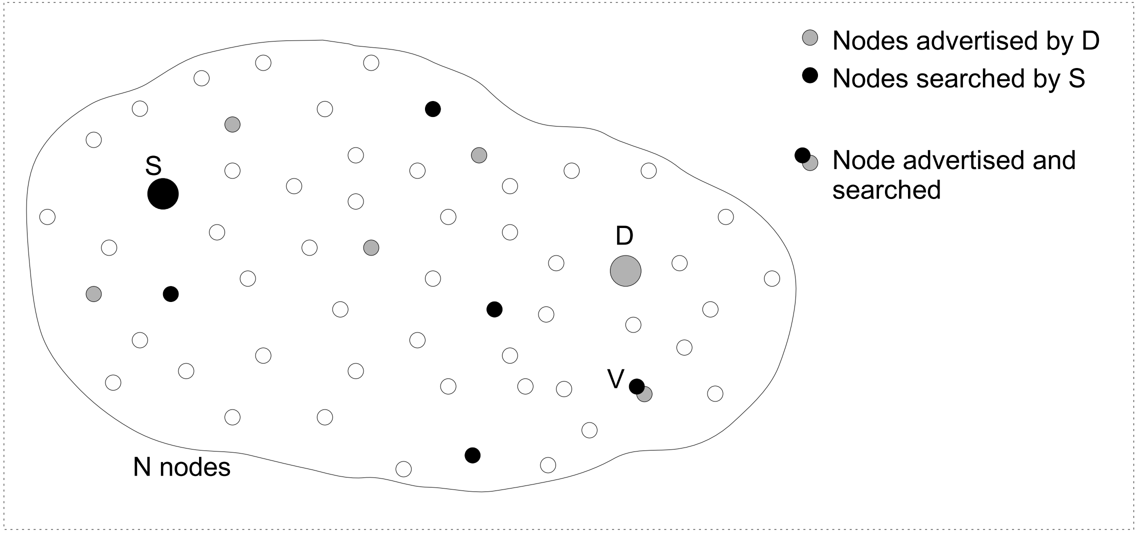


Figure 2.1: *The* RANDOMENCOUNTER *concept. In this example, 5 nodes were advertised by* ***D*** *and 5 nodes were queried by* ***S****. The path discovery was successful since node* ***V*** *was both queried and advertised.*

This concept is very general. It does not specify how advertisements are placed, what information they contain and how they are searched. Nevertheless, three properties can be stated:

- ▷ **Probabilistic path discovery.** Paths discovery fails with some probability even if a path exists.
- ▷ **Suboptimal paths.** The discovered path may be longer than the optimal path. No guarantee about path optimality is given.
- ▷ **Fully distributed.** All nodes have the same esteem and importance. Neither are there leader nodes nor is a centralized infrastructure used.

Furthermore, all algorithms based on the RANDOMENCOUNTER concept inherently have two parts:

- ▷ The **proactive part** specifies how advertisements are placed (**advertising algorithm**) and when they are placed.
- ▷ The **reactive part** specifies how advertisements are searched (**search algorithm**) and how the path is discovered.

Hence, routing protocols using a path discovery algorithm based on this concept belong to the class of hybrid protocols.

2.1 Minimum Information of an Advertisement

Depending on the scenario, an advertisement does not need to contain a path to the advertising node. It could consist of a geographic position, for example, or just contain some hint which allows to find the destination node. The following theorem states the sufficient and necessary condition for the minimum information that an advertisement must contain:

Theorem 1. An advertisement must contain enough information to establish a path from the node **V** holding the advertisement to the advertising node **D**.

Proof. Assume a node **S** wants to discover a path to **D**. If **S** finds an advertisement at node **V**, there exists a path from **S** to **V** since these two nodes could communicate with each other. By the definition, a path from **V** to **D** exists as well. Hence, the path **S** - **V** - **D** exists, which proves sufficiency.

Necessity can be proven by contradiction: Assume that the advertisement does not enable a path from **V** to **D** to be established. If **S** = **V**, **S** finds the advertisement in its own memory. By the assumption taken, it is not able to establish a path to **D** - a contradiction. □

Chapter 3
The RandomQuery Algorithm

In this chapter, we derive the mathematical foundations of the Random Encounter model. In particular, we show how the path discovery probability relates to the number of advertisements in the network and to the number of searched nodes. To do that, we introduce the RandomQuery algorithm.

3.1 Algorithm

Assume a network of N connected[1] nodes and a function *QueryNode*(node, $\langle$query$\rangle$) which sends the given query to the node. The algorithm works as follows:

Advertising algorithm (proactive): At network setup, each node advertises itself by randomly placing advertisements at some other nodes. The probability that a node **D** places an advertisement at node **W** is denoted by p_A. On average, each node therefore places $\bar{a} = p_A N$ advertisements in the network where a follows a binominal distribution.

Search algorithm (reactive): A node **S** looking for another node **D** randomly queries some nodes in the network. The probability that **S** searches at **U** is denoted by p_S. Therefore, $\bar{s} = p_S N$ nodes are searched on average. If **S** finds at least one advertisement of **D**, we define that the path discovery was successful.

Figure 3.1 formally defines these two algorithm.

Note that the RandomQuery algorithm is theoretical and very generic. We are neither interested in how the *QueryNode* function could be implemented nor in what information the advertisement should contain in order to make path establishment possible.

3.2 Path Discovery Probability

The probability that a node is advertised by **D** and searched by **S** is

$$p_A p_S = \frac{\bar{a}\bar{s}}{N^2}$$

Hence, the probability that no path is discovered, $q = 1 - p$, is

$$q = (1 - p_A p_S)^N = \left(1 - \frac{\bar{a}\bar{s}}{N^2}\right)^N \tag{3.1}$$

This equation reveals that the model is symmetric. Advertising nodes with probability p_A and searching with probability p_S yields the same path discovery probability as advertising with p_S and searching with p_A. Hence p_A and p_S (or $\bar{a}$ and $\bar{s}$) are interchangeable.

[1]Two nodes are connected if there exists a multi-hop path between them.

```
uses $QueryNode$(node, ⟨Advertise(node)⟩ )
uses bool = $QueryNode$(node, ⟨Search(node)⟩ )

function Advertise($p_A$)
    for each node in the network
        with probability $p_A$
            $QueryNode$(node, ⟨Advertise(Me)⟩ )
        end with
    end for
end function

function success = Search($p_S$, findnode)
    for each node in the network
        with probability $p_S$
            if $QueryNode$(node, ⟨Search(findnode)⟩ )
                return true
            end if
        end with
    end for
    return false
end function
```

Figure 3.1: *The advertisement and search functions in the* RANDOMQUERY *algorithm.*

3.3 Asymptotic Path Discovery Probability

For large networks, we can derive the following asymptotic behavior of equation (3.1):

$$\begin{aligned}
\lim_{N\to\infty}\left(1-\frac{\bar{a}\bar{s}}{N^2}\right)^N &= q \\
\lim_{N\to\infty}\left(\left(1-\frac{\bar{a}\bar{s}}{N^2}\right)^{N^2}\right)^{\frac{1}{N}} &= q \\
\lim_{N\to\infty}\left(\mathrm{e}^{-\bar{a}\bar{s}}\right)^{\frac{1}{N}} &= q \\
\lim_{N\to\infty}\mathrm{e}^{-\frac{\bar{a}\bar{s}}{N}} &= q
\end{aligned}$$

Hence,

$$\boxed{\frac{\bar{a}\bar{s}}{N} = -\ln q \qquad \text{for } N\to\infty} \tag{3.2}$$

This is the central equation describing the performance of the RANDOMENCOUNTER concept. Let us discuss some properties and corollaries of this result:

▷ Numerical evaluation shows that equation (3.2) is a good approximation and an upper bound for q if $N \geq 100$ and $q \geq 10^{-6}$, i. e., for finite N, this equation gives a good lower performance bound.

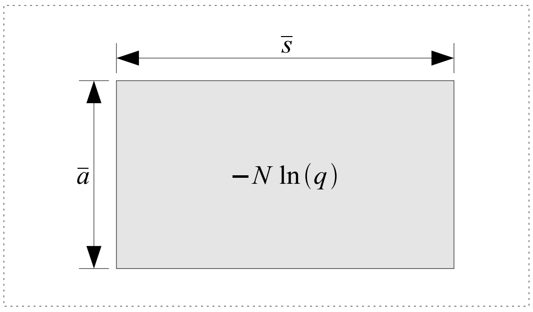


Figure 3.2: *Graphical representation of equation (3.2).*

- ▷ The expression $\frac{\bar{a}\bar{s}}{N}$ only depends on the path discovery probability $p = 1-q$. Hence, for a fixed network size N, the path discovery probability is a function of the product $\bar{a}\bar{s}$. A simple intuition for this is shown in Figure 3.2. If $\bar{a}$ and $\bar{s}$ are the dimensions of a rectangle, its area is an indicator for the probability. The number of nodes involved in path discovery is $\bar{a}+\bar{s}$, which is half the circumference of the rectangle.

- ▷ The best performance, i. e., the least number of nodes to obtain a certain probability, is obtained if $\bar{a} = \bar{s}$. This is quite obvious, since a square minimizes the circumference with respect to its area. To achieve a given path fail probability, q,

$$\bar{a} = \bar{s} = c\sqrt{N} \qquad \text{where } c = \sqrt{-\ln q} \tag{3.3}$$

  Hence,

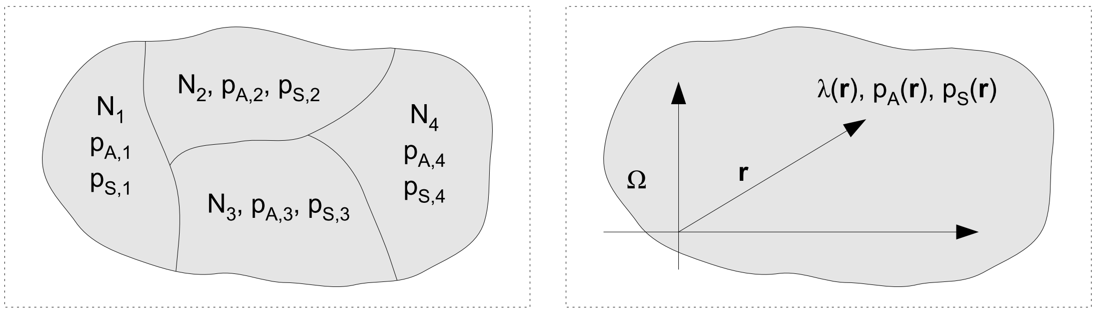
$$\begin{array}{lcll} \bar{a} & \in & \mathrm{O}(\sqrt{N}) & \text{and} \\ \bar{s} & \in & \mathrm{O}(\sqrt{N}) & \text{for } N \to \infty \end{array}$$

  with the remaining constant depending on the desired path discovery probability only. Some values for c are listed in Table 3.1.

This is a very promising result when compared to flooding which requires all N nodes to be involved in path discovery. However, it is only valid if the advertisements and the searched nodes are uniformly distributed over the whole network. This is difficult to achieve in an ad hoc network unless each node knows a substantial part about the whole network. Hence, the major challenge is to find practical and cheap algorithms that obtain distributions close to a uniform distribution. We describe such algorithms in Chapter 6.

p	q	$c^2 = -\ln q$	$c = \sqrt{-\ln q}$
90 %	0.1000	2.3026	1.5174
95 %	0.0500	2.9957	1.7308
98 %	0.0200	3.9120	1.9779
99 %	0.0100	4.6052	2.1460
99.9 %	0.0010	6.9078	2.6283
99.99 %	0.0001	9.2103	3.0349

Table 3.1: *Path discovery probability constants to calculate $\bar{a}$ and $\bar{s}$.*



(a) (b)

Figure 3.3: *(a) Partitioning the network into homogeneous areas for equation (3.4). (b) The model for equation (3.7).*

3.4 Heterogeneous Densities

3.4.1 Discrete Case

Equation 3.1 can easily be generalized for discrete heterogeneous advertisement/search distributions. If we partition the network into m areas with constant p_A and p_S parameters (see Figure 3.3 (a)) the path discovery failure equation can be written as

$$q = \prod_{i=1}^{m} (1 - p_{A,i} p_{S,i})^{N_i} \tag{3.4}$$

3.4.2 Continuous Case

For continuous heterogeneous advertisement/search distributions, we can write equation (3.1) in log notation:

$$\ln q = N \ln(1 - p_A p_S) \tag{3.5}$$

To integrate over an area, we need to replace the number of nodes N by the node density λ and the area size A. Note that the number of nodes then becomes a Poisson distributed random variable which we denote by $N_{\text{r.v.}}$. Hence, the path discovery probability can be expressed as

$$\begin{aligned} \ln q &= \sum_{n=0}^{\infty} \mathrm{P}(N_{\text{r.v.}} = n) \cdot n \ln(1 - p_A p_S) \\ &= \bar{N} \ln(1 - p_A p_S) \qquad \text{where } \bar{N} = \mathrm{E}[N_{\text{r.v.}}] = A\lambda \\ &= A\lambda \ln(1 - p_A p_S) \end{aligned} \tag{3.6}$$

This expression can now be integrated over the network area Ω. Formally,

$$\ln q = \iint_{\Omega} \lambda(\mathbf{r}) \ln(1 - p_A(\mathbf{r}) p_S(\mathbf{r}))\ \mathrm{d}\mathbf{r} \tag{3.7}$$

3.5 Example

To get a intuition for these equations and a feeling for the achievable gain, let us discuss some numerical examples.

Assume we have a network with $N = 1398$ nodes[2] and we want to obtain a path discovery probability of 99 %. Using equation (3.2), we obtain the following approximation for the $\bar{a}\bar{s}$ product:

$$\bar{a}\bar{s} \approx -N \ln q = 1398 \cdot 4.6052 = 6438 \qquad \text{since } q = 1 - p = 0.01$$

In the optimal case,

$$\bar{a} = \bar{s} \approx \sqrt{-N \ln q} = 80.24 \text{ nodes}$$

Hence, if we place 81 advertisements on average in the network and search at 81 nodes on average, we expect to discover paths with more than 99 % probability. We can verify this using equation (3.1):

$$q = \left(1 - \frac{81 \cdot 81}{1398^2}\right)^{1398} = 0.0091 \qquad \Longrightarrow \qquad p = 99.09\ \%$$

In a network with 1398 nodes, contacting 162 nodes is therefore sufficient to discover path with that high probability. This is less than 12 % of the network size. Since equation (3.2) is an upper bound for the $\bar{a}\bar{s}$ product and because we rounded the exact values for $\bar{a}$ and $\bar{s}$ up to the next integer, the performance is even slightly better than desired.

We could have chosen to advertise more nodes to reduce the number of nodes to search at. If we choose $\bar{a} = 111$ nodes, for example, then we only need to query

$$\bar{s} \approx \frac{6438}{111} = 58 \text{ nodes}$$

on average to obtain a path discovery probability of 99.01 %. The total number of involved nodes is slightly bigger (169 nodes) than before, but decreasing $\bar{s}$ is an advantage if the search algorithm is applied more frequently than the advertising algorithm.

Let us increase the path discovery probability to 99.99 % ($q = 10^{-4}$) now. Hence, only one in 10000 path discovery attempts is expected to fail. The number of nodes required in the optimal case are then

$$\bar{a} = \bar{s} \approx \sqrt{-N \ln q} = 113.4727 \text{ nodes}$$

Hence, only 228 nodes need to be involved, which is slightly more than 16 % of 1398.

[2]We use this average number of nodes in the simulations the we present in Section 6.3.

Chapter 4
The FIXRANDOMQUERY Algorithm

In this chapter, we study an algorithm that is similar to the RANDOMQUERY algorithm introduced in the previous chapter. Instead of advertising and searching each node with some probability, the FIXRANDOMQUERY algorithm places advertisements and searches at a fixed number of nodes.

Although the analysis is different and more complicated for this scenario, the results are the same.

4.1 Algorithm

In a network of N connected[1] nodes, the advertising and search algorithms are defined as follows:

Advertising algorithm (proactive): At network setup, each node advertises itself by randomly placing advertisements at exactly a distinct nodes.

Search algorithm (reactive): A node **S** looking for node **D** randomly queries exactly s nodes in the network. If at least one advertisement of **D** is found, we define that the path discovery was successful.

In the RANDOMQUERY algorithm, a and s were random variables with a binominal distribution and the analysis was based on their averages. Here, a and s are deterministic values.

For a formal description of the algorithm, refer to Figure 4.1. Just as for the RANDOMQUERY algorithm, each node is assumed to have a function *QueryNode*(Node, ⟨Query⟩) available which sends the given query to the node. Furthermore, the algorithm requires a function *GetRandomDistinctNodes*(n) which returns exactly n distinct, random nodes, uniformly distributed over all existing nodes in the network.

4.2 Path Discovery Probability

The path discovery probability is the probability that at least one of the nodes carrying an advertisement for the node we are looking for is being queried. Conversely, the probability of being unsuccessful is the probability of having no encounter, which can be written as

$$\begin{aligned} q &= \frac{N-a}{N}\cdot\frac{N-a-1}{N-1}\cdots\frac{N-a-s+1}{N-s+1} \\ &= \prod_{i=0}^{s-1}\frac{N-a-i}{N-i} \\ &= \frac{(N-a)!(N-s)!}{(N-a-s)!N!} \end{aligned} \tag{4.1}$$

[1]Two nodes are connected if there exists a multi-hop path between them.

```
uses QueryNode(node, ⟨Advertise(node)⟩)
uses bool = QueryNode(node, ⟨Search(node)⟩)
uses list = GetRandomDistinctNodes(n)

function Advertise(a)
    nodelist = GetRandomDistinctNodes(a)
    for each node in nodelist
        QueryNode(node, ⟨Advertise(Me)⟩)
    end for
end function

function success = Search(s, findnode)
    nodelist = GetRandomDistinctNodes(s)
    for each node in nodelist
        if QueryNode(node, ⟨Search(findnode)⟩)
            return true
        end if
    end for
    return false
end function
```

Figure 4.1: *The advertisement and search algorithms in the* FixRandomQuery *algorithm.*

The path discovery probability is therefore

$$p = 1 - q = 1 - \frac{(N-a)!(N-s)!}{(N-a-s)!N!}$$

Figure 4.2 (a) shows how this probability increases with higher values for *a* and *s*.

4.3 Optimal Performance

Just as for the RandomQuery algorithm, the optimal performance is obtained when $a = s$. Equation 4.1 then becomes

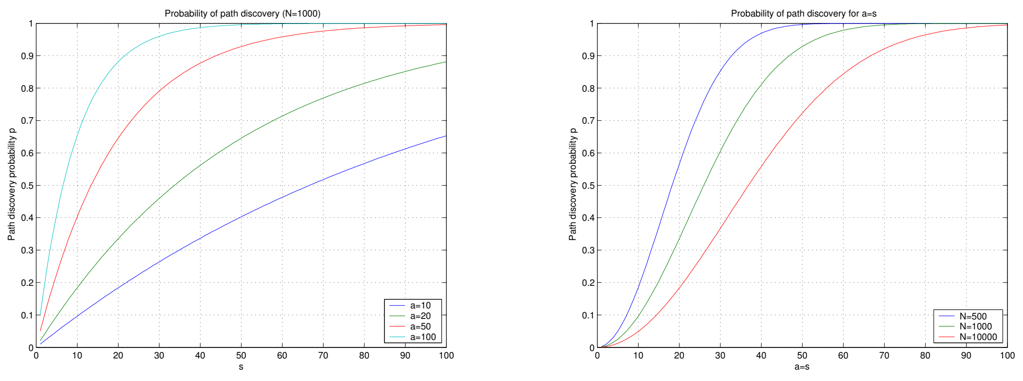
$$q = \prod_{i=0}^{a-1} \frac{N-a-i}{N-i} = \frac{((N-a)!)^2}{(N-2a)!N!}$$

This minimizes the value for q under the constraint that the sum $a+s$ remains constant. Figure 4.2 (b) displays the curves of the optimal performance for some chosen network sizes.

4.4 Upper and Lower Bounds

Sometimes, we are interested in how many nodes, $N_{\max}$, a system can support when a, s and the probability p (or q) are given. Since equation (4.1) is a product of a variable number of terms, transforming it is difficult. To simplify the calculations, we can use the following lower and upper bounds:

$$\prod_{i=0}^{s-1} \frac{N-a-s+1}{N-s+1} \leq \prod_{i=0}^{a-1} \frac{N-a-i}{N-i} \leq \prod_{i=0}^{s-1} \frac{N-a}{N}$$



(a) (b)

Figure 4.2: *(a) Path discovery probability as a function of a and s for $N = 1000$ nodes. (b) Optimal path discovery probability ($a = s$) for different network sizes.*

$$\left(\frac{N-a-s+1}{N-s+1}\right)^s \leq q \leq \left(\frac{N-a}{N}\right)^s$$

Equality holds if and only if $s = 0$ or $s = 1$. The tightness of the bounds only depends on the $\frac{N}{s}$ ratio and is independent of a. If $N >> s$, the bounds are tight. If $a < s$, one might want to switch the values for a and s to achieve tighter bounds.

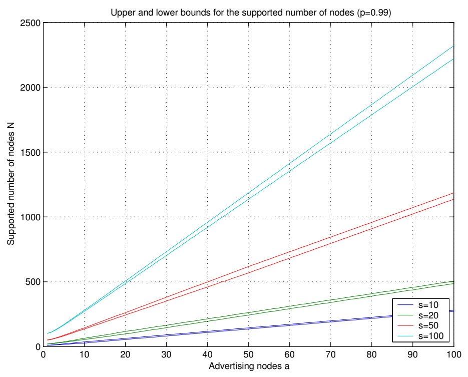
The upper bound can be transformed as follows:

$$\begin{aligned}
q &= \left(\frac{N_1-a}{N_1}\right)^s \\
\ln q &= \ln\left(\frac{N_1-a}{N_1}\right)^s \\
\ln\left(\frac{N_1-a}{N_1}\right) &= \frac{\ln q}{s} \\
\frac{N_1-a}{N_1} &= \exp\left(\frac{\ln q}{s}\right) \\
N_1 &= \frac{a}{1-\exp\left(\frac{\ln q}{s}\right)}
\end{aligned}$$

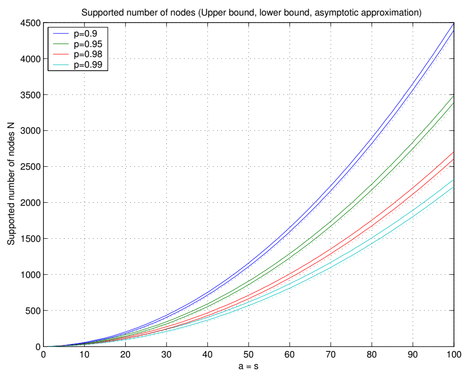
Similarly, the lower bound is transformed into:

$$\begin{aligned}
q &= \left(\frac{N_2-a-s+1}{N_2-s+1}\right)^s \\
\ln q &= \ln\left(\frac{N_2-a-s+1}{N_2-s+1}\right)^s \\
\ln\left(\frac{N_2-a-s+1}{N_1-s+1}\right) &= \frac{\ln q}{s} \\
\frac{N_2-a-s+1}{N_2-s+1} &= \exp\left(\frac{\ln q}{s}\right) \\
N_2 &= \frac{a+s-1+(1-s)\exp\left(\frac{\ln q}{s}\right)}{1-\exp\left(\frac{\ln q}{s}\right)}
\end{aligned}$$

The maximum number of supported nodes, $N_{\max}$, lies somewhere between the two bounds N_1 and N_2. Figure 4.3 shows these bounds for various values of a, s and p.



(a)



(b)

Figure 4.3: *(a) Upper and lower bounds for the number of nodes N as a function of a and s ($p = 0.99$). (b) Upper and lower bounds for the number of nodes in the optimal case $a = s$.*

4.5 Asymptotic Performance

With these bounds, we can analyze the asymptotic performance of the algorithm. Let us first have a look at the optimal case when $a = s$ and generalize this result afterwards for any a and s.

Theorem 2. For a given probability q, $N_{\max}$ asymptotically grows with a^2 if $a = s$. Formally,

$$N_{\max} = -\frac{a^2}{ln(q)} \qquad \text{if } a = s \text{ and } a \to \infty$$

Proof. To prove this theorem, we calculate the asymptotic behavior of both bounds. For the upper bound, we obtain

$$\begin{aligned}
\lim_{a\to\infty} \frac{N_1}{a^2} &= \lim_{a\to\infty} \frac{a}{a^2\left(1-\exp\left(\frac{\ln q}{a}\right)\right)} \\
&= \lim_{a\to\infty} \frac{\frac{1}{a}}{1-\exp\left(\frac{\ln q}{a}\right)} \\
&= \lim_{a\to\infty} \frac{\frac{-1}{a^2}}{\frac{\exp\left(\frac{\ln q}{a}\right)\ln q}{a^2}} && \text{(Bernoulli - de l'Hospital)} \\
&= \lim_{a\to\infty} \frac{-1}{\exp\left(\frac{\ln q}{a}\right)\ln q} \\
&= \frac{-1}{\ln q} && \text{since } \frac{\ln q}{a} \to 0
\end{aligned}$$

Similarly, the limit for the lower bound is

$$\begin{aligned}\lim_{a\to\infty}\frac{N_2}{a^2} &= \lim_{a\to\infty}\frac{2a-1+(1-a)\exp\left(\frac{\ln q}{a}\right)}{1-\exp\left(\frac{\ln q}{a}\right)}\\ &= \frac{-1}{\ln q}\end{aligned}$$

Both limits can easily be calculated using Mathematica [26] or similar software packages.

From $N_1 \leq N_{\max} \leq N_2$, it follows that

$$\lim_{a\to\infty}\frac{N_{\max}}{a^2} = \frac{-1}{\ln q}$$

which concludes the proof. □

This result can be generalized for the case when a and s are not necessarily equal, but still tend towards infinity.

Theorem 3. For a given probability q, $N_{\max}$ asymptotically grows with as if both a and s go to infinity. Formally,

$$N_{\max} = -\frac{as}{ln(q)} \qquad \text{if } a\to\infty \text{ and } s\to\infty$$

Proof. As before, we calculate the asymptotic behavior of both bounds. For the upper bound,

$$\begin{aligned}\lim_{s\to\infty}\lim_{a\to\infty}\frac{N_1}{as} &= \lim_{s\to\infty}\lim_{a\to\infty}\frac{a}{as\left(1-\exp\left(\frac{\ln q}{s}\right)\right)}\\ &= \frac{-1}{\ln q}\end{aligned}$$

Similarly, the limit for the lower bound is

$$\begin{aligned}\lim_{s\to\infty}\lim_{a\to\infty}\frac{N_2}{as} &= \lim_{s\to\infty}\lim_{a\to\infty}\frac{a+s-1+(1-s)\exp\left(\frac{\ln q}{s}\right)}{1-\exp\left(\frac{\ln q}{s}\right)}\\ &= \frac{-1}{\ln q}\end{aligned}$$

Both limits were calculated using Mathematica [26].

From $N_1 \leq N_{\max} \leq N_2$, it follows that

$$\lim_{a\to\infty}\frac{N_{\max}}{as} = \frac{-1}{\ln q}$$

which concludes the proof. □

Hence, for large scale networks, the as product grows linearly with the number of nodes in the network.

4.6 Comparison with the RandomQuery Algorithm

Although expressed differently, both the RandomQuery and the FixRandomQuery algorithm come up with the same asymptotic result. The result obtained for the RandomQuery is slightly more general. It only requires the *as* product to go to infinity whereas the proof for the FixRandomQuery algorithm has been carried out with both a and s individually tending towards infinity. Moreover, the equations for the RandomQuery algorithm are simpler and easier to handle.

For finite N, the FixRandomQuery algorithm yields a slightly better path discovery probabilities than the RandomQuery with the same average number of nodes being advertised and searched. For $a = s = 81$ and $N = 1398$ nodes, for example, the path failure probability according to equation (4.1) is

$$q_{\text{FixRandomQuery}} = \prod_{i=0}^{s-1} \frac{N-a-i}{N-i} = 0.69\ \%$$

as opposed to 0.91 % obtained with equation (3.1) in Section 3.5. This difference arises from the probabilistic distribution of the number of nodes in the RandomQuery algorithm and the convexity of the path discovery probability function. Mathematically, the two probabilities are related by the following formula

$$q_{\text{RandomQuery}}(N, p_A, p_S) = \sum_{a=0}^{\infty} \sum_{s=0}^{\infty} P(A=a)P(S=s)q_{\text{FixRandomQuery}}(N, a, s)$$

where A and S are binominal random variables with probability p_A and p_S respectively, i. e.

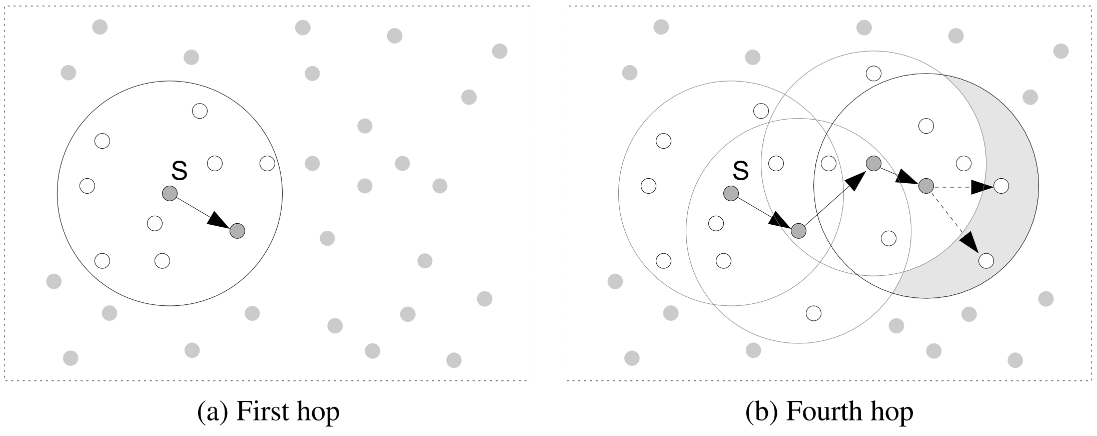
$$P(A=a) = \binom{N}{a} p_A^a (1-p_A)^{N-a} \qquad \text{and} \qquad P(S=s) = \binom{N}{s} p_S^s (1-p_S)^{N-s}$$

Both the RandomQuery and the FixRandomQuery algorithm cannot be implemented in a straightforward and cheap manner. To our knowledge, distributing advertisements homogeneously requires either flooding or the knowledge of the network (nodes and links), both of which are not viable for large scale ad hoc networks. These algorithms mainly serve the purpose of mathematical analysis.

Chapter 5
Snakes

In Chapter 3 and Chapter 4, we introduced the RANDOMQUERY algorithm and the FIXRANDOMQUERY algorithm to analyze the performance of the RANDOMENCOUNTER concept. We have shown that path discovery is possible with $O(\sqrt{N})$ nodes. Both these algorithm achieve optimal performance by distributing the advertisements and the searched nodes uniformly in the network. However, they are theoretic and cannot be implemented in a straightforward and cheap manner.

Contacting random nodes in a network with a uniform distribution is not easy. In Chapter 6 and Chapter 7, we present algorithms that try to approximate a uniform distribution when placing advertisements or searching for them. These algorithms use *snakes*, a novel network primitive that we introduce in the following paragraphs.



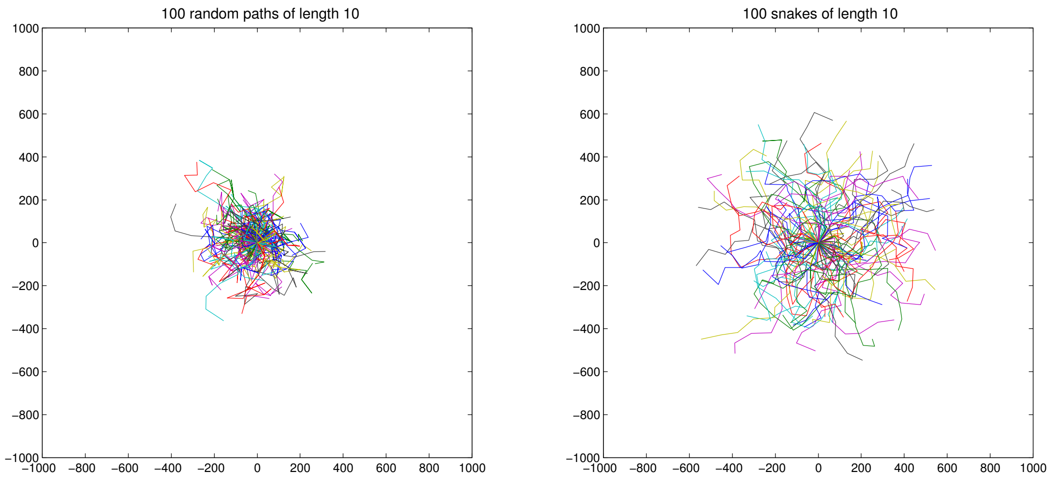
(a) First hop (b) Fourth hop

Figure 5.1: *Snake propagation.* ***S*** *is the snake head. The circles delimit the communication range. The node after the third hop can forward the snake packet to one of the nodes in the grey area, since these nodes are not yet in the neighborhood list of the snake packet.*

5.1 Description

Snakes are built with unicast packets that travel in a random fashion through the network. Unlike normal unicast packets, however, snake packets do not contain a destination address.

To start a snake, a node - called *snake head* - prepares a snake packet with a list of all its neighbors (node IDs) and transmits this packet randomly to one of these neighbors. A node receiving a snake packet randomly selects a neighbor whose ID is not in the snake packet. It then adds its own neighbors to the snake packet and forwards the packet to the selected neighbor. This procedure is repeated at each node until the snake reaches a certain length. Figure 5.1 illustrates this algorithm schematically. Figure 5.3 gives a sample pseudo-code implementation of snakes.



(a) Random paths (b) Snakes

Figure 5.2: *Qualitative comparison between (a) random paths (without neighborhood list) and (b) snakes (with neighborhood list).*

By using this neighborhood list, snake packets are preferably sent to areas where the snake has not been seen yet. This increases the probability that the snake reaches areas far away from the snake head. If snake packets were forwarded randomly to any neighbor (without considering the neighborhood list), the snake would resemble Brownian motion. For a visual comparison of random paths and snakes, refer to Figure 5.2.

5.2 Snake Packets

The basic snake packet contains the following information:

▷ **TTL.** A time-to-live field which defines the snake length l, measured in number of hops.

▷ **Neighborhood list.** A list of identifiers of all nodes that the snake packet passed and its neighbors.

▷ **Data.** Additional data depending on the algorithm that uses the snake. This field may be used by the above layer.

A snake causes exactly l transmissions in the network. These transmissions do not interfere with each other, since they are performed consecutively. I. e., when a node forwards a snake packet, it must have received the packet completely before starting the transmission. Transmissions of two different snakes may collide, however.

5.3 Neighborhood List

The neighborhood list collects the IDs of all nodes along the snake and their neighbors. This list grows with the length of the snake and can potentially become very long.

```
uses MACSendPacket(toNode, ⟨ttl, neighborhoodList, data⟩ )
uses list = GetCurrentNeighbors()
uses element = RandomElement(list)
expects ⟨continue, newData⟩ = SnakeReceive(fromNode, ⟨ttl, neighborhoodList, data⟩ )

function SnakeSend(length, data)
    neighborhoodList = GetCurrentNeighbors()
    toNode = RandomElement(neighborhoodList)
    MACSendPacket(tonode, ⟨length, neighborhoodList, data⟩ )
end function

upon MACReceivePacket(fromNode, ⟨ttl, neighborhoodList, data⟩ )
    ⟨continue, newData⟩ = SnakeReceive(fromNode, ⟨ttl, neighborhoodList, data⟩ );
    return if (ttl == 0) or (continue == false)

    currentNeighbors = GetCurrentNeighbors()
    unknownNeighbors = $\emptyset$
    for each Node in currentNeighbors
        if Node $\notin$ neighborhoodList
            unknownNeighbors = unknownNeighbors $\cup$ {Node}
            neighborhoodList = neighborhoodList $\cup$ {Node}
        end if
    end for

    if unknownNeighbors = $\emptyset$
        toNode = RandomElement(currentNeighbors)
    else
        toNode = RandomElement(unknownNeighbors)
    end if

    MACSendPacket(toNode, ⟨length, neighborhoodList, newData⟩ )
end upon
```

Figure 5.3: *Creating and forwarding snakes. The SnakeReceive function must be provided by the algorithm using the snakes.*

In our simulations, we used a list of infinite size, i. e., we let the list grow to the full required size. Despite being feasible, this strategy is not practical for real-world implementations. Large snake packets generate unnecessary overhead, with regards to bandwidth and transmission time.

It is more appropriate to limit the number of carried node IDs and to carry an incomplete node neighborhood list in the snake packet. Figure 5.1 reveals that the most recently added neighbors are the most important ones. Hence, the neighborhood list can be restricted to the neighbors of the last 3 to 5 snake nodes. Because of transmission range overlapping, the list of neighbors of 3 consecutive nodes along the snake contains only about twice the average number of neighbors. A neighborhood list of about 30 entries should therefore be sufficient for most applications. This is much more efficient and does not significantly affect the performance.

5.4 Analysis

The propagation of snakes is the crucial factor for the advertisement distribution of the algorithms presented in Chapter 6. In this section, we therefore attempt to quantify the important measures of snake propagation. We first derive the equations that describe the distribution of the hop distance and show that, despite some simplifying assumptions, they are difficult to solve. In the second part, we present a Monte Carlo simulation to estimate the important values statistically.

5.4.1 Formal Analysis

Assume an infinite network with uniformly distributed nodes of a certain density λ. Nodes have a transmission range r and know all neighbors within this range.

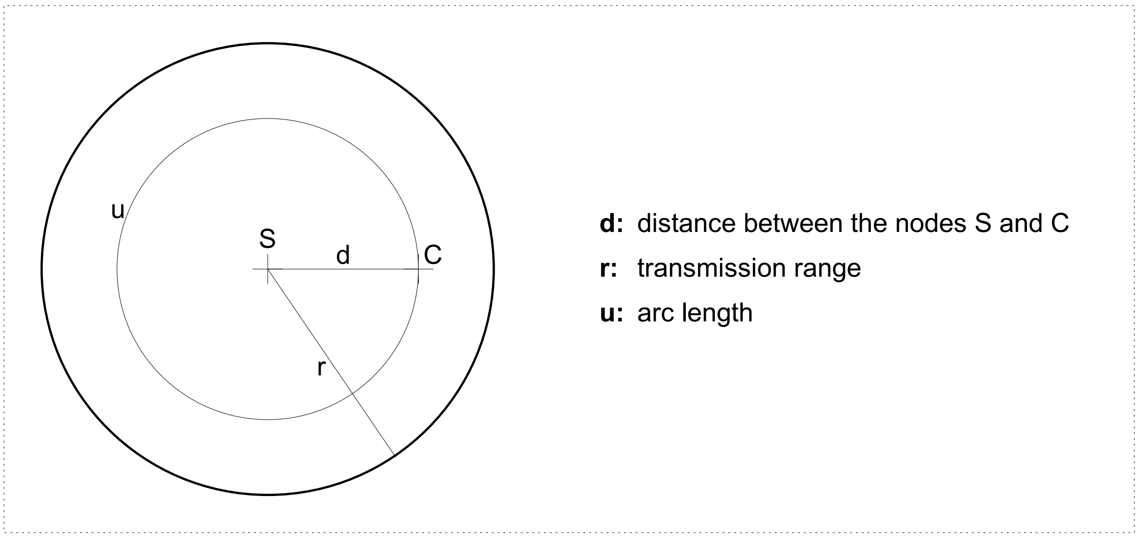


Figure 5.4: *Calculating the distance distribution of the first hop.*

Calculating the hop distance distribution for the first hop, D_1 is simple: the probability increases linearly with the distance. With the notation introduced in Figure 5.4, the distribution can be written as

$$P(D_1 = d) = \frac{2d}{r^2}$$

where $\frac{2}{r^2}$ is used to normalize the pdf. Note that this equation assumes that the node has at least one neighbor. Hence, we assume that the node density is high enough to neglect the probability that a node has no neighbor at all.

Because of the neighborhood list, the hop distance distribution for subsequent steps, D_n, is more difficult to calculate. The distribution for one hop depends on the distribution of the previous hop. In fact, this distribution depends on the whole snake build so far, but for simplicity, we only consider the previous hop.

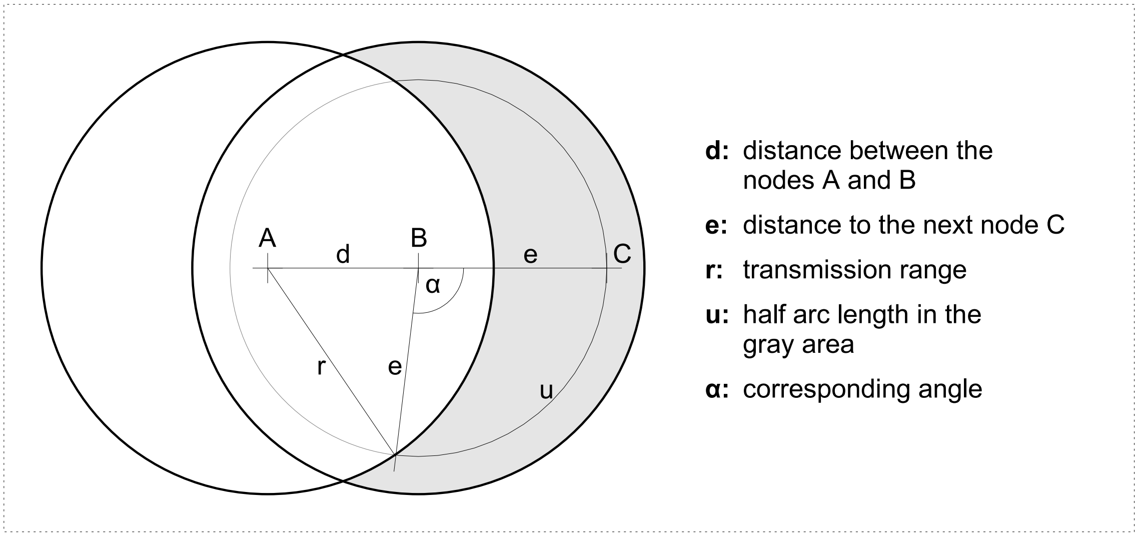

A
B
C
d
e
α
r
e
u
d: distance between the nodes A and B
e: distance to the next node C
r: transmission range
u: half arc length in the gray area
α: corresponding angle

Figure 5.5: *Calculating the distance distribution of all but the first hop.*

Figure 5.5 schematically shows the setup used for the calculation. Assume the distance of the previous hop was d. The next hop node must then lie somewhere in the grey shaded area. (Again, we assume that the node density is high enough for a node to exist in this area.) The probability linearly increases with the arc length $u = e\alpha$, because a node at a distance of e must lie on the arc of length $2u$. To calculate u as a function of d, e and r, we first have to derive an expression for α. The triangle with side lengths d, e and r gives us the relationship

$$r^2 = d^2 + e^2 - 2de\cos(\pi - \alpha) = d^2 + e^2 + 2de\cos(\alpha)$$

which can be transformed into

$$\alpha = \arccos\left(\frac{r^2 - d^2 - e^2}{2de}\right)$$

If $d + e = r$, we obtain

$$\alpha = \arccos\left(\frac{(d+e)^2 - d^2 - e^2}{2de}\right) = \arccos(1) = 0$$

For $e < r - d$, the probability is 0. When the next hop distance, e, is maximal, i. e., $e = r$, the angle

$$\alpha = \arccos\left(\frac{-d^2}{2de}\right) = \arccos\left(\frac{-d}{2r}\right)$$

is obtained. For $e > r$, the probability is again 0, since r is the maximum transmission range. The probability therefore linearly depends on the function

$$u(r,d,e) = \begin{cases} e \cdot \arccos\left(\frac{r^2-d^2-e^2}{2de}\right) & \text{for } r - d \leq e \leq r \\ 0 & \text{otherwise} \end{cases}$$

Note that we have chosen a constant d so far. The input to this function, however, is the distribution D_{n-1}. Hence, the hop distance distribution, D_n, can be expressed as

$$P(D_n = e) = \frac{1}{c}\int_0^r P(D_{n-1} = d)u(r,d,e)\,\mathrm{d}d \tag{5.1}$$

where c is the constant

$$c = \int_0^r \int_{r-d}^r P(D_{n-1} = d)u(r,d,e)\,\mathrm{d}e\,\mathrm{d}d$$

to normalize the pdf.

In view of the function $u(r,d,e)$ containing an arccos and being integrated at each hop, these formulas unfortunately serve mathematical beauty more than practical usefulness. Even for simple D_{n-1} distributions, solving the integrals analytically is hardly possible. Note, furthermore, that the simplifying assumptions taken are quite strong already. In particular, the assumption that there exists a node in the grey shaded area may influence the result even for high node densities. Namely, if $d \to 0$, the shaded area tends to zero as well and the probability to find a node in there vanishes.

Setting $D_{n-1} = D_n$ leads to a differential equation which implicitly expresses the steady-state hop distance distribution. With todays equation solving tools, however, it is not possible to obtain an explicit expression for D_n.

The hop distance is not the only parameter of interest, of course. The angular distribution between two hops as well as the total distance to the source node **S** are important, too. Unfortunately, both these distributions depend on the hop distance distribution.

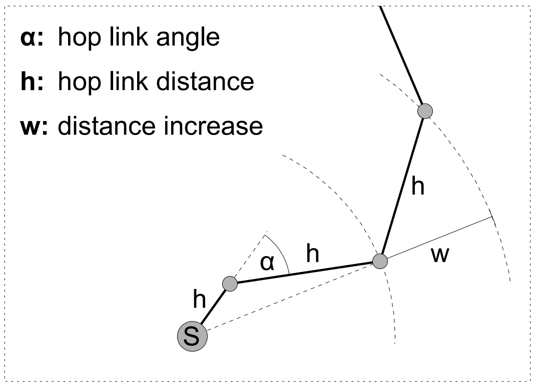
5.4.2 Numerical Analysis

Since the analytical means are exhausted, further analysis has to be carried out numerically. There are basically two possibilities: First, one may try to solve the formal equations numerically. Second, the problem can be tackled completely numerically using a Monte Carlo simulation. Due to the numerous approximations made in Section 5.4.1, we have chosen the second way.

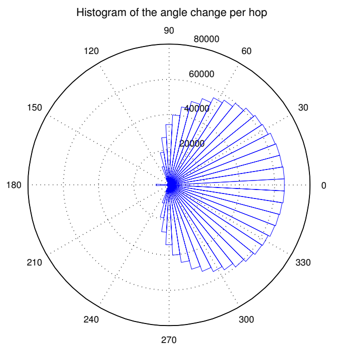
The following experiment was repeated 10^5 times: Nodes were distributed uniformly[1] with a density of $\lambda = 4.45 \cdot 10^{-4}$ nodes/m^2 over a circular area of radius 2.1 km. The transmission range was $r = 100$ m. The snake head **S** was randomly picked within a circle of 100 m radius around the center of the area. (If no nodes were in this area, a new set of nodes was generated.) From this snake head, a snake of length $l = 20$ hops was build and the positions of all nodes along the snake were recorded. Since the radius of the circular area is greater than the maximum distance a snake may ever depart from the snake head, the probability of a snake hitting the border is zero.

Figure 5.6 shows various diagrams of the gathered data. Let us first have a look at the hop distance plotted in diagrams (c) and (d). As derived analytically in Section 5.4.1, the distribution of the first hop grows linearly with the distance. The mean distance is about 66.8 m for the first hop, which is close to the theoretical value of 66.67 m. Because this first hop is short as compared to the steady-state hop distance of about 74 m, the second hop is much longer. This can be explained intuitively by looking at Figure 5.5: If d is small, then e tends to be big, since $r - d \leq e \leq r$. Hop 3 is again a bit shorter and from hop 4 on, no difference is visible as compared to the steady-state hop distance. From hop 9 onwards, the mean distance decreases slightly. This can be explained with the possibility that snakes turn around and generate a geographical loop. In these cases, they may end up at a node which has no neighbors that are not in the neighborhood

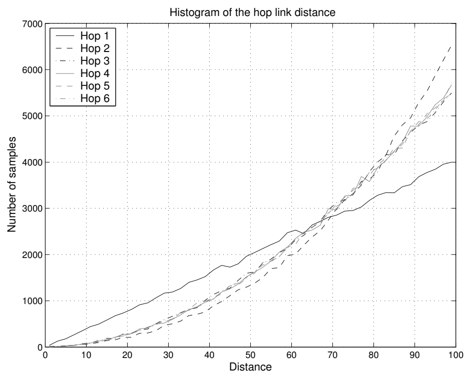
[1]For all simulations, we used the Mersenne Twister random number generator [27] with a period of $2^{19937} - 1$ and an order of equidistribution of 623 dimensions.



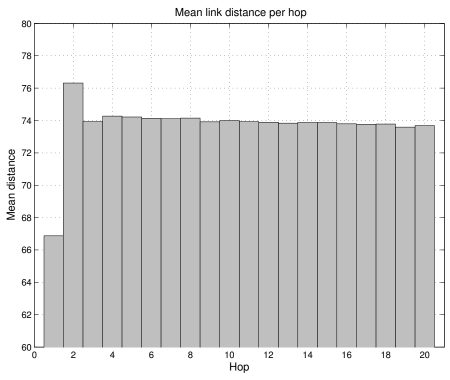
(a)



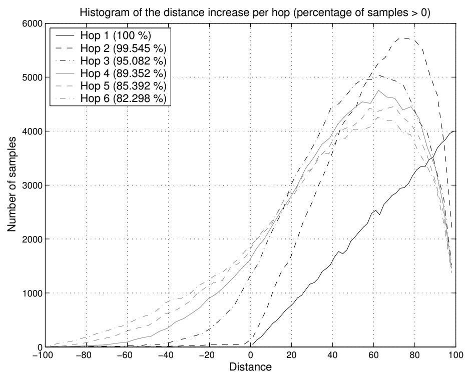
(b)



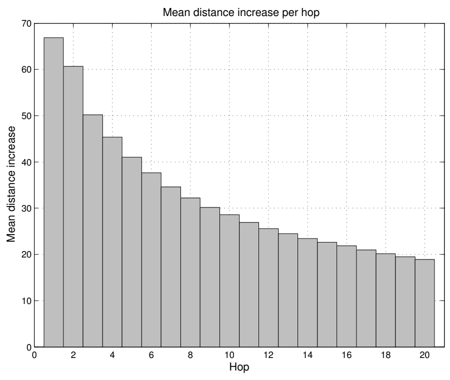
(c)



(d)



(e)



(f)

Figure 5.6: *Propagation of snakes of length $l = 20$ hops in a network with 13.98 neighbors on average. (a) Schematic explanation of the values. (b) Distribution of the change in direction at each hop. (c) Hop link distance distribution and (d) mean hop link distance. (e) Increase in distance from the source node and (f) mean increase in distance from the source node.*

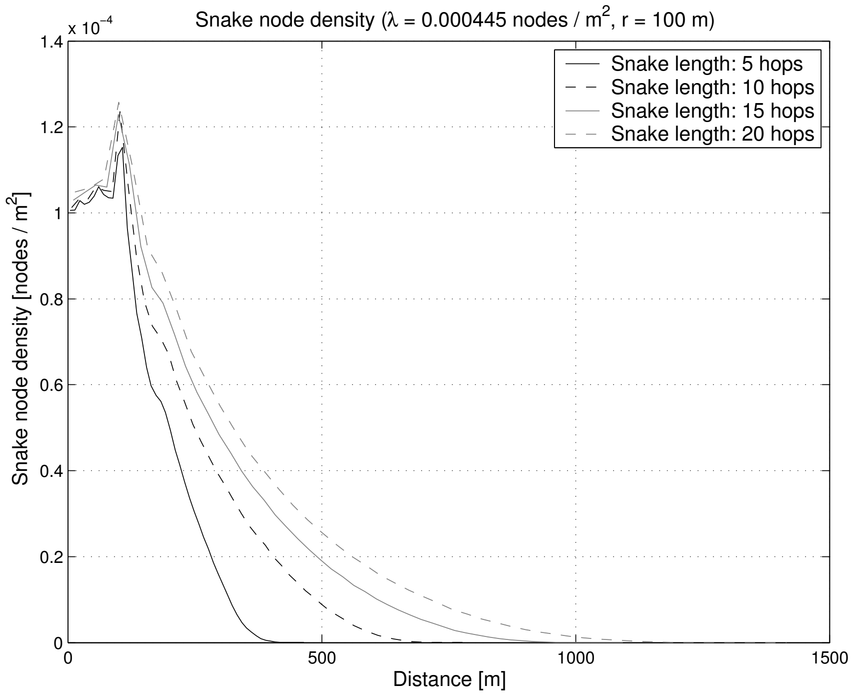
list already. Such nodes forward the snake packet to any random neighbor. Note that the formal analysis in Section 5.4.1 did not take this effect into account.

Figure 5.6 (b) shows a histogram of the change in direction at each hop. Most of the time, snakes turn by less than 60 degrees to the left or to the right. The probability that snakes turn by more than 90 degrees is very small. By looking carefully at the plot, one will discover a small fraction of the samples at 180 degrees. These belong to nodes that did not have any other neighbor than the one from which they received the snake packet. Therefore, they had to send it the same hop back.

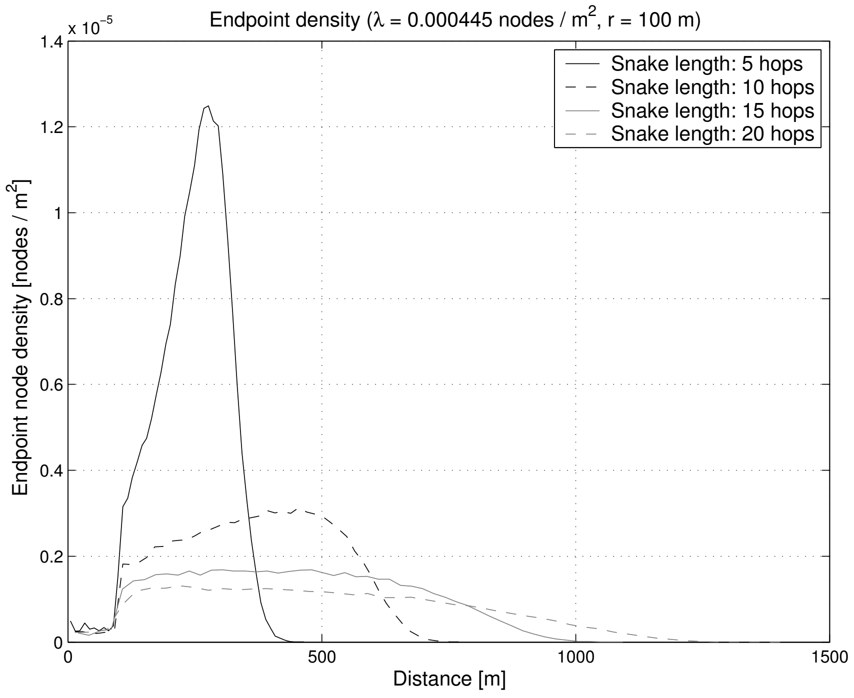
Since snakes can turn, the effective distance that they depart from the snake head is shorter than the cumulative hop distance. We therefore represented the increase in distance from the snake head at each hop in Figure 5.6 (e) and (f). For the first hop, the histogram as well as the mean coincides, for obvious reasons, with the first hop of the hop distance plot. After the second hop, there is a possibility that the distance to the snake head decreases. The probability that this happens increases with each hop and the mean distance increase therefore drops.

Figure 5.7 (a) shows the density of nodes hit by snakes with respect to the distance to the snake head. The density is maximal at the transmission range ($r = 100$ m) and decreases quickly afterwards.

The density of the snake endpoints (the last node belonging to the snake) is drawn in Figure 5.7 (b). For short snakes ($l = 5$ and $l = 10$), this density has a clear peak at about half of the maximum achievable distance. For long snakes, however, the distribution is almost uniform over a wide range and then slowly decreases. This shape is advantageous to spread information uniformly around a node and will be used for the BREEDINGSNAKE advertisement algorithm presented in Section 6.2.3.



(a)



(b)

Figure 5.7: *(a) Density of nodes hit by snakes. (b) Density of snake endpoint nodes.*

Chapter 6
Snake Algorithms for Static Networks

In this chapter, we study three algorithms for static networks based on snakes. All algorithms are based on the RANDOMENCOUNTER concept and therefore consist of two parts: an advertisement phase for nodes to advertise their existence and a search algorithm that is executed whenever a path is to be discovered.

6.1 Model

We assume a static network in which nodes neither move nor fail. In particular, we do not consider a time axis. Furthermore, we assume a perfect MAC layer, i. e., nodes can transmit packets of any size at any time to any neighboring node. Effects such as interference, contention and bandwidth constraints are not considered. Although these assumptions are unrealistic, they allow us to analyze the foundations of snake algorithms while minimizing the effects of hidden factors.

6.2 Algorithms

6.2.1 SNAKE-SNAKE Algorithm

The simplest algorithm is the SNAKE-SNAKE algorithm. At network setup, each node advertises itself by sending m_A advertisement snakes of length l_A. The snake packet carries the node ID of the snake head in its data section. Each node along such a snake stores a routing entry (unless one exists already) to its predecessor before forwarding the snake packet. Hence, a node along the snake can find the snake head by following these routing entries.

If a node **S** wants to establish a path to a node **D**, it launches m_S snakes of length l_S. The corresponding snake packets contain the node IDs of **S** and node **D** in their data sections. If such a snake intersects with a snake advertising for **D**, a route has been discovered. The encounter node is denoted by **V**. Figure 6.1 schematically illustrates how the algorithm works.

Once a path has been discovered, it needs to be set up.

- ▷ To set up a *distance vector* path, each node along the search snakes stores a routing entry for **S** to the previous node on the snake. Upon encountering with an advertisement snake, the encounter node **V** sends a packet along the snake towards **D** to add the routing entries for **S** to these nodes. As soon as this packet reaches **D**, a reply packet (similar to an AODV RREP packet [11]) is sent back to **S**. It establishes the route to **D** at all intermediate nodes and reports to **S** that a path has been found.

- ▷ To setup a *source routing* path, the search snake packet records the route in its data section. Upon encountering an advertisement, the packet is sent to **D** along the advertisement snake and then returned to **S** along the recorded route.

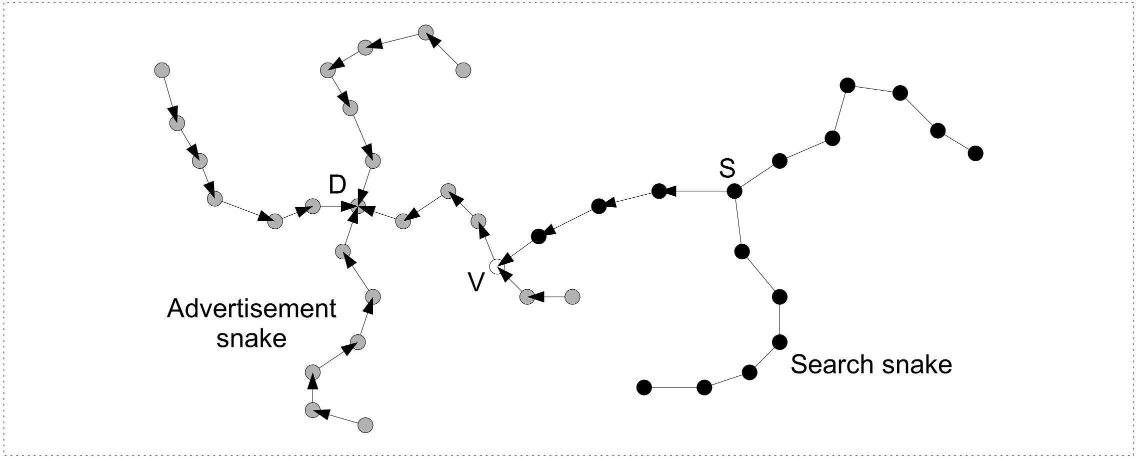


Figure 6.1: *The* SNAKE-SNAKE *algorithm. Advertised nodes are drawn as grey circles and searched nodes as black circles. Each line represents one hop.* ***V*** *is the encounter node.*

Note that several paths to the destination may be found. More precisely, each search snake may report a path. If the routing protocol in use supports multiple paths (e. g. multipath protocols like CHAMP [28] that cache additional paths), all of them may be used. Otherwise, the best path according to some performance metric should be chosen. A performance criteria may be the path length in number of hops.

The number of advertisements per node in the network is

$$a \leq 1 + m_A l_A$$

Equality holds if the snakes do not overlap at any node. Similarly, the number of searched nodes is

$$s \leq 1 + m_S l_S$$

Since the advertised and searched nodes are not uniformly distributed in the network, these numbers cannot be plugged into equation (3.2); due to the correlation among the advertised and searched nodes, the performance is expectedly lower. We discuss this in Section 6.3.

6.2.2 SNAKE-LONGSNAKE Algorithm

The SNAKE-LONGSNAKE algorithm is the same as the SNAKE-SNAKE algorithm except that advertisements are only placed at every w_Ath node along the advertisement snake. To compensate that, much longer snakes are chosen. If $w_A = 1$, this algorithm coincides with the SNAKE-SNAKE algorithm.

Each advertised node must store a routing entry that contains the path to the previous advertisement on the snake. This path is w_A hops long. Unless $w_A = 1$, storing solely the ID of the previous node on the snake is not enough, since this node does not store any information about the snake head. It only relays the snake packet.

The number of advertisements per node in the network can be expressed as

$$a \leq 1 + \left\lfloor \frac{m_A l_A}{w_A} \right\rfloor$$

The disadvantage of this algorithm is that the advertisement snakes are much longer than the number of advertised nodes. Hence, the ratio of sent packets over advertised nodes is much higher than for the SNAKE-SNAKE algorithm.

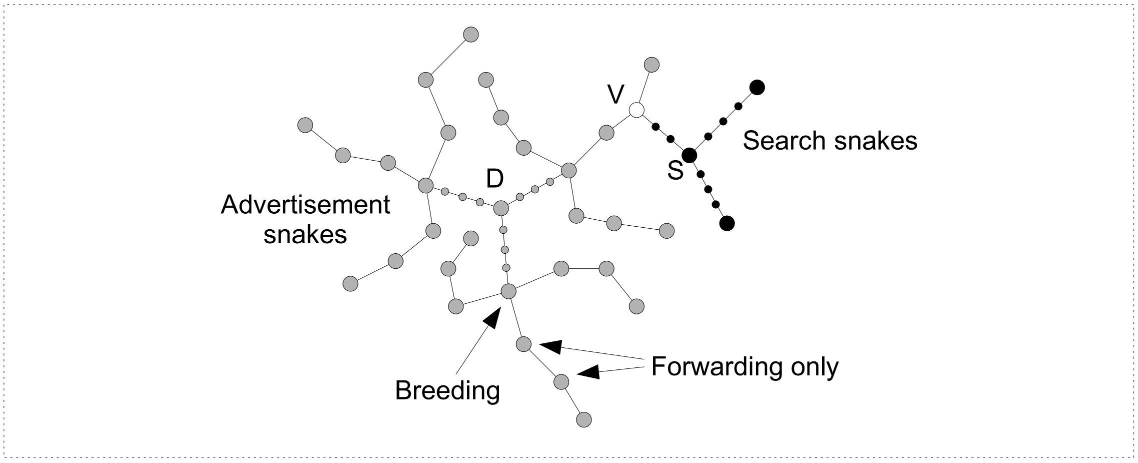


Figure 6.2: *The* SNAKE-BREEDINGSNAKE *algorithm. Advertised nodes are drawn as grey circles and searched nodes as black circles. Each line between two big circles represents one snake. The small circles are nodes along the snake.* ***V*** *is the encounter node.*

6.2.3 SNAKE-BREEDINGSNAKE Algorithm

The SNAKE-BREEDINGSNAKE algorithm overcomes this problem by reusing snakes of other nodes. It is however a bit more complex. The advertisement procedure for a node **D**, depicted in Figure 6.2, works as follows:

▷ **D** sends m_A snakes of length l_A. These snakes are called *first-order* snakes of **D**. Each node along the snakes stores a routing entry for **D** to its predecessor node before forwarding the snake packet.

▷ The endpoint nodes of these first-order snakes include **D** in their own advertisement snake packets. The new snakes are therefore *second-order* snakes of **D**. The advertisement for **D** is only placed at the endpoint of such snakes. It consists of the node ID of the snake head, as this is enough to completely reconstruct the path back to **D**. (This avoids having too many advertisements placed too close to each other.)

  The endpoint nodes of second-order snakes breed again to create third-order snakes, and so on, until the order w_1 is reached. During this phase, the number of snakes advertising for **D** grows exponentially.

▷ After w_1 breeding steps, the endpoint nodes only forward the advertisement in one of their snakes. This is repeated until order w_2 is reached. In this last phase, the number of snakes advertising for **D** remains constant.

To search a node **D**, **S** sends m_S snakes of length l_S. A path is discovered if an advertisement (either along the first-order snake or at the endpoint of a higher-order snake) is found. The discovered path consists of the snake from **S** to the encounter node **V** and the snakes from **V** to **D**. Path setup is similar to that for the SNAKE-SNAKE algorithm.

The number of advertisements placed by the BREEDINGSNAKE procedure is

$$a \leq 1 + m_A l_A + \left(\sum_{n=2}^{w_1} m_A^n \right) + m_A^{w_1} (w_2 - w_1)$$

The summed terms correspond to the advertising node, the first-order snakes, the endpoints of the snakes until order w_1 and the endpoints of the snakes between the orders $w_1 + 1$ and w_2.

Note that if $w_1 = w_2 = 1$, this algorithm coincides with the SNAKE-SNAKE algorithm.

6.3 Simulation Results

To measure the performance of the algorithms proposed in Section 6.2, we simulated them based on the model described in Section 6.1.

The following experiment was repeated 1000 times: First, nodes were distributed uniformly over a circular area with radius 1 km. We have chosen a node density of $\lambda = 4.45 \cdot 10^{-4}$ nodes/m^2 and a communication range of $r = 100$ m. With these values, the expected number of neighbors (theoretical, without border effects) is 13.98 and the expected number of nodes is $\mathrm{E}[N] = 1398$. We run the advertisement algorithm for an arbitrary node **D**. Among the remaining nodes, we then randomly picked 40 nodes and applied the search algorithm on each of them.

By doing so, we generated 40000 samples in 1000 blocks. Samples of the same block are slightly correlated since they all belong to the same network and the same destination node. The gathered data allows us to study the path discovery probability as well as the path length.

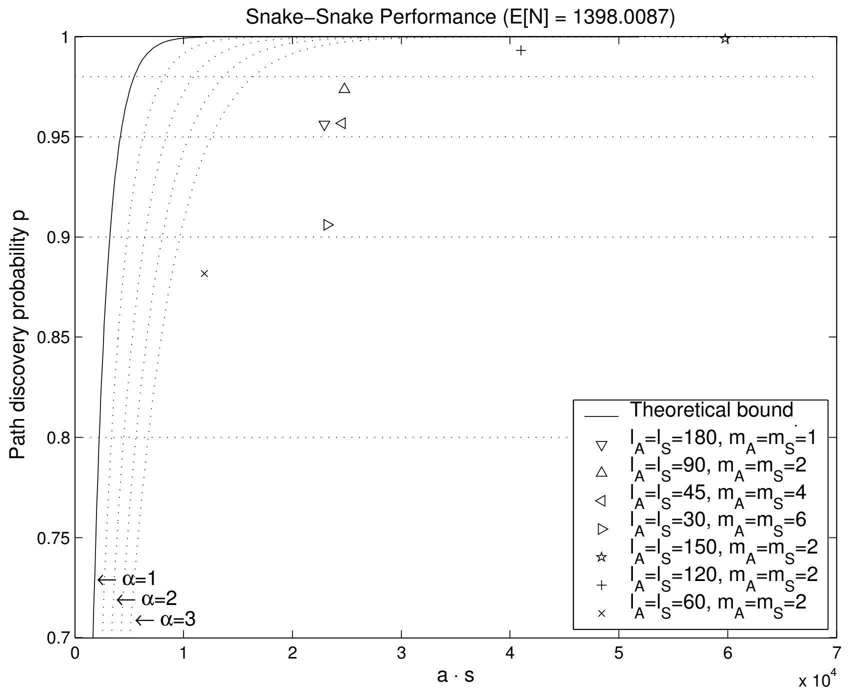


Figure 6.3: *as-p plot for the* SNAKE-SNAKE *algorithm applied in a network with* $\mathrm{E}[N] = 1398$ *nodes uniformly distributed with a density of* $4.45 \cdot 10^{-4}$ *nodes/m*^{2} *on a circular area of 1 km radius.*

6.3.1 Path Discovery Probability

We compare the path discovery probability on *as-p* plots. That is, we count the number of advertised nodes, a_i, and searched nodes, s_i, of each sample and calculate their averages, $\hat{a}$ and $\hat{s}$. The estimated success probability, $\hat{p}$, is calculated as the number of successes divided by the number of samples (40000). The tuple $(\hat{a} \cdot \hat{s}, \hat{p})$ can then be compared to the theoretical bound,

$$p = 1 - \left(1 - \frac{a \cdot s}{\mathrm{E}^2[N]}\right)^{\mathrm{E}[N]} \tag{6.1}$$

To ease the visual comparison, we indicated the curves

$$p = 1 - \left(1 - \frac{a \cdot s}{\alpha \mathrm{E}^2[N]}\right)^{\mathrm{E}[N]} \tag{6.2}$$

for $\alpha \in \{1.5, 2, 2.5, 3\}$ on the plots as well.

The α factor is a good quality measure. It expresses by what factor the *as* product must be multiplied with to obtain the same probability for the optimal case. Since equation (6.1) is optimal, $\alpha \geq 1$ with equality if and only if the algorithm is optimal.

6.3.1.1 Path Discovery Probability of the SNAKE-SNAKE Algorithm

The *as-p* plot for the SNAKE-SNAKE algorithm is shown in Figure 6.3. The first four configurations listed in the legend all advertise and search the same number of nodes. The comparison shows that fewer, but longer snakes give a significantly better performance. Many short snakes tend to place many advertisements close to the advertising node, creating a very non-uniform advertisement distribution. Longer snakes cover distant areas with higher probability.

It is possible to reach path discovery probabilities close to 1 with this simple algorithm. For two snakes of length $l_A = l_S = 150$, a path was found in 99.88 % of the samples. The number of nodes involved in path discovery is at most 602 (301 advertised and 301 searched) with this configuration, which is less than half the total number of nodes.

6.3.1.2 Path Discovery Probability of the SNAKE-LONGSNAKE Algorithm

If we advertise at all nodes along the snakes, the advertisements are highly correlated. To distribute the advertisements more evenly, the LONGSNAKE advertisement algorithm places advertisements at every w_A's node only. This has a big impact on the performance, as shown in Figure 6.4 and Table 6.1. The runs with $w_A = 3$ and $w_A = 4$ have a similar *as* product as the reference simulation with $w_A = 5$, but the probability increases from 91.94 % ($w_A = 3$) to 96.30 % ($w_A = 5$), causing the α-factor to drop from 2.63 ($w_A = 3$) to 2.02 ($w_A = 5$). In our simulated network, the probability increased only insignificantly for higher values of w_A.

Simulation	$\bar{a}$	$\bar{s}$	$a_{\max}$	$s_{\max}$	p [%]	α
Reference	84.86	109.8	91	126	96.30	2.02
$l_A = 100$	57.62	109.7	61	126	90.81	1.90
$l_A = 200$	111.5	109.7	121	126	98.48	2.09
$l_S = 15$	84.90	67.09	91	76	89.62	1.80
$l_S = 35$	85.07	150.7	91	176	98.74	2.10
$w_A = 3$, $l_A = 90$	84.28	109.7	91	126	91.94	2.63
$w_A = 4$, $l_A = 120$	84.51	109.8	91	126	94.98	2.22
$m_A = 4$	111.6	109.7	121	126	98.55	2.07
$m_A = 5$	138.2	109.7	151	126	99.50	2.05

Table 6.1: *Results for the* SNAKE-LONGSNAKE *algorithm (Figure 6.4).*

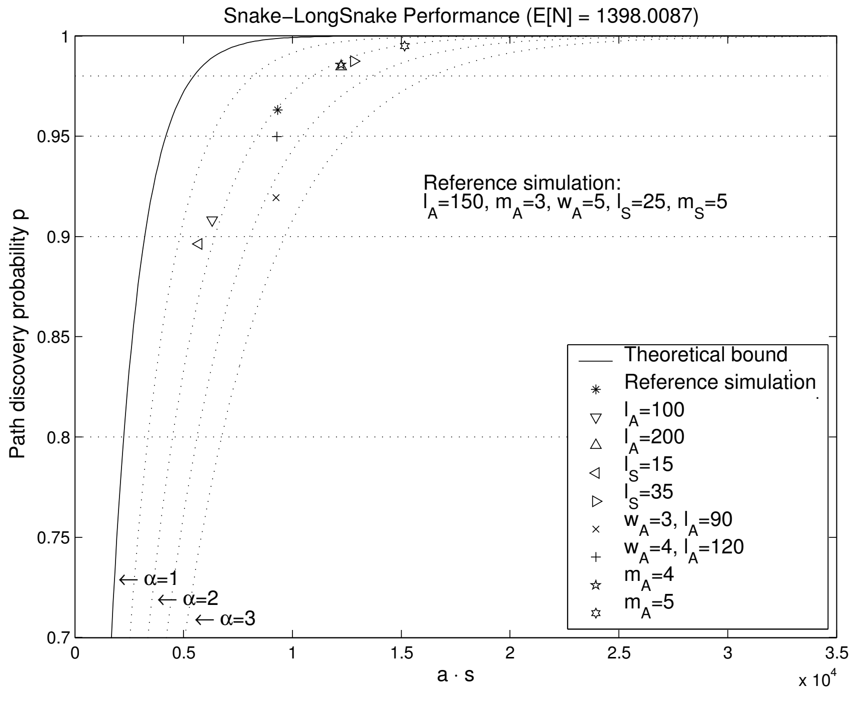


Figure 6.4: *as-p plot for the* SNAKE-LONGSNAKE *algorithm applied in a network with* $\mathrm{E}[N] = 1398$ *nodes uniformly distributed with a density of* $4.45 \cdot 10^{-4}$ *nodes/m*^{2} *on a circular area of 1 km radius. The legend indicates the parameters that differ from the reference simulation.*

The best results are beyond the theoretical curve for $\alpha = 2$ (see equation (6.2)). This means that as compared to the optimal RANDOMQUERY algorithm presented in Chapter 3, the *as* product only needs to be about twice as high to reach the same probability.

6.3.1.3 Path Discovery Probability of the SNAKE-BREEDINGSNAKE Algorithm

Similar results can be obtained with the SNAKE-BREEDINGSNAKE algorithm (Figure 6.5, Table 6.2).

The performance is in general better for longer advertisement snakes. For $l_A = 20$, for example, the path discovery probability is 98.80 %, which corresponds to $\alpha = 1.76$.

A similarly low α value ($\alpha = 1.83$) is obtained for $l_A = 20$ and $l_S = 20$. 99.63 % of the existing paths are discovered with this configuration. To obtain this high path discovery probability, less than one fifth of the nodes are involved in path discovery.

Increasing the length of the search snakes, l_S, increases the path discovery probability. The performance with regard to the α factor, however, decreases: the reference simulation with $l_S = 15$ reaches $\alpha = 2.07$ whereas for $l_S = 25$, $\alpha = 2.24$ is obtained. This behavior is due to the correlation between the searched nodes which is much higher than the correlation between the advertised nodes. Unfortunately, the BREEDINGSNAKE algorithm cannot be applied in the search phase. Forwarding a search snake packet without checking for an advertisement would be unwise, as the expensive operation (in terms of energy and bandwidth) is clearly the packet forwarding. Breeding and searching at all nodes along the higher-order snakes would result in an even higher correlation between the searched nodes.

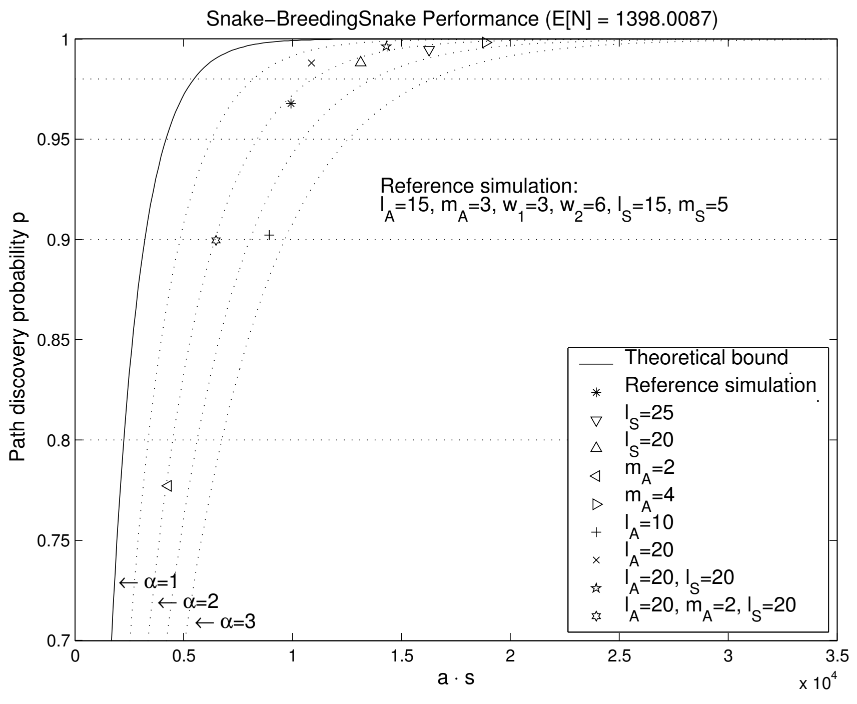


Figure 6.5: *as-p plot for the* SNAKE-BREEDINGSNAKE *algorithm applied in a network with* $\mathrm{E}[N] = 1398$ *nodes uniformly distributed with a density of* $4.45 \cdot 10^{-4}$ *nodes/m*^{2} *on a circular area of 1 km radius. The legend indicates the parameters that differ from the reference simulation.*

6.3.1.4 LONGSNAKE and BREEDINGSNAKE Comparison

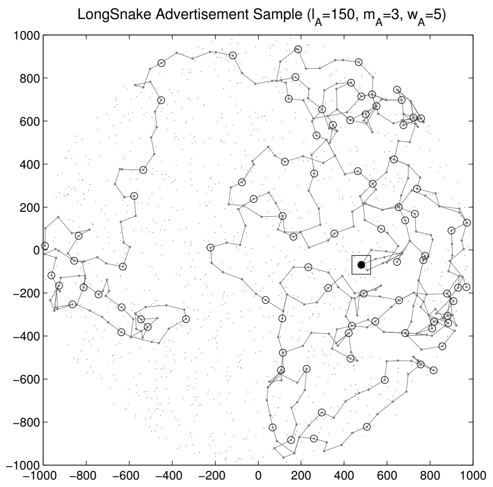
Although the SNAKE-LONGSNAKE and the SNAKE-BREEDINGSNAKE algorithms both reach similar α values, the way they spread the advertisements is quite different. Figure 6.6 qualitatively compares the two advertisement placing algorithms. We have chosen a typical sample for each algorithm.

The snakes of the LONGSNAKE advertisement algorithm are far longer than the network diameter. These snakes explore a large part of the network and place advertisements at regular intervals. As Figure 6.6 (a) shows, quite substantial parts of the network may remain unadvertised, whereas the advertisement density can become quite high in areas where the snakes get stuck for several hops (e. g. at the borders).

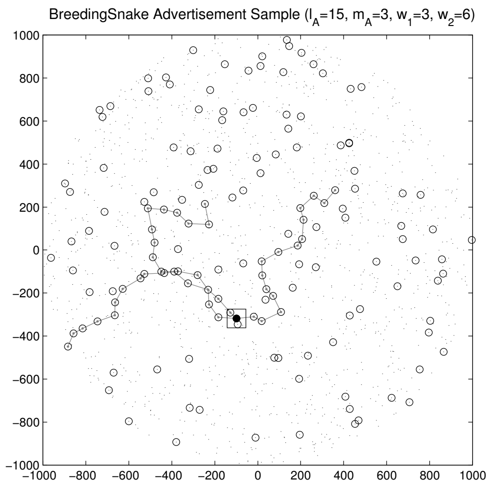
The BREEDINGSNAKE algorithm spreads the advertisements by recursively sending short snakes. Since these short snakes are sent from different snake heads, the advertisements are distributed without leaving big holes. The advertisement density decreases with increasing distance from the advertising node. Around the advertising node, the density is particularly high because of the first-order snakes.

For the LONGSNAKE algorithm, the primary parameter to adjust to the network size is the snake length l_A. It should be significantly larger than the network diameter. Since the number of sent packets increases linearly with the length of a snake, this algorithm may become expensive.

The primary parameters to tune the BREEDINGSNAKE algorithm are w_1 and w_2. The snake length l_A can be modified for fine tuning. It mainly influences the distance between the advertisements and has no significant impact on the number of advertised nodes except for small w_1 and w_2.



(a) LONGSNAKE algorithm



(b) BREEDINGSNAKE algorithm

Figure 6.6: *Sample nodes advertised by the* LONGSNAKE *and the* BREEDINGSNAKE *advertisement algorithm. Each dot represents one node. A single node, highlighted with a square, advertises itself. The first-order snakes are drawn as lines. Advertised nodes are marked with a circle.*

BREEDINGSNAKE packets carry several node IDs to advertise (depending on w_1 and w_2). They are therefore bigger than LONGSNAKE packets. Packet size is traded off against the number of sent packets.

In most networks, moderately increasing the packet size is less costly than sending more packets. Hence, the BREEDINGSNAKE algorithm is preferred.

6.3.2 Path Length

We measured the path length in number of hops. Beyond the total path length, we counted the number of hops from the source node **S** to the encounter node **V** as well as the hops from **V** to the destination node **D**.

Recall that path discovery usually finds several encounter nodes. If search snakes stop as soon as they find an encounter, each search snake may report up to one discovered path. If search snakes

Simulation	$\bar{a}$	$\bar{s}$	$a_{\max}$	$s_{\max}$	p [%]	α
Reference	147.9	67.09	163	76	96.79	2.07
$l_S = 25$	148.3	109.7	163	126	99.45	2.24
$l_S = 20$	148.1	88.67	163	101	98.82	2.12
$m_A = 2$	63.94	67.09	67	76	77.71	2.05
$m_A = 4$	280.9	67.13	333	76	99.82	2.14
$l_A = 10$	132.9	67.13	148	76	90.22	2.75
$l_A = 20$	161.9	67.12	178	76	98.80	1.76
$l_A = 20, l_S = 20$	161.6	88.60	178	101	99.63	1.83
$l_A = 20, m_A = 2, l_S = 20$	73.11	88.65	77	101	89.95	2.02

Table 6.2: *Results for the* SNAKE-BREEDINGSNAKE *algorithm (Figure 6.5).*

grow to their full length, each of them may report several paths. In our simulations, we stopped the search snakes at the first encounter and chose the shortest reported path.

6.3.2.1 Path Lengths Using the SNAKE-BREEDINGSNAKE Algorithm

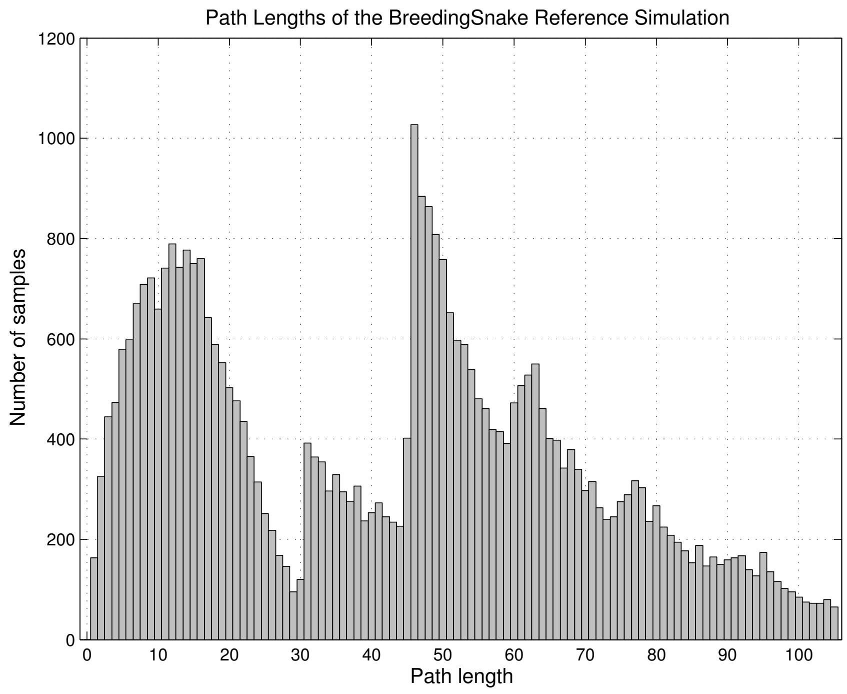


Figure 6.7: *Path length histogram (**S** - **D**) for the* SNAKE-BREEDINGSNAKE *algorithm.*

Figure 6.7 shows a path length histogram for the reference simulation (see Figure 6.5) of the SNAKE-BREEDINGSNAKE algorithm. In view of the network diameter being about 27, the paths are highly suboptimal. The longest paths are 105 hops long, the maximum length with the used parameters. The shortest path is one hop long, i. e., source and destination node were neighbors. Note that we chose different source and destination nodes. Therefore, there is no path of zero length.

The various peaks and holes are not statistical noise - they originate from the structure of the advertising algorithm. To explain them, let us have a look at the path length between **V** and **D** displayed in Figure 6.8 (b). For path lengths up to 15 hops, the encounter node **V** was found on a first-order snake. Note that in about 500 cases, the search snake hit the destination node directly (path length 0). This probability is smaller then hitting a node on the first-order snake because there were 3 snakes but only a single destination node.

The bar for 15 hops belongs to the endpoint of the first-order snake. The number of samples that hit this node is a bit higher. This is because endpoint nodes only have one advertised node next to them. Informally speaking, they have less ”competition”.

The bars for 30 and 45 hops belong to the endpoints of the second-order and third-order snakes. These snakes were sent out in the breeding steps of the BREEDINGSNAKE algorithm. The number of samples therefore grows exponentially. Since nodes along these snakes were not advertised, the values between the peaks are zero.

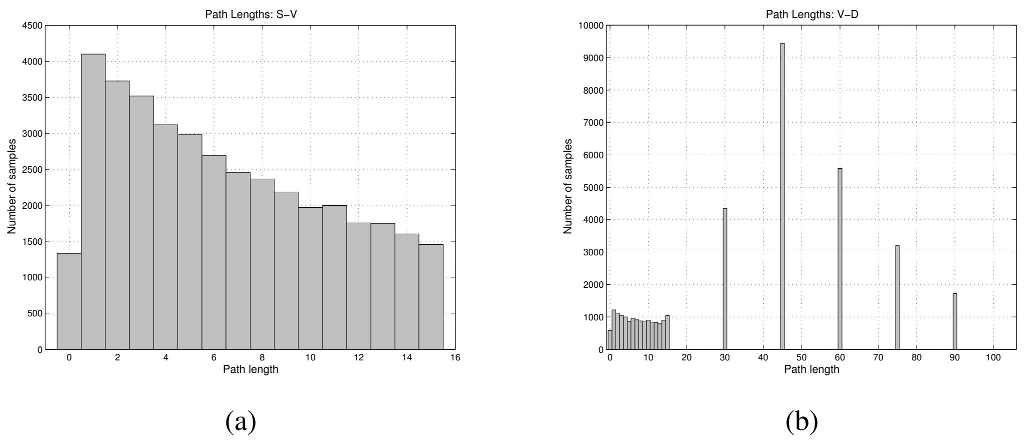


Figure 6.8: *Path length histograms for the* SNAKE-BREEDINGSNAKE *algorithm. (a) Source node **S** - encounter node **V**. (b) Encounter node **V** - destination node **D**.*

The bars for 60, 75 and 90 hops stem from the endpoints of the continuation snakes. Despite the number of snakes (and therefore endpoints) remains constant, the probability of establishing a path through these nodes decreases. This is simply due to our path selection strategy: if paths were discovered, one of the other paths was likely to be shorter. The probability of not finding any shorter paths decreases with increasing path length.

For the same reason, the number of samples are decreasing with increasing number of hops of the search snake. This is shown in Figure 6.8 (a). The first sample on this plot belongs to the case when the source node **S** itself carried an advertisement of **D**. Since this node is the same for all 5 search snakes, the probability for this event is much smaller.

Since the lengths of the sections **S** - **V** and **V** - **D** are almost independent, the path length plot shown in Figure 6.7 is almost a correlation of the **V** - **D** histogram and a flipped version of the **S** - **V** histogram.

6.4 Discussion

6.4.1 Probabilistic Path Discovery

Algorithms based on the RANDOMENCOUNTER concept discover paths probabilistically. Unless both the advertised node density, p_a, and the searched node density, p_s, are both 1, the probability of not discovering an existing path is non-zero.

This is actually not a main concern as long as the path discovery probability is high (e. g. 99 %). Wireless networks are – even in the absence of mobility – inherently probabilistic. Although flooding discovers paths with unit probability[1] if an ideal MAC layer is considered, path discovery may fail with a non-ideal MAC layer.

At this stage, it is not possible to compare the path discovery probability of flooding to that of snake protocols. This very much depends on the chosen MAC layer and its implementation, the environmental conditions (interference, noise), the distribution of the nodes and the RF propagation. Nevertheless, it is safe to say that snake path discovery algorithms can reach probabilities that are high enough to be useful in most applications.

[1]Some optimized broadcast techniques [13], e. g. gossip protocols [29], are probabilistic as well.

6.4.2 Path Optimality

The path discovery algorithms presented in this paper find non-optimal paths. No guarantee about optimality can be given.

For most routing algorithms, this may appear as a non-negligible disadvantage that reduces the total network throughput and increases the average end-to-end delay. For short connections with few data packets, this overhead may be small. For connections with a lot of data, however, the overhead due to non-optimal paths exceeds the savings due to the cheaper path discovery algorithm. One may therefore question whether snake algorithms (or other algorithms based on the RANDOMENCOUNTER concept) are suitable for ad hoc networks.

We strongly believe that they are. Recently, the routing algorithm AntHocNet [25] has been proposed. This algorithm continuously samples existing paths and improves them while data is being transmitted (stochastic routing). For path discovery, AntHocNet uses a strategy similar to flooding, but takes potentially available information about the destination node into account.

Snake algorithms are an ideal candidate for use with AntHocNet. For short connections with few data packets, the utilized path may be highly suboptimal. For longer lasting connections, however, the initial suboptimal path is improved throughout the transmission and – in static networks – converges towards the optimum. Assuming that the convergence speed is high enough, all but the first few data packets will be sent along a fairly optimal path.

Chapter 7
Snake Algorithms for Mobile Networks

In all algorithms presented in Chapter 6, we have not considered time. The network was static and nodes as well as links were assumed to be perfect.

In this chapter, we generalize the SNAKE-BREEDINGSNAKE algorithm (see Section 6.2.3) for mobile networks with node and link failures. We have chosen this algorithm because it exhibits the best results (see Section 6.3). The SNAKE-SNAKE and the SNAKE-LONGSNAKE algorithms can be generalized in a similar fashion.

Note that we have not implemented the algorithms introduced in this chapter. We only present them as ideas and leave the evaluation of their performance to future work.

7.1 MOBILESNAKE-MOBILEBREEDINGSNAKE Algorithm

In a model with mobility and node failure, there are two new problems that need to be addressed:

▷ Advertisements age and become stale.

▷ Snakes may break.

The first problem is due to the mobility of a snake head whereas the second problem appears when nodes along a snake move away or fail. Hence, advertisements cannot be spread once at network setup only. Advertising is a continuous process that takes place as long as a node is connected to the network. Moreover, when searching for a node, snakes can only be regarded as hints. Encountering an advertisement no longer assures that a path has been discovered.

To cope with these issues, we propose the following advertising and search algorithms:

Advertising algorithm. To keep the advertisement structure up-to-date, each node **D** periodically sends two types of snakes:

- ▷ Each t_l seconds (long interval), the node sends one breeding snake of length l_A. As usual, all nodes along this first-order snake keep an advertisement for **D**. Such an advertisement contains the address of **D**, the time at which the snake was sent (t_b) and the distance to **D** in number of hops. The endpoint node of the snake marks the advertisement for breeding and includes **D** in its own snakes with probability $b_2(t-t_b)$. First-order snakes for this endpoint node are second-order snakes for **D**. Therefore, the advertisement for **D** will only be placed at the endpoint of these snakes. Here again, the advertisement is marked for breeding and will cause third-order snakes for **D** being sent with probability $b_3(t-t_b)$, and so on.
- ▷ Each t_s seconds (short interval), the node sends one simple non-breeding snake to place advertisements in its close neighborhood.

Note that the parameters w_1 and w_2 of the original BREEDINGSNAKE algorithm (see Section 6.2.3) have disappeared. Instead, we use more general breeding probability functions $b_w(t - t_b)$ for a wth-order snake. These functions define the probability of including an advertisement in a snake with respect to the age of the advertisement. $b_w(t - t_b)$ should be decreasing with time and eventually reach 0. Moreover, $b_w(t - t_b) \equiv 0$ for $w > w_{\max}$ to limit the number of breeding steps.

If a node receives an advertisement for **D** although it already has one, it keeps the new advertisement if and only if it is more recent (t_b bigger).

Advertisements are deleted if they get older than t_d seconds (i. e., $t \geq t_b + t_d$). Hence, if a node does not advertise itself for t_d seconds, it is assumed to have left the network. To make sure that a node is advertised at any time, $t_s < t_d$.

We do not attempt to specify and evaluate concrete $b_w(t - t_b)$ functions in this thesis. We believe that good functions as well as optimal t_l, t_s and t_d intervals significantly depend on the chosen mobility and node failure model. Determining suitable functions would therefore require simulations in a chosen scenario.

Search algorithm. A node **S** looking for a node **D** sends m_S snakes of length l_S. Upon encounter, the snake packet is sent along the advertising snakes in the hope to reach the destination node. If the snake is broken, the last node on the snake fragment sends again m_S snakes. Such snakes may be shorter, however, since other advertisements are expected to be in the proximity. To avoid loops, these snakes only consider advertisements for **D** (or the snake head of the currently followed snake) that are either more recent (t_b bigger) or closer to the destination with the same age (t_b equal). If no such advertisements are found, the path discovery stops.

Through the infrequently sent breeding snakes, each node makes itself known in the network. Because of mobility, however, these advertisements are not very accurate. The short snakes come in to solve this problem: they frequently update their neighborhood with their current location. Hence, search snakes will in most cases first encounter a node advertised by breeding snakes.

Loose time synchronization among all nodes in the network is necessary. A small shift between the node clocks can be tolerated because this does not affect loop-freedomness. To delete an advertisement and to determine the breeding probability, however, a time duration using two times from different clocks is calculated and the clock shift therefore adds to the duration.

The effective number of advertisements in the network is not only function of the snake length and the breeding probability functions, but also of the time intervals t_l, t_s and t_d. Furthermore, the path discovery probability depends on more factors than in the static case. In particular, we believe that the frequency at which snakes break and the probability of finding another advertisement nearby influence the success rate significantly. But for more precise statements on that, simulations as well as analytical work would be necessary.

Without running simulation, we believe that snake-based path discovery algorithms work best in networks with low or moderate mobility where snakes persist for some time. If snakes break too quickly after setup, their advantage diminishes, causing the path discovery probability to drop. Moreover, snake algorithms may show good results in networks where many nodes have low mobility and only a few nodes move at higher speeds.

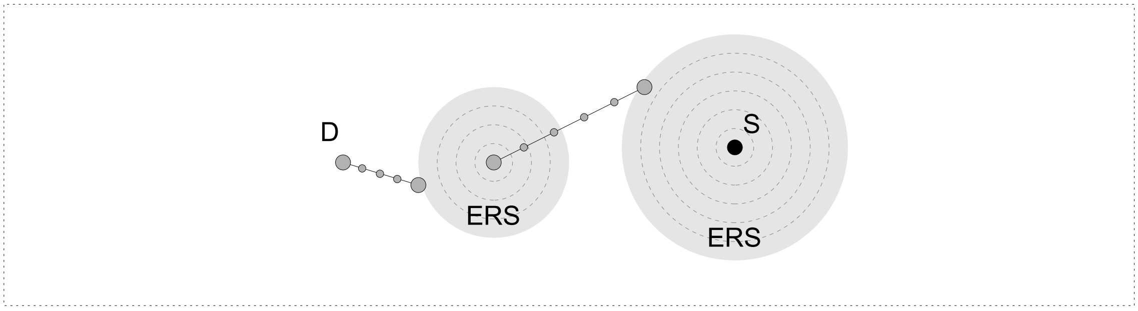


Figure 7.1: *The* ERS-BREEDINGSNAKE *algorithm. Instead of using snakes, the search phase uses expanding ring search (ERS).*

7.2 ERS-MOBILEBREEDINGSNAKE Algorithm

For very important or indispensable communication (e. g., for emergency calls), one may not want to trade-off success probability against a cheaper path discovery algorithm. The goal is to establish a call by all means.

In order to discover existing paths with unit probability, the search phase can be replaced by an algorithm using expanding ring search (ERS) [15]. If a node **S** is looking for a path to **D**, it floods the network using ERS until an advertisement for **D** is found. It then follows the snake to the destination **D**. If the snake is incomplete (e. g. nodes have moved away or disappeared), ERS is launched again to find a new advertisement that has not been seen yet by the current search. The search snake packet must therefore carry a list of all seen nodes with an advertisement for **D**. Figure 7.1 schematically shows a run of this algorithm.

It can be proven that this algorithm finds a path if there exists one.

Theorem 4. If an ideal MAC layer is assumed and if the ERS algorithm is not restricted to a maximum hop radius, the ERS-MOBILEBREEDINGSNAKE algorithm discovers existing paths with unit probability.

Proof. Consider a network of N connected nodes and let U be the set of all nodes visited by the current search packet.

At each node, the algorithm performs one of the following steps:

1. **Following a snake.** If the next node along the snake can be reached, the search packet follows the snake. The new node is added to U if it is not there yet.

2. **ERS.** Otherwise, ERS guarantees to find a next node that has not been seen yet. The found node is added to U and U therefore grows by exactly one element.

Since snakes do not contain cycles (by construction), the algorithm eventually performs ERS steps. As soon as $|U| = N$ (or before), we have $\mathbf{D} \in U$ because **D** is part of the network. Therefore, a path has been discovered. □

This algorithm is comparable to multi-step flooding [16] except that it follows the snakes as long as possible. If the advertisements are well distributed in the network, the cost of this search algorithm is reasonable. In the worst case, however, it is more expensive than simple flooding. ERS especially reveals bad performance if the destination node is not known in the network. The search will necessarily flood the whole network in that case. This also happens for temporarily unavailable or disconnected nodes which still have some advertisements in the network.

Since ERS is used in the search phase only, this scheme can easily be combined with other snake algorithms as well. It can even coexist with another snake search algorithm. In a voice communication network, for example, the usual snake search algorithm could be used for ordinary calls whereas the ERS algorithm would be chosen for emergency calls.

7.3 Discussion

7.3.1 Snake Path Discovery vs. Proactive Routing Algorithms

Early research in ad hoc networks came up with proactive routing algorithms (e. g. DSDV [20]). These algorithms update their routing tables in a proactive manner, i. e. when a link between two nodes breaks, the two nodes would inform the network that this link is not available any more. Because of the number of messages that such algorithms exchange to keep their routing tables up to date, proactive algorithms have been shown to be expensive (with regard to bandwidth and energy) in ad hoc networks.

Snake algorithms also require the nodes to periodically send information to other nodes in the network. There are, however, important differences:

- ▷ **No trigger.** Advertising snakes are not triggered by broken links. Hence, a moving node does not cause each old and new neighbor to take action.
- ▷ **No flooding.** Advertisements are sent to a small part of the nodes only.
- ▷ **No path optimality.** Snake path discovery does not give any guarantees about path optimality. Proactive routing algorithms typically do so.
- ▷ **Advertisements are hints.** Advertisements are considered as hints that help discovering a path. Proactive routing protocols typically rely on the information stored in the routing tables.

This last point expresses a fundamental difference in the two approaches. Where proactive protocols try to deterministically solve the problem, the snake algorithms provide a cheap heuristic method. There is an interesting analogy in the field of algorithmics: NP-complete problems, such as the traveling salesman problem or the graph coloring problem, are believed to be hard to solve exactly (deterministically) [30]. Nevertheless, good and relatively cheap heuristics exist for many of these problems.

Chapter 8
Optimizations

In the two previous chapters, we have presented a several algorithms based on the RANDOM-ENCOUNTER concept and partly analyzed them. All algorithms so far were bounded by the theoretical optimum derived in Chapter 3.

In this chapter, we present three optimization strategies. While the first two strategies still stay within the RANDOMENCOUNTER concept, the third breaks the borders of this concept to beat the theoretical optimum.

8.1 Heterogeneous Advertisement Strategies

In the algorithms in Section 6.2 and Chapter 7, each node advertises itself to the network in the same way as all other nodes. This is not necessary; nodes may use different advertisement strategies according to their importance or role in the network.

To place more advertisements in the network, a node can send longer or more snakes than the other nodes. When the BREEDINGSNAKE advertisement algorithm is employed, the number of breeding or continuation steps can be increased as well. The more advertisement a node places, the higher is the probability of discovering a path to this node.

Three exemplary situations where this could be useful are briefly discussed in the following paragraphs. Note that this list is not exhaustive and many more examples could be given.

Wireless network connected with a gateway. An ad hoc network may be connected to other networks (e. g. the Internet) through one or more gateways. Since most nodes will communicate with the gateways, it makes sense to advertise them better than normal nodes.

Disaster recovery. In a disaster recovery mission, there are likely to be important nodes that need to be available. (These nodes are not necessarily searched more often.)

Figure 8.1 depicts a mission with three teams and the headquarters in the center. The teams need communication services within their area and although all three teams are connected to each other, inter-team communication may be less important. The headquarters, however, need to be reachable by the nodes of all teams. It therefore makes sense to advertise the nodes at the headquarters better.

Uneven energy supply. If a couple of nodes in the network provide the same service, but have different energy supply, it makes sense to advertise the nodes with respect to their energy reserves. Nodes looking for that specific service would then find nodes with a good energy supply with higher probability.

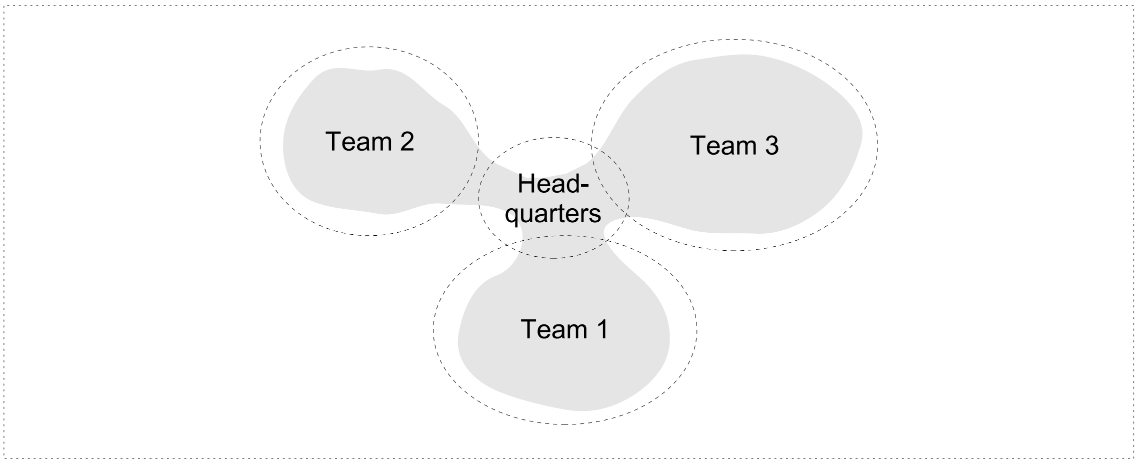


Figure 8.1: *Possible disaster recovery situation. Communication between teams is less important than communication inside a team. The headquarters need to communicate with all teams.*

8.2 Overhearing Search Snakes

Many MAC layers for wireless networks allow a node to overhear messages sent to other nodes in its vicinity. Hence, nodes hear search snake packets that their neighbors transmit to other nodes. If a node happens to carry an advertisement of a desired destination node, it can opportunistically [31] reply to report the encounter. A sample situation is drawn in Figure 8.2.

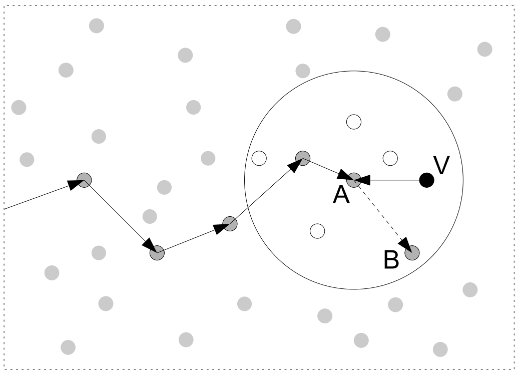


Figure 8.2: *Overhearing a search snake packet.* ***A*** *forwards a snake packet to* ***B*** *and* ***V*** *overhears this transmission. Since* ***V*** *carries an advertisement for the searched destination, it opportunistically replies.*

While saving network bandwidth, the cost of this optimization is additional processing at each node in the vicinity of a search snake. Bandwidth is therefore traded off against energy.

In some very specific situations, a similar strategy may be useful for advertisements snakes as well. If a node overhears an advertisement snake packet of a node that it wants to contact soon (if such knowledge is available), it can store the appropriate advertisement as well.

8.3 The FRIENDNODESNAKE Algorithm

If one looks for someone and does not find him or her, it is a good strategy to ask a friend of that person. Since most people have several good friends, the probability of reaching at least one of them is high. Not all friends know where the person can be found, of course, but the chances of finding the person in that way are good.

The same strategy can be applied in ad hoc networks. All we need is a *friend function* $f(\mathbf{W})$ mapping a node **W** to a set of friend nodes. This function should be easy to calculate from the ID of a node, such that each node in the network can determine the friends of any other node in a straightforward manner.

Such a function can be build using the XOR operator ($\oplus$) and the Hamming weight, for example. Assume a network of N nodes in which each node in the network can be identified by a unique number from the set $\{0,...,2^m-1\}$. We define the friend function as

$$f(\mathbf{W}) = \{\mathbf{X} \mid h(\mathbf{W} \oplus \mathbf{X}) = 1\} \tag{8.1}$$

where $h(\cdot)$ the hamming weight. In words, two nodes are neighbors if their node IDs in binary representation differ in one position only (Hamming distance). Each node therefore has a maximum of m friends and friendship is symmetric, i. e., if $\mathbf{W}_2$ is a friend of $\mathbf{W}_1$, then $\mathbf{W}_1$ is a friend of $\mathbf{W}_2$.

Other suitable functions may arise from deterministic random number generators. The friend node IDs would be the first m random numbers when the generator is seeded with the node ID. Similar friendship sets can be constructed with hash functions, iteratively calculated from the node ID. Since such friendship sets consist of random sequences that are independent of each other, the friend relationship won't be symmetric in these cases.

8.3.1 The FRIENDNODESNAKE search algorithm

The FRIENDNODESNAKE search algorithm works as follows: A node **S** looking for **D** sends out a couple snakes. If one of the snakes finds an advertisement for **D**, the algorithm follows this advertisement and proceeds in the same way as the simple SNAKE search algorithm. If no hint for **D** but an advertisement for one of its friends, say **F**, is found, the path towards **F** is followed. Node **F** then starts a search for **D** unless it already knows a path to **D**.

Subsequent searches for **D** will succeed with a higher probability, as **F** already knows a path to **D**. And if the friendship function is symmetric, **F** benefits as well: since node **D** knows a path to **F**, searches for **F** will succeed with an increased probability.

8.3.2 Analysis

If the path discovery probability to a single node is p, then the probability of finding at least one out of m friends of **D** is

$$p_1 = 1-(1-p)^m$$

Assuming that the selected friend node **F** already knows a path to **D** with probability η, the probability of discovering a path through **F** is

$$p_2 = \eta + p(1-\eta)$$

The path discovery probability using friend nodes is therefore

$$p_f = 1-(1-p)(1-p_1p_2) = 1-(1-p)(1-(1-(1-p)^m)(\eta+p(1-\eta)))$$

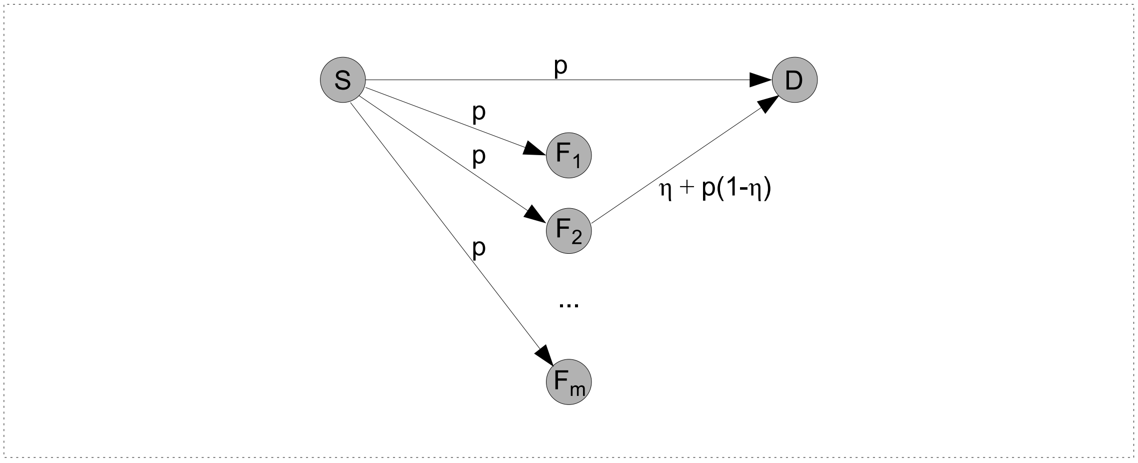


Figure 8.3: FRIENDNODESNAKE *model.* η *is the probability that the contacted friend node knows a path to* ***D*** *already.*

Note that this equation assumes that the nodes and their advertisements are uncorrelated. Such an assumption is reasonable because the friends are chosen using a mathematical function on the node ID and these node IDs are randomly spread in the network. It would not be valid, however, if the node neighbors were selected as friends, for example.

p_f increases linearly with η between the minimum

$$p_f = 1-(1-p)(1-p(1-(1-p)^m)) \qquad\qquad \text{if } \eta = 0$$

and the maximum

$$p_f = 1-(1-p)(1-(1-(1-p)^m)) \qquad\qquad \text{if } \eta = 1$$

The graphs for some selected parameters are drawn in Figure 8.4 and Figure 8.5. They shows that high path discovery probabilities can be reached with high values for η and many friend nodes, even if the path discovery probability without friends, p, is small (e. g. 50 %).

For high values of p, the gains are less substantial. In particular, a few friend nodes are enough to raise the probabilities.

8.3.3 Optimal Performance - a Contradiction?

In Chapter 3, we derived the theoretically optimal performance of the RANDOMQUERY algorithm. Assume we use this algorithm in combination with the FRIENDNODESNAKE search algorithm. If p is the theoretically achievable path discovery probability, then $p_f \geq p$ beats this limit. This may look astonishing at first sight, but there are three important differences:

Friend relationship. The RANDOMQUERY algorithm assumes that there is no structure whatsoever in the network. The derived optimal probability therefore applies to unstructured networks only. Using a friend relationship introduces a structure in the network.

Several searches. The RANDOMQUERY algorithm performs a single search for a single node and reports failure if the destination node was not found. The FRIENDNODESNAKE algorithm searches for multiple nodes at the same time. If the destination node was not found, it performs an additional search at one of the friend nodes.

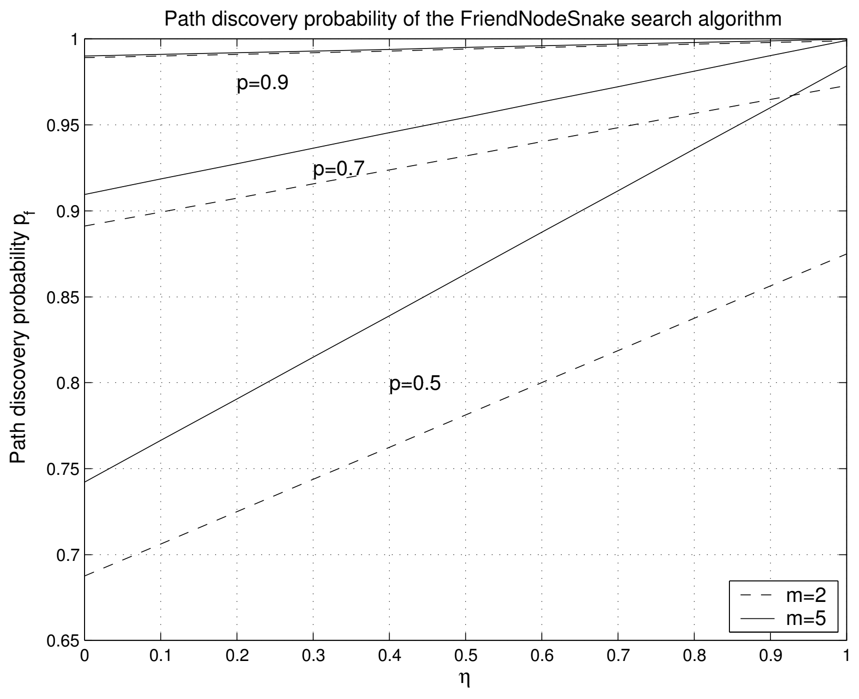


Figure 8.4: *Path discovery probability of the* FRIENDNODESNAKE *algorithm in function of* η.

Cached (known) paths. The RANDOMQUERY algorithm executes each search independently. It does not take advantage of similar paths that were previously discovered. The FRIEND-NODESNAKE algorithm caches routes between friend nodes.

8.3.4 Distributed versus Centralized

An interesting result is found if all nodes share the same friend node **F**. Formally,

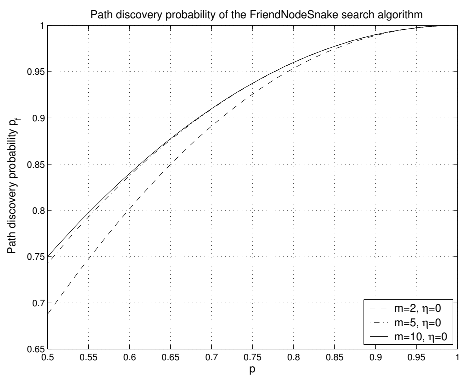
$$f(\mathbf{W}) = \{\mathbf{F}\} \qquad\qquad \forall\, \mathbf{W} \text{ in the network}$$

F can then be regarded as a rendezvous point [18]. Since each node only needs to stay in touch with this node, high path discovery probabilities could be achieved at relatively low cost. This, however, is a centralized scheme rather than a distributed scheme and has two important disadvantages:

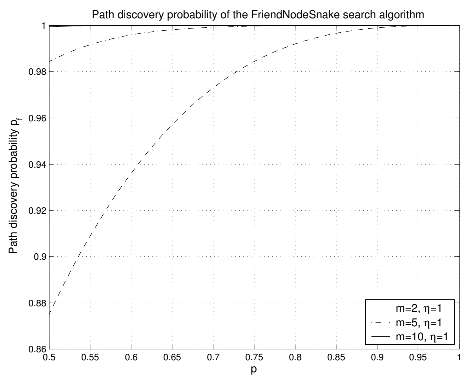
- ▷ **Scalability.** It does not scale since **F**'s traffic increases linearly with the number of nodes.
- ▷ **Dependence.** The network heavily depends on node **F** and possibly experiences a substantial degradation when **F** fails or disconnects.

Both problems should be avoided in ad hoc networks. One of the goals of ad hoc networks (and self-organization in general) is to avoid such dependence and to react smoothly to errors and failure.

Nevertheless, this shows that the FRIENDNODESNAKE algorithm can be used to build fully distributed as well as centralized or semi-centralized protocols.



(a) $\eta = 0$



(b) $\eta = 1$

Figure 8.5: *Path discovery probability of the* FRIENDNODESNAKE *algorithm as a function of the path discovery probability without friend nodes. (a) The friend nodes do not know a path to the destination node. (b) All friend nodes know a path to the destination node.*

8.3.5 Variants

In large networks, the FRIENDNODESNAKE search algorithm could be extended to consider second-order friends (friends of friends) or higher-order friends. If the transitive closure of the friend relationship includes all nodes of the network, all nodes can be reached through friends. With the friend relation based on equation (8.1), for example, all nodes can be reached within m steps or less.

In addition, the paths between friends could be maintained proactively in order to have $\eta \approx 1$. Due to the cost of such an operation, this probably makes sense in large networks with very low mobility only or if the proactive maintenance is restricted to some important nodes only.

Chapter 9
A Geometric Approach

When analyzing and discussing the path discovery probability in Chapter 3 and Chapter 4, we regarded the network as a set of nodes. With that approach, we could determine the expected probability of discovering a path between two nodes independent of their relative or absolute position in the network. These equations assumed that advertisements are uniformly distributed and searched in the network.

This assumption, however, is difficult to fulfill. In the analysis of Chapter 5, we have seen that algorithms based on snakes will advertise and search their neighborhood in a non-uniform way. The density of advertised nodes is function of the distance to the destination node **D**. Similarly, the density of queried nodes is function of the distance to the source node **S**.

In this chapter, we therefore analyze the path discovery probabilities using a model that respects these properties. After stating the assumptions, we present two models: the overlapping circles model (Section 9.2) and the more general overlapping rings model (Section 9.3).

9.1 Model and Assumptions

We assume an infinite network with homogeneously distributed nodes. The node density, λ, must be high enough for the *geometric approximation* to be valid.

By the geometric approximation, we refer to the assumption that the physical distance between two nodes increases approximately linearly with the number of hops of the shortest path connecting them. Intuitively, this seems to hold for high node densities. Figure 9.1 shows two hop-limited floods in networks with different node densities. In the dense network, the flood covers an almost circular area whereas its shape is very irregular in the other network.

In a network with these properties, we select two nodes **S** and **D** and denote their physical distance by d. The destination node **D** places advertisements at random nodes **X** with probability $p_A(\|r_\mathbf{D} - r_\mathbf{X}\|)$, where $\|r_\mathbf{D} - r_\mathbf{X}\|$ stands for the physical distance between **D** and **X**. Similarly, the probability that **S** searches at a node **Y** depends on the distance of these two nodes and is denoted $p_S(\|r_\mathbf{S} - r_\mathbf{Y}\|)$.

9.2 Overlapping Circles Model

Let us first consider the case where the advertisements are put and searched in a circular area around the destination node **D** and the source node **S**, respectively, with a constant probability. If this constant probability is inversely proportional to the circle area, we assure that the expected number of advertised and queried nodes remains constant for a any circle radius, $r_{\max}$. Hence, we define

$$p_A(r) = p_S(r) = \begin{cases} \frac{c^2}{r_{\max}^2} & \text{for } r \le r_{\max} \\ 0 & \text{otherwise} \end{cases}$$

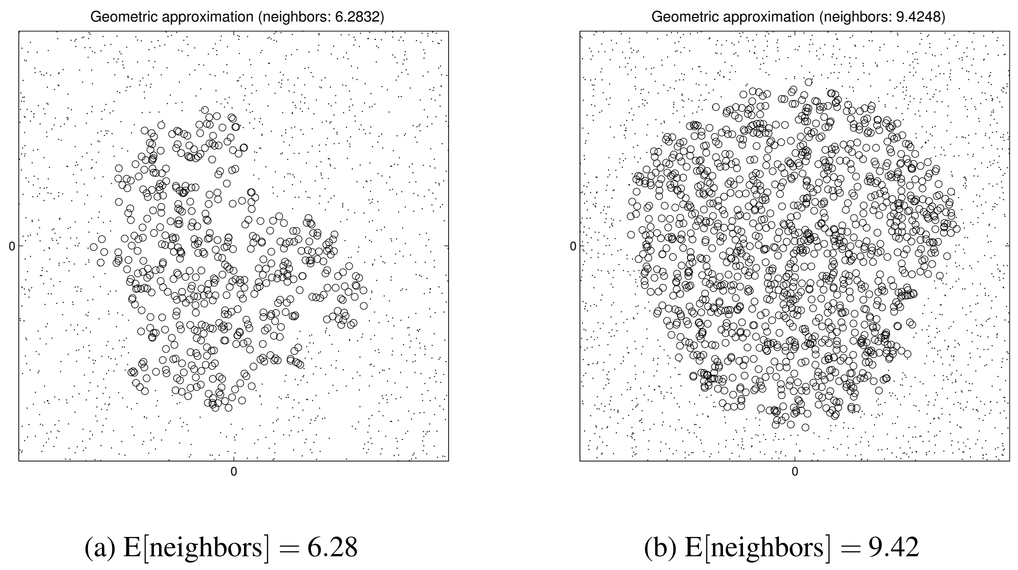


Figure 9.1: *Random networks with a hop-limited flood. For high node densities, the reached area is almost circular.*

To assure that the density does not exceed 1, $r_{\max} \geq c$ is required. The expected number of advertised and queried nodes is

$$\bar{a} = \bar{s} = \pi r_{\max} \frac{c^2}{r_{\max}} = \pi c^2$$

and therefore independent of $r_{\max}$, as desired. In this model, we are interested in the path discovery probability in function of d and $r_{\max}$.

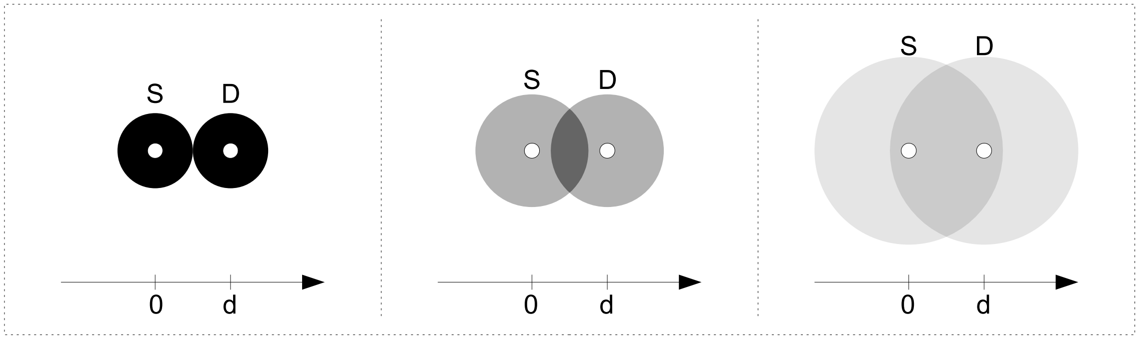


Figure 9.2: *Overlapping circles model. When increasing the circle radius, $r_{\max}$, and decrease the advertisement and search density to keep the number of advertised and queried nodes constant.*

9.2.1 Path Discovery Probability

The path discovery probability depends on the overlapping area as well as on the density. Since the nodes are homogeneously distributed, the number of nodes in a given area is Poisson distributed. Applying equation (3.6) of the RANDOMQUERY algorithm on the overlapping area, we obtain

$$\ln q = A_c(d, r_{\max}, r_{\max}) \lambda \ln \left(1 - \frac{c^2}{r_{\max}^2} \cdot \frac{c^2}{r_{\max}^2}\right)$$

where $A_c(d, r_{\max}, r_{\max})$ refers to the the circle intersection area derived in Section B.1.

9.2.2 Results and Discussion

Figure 9.3 shows the path discovery probability in function of d for several values of $r_{\max}$. We have chosen $c = 1$.

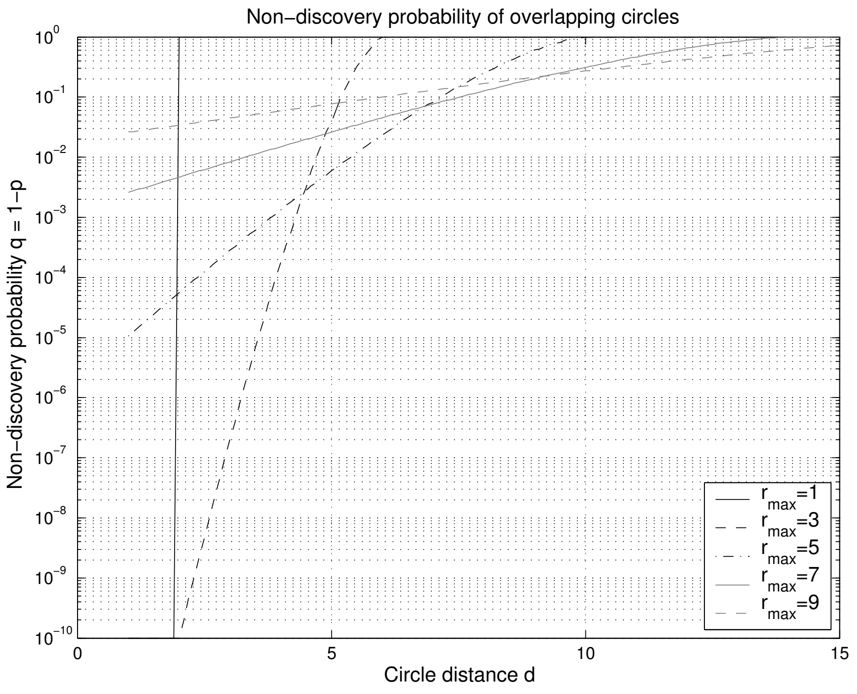


Figure 9.3: *Path discovery probability in the overlapping circles model with $c = 1$ for various $r_{\max}$. The optimal value for $r_{\max}$ depends on the desired probability.*

Obviously, we can reach more nodes by increasing the circle area. Decreasing the advertisement and search density, however, decreases the probability of path discovery. The optimal value for $r_{\max}$ therefore depends on the desired probability. If we aim a path failure probability of at most $q = 10^{-2}$, for example, $r_{\max}$ should be chosen at around 5 (if $c = 1$), as this maximizes the distance d.

But more interestingly, this model allows us to compare flooding to probabilistic path discovery. If $r_{\max} = c = 1$,

$$p_A(r) = p_S(r) = \begin{cases} 1 & \text{for } r \leq c \\ 0 & \text{otherwise} \end{cases}$$

This corresponds to hop-restricted flooding, achievable using ERS [15] for example. Path discovery is deterministic in this case, but the reachable area is minimal, obviously. On Figure 9.3, this case is represented by the vertical line at $d = 2c = 2$.

If we increase $r_{\max}$, we rotate this line (note that it is not a straight line any more) towards the horizontal axis. For $r_{\max} = \infty$, the line would end up being horizontal at $q = 1 = 10^0$. This case, however, is not interesting because we would never discovery any paths.

From Figure 9.3, it is clear that restricted flooding is the optimal choice if and only if we require deterministic path discovery, i. e., $q = 0$. In all other cases, probabilistic path discovery is advantageous. Although this may look like a simple play with numbers, it raises serious doubts about whether flooding makes sense to discover paths in ad hoc networks. As mentioned in Section 6.4.1 already, wireless networks are inherently probabilistic. If time is limited – which is always the case in real-world scenarios – flooding cannot (mathematically) guarantee discovering a path if one exists. Therefore, q is always bigger than zero.

From a more pragmatic point of view, flooding can possibly achieve very high path discovery probabilities. If $q = 10^{-20}$, for example, the optimal value for $r_{\max}$ would be close to 1, making it clearly not worth using a more complicated algorithm. Yet, probabilities as low as this are hardly ever demanded in path discovery. The failure rate due to other effects in a wireless network is much higher than that. And since the curves are steep at the beginning, we can hope for a quite substantial gain by lowering our expectations.

9.3 Overlapping Rings Model

In the overlapping circles model, we have chosen a specific functions for $p_A(r)$ and $p_S(r)$. In practical cases, such uniform densities and sharp cut-offs require expensive algorithms. We would therefore like to generalize the calculations for any functions $p_A(r)$ and $p_S(r)$. Theoretically, this can be done with the equation for continuous heterogeneous densities (equation (3.7)) derived for the RANDOMQUERY algorithm. But because of the circular shape of the densities, the formulas quickly become complicated even for simple density functions (e. g., linear functions). Finding analytical solutions is hardly possible, if at all.

We therefore propose a numerical approximation using piece-wise constant density functions over a finite domain. Formally, we define

$$p_A(r) = \begin{cases} p_{A,i} & \text{for } r_{A,i-1} \leq r < r_{A,i} \\ 0 & \text{otherwise} \end{cases}$$

for $i \in \{1, 2, ..., n_A\}$ and $r_{A,0} = 0$. Similarly,

$$p_S(r) = \begin{cases} p_{S,j} & \text{for } r_{S,j-1} \leq r < r_{S,j} \\ 0 & \text{otherwise} \end{cases}$$

for $j \in \{1, 2, ..., n_S\}$ and $r_{S,0} = 0$. Each constant piece of these functions generates one homogeneous ring. Hence, we end up with two series of rings, overlapping with each other.

9.3.1 Path Discovery Probability

Each pair of overlapping rings generates an area[1] with homogeneous path discovery parameters. In logarithmic notation, we can sum up these contributions. Using equation (3.6), the path discovery failure probability can therefore be expressed as

$$\log q = \sum_{i=1}^{n_A} \sum_{j=1}^{n_S} A_r(d, r_{A,i-1}, r_{A,i}, r_{S,i-1}, r_{S,i}) \lambda \ln(1 - p_{A,i} p_{S,j})$$

where $A_r(d, u_1, u_2, v_1, v_2)$ denotes the intersection area of two rings distant by d with radii u_1 and u_2, and v_1 and v_2, respectively. Section B.2 gives an analytical expression for this function.

[1]This area may consist of zero, one or two pieces, depending on how the rings are overlapping.

9.3.2 Results and Discussion

The overlapping rings model allows us to compare the path discovery probability for various density function shapes. In the following paragraphs, we present and discuss three cases. The density functions were normalized to obtain the same $\bar{a} \cdot \bar{s}$ product. This allows us to compare the shapes not only qualitatively, but quantitatively, too.

9.3.2.1 Two Equally-Sized Circles

To begin with, let us verify the the path discovery probability of two overlapping circles. As shown in Figure 9.4, the shape matches with the one calculated using the circle overlapping model and drawn in Figure 9.3. This shape is not very good, however. For short distances, the probability is unnecessarily high whereas for long distances, it slowly fades out. There is no sharp cut-off. A 99 % probability ($q < 10^{-2}$) is reached up to $d = 14$. At $d = 24$, the probability drops below 90 %.

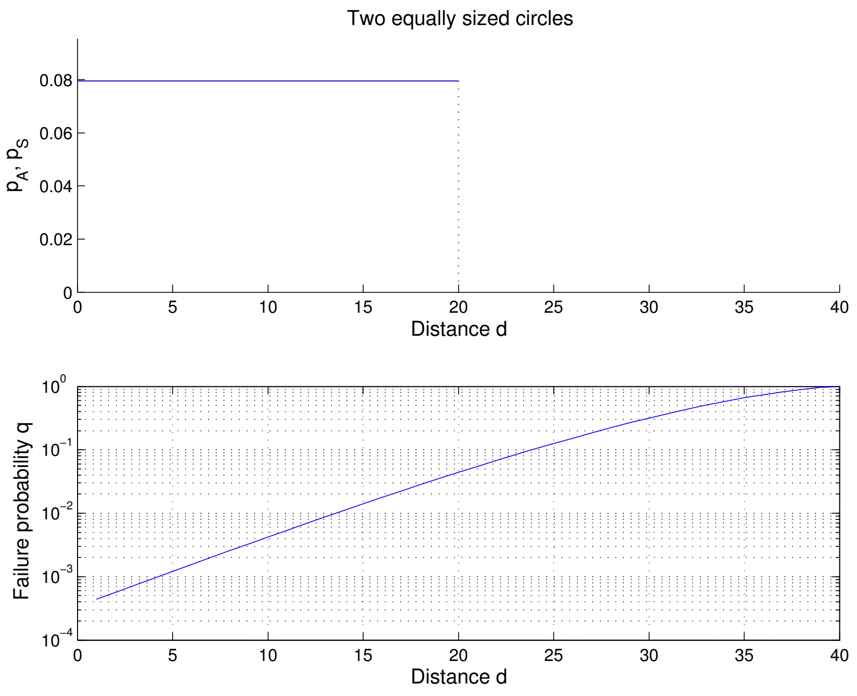


Figure 9.4: *Path discovery probability of two overlapping equally-sized circles.*

9.3.2.2 Two Equally-Sized Rings

The shape of the path failure probability generated by two overlapping rings looks more promising. As seen on Figure 9.5, the probability is almost constant at about 99 % over a wide range and the cut-off is much sharper than with circles. For very low distances, the path failure probability drops to very low values. This is due to the fact that the rings overlap almost completely. The probability drops below 99 % at $d = 21$ and below 90 % at $d = 24$.

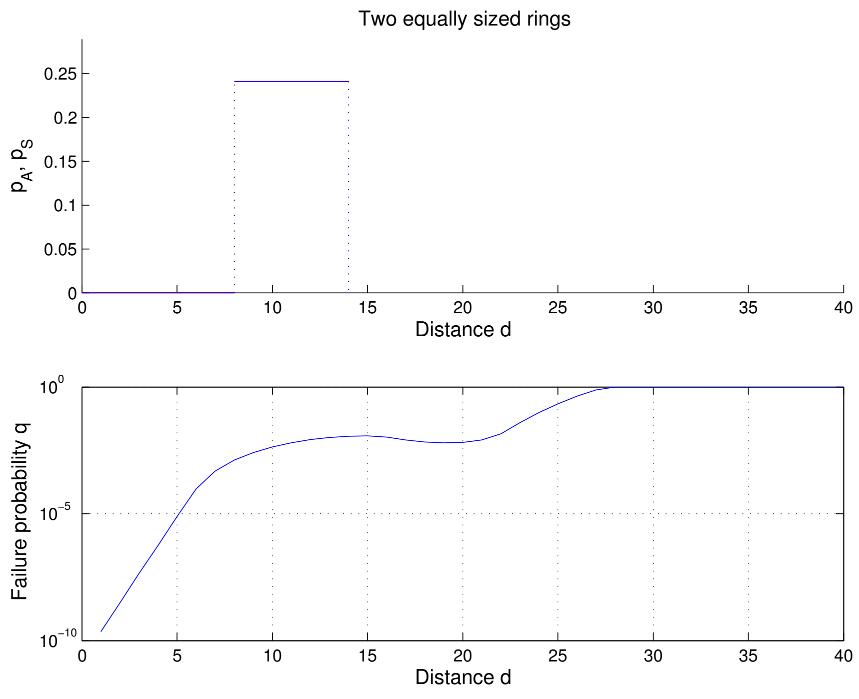


Figure 9.5: *Path discovery probability of two overlapping equally-sized rings.*

9.3.2.3 Big and Small Circle

Interesting shapes can also be obtained with circles of different sizes. Figure 9.6 shows the shape for a big circle and a small circle with decaying density. When normalizing, we gave the big circle 16 times the weight of the small circle.

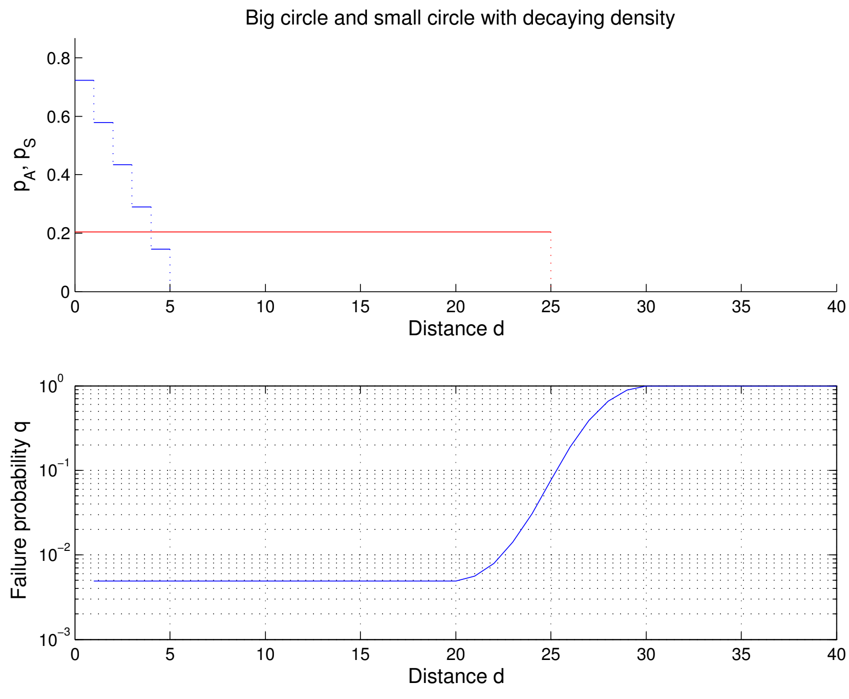
As long as the small circle is completely within the big circle, the probability remains constant. When the small circle moves out of the big one, the path discovery probability quickly drops. In the example on Figure 9.6 (a), a path discovery probability of 99 % can be maintained up to $d = 22$ and the 90 % bound is crossed at $d = 25$. On Figure 9.6 (b), the cut-off is less steep: 99 % is found at $d = 19$ and 90 % at $d = 25$.

This case models the SNAKE-BREEDINGSNAKE algorithm in a very simplified way: advertisements are placed far away from the destination node using breeding snakes whereas the search is done with comparably short search snakes.

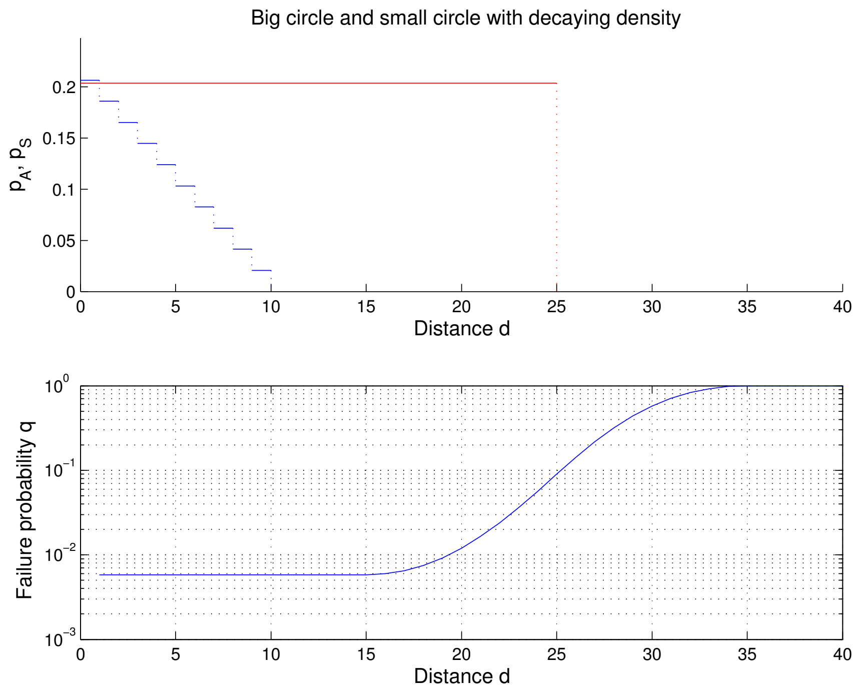
9.3.3 Discussion

The overlapping rings model offers an alternative way of studying path discovery probabilities. Its assumptions are not very realistic, though. It us unlikely that large networks with uniformly distributed nodes will be deployed. Moreover, we usually want to be able to discover path between all nodes in a network, not only between nodes that within a certain distance of each other.

Nevertheless, it gives us a useful tool to study and compare – qualitatively and quantitatively – different advertisement and search strategies in a top-down manner. And last but not least, it allows us to develop some intuition for good and bad densities.



(a)



(b)

Figure 9.6: *Path discovery probability obtained with a big circle and a small circle with decaying density.*

Chapter 10 Applications of Path Discovery Algorithms Based on Snakes

So far, we have presented and analyzed various algorithms based on snakes and the RANDOM-ENCOUNTER concept. In this chapter, we discuss selected real-world applications where snakes could be useful.

In general, snake algorithms are more suitable for medium and large scale ad hoc networks. Recall from Chapter 3 that the number of nodes to advertise and search can be approximated by $c\sqrt{N}$ where c is a constant depending on the path discovery probability. For small networks (up to 20 nodes), $c \approx \sqrt{N}$ and therefore $c\sqrt{N} \approx N$. The gain as compared to simple flooding-based algorithms (e. g. AODV [11], DSR [12]) is marginal. In bigger networks, however, $c \ll \sqrt{N}$, making the snake algorithms much cheaper than flooding.

Furthermore, snake algorithms are suited for networks with moderate and low node mobility. In networks with no mobility at all, snake algorithms may be used as well. In case of high node mobility, the snakes will be destroyed quickly after creating them. Purely reactive protocols therefore work better in such scenarios.

Note that is is not required that all nodes travel at moderate speed. If a single node moves fast, it mainly decreases its own probability of being found in the network. The probability of discovering the other nodes won't suffer a lot.

In the following, we first discuss two concrete applications: sensor networks and multi-hop mobile phone networks. Our purpose is to give a short overview and to explain why snakes could be useful in these networks. A complete study is left for future work.

Finally, we show how snakes can be combined with location based protocols.

10.1 Mobile Sensor Networks

Today's probably largest deployments of ad hoc networks are sensor networks. Some of them are static with occasional short link failures. Others, however, include mobility or frequent topology changes due to environmental conditions. In a forest, for example, the RF propagation depends on the humidity, which changes throughout the day and with the weather. Links may therefore break or reappear just as if the nodes had moved around.

The goal of most sensor networks is to transmit the collected data to a sink. Therefore, only nodes acting as sinks need to advertise themselves in the network. If it does not matter to which sink to transmit the data, a node can simply search for any advertisement in the network and follow the corresponding snake towards the sink. In that case, the total number of advertisement (placed by all nodes) in the network defines the path discovery probability.

Snakes are particularly well-suited for sensor networks in which very few nodes need to transmit data to a sink. This could be the detection of rare events in a big area, for example. In such networks, it is expensive to maintain routes from all nodes to their sinks. Heuristic path discovery with snakes provides a much cheaper solution. Furthermore, path optimality is not a big concern since the amount of data is usually small. In fact, instead of setting up a path, the data could directly be included in the search snake packet.

We propose to use the MOBILESNAKE-MOBILEBREEDINGSNAKE algorithm for mobile networks (see Section 7.1). The parameters depend on the size and shape of the network. Since the advertisement density decreases with increasing distance from the sinks, it is advantageous to have the sinks in the center of the network rather than at the edge. If several nodes act as a sinks, these nodes should be well distributed over the whole network in order to achieve optimal performance.

10.2 Multi-hop Mobile Phone Networks

In today's mobile phone networks, a mobile device always communicates with a base station. Mobile phones never transmit packets directly to each other. This implies that the desired service area must be covered completely by base stations. To lower the amount of necessary base stations, the deployment of multi-hop mobile phone networks has been proposed [5]. In such networks, a mobile phone would use other mobile phones as relays to reach either a base station or the destination mobile phone.

While stochastic routing would be a suitable routing strategy, snake algorithms could be used for path discovery in such networks. We propose to use the MOBILEBREEDINGSNAKE advertisement algorithm to advertise the mobile devices as well as the base stations. Since base stations play an important role and have completely different energy constraints, it makes sense to advertise them more heavily.

A slightly modified search algorithm must be used for such networks. A node **S** looking for a **D** launches search snakes to find either an advertisement for **D** or an advertisement for a base station. In the latter case, the base station routes the packet through the backbone network of the provider to the base station closest to **D**, where another set of search snakes are sent to find an advertisement for **D**. For this scheme to work, each node must register itself at a nearby base station from time to time.

10.3 Location Based Protocols

The RANDOMENCOUNTER concept as well as the snake network primitive can be combined with a number of other protocols. In this section, we briefly discuss how the snake algorithms can be used in combination with location based routing strategies.

If all nodes know their own geographic position in the network (e. g. using GPS or a position estimation method [32] [33]), a good path can be discovered by using a geographic or Cartesian routing protocol (e. g. LAR [17]). Nevertheless, there are two problems with this approach:

- ▷ The position of the destination **D** node must be known when a path is build.
- ▷ In case of mobility, the exact position of the destination node cannot be known. At the estimated position, a local search is necessary.

To solve both these points, snake algorithms can be used.

The first problem can be solved by using snake algorithms as a probabilistic distributed location service. Each node uses the MOBILEBREEDINGSNAKE algorithm to periodically advertise its current position to some other nodes. In contrast to the usual MOBILEBREEDINGSNAKE algorithm, however, an advertisement only consists of the position of the advertising node together with a timestamp. A node **S** looking for a destination node **D** then searches an advertisement for **D** using snakes. The position of the most recent advertisement found is used to construct a path to the destination node. Note, however, that more sophisticated distributed location services have been proposed. GLS [18], in particular, achieves higher performance by using the position information for the location service part as well.

To solve the second problem, each node can periodically send out a couple of short snakes to advertise itself in the surroundings. The length of these snakes can be chosen as a function of the node's speed. The local search at the end of geographic routing algorithms is then performed by sending search snakes.

Chapter 11
Conclusion

11.1 Summary

In this work, we studied probabilistic path discovery algorithms for ad hoc networks based on the RANDOMENCOUNTER concept.

We first introduced two theoretical algorithms (the RANDOMQUERY and the FIXRANDOMQUERY algorithm) and could analytically prove that path discovery is possible using $O(\sqrt{N})$ nodes only. This is substantially better than flooding techniques which typically contact $O(N)$ nodes for path discovery. The result is valid if and only if advertisements are placed and searched uniformly among all nodes, which is not easy to implement in a cheap way.

To place and search advertisements in a random (although non-uniform) fashion, we introduced a network primitive that we call *snakes*. Snakes are random paths built with unicast packets. A neighborhood list allows them to reach distant areas.

We then presented three snake-based path discovery algorithms for static networks and analyzed them through simulations. With regard to the path discovery probability, the SNAKE-BREEDINGSNAKE algorithm showed the best performance. In the chosen simulation network, the best results required only about $\sqrt{2}$ more nodes to be contacted as compared to the theoretical optimum. The drawback of all snake algorithms is the length of the discovered paths. Since the algorithms do not give any guarantee about path optimality, the discovered paths can be significantly longer than the optimal path and even be much longer than the network diameter. Probabilistic path discovery algorithms are therefore best suited in combination with (stochastic) routing algorithms that continuously optimize paths. One such algorithm is AntHocNet [25].

For ad hoc networks with mobility and node failure, we presented the MOBILESNAKE-MOBILEBREEDINGSNAKE algorithm. Both the advertisement and the search phase are slightly more complicated in order to deal with broken snakes and aging advertisements. We also proposed a search algorithm based on ERS flooding to obtain a deterministic path discovery algorithm. Furthermore, we introduced and analyzed the FRIENDNODESNAKE search algorithm as an optimization technique. This algorithm is able to beat the theoretical maximum performance of the RANDOMENCOUNTER concept by adding a lightweight probabilistic structure to the network. The evaluation and verification of these algorithms in a network simulator is left to future work.

Finally, we approached probabilistic path discovery in large ad hoc networks from a geometrical point of view. Within the overlapping circles model, we could show that flooding is optimal if and only if deterministic path discovery is required.

11.2 Future Work

Figure 11.1 summarizes what we have shown in this thesis and proposes future research directions in the area of probabilistic path discovery.

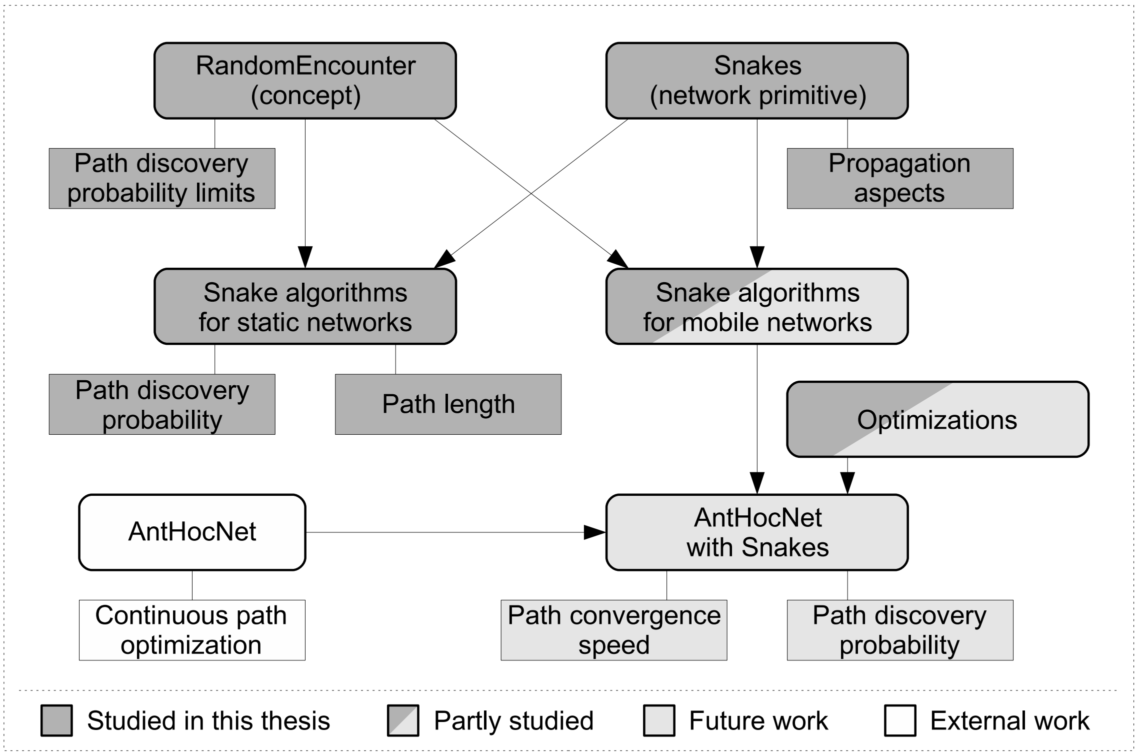

RandomEncounter (concept)
Snakes (network primitive)
Path discovery probability limits
Propagation aspects
Snake algorithms for static networks
Snake algorithms for mobile networks
Path discovery probability
Path length
Optimizations
AntHocNet
AntHocNet with Snakes
Continuous path optimization
Path convergence speed
Path discovery probability
Studied in this thesis
Partly studied
Future work
External work

Figure 11.1: *Overview of achieved and future work.*

As mentioned before, the snake algorithms remain to be simulated in a scenario with node mobility and failure. Both the probability of path discovery and the resulting path length need to be evaluated under different mobility models.

Furthermore, the snake algorithms should be plugged in into a routing protocol like AntHocNet [25] to evaluate the path length convergence.

Finally, the optimization techniques were presented on the level of ideas with feasability arguments. More work is required to evaluate them in different scenarios.

11.3 Main Contribution and Conclusion

Although non-deterministic path discovery algorithms have been proposed before [29] [34], the probabilistic nature of these existing schemes is rather a side-effect than a design choice. To our knowledge, this is the first attempt to cut the costs of path discovery by leveraging on the path discovery probability.

Our attempts differ from many other proposals in that we try to minimize the number of nodes involved in path discovery instead of minimizing the number of packets. To achieve that, we do not require any additional knowledge (i. e., position) at the nodes. Hence, our algorithms can be plugged in as a replacement (not an optimization) for flooding.

Most ad hoc networks can accept a small fraction of unsuccessful path discoveries. They must so, since path discovery may fail due to simple packet loss as well. Therefore, we believe that probabilistic path discovery is extremely viable in ad hoc networks.

Appendix A
List of Symbols

Variables

a	Number of advertisements placed (random variable)
$\bar{a}$	Average number of advertisements placed
d	Physical distance
l_A	Length of an advertisement snake (number of hops)
l_S	Length of a search snake (number of hops)
m_A	Number of advertisement snakes
m_S	Number of search snakes
N	Number of nodes in the network
p_A	Probability of placing an advertisement at a specific node
p_S	Probability of searching a specific node
s	Number of nodes searched for an advertisement (random variable)
$\bar{s}$	Average number of nodes searched for an advertisement
t	Time
t_b	Time at which an advertising snake is sent (MOBILEBREEDINGSNAKE advertisement algorithm)
t_d	Lifetime (time duration) of an advertisement (MOBILEBREEDINGSNAKE advertisement algorithm)
t_l	Interval between sending breeding snakes (MOBILEBREEDINGSNAKE advertisement algorithm)
t_s	Interval between sending short, non-breeding snakes (MOBILEBREEDINGSNAKE advertisement algorithm)
w_1	Number of breeding steps (BREEDINGSNAKE advertisement algorithm)
w_2	Number of continuation steps (BREEDINGSNAKE advertisement algorithm)
w_A	Number of hops between advertisements (LONGSNAKE advertisement algorithm)

α	Quality measure for algorithms based on the RANDOMENCOUNTER concept
λ	Node density (nodes/m^2)
λ_A	Advertised node density (nodes/m^2)
λ_S	Searched node density (nodes/m^2)

Functions

$A_c(d,u,v)$	Intersection area of two circles of radii u and v. The distance between the circle centers is d.
$A_r(d,u_1,u_2,v_1,v_2)$	Intersection area of two rings. The distance between the two ring centers is d. The first ring has radii u_1 and u_2 and the second ring has radii v_1 and v_2.
$b(\Delta t)$	Breeding probability (MOBILEBREEDINGSNAKE advertisement algorithm)
$f(\mathbf{W})$	Friend relationship function (FRIENDNODESNAKE search algorithm)
$h(x)$	Hamming weight (binary) of x

Nodes

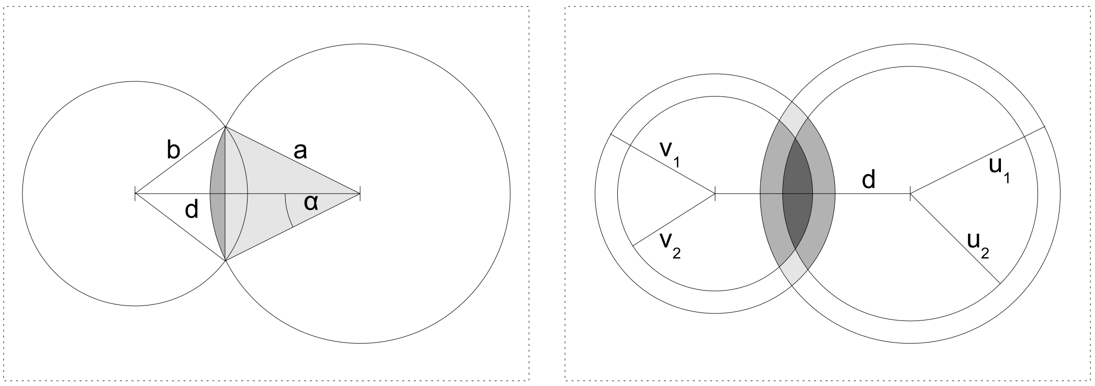
D	Destination node
F	Friend node (FRIENDNODESNAKE search algorithm)
S	Source node that wants to establish a path to **D**
V	Encounter node

Appendix B
Intersection Areas

In Chapter 9, we need the circle intersection area, $C(d,u,v)$, as well as the ring intersection area, $R(d,u_1,u_2,v_1,v_2)$, to calculate the path discovery probability. In the following sections, we derive the appropriate equations.

B.1 Circle Intersection Area

Assume two circles of radius a and b respectively. The distance between the circle centers is denoted by d, as shown in Figure B.1 (a). The intersection area consists of two chords. One of them is filled in dark grey in Figure B.1 (a). The area of the chord is the difference between the corresponding segment and the triangle indicated in light grey.



(a) (b)

Figure B.1: *(a) Schema to calculate the intersection area of two circles. (b) Schema to calculate the intersection area of two rings.*

Let us first calculate α using the equation

$$\cos(\alpha) = \frac{d^2 + a^2 - b^2}{2ad}$$

Note that the expression on the right hand side is in the range [-1, 1] if and only if $|a-b| \leq d \leq a+b$. Indeed, if $d > a+b$, the two circles don't intersect and the intersection area is zero. If $d < |a-b|$, the smaller circle is completely within the bigger circle. The intersection area in that case is the area of the smaller circle.

In the good range, the area of the section is

$$A_{\text{section}} = \pi a^2 \frac{2\alpha}{2\pi} = a^2 \alpha$$

and the area of the light grey triangle can be written as

$$A_{\text{triangle}} = \frac{2a^2 \sin(\alpha)\cos(\alpha)}{2\sin\frac{\pi}{2}} = a^2 \sin(\alpha)\cos(\alpha)$$

Hence, the area of the chord is

$$A_1 = A_{\text{section}} - A_{\text{triangle}} = a^2\left(\alpha - \sin(\alpha)\cos(\alpha)\right)$$

For reasons of symmetry, the area of the second chord is

$$A_2 = b^2\left(\beta - \sin(\beta)\cos(\beta)\right)$$

with

$$\cos(\beta) = \frac{d^2 + b^2 - a^2}{2bd}$$

The intersection area between the two circles is therefore

$$A_c(d,a,b) = \begin{cases} 0 & \text{if } d > a+b \\ \pi\min(a^2, b^2) & \text{if } d < |a-b| \\ a^2\left(\alpha - \sin(\alpha)\cos(\alpha)\right) + b^2\left(\beta - \sin(\beta)\cos(\beta)\right) & \text{otherwise} \end{cases}$$

A slightly different, but equivalent expression can be found on MathWorld [35].

B.2 Ring Intersection Area

The intersection area of two rings can be calculated with four circle intersection areas, as shown in Figure B.1 (b). If the ring radii are a_1 and a_2 (with $0 \le a_2 \le a_1$), and b_1 and b_2 (with $0 \le b_2 \le b_1$), respectively, the intersection area in function of their distance d and radii is

$$A_r(d, a_1, a_2, b_1, b_2) = A_c(a_1, b_1) - A_c(a_2, b_1) - A_c(a_1, b_2) + A_c(a_2, b_2)$$

Note that this equation also holds for special cases, e. g. if one ring is within the other ring or if the rings do not intersect.

Bibliography

[1] Bluetooth, http://www.bluetooth.com.

[2] WLAN 802.11 Working Group, http://grouper.ieee.org/groups/802/11/.

[3] M. Conti I. Chlamtac and J. J.-N. Liu. Mobile ad hoc networking: imperatives and challenges. In *Ad Hoc Networks*, volume 1, pages 13–64, July 2003.

[4] Ch. E. Perkins, editor. *Ad Hoc Networking*. Addison-Wesley, 2001.

[5] J. P. Hubaux, J. Y. Le Boudec, Th. Gross, and M. Vetterli. Toward Self-Organizing Mobile Ad-Hoc Networks: the Terminodes Project. *IEEE Communications Magazine*, 39(1):118–124, January 2001.

[6] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Mobile Computing and Networking*, pages 85–97, 1998.

[7] E. M. Royer and C.-K. Toh. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks, April 1999.

[8] T. Wysocki M. Abolhasan and E. Dutkiewicz. A review of routing protocols for mobile ad hoc networks. In *Elsevier Journal of Ad hoc Networks*, volume 2, pages 1–22, June 2003.

[9] The SECAN-LAB. Ad Hoc Protocols. http://wiki.uni.lu/secan-lab/Ad-Hoc+Protocols.html.

[10] Wikipedia, the free encyclopedia. Ad Hoc Protocol List.
http://en.wikipedia.org/wiki/Ad_hoc_protocol_list.

[11] Ch. E. Perkins and E. M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 1999)*, pages 90–100, February 1999.

[12] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. In Ch. E. Perkins, editor, *Ad Hoc Networking*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[13] B. Williams and T. Camp. Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks. In *Proceedings of the 3th ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc 2002)*, pages 194–205, June 2002.

[14] Ph. Jacquet, P. Mühlethaler, Th. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized Link State Routing Protocol. In *Proceedings of the International Multitopic Conference (INMIC 2001)*, pages 62–68, December 2001.

[15] J. Hassan and S. Jha. On the Optimization Trade-Offs of Expanding Ring Search. In *Proceedings of the 6th International Workshop on Distributed Computing (IWDC 2004)*, pages 489–494, December 2004.

[16] H. Dubois-Ferriere, M. Grossglauser, and M. Vetterli. Age matters: efficient route discovery in mobile ad hoc networks using encounter ages. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc 2003)*, pages 257–266, June 2003.

[17] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, 2000.

[18] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 120–130, August 2000.

[19] Y. Xue, B. Li, and K. Nahrstedt. A scalable location management scheme in mobile ad-hoc networks. In *Proceedings of the 26th Annual IEEE Conference on Local Computer Networks (LCN 2001)*, pages 102–111, November 2001.

[20] Ch. E. Perkins and P. Bhagwat. DSDV Routing over a Multihop Wireless Network of Mobile Computers. In Ch. E. Perkins, editor, *Ad Hoc Networking*, chapter 3, pages 53–74. Addison-Wesley, 2001.

[21] Z. J. Haas, M. R. Pearlman, and P. Samar. The Zone Routing Protocol (ZRP) for Ad Hoc Networks. Internet-draft (work in progress), IETF MANET Working Group, July 2002.

[22] P. Sinha, R. Sivakumar, and V. Bharghavan. CEDAR: Core extraction distributed ad hoc routing. In *Proceedings of the IEEE INFOCOM 1999*, volume 1, pages 202–209, March 1999.

[23] M. Joa-Ng and I-T. Lu. A Peer-to-Peer Zone-Based Two-Level Link State Routing for Mobile Ad Hoc Networks. *IEEE Journal on Selected Areas In Communication*, 17(8):1415–1425, August 1999.

[24] R. Castaneda and S. Das. Query Localization Techniques for On-demand Routing Protocols in Ad Hoc Networks. In *Proceedings of the 5th Annual International Conference on Mobile Computing and Networking (MobiCom 1999)*, pages 186 – 194, August 1999.

[25] G. Di Caro, F. Ducatelle, and L. M. Gambardella. AntHocNet: an ant-based hybrid routing algorithm for mobile ad hoc networks. In *Proceedings of PPSN VIII - Eight International Conference on Parallel Problem Solving from Nature*, number 3242 in Lecture Notes in Computer Science. Springer-Verlag, September 2004.

[26] Wolfram Research, Inc.. Mathematica. http://www.wolfram.com/.

[27] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

[28] A. Valera, W. Seah, and S. V. Rao. Cooperative Packet Caching and Shortest Multipath Routing In Mobile Ad hoc Networks. In *Proceedings of the IEEE INFOCOM 2003*, volume 1, pages 260 – 269, April 2003.

[29] Z. J. Haas, J. Y. Halpern, and E. L. Li. Gossip-Based Ad Hoc Routing. In *Proceedings of the IEEE INFOCOM 2002*, volume 3, pages 1707 – 1716, June 2002.

[30] G. Brassard and P. Bratley, editors. *Fundamentals of Algorithmics*. Prentice Hall, 1996.

[31] Sanjit Biswas and Robert Morris. Opportunistic routing in multi-hop wireless networks. *ACM SIGCOMM Computer Communication Review*, 34(1):69–74, January 2004.

[32] S. Capkun, M. Hamdi, and J.-P. Hubaux. GPS-Free Positioning in Mobile ad-hoc Networks. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, January 2001.

[33] A. Rao, Ch. Papadimitriou, S. Shenker, and I. Stoica. Location information: Geographic routing without location information. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom 2003)*, pages 96–108, September 2003.

[34] Xiang-Yang Li, Kousha Moaveninejad, and Ophir Frieder. Regional gossip routing for wireless ad hoc networks. *Mobile Networks and Applications*, 10(1-2):61–77, 2005.

[35] E. W. Weisstein. Circle-Circle Intersection. From MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/Circle-CircleIntersection.html.