Technique Report TR08-05

Huilong Huang

Department of Computer Science

University of Arizona

8/18/2008

EFFICIENT ROUTING IN WIRELESS AD HOC NETWORKS

by

Huilong Huang

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF COMPUTER SCIENCE

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2 0 0 8

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Final Examination Committee, we certify that we have read the dissertation prepared by Huilong Huang
entitled Efficient Routing in Wireless Ad Hoc Networks
and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Date: 4 August 2008

John H. Hartman

Date: 4 August 2008

Stephen Kobourov

Date: 4 August 2008

Beichuan Zhang

Date: 4 August 2008

Marwan Krunz

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Date: 4 August 2008

Dissertation Director: John H. Hartman

STATEMENT BY AUTHOR

SIGNED: Huilong Huang

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. John H. Hartman, for his guidance during the last 4 years in my PhD study. He helped me generate ideas, develop research projects and write papers. I learned how to do research and write academic papers under his guidance. I would never forget the days we were discussing research ideas, project progress and results, working together to revise and submit papers. I am especially grateful for his great effort in reviewing and revising my dissertation. Without his encouragement and help, my dissertation could never be finished.

I would also thank the committee members, Dr. Stephen Kobourov, Dr. Beichuan Zhang and Dr. Marwan Krunz, for their help in my PhD study and research projects. I got important ideas and feedbacks from the discussions with them. I would also thank them for their effort in reviewing my dissertation and providing comments.

Many thanks to Dr. Richard Snodgrass. I learned a lot on research project management and academic writing during the time I worked as his RA.

Special thanks to Dr. Stephen Pink. He guided me during the first stage of my PhD study and leaded me into the research on computer networking.

I would also like to thank Dr. Terril Hurst. It was a nice experience working with him.

Many of my officemates helped me during my PhD study. I benefited a lot from the discussions with them on research ideas and technical problems. Many thanks to Haitao Liu, Haifeng He, Somasundaram Perianayagam, Justin Cappos, Quanfu Fan, Jesus Arango, Man Zhang.

I got a lot help from the department staff people, including Rhonda Leiva, John Cropper and many other people. I would like to thank them.

Finally I want to thank my wife Wei and my parents for their unconditional support during these years.

DEDICATION

Dedicated to my family, including my father Shuilai Huang, my mother Chaqing Deng, my wife Wei Yuan, our coming baby, my sister Lin Huang and brother-in-law Rongbin Wang.

In memory of my grandmother, Heli He.

TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

TABLE OF CONTENTS – *Continued*

LIST OF FIGURES

LIST OF FIGURES – *Continued*

LIST OF TABLES

ABSTRACT

Routing is the fundamental problem for Wireless Ad hoc networks, including Wireless Mobile Ad hoc networks (MANETs) and Wireless Sensor networks (WSNs). Although the problem has been extensively studied in the past decade, the existing solutions have deficiencies in one or more aspects including efficiency, scalability, robustness, complexity, etc.

This dissertation proposes several new solutions for routing in WSNs and MANETs. *Spiral* is a data–centric routing algorithm for short–term communication in unstructured static WSNs. Spiral is a biased walk that visits nodes near the source before more distant nodes. This results in a spiral–like search path that is not only more likely to find a closer copy of the desired data than random walk, but is also able to compute a shorter route because the network around the source is more thoroughly explored. Compared with existing flooding and random walk approaches, Spiral has a lower search cost than flooding and returns better routes than random walk.

Closest Neighbor First Search (CNFS) is a query processing algorithm for mobile wireless sensor networks. It is also walk-based and biased to visit nodes close to the source first. Different from Spiral, CNFS collects topology information as the search progresses. The topology information is used to compute the shortest return path for the query result and to tolerate the network topology changes caused by node mobility, which could otherwise cause the query to fail. CNFS requires fewer messages to process a query than flooding–based algorithms, while tolerating node mobility better than random walk–based algorithms.

Address Aggregation-based Routing (AAR) is a novel routing protocol designed for MANETs. It reactively performs route discovery, but proactively maintains an index hierarchy called a Route Discovery DAG (RDD) to make route discovery ef-

ficient. The RDD contains aggregated node address information, requiring fewer packets for route discovery than the flooding used in existing protocols, while handling mobility better than pre-computing routes to all nodes. Compared with some existing popular protocols, AAR shows better performance in delivery rate, message overhead, latency and scalability.

CHAPTER 1

INTRODUCTION

1.1 Wireless Ad Hoc Networks and Sensor Networks

Mobile Ad-hoc Networks (MANETs) are networks of mobile computers ("nodes") connected by wireless links. The network does not have a fixed infrastructure and each node moves in a physical area at various speeds. Each node serves as both a communication end-point and a router, so that communication between two nodes may require sending the packets through a sequence of intermediate nodes. MANETs are widely used in emergency and disaster rescue, vehicle communication, military systems, personal area networks, and so on.

Wireless Sensor Networks (WSNs) are networks consisting of numerous smart sensors. The sensors may be static or mobile. Each sensor node has limited processing power & communicates with each others through wireless radio. Sensor networks are widely used in commercial and industrial applications, such as monitoring manufacturing lines; they are also used in military systems as well as environmental monitoring systems.

There are many issues to consider when designing MANETs and WSNs, including routing, security, power consumption, and quality of service. Among these issues, routing is the most fundamental yet challenging problem for both MANETs and WSNs. This dissertation addresses several important routing problems in MANETs and WSNs. It proposes and implements several novel routing algorithms/protocols that outperform the existing ones in one or more performance metrics, including communication overhead, latency, energy consumption, robustness and scalability to larger networks.

1.2 Routing in Wireless Ad Hoc Networks and Sensor Networks

Routing is an important issue in both Wireless Mobile Ad Hoc Networks and Wireless Sensor Networks. Both networks do not have fixed infrastructure and the routing requires distributed and cooperative actions from all nodes in the networks. Both networks have limited power supply and bandwidth constrains, thus power and bandwidth efficiency are important issues in designing proper routing algorithms or protocols. Mobile Ad Hoc Network routing and Wireless Sensor Network routing share some common features, but also have significant differences.

1.2.1 Routing in MANETs

Mobile Ad Hoc Networks provide point-to-point routing similar to Internet routing. Routing is based on addresses that are unique in the network. The source node specifies the destination address, and the network routing service returns a route that contains multiple intermediate nodes between the source and the destination. Packets are routed through the intermediate nodes and every node forwards the packets according to the destination address. When receiving packets, the destination node further demultiplexes the packets to the appropriate applications.

The major difference between MANET routing and regular Internet routing is the route discovery mechanism. Internet routing protocols, such as RIP [32] and OSPF [54], have relatively long converge times, which is acceptable for a wired network that has infrequent topology changes. However, a MANET has rapid topology changes due to node mobility, thus the traditional Internet routing protocols are not appropriate; as a result, many MANET-specific routing protocols have been proposed. The MANET routing protocols handle topology changes well, however, most of them have large control overhead and are not scalable to large networks.

Another difference between MANET routing and Internet routing is the network address. In Internet routing, the network address (IP address) is hierarchical, containing a network ID and a computer ID on that network. The routers use the destination network ID to decide the next hop. Only the router on the destination network uses the computer ID to deliver the packet to the destination computer. In contrast, in most MANETs the network address is simply an ID of the node in the network and is not hierarchical. The routing protocol must use the entire address to decide the next hop.

During the past decade, numerous routing protocols for MANETs have been proposed, including DSDV [60], AODV [61], DSR [43], TORA [57], ZRP [31], HSR [40], CGSR [18], CBRP [42], LAR [45], GPSR [44], OLSR [21], and SHARP [63]. These protocols discover routes either proactively or reactively. Proactive route discovery has high overhead and cannot maintain accurate routes in a highly mobile network. On the other hand, reactive route discovery usually relies on either flooding or location information. Flooding has high communication cost, can congest the network, and does not scale to large networks. Location information requires special equipment and is not generally available.

1.2.2 Routing in WSNs

Routing in Wireless Sensor Networks is usually application specific. Similar to MANET routing, the network address in Wireless Sensor Networks is just an ID for the sensor node and is not hierarchical. Unlike MANET, in WSN the routing is often tightly integrated with the sensor network application. Messages are often routed based on the data, rather than the network addresses. Nodes search the network for particular data, for example, an update of the driver code, the stored sensor data, etc. A sensor node (or base station) may also inject a query into the network and wait for a reply.

The data content oriented routing scheme in WSN is called *Data Centric Routing* [38, 48, 65]. In this routing scheme, instead of specifying the destination address, the source node specifies the desired data, regardless of where it is located. The routing layer should locate the desired data object and return a valid route to the node that has the data object.

Due to the large data redundancy in sensor networks, data–centric routing has proven to be a good scheme for minimizing communication overhead and energy consumption. Data–centric routing normally has two phases: *route discovery* and *communication*. The route discovery phase determines the best route between the source node and a node with the desired data. During the communication phase the data are transmitted from the destination to the source. The relative lengths of these two phases influences the choice of routing algorithm. At the two extremes, either the communication is long–lived, or it consists of a single response to the query (*one–shot*). For the former the route discovery overhead is less important, as it will be amortized over the long communication [38]. What is important is the resulting route length, as the subsequent communication must traverse the route and a sub–optimal route will have high overhead. Many routing protocols, such as TAG [52] and Directed Diffusion [38], use flooding–based techniques to discover close-to-optimal routes for the long–lived communication. For one–shot communication the route discovery cost is much more important than the route length; it may not pay to find a short route. Some query processing protocols, such as ACQUIRE [65], Rumor Routing [12], rely on walk–based techniques to reduce the cost of query processing and route discovery.

1.3 Contributions

This dissertation proposes three novel protocols for routing in MANETs and WSNs. The *Spiral* protocol is designed for short–term data centric routing in wireless sensor networks; the *CNFS* protocol is designed for query processing in mobile sensor networks; and the *AAR* protocol is designed for routing in Mobile Ad hoc Networks. Extensive experiments have been done to evaluate the new protocols. The results show the new routing protocols outperform existing ones in either message overhead, delivery rate or end-to-end delay.

1.3.1 Spiral Protocol for Data Centric Routing

This dissertation presents a data–centric routing protocol called *Spiral* for *data–centric* routing [38, 48, 65]. In Data Centric routing, the length of data communication can be either long–term, short–term, or one–shot. Spiral is intended for short–term communication in wireless sensor networks, in which both the route discovery cost and resulting route length are important. For example, in data queries for historic data [47], the collected data may be much bigger than a single packet, and hence may need to be encapsulated in multiple packets for transmission. Another example is short–term monitoring, in which the observer searches for an interesting sensor and then monitors it for a short time. These data transfers are likely to require multiple messages, but will not be long–term.

Spiral balances the cost of route discovery and route length, making it efficient for short–term communication of tens or hundreds of messages. This reduces the overall communication overhead, and hence the energy consumption, for short–term data transmission in wireless sensor networks.

Spiral reduces the communication overhead and the energy consumption for both route discovery and subsequent communication. Like the route discovery protocols for one–shot communication, Spiral uses a walk–based mechanism to reduce the overhead of searching for the desired data; Spiral uses a spiral–like search path that visits nodes near the source before more distant nodes. This is particularly beneficial when there are multiple copies of the desired data, as Spiral tends to find a close one first. Our experimental results show that for route discovery in a 500-node network with average node degree of 20 and two copies of every data object, Spiral's message overhead is only 8% higher than the random walk–based protocols, but only 46% of that in the flood–based protocols. Spiral's route lengths are only 10% longer than the optimal routes (from the flood–based protocols), but only about 66% as long as those produced by the random walk–based protocols. Overall, Spiral is most efficient for short–term communication of tens to hundreds of messages in dense networks. For example, Spiral's total cost for a short–term communication of 40 packets is only 72% of that by flooding, 81% of ERS, 74% of random walk, and 73% of DFS.

1.3.2 CNFS Protocol for Query Processing

This dissertation also proposes an efficient and robust protocol called *Closest Neighbor First Search (CNFS)* for query processing in Mobile Wireless Sensor Networks. Two of the most important metrics when evaluating query processing schemes are efficiency and robustness. The efficiency is measured as the number of messages and the latency required to satisfy the query. Minimizing the number of messages is important because sensor nodes normally have very limited power and message transmissions dominate a node's power consumption. Excessive messages can also increase network congestion. Many application scenarios also have requirements on query latency. Robustness is important because queries may fail due to mobility and unreliable communication. An acceptable query processing scheme should maximize the success rate despite network dynamics. Compared to algorithms based on flooding or random walk, our algorithm requires fewer message transmissions and is robust against network dynamics.

Conventional query processing algorithms use blind search to satisfy the query – that is, no global information about the network is gathered prior to initiating the search. Such information might include a hierarchy of nodes or an index of the data they contain. This information would obviously be useful, but is impractical to maintain in the face of mobile nodes that cause the network topology to change rapidly. The potentially high data generation rate from sensors in a sensor network also makes the maintenance of the indexing mechanism costly. For these reasons query processing algorithms typically employ either flooding or random walk. Flooding floods the network with the query to find the answer. It returns the answer very quickly, and is therefore highly-tolerant of changing network dynamics, but it requires an excessive number of messages and can congest the network. In contrast, a random walk uses a single message that randomly visits the network nodes. This reduces the message overhead and avoids congestion, but can be very slow and may not complete at all if the network topology changes so that the source or destination cannot be visited. For this reason it has a much lower success rate than flooding.

CNFS is based on a directed blind search — no pre-existing structure is imposed on the nodes or the data they contain. The search is directed by topology information collected as it progresses, allowing CNFS to be both efficient and robust. CNFS collects and maintains a *partial map* of the network during query processing, from which it finds an optimal order of visiting nodes. The algorithm uses a biased search scheme, i.e., whenever there is a choice of unvisited nodes to visit next, it chooses the one closest to the source node. The partial map is also used to compute the shortest return to the source, as well as determine alternative routes when links break, helping to improve both efficiency and robustness. For example, the experimental results show that for query processing in a 100-node network with average node degree of 8.3, CNFS requires about 38% fewer messages compared with the other methods. In addition, its success rate is significantly higher than random walk-based algorithms for all network densities (up to 37% higher), and comparable to flooding in dense networks (1% lower in a 100 nodes network with maximum node speed at 10m/s and average node degree of 16.6).

1.3.3 AAR for MANET Routing

This dissertation proposes a novel routing protocol called *Address Aggregation-based Routing (AAR)* for MANET Routing. AAR differs from existing protocols in that it performs route discovery reactively, but it proactively maintains an index hierarchy that makes route discovery more efficient while avoiding flooding. The index hierarchy dramatically reduces route discovery cost, leading to lower packet overhead and better support for large networks. In addition, AAR has a lower loss rate and end-to-end delay for data transmission, due to its dynamic route repair and shortening mechanism.

The index hierarchy in AAR is a *Route Discovery DAG (RDD)*. The RDD is a structured DAG (Directed Acyclic Graph) whose edges are physical links in the underlying network. The RDD is structured so that every node knows its distance (in hops) from a chosen *root* node, and every node has directed edges from itself to its neighbors that are closer to the root. Address information is aggregated along the edges of the RDD, so that the root node has information about every node in the network, while the *leaf* nodes (those who have no neighbors farther from the root) have information only about themselves. The address information is aggregated using Bloom filters [11], allowing aggregated address information to fit into a single network packet even for a network with one thousand nodes.

Route discovery is performed by searching the RDD starting at the source node and following the edges towards the root until a match for the destination address is found in the current node's Bloom filter. The search then proceeds along the RDD edges in reverse, visiting those nodes whose Bloom filters match the destination address until the destination is reached. Restricting the search to move towards the root and then away from it prevents routing loops. AAR makes use of several optimizations to "short-circuit" the RDD during route discovery by having nodes keep track of the Bloom filters of their siblings (neighbors at the same distance from the root), as well as a 2-hop map of their neighbors and their neighbor's neighbors. This helps offload the nodes at the center of the RDD. AAR also uses the 2-hop map to repair routes that break due to node mobility or failures, and to shorten routes once they are discovered.

Compared with existing routing protocols such as AODV [61] and DSR [43], AAR requires lower communication cost, loses fewer data packets, has a smaller end-to-end delay, and scales to larger networks. Simulation results show that for a network with 400 nodes, 80 CBR flows, randomly-distributed nodes, a density of 15 nodes per transmission range, and a maximum speed of 10 m/s, the packet overhead of AAR is 37% of AODV and 40% of DSR; its data packet loss rate is 16% of AODV and 19% of DSR; and its end-to-end delay is 64% of AODV and 50% of DSR.

1.3.4 Dissertation Organization

The rest of the dissertation is organized as follows: Chapter 2 contains related work; Chapter 3 presents the Spiral algorithm and its experiment results; Chapter 4 presents the CNFS algorithm and the experiment results; Chapter 5 describes the design of AAR protocol and presents the performance comparison results with existing protocols; Chapter 6 concludes the contribution of this dissertation and describes the future work.

CHAPTER 2

RELATED WORK

2.1 Routing in Sensor Networks

Routing in sensor networks is usually data–centric and application– specific, and often takes the form of data queries. According to Sadagopan et al. [65], there are many ways to categorize the queries: Continuous versus one-shot; aggregated versus non-aggregated, complex versus simple, and replicated data versus unique data. In addition to the various different query types, the sensor networks may also have unique features or requirements. For example, some sensor networks use a base station for centralized control and mainly consider queries from the base station; while other networks run in an autonomous way and the queries may originate from any node. Some sensor networks have location service or the nodes know their geographic location; while the others do not have such an assumption.

Due to the various application scenarios and network features, there are numerous data centric routing protocols proposed. Their representives are categorized and summarized as follows.

2.1.1 Flooding

In flood–based protocols the data query is flooded into the entire network, perhaps with a limit on the flood scope. During the flooding, routing trees or gradient paths to the source node are generated. The sensor nodes with the desired data transmit the data back along the routes discovered, possibly aggregating the data in–network. Some well–known protocols, such as TAG [52] and Directed Diffusion [38], are of this type.

In Directed Diffusion [39], flooding propagates the query throughout the entire network, allowing every node to learn its distance from the query source. Gradient paths toward the query source are formed based on the nodes' distances. When the destination nodes receive the request, they send responses back to the query source along the shortest path.

TAG uses routing mechanism similar to Directed Diffusion, except that it primarily considers queries from a fixed root node of the network, which is either the base station or the sensor node that the user accesses. The root node broadcasts a routing message and organizes the network in a tree. Every node takes a node closer to the root as its parent. The queries have two phases: a distribution phase and a collection phase. In the distribution phase the queries are pushed into the network using flooding. In the collection phase the query results are continually aggregated and sent from nodes to the root through the tree.

There are numerous variations of flooding in ad hoc routing protocols. Perkins et al. [61] used Expanding Ring Search (ERS) as a better alternative to simple flooding. ERS is a sequence of floods that have expanding scope; the intent is to avoid flooding the entire network on each route discovery. There are many ways of controlling the flooding scope, the most common one is achieved by setting the TTL field in the IP header.

In Perkins et al. [61], the ERS mechanism increases the flooding scope linearly starting from a small value. The source node waits for a timeout equivalent to the estimated round trip time for that scope. If no response is received, the source node starts another round of flooding with scope increased by a fixed value (thus the linear increment of the flooding scope). This process continues until either the desired response is received, or the flooding scope exceeds a predefined threshold. For the latter case, the subsequent flooding has no limit on the scope and will reach the whole network.

Sadagopan et al. [65] and Cheng et al. [74] further studied the efficiency of ERS for data centric routing in wireless sensor networks. Sadagopan et al. studied the ERS scheme with linear growth and show the ERS scheme has lower energy consumption than basic flooding. Cheng et al. studied the ERS scheme with exponential growth of the flooding scope and showed that this scheme works reasonably well.

Flooding is a fast way to process queries. The quick response makes flooding robust against network dynamics. The network is likely to remain static during the short lifetime of a flooding search, thus the gradient paths formed during the flooding do not break and the query response is successfully returned.

2.1.2 Clusters and Backbones

One way to reduce the route discovery cost is to limit the search to a subset of the network nodes. This can be accomplished by *clusters* or *backbones*. A cluster is a collection of nodes whose collective contents are known by a designated *cluster head* node. Route discovery only requires searching the cluster heads. Similarly, a network backbone is a connected subset of nodes such that every node in the network is either in the backbone or the neighbor of a node in the backbone. Again, route discovery only requires searching all the backbone nodes, because the query messages from the backbone nodes will be heard by all the nodes in the network.

Clusters and backbones are typically generated by Dominating Set or Connected Dominating Set (CDS) protocols. In our research, so as to not underestimate the performance of flood–based protocols, we assume that a distributed CDS protocol, such as the one proposed by Wu et al. [72], is used to create a backbone on which route discovery is performed. The non-backbone nodes overhear the transmissions on the backbone and respond to the query source if they have the desired data.

Protocols based on either clusters or backbones include LEACH [33], HEED [73], SPAN [17], and Virtual Backbone [71].

LEACH [33] organizes the sensor network nodes in local clusters with one node elected as the local cluster head. The cluster head is randomly chosen and rotated to avoid draining power of a particular sensor node. The nodes in a cluster do not directly communicate with the base station or other clusters; instead, they send data to the local cluster head, which aggregates and compresses the data and communicates with the base station or other clusters.

HEED [73] works in a way similar to LEACH for node clustering, except that

HEED takes the residual energy and some other parameters for calculation of the clustering, so that the cluster heads are well–distributed and load-balanced.

SPAN [17] is designed for saving power in wireless ad hoc networks. In SPAN, each node makes periodical decision on whether it should sleep or stay awake and be part of the backbone to forward packets. The nodes make the decision based on the remaining battery energy and the neighborhood connectivity.

The Virtual Backbone [71] uses a distributed algorithm to compute a Connected Dominating Set in an ad hoc network. The Connected Dominating Set (CDS) is a set of nodes in which the set of nodes is connected, and every node in the network is either in the set or a neighbor of a node in the set. The Connected Dominating Set helps to reduce the query or search space; for example, if the nodes in the CDS are also cluster headers then only the CDS nodes need to be visited. CDS is especially useful in reducing the flooding scope. Due to the broadcast nature of wireless communication, the broadcasts from the nodes in CDS will cover every node in the network. Flooding search on the whole network can be reduced to flooding the CDS only. The CDS algorithm in Virtual Backbone [71] protocol computes the CDS in a distributed and localized way, with only O(1) message cost per node.

2.1.3 Walk–based protocols

Flooding–based routing protocols are quick to find a desired object or get a query response. However, flooding has high communication overhead because every node in the search space must be visited and must forward the message. There is no control for the flooded messages, even if the response has been sent back to the source, the query messages on other directions are still flooded to the rest of network. The high communication overhead not only causes high energy consumption, but is also unscalable to large networks and easily causes congestion. Flooding is especially wasteful in communication overhead when the network has large data redundancy.

For these reasons, walk-based routing protocols have been developed. Instead of having every node forward the routing message, walk-based routing has only one message “walking” the network, searching for the destination object or collecting answers for a query. The message is often called the "walker". Once the destination object is found, the walker returns back immediately to the source node with either a route to the destination object or the answer to the query.

There are many ways to direct the walk-based search, including random walk and search-based walk. Protocols based on random walk include ACQUIRE [65], Rumor Routing [12], and Biased Random Walk [4].

ACQUIRE uses a d-hop look-ahead mechanism in which each node stores data from all neighbors within d hops. This allows the random walk to take fewer steps to find the data; however, it does not change the nature of the random walk, it merely reduces the effective size of the network by a constant factor d.

Rumor Routing [12] proposes that both the query and the event (i.e. the desired data) do random walks and leave traces on the nodes traveled; when the two random walks intersect, the route is determined according to the traces and reported to the source node.

Avin et al.[4] propose a biased random walk in which a node gives higher priority to unvisited neighbors when choosing the node to visit next. The bias significantly reduces the number of search steps.

Though simple, a random walk can not avoid revisiting nodes, which cause unnecessary delay and higher message overhead. Various search–based walk approaches, including Depth First Search (DFS) and Space–filling Curve–based Routing (SCR), have also been considered.

Depth First Search (DFS) is a simple protocol used in both peer–to–peer systems [20] and in wireless ad hoc routing [68]. Unlike a normal random walk that does not backtrack, DFS backtracks to find an unvisited node whenever the walker comes to a dead end. As long as the desired data exist and the network is not partitioned, DFS can guarantee that the search will locate the data in fewer steps than twice the network size.

Space-filling Curve-based Routing(SCR) was recently proposed for routing in wireless sensor network [58]. In SCR, the routing message travels through the network along a *space-filling curve*, which is an imaginary curve that runs through all nodes in the network. SCR assumes every node knows its own and its neighbors physical location, so that the space-filling curve can be dynamically computed according to the locations. Various space-filling curves can be used, including Sierpinski, sweep, and spiral curves.

Compared with the flooding approach, most of the walk–based protocols reduce the message overhead, but at the cost of much longer search time. They work well for static networks, but have robustness problems for network dynamics.

2.1.4 Push and Pull approaches

The work discussed above mainly considers how to discover the destination data object through exhaustive search in the network, assuming the data objects reside only in certain nodes. This is known as a “pull” approach. A pull–based protocol needs to search most of the network if there are few nodes satisfying the query or having the desired data object, and the distance between source and destination nodes is large.

Some recent research proposes a push–and–pull scheme. The idea is to let the sensor nodes actively “push” their data or metadata into the network, either by generating large amounts of redundant data or forming a hierarchy/index to aid the query processing. Either alternative reduces the messages required to satisfy a query (the “pull”). Algorithms employing this scheme include Comb-Needle Search [50] and Stalk [25].

The Comb-Needle Search [50] assumes a grid-like network. It lets the sensor node push their data “vertically” in the area (hence a “needle”). The query node initially sends the query message “vertically” and at some interval the query message is also broadcasted “horizontally” (hence a “comb”). The Comb-Needle search can locate the desired data object in O($\sqrt{n}$) time and message transmissions, as opposed to O(n) time in pull-only methods.

The Stalk [25] protocol provides a hierarchical tracking service for mobile objects in the sensor network. Stalk maintains the tracking information with accuracy related to the distance to the mobile objects. The closer sensor nodes have more recent and accurate information about the object; while the farther nodes have older and inaccurate information about the object. Both the sensed information about the moving object and the query about the moving object is propagated through a hierarchy based on clustering, so that the cost of both updating and querying for the sensed information is kept low.

Although a push–and–pull technique can reduce the number of messages during query processing, it does so at the expense of increased number of messages needed to push the data. When the data generation rate or node mobility is high and the query rate is low, the gain from reducing the cost of the pull may be outweighed by the cost of the push.

2.1.5 Location Aided approach

Another group of sensor network routing protocols relies on the location service to search data objects or handle queries. A representative protocol is Geographic Hashing Table (GHT) [64].

GHT [64] employs a data dissemination scheme called "data-centric storage". GHT generates a key for each data object. It hashes the keys into geographic coordinates and stores the key and data object on the nodes closest to the coordinate. The keys and data objects are usually replicated in case the sensor nodes fail. Search for the desired data object is similar. The key is hashed using the same hash function to get the geographic coordinate. The routing or query message is sent to the destination nodes at that coordinate through geographic routing.

The Comb-Needle Search [50] described in last subsection also relies on the location service. It assumes that every node knows its geographic location and knows how to propagate messages "vertically" or "horizontally".

Using a location service to aid data centric routing or query processing does improve the routing efficiency. However, these protocols usually assume that nodes know their geographic locations, which is not true for many networks. In addition, they actively push data or build indices in the network, which incurs additional cost and may cause consistency problems.

2.2 Routing in Mobile Ad Hoc Networks (MANETs)

Compared to Sensor Network routing, routing in MANETs is closer to traditional Internet routing but still has significant differences. It is similar to traditional Internet routing in that MANET routing is point-to-point and independent of the application. The routers in a MANET simply forward the data packet without checking its content. However, unlike traditional Internet where computers and routers are separate and the network topology and hierarchy are static, in a MANET every computer is also a router and there is no static network topology or infrastructure. Although Internet routing protocols also handle network topology changes, the changes are infrequent. In a MANET the topology changes are so frequent that the traditional routing protocols may not even work.

One of the biggest challenges in MANET routing is to design a proper routing protocol that handles node mobility and rapid topology change. During the past decade, numerous MANET routing protocols have been proposed, which fall into two broad classes: proactive and reactive. Proactive protocols compute routes between all nodes in the network in advance, whereas reactive protocols compute routes only when they are needed for communication. Routing protocols that rely on the location service are called Location Aided Protocols. Most early MANET routing protocols did not maintain a hierarchy in the network. As those protocols do not scale to a large network, various hierarchical routing protocols were proposed to solve the problem. The different types of MANET routing protocols are discussed in the following subsections.

2.2.1 Proactive Protocols

Traditional Internet routing protocols are proactive – the routing table is proactively built and maintained. Whenever there are data packets to send, the route has already been determined. There are two major types of Internet routing protocols, link–state routing such as Routing Information Protocol (RIP) [32], and distance vector routing such as Open Shortest Path First (OSPF) [54]. In link–state routing, the nodes broadcast information about their neighboring links to the whole network, so that every node in the network has global knowledge of the network links. Every node independently computes the shortest the path to every other node in the network and builds up a routing table that contains the next hop information for each destination. The route computation can use Dijkstra's single source shortest path algorithm [54].

Distance vector routing uses the Distributed Bellman-Ford (DBF) [9] algorithm for route calculation. In distance vector routing, the nodes exchange the distance information instead of the link state. Originally a node only knows its local neighbors and the distance to the neighbors. Every node periodically announces its distances to other nodes, so that the other nodes knows which nodes are reachable through it as well as the distance. During the process, every node knows more non-local nodes and shorter distances to those nodes. The process continues until finally it converges and stabilizes.

Many routing protocols for MANET also adopted either distance vector routing schemes such as Dynamic Destination-Sequenced Distance-Vector(DSDV) routing [60] and Wireless Routing Protocol (WRP) [55], or Optimal Link State Routing (OLSR) [21].

DSDV [60] is derived from the traditional distance vector routing protocols for Internet. The improvement made by DSDV is the way every node exchanges routing information. To handle the frequent topology changes in a MANET, DSDV requires that the nodes advertise their routing tables both periodically and when triggered. In order to reduce the communication overhead, two types of update messages are used. One gives the "full dump" of the entire routing table, while the other "incremental" one just gives the changed information from last full dump. The full dump messages are less frequent than the incremental messages.

WRP [55] is another distance vector routing protocol for MANETs, developed about the same time as DSDV. WRP uses information about the second-to-last hop (so called "predecessor") to avoid temporary routing loops. However, the routing table data structure is much more complicated and larger than in that of DSDV.

OLSR [21] is link–state routing protocol optimized for MANET routing. Link state routing requires the link state information be propagated (usually through flooding) to the whole network. In order to reduce the communication cost of flooding, OLSR implements a mechanism to select a group of "multipoint distribution relays" (MPRs). In flooding, only the MPR nodes are required to forward the messages. The MPR nodes are selected so that the message transmissions from them cover the whole network.

VRR (Virtual Ring Routing) [16] is one protocol recently proposed for scalable routing in wireless networks. VRR builds a Distributed Hash Table (DHT) for the addresses over the mobile network nodes. Routing in VRR follows the DHT indices.

With proactive routing, data communication is not delayed by route discovery; however, the routes may be stale and which wouldresult in poor data delivery. In addition, the proactive route computation may have high overhead.

2.2.2 Reactive Protocols

Reactive protocols compute routes only when they are needed for communication. Reactive routing protocols have longer delays to determine routes, however they provide more accurate routes. Reactive routing also has less overhead when network traffic is light because the route discovery happens only on demand, while proactive routing continually computes and refreshes the routes for the entire network.

Due to the mobility of MANETs, many routing protocols are reactive, such as Ad hoc On-Demand Distance Vector Routing (AODV) [61], Dynamic Source Routing (DSR) [43] and Temporally-Ordered Routing Algorithm (TORA) [57]. AODV [61] and DSR [43] are the two most popular reactive routing protocols, and both use flooding for route discovery. Their major difference is the way they maintain and use the routes.

AODV uses distance vector routing. In route discovery, the route request message is flooded in the network. During this process, each node sets up a route entry for the source node, containing the node from which it heard the message and the distance to the source. When the destination node hears the route discovery message, it sends route reply to the source node along the route to the source node. Each node forwarding the route reply sets up a route entry for the destination, also containing the next hop node and the distance.

Unlike AODV, DSR uses source routing. The benefit of source routing is that there is no need to maintain states in the intermediate nodes along the route. The source node learns and maintains the route, specifying the route in every data packet. The intermediate node simply forwards the packet along the specified route. DSR floods route requests in a way similar to AODV. However, instead of having all nodes along the route set up and maintain routing entries for forwarding packets, DSR has the route request packet carry the path the packet traveled. When the destination node receives the route request, it sends a route reply back to the source node along the reversed path from the route request packet. The route reply packet carries the path as well, thus the intermediate nodes can also learn the routes to other nodes and store them in their route cache. Although source routing increases the amount of routing information in the data packets, it provides opportunities for nodes to learn routes to more nodes and optimize current routes. DSR also proposes using "promiscuous" mode to learn routes from the neighborhood traffic.

TORA builds its routes using DAGs rooted at the destination nodes. TORA uses a query/reply process to establish the DAGs. When a source node wants to find a route to a destination node, it floods the query to the network. When the destination receives the query, it returns replies to the sending nodes. In this way a DAG rooted on the destination node is generated. Every node in the network is assigned a distance to the destination node. The source node simply forwards the packet to the neighbor node with the smallest distance. In this way, TORA actually provides a multipath solution for the route.

Reactive routing has the advantages of quick adaptation to topology changes, and low overhead when data communication load is low. However, the fact that most reactive routing protocols rely on flooding for route discovery limits its scalability. Flooding has high communication cost and affects every node in the network. For large networks and high traffic loads, flooding can easily congest the network.

2.2.3 Location Aided Protocols

Not all existing reactive routing protocols rely on flooding for route discovery. Some protocols, such as Location-Aided Routing (LAR) [45], Greedy Perimeter Stateless Routing (GPSR) [44] and Zone-Based Hierarchical Link State (ZHLS) [53], use node location information in route discovery. With the node location information, blind flooding can be replaced by directional and regional flooding or even directed walk-based search.

LAR assumes the source node has the knowledge about the destination node location at a previous time point t_0. When the source node wants to find a route to the destination at time t_1, it estimates the expected zone where the destination node resides on and then computes the request zone that the request will be flooded. In most case the request zone is much smaller than the whole network area. In the case that the knowledge of the destination position is absent, LAR turns to regular flooding that floods the whole network.

GPSR also assumes the source node knows the destination location. Instead of flooding to find the route, GPSR uses greedy forwarding of data packets. The greedy decision is made on each hop based on the neighborhood location information. It chooses the neighbor geographically closest to the destination node as the next hop. If greedy forwarding is not possible, GPSR uses perimeter routing to bypass the non-connected region.

ZHLS uses location information to divide the network into non-overlapping zones. Each node has its Zone ID and network ID. The zone ID is computed from its location information. A two-level hierarchy is maintained in routing: routing within the zone is proactive, whereas routing between zones is reactive. A node knows the topology of its own zone and always has the routes ready. When a node wants to discover the route to a node in a different zone, it broadcasts a zone–level location request to all the other zones, which has less overhead than flooding the entire network.

GDSTR (Greedy Distributed Spanning Tree Routing) [49] is a protocol proposed as an enhancement to normal geographic routing. GDSTR maintains a spanning tree for aggregation of location and coverage information. It uses greedy forwarding for normal cases. When greedy forwarding fails, GDSTR resorts to the spanning tree for routing until it reaches a point that greedy forwarding works again.

Location aided routing has the benefits of simplicity and low overhead on route discovery. It is scalable to large networks. However, for MANETs, location information is normally obtained from special equipment such as GPS (Global Positioning System) which is not generally available, making these protocols are of limited utility.

2.2.4 Hierachical Protocols

In order to handle the scalability problem, the idea of imposing a hierarchy on MANETs have also been experimented. There are several routing protocols that use hierarchical routing, such as Zone Routing Protocol (ZRP) [31], Cluster-Head Gateway Switch Routing (CGSR) [18], Cluster Based Routing Protocol (CBRP) [42], Hierarchical State Routing (HSR) [40] and Sharp Hybrid Adaptive Routing Protocol (SHARP) [63].

ZRP is more of a framework than a specific routing protocol. It is a hybrid of proactive and reactive routing schemes. Through proactive routing, each node learns the route to other nodes within R hops (called the Zone Range). If the destination node is within the Zone Range, its address is always available. However if the destination is out of the Zone Range, ZRP uses flooding-based reactive routing for route discovery.

SHARP [63] is another zone-based routing protocol. Unlike ZRP, the zone radius is variable. Popular destinations have a larger zone radius, while the less popular nodes have smaller zone radius. The nodes proactively flood their zone to form a Directed Acyclic Graph that provides routes to the zone center. A node can directly send data to the zone center if it is within the zone. However, if the node is outside of the zone, it must use flooding to find the route.

CGSR [18] and CBRP [42] use clusters to divide the network into many small subnets. Each cluster has a cluster head that has routes to all nodes in the cluster. Routing within a cluster is done by having the cluster head forward the data to its destination. However, routing between nodes in different clusters uses flooding.

HSR [40] builds a hierarchy of clusters over the nodes. The cluster headers of lower levels are aggregated to form higher level clusters. A node maintains routes to other nodes in the same cluster. HSR routes packets using hierarchical addresses. The packet is forwarded upward in the hierarchy first to find the matching cluster of the destination, then downward to the specific destination node. The hierarchical address requires a mapping service to translate between the IP address and HSR hierarchical address. In addition, two nodes in the same cluster may be distant in the physical network, making it difficult and costly to maintain routes between them.

Current hierarchical routing protocols are either partially hierarchical, e.g. ZRP, CGSR, CBRP and SHARP, or have a full hierarchy that is hard to maintain and has high control overhead, e.g. HSR. For those partially hierarchical protocols, only the nodes in the same zone or cluster can easily find routes to each other without flooding; the nodes in different zones or clusters still use flooding to find the route. For those with a full hierarchy, the hierarchy causes not only high maintenance cost, but also a high failure rate for route discovery. A complete, simple, and robust hierarchical design is highly desirable for MANET routing.

CHAPTER 3

DATA CENTRIC ROUTING IN STATIC SENSOR NETWORKS

3.1 Introduction

Sensor networks are widely used in environment observation, habitat monitoring, health applications, battlefield surveillance, industrial manufacturing, taffic control and home/office applications [19, 34].

Early sensor networks were in relatively small scale (tens to hundreds nodes) and the sensors had low intelligence. The sensor network was normally attached to an external base station via gateway nodes [2]. The communication pattern was simply driving sensor data from sensors to the base station in one or multiple hops, hence forming a many-to-one communication pattern. Routing with this simple communication pattern may have a simple solution such as a spanning tree (generated through flooding) [38].

With technology advances, modern and future sensor networks will have smarter sensor nodes capable of more complex information processing; the network sizes also tend to be large (hundreds to thousands nodes) [3]. The sensor network may have multiple cluster head nodes for distributed computing; or the network may work in a peer to peer fashion without a centralized base station for data processing. One application example is a hospital in which all doctors and patients are part of the sensor network. A doctor may check one or multiple patients for their health information. The doctor may also check with other doctors for collaboration on diagnostics. The communication pattern here is a many-to-many pattern, which provides more design space as well as more challenges for efficiently routing in the sensor network. Many routing strategies have been proposed for this more complicated communication pattern and to cope with requirements for different applications. Due to the application-specific nature of sensor network routing and various application possibilities, there are still many routing scenarios that are not well explored and do not have good solutions.

This chapter presents a data–centric routing algorithm called "*Spiral*" for short–term communication in wireless sensor networks that work in a peer to peer fashion. Short–term communication is a common communication scenario for sensor networks. Take the hospital sensor network as an example, a doctor may want to check a patient's medical measurement record during the last week; or he may want to exchange diagnostic records with another doctor. These communications exchange considerable amount of data, but the transmission does not last forever. Spiral is intended for *data–centric* routing [38, 48, 65]. Unlike *address–centric* routing in which the source node searches for a route to the destination node, in data–centric routing the source searches for a particular data object stored on an unknown subset of nodes. Hence the routing problem is actually a query problem — the routing algorithm must search for the route to a node with the desired data, then the data can be transmitted along the discovered route.

Spiral balances the cost of route discovery and route length, making it efficient for short–term communication of tens or hundreds of messages. This reduces the overall communication overhead, and hence the energy consumption, for short–term data transmission in wireless sensor networks.

Due to the large data redundancy in sensor networks, data–centric routing has proven to be a good scheme for minimizing communication overhead and energy consumption. Data–centric routing normally has two phases: *route discovery* and *communication*. The route discovery phase determines the best route between the source node and a node with the desired data. During the communication phase the data are transmitted from the destination to the source. The relative lengths of these two phases influences the choice of routing algorithm. At the two extremes either the communication is long–lived, or it consists of a single response to the query (*one–shot*). For the former the route discovery overhead is less important, as it will be amortized over the long communication [38]. What is important is the resulting route length, as the subsequent communication must traverse the route and a sub–optimal route will have high overhead. For the one–shot communication the route discovery cost is much more important than the discovered route length; it may not pay to find a short route if the communication is brief [65, 4].

Spiral reduces the communication overhead and the energy consumption for both route discovery and subsequent communication. Like the route discovery algorithms for one–shot communication, Spiral uses a walk–based mechanism to reduce the overhead of searching for the desired data; Spiral uses a spiral–like search path that visits nodes near the source before more distant nodes. This is particularly beneficial when there are multiple copies of the desired data, as Spiral will tend to find a close one first. Our experimental results show that for route discovery in a 500-node network with average node degree of 20 and two copies of every data object, Spiral's message overhead is only 8% higher than the random walk–based algorithms, but only 46% of that in the flood–based algorithms. Spiral's route lengths are only 10% longer than the optimal routes (from the flood–based algorithms), but only about 66% as long as those produced by the random walk–based algorithms. Overall, Spiral is most efficient for short–term communication of tens to hundreds of messages in dense networks. For example, Spiral's total cost for a short–term communication of 40 packets is only 72% of that by flooding, 81% of ERS, 74% of random walk, and 73% of DFS.

3.2 Problem Description

3.2.1 Data–Centric Routing

We define the data–centric routing problem as follows. Consider a sensor network with N sensors and M copies of the desired data. The M copies are equivalent: the search stops when any one of them is found. Let X be the total number of transmissions to discover a copy.

Once located, the desired data must be transmitted back to the source node along a route discovered during the search. Let K be the size of the data in packets, and R be the length of the route in hops. The equation for the total transmissions

C is therefore $C = X + K \times R$. We use number of transmissions C as a rough indication of both communication cost and energy cost, as both are important to wireless sensor networks. The route discovery cost X depends on the network size N and the data redundancy M, as well as the search algorithm. The route length R also depends on these factors as well as the network density (represented as average node degree D).

The goal of the Spiral algorithm is to enable short–term communication with a smaller C than other data–centric routing algorithms, such as flooding, ERS, random walk, DFS, etc, for some ranges of K and D. The possible ways to reduce C are to decrease the route discovery cost X and/or decrease the route length R. Flood–based algorithms find the optimal route, and therefore the minimal R, but have a high X since every node must retransmit the message. On the other hand, random walk–based algorithms have a smaller X, but R is far from optimal.

Spiral finds a middle ground between these extremes. Its route discovery cost X is lower than flooding, and its route length R is shorter than random walk. Thus, for the short–term communication prevalent in sensor networks Spiral results in a smaller C.

3.2.2 Backbones

Clustering and building backbones helps to reduce the search space. They are especially beneficial to flooding searches. So as to not underestimate the performance of flood–based algorithms, we assume that backbone techniques are applied on the network. The search algorithms only explore the backbone; while the non-backbone nodes overhear the transmissions on the backbone and respond to the query source if they have the desired data.

Because the overhearing node may send back a response without notifying the walker, in our design, the walker must periodically visit the query source to check whether or not the desired data have already been found. The check interval is initially small and is exponentially increased to a maximum value. In our experiments, we found this method balances the check interval overhead and the overall search cost.

3.2.3 Network Characteristics

The Spiral algorithm assumes the following about the network on which it runs:

- The network is unstructured. No infrastructure or hierarchy pre-exists to aid the route discovery, and no push–and–pull mechanism is employed.
- The network topology is relatively static. Nodes may move, but not significantly during the duration of a search. Like DFS, Spiral builds a search tree in the network that records which nodes have already been visited. We assume a static or low–mobility sensor network, so that the network topology is unchanged during a search. Although this assumption is not true for all sensor networks, it is true for many [69].
- The network is well–connected. The sparser the network, the more difficult it is to search efficiently. In order for walk–based algorithms to work reasonably well, the networks must be dense (average node degree of 10 or higher). Fortunately, many sensor networks have this feature. Shih, et al [67] gave examples of networks with node density of 12 nodes/m^2 and a transmission range of 10 meters. Several studies have used networks with average node degrees as high as 40 [37, 36].
- The network links are symmetric. This assumption is true for all algorithms we studied.

3.3 Spiral Design

3.3.1 Overview

The key idea behind Spiral is to bias the walk so that nodes nearer to the source are visited before nodes farther away. This allows Spiral to discover a close copy of the desired data first, leading to a short route. An ideal case is shown in Figure 3.1.

Nodes are labeled with their distance to the source node S, which we refer to as their *level*. There are two nodes containing data that satisfy the query, labeled D, one in level 2 and the other in level 4. The walker travels in a spiral trajectory: it first visits nodes in level 1, then nodes in level 2, etc., until a node D is found. Due to the spiral trajectory, the closer node D (in level 2) is found and a route of length of 2 is discovered and returned to the source.

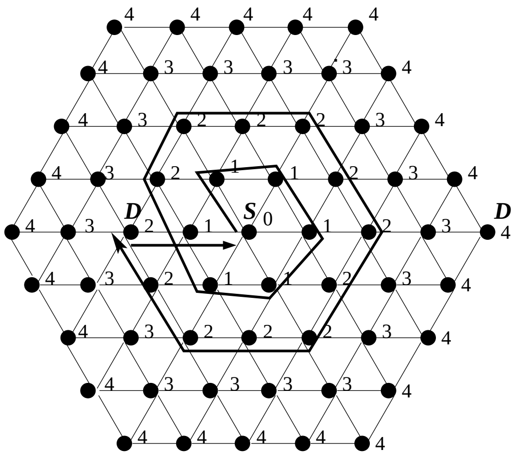


Figure 3.1: An ideal Spiral search. Nodes are labeled with their distance to the source node S.

Spiral uses the bias heuristic to simulate a breadth–first search similar to the flood–based algorithms — near nodes are visited before far nodes. This behavior has two effects on the discovered routes. First, close copies of the desired data are discovered before distant. Second, because the lower levels are well-explored when the desired data are found, a short route back to the source is easily found. DFS or random walk algorithms cannot guarantee that either of these is true. Our experiments show that they are likely to locate a node farther away and return sub–optimal routes.

Spiral is a biased DFS search with iterative deepening [46] on the levels. During the search every node estimates its distance and its neighbors' distance to the source via an auxiliary mechanism. Each time a node forwards the search message it chooses an unvisited neighbor with the smallest estimated distance as the recipient. Spiral uses iterative deepening to limit the distance the walker can travel from the source.

Hence all nodes closer to the source must be visited before nodes farther away. The iterative deepening mechanism is applied on the levels, not on the length of the current search path [46]; this prevents unnecessary backtracking if the current search path is long, but there are still unvisited nodes at the same level as the current node.

One algorithm related to Spiral is Closest First Exploration(CFX) [27] which is designed for constrained graph exploration. CFX does multiple bounded DFS Explorations (bDFX) which bound the exploration distance to a constant. For each bDFX, the root is selected as the unvisited node closest to the source. The CFX algorithm can also be used for searching in sensor network and provides the solution for finding close data object, however the bounded DFS search excessively revisits nodes, thus it is less efficient than Spiral, especially when the network is dense.

Similar to all other walk-based search schemes, Spiral faces the problem that the loss of the walker will end the search. Spiral relies on a time–out and retransmission scheme for handling such losses, as most other walk-based schemes do.

There are two important design problems for Spiral. First, how does a node properly estimate its neighbors' levels? Second, how does the walker know when all nodes in a level have been visited and it should move on to the next level? These problems are easily solved if the walker carries with it the topology of the explored network. However, this makes the walker size proportional to the network size. Although this might be feasible for small networks, it is clearly infeasible for large ones. We therefore focused on solutions in which the walker is fixed–size and does not carry the full network topology. These mechanisms are described in the following sections.

3.3.2 Level Estimation

In a Spiral walk, every node estimates both its own level and its neighbors' levels. Spiral uses a simple mechanism for doing this. Initially every node, except for the source, estimates that it and its neighbors are in level *Infinity*. The source node knows it is in level 0 and all its neighbors are in level 1. When a node forwards the walker, it includes its estimations of its own level and its neighbors' levels.

All neighbor nodes “eavesdrop” the message and dynamically adjust their level estimations. When node u at estimated level l overhears a message it performs the following actions:

- If the message contains estimated level l' for u, u sets $l = min(l, l')$.
- Node u sets its level estimate k for a neighbor node v to $k = min(k, l + 1)$.
- If the message contains estimated level k' for neighbor node v, u sets $k = min(k, k')$.

Every node has level estimates for its neighbors, and can therefore keep track of its neighbor(s) with the lowest level. This information allows *gradient paths* to be formed from any node back to the source such that there is only one node from each level in a path. The gradient paths are used by the walker to return to the source either to return the desired data or to check the status of the search. DFS and random walk generate gradient paths similarly, but because they search the network randomly their gradient paths are much longer.

3.3.3 Level Window

Spiral limits how far the walker can travel from the source so as to ensure that lower levels are searched before higher. The obvious thing to do is search the levels one at a time, but this leads to very high search costs in sparse networks. Instead, Spiral uses a sliding window mechanism that allows the walker to visit any node whose level is within the window. The window is defined on levels: if the lowest level with unvisited nodes is x and the window size is W, then the search may only visit nodes with estimated levels in the range $[x, x + W)$. When choosing the next node to visit Spiral chooses the neighbor with the lowest estimated level that is inside the current window. If no neighbor is within the window then the walker backtracks and searches the visited nodes to find one with unvisited neighbors within the window. When all nodes in level x are visited, the new value of x is computed and the window adjusted to the new bounds.

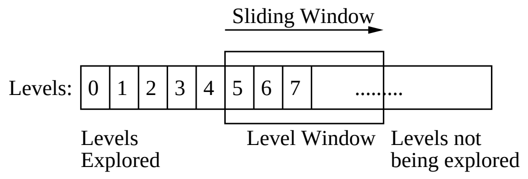


Figure 3.2: Spiral's sliding window mechanism. Only nodes within the window are visited.

This bias in choosing unvisited nodes makes the walk spiral–like. The window mechanism enforces the rule that lower–level nodes must be visited before higher so that the walk will largely follow a "near to far" progression, while providing for some leeway in the search so that the walker does not have to unnecessarily revisit nodes to find nodes at the proper level. The proper window size depends on the network density: when the network is dense the graph component in Figure 3.1 will be well embedded and the window size can be small; when the network is sparse, however, the graph component will be poorly embedded and the window size must be big enough to achieve a good trade–off between search efficiency (i.e. having as few revisits of nodes as possible) and the route optimality (i.e. finding the closest data object and returning an optimal route).

The window is adjusted when all nodes at the lowest level in the window have been visited. This raises the question of how the walker determines that all nodes in a level have been visited. This is particularly difficult because the walker does not carry the topology of the explored graph with it. Spiral solves the problem using a "level counting" mechanism: the walker counts the unvisited nodes at each level in the window. These counters are initialized to zero when the search begins. When a node forwards the walker it increments the walker's counters based on the levels of its unvisited neighbors. Each unvisited neighbor overhears the message and records the fact that it was accounted for in the counters. When a neighbor is subsequently visited it uses this information to decrement the proper counter. When all nodes in a level are visited the counter for that level will drop to zero and the window adjusted accordingly.

For sparse networks, the sliding window size has a big impact on the route discovery cost and the resulting route length: the smaller the window the higher the route discovery cost and the shorter the route. However, for dense networks, the impact of the sliding window size is minor: the search can easily follow a perfect "spiral" trajectory, resulting both a short route and minimum cost.

Spiral adapts the window size dynamically based on the average node degree of the network backbone. The window size W is initially set to a small value (e.g. two). As it progresses the walker measures the average node degree d_b of the backbone and sets $W = C/d_b$, where C is a constant. Our experiments indicate that this simple mechanism achieves a good balance between the route discovery cost and the route length for sparse networks.

3.3.4 Spiral Details

In Spiral the walker contains the fields shown in Table 3.1.

Table 3.1: Fields in a Spiral walker

Field	*Meaning*
$M.source$:	source node
$M.query$:	query body
$M.state$:	state of the algorithm
$M.next$:	next node in the walk
$M.parent$:	parent node in the search tree
$M.flag$:	indicate whether current node is a dead–end
$M.C[0..W]$:	level counters (W is the size of the level window)
$M.l_{min}$:	level window lower bound
$M.l_{max}$:	level window upper bound
$M.N$:	the neighbor set of current node
$M.L$:	$L(v)$ ($v \in M.N$) is the estimated level of v

There are 4 algorithm states: FWD, indicating the walker is searching normally; BACK, indicating the walker is backtracking to find an unvisited node in the window; CHECK, indicating the walker is returning to the source to see if the search should stop because a route has been returned; and RESPONSE, indicating that the desired data have been found and the message is a response back to the source.

At a network node v, the variables used in Spiral are shown in Table 3.2. The neighbor set N_v and the backbone neighbor set D_v are computed by the underlying backbone generation algorithm. Other variables are maintained by Spiral. Initially, $parent_v$ is $null$, $level_y$ and all elements in L_v are $infinity$, U_v equals D_v, B_v, H_v and R_v are empty.

Table 3.2: Variables maintained in node v by Spiral

Variable	*Meaning*
N_v:	neighbor set of node v
D_v:	backbone neighbor set of v, $D_v \subseteq N_v$
$parent_v$:	the parent of v in the search tree
$level_v$:	the estimated level of v
L_v:	$L_v(w)$ ($w \in N_v$) is the node w's estimated level at node v
U_v:	the set of unvisited nodes in D_v;
B_v:	the children of node v in the search tree;
H_v:	$H_v(w)$ ($w \in B_v$) is the lowest level of unvisited nodes in the subtree rooted at child w;
R_v:	level counters for v;

The Spiral algorithm is formally described in Procedures 1–3. For Procedure 2 message M is received by node x. For Procedure 3 the message M is sent from node x and overheard by node y.

Procedure 1 Spiral(s, $query$)

```
$M.source \leftarrow s$;
$M.query \leftarrow query$;
$M.state \leftarrow FWD$;
$W \leftarrow$ default size
$M.C[0..W] \leftarrow 0$; $M.l_{max} \leftarrow W - 1$; $M.l_{min} \leftarrow 0$
$\forall x \in N_s : L_s(x) \leftarrow 1$;
$level_s \leftarrow 0$;
$parent_s \leftarrow null$;
Call SpiralWalk($s$, $M$) on $s$;
```

Spiral algorithm is in nature a DFS search with iterative deepening, bias on the node level and control on the levels allowed to visit. DFS is known to be a graph search algorithm that visits all vertices of the graph [22]. When the network is

Procedure 2 SpiralWalk(x, M)

```
if check interval has elapsed then
  $M.state$ ←CHECK
  Send $M$ to $M.source$ along the gradient path $r$
end if
if node $x$ satisfies $M.query$ then
  $M.state$ ←RESPONSE;
  Send $M$ to $M.source$ along the gradient path $r$
end if
/*Decide next hop*/
$n \leftarrow null$;
if $\exists y, y \in U_x$ and $L_x(y) \leq M.l_{max}$ then
  $n \leftarrow y : L_x(y)$ is minimal
end if
if n≠null then
  M.state←FWD;
  $B_x \leftarrow B_x + \{n\}$; $H_x(n) \leftarrow L_x(n)$
else
  $M.state$ ←BACK;
  if $B_x = \{\}$ and $U_x = \{\}$ then
    $M.flag$ ←TRUE
  else
    $M.flag$ ←FALSE
  end if
  if $\exists z, z \in B_x$ and $H_x(z) \leq M.l_{max}$ then
    $n \leftarrow z : H_x(z)$ is minimal
  else
    $n \leftarrow parent_x$;
  end if
end if
/*Level counting*/
if $x$ has not been visited previously then
  $\forall c \in R_x$: $M.C[c - M.l_{min}] - -$;
  $\forall y \in U_x$: $M.C[L_x(y) - M.l_{min}] + +$;
end if
/*Sliding window maintenance */
Shift $M.C[0..W]$ leftward for $k$ elements until $M.C[0] \neq 0$
$M.l_{min} + = k$;
/*Adjust window size*/
Update $d_b$ for average node degree;
$W \leftarrow max(C/d_b, M.l_{max} - M.l_{min})$
$M.l_{max} = M.l_{min} + W - 1$
$M.N \leftarrow N_x$; $M.L \leftarrow L_x$;
$M.parent \leftarrow parent_x$; $M.next \leftarrow n$;
$M$ is forwarded from $x$ to $n$;
$\forall y \in N_x$: Call HearMsg$(x, y, M)$ on $y$;
Call SpiralWalk$(n, M)$ on $n$;
```

Procedure 3 HearMsg(x, y, M)

```
if $M.state$ = CHECK or RESPONSE then
  return;
end if
/*Maintain the search tree*/
if $M.next = y$ and $M.state = FWD$ then
  $parent_y \leftarrow x$
else if $M.parent = y$ and $M.state = BACK$ then
  if $M.flag = TRUE$ then
    $B_y \leftarrow B_y - \{x\}$
  else
    $H_y(x) \leftarrow M.l_{max} + 1$
  end if
end if
/*Maintain the level counting records*/
if $y$ is an unvisited dominating node and $x \in U_y$ then
  $R_y \leftarrow R_y + \{M.L(y)\}$
end if
$U_y \leftarrow U_y - \{x\}$
/*Update the level estimates*/
$level_y \leftarrow min(level_y, M.L(y))$
$\forall v \in N_y$: $L_y(v) \leftarrow min(L_y(v), level_y + 1)$
$\forall v \in N_y \cap M.N$: $L_y(v) \leftarrow min(L_y(v), M.L(v))$
if node $y$ satisfies $M.query$ and $y$ hasn't responded yet then
  Send response $M_r$ to $M.source$ along the gradient path $r$;
end if
```

connected, DFS search returns when it finds the data object, or eventually visits all nodes in the network. As long as the destination data object is in the network and the network topology is a connected graph, DFS search will find it. DFS search with iterative deepening is also proven to find the destination with running time asymptotically the same as DFS [46]. Spiral is a variation of DFS search with iterative deepening; it keeps track of all detected but unvisited nodes and accesses them following the level order. Similar to DFS search and DFS search with iterative deepening, it is also guaranteed that if the network is connected and the destination data object is in the network, Spiral search will find the data object.

3.4 Experimental Results

3.4.1 Experimental Setup

I simulated the performance of Spiral and the other algorithms, including Flood, ERS, Random Walk and DFS, on a custom simulator. This was preferable to using a heavy–weight simulator such as NS–2 [70], GlomoSim [75], etc., because we were interested in comparing the high–level search characteristics of the algorithms and not the low–level protocol performance. In the simulation only the high–level network characteristics are considered, such as the nodes' connectivity, message transmissions, etc. Other details, such as the MAC level protocol, signal propagation, collisions, etc, are ignored. Not simulating the low level details does not impair the comparison and the conclusion. Flooding is known for its *broadcast storm* problem [56], which causes serious collisions and makes packet transmissions more costly. Comparing the algorithms on high–level is actually more favorable for flooding algorithms. I anticipate that the flooding algorithms will have even worse performance in experiments with lower level details simulated.

In the simulations, the graphs corresponding to a sensor network topology are generated at random, and the data objects are randomly distributed to the nodes of the graph. For each graph, a backbone is computed using the CDS algorithm by Wu and Li [72]. All algorithms search the backbone during route discovery. We generate queries from randomly chosen nodes for randomly chosen data objects. All algorithms are run in the same graphs using the same queries. The performance results of each algorithm on each search configuration are recorded and averaged.

The graphs for the simulation are generated using the *random unit graph* model [41]. In order to study the algorithms' performance on graphs of different densities, we generated graphs according to the desired average node degree d (which represents the density). We adopted the same method used by Stojmenovic et al. [68] in which the nodes' (x, y) coordinates are selected as random values in the interval $[0, 100)$, all $n(n-1)/2$ potential edges are sorted by the length, and the wireless transmission radius R is set to the $nd/2$-th shortest length. Every pair of nodes within distance R is connected by an edge. After the graphs are generated, only the fully connected graphs are kept. Although the selection results in graphs that are not stictly random, it is used as we do not consider the problem of searching in disconnected networks and this graph model has been widely used [68, 35, 24]. Other graph models, such as *grid with perturbations* [14], have been proposed. I used the random unit graph model because of its simplicity and its wide usage. For each value of average node degree, multiple graphs are generated and used for experiments, and the average of the results is computed.

In addition to Spiral we simulated flood, DFS, random walk, and ERS. We do not include Space-filling Based Routing [58] and Push–and–Pull strategy [1, 25, 50, 64] because they require geographic location information. The flood algorithm is a pure flood without constraints on the scope (i.e. flooding the whole network), we simply call this algorithm "Flood"; the DFS algorithm is also a regular DFS search (i.e. with random choice of the next unvisited node, tracing back along the coming path for unvisited nodes whenever entering a dead end); the random walk algorithm is the biased random walk proposed by Avin et al. [4], we simply call it "Random". The ERS algorithm proposed by Sadagopan et al. [65] linearly increased the search radius of the ring; however, in our experiments we found this to be extremely inefficient. Instead, we used a scheme proposed by Cheng et al. [74], in which the search radius of the ring is increased exponentially after each failed search.

Spiral adjusts the level window size dynamically according to the formula $W = C/d_b$ as described in Section 3.3.3. We set C to 40 in our experiments because this produced reasonable results; the algorithm was not sensitive to the particular value of C. For the periodic checking mechanism described in Section 3.2.2, we set the initial interval to 5 steps and the maximum to 200 steps.

The metrics to evaluate the performance of the algorithms include the route discovery cost, the resulting route length, and the search time to find the route. The route discovery cost is a measure of communication overhead and energy cost. Assuming that the control overhead in the messages is much smaller than the message body (i.e., the query being carried), the messages in all algorithms have roughly the same size so that the communication and energy cost are proportional to the number of message transmissions. The resulting route length, measured in hops, reflects the cost of communication after the route has been discovered. The search time is the elapsed time from when the source node issues the query until a route is found. Minimizing search time is not our main focus, because as a rule, a walk–based route discovery always sacrifices search time for savings in communication cost and energy consumption. However, we measured it in our experiments and present the results here.

3.4.2 Network Density

In these experiments there are 500 nodes in the network, two copies of each data object, and the network density varied. For each density we generated 30 different random networks, and for each network we generated 1000 different random queries. We computed the average of the 1000 cases for each network, and the average of the networks to obtain the result for each density. The deviations of the average values are not shown but mostly less than 10%.

Figure 3.3 shows that when the network is sparse (with average node degree of about 6), the walk–based algorithms (DFS, Spiral) have route discovery costs close to that of ERS; Random has much higher costs. However, when the network is sufficiently dense, the walk–based algorithms have significantly lower route discovery

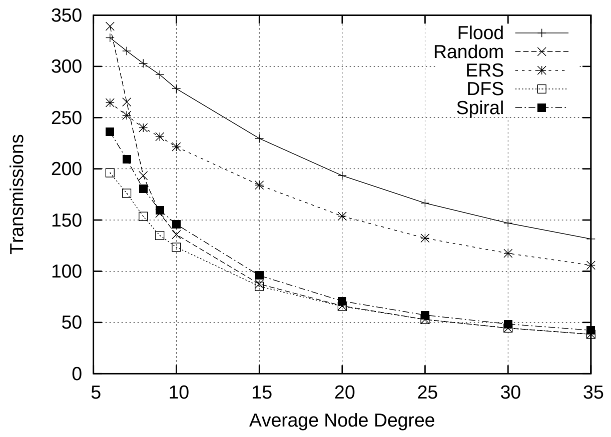


Figure 3.3: Route discovery cost vs. network density.

costs. When the average node degree is 20, Spiral requires only 46% of the messages required by ERS. Spiral requires more transmissions for route discovery than DFS and Random, however, when the network is dense (average node degree greater than 20), the difference is minor, about 8% to 10%.

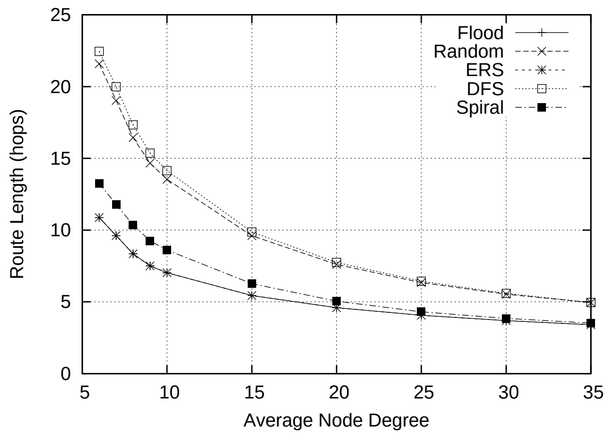


Figure 3.4: Route length vs. network density.

Figure 3.4 shows that the resulting route lengths in DFS and Random are similar, but both are significantly longer than Flood and ERS. The curves of Flood and ERS overlap because both find optimal routes. Spiral produces route lengths close to that from Flood and ERS, especially when the network is dense. When the average node degree is 20, the routes returned by Spiral are only 10% longer than the flood–based

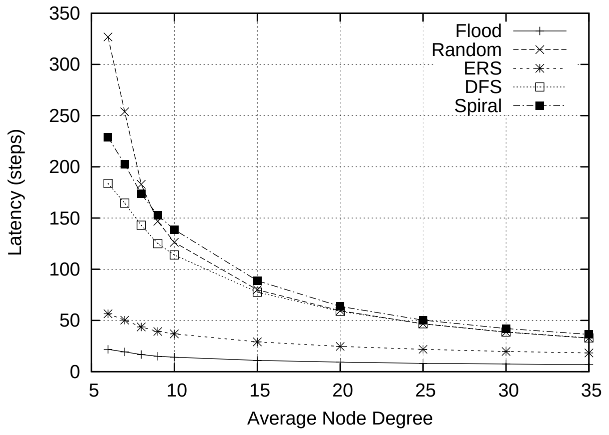
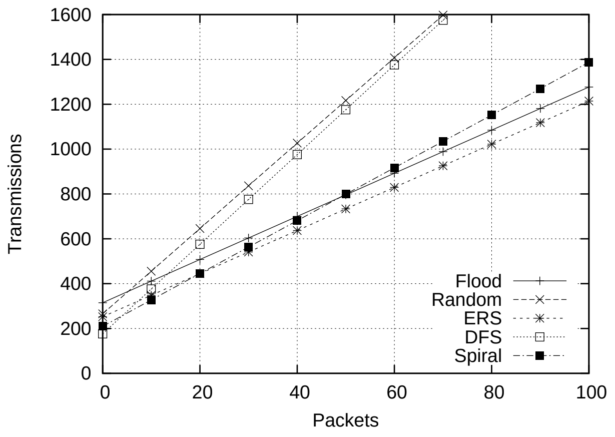


Figure 3.5: Search time vs. network density.

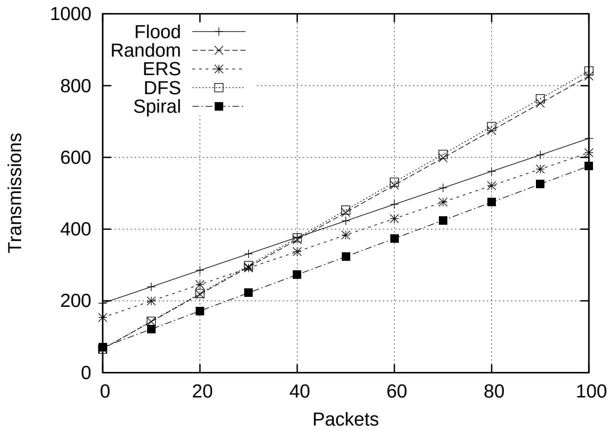
algorithms, and only about 66% of the route lengths returned by DFS or Random.

Figure 3.5 shows that all walk–based algorithms (Spiral, Random and DFS) have search times significantly higher than the flood–based algorithms (Flood and ERS). Spiral has a search time slightly higher than DFS and Random due to its level window mechanism, which leads to more frequent backtracking for unvisited nodes. The results indicate that walk–based algorithms are good for those applications that do not have stringent requirement on the search time.

The results in Figures 3.3 and 3.4 allow us to compute the benefit of Spiral as a function of the *communication length* (the number of packets transferred during the communication). Since Spiral returns longer routes than Flood and ERS but has a lower route discovery cost, we would expect Spiral to be beneficial for short–term communication but not long–term. Figure 3.6 shows the total cost for the various algorithms as a function of the communication length. Two cases are shown, $d = 7$ and $d = 20$ (where d is the average node degree), representing sparse and dense networks, respectively. The results show, when the network is sparse the Spiral algorithm is efficient only for very short–term communication (less than 20 successive packets). However, when the network is dense (the case of $d = 20$), the Spiral algorithm is also good for longer communication because of the lower route discovery costs and close-to-optimal routes. When the communication length is 40,



(a) Sparse network (d=7)



(b) Dense network (d=20)

Figure 3.6: Total cost vs. communication length

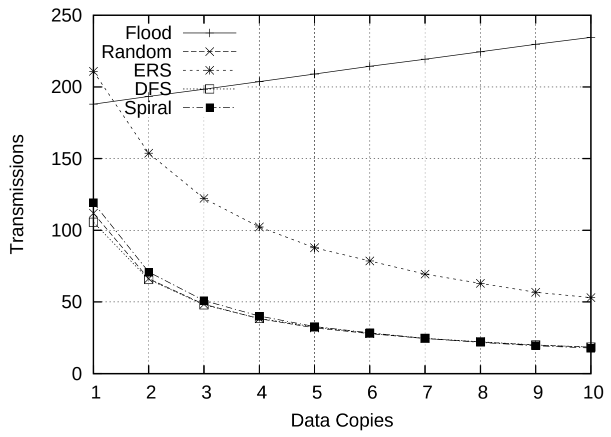


Figure 3.7: Route discovery cost vs. data redundancy.

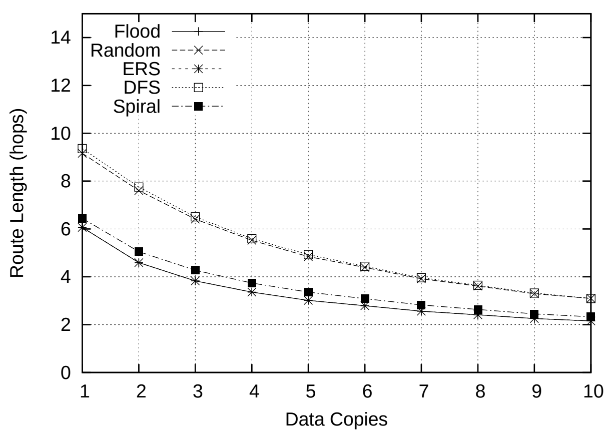


Figure 3.8: Route Length vs. data redundancy.

for example, the total communication cost by Spiral is only 72% of that by Flood, 81% of ERS, 74% of random walk, and 73% of DFS.

3.4.3 Data Redundancy

The second set of experiments measures sensitivity to the number of copies of each data object. The network contains 500 nodes, the average node degree (network density) is fixed at 20, and the number of copies of each data object varied. The results are shown in Figures 3.7, 3.8 and 3.9.

As can be seen, the walk–based algorithms all have about the same route discov-

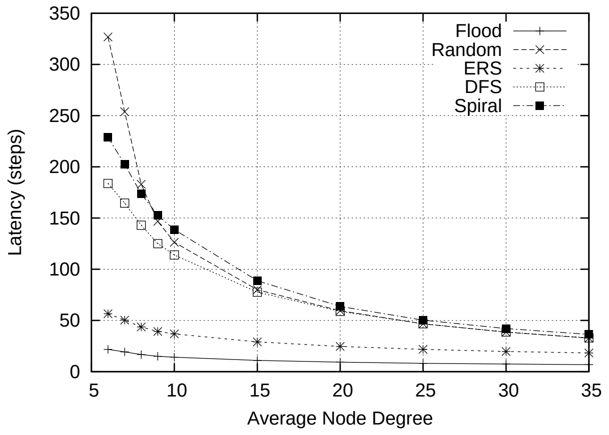


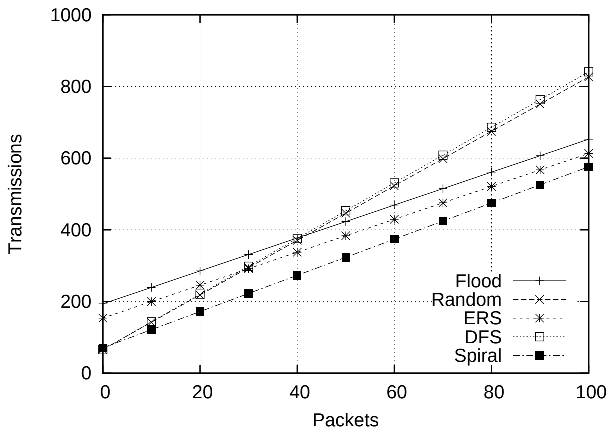
Figure 3.9: Search time vs. data redundancy.

ery costs, which are much less than flood–based algorithms. When data redundancy increases, the route discovery cost decreases for all algorithms except Flood. The cost in Flood increases because each node with a copy of the desired data responds to the request. The route discovery cost in Spiral (and also DFS and Random) is up to 66% less than in flood–based algorithms.

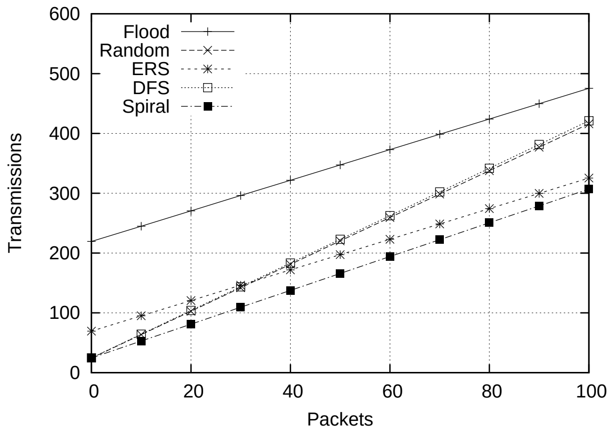
Again, DFS and Random return routes up to 70% longer than the optimal routes returned by flooding, while Spiral returns routes within 12% of optimal. However, as the network density increases the difference in route lengths decreases.

The results show that, for dense networks (average node degree of 20), and for all data redundancy cases, Spiral's route discovery cost and route lengths are close to optimal. However, as the data redundancy increases, the difference between ERS, DFS, Random, and Spiral becomes smaller (Flood is an exception as its discovery cost increases as the data redundancy increases).

Figure 3.10 shows the total cost for the various algorithms as a function of the communication length. Two cases are shown, $m = 1$ and $m = 10$ (where m is the number of data copies), representing networks with low and high data redundancies. In both cases, Sprial has minimum overall communication cost among the algorithms for the communication length shown. DFS and Random walk have low communication cost for short communication but the cost quickly increases as



(a) Low Redundancy (m=1)



(b) High Redundancy (m=10)

Figure 3.10: Total cost vs. communication length

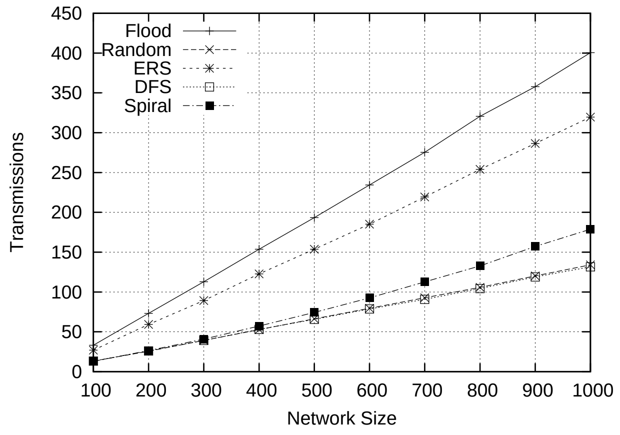


Figure 3.11: Route discovery cost vs. network size.

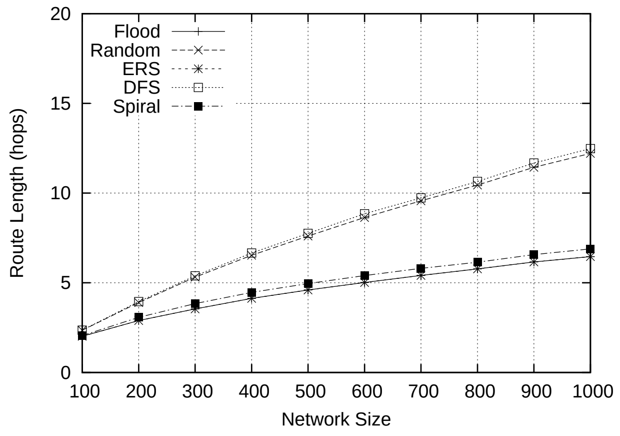


Figure 3.12: Route length vs. network size.

the communication length increases. Flood has cost close to ERS when the data redundancy is low; but when the data redundancy gets high, its cost gets much higher.

3.4.4 Network Size

The third set of experiments measures sensitivity to the network size. The average node degree (network density) is fixed at 20, and the number of data copies (data redundancy) is fixed at 2. We varied the network size from 100 to 1000. The results are shown in Figures 3.11, 3.12, and 3.13.

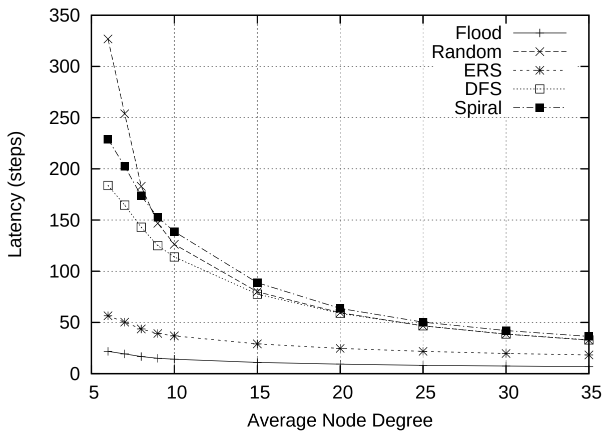


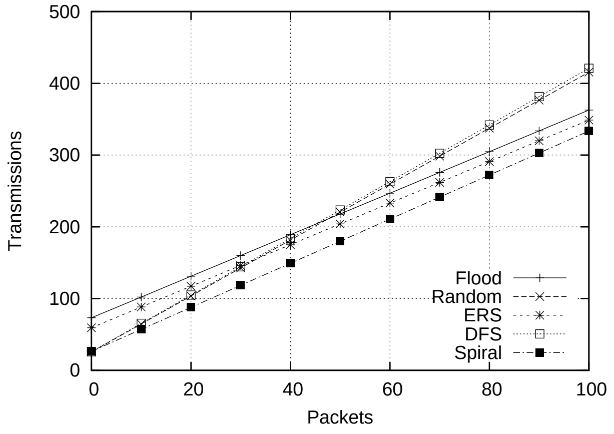
Figure 3.13: Search time vs. network size.

The results show that, for all algorithms we studied, the route discovery costs increase linearly with the network size. Spiral has a growth rate slightly greater than DFS and Random, but still significantly smaller than Flood and ERS. On the other hand, Spiral always returns routes much shorter than those from DFS or Random (as much as 55% shorter), and are close to the optimal routes returned by Flood. Overall Spiral shows good scalability as the size of the network increases.

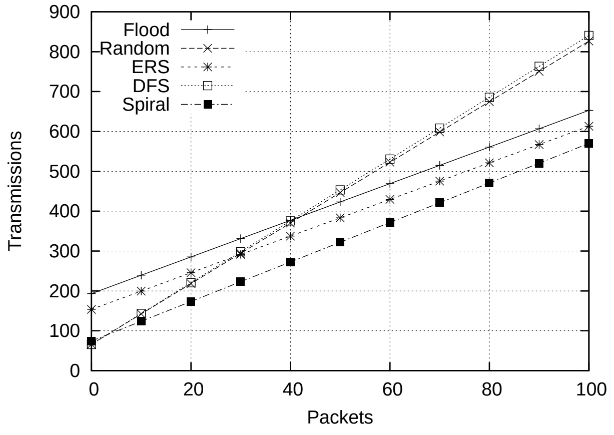
Figure 3.14 shows the total cost for the various algorithms as a function of the communication length. Two cases are shown, 100 nodes and 1000 nodes, representing small and large networks, respectively. Again, Spiral has the minimum total cost among the algorithms being compared. Note that both figures show similar patterns for the comparison between Spiral and other algorithms — Spiral's advantages are independent of network size.

3.5 Conclusions

Spiral is a novel algorithm designed for short–term communication in wireless sensor networks. Spiral balances the cost of route discovery against the length of the route discovered. Its route discovery is much cheaper than flood-based algorithms (typically less than half of their cost for dense networks) and slightly higher than DFS and random walk, but the lengths of the resulting routes are close to those by



(a) small network (N=100)



(b) large network (N=1000)

Figure 3.14: Total cost vs. communication length

flooding, and significantly shorter than those by DFS and random walk. For short–term communication of tens to hundreds of packets, Spiral achieves the lowest overall communication cost.

Spiral works especially well when the network is densely connected. Our experiments show that in dense networks both the route discovery cost and the route length in Spiral are close to optimal. Spiral also works well for both low and high data redundancy cases and shows good scalability as the network size increases.

Walker–based search algorithms can use multiple walkers to search the network in parallel and reduce the search time. Multiple Random Walker approaches are initially proposed and applied for P2P networks [51, 10, 26]; the same approach is also proposed for WSN query processing [66, 28, 4]. Multiple Random Walkers reduce the search time at the cost of potentially higher number of message transmissions, as different random walkers may have overlap in their coverage. Multiple walkers approach may also be applied to Spiral algorithm to reduce the search time, although the level window and level counting mechanism need changes to cope with multiple walkers. One possible method is to maintain the level window and level counting states in the query source, and have each walker periodically check with the query source to synchronize its level window and counts. The detailed design of Spiral search algorithm for multiple walkers is future work.

CHAPTER 4

QUERY PROCESSING IN MOBILE SENSOR NETWORKS

4.1 INTRODUCTION

This chapter presents an efficient and robust algorithm, called CNFS (Closest Neighbor First Search) for query processing in mobile wireless sensor networks. While Spiral algorithm is efficient for short–term data–centric routing in static sensor networks, it does not fit the mobile sensor networks as the algorithm does not handle topology changes. CNFS provides a solution for efficient and robust query processing in mobile sensor networks.

To process a query a base station initiates a query message that is propagated among the sensors until the query is satisfied. The response is returned to the base station along a path discovered during the query propagation. Two of the most important metrics when evaluating query processing schemes are efficiency and robustness. The efficiency is measured as the number of messages and the latency required to satisfy the query. Minimizing the number of messages is important because sensor nodes normally have very limited power and message transmissions dominate a node's power consumption. Excessive messages can also increase network congestion. Many application scenarios also have requirements on query latency. Robustness is important because queries may fail due to mobility and unreliable communication. An acceptable query processing scheme should maximize the success rate despite network dynamics. Compared to algorithms based on flooding or random walk, CNFS requires fewer message transmissions and is robust against network dynamics.

Conventional query processing algorithms use blind search to satisfy the query – that is, no global information about the network is gathered prior to initiating the search. Such information might include a hierarchy of nodes or an index of the data they contain. This information would obviously be useful, but is impractical to maintain in the face of mobile nodes that can cause the network topology to change rapidly. The potentially high data generation rate from sensors in a sensor network also makes the maintenance of the indexing mechanism costly. For these reasons query processing algorithms typically employ either flooding or random walk. Flooding floods the network with the query to find the answer. It returns the answer very quickly, and is therefore highly-tolerant of changing network dynamics, but it requires an excessive number of messages and can congest the network. In contrast, random walk uses a single message that randomly visits the network nodes. This reduces the message overhead and avoids congestion, but can be very slow and may not complete at all if the network topology changes such that the source or destination cannot be visited. For this reason it has a much lower success rate than flooding.

CNFS is based on a directed blind search, so that no pre-existing structure is imposed on the nodes or the data they contain. The search is directed by topology information collected as it progresses, allowing CNFS to be both efficient and robust. CNFS collects and maintains a partial map of the network during query processing, from which it finds an optimal order of visiting nodes. CNFS uses a biased search scheme, i.e., whenever there are several unvisited nodes to choose from as next step, it chooses the one closest to the source node. The partial map is also used to compute the shortest return to the source, as well as determine alternative routes when links break, helping to improve both efficiency and robustness. For example, the experimental results show that for query processing in a 100-node network with average node degree of 8.3, CNFS requires about 38% fewer messages compared with the other methods. In addition, its success rate is significantly higher than random walk-based algorithms for all network densities, and comparable to flooding in dense networks.

4.2 Problem Description

Processing a query in a mobile wireless sensor network involves finding a desired data object located on an unknown sensor in the network. Consider a sensor network with N sensor nodes. One of these nodes, node S, initiates a query for a data object D that resides on an unknown node in the network. The query processing algorithm must find D and return it to S within a given time interval T. To simplify the analysis we assume that the query request and response each fit into a single message. The goals are to minimize the number of messages required to satisfy the query, while maximizing the number of queries that succeed before the time-out T.

We consider mobile sensor networks with the following characteristics:

- There is no pre-existing hierarchy or indexing mechanism in the network, thus query processing requires blind search.
- There is a low–level mechanism, such as beacon messages, that enables each node to know its neighbors. This is necessary for all walk–based algorithms.
- The network contains at most hundreds of nodes, allowing CNFS to collect topology information while processing a query and store that information in the query packet.
- The nodes are mobile and move in unpredictable ways. Node speed is high enough that the network topology may change during query processing, which may cause the query to fail.

4.3 Algorithm Design

4.3.1 Overview

The key idea behind CNFS is to collect network topology information while searching the network. We assume every node in the network can detect its neighbors by either beacon messages or overhearing the neighborhood traffic. When the query message visits a node it integrates the neighborhood information from that node into the network topology. The collected topology information is pieced together to form a map for the part of the network that has already been explored, plus the unvisited nodes that are directly adjacent to the explored part.

The map is used in several ways. First, it is used to bias the search to visit nodes close to the source (when choosing a neighbor to visit next, the unvisited neighbor estimated to be closest to the source is chosen). For dense networks, this bias leads to a spiral–like search path. If there are multiple nodes that satisfy the query this is likely both to find one close to the source and discover a short route to it. Note that this bias may not find the shortest path for a given query in a given network; however, the spiral–like path avoids unnecessarily revisiting nodes and locates an answer closest to the query source, hence reducing the average cost.

Second, the map reduces revisits of nodes. When a CNFS search reaches a dead end (all neighbors have already been visited), it uses the map to compute the shortest path to an unvisited node, where the search resumes. In contrast, DFS must backtrack to find an unvisited node and random walk simply continues visiting nodes randomly.

Third, the topology information allows the search to tolerate changing network topologies. DFS is sensitive to broken links, and may fail altogether if it encounters a broken link while backtracking. Random–walk algorithms, such as ACQUIRE [65], use the search path to return the result; the query may fail if a link breaks. CNFS uses the topology information to compute an alternate route when a link breaks.

4.3.2 Maintaining a Routing Tree

A potential issue with CNFS is the size of the topology information it collects. In the worst case, the topology information is O(N^2), where N is the network size. This causes two problems, one of which is that the information may not fit in a single query message. The other is that computing the shortest path takes time O($M + NlogN$) [22], where M=O(N^2) is the number of edges in the graph. This may be too slow for large graphs.

As a result, CNFS does not collect the topology of the entire network, but instead constructs a tree rooted at the query source. The visited sensor nodes are the internal nodes of the tree, while the unvisited sensor nodes are the leaves. Whenever a node v is visited it becomes an internal node whose leaves are its neighbors that are not already in the tree. Neighbors that are already in the tree are moved so that v is their parent if that makes them closer to the root. The routing tree is actually an approximation of the shortest path tree rooted at the source node; from this routing tree it is easy to compute every node's distance to the query source. The routing tree itself, plus the local one–hop topology, serves as a map for computing shortest paths from the current node to any other unvisited node in the network. Because the number of edges in the tree is $M = O(N)$, the computation only takes O($NlogN$) time. The routing tree and the local topology are shown in Figure 4.1. The node s is the query source. The node v is the current node being visited by the query message. The dashed lines are the network links. The bold solid lines are the tree links stored in the message. The circled portion of the network is the local topology that is accessible by the query message on the current node.

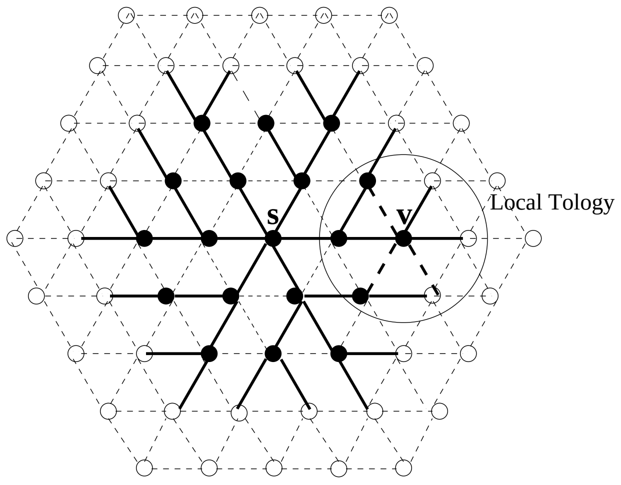


Figure 4.1: Routing tree and local topology

Although maintaining a routing tree does have several advantages, network dynamics may break tree links. CNFS uses a simple repair mechanism to handle broken links. When the current node detects a broken link in the routing tree it removes the link, which partitions the tree. The node then checks the local one–hop

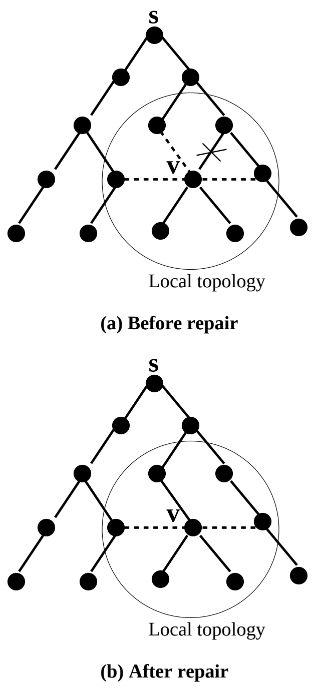


Figure 4.2: Tree repair

topology and if there exists at least one valid link connecting the two partitions, it adds the link to the tree and the tree is repaired. Otherwise does not repair the tree and continues the search in the reachable partition. The tree repair process is shown in Figure 4.2. Figure 4.2.(a) shows that the query message encounters a broken link at node v when tracing back along the routing tree. Figure 4.2.(b) shows the tree is repaired by replacing the broken link with a valid link from the local topology. The query message continues traversing the new tree. Our experiments show that this simple mechanism greatly enhances the robustness of the algorithm against network dynamics.

4.3.3 CNFS Details

The CNFS algorithm is described by the pseudo-code shown as Procedures 4–6. The query Q starts at node s; x is the current node being visited; T is the routing tree encoded in the query message; p_x is the parent of node x in T; U is the set of unvisited leaf nodes in T; N_x is the neighbor set of x; G_x is the one–hop topology of x; and *state* indicates the state of the algorithm. The algorithm has two states, SEARCHING and REPORTING, indicating whether the query is still searching for the destination or the answer is being returned to the query source, respectively. For Procedure 5 message Q is received by node x. Procedure 6 specifies how a tree link break is repaired.

According to the algorithm, the computation on each node takes O($NlogN$) time, where N is the number of nodes in the network. Although it is non-trivial, when the network size is small (tens to hundreds of nodes), the computation time takes less than one millisecond on modern processors.

Procedure 4 CNFS(s, Q)

```
$T \leftarrow empty\_tree$
add node $s$ to T, $p_s \leftarrow null$, $U \leftarrow \{s\}$
$state \leftarrow SEARCHING$
call CNFSWalk($s$, $s$, $Q$, $state$)
wait until response returns or time out
if response returns then
  return query result
else
  return $fail$
end if
```

Procedure 5 CNFSWalk(s, x, Q, $state$)

```
if $state$=REPORTING then
  /* In REPORTING state */
  if $x = s$ then
    stop and deliver the answer
  else
    repeat
      $y \leftarrow p_x$;
      if link $(x, y)$ is broken then
        call RepairTree($T$, $x$, $(x, y)$)
      end if
    until link $(x, y)$ is alive OR $y = null$
  end if
else if $state$=SEARCHING then
  /* In SEARCHING state */
  if $x$ is not visited before then
    mark $x$ as visited, $U \leftarrow U - \{x\}$
    $\forall v$, $v \in N_x$ and $v \notin T$:Add $v$ to $T$, $p_v \leftarrow x$, $U \leftarrow U \cup \{v\}$
    $\forall v$, $v \in N_x$ and $p_v \neq x$:If($x$ is closer to $s$ than $p_v$) $p_v \leftarrow x$
  end if
  if $x$ satisfies $Q$ then
    Put query answer into $Q$
    $state \leftarrow REPORTING$
    $y \leftarrow x$
  else
    /* Choose next hop for search, handling possible link breaks */
    repeat
      if $N_x \cap U$ is not empty then
        $y \leftarrow z$: $z \in N_x \cap U$ and $z$ is closest to $s$ in $T$
      else
        $G \leftarrow T \cup G_x$
        compute $r$ as the path to the closest unvisited node in $G$
        $y \leftarrow z$: z is the next hop in $r$ ($z = null$ if $r$ does not exist)
      end if
      if link $(x, y)$ is broken then
        if $(X, Y) \in T$ then
          call RepairTree($T$, $x$, $(x, y)$)
        else
          $G_x \leftarrow G_x - (x, y)$
        end if
      end if
    until link $(x, y)$ is alive OR $y = null$
  end if
end if
if $y = null$ then
  Drop Q;
else
  Send Q from $x$ to $y$, call CNFSWalk($s$, $y$, $Q$, $state$);
end if
```

Procedure 6 RepairTree(T, x, e)

```
delete $e$ from $T$, partition $T$ into $T_1$ and $T_2$, $x \in T_1$
if $N_x \cap T_2 \neq \emptyset$ then
    pick $w : w \in N_x \cap T_2$
    $T \leftarrow T_1 \cup T_2 \cup \{(x, w)\}$
    Update the parents of the nodes in $T$
else
    $T \leftarrow T_1$
end if
```

4.4 Evaluation

4.4.1 Experimental Setup

We measured the performance of CNFS and competing algorithms on a simulator we developed that allowed us to compare the overall search characteristics of the algorithms. The simulator only simulates the core network characteristics, such as the nodes' connectivity, message transmissions, etc., as these are integral to understanding the algorithms' behavior. Other details, such as the MAC level protocol, signal propagation, collisions, etc, are ignored as they have lesser effects.

The simulations generate random sensor network topologies. The queries are also randomly generated: the query source is a randomly selected node, while the answer resides on another randomly selected node. All algorithms are run in the same networks using the same queries. The performance results of each algorithm on each search configuration are recorded and averaged.

The networks for the simulation are generated based on the random unit graph model [41]. The network nodes are randomly distributed in a square area with edge length D, so that all nodes have (x, y) coordinates in the range $[0, D)$. The wireless transmission range is R, meaning that every pair of nodes within distance R of each other is connected by an edge. In the experiments, we used the average node degree as a measure of network density, and varied the size of the area to experiment with different densities. Because the nodes are randomly distributed throughout the area, multiple randomly generated networks have roughly the same average node degree. We use the average value in our results.

In addition to CNFS we simulated Flood, DFS, Random Walk (RW), and ERS. The Flood algorithm we used is a pure flood without constraints on the scope (i.e. flooding the whole network); the DFS algorithm is a regular DFS search (i.e. with random choice of the next unvisited node, tracing back along the search path for unvisited nodes whenever encountering a dead end); and the Random Walk algorithm is a biased Random Walk [5]. The ERS algorithm proposed by Perkins et al. [59] linearly increased the search radius of the ring; however, in our experiments we found this to be extremely inefficient. Instead, we used a scheme proposed by Cheng and Heinzelman [74], in which the search radius of the ring is exponentially increased on each search failure. In both the DFS and Random Walk algorithms, the query answer is returned back via the search path. In Flood and ERS, the query answer is returned along the gradient path formed in the flooding.

The metrics used to evaluate the performance of the algorithms include the *efficiency*, measured as the average message overhead and latency, and the *robustness*, measured as the success rate. We use static networks to measure efficiency so that all the algorithms have about the same success rate of 100%. To measure the robustness against network dynamics we vary the node velocity and measure the query success rate.

4.4.2 Efficiency

This experiment compares the *message overhead* and the *latency* of the various algorithms. The message overhead is the number of messages to satisfy a query, including the messages for search and the messages for sending back the result. We assume either the query includes the routing information or the query result fits into one message. The latency is defined as the time from when a node issues a query until it receives the results.

We use static networks to control the success rate so that all algorithms except for Random Walk have a 100% success rate. Controlling the success rate is important — if failures are simply ignored, for example, an algorithm might achieve high efficiency but at the cost of only a few queries succeeding. Using static networks puts all algorithms on a level playing field in terms of the success rate, allowing for meaningful comparisons of message overhead.

We performed three set of experiments, testing the relationship between the efficiency of different algorithms and various system properties such as *network density*, *data redundancy*, and *network size*. Network density is defined as the average number of neighbors of every node in the network, which is also called the average node degree. Data redundancy is the number of nodes satisfying the same query.

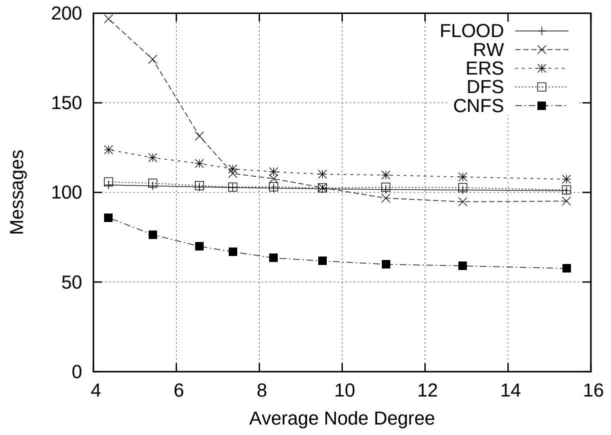


Figure 4.3: Message Overhead vs. Network Density

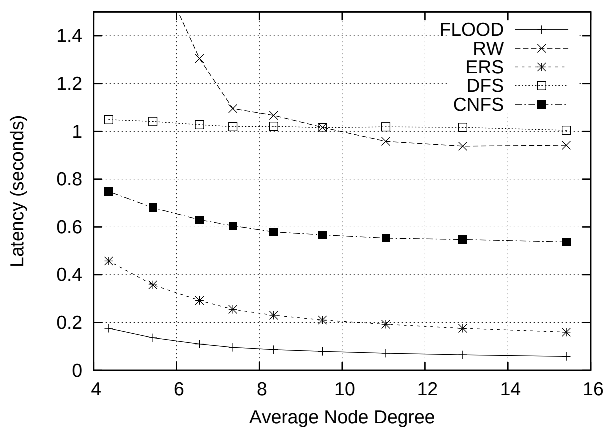


Figure 4.4: Latency vs. Network Density

Network size is the number of nodes in the network.

The first set of experiments measures the relationship between the algorithm efficiency and the network density, given that the data redundancy and the network size are fixed. We set the data redundancy to 1, i.e., each data object has only one copy. The network size is set to be 100. The nodes are randomly distributed across a given area. For each area we generated 20 different random networks, and for each network we generated 1000 different random queries. We computed the average of the 1000 cases for each network, and the average of each network to obtain the result for each density. We assume one message forwarding step takes 0.01 seconds (including message processing and transmission time). The query source sets the query time–out as 5 seconds. If the result is not returned before the time–out the query is considered to have failed. We measured the average number of message transmissions per query for different network densities. The results are shown in Figure 4.3 and Figure 4.4. Note that the presented value is the average value with deviation typically less than 10%, except for Random Walk on sparse networks (as explained below).

The figures shows that CNFS requires fewer messages than the other four algorithms: when the average node degree is 8.3 (the median density in our experiments), CNFS has 38% fewer messages than Flood, 43% fewer than ERS, 38% fewer than DFS, and 41% fewer than Random Walk. Network density has a large effect on CNFS and Random Walk — the message overhead of CNFS and Random Walk decreases as the density increases, although Random Walk does so more rapidly. CNFS has latency greater than Flood and ERS, however, its latency is significantly less than DFS and Random Walk. When the average node degree is 8.3, CNFS has latency 5.68 times higher than Flood, 1.51 times higher than ERS, but 43% lower than DFS, and 46% lower than Random Walk.

All algorithms except for Random Walk experience no query time–outs. Random Walk has a low success rate on sparse networks. This is caused by it visiting nodes at random, which in a sparse network may cause it to get "stuck" in a relatively dense part of the network and revisit the same nodes repeatedly. Our results show that when the average node degree is 4.4, Random Walk has a success rate as low as 0.83. However, the success rate increases as the network density increases, to 0.96 when average node degree is 6.6 and greater than 0.99 when the average node degree is greater than 9.5.

The second set of experiments measures the relationship between the algorithm efficiency and the data redundancy, for a fixed network density and network size. Data redundancy means there are multiple nodes in the network that satisfy the query. Whenever the query message reaches any of them, it returns the result to the query source. In the experiments we vary the data redundancy from 1 to 10 and

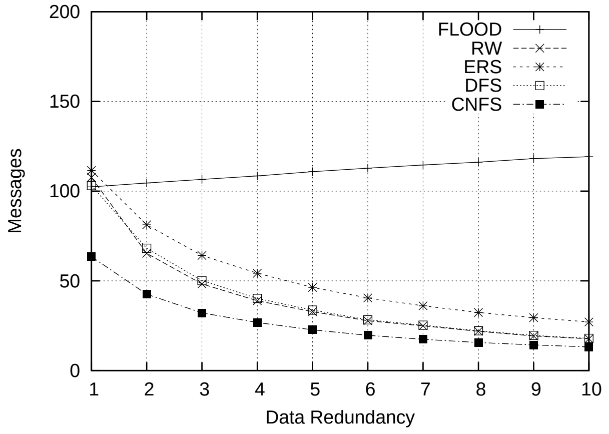


Figure 4.5: Message Overhead vs. Data Redundancy

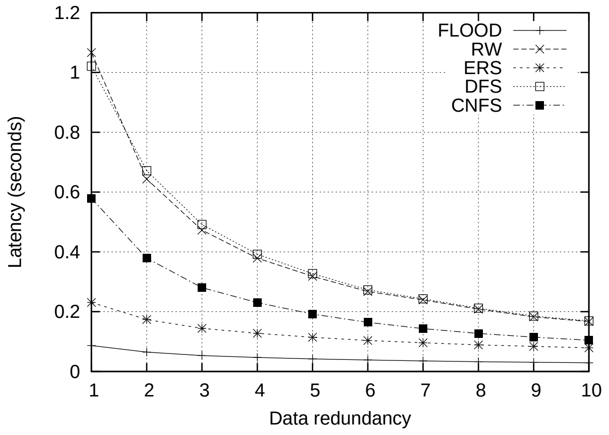


Figure 4.6: Latency vs. Data Redundancy

assume the redundant nodes are randomly distributed. We set the network size to be 100 and adjust the area size so that the average node degree is around 8.3. The networks and queries are generated in the same way as the first set of experiments. Figure 4.5 shows that increasing data redundancy reduces message overhead for all the algorithms except Flood. The message overhead increases in Flood because every node with a copy of the data object responds to the query. CNFS has the smallest message overhead; DFS and Random Walk have similar message overheads, both of which are less than ERS. There are small differences between CNFS, DFS, and Random Walk when the data redundancy is high.

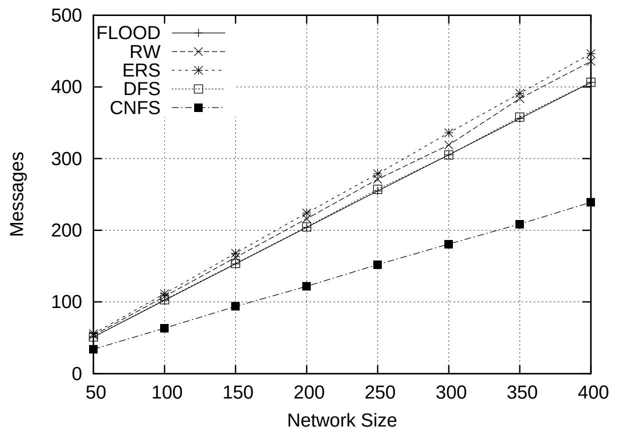


Figure 4.7: Message Overhead vs. Network Size

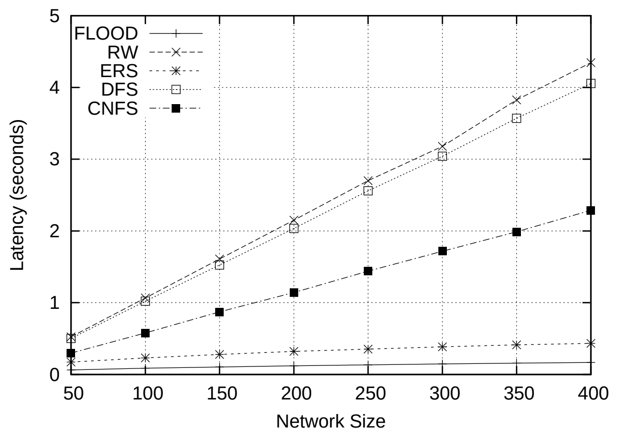


Figure 4.8: Latency vs. Network Size

The Figure 4.6 shows that increasing data redundancy also reduces latency for all the algorithms studied. CNFS has higher latency than Flood and ERS, but much lower than DFS and Random Walk. Again, there are small differences between ERS, CNFS, DFS, and Random Walk when the data redundancy is high.

The third set of experiments measures the relationship between the algorithm efficiency and the network size, for fixed network density and data redundancy. We vary the network size from 50 to 400 with data redundancy fixed at 1. In order to achieve about the same network density for different network sizes, we fixed the average number of nodes per square meter. Since the nodes are randomly distributed, this method results in networks with similar densities. In our experiments there were 5.1x10^{-5}nodes/meter^2 and a 250 meter transmission range, resulting in an average node degree between 8 to 9, with a maximum difference of less than 12%. The networks and queries are generated in the same way as the first set of experiments. The Figure 4.7 shows that increasing the network size increases of message overhead for all the algorithms. However, CNFS has the smallest increase. The Figure 4.8 shows that increasing the network size also increases the latencies for all the algorithms. The latency for CNFS is higher than Flood and ERS, but less than DFS and Random Walk.

4.4.3 Robustness

For this experiment the nodes move according to the Random–Waypoint mobility model [43] in which nodes constantly move at a given speed between randomly–chosen locations (We maximize the mobility by setting the pause time to 0). The moving nodes decrease the query success rate; broken links can either cause the search to fail or the result to not reach the source. The networks tested contained 100 nodes each with a wireless transmission range of 250 meters (assuming an 802.11 outdoor radio range). Two size areas were used: 1200x1200 meters and 2000x2000 meters, representing a dense network (average node degree of 16.6) and a sparse network (average node degree of 6.3), respectively. The node speed was varied and the resulting success rate measured. For each area, we generated 20 different networks, and for each network we generated 1000 different random queries. Using these random queries, we measured the success rates for various node speeds. For a given speed, the average of each network is reported as the success rate for that speed.

Three sets of experiments were done to measure the robustness against node mobility in various scenarios. The first set of experiments test the robustness for both dense and sparse networks. The results are shown in Figures 4.9 and 4.10. For dense networks the CNFS success rate is almost the same as Flood (the two curves almost overlap), only slightly worse than ERS, but much higher than DFS

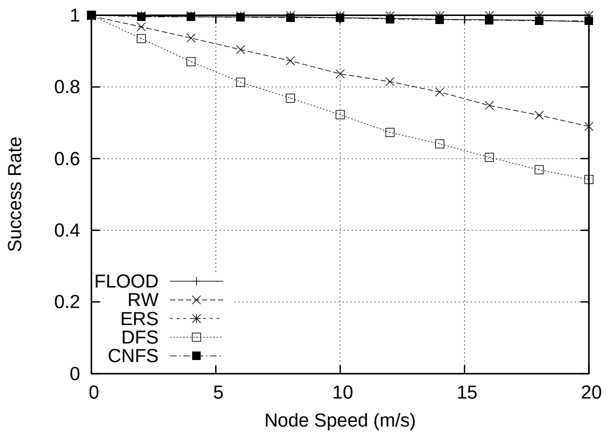


Figure 4.9: Success rate in a dense network (average node degree is 16.6)

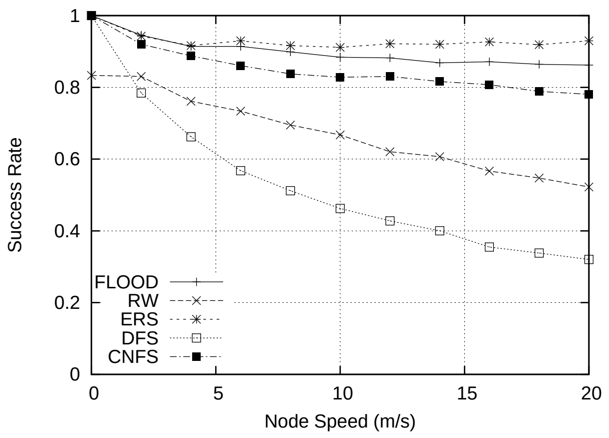


Figure 4.10: Success rate in a sparse network (average node degree is 6.3)

and Random Walk. When the node speed is 10 m/s (the median speed in our experiments), CNFS has a 0.993 success rate, which is only 0.1% lower than Flood, 0.6% lower than ERS, but 19% higher than Random Walk and 37% higher than DFS.

For sparse networks the CNFS success rate is lower than Flood and ERS, but still significantly better than DFS and Random Walk. When the node speed is 10 m/s, CNFS has a 0.828 success rate, which is only 9.1% less than ERS and 6.3% less than Flood, but 24% higher than Random Walk and 79% higher than DFS.

The second set of experiments test the robustness for networks with low and

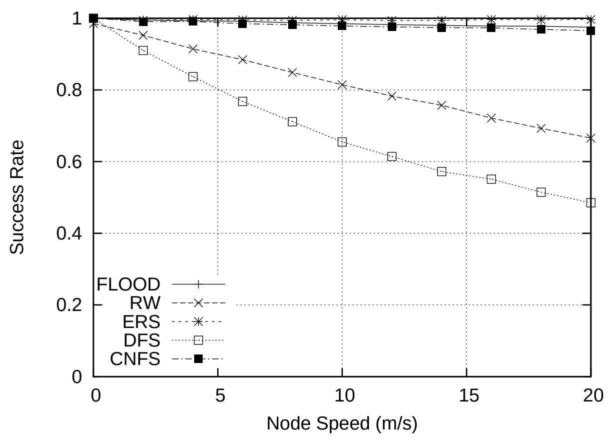


Figure 4.11: Success rate in a network with low data redundancy (r=1)

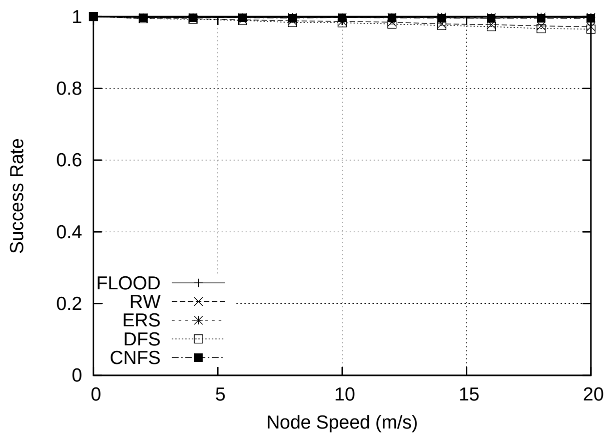


Figure 4.12: Success rate in a network with high data redundancy (r=10)

high data redundancy. The results are shown in Figures 4.11 and 4.12. For both cases CNFS has success rate close to FLOOD and ERS. The maximum difference between CNFS and FLOOD in success rate is 1%, while the maximum difference between CNFS and ERS is 3%.

Although DFS and Random Walk exhibit good success rates (higher than 0.95) for the high redundancy case, in the low data redundancy case their success rates are significantly lower than CNFS. When the node speed is 10 m/s and data redundancy is 1, CNFS has success rate of 0.978, which is 20% higher than Random Walk (success rate of 0.814) and 49% higher than DFS (0.655).

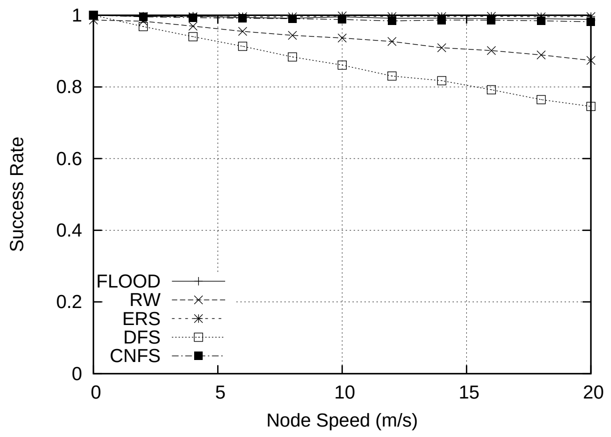


Figure 4.13: Success rate in a small network (50 nodes)

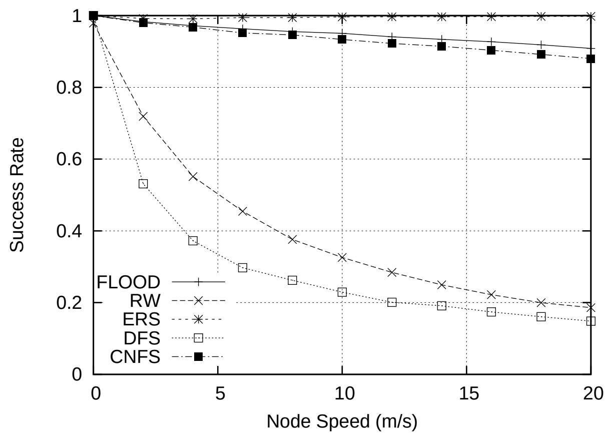


Figure 4.14: Success rate in a large network (500 nodes)

The third set of experiments test the robustness for both small and large networks. The results are shown in Figures 4.13 and 4.14. For small networks (50 nodes) the CNFS success rate is almost the same as Flood and ERS, with less than 1% difference, noticeably higher than DFS and Random Walk. When the node speed is 10 m/s, CNFS has success rate of 0.988, with difference less than 1% compared with Flood (0.996) and ERS (0.998); compared with Random Walk (0.935) and DFS (0.861), CNFS's success rate is 5.6% and 15% higher respectively.

For large networks (400 nodes), CNFS has a success rate lower than ERS, but still close to Flood; DFS and Random Walk have much lower success rates. When the node speed is 10 m/s, CNFS has success of 0.933, while Random Walk and DFS have only 0.326 and 0.229 respectively. CNFS is much more robust than Random Walk and DFS for large networks.

Overall, CNFS is more robust than DFS and Random Walk for all cases we measured, and comparable to Flood and ERS.

4.5 Conclusions

CNFS is a robust and efficient algorithm for query processing in mobile wireless sensor networks. CNFS requires fewer messages than other blind–search algorithms such as Flood, ERS, Random Walk and DFS, resulting in increased efficiency and reduced power consumption. It has query latency longer than flooding but less than Random Walk and DFS. CNFS is also more tolerant of node mobility than random walk–based algorithms, experiencing significantly fewer failures in both sparse and dense networks. CNFS's success rate is comparable to flooding in dense networks, and slightly worse in sparse networks with highly–mobile nodes. For those networks the topology is changing so rapidly that CNFS's topology information isn't as effective as flooding at preventing failures. The combination of fewer messages and failures makes CNFS an efficient and robust query processing algorithm for mobile wireless sensor networks.

CHAPTER 5

ADDRESS AGGREGATION BASED ROUTING FOR MOBILE AD HOC NETWORKS

5.1 Introduction

This chapter proposes a novel routing protocol for Mobile Ad-Hoc Networks called *Address Aggregation-based Routing* (AAR). Whereas Spiral was designed for data-centric routing in static sensor networks, and CNFS for query processing in mobile sensor networks, ARR is designed for routing in mobile ad-hoc networks. AAR performs route discovery reactively, but proactively maintains an index hierarchy that makes route discovery more efficient and avoids flooding in most cases. The index hierarchy dramatically reduces the route discovery cost, leading to lower packet overhead and better support for large networks. AAR resorts to flooding when a route cannot be found using the index, which can happen if the index gets out-of-date as nodes move and links break. This occurs in less than 20% of the route discoveries, so that overall AAR significantly reduces route discovery costs. In addition, AAR has a lower loss rate and end-to-end delay than popular routing protocols because of to its dynamic route repair and route shortening mechanisms.

The index hierarchy in AAR is a *Route Discovery DAG (RDD)* (Figure 5.1). The RDD is a structured DAG (Directed Acyclic Graph) whose edges are physical links in the underlying network. The RDD is structured so that every node knows its distance (in hops) from a chosen *root* node, and every node has directed edges from itself to its neighbors that are closer to the root. Address information is aggregated along the edges of the RDD, so that the root node has information about every node in the network, while the *leaf* nodes (those who have no neighbors farther from the root) have information only about themselves. The address information is aggregated using Bloom filters [11], allowing aggregated address information to fit into a single network packet even for a network with as many as one thousand nodes.

Route discovery is performed by searching the RDD starting at the source node and following the edges towards the root until a match for the destination address is found in the current node's Bloom filter. The search then proceeds along the RDD edges in reverse, visiting those nodes whose Bloom filters match the destination address until the destination is reached. Restricting the search to move towards the root then away from it prevents routing loops. AAR makes use of several optimizations to "short-circuit" the RDD during route discovery by having nodes keep track of the Bloom filters of their siblings (neighbors at the same distance from the root), as well as a 2-hop map of their neighbors and their neighbor's neighbors. This helps offload the nodes at the center of the RDD. AAR also uses the 2-hop map to repair routes that break due to node mobility or failures, and to shorten routes once they are discovered.

Compared with existing routing protocols such as AODV [61] and DSR [43], AAR requires lower communication cost, loses fewer data packets, has a smaller end-to-end delay, and scales to larger networks. Simulation results show that for a network with 400 randomly-distributed nodes, 80 CBR flows, 15 nodes per transmission range, and 10 m/s maximum node speed, the packet overhead of AAR is 38% of AODV and 40% of DSR; its packet loss rate is 17% of AODV and 20% of DSR; and its packet latency is 67% of AODV and 51% of DSR.

5.2 Problem Description

Given a network with N mobile nodes that each has its own its unique ID (address), route discovery is the problem of searching the network starting from the source node S for the destination node D. The search returns a path from node S to D that can be used for subsequent data packets. The goals are to minimize the number of messages required to complete the search, and to return a path between S and D with the minimal number of hops. The two goals can conflict. Flooding search, for example, has search cost of O(N) messages while returning routes close to optimal which is O($\sqrt{N}$). This thesis designs a novel protocol that achieves a good balance between the two goals, with search cost of O($\sqrt{N}$) messages and returning routes with length O($\sqrt{N}$) as well. The low search cost is achieved by constructing and maintaining an efficient indexing mechanism in the network. Building the index requires O(N) messages, but it only done on the order of every 10 seconds and the cost is amortized across all route discoveries.

The new protocol supports MANETs with following characteristics:

- The entire network is connected, i.e., for any two nodes in the network, there is at least one path between them at any time. Although there has been research about routing in MANETs with intermittent connections, that is not our focus.

- The communication link is bidirectional. A node that hears another node can also send message to that node. This requires every node in the network has the same wireless transmission and reception power.

- The nodes move at moderate speed. The network is not static; however, the nodes' mobility is not so high that the index is unmaintainable. For commonly used 802.11 networks, a moving speed of tens of meters per second is assumed, which is the typical moving speed of vehicles.

5.3 AAR Protocol Overview

The key idea of Address Aggregation Routing (AAR) is to organize the network into a hierarchy so that for most cases route discovery searches only the hierarchy rather than flooding the entire network. AAR is an optimization mechanism that reduces the overhead for the majority of route discoveries.

A hierarchical search is much less expensive than flooding: assume n is the number of nodes in the network, d is the diameter of the network (the maximum shortest path length between any two nodes in the network), then $d = O(\sqrt{n})$. The hierarchy is constructed based on distances to the root node, so that the length of a search in the hierarchy is $O(d)$. Flooding search requires $O(n)$ messages, whereas searching the hierarchy only requires $O(d) = O(\sqrt{n})$ messages, and $\sqrt{n} << n$ as n gets large.

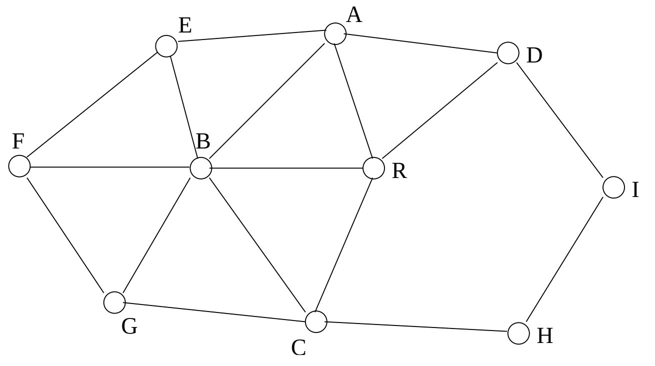
In AAR the Routing Discovery DAG (RDD) is the hierarchy that is searched during route discovery. AAR uses a Directed Acyclic Graph (DAG) instead of a tree for robustness – a tree is easily partitioned when one edge breaks whereas a DAG is not. In addition, in a tree each node has only one parent (a node closer to the root), whereas in a DAG each node may have many "parents". This decreases the search length and offloads traffic from the nodes at the center of the RDD.

Figure 5.1 gives an example of a MANET and its corresponding RDD. Each node has aggregated address information for the transitive closure of all nodes that have edges to it. In other words, each node has aggregated address information for a sub-RDD for which it is the root. This aggregated address information is encoded in a Bloom filter and is known as a node's *Address Set* (AS).

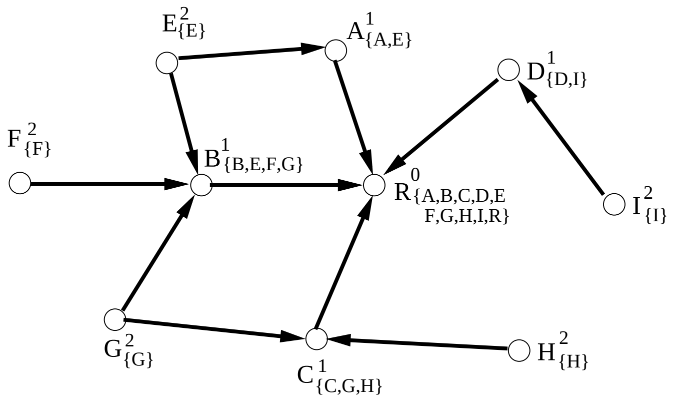
Address aggregation propagates from the leaves to the root following the edges of the RDD. Each node aggregates the Address Sets from its neighbors that have edges to it (neighbors that are farther away from the root) to create its own Address Set, which it broadcasts to its neighbors. In this way, every node creates an Address Set for all nodes in its sub-RDD. A leaf's Address Set contains only itself, while the root's Address Set contains all nodes in the network.

The RDD provides an efficient way to find a route from a source node to a destination node. A route discovery request starts at the source and searches for the destination by moving along the RDD edges towards the root and looking for the destination address in the Address Sets of each node visited. Eventually the destination address will be found in an Address Set, perhaps by visiting the root, at which point the search proceeds down the RDD edges in reverse visiting those nodes that have the destination address in their Address Sets.

AAR periodically rebuilds the RDD to keep it up-to-date with the network topology, but also makes use of a 2-hop map on every node to tolerate link breaks, repair



(a) Network topology



(b) RDD

Figure 5.1: Network topology and corresponding RDD. Figure (a) shows the network topology; Figure (b) shows the RDD rooted at node R. Each node in the RDD is labeled with its distance to the root (superscript) and its Address Set (subscript).

routes, and shorten routes. The 2-hop maps are maintained using a beaconing mechanism.

The following sections provide details on the different aspects of the AAR protocol, including leader election, RDD construction, beaconing, route discovery, packet forwarding, route maintenance, route repair, and route shortening.

5.4 Leader Election

The AAR uses a *leader* node to manage the RDD. The leader is responsible for periodically initiating a rebuild of the RDD. AAR chooses as its leader the node with the lowest node ID (network address) using a leader election algorithm similar to that used in Ethernet spanning tree construction [62].

During the leader election nodes continually broadcast the ID of the lowest node they know of; initially each node broadcasts its own ID, but when it hears of a lower ID it broadcasts that ID. Eventually the only node broadcasting its own ID is the node with the lowest ID, and every other node will be broadcasting the lowest ID as well. When announcing itself as the lowest ID node, a node sets a timeout of a few seconds. If the timeout expires without learning of a node with a lower ID, the node knows it has the lowest ID and starts the RDD construction.

AAR uses `LOWID` messages for the leader election. A `LOWID` message m contains the lowest ID (m_{id}) and the timestamp (m_t) when the message is generated from the lowest ID node. The timestamp value does not change when the network nodes rebroadcast the `LOWID` message. Every node maintains the current known lowest ID (id) and its timestamp (t). When receiving a `LOWID` message, a node checks the lowest ID and timestamp in the message:

- If $m_{id} < id$, $id = m_{id}$, $t = m_t$, rebroadcast the `LOWID` message;
- If $m_{id} = id$, $t > m_t$, $t = m_t$, rebroadcast the `LOWID` message;
- If $m_{id} > id$, replace $m_{id} = id$, $m_t = t$, broadcast the new LOWID message;
- Otherwise drop the LOWID message;

The nodes periodically check for the timestamp of the lowest ID node. If the timestamp is older than `LOWID_EXPIRE_TIME` seconds, the stored lowest ID is discarded and reset to be the node itself. It then announces itself as the lowest ID and triggers a new round of leader election.

The leader node announces its existence during the periodic RDD rebuilds, so that every node can refresh the timestamp for the lowest ID and avoid restarting the leader election. If the leader dies or leaves the network, the lowest ID information stored on each node eventually expires and a new leader is elected.

When a new node joins the network, it initially takes itself as the lowest ID node and sends the corresponding `LOWID` message. If it is the new lowest ID node, the

LOWID message will propagate through the network and the new node is elected as the leader. Otherwise the neighbors of the new node will tell the actual lowest ID and the new node will recognize it.

5.5 Spanning Tree versus DAG

Two indexing hierarchies were considered for AAR. One is a Spanning Tree (ST) with node addresses aggregated along the tree path. The other one is a directed acyclic graph (DAG), in which the node addresses are aggregated along the path in the acyclic graph towards a common root.

The major benefit of ST is the simplicity, since each node has one parent node. During the address aggregation process each node sends the aggregated Address Set to its parent, which is simple and efficient in communication cost.

However, ST has two drawbacks. One is the robustness, as any link break partitions the tree. Although using a 2-hop local map can bypass the broken link (discussed in Section 5.7), frequent link breaks cause excessive bypass overhead. The other drawback is congestion. The ST returns only one route for any source/destination pair. Many of these routes traverse the nodes near the root, causing unbalanced traffic and congestion on these nodes when the traffic load on the network is high.

For these reasons, AAR uses the more complex DAG instead ST. In a DAG, a node has multiple parents to whom it must send the aggregated address set. AAR broadcasts to make this efficient. Instead of sending the message to the parents individually, AAR broadcasts the aggregated address set so that all the parents can receive the message. However since the wireless broadcast is not reliable, AAR uses acks and polling for reliability.

The DAG alleviates the major drawbacks of ST. Since every node has multiple parents, if one fails it can choose another one for route look up. Hence DAG is more robust than ST. In addition, because a node address is sent along all paths towards the root node, the route lookup for that node will choose the one that has minimum distance and off the root of the DAG. In this way, the congestion problem is alleviated.

5.6 RDD Construction

The leader is responsible for periodically constructing the RDD. It first chooses a root node for the RDD. Ideally, the RDD would be rooted in the center of the network so as to minimize the average search cost. For simplicity's sake the current AAR implementation uses the leader as the root.

Once chosen, the root node constructs the RDD in 3 stages: (1) Constructing the DAG; (2) Address aggregation; and (3) Completion.

5.6.1 Constructing the DAG

In the first stage, the root initiates the construction process by broadcasting a `CONSTRUCT` message that contains the root node address, the RDD sequence number, and the sending node's distance from the root (initially 0). The message is encapsulated in a single MAC packet (except when stated otherwise, every AAR control message is encapsulated in a single MAC packet). When a node receives its first `CONSTRUCT` message, estimates its distance as 1 more than the distance in the message. It then rebroadcasts the message, replacing the distance field with its own distance. In this way the `CONSTRUCT` message floods the network.

Although a node may receive multiple `CONSTRUCT` messages, it determines its distance based on the first message it receives and rebroadcasts the message only once. During RDD construction each node keeps track of the distances of its neighbors to the root, and segregates its neighbors into three groups: *Up* nodes, *Down* nodes, and *Sibling* nodes. Up nodes are neighbors that are closer to the root than the current node, Down nodes are neighbors farther from the root, and Sibling nodes are neighbors at the same distance from the root.

In the RDD there is an edge from each node to every node in its Up group, and every node in the network except the root has a non-empty Up group. Most nodes have non-empty Down groups as well, except for the leaves. A node waits a short time until all its neighbors send CONSTRUCT messages before deciding it is a leaf.

5.6.2 Address Aggregation

Once distances have been determined and DAG edges created, the DAG is used to aggregate addresses from the leaves to the root and create the Address Sets on each node. The Address Sets are implemented using Bloom Filters because of their compact size for representing objects: 1000 network addresses can be easily represented by a 1250-byte Bloom Filter that fits in a single 802.11 MAC frame and has a false positive rate of less than 1%. The Bloom Filter is encapsulated in an ADDRESSES message that is broadcast by each node to its neighbors.

The Address Set of a leaf node contains only that node. When a node receives an ADDRESSES message from a neighbor in its Down group it aggregates the received Address Set into its own Address Set. After it has received an ADDRESSES message from all nodes in its Down group each node broadcasts its Address Set in an ADDRESSES message. Once the root gets Address Sets from all nodes in its Down group the address aggregation ends and the root sends a notification message signaling the completion of the RDD.

During address aggregation, a node may miss an ADDRESSES message due to unreliable wireless communication or node mobility. To avoid waiting forever, each node periodically polls the nodes in its Down group on whom it is waiting. To reduce overhead this poll consists of a broadcasted POLL message that identifies the missing neighbors.

When receiving a POLL message, a node first checks the node list in the message to determine if it is one of the nodes being polled. If so, the node responds with its Address Set if the set is complete, otherwise it sends a POLL_ACK message indicating that it is alive and still aggregating its Address Set. The polling node assumes nodes that fail to respond are dead or have moved out of transmission range; in either case they are removed from the Down group. This allows aggregation to complete even though the wireless communication is unreliable and the nodes are mobile.

The ADDRESSES messages are broadcast to all neighbors, allowing a node's neighbors to learn its Address Set. Knowing the Address Sets of one's neighbors is useful during route discovery, as it allows the search to go directly to a sibling with a matching Address Set rather than be forwarded through a node in the Up set. This improves latency and reduces message overhead, but perhaps more importantly, reduces network traffic through the nodes near the root of the RDD.

5.6.3 Completion

When the root receives Address Sets from its entire Down group it signals the completion of current RDD by flooding a COMPLETE message to all nodes. This causes the nodes to start using the new RDD. While the new RDD is under construction the nodes continue to use the previous RDD. It is possible that route discoveries in progress at the time of the switch-over will be mis-routed, causing route discovery failures. The switch-over time is very short (about 0.1 second in a 200-node network), so the penalty for failed route discoveries is minor.

As nodes move, fail, and join the network the RDD topology will gradually deviate from the underlying network topology. Although AAR has mechanisms for dealing with such problems locally, it rebuilds the RDD periodically to ensure that the RDD corresponds to the network. The rebuild interval is a function of the network mobility — highly mobile networks should rebuild the RDD frequently, whereas relatively static networks should not. Because every node keeps track of its 2-hop map (as explained in Section 5.7) an upper bound on the rebuild interval is the time it takes a node to move one transmission range. This bound makes it likely that most edges in the RDD will not break before the RDD is rebuilt. Assuming an 802.11 transmission range of several hundred meters and a maximum node speed of tens of meters per second, the rebuild interval should be on the order of tens of seconds.

5.7 Beaconing

Nodes use a beaconing mechanism to maintain a 2-hop map of its neighbors and their neighbors. Every a few seconds, each node broadcasts a `BEACON` message containing a list of its current neighbors. Every node computes its 2-hop map from the neighbor lists it receives. If the node does not receive beacons from a neighbor, it assumes the neighbor is no longer in its neighborhood and deletes it from its list of neighbors.

The `BEACON` message also includes the lowest ID and its timestamp. This helps to inform the new joining nodes about the current lowest ID in the network or update the timestamp in case a neighbor node missed the update from the leader node. This helps avoid triggering unnecessary leader elections. It is possible to perform the leader election using only the Beacon message containing the lowest ID and timestamp. Eventually the global lowest ID will be propagated through out the network and the leader is elected. However this method takes too long to converge. Given the network diameter of d and the beacon interval of t_B, the time to elect a leader is $d \times t_B$. For big networks, the time can be long and may not keep up with network changes. For this reason, AAR relies on flooding the `LOWID` message for leader election and the `CONSTRUCT` message to refresh the timestamp of the lowest ID.

The 2-hop map built from the beaconing effectively solves the broken link problem in the RDD. As long as the two endpoints of an edge remain within 2-hop range of each other, a detour can be found between them using the 2-hop maps.

The 2-hop map is also useful for speeding up route discovery. When a node receives a ROUTE message it checks its 2-hop map for the destination. If found, the message can be forwarded directly to the destination, bypassing the RDD. The 2-hop map is also useful for route shortening and repair, as discussed in Section 5.9.

5.8 Route Discovery

Route discovery is a search along the edges in the RDD by a ROUTE message that originates at the source node. When a node receives a ROUTE, it first checks its 2-hop map for the destination, and if found sends it the request. Otherwise it checks for the destination address in the Address Sets of its neighbors, and if matches are found, sends the search to the neighbor that matches and is farthest from the root. Although sending the search to any neighbor with a matching Address Set will discover a route, sending it to the neighbor farthest from the root results in a shorter route and offloads the nodes near the root of the RDD.

The ROUTE message contains the search path of the nodes visited so far. When the ROUTE message arrives at the destination node, the search path is returned to the source node, which adds it to its route cache and uses it as the route for data transmission.

Figure 5.2 shows an example route discovery. Node `S` is the source and node `T` is the destination.

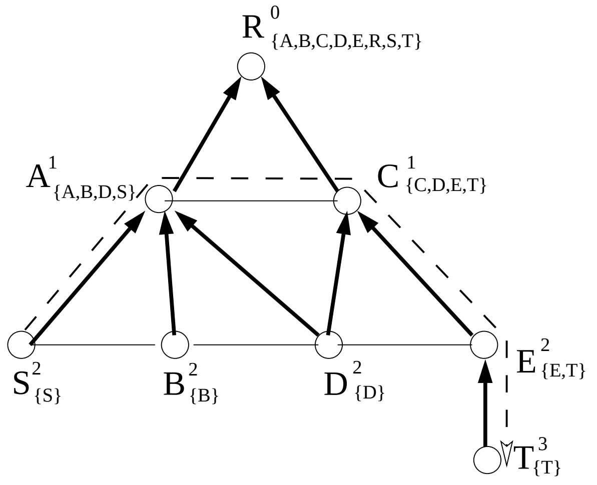


Figure 5.2: Route Discovery. The bold lines are RDD edges. The solid lines are network links not in the RDD. The dashed line shows the search path for the route discovery. Each node is labeled with its distance from the root and its Address Set.

The route discovery in RDD follows 2 phases: SEARCH and MATCH. In the SEARCH phase, the ROUTE message follows the RDD edges until it arrives at a node with the matching address set. In the MATCH phase, the ROUTE message follows the RDD edges in reverse visiting nodes with matching address sets until it arrives at the destination. If the source node finds a match in its address set, the route discovery skips the SEARCH phase. The 2-phase search is a simple way to prevent routing loops — a node can not be visited in both SEARCH and MATCH phases because it is visited in the MATCH phase only if its address set matches, and if that is the case, the search would have entered the MATCH phase when the node was visited the first time.

RDD edges may break due to changes in the network topology. In such cases, AAR uses the 2-hop maps to detour around the broken edge. In Figure 5.3, the RDD edge between node C and node E breaks, causing C to route the message to E via D. Node C sets the intended next hop in the message to E and sends the message to D. When node D receives the message and discovers it is not the intended next hop, it forwards the message to E. Node D also adds itself to the search path in the message. It is also possible for the link between node D and E to break causing the forward from D to E to fail. If so, node D sends the message back to C. Node C updates its 2-hop map and computes another detour. If one is not found, node C sends an error message back to S.

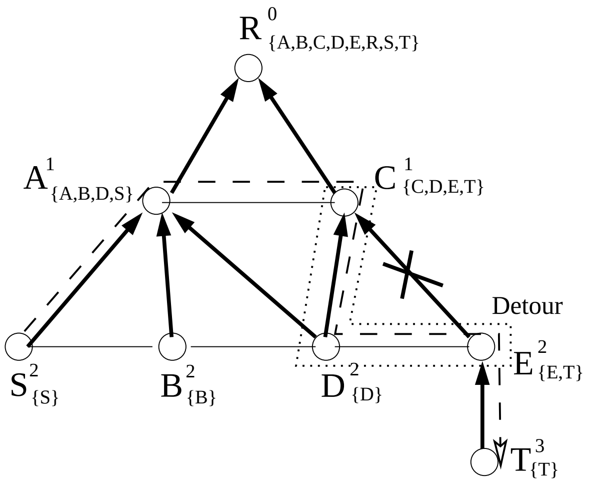


Figure 5.3: RDD Detours. The link between nodes C and E is broken, so C routes the message to E via D.

The fields in the ROUTE message are summarized in Table 5.1, and the RDD state stored on each node in Table 5.2.

Figure 5.4 describes how a node processes a ROUTE message. The argument

Field	*Meaning*
$M.source$	Source address
$M.dest$	Destination address
$M.phase$	Search phase
$M.next$	Intended next hop
$M.hops$	Number of hops traveled
$M.path$	Search path

Table 5.1: ROUTE Message Contents. The *phase* field is either SEARCH or MATCH. The *next* field contains the intended next hop. The *path* field contains the nodes visited so far during route discovery.

Field	*Meaning*
H_x:	Distance from node x to root (hops)
A_x:	Address set of node x
U_x:	Up group of node x
D_x:	Down group of node x
N_x:	Neighbor set of node x
L_x:	2-hop map of node x
A_y:	Address Set of neighbor y ($y \in N_x$)

Table 5.2: RDD Node State.

x is the node processing the message, M is the message, and $from$ is the node from which node x received the message.

Route discovery may fail for several reasons. First, the destination address may not appear in the RDD. This can happen due to wireless broadcast unreliability and node mobility. Second, a broken edge in the RDD could not be bypassed by detouring. Third, the Bloom Filters used to encode Address Sets can have false positives, causing the route discovery message to go astray and route discovery to fail.

Should route discovery fail an error message is sent back to the source node. The question is whether or not the source node should retry the route discovery using the RDD. Our experiments show that if a route discovery using the RDD fails the first time, it is highly likely to fail a second time. On the other hand, over 80% of the initial route discoveries using the RDD succeed, so failures are relatively rare. AAR therefore responds to a route discovery failure by performing route discovery

RDDSearch(x, M, $from$)

```
if $x = M.dest$ then
  Send route reply to $M.source$ along $M.path$
  Return
end if
Add $x$ to $M.path$, increase $M.hops$
if $M.next \neq x$ then
  Forward $M$ to $M.next$
  Return
end if
if $M.dest \in L_x$ then
  Find a path $p$ to $M.dest$ in $L_x$
  Send $M$ to $M.dest$ along $p$
  Return
end if
if $M.phase$ =SEARCH then
  if $\exists y, y \in N_x \cup D_x$, $y \in L_x$, $M.dest \in A_y$ and $H_y$ is maximum then
    $M.phase \leftarrow$ MATCH
    $nextHop \leftarrow y$
  else
    if $\exists z, z \in U_x$ and $z \in L_x$ then
      $nextHop \leftarrow z$
    else
      $nextHop \leftarrow null$
    end if
  end if
else
  if $\exists y, y \in D_x$ and $y \in L_x$ and $M.dest \in A_y$ then
    $nextHop \leftarrow y$
  else
    $nextHop \leftarrow null$
  end if
end if
if $nextHop = null$ then
  Send search fail to $M.source$
else
  $M.next \leftarrow nextHop$
  if $nextHop \in N_x$ then
    Send $M$ to $nextHop$
  else
    $t \leftarrow$ detouring node in $N_x$ to $nextHop$
    Send $M$ to $t$
  end if
end if
```

Figure 5.4: Route Discovery Algorithm.

using an optimized flood. In this way, routes can always be found assuming the network is connected.

The optimized flood in AAR is similar to the Multipoint Relays (MPR) flooding in OSLR [21]. The idea is to use the local 2-hop map when forwarding the search message. The forwarding node v specifies in the search message a set of neighbor nodes that should relay this message. The set of neighbors are selected so that all nodes in the 2-hop range are covered (the node from which v get the message and its neighbors are excluded). Only those neighbors specified in the message forward the message. The use of an optimized flood allows AAR to recover from failed RDD route discoveries without significantly increasing overhead.

One possible optimization for route discovery is to allow intermediate nodes in the search path return routes when they find routes to the destination in their routing caches. Although this seems reasonable, AAR does not use this optimization because the routes stored in the intermediate nodes can be stale and using them causes extra data packet drops.

5.9 Route Shortening and Repair

The route returned by route discovery in AAR is likely longer than the shortest path between the source and destination since it follows the edges in the RDD. These sub-optimal routes increase the number of hops required to deliver a packet, increasing delay, traffic, and power consumption. They also increase traffic on the nodes near the center of the RDD. The root node is of particular concern since it will be overwhelmed if all routes pass through it.

To mitigate these effects, AAR shortens routes using the 2-hop maps. As a REPLY message follows the search path back to the source, each node along the path checks its 2-hop map and follows short-cuts to shorten the path. The route in the reply message is modified to reflect the short-cut, causing the source to use the shorter path for subsequent data packets.

Various path shortening techniques have been proposed for MANET routing, such as SHORT [30] and PCA [29]. The basic idea is to have nodes promiscuously hear neighborhood traffic and notify the sending node of short-cuts. These techniques require promiscuous mode and need extra messages to notify the shortened routes. Compared with existing techniques, AAR's path shortening mechanism is effective, yet requires neither promiscuous mode nor extra messages.

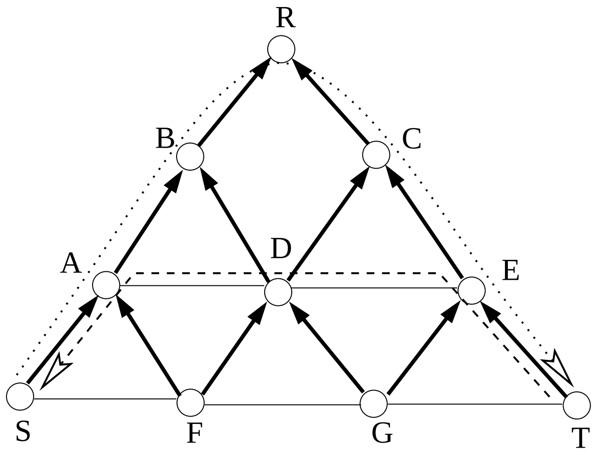


Figure 5.5: Route Shortening. The dotted line shows the search path of the route discovery. The dashed line shows the route after shortening using the 2-hop maps.

Figure 5.5 shows an example of route shortening. The source node is node S, the destination node is node T, and the root node is node R. When the route reply arrives at node E, it decides that it can bypass nodes C, R, and B via a short-cut through node D to A. The route is thus shortened by 2 hops. The source node S uses the route "S-A-D-E-T" instead of the search path "S-A-B-R-C-E-T" to send data. The route shortening not only shortens the routes, but also alleviates the traffic load on the nodes close to the root.

Even with route shortening AAR can produce sub-optimal routes. In Figure 5.5, the path "S-F-G-T" is 1 hop shorter than the route AAR finds. However, this path cannot be found using 2-hop maps. Our results show that the sub-optimal routes it produces are more than offset by AAR's lower route discovery cost.

The 2-hop map is also used to salvage data and repair a route when a data packet encounters a broken link. Before dropping the packet, the node checks its 2-hop map to find a detour around the broken link. If a detour is found, it is added to the route and the data packet is forwarded via the detouring node. When packet arrives at the destination, the destination node sends back a REPLY to the source so that the source replaces the broken route with the repaired one. The new route is also subjected to shortening when returned to the source node. Our results show that this route repair technique effectively reduces data packet losses, leading to a loss rate significantly lower than existing protocols.

5.10 Packet Forwarding and Route Maintenance

AAR is similar to DSR [43] in how it handles packet forwarding and route maintenance. AAR using source routing on the data packets; the ROUTE message records the search path and the REPLY message sends the optimized path to the source node as the route. The source node stores the route in its route cache.

When the REPLY message is sent back to the source node, all the intermediate nodes also process the message, learning the routes to the other nodes along the path and storing them to their routing cache. The organization of routing cache in AAR is similar to that being used in DSR. Each entry of the routing cache records the destination address and the full route to the destination. AAR has an expiration time for each cache entry. Each time the cache entry is accessed, its expiration time is extended. Expired cache entries are removed from the route cache.

When determining route to a destination node, a node first checks its own route cache. If the route to the destination address is available in the cache, it gets the route and put the full route into the IP header as an IP option. The intermediate hops simply forward the packet to the next hop along the route. If the route to the destination node can not be found in the routing cache, the source node initiates the route discovery process. Before the route is discovered, the data packets are temporarily stored in a sending buffer. Whenever a new route is found, the sending buffer is scanned and those data packets to the same destination are sent along the route.

Whenever there is a link break along the route, an ERROR message is generated by the node that encountered the broken link and sent back to the source node. The ERROR message contains the broken link and the return path to the source node. The message is forwarded along the return path. At each hop, the node checks its routing cache and deletes routes that contain the broken link.

DSR uses promiscuous mode to learn more routes and reduce route discoveries. AAR does not do so, primarily because promiscuous mode is not always possible, but also because it requires that every node receive and process all messages sent in its neighborhood, even it is not the recipient. This adds to the nodes' processing load and power consumption. Our results show that AAR out-performs DSR without resorting to promiscuous mode so AAR does not use it.

5.11 Evaluation

5.11.1 Experimental Setup

We simulated AAR on GlomoSim 2.0.3 [7], a simulator for mobile ad-hoc networks. We choose GlomoSim because of its speed and ability to simulate large networks. The experiments first compare AAR with popular MANET routing protocols including AODV and DSR; then justify some design choices in AAR with experimental results. As per the suggestions by Bai [6], we disabled route replies from intermediate nodes in DSR to improve its performance. For large networks and long simulations the intermediate nodes tend to have stale routes that cause excessive data packet losses if they are allowed to reply to route requests. We also modified DSR so that the destination responds to only the first route request it receives, improving performance when network traffic is high. We also fixed a few bugs we found in the AODV and DSR implementations. All three protocols use a maximum route length 64, which is more than enough for the maximum network size simulated (400 nodes). AAR uses 512-byte Bloom Filters, a 10-second RDD rebuild interval, and a 2-second beacon interval.

The MAC layer protocol is 802.11. The channel bit rate is 11Mbps, and the wireless transmission range is 250 meters. The network nodes are initially randomly distributed in a square space. The mobility model follows the Random-Way Point model [43], in which each node randomly selects a destination and travels there in a straight line at a constant, randomly-selected speed. The maximum speed is varied in the experiments, but the pause time once a node reaches a destination and before it moves to a new destination is always zero. The network queue length is set to 100 packets per priority. All experiments simulate the network activity for 900 seconds (15 minutes). AAR starts with an RDD already constructed as this represents the steady-state (the RDD is rebuilt periodically during the experiment).

The traffic model is Constant Bit Rate (CBR) [8]. Each flow lasts for the entire experiment, which means that any route discoveries performed after those at the beginning of the experiment are caused by route breaks (and not new flows). We follow a randomized traffic pattern that has been commonly used [23, 13, 15]: the source nodes and destination nodes are randomly selected but are not the same node. For each flow, the data packet size is 512 bytes and the packet rate is 2 packets/second. The number of flows is varied to simulate different traffic loads.

The performance metrics include the data packet loss rate, end-to-end delay, and the normalized packet overhead. The data packet loss rate (loss rate) is the ratio of the number of application data packets not received by the destinations to the total number of application data packets sent out from the sources. The end-to-end delay is the average end-to-end delay of all the application data packets received by the destinations. The data packet loss rate and end-to-end delay are measured end-to-end at the application layer and indicate the communication quality from the end user's perspective.

We use normalized packet overhead, which is the number of packets sent to the MAC layer per node per second, to represent protocol efficiency. We use this metric rather than simply counting control packets because while different routing protocols may send different numbers of control packets, they also create routes of different lengths. This makes comparing control packet counts while ignoring the cost of sending data on these different routes an inaccurate measure of protocol efficiency. The normalized packet overhead is a better measure. Counting packets instead of bytes is more commonly used for evaluating MANE's performance [13, 23] as the cost of medium for packet transmission is more expensive than incrementally adding bytes to packets. On the other hand, in AAR all data packets have the same size as those in DSR; the route discovery packets have the same size as those in DSR; the beacon messages are of small size; the Bloom filter messages have larger size but are sent in low frequency (one per tens of seconds). Taking all above into consideration, comparing bytes will have similar results as comparing packets.

The simulation does not measure and compare memory consumption. AAR must store its neighbors' and its own Bloom filters, as well as the 2-hop map. Assume the network size is N and the nodes have B neighbors on average, the size of additional storage is $\mathrm{O}(BN + B^2)$. However, considering that for the normal cases that N is in the magnitude of hundreds to thousands, while B is in the order of tens, the additional memory consumption is normally only tens of kilobytes, which is not a big requirement for modern computers with hundreds of megabytes memory. Improvement of communication performance is more important than memory savings for wireless network systems.

Routing protocol performance is affected by many parameters, such as data traffic load, network size, node mobility, and network density. Our experiments vary these parameters to test the performance of AAR and compare it with AODV and DSR: data traffic load is varied to test the protocols for different loads; network size is varied to test scalability; maximum node speed is varied to measure the effect of different node mobility; and node density is varied to test the protocols for dense and sparse networks. When varying one parameter, the others are set to default values (Table 5.3). For data traffic load the default value is a constant ratio of CBR flows to network size. This is more reasonable than simply fixing the number of CBR flows because traffic load increase as the number of nodes increase (it seems unlikely that the additional nodes would not generate additional traffic). The default traffic load is 20% of the network size, e.g. 20 flows for a 100-node network.

Node density is the average number of nodes per transmission range. We vary the edge length of the square network area to achieve the desired average node density.

For each configuration we randomly generate 7 networks, run the experiments and take the average of the results. Thus AODV, DSR and AAR are simulated and evaluated on exactly the same networks and with the same CBR flows.

Parameter	*Range*	*Default*
Network size	50 – 400 nodes	200 nodes
CBR flows	10 – 100 flows	20% of nodes
Max speed	1 – 20 m/s	10 m/s
Node density	10 – 30 nodes/range	15 nodes/range
RDD interval	NA	10 seconds

Table 5.3: Simulation Parameters. The *Range* column gives the range of values when that parameter is varied. The *Default* column gives the fixed value of each parameter when varying other parameters.

5.11.2 Varying Traffic Load

This experiment varies the number of CBR flows from 10 to 100, while fixing the other parameters to the values in Table 5.3. Figure 5.6(a) shows that AAR has slightly higher overhead than AODV and DSR when there are only a few flows. This is because the cost of building the RDD and maintaining the 2-hop maps outweighs the benefits of reducing the route discovery cost in a network where there is little data traffic. However, for moderate to high traffic load, AAR has significantly less overhead. For example when the number of flows is 60, AAR's overhead is only 57% of DSR's and 55% of AODV's.

When the traffic load is high (100 flows), AODV easily gets congested, leading to excessive packet overhead, loss rate, and end-to-end delay. AAR and DSR are much less congested for the same traffic load. With 100 CBR flows, AODV has normalized packet overhead of 48.3 packets/node/sec., loss rate of 96%, and end-to-end delay of 6774 ms, as compared to 12.8 packets/node/sec., 17.9%, and 492 ms for DSR, and 7.9 packets/node/sec., 14.9%, and 671 ms for AAR.

Of the three, AAR has the lowest loss rate. When the number of CBR flows is less than 80, AAR's loss rate is less than 2.5%, while DSR's ranges from 4.2% to 6.1% and AODV's from 4.7% to 26.1%. AAR loses few data packets due to its effective route repair mechanism. Whenever a data packet encounters a broken link, AAR uses the 2-hop map to detour around it, if possible.

AAR has end-to-end delay slightly less than AODV and DSR when the number of flows is less than 80. When the number of CBR flows is 80, AAR's delay is 90.7 ms, compared to 94 ms for DSR and 244.8 ms for AODV. AAR's shorter delay is due to its lower packet overhead and its route shortening mechanism. However when the number of CBR flows is 100, AAR's delay is 671 ms which is higher than DSR' 492 ms, but still much lower than AODV's 6774 ms. There are two reasons for this. First, AAR has a lower loss rate meaning that it delivers some packets that DSR simply drops; these packets incur a higher-than-average delay to deliver and therefore increase the average delay. Second, the traffic distribution in AAR is less uniform than DSR because its routes tend to follow the edges in the RDD. The higher load on some nodes causes longer queuing delays and increases the end-to-end delay. This effect becomes noticeable at high traffic loads.

5.11.3 Varying Network Size

In this experiment the network size is varied from 50 to 400 nodes. The results are shown in Figure 5.7.

The overhead in AODV and DSR increases faster than linearly as the network size increases, while AAR's overhead has close to linear growth and with a much lower slope. When the network size is less than 100 nodes, AAR has higher cost than AODV and DSR; for example, when the network size is 50 nodes, the normalized packet overhead in AAR is 1.8 packets/node/sec., while DSR and AODV have only 1.2 packets/node/sec. This is because flooding small networks is relatively cheap, hence the lower route discovery cost in AAR does not offset the extra cost of the maintaining the RDD and the 2-hop maps. However, AAR's overhead grows at a smaller rate than AODV and DSR. When the network has more than 200 nodes, the overhead of AAR is significantly lower than the other protocols. For instance, when the network has 400 nodes, the normalized packet overhead in AAR is only 38% of AODV and 40% of DSR.

AAR also has the lowest loss rate among the three protocols. AAR's maximum loss rate is 2.5%, while AODV and DSR reach 14.9% and 13.0%, respectively. AAR's rate increases less as the network scales; for a 400-node network the data loss rate in AAR is only 17.0% of AODV and 19.5% of DSR.

Figure 5.7(c) shows that the end-to-end delay in all the three protocols increases with the network size, but AAR increases at the slowest rate. For small networks, AAR has an end-to-end delay slightly higher than AODV and DSR; for a network of 50 nodes AAR has 5.0 ms end-to-end delay while AODV and DSR have 4.3 ms and 5.0 ms, respectively. However, when the network has 400 nodes, AAR's end-to-end delay is 107.9 ms, 67% of AODV's 161.5 ms and 51% of DSR's 210.2 ms.

In summary, AAR supports large networks better than AODV and DSR. The packet overhead and loss rate in AODV and DSR grow much faster than AAR as network size increases. Although AAR is slightly worse than AODV and DSR for networks with tens of nodes, it is much better for networks with hundreds of nodes.

5.11.4 Varying Mobility

In this experiment the maximum node speed varies from 1 m/s to 20 m/s. Figure 5.8(a) shows that the overhead in AODV and DSR increases with node speed, while AAR's overhead is relatively constant. When the node speed is low, AAR has higher overhead than AODV and DSR. Few routes break, so that few route discoveries are performed, and hence AAR's lower route discovery costs do not offset the cost of maintaining the RDD and 2-hop maps. For example, when the maximum node speed is 1 m/s, AAR has normalized packet overhead of 3.1 packets/node/sec., while AODV and DSR have 2.4 and 2.5 packets/node/sec., respectively. However, when nodes are more mobile, routes break more frequently causing more route discoveries. The flooding AODV and DSR use for route discovery is much more costly than searching the RDD in AAR, leading to higher overhead. In addition, the route repair mechanism in AAR helps avoid route discoveries and thereby reduces overhead. When the maximum node speed is 20 m/s, the overhead in AAR is 3.4 packets/node/sec., only 53% of AODV (6.4 packets/node/sec.) and 49% of DSR

(7.0 packets/node/sec.).

The loss rate in AAR is much lower than AODV and DSR for all speeds measured. The loss rate in AAR is 0.06% when the maximum node speed is 1 m/s, as compared to 0.67% for AODV and 0.52% for DSR. When the maximum node speed is 20m/s, AAR has loss rate of 0.81%, while AODV and DSR experience 8.31% and 7.19%, respectively. The results indicate that AAR's route repair mechanism effectively handles link breaks even when the mobility is high.

All three protocols have comparable end-to-end delay when the node mobility is low. When the maximum node speed is 1m/s, AAR has end-to-end delay of 9.96 ms, while AODV and DSR have 9.64 ms and 10.15 ms, respectively. However, when the maximum node speed increases, the end-to-end delay of AAR is noticeably lower than AODV and DSR, due to its lower overhead. Lower overhead means fewer control packets, which shortens queue lengths and queuing times on each node, and hence reduces the end-to-end delay. When the maximum node speed is 20 m/s, the end-to-end delay in AAR is 21.71 ms, 17% less than AODV (26.70 ms) and 27% less than DSR (29.81 ms).

Overall the experiment shows that AAR handles mobility well. Its performance is comparable to AODV and DSR when mobility is low, but for networks with moderate or high mobility, AAR has better performance, with less overhead, loss rate, and end-to-end delay.

5.11.5 Varying Node Density

In this experiment the average node density is varied from 10 nodes per transmission range to 30 nodes per transmission range. The results are shown in Figure 5.9. When the node density increases, the normalized packet overhead, the loss rate, and the end-to-end delay decrease for all three protocols. The reason is that the network diameter decreases as the node density increases, for a fixed-size network. A smaller network diameter means shorter routes, which in turn means fewer transmissions required to deliver a data packet, resulting in less overhead and delay. In addition, shorter routes also means fewer route breaks, smaller data packet loss rate, and fewer route discoveries.

Figure 5.9(a) shows when the node density is low, AAR has much lower packet overhead than AODV and DSR. For example, when the node density is 10 nodes per transmission range, AAR has normalized packet overhead of 3.87 packets/node/sec., only 59.2% of AODV (6.54 packets/node/sec.) and 58.7% of DSR (6.59 packets/node/sec.). However, as the node density increases the difference between AAR, DSR and AODV decreases. When the node density is 30 nodes per transmission range, AAR has normalized packet overhead of 2.53 packets/node/sec. as compared to 2.92 packets/node/sec. and 3.30 packets/node/sec. for AODV and DSR, respectively. AAR's savings from more efficient route discoveries has less effect because the shorter routes cause fewer route breaks and route discoveries.

AAR has a lower loss rate than AODV and DSR for all node densities. For instance, with 10 nodes per transmission range, the loss rate in AAR is 1.96%, compared to 7.51% in AODV and 6.60% in DSR. The loss rate decreases when the node density increases. For node density of 30 nodes per transmission range, the loss rate in AAR is 0.04%, while AODV is 3.22% and DSR is 2.59%. Again, AAR is much lower than the other protocols.

The end-to-end delay in AAR is also lower than AODV and DSR for all node densities. When the node density is 10 nodes per transmission range, the end-to-end delay in AAR is 27.4 ms, 11% lower than AODV (30.8 ms) and 13% lower than DSR (31.6 ms). When the node density increases to 30 nodes per transmission range, the end-to-end delay AAR is 10.0 ms, 5% lower than AODV (10.5 ms) and 31% lower than DSR (14.4 ms).

Overall, AAR performs better than AODV and DSR for all node densities, although the difference decreases for dense networks.

5.11.6 Spanning Tree vs. DAG

Figures 5.10, 5.11 and 5.12 show the performance comparison between the ST and DAG design alternatives for varying traffic loads (in terms of CBR flows) on a 200-node network. The experiment settings are the same as described earlier in this section. Figure 5.10 gives the normalized packet overhead; Figure 5.11 shows the loss rate; Figure 5.12 presents the average end-to-end delay of the received application packets.

The experiments show that ST and DAG have comparable packet overheads, although the ST overhead is slightly lower because it requires fewer messages for building and maintaining the hierarchy. However, ST has higher loss rates and latencies than DAG, especially for heavy traffic loads. ST is less robust than DAG when links break, leading to a higher loss rate. ST also experiences more congestion near the root when the traffic load is high. When congestion occurs, ST has a higher message overhead than DAG.

5.11.7 2-Hop Map

Another important design choice in AAR is to maintain a 2-hop map. The 2-hop map significantly improves AAR's performance with little additional cost. AAR requires periodical beacon messages for maintaining the RDD, so encoding neighborhood information in the beacon messages imposes little overhead, but results in significant performance improvement, especially when the traffic load is high or when the network is highly mobile.

Figures 5.13, 5.14 and 5.15 show the performance comparison AAR with and without a 2-hop map, for varying traffic loads on a 200-node network. The results show that the 2-hop map significantly reduces the communication cost. The reason is that 2-hop map keeps the RDD connected so that most route queries (over 80%) succeed; while without 2-hop map the success rate is much lower (less than 40%). The RDD query failure causes flooding and significantly increases the cost. The 2-hop map also helps to maintain existing routes when links break, eliminating route discoveries.

On the other hand, the 2-hop map also dramatically reduces the loss rate and latency, especially when the traffic is heavy. In AAR, the 2-hop map is used to salvage the packets that will be dropped otherwise. The route optimization based on the 2-hop map also helps to reduce the packet transmission latency as well as balance the traffic load. This effect is hardly noticeable when the traffic load is light but becomes more obvious as the load increases.

Figure 5.16, 5.17 and 5.18 show the performance comparison for varying network mobility (measured as nodes' moving speed). AAR with the 2-hop map has packet overhead relatively stable regardless network mobility; while without the 2-hop map the packet overhead increases significantly when mobility increases. Mobility causes link breaks on both RDD and existing routes. The 2-hop map well handles both cases, thus significantly reduces the cost and keeps the packet overhead stable against mobility. The 2-hop map also helps to achieve low packet loss rate and end-to-end delay, as the 2-hop map salvages packets, shortens the route and balances the traffic load.

5.12 Conclusion and Future Work

AAR is unique in that it proactively maintains a Route Discovery DAG (RDD) and uses it to reactively discover routes. Compared with flooding, search in the RDD has lower the route discovery cost, making AAR highly efficient and scalable.

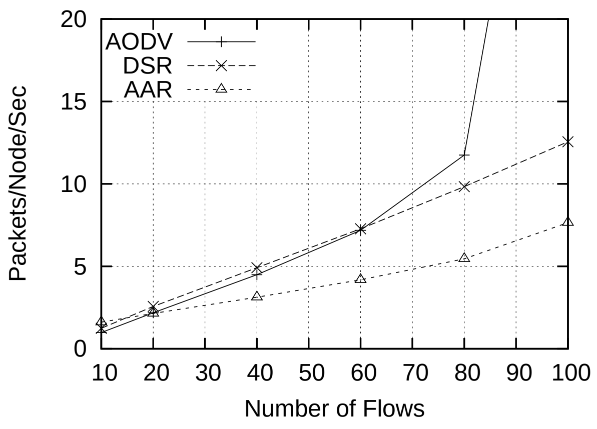
Our experimental results show AAR not only has less packet overhead than AODV and DSR, but also has a lower loss rate and lower latency in most cases. It works well for a wide range of traffic loads; it is scalable to large networks; it handles mobile nodes well; and it performs well for both dense and sparse networks. Overall, AAR is an efficient and reliable routing protocol with low communication cost, low loss rate, and low end-to-end delay.

We are currently working to improve several aspects of AAR. First, the current implementation uses the leader as the root of the RDD, which is unlikely to be the best choice. The leader may not be in the center of the network, and choosing the same node as the RDD root repeatedly increases its traffic load and power consumption. Having the leader choose a node in the center of the network to be the root is not difficult and we plan to implement it.

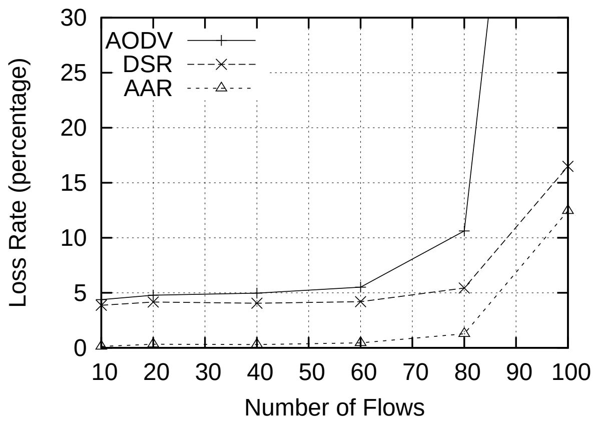
Second, AAR uses the route discovery paths as the routes. This places higher traffic loads on the nodes near the root of the RDD. Although this is not an issue for the range of traffic loads we measured, networks with very high loads may congest the central RDD nodes. We are investigating methods for balancing loads across all nodes.

Third, the current AAR implementation uses a fixed interval for rebuilding the RDD. We are looking at ways to make the RDD interval adaptive, so that the RDD would be rebuilt frequently in highly-mobile networks and infrequently in relatively static networks.

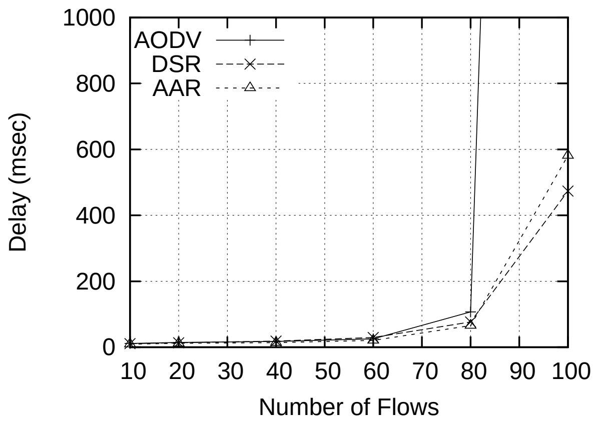
Even without these improvements AAR offers significant advantages over AODV and DSR, particularly in highly-mobile and large-scale networks. Constructing the RDD proactively and performing route discovery reactively offers the best of both worlds, reducing packet overhead, delivering more packets, and improving end-to-end delay.



(a) Normalized Packet Overhead

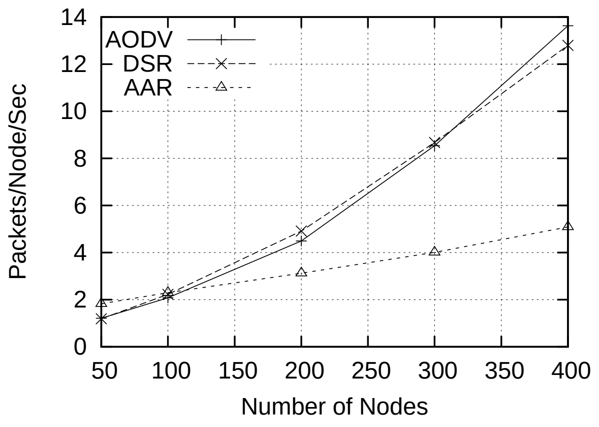


(b) Data Packet Loss Rate

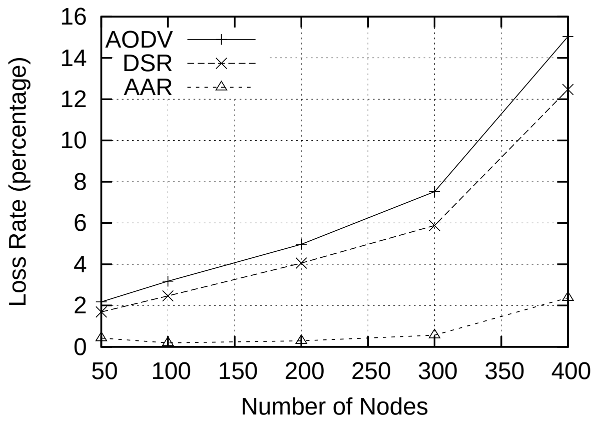


(c) End-to-end Delay

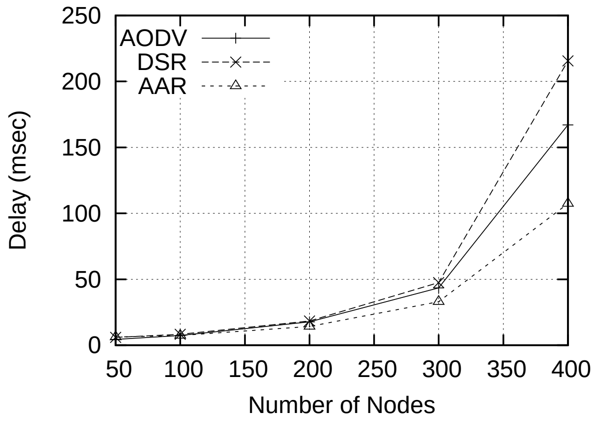
Figure 5.6: Varying Traffic Load.



(a) Normalized Packet Overhead

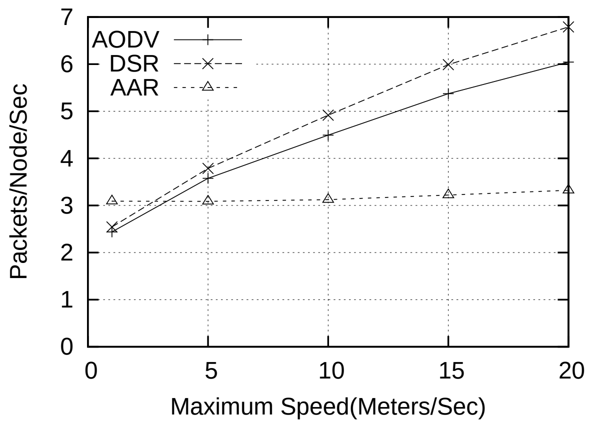


(b) Data Packet Loss Rate

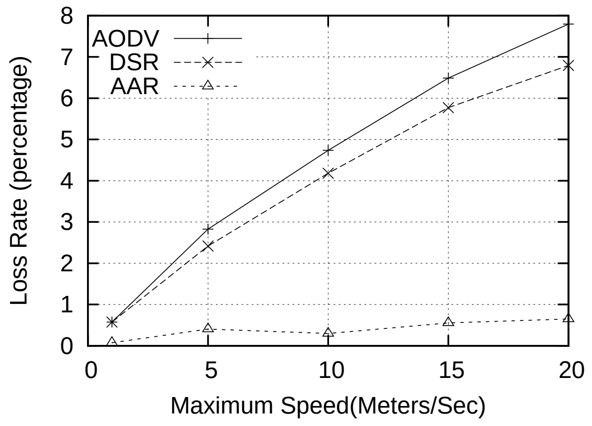


(c) End-to-end Delay

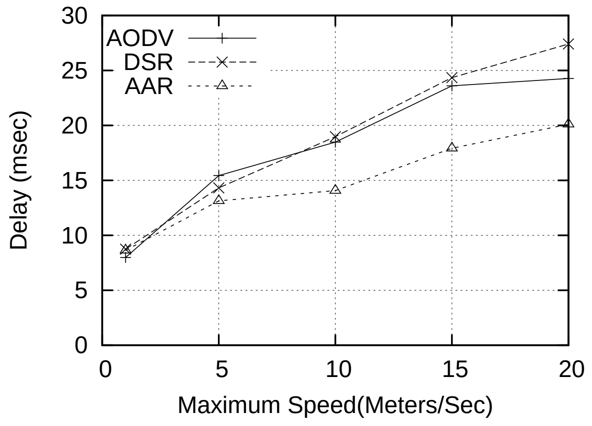
Figure 5.7: Varying Network Size.



(a) Normalized Packet Overhead

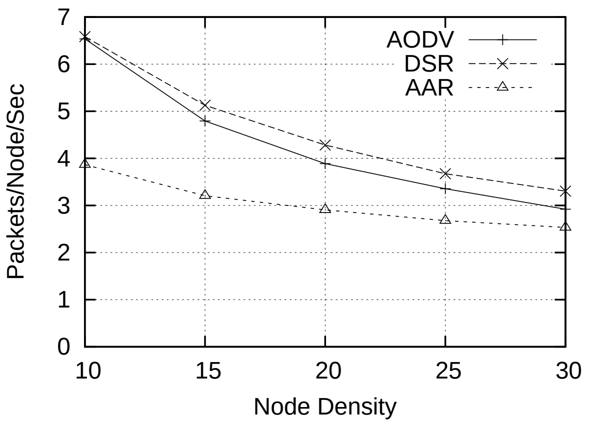


(b) Data Packet Loss Rate

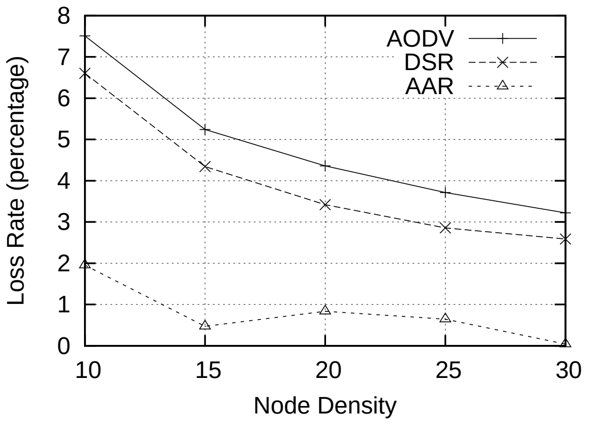


(c) End-to-end Delay

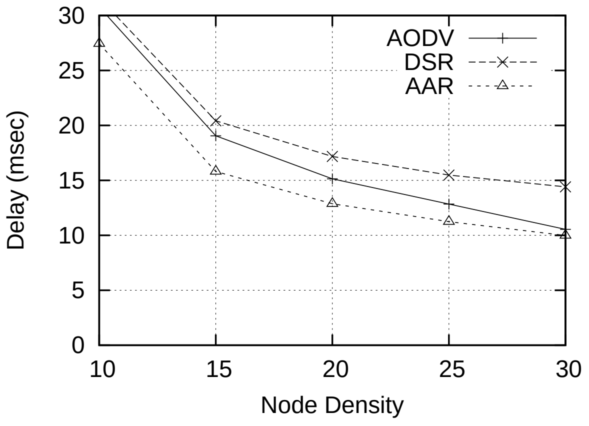
Figure 5.8: Varying Mobility



(a) Normalized Packet Overhead



(b) Data Packet Loss Rate



(c) End-to-end Delay

Figure 5.9: Varying Node Density.

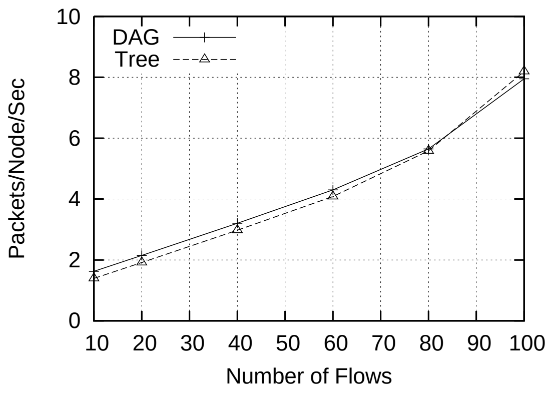


Figure 5.10: Tree versus DAG: Packet Overhead

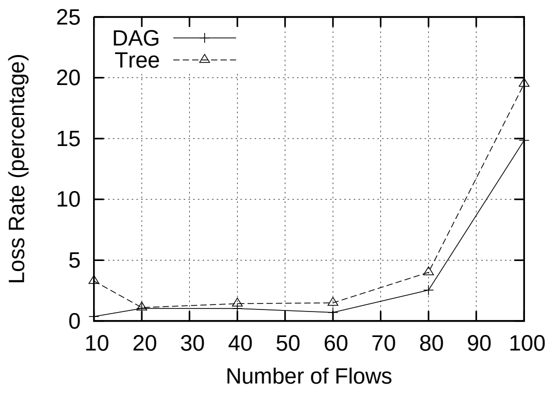


Figure 5.11: Tree versus DAG: Loss Rate

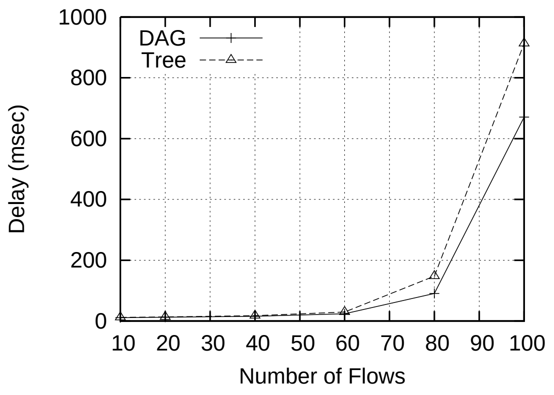


Figure 5.12: TREE versus DAG: Latency

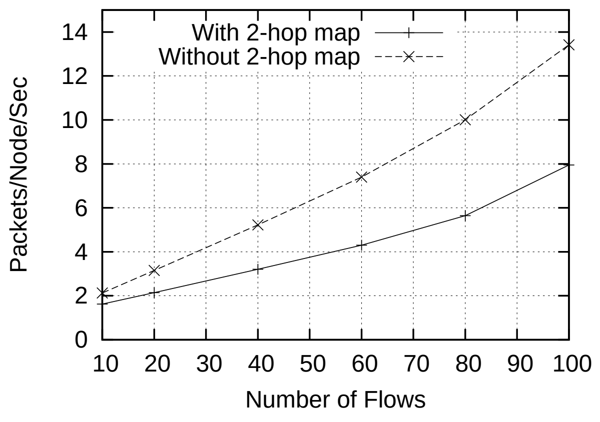


Figure 5.13: Packet overhead for varying traffic loads

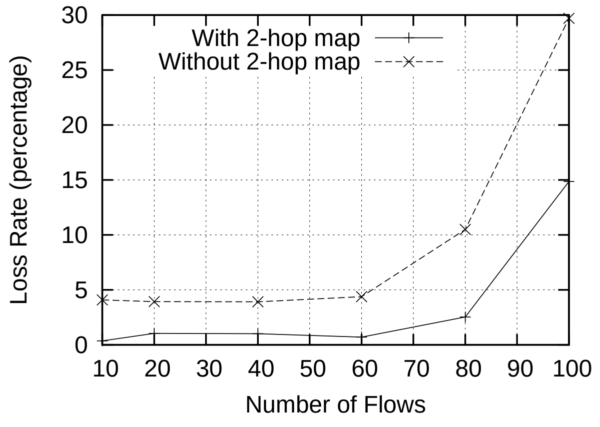


Figure 5.14: Packet loss rate for varying traffic loads

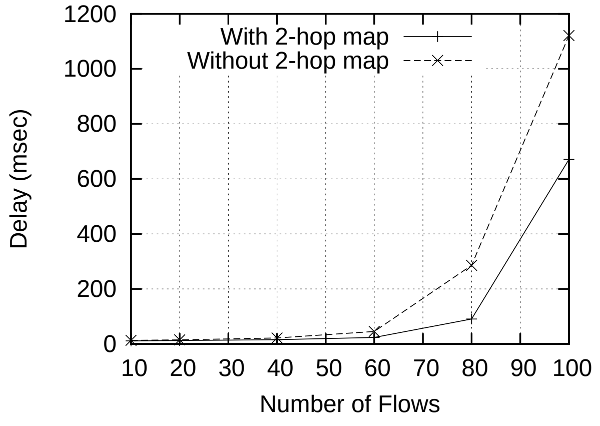


Figure 5.15: End-to-end delay for varying traffic loads

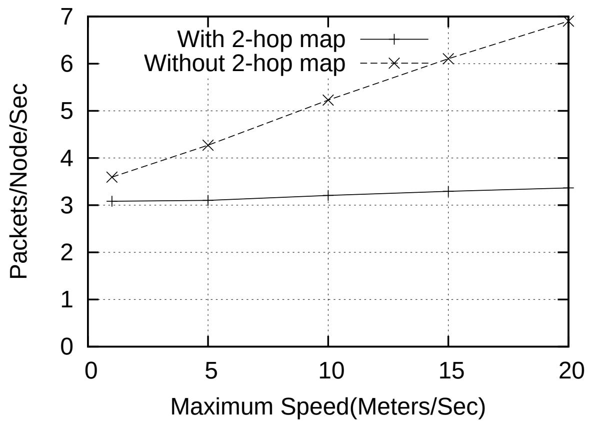


Figure 5.16: Packet overhead for varying mobility

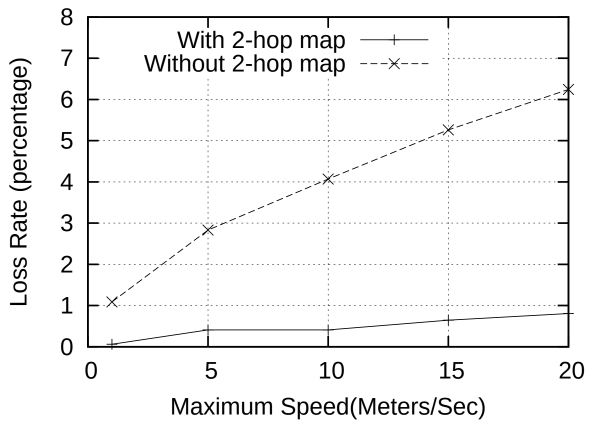


Figure 5.17: Packet loss rate for varying mobility

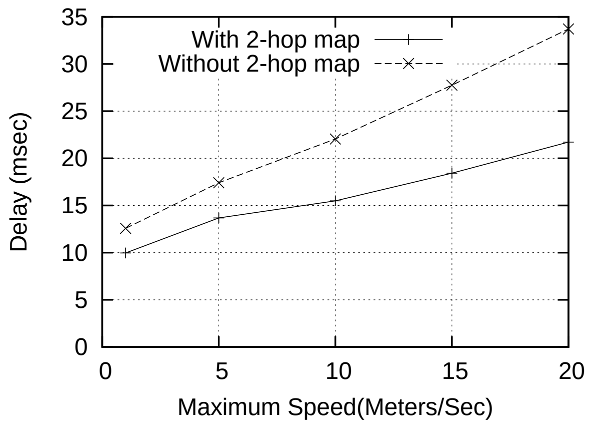


Figure 5.18: End-to-end delay for varying mobility

CHAPTER 6

CONCLUSIONS

This dissertation makes contributions in the areas of Mobile Ad hoc Networks (MANETs) routing and Wireless Sensor Networks (WSNs) routing. Traditional routing algorithms/protocols for these networks rely on flooding search (and random walk for WSNs) that are neither efficient nor scalable, or special positioning equipment that limits the application. This dissertation proposes new solutions that avoid or minimize flooding and do not rely on special equipment. The experimental results show that the new solutions outperform the traditional ones in either message overhead, delivery rate or end-to-end delay. This chapter has two sections: Section 6.1 summarizes the contributions; section 6.2 discusses future directions of the work.

6.1 Contributions

This dissertation first studies the routing problems in Wireless Sensor Networks (WSNs) and explores the unique features of WSN routing, including data–centric routing, duplicate data storage and query–oriented processing. Two novel algorithms are proposed to cope with the data centric nature of WSN routing. Compared with existing solutions, the new algorithms are either more efficient in message overhead, or more robust against the node mobility.

The *Spiral* algorithm is proposed for short-term data-centric routing in static WSNs. In this scenario both the route discovery cost and route length are important. Spiral uses a walk–based mechanism to reduce the communication cost of route discovery. The walk follows a spiral–like search path that visits nodes near the source before more distant nodes. This is particularly beneficial when there are multiple copies of the desired data, as Spiral will tend to find a close one first and return a route close to optimal. Spiral algorithm discovers close-to-optimal route to the desired data object at the cost far less than flooding, achieving good balance between the route discovery cost and route length. For short–term communication of tens or hundreds of messages, Spiral algorithm is more efficient than either flooding or a random walk approach.

The second algorithm, *CNFS* (Closest Neighbor First Search) is designed for query processing in mobile wireless sensor networks, in which the communication cost and robustness are important performance metrics. CNFS is based on a directed blind search. The search is directed by topology information collected as it progresses, allowing CNFS to be both efficient and robust. CNFS uses a biased search scheme, always choosing the neighbor closest to the source node as the next hop. The partial map is also used to compute the shortest return to the source, as well as determine alternative routes when links break, helping to improve both efficiency and robustness. CNFS has a significantly higher success rate than random walk-based algorithms for all network densities and comparable to flooding in dense networks, while its message overhead is about 30% to 40% less than the other methods.

The second part of this dissertation studies the routing problem in Mobile Ad hoc Networks (MANETs) and identifies the drawbacks of the traditional routing protocols in route discovery overhead and scalability. The study further designs a novel routing protocol, *Address Aggregation-based Routing (AAR)* for MANET routing. Compared with existing routing protocols such as AODV [61] and DSR [43], AAR requires lower communication cost, loses fewer data packets, has a smaller end-to-end delay, and scales to larger networks.

Traditional MANET routing protocols relies on flooding search or positioning equipment for route discovery. AAR differs from existing protocols in that it performs route discovery reactively, but it proactively maintains an index hierarchy to make the route discovery more efficient and minimizes flooding. The index hierarchy dramatically reduces the route discovery cost, leading to lower packet overhead and better support for large networks. In addition, AAR has lower loss rate and end-to-end delay for data transmission, due to its dynamic route repair and shortening mechanism.

AAR protocol has been simulated on the Glomosim network simulator. Experiment results show that AAR saves up to 60% message overhead compared with AODV and DSR; while having lower packet loss rate and latency. AAR has better scalability than AODV and DSR when the network size increases.

6.2 Future Directions

Efficient data–centric routing is a challenging problem for sensor networks. This dissertation proposes two algorithms, Spiral and CNFS, for different application scenarios. Spiral algorithm is good for static or low mobility sensor networks and scalable to large networks; while CNFS algorithm is robust against node mobility but less scalable for large networks because the message collects and carries the topology information.

It is possible to derive a new algorithm that integrates features from both Spiral and CNFS, achieving a good tradeoff between the scalability and robustness. The basic idea is to replace the level counters in Spiral with a partial map similar to that in CNFS. The partial map contains topology information only for nodes with levels in the "level" window. The topology information is useful in identifying the unvisited nodes, bypassing the broken links and finding short paths to unvisited nodes, making the algorithm more robust against network mobility. Adding the topology information increases the message size, however since the topology is only for the portion of network that is within the "level" window, the map size is much smaller than the global map. Assuming the network size is N, the network diameter is $\sqrt{N}$. Suppose the level window contains k levels. The number of nodes with levels in the window is $O(k\sqrt{N})$, which is also the size of partial map that every message should carry. Although it is bigger than the message size of $O(k)$ in Spiral algorithm, it is smaller than the size of $O(N)$ in CNFS algorithm and is therefore more scalable.

The AAR protocol designed in this dissertation provides a good solution for the efficient route discovery problem in MANETs. The experiment results show it has less overhead, lower loss rate and lower latency than several existing popular protocols.

AAR can still be improved in several aspects. First, the current implementation uses the leader as the root of the RDD, which is unlikely to be the best choice. The leader may not be in the center of the network, and choosing the same node as the RDD root repeatedly increases its traffic load and power consumption. Having the leader choose a node in the center of the network to be the root is not difficult and we are working to implement it. Second, AAR uses the route discovery paths as the routes. This places higher traffic loads on the nodes near the root of the RDD. Although this is not an issue for the range of traffic loads we measured, networks with very high loads may congest the central RDD nodes. We are investigating methods for balancing loads across all nodes. Third, the current AAR implementation uses a fixed interval for rebuilding the RDD.We are looking at ways to make the RDD interval adaptive, so that the RDD would be rebuilt frequently in highly-mobile networks and infrequently in relatively static networks.

REFERENCES

[1] I. Abraham, D. Dolev, and D. Malkhi. ‘LLS: a locality aware location service for mobile ad hoc networks’. In *Proceedings of the Joint Workshop on Foundations of Mobile Computing (DIAL M-POMC)*, Philadelphia, USA, Oct. 2004.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications*, 40(4):102–114, 2002.

[3] A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, H. Cao, M. Sridharan, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, C. Zhang, M. Nesterenko, R. Shah, S. S. Kulkarni, M. Aramugam, L. Wang, M. G. Gouda, Y. ri Choi, D. E. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. Exscal: Elements of an extreme scale wireless sensor network. In *In Proceedings of the 11th IEEE Conference on Embedded and Real - Time Computing Systems and Applications (RTCSA)*, pages 102–108, 2005.

[4] C. Avin and C. Brito. Efficient and robust query processing in dynamic environments using random walk techniques. In *IPSN’04*.

[5] C. Avin and C. Brito. ‘Efficient and robust query processing in dynamic environments using random walk techniques’. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN)*, California, USA, Apr. 2004.

[6] R. Bai and M. Singhal. DOA: DSR over AODV routing for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(10):1403–1416, 2006.

[7] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla. Glomosim: A scalable network simulation environment. In *Technical Report 990027, UCLA Computer Science Department*, 1999.

[8] S. Basagni, M. Conti, S. Giordano, and I. S. (Editors). Mobile ad hoc networking. Wiley-IEEE Press, August 2004.

[9] D. P. Bertsekas and R. G. Gallager. Distributed asynchronous bellman-ford algorithm. In *Data Networks*, chapter 5.2.4, pages 325–333. Prentice Hall, Englewood Cliffs, 1987.

[10] N. Bisnik and A. A. Abouzeid. Optimizing random walk search algorithms in p2p networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51(6):1499–1514, April 2007.

[11] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[12] D. Braginsky and D. Estrin. 'Rumor routing algorithm for sensor networks'. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 22–31, Georgia, USA, Oct. 2002.

[13] 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 *International Conference on Mobile Computing and Networking (MobiCom'98)*, October 1998.

[14] J. Bruck, J. Gao, and A. A. Jiang. Localization and routing in sensor networks by local angle information. In *In Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 181–192, 2005.

[15] M. Caesar, M. Castro, E. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by dhts. In *ACM SIGCOMM*, September 2006.

[16] M. Caesar, M. Castro, E. B. Nightingale, G. O, and A. Rowstron. Virtual ring routing: Network routing inspired by dhts. In *Proceedings of ACM Sigcomm 2006*, Sept 2006.

[17] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *Proc. of MobiCom*, July 2001.

[18] C. Chiang, H. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks with fading channel. In *Proc. IEEE Singapore Int'l Conf. on Networks (SICON)*, April 1997.

[19] C.-Y. Chong and S. P. Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proceeding of IEEE*, 91(8), Aug 2003.

[20] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009, 2001.

[21] T. Clausen and P. Jacquet. Optimized link state routing protocol (OLSR). *RFC 3626*, Oct 2003.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (Second Edition)*, pages 581–599. The MIT Press, 2001.

[23] S. Das, C. Perkins, and E. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *INFOCOM*, 2001.

[24] B. Deb, S. Bhatnagar, and B. Nath. Reinform: Reliable information forwarding using multiple paths in sensor networks. In *In Proceedings of 28th Annual IEEE Conference on Local Computer Networks*, pages 406–415, 2003.

[25] M. Demirbas, T. Nolte, A. Arora, and N. Lynch. 'STALK: a self-stabilizing hierarchical tracking service for sensor networks'. *Technical Report OSU-CISRC-4/03-TR19 Ohio State University*, 2003.

[26] S. S. Dhillon and P. V. Mieghem. Searching with multiple random walk queries. In *In Proceedings of the 18th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2007.

[27] C. A. Duncan, S. G. Kobourov, and V. S. A. Kumar. Optimal constrained graph exploration. volume 2, pages 380–402, 2001.

[28] J. Faruque and A. Helmy. Gradient-based routing in sensor networks. *Mobile Computing and Communications Review*, 2, 2003.

[29] V. C. Giruka, M. Singhal, and S. P. Yarravarapu. A path compression technique for on-demand ad-hoc routing protocols. In *In Mobile Ad-hoc and Sensor Systems (MASS) 2004*, Oct 2004.

[30] C. Gui and P. Mohapatra. A self-healing and optimizing routing technique for ad hoc networks. In *Proceedings of ACM MobiHoc*, June 2003.

[31] Z. J. Haas, M. R. Pearlman, and P. Samar. The zone routing protocol (ZRP) for ad hoc networks, July 2002.

[32] C. Hedrick. RFC 1058: Routing information protocol, June 1988.

[33] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. In *Proc. Hawaiian Int'l Conf. on Systems Science*, January 2000.

[34] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18:2004, 2004.

[35] Q. Huang. Reliable mobicast via face-aware routing. In *In Proceedings of IEEE Infocom*, pages 325–331, 2004.

[36] Q. Huang, C. Lu, and G.-C. Romand. Spatiotemporal multicast in sensor networks. In *Proc. SenSys'03*, pages 205–217, Nov. 2003.

[37] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proc. ICDCS'02*, July 2002.

[38] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. of MobiCom 2000*, Aug. 2000.

[39] C. Intanagonwiwat, R. Govindan, and D. Estrin. 'Directed diffusion: a scalable and robust communication paradigm for sensor networks'. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (Mobicom)*, Massachusetts, USA, Aug. 2000.

[40] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen. Scalable routing strategies for ad hoc wireless networks. *IEEE Journal on Selected Areas in Communications,Special Issue on Ad-Hoc Networks*, pages 1369–1379, August 1999.

[41] P. Jacquet, A. Laouiti, P. Minet, and L. Viennot. 'Performance of multipoint relaying in ad hoc mobile routing protocols'. In *Proceedings of the Networking 2002*, pages 387–398, Pisa, Italy, 2002.

[42] M. Jiang, J. Li, and Y. Tay. Cluster-based routing protocol (CBRP). *Internet draft, draft-ietf-manet-cbrp-spec-01.txt*, Aug 1999.

[43] D. B. Johnson and D. A. Maltz. 'Dynamic source routing in ad hoc wireless networks'. *Mobile Computing(edited by T. Imielinski and H. Korth)*, 353, 1996.

[44] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of ACM MOBICOM*, August 2000.

[45] Y. Ko and N. Vaidya. Location-aided routing (LAR) mobile ad hoc networks. In *Proc. of ACM Mobicom*, Oct 1998.

[46] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[47] B. Krishnamachari. *Networking Wireless Sensors*, pages 147–156. Cambridge University Press, 2005.

[48] B. Krishnamachari, D. Estrin, and S. Wicker. Modeling data-centric routing in wireless sensor networks. In *Proc. of IEEE INFOCOM'02*, June 2002.

[49] B. Leong, B. Liskov, and R. Morris. Geographic routing without planarization. In *Proceedings of USENIX NSDI 2006*, May 2006.

[50] X. Liu, Q. Huang, and Y. Zhang. ‘Combs, needles, haystacks: balancing push and pull for discovery in large-scale sensor networks’. In *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (SenSys)*, Maryland, USA, November 2004.

[51] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *In Proceedings of the 16th ACM International Conference on Supercomputing*, 2002.

[52] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002.

[53] I.-T. L. Mario Joa-Ng. 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.

[54] J. Moy. RFC 1583: OSPF version 2, March 1994.

[55] S. Murthy and J. Garcia-Luna-Aceves. A routing protocol for packet radio networks. 1995.

[56] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *In Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162, 1999.

[57] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings IEEE INFOCOM*, April 1997.

[58] S. Pati, S. R. Das, and A. Nasipuri. ‘Serial data fusion using space-filling curves in wireless sensor networks’. In *Proceedings of The First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, California, USA, October 2004.

[59] C. Perkins, E. Belding-Royer, and S. Das. ‘RFC 3561: ad hoc on-demand distance vector (AODV) routing’, July 2003.

[60] C. E. Perkins and E. M. Royer. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proc. of ACM SIGCOMM 94*, August 1994.

[61] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proc. of 2nd IEEE Workshop on Mobile Computing Systems and Applications*, February 1999.

[62] R. Perlman. *Interconnections: Bridges and Routers.* Addison-Wesley, May 1992.

[63] V. Ramasubramanian, Z. Haas, and E. Sirer. SHARP: a hybrid adaptive routing protocol for mobile ad hoc networks. In *Proceedings of ACM MobiHoc*, June 2003.

[64] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. 'GHT: a geographic hash table for data-centric storage in sensornets'. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 78–87, Georgia, USA, Sept. 2002.

[65] N. Sadagopan, B. Krishnamachari, and A. Helmy. 'Active query forwarding in sensor networks (ACQUIRE)'. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications (SNPA)*, May 2003.

[66] G. B. Sergio D. Servetto. Constrained random walks on random graphs: routing algorithms for large scale wireless sensor networks. In *In Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[67] E. Shih, S.-H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks. In *Proc. of MobiCom'01*, July 2001.

[68] I. Stojmenovic, M. Russell, and B. Vukojevic. 'Depth first search and location based localized routing and qos routing in wireless networks'. In *Proceedings of the 29th International Conference on Parallel Processing (ICPP)*, pages 173–180, Toronto, Canada, August 2000.

[69] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman. A taxonomy of wireless micro-sensor network models. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(2):28–36, 2002.

[70] UCB/LBNL/VINT. The network simulator NS-2, 2002.

[71] J. Wu and F. Dai. A distributed formation of a virtual backbone in MANETs using adjustable transmission ranges. In *Proc. of ICDCS*, 2004.

[72] J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proc. of DIAL M'99*, 1999.

[73] O. Younis and S. Fahmy. Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In *Proceedings of IEEE INFOCOM*, March 2004.

[74] W. B. H. Z. Cheng. 'Searching strategy for multi-target discovery in wireless networks'. In *Proceedings of the Fourth Workshop on Applications and Services in Wireless Networks (ASWN)*, Massachusetts, USA, Aug. 2004.

[75] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: A library for parallel simulation of large-scale wireless networks. In *PADS98*, pages 154–161.