A Fault-Tolerant Routing Algorithm for Wireless Sensor Networks

Ibrahim Abdul Rahim

This report is submitted as partial fulfilment of the requirements for the Honours Programme of the School of Computer Science and Software Engineering, The University of Western Australia, 2004
Abstract

Wireless Sensor Networks are large and dense networks that depend on simple and limited sensors for routing. The sensors are stationary, but the topology is dynamic. The topology changes as sensors die either by running out of energy or by being destroyed. As new sensors are added, or as sensors sleep or awaken, the topology continues to change. We propose a new shortest distance routing protocol for wireless sensor networks which use link reversal routing techniques. The protocol, SDLR, correctly finds the shortest path to some base station, even as the topology changes. Partitions and changes in the network are handled using trusted neighbours. SDLR also maintains multiple-path routing similar to the link-reversal family of algorithms. Through simulation, we present the cost of running the algorithm and how effective having multiple paths for routing is on evenly distributing the load on individual sensors. We also examined the effect of node failure on randomly generated topologies which cause the network to become sparse and partitioned. We present our analysis of changes to connectivity and shortest distance as nodes within the network fail. We then analyse the cost of reconfiguring the network with respect to the above changes in topology.

Keywords: Wireless Sensor Networks, Network Protocols, Link Reversal Routing
CR Categories: C.2.2 Network Protocols, C.2.4 Distributed Systems
Acknowledgements

I would like to thank A. Prof Amitava Datta, who has guided me throughout my research, who was always willing to discuss with me and bring me a higher level of understanding.

I would like to thank my parents and sisters, who gave me support and encouragement from far away. Who would always remind me of my priorities, and tell me to take a break when I needed one. Thank you.

To all my friends in UWA and in Perth, final year would of been unbearable without you.
Contents

Abstract ii
Acknowledgements iii

1 Introduction 1
  1.1 Challenges in Wireless Sensor Networks . . . . . . . . . . . . . . . 1
  1.2 Routing Techniques . . . . . . . . . . . . . . . . . . . . . . . . . 2
  1.3 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 Wireless Sensor Networks 4
  2.1 Factors in Routing . . . . . . . . . . . . . . . . . . . . . . . . . . 4
    2.1.1 Distributed Routing . . . . . . . . . . . . . . . . . . . . . 4
    2.1.2 Reliability . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
    2.1.3 Limitation on Sensors . . . . . . . . . . . . . . . . . . . . 5
    2.1.4 Scalability . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
    2.1.5 Flooding . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
  2.2 Existing Routing Algorithms . . . . . . . . . . . . . . . . . . . . 6
    2.2.1 Link Reversal Routing . . . . . . . . . . . . . . . . . . . . 7
    2.2.2 The GB Algorithm . . . . . . . . . . . . . . . . . . . . . . 8
    2.2.3 TORA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
    2.2.4 LMR . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
    2.2.5 DSDV . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.2.6 Shortest Path Algorithm . . . . . . . . . . . . . . . . . . . 13

3 Sourced Distance Link Routing 15
  3.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
  3.2 Core Concepts . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
3.2.1 Downstream Nodes ........................................ 17
3.2.2 Shortest Distance ........................................ 17
3.2.3 Slack .................................................. 18
3.3 The Execution of the Algorithm .............................. 18
  3.3.1 Initialization ......................................... 18
  3.3.2 Recovery ............................................ 20
  3.3.3 Partitions ............................................ 21
3.4 Feature Comparisons ....................................... 22
3.5 Simulation ............................................... 23
  3.5.1 Model for Simulation ................................ 23
  3.5.2 Experiment Description ............................... 24

4 Topological Analysis ........................................ 26
  4.1 Experiment Description ................................... 26
  4.2 Connectivity ............................................ 27
  4.3 Hopcount ............................................... 29
  4.4 Downlinks .............................................. 33

5 Efficiency .................................................... 35
  5.1 Initialization cost ....................................... 35
  5.2 Message sending cost .................................. 37
  5.3 Cost of Node Failure ................................... 41
  5.4 Load on Nodes .......................................... 43

6 Conclusion .................................................. 45

A Original Honours Proposal ................................. 47
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Link Reversal</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Full Reversal Initialization</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Full Reversal Recovery</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>LMR Initialization</td>
<td>11</td>
</tr>
<tr>
<td>2.5</td>
<td>LMR Recovery</td>
<td>12</td>
</tr>
<tr>
<td>2.6</td>
<td>Shortest Distance Initialization</td>
<td>14</td>
</tr>
<tr>
<td>2.7</td>
<td>Shortest Distance Recovery</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>Load Imbalance due to Optimal Routing</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>SDLR Initialization Without Slack</td>
<td>19</td>
</tr>
<tr>
<td>3.3</td>
<td>SDLR Initialization With Slack</td>
<td>19</td>
</tr>
<tr>
<td>3.4</td>
<td>SDLR Recovery</td>
<td>20</td>
</tr>
<tr>
<td>3.5</td>
<td>SDLR Partition Detection</td>
<td>21</td>
</tr>
<tr>
<td>3.6</td>
<td>SDLR Partition Undetected</td>
<td>21</td>
</tr>
<tr>
<td>3.7</td>
<td>Sample Topology with Base Location</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Connectivity on Varying Topologies</td>
<td>28</td>
</tr>
<tr>
<td>4.2</td>
<td>Change in Connectivity due to Node Failure</td>
<td>29</td>
</tr>
<tr>
<td>4.3</td>
<td>Hop Count vs. Nodes</td>
<td>29</td>
</tr>
<tr>
<td>4.4</td>
<td>Hop Count vs. Range</td>
<td>30</td>
</tr>
<tr>
<td>4.5</td>
<td>Effect of Range on Hopcount</td>
<td>30</td>
</tr>
<tr>
<td>4.6</td>
<td>Change to Hop Count due to Node Failure</td>
<td>32</td>
</tr>
<tr>
<td>4.7</td>
<td>Downlinks on Varying Topologies</td>
<td>33</td>
</tr>
<tr>
<td>4.8</td>
<td>Downlinks vs. node failure</td>
<td>33</td>
</tr>
<tr>
<td>5.1</td>
<td>Initialization cost vs. slack</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>Initialization cost vs. nodes</td>
<td>36</td>
</tr>
<tr>
<td>5.3</td>
<td>Initialization cost vs. range</td>
<td>37</td>
</tr>
</tbody>
</table>
5.4 Cost of Message Sending vs. slack ........................................ 38
5.5 Efficiency of Message Sending ............................................ 39
5.6 Cost of Message Sending vs. nodes ..................................... 40
5.7 Cost of Message Sending vs. range ..................................... 40
5.8 Cost of Message Sending vs. number of messages ............... 41
5.9 Cost of Node Failure ....................................................... 42
5.10 Load Distribution on Nodes .............................................. 44
CHAPTER 1

Introduction

This project is an investigation into fault tolerant routing in wireless sensor networks.

The final desired outcome is to construct an algorithm that

• Considers the constraints within a wireless sensor network
• Leverages on some of the properties of a wireless sensor network
• Survives in the face of faults common to a wireless sensor network

Towards this aim, the project investigates the nature of faults and the topology of wireless sensor networks, and an alternative routing algorithm compared to those for other wireless and wired networks.

1.1 Challenges in Wireless Sensor Networks

Wireless sensor networks are large, as a large number of sensors are required to accurately collect and disseminate information about a given area. The topology could be unknown if the sensors are simply scattered as they are deployed. The network, being both large and random, makes it difficult for a global view of the network to be established.

Another complication is the dynamic nature of the topology. Nodes are dispensable. They run out of energy, they get destroyed by the environment, or they go into sleep mode. Nodes can also be awakened or added to replenish the network.

The network may also get partitioned. The failure of some nodes will cause other nodes to lose their connection with the network. These nodes have energy that should not be wasted trying to submit data or control packets to the base station. Partitions must be detected, and the nodes allowed to remain idle until
some later time when new nodes are added and the node regains a path to the base station. Unlike wireless mobile networks, these partitions are not short-lived.

Wireless sensor networks have sensors that are limited in energy, computational power and bandwidth. The algorithms must be simple and efficient. In terms of routing, the algorithm must efficiently create routes, as well as reconfigure routes destroyed by faults in the network. Additionally, the load placed upon any node should be balanced, to maintain an overall healthy network. Any algorithm used must also be limited in its complexity, and limited in the amount of information required to run it.

1.2 Routing Techniques

Link-Reversal Routing (LRR) is a technique developed for dynamic mobile networks used in [9, 3, 1]. Briefly, the links have a status of up or down between every source and destination pair. From the source, messages are forwarded through a single downlink. If no downlink exists, links are reassigned to ensure that all nodes have downlinks. Usually links are reversed; links change from being up to down.

LRR concerns the assignment of these links both in the creation of routes, and recovery when links are created or destroyed. A correct assignment will ensure that messages that are forwarded through downlinks, eventually reach the destination.

The LRR technique gives many advantages. These include providing multiple routes for each node, and reacting only when necessary without interference or intervention of a central authority. Additionally, the LRR algorithms are distributed.

LRR routes are not optimal; optimization is not a priority. In a mobile network, no route is optimal for long, so there is little point in maintaining optimal routing. Wireless sensor networks, though, are far less dynamic, creating an opportunity for improving the quality of the routes.

Shortest distance routing tries to maintain optimal routes. Shortest distance routes minimize the number of times a message is forwarded from source to destination. Shortest distance is hard to maintain. If some nodes die, other nodes may have to change their routes, thus changing their shortest distance. A change in the shortest distance of one node may cause a cascading effect to other nodes.

The DSDV [10] protocol, is a wireless mobile network adaptation of shortest
distance routing. Routing tables are shared to maintain consistent and correct routing information. They are periodically advertised, and distributed in the event of some major change in the network. A wired implementation of a shortest distance algorithm [6] floods the network to construct the routes and again floods the network to reconstruct routes in the event of a link failure. Both DSDV and the shortest distance algorithm maintain a single route to the destination.

1.3 Overview

This thesis will first look into the nature of wireless sensors networks, and their effect on routing. The behaviour and running of some routing protocols in literature will also be investigated.

The thesis will then discuss a new fault tolerant and efficient routing algorithm for wireless sensor networks. Next it will investigate the algorithm with respect to certain issues raised, including efficiency, scalability, and its behaviour in the face of faults. The implementation of this algorithm is then described in detail, along with a description of the simulation model and experiments. We will then compare our new algorithm with the other algorithms in literature. The algorithm will also be compared to some of its variants.

To understand the working of the algorithm, there is first an investigation into the nature of random large topologies for sensor networks. The investigation will focus on their connectivity, and the distances in hopcounts of the nodes from the base station. This investigation will also focus on the changes in the network as a result of node failure.

We found relationships between the topology and the performance of the protocol. Initialization cost is strongly related to the number of connected nodes. The cost of sending a message relates closely with the hopcount of the network. Using shortest distance estimates does route messages through near optimal routes. We also found a mechanism that effectively reduces the cost of initialization, by reducing the accuracy of the shortest distance estimate.

We also found that the cost of recovery is high and is related to the amount of change to the network. When large disruptions of the network occur, we found the cost of recovery to be high, comparable to the cost of reinitialization. The use of multiple links as alternative routes was not as effective, as we thought initially. Even small changes to the network incurred some reconfiguration cost.

We found that the load balancing provided by having multiple links was effective. However, bottlenecks still occur, and they carry a disproportionately large load.
CHAPTER 2

Wireless Sensor Networks

2.1 Factors in Routing

There are several issues that govern the construction of any wireless sensor network routing. These issues arise from the limitations of the sensors themselves, the nature of the topology, and the environment of deployment. Sensors are cheap, limited and disposable. The topology is large, dense and dynamic as these sensors lose power or become faulty. The topology could also be unknown, as sensors are scattered randomly during deployment. The environment could be hostile; there is little or no supervision of the sensors.

Hundreds or thousands of wireless sensors are scattered within a set area for the deployment of wireless sensor networks. These sensors communicate with each other and take measurements of their surrounding. The data gathered by these measurements are then routed to a central station.

Together the sensors, topology and environment, create many constraints and issues on the routing protocol. The following are some of these issues and constraints.

2.1.1 Distributed Routing

According to Merlin and Segall [6], distributed routing means that there is no global knowledge of the network, and that no central tables are required. Each node only has knowledge of its adjacent nodes, and for each node for a given time, there is a preferred neighbour through which it routes its messages. Paths are formed by nodes forwarding messages to their preferred neighbour, until the message request reaches its destination.

Gafni and Bertsekas [3] add that nodes should not have to depend on the station for instruction if they require new routes. Central topological control has several problems.
• Firstly, there is a need for the base station to be aware of topological change. This need necessitates the transfer of great deal of topological information if the network is constantly changing.

• Secondly, the routing of topological information change becomes complicated as the routes to the centre also change. Links are created and destroyed, leaving the node uncertain of how to direct information to the centre.

2.1.2 Reliability

Merlin and Segall [6] elaborate on reliability saying that for a network to be reliable, it must be able to adapt to changes. The entire network or large portions of it should not be disrupted because of the changes. Furthermore, a network should be able to resume its normal operations, i.e. recover, after a finite, preferably short, amount of time.

Small changes in the network should not cause large disruptions to it.

Gafni and Bertsekas [3] suggest that for reliability and stability of the network, nodes should have multiple routes to the base station. Multiple routes allow nodes to choose from alternative routes in the event of the primary route failing, making the routes more reliable. The failure of these redundant routes places little or no cost on the network.

2.1.3 Limitation on Energy, Computational Power and Communication

Kulik, Heinzelman and Balakrishnan [5] discuss the limitation of the sensors. Sensors have limited energy. Routing must conserve energy in both communication and computation. Protocols should be resource aware. The behaviour of any protocol should be influenced by the amount of energy available. Protocols should adapt their communication to the nodes’ state of energy resource. Additionally, sensors have limited computational power. Routing must be simple as there is not enough computational power to run sophisticated network protocols. There is also a limit on the bandwidth available, of the order of a few hundred Kbps. These factors constrain inter-sensor communication.

Estrin et al. [2] add that resources are limited because no human operator will usually maintain the sensors. The sensors are dispensable. They have finite energy but are expected to last for a long time.
They also add that sensors have limited CPU speed and RF bandwidth to save power, as computation and communication consume power. Size is also a constraint on sensors in some applications, such as military applications to reduce detection. This puts a further constraint on the CPU speed, memory, bandwidth and even battery size.

2.1.4 Scalability

Estrin et al. [2] argue that any sensor networks are necessarily large. High numbers are necessary because of a few reasons. Firstly, the radio transmission range is limited so many nodes are needed to have a connected network over an area. Sensors also have limited sensing range and their data is distorted with noise. Subsequently, overlap and redundancy is needed for more accurate sensing. Failure of nodes is also common, so redundancy is needed for the network to survive.

2.1.5 Flooding

Gafni and Bertsekas [3] argue that flooding in wireless networks cause simultaneous burst of messages. Nodes within the network then receive simultaneous messages, causing a collision. Collisions necessitate retransmissions, which cause more collisions. This ultimately leads to instability and collapse.

However, Corson and Ephremides [1] argue that flooding does not require a priori topological knowledge. Therefore it becomes an important part of any protocol that makes no assumption on the topology. Additionally, the ultimate collapse of a network due to flooding depends on collisions, and collisions are managed and controlled by the channel access mechanism. Thus flooding collapse is not directly a property of a given protocol, but dependant on the channel access mechanism.

They also add that both their algorithm, LMR, and the Gafni and Bertsekas algorithm have localized flooding when nodes lose all their downstream links, as a single loss may trigger other downstream link losses. But these simultaneous bursts rarely create message bursts that traverse the entire network.

2.2 Existing Routing Algorithms

Several routing protocols are discussed in the literature to handle the dynamic nature of mobile wireless networks.
Iwata et al. [4] classify three broad categories of wireless routing algorithms:

- Global Precomputed Routing where all routes are precomputed and maintained through periodic updates.
- On Demand Routing where routes are computed only when they are needed.
- Flooding where a packet is sent to all destinations, ensuring that it reaches the intended destination. Scoping may be used to limit the overhead of flooding.

Global precomputed routing is either flat or hierarchical. It includes most distance vector and link state techniques including DSDV [10] and WRP [7]. These techniques do not scale well as knowledge of all nodes in the network is needed.

Hierarchical techniques are employed to enhance scaling. In hierarchical routing, nodes act as switches or end points and organize themselves as clusters. Hierarchical routing greatly reduces the size of routing tables making it more scalable. However, they face many problems such as ensuring that cluster IDs which are generated dynamically are unique. It is also difficult to control the merging and splitting of clusters. The overhead of creating and ensuring membership could also be prohibitive.

On demand routing algorithms includes LMR [1] and TORA [9]. On demand routing is based on a query/reply approach. A query is generated and is routed, usually by flooding, to a destination. The reply reaches the source. In this process, routes are constructed based on the paths that the query and reply have taken. The process scales well because paths don’t have to be remembered for extended periods of time.

On demand routing however faces the problem of initial searching delay, and there is no way to know in advance the quality of the paths constructed.

### 2.2.1 Link Reversal Routing

The GB Algorithm [3] provides a basis for link reversal routing protocols. All link reversal strategies have links with a status such as up or down between every two neighbouring nodes, for every destination. If node \(i\) has a down link to \(j\), \(j\) is a downstream neighbour of \(i\). Nodes forward messages downstream until they reach their destinations.

The algorithm reacts when a node loses all of their downstream links. Nodes with no downstream neighbour cannot forward messages. These nodes cause
“link reversals”, changing the status of their links. The change could possibly cause other nodes to lose all their downstream links, which cause further link reversals.

The way links are assigned is the central computation of all link reversal algorithms.

The problem of routing is conceptualized using graph theory. Let the nodes be a set of vertices on a directed acyclical graph (DAG) where one of the nodes is denoted as the destination. The graph is then destination oriented if there is a directed path originating from every vertex to the destination. The problem of routing is therefore to transform a DAG into a destination oriented DAG.

![Figure 2.1: Link Reversal Example](image)

In Figure 2.1.a, node $b$ has no downlinks. It can choose to reverse any of them, to transform the DAG into a destination oriented DAG. Figures 2.1.b, 2.1.c and 2.1.d are all destination oriented DAG.

In any of the link reversal algorithms, for message forwarding, every node has to only know about its neighbours one hop away. Nodes only choose to which of its downstream neighbours it will forward next. Additionally, every node can have several outgoing links, depending on the number of downlinks it has.

### 2.2.2 Gafni and Bertsekas Algorithm (GB Algorithm)

The GB Algorithm [3] is described in depth by their creators Gafni and Bertsekas. There are two versions of the GB algorithm; a full reversal method and a partial reversal method.

In full reversal, after every iteration each node other than the destination that has no outgoing edges reverses the direction of all its edges. In partial reversal, every node other than the destination keeps a list of edges that have been reversed. After every iteration, if it finds that it has no outgoing edges,
then all edges not on the list, the “unreversed edges”, are reversed. If there are no “unreversed edges”, it reverses all of its edges, and empties the list.

![Figure 2.2: Full Reversal Initialization Example](image)

In Figure 2.2, the graph begins as a destination disoriented DAG. After reversing all the links on node $b$, all nodes now have downlinks and paths to the base.

In Figure 2.3.a, the link between node $d$ and node $g$ fails. Node $d$ now has no downlink, causing it to reverse all of its links (Figure 2.3.b). This causes node $a$ to lose all its link and cause another link reversal (Figure 2.3.c). Again, after reversing links, all nodes now have downlinks and paths to the base.

A computational model for the full reversal model follows. For every node, there is a pair containing a unique id, and an associated height $h$. A total ordering is enforced on the nodes. A node $i$ is higher than a node $j$ if $h_i > h_j$. However if the heights are equal, the ordering is based on the unique id. A higher node, has a downlink to a lower node.

Consequently, a node has no outward link if it is lower than all of its neighbours. At every iteration, if a node $i$ has no outgoing links, node $i$ resets its height.
\[ h_i = \max(h_j | j \text{ is a neighbour of } i) + 1 \]

Now node \( i \) has an outgoing link to all its neighbours, all its links have been reversed.

Nodes lose their downlinks as a result of link failure, or as a result of link reversals.

The protocol is fully distributed, both in its forwarding of messages and its construction of routes. Every node has to only know about its neighbours one hop away, and no commands are given by the base station. Like other link reversal algorithms, it too has multiple routes for each node. The algorithm is also proven to have no cycles, and terminates in a finite number of iterations, with a destination oriented directed acyclic graph.

The GB algorithm is unstable in the face of partitions. Nodes will continuously reverse their links, resulting in arbitrarily large heights, and waste energy as both control and data packets are transmitted. The authors suggest that partitions can be detected by setting a limit on the height. However, this only indirectly detects partitions while the wastage occurs prior to its detection.

2.2.3 Temporally Ordered Routing Algorithm (TORA)

TORA [9], created by Park and Corson, is another link reversal routing algorithm. In TORA, the reversal is more controlled. Instead of a pair, it has a quintuple. A reference level, timestamp and reversing node id is kept. Whenever a node loses all its downlinks due to a link failure or other change in topology, the node defines a new (higher) reference level, and updates the “reversing node id” and “time stamp” value.

A reflected value is also kept. Information is considered to be reflected in case when it is received, all of a node’s neighbours are at the same reference level. If a reflected packet is received by the originator of the reference level, a partition is detected and all routes are cleared.

A control mechanism allows the nodes to detect partitions, and clear invalid routes. Additional control localizes the change better, minimizing the number of link reversals.
2.2.4 Lightweight Mobile Routing (LMR)

LMR [1] is a distributed routing algorithm for mobile wireless networks by Corson and Ephremides. LMR adapts quickly to changes in the network topology. It builds routes only when necessary, and it builds them quickly. It also reacts quickly to changes, and reacts only if a reaction is necessary.

The algorithm starts by flooding a query (QRY) packet, from a source to a destination. In Figure 2.4.a, node $a$ starts the protocol. This QRY is further propagated by the nodes in Figure 2.4.b, 2.4.c, and 2.4.d.

Upon reaching the destination, the destination floods a reply (REP). This flood constructs the link status of the network. Nodes that receive a reply have downstream links to the nodes that relayed the reply message. In Figure 2.4.c, the destination $g$ (base) receives the QRY, it sends REP. Node $g$ becomes a downstream neighbour of node $d$ and node $e$ in Figure 2.4.d. The network is stable by Figure 2.4.h.

If a node loses all of their downstream links, the node then sends a Fault Query (FQ) upstream. FQ erases the direction on the links, links become unassigned. If a node loses all of their downwards link, as a result of an FQ, it too will send an FQ upstream. If a node instead has valid downstream links, it will instead send a REP, reconstructing the network as it was before.

![Figure 2.4: LMR Initialization Example](image-url)
In Figure 2.5.a, node $d$ sends the FQ. In Figure 2.5.b, node $a$ loses all its downlinks and sends the FQ. In Figure 2.5.c, node $e$, receiving an FQ from $d$, replies with a REP, because it still has downstream nodes. The network is reconfigured by Figure 2.5.d.

The FQ construct allows the distributed detection of partitions in the network. A partitioned section of the network will eventually have all their links erased through the sending of FQs.

LMR introduces additional constructs to prevent deadlocks and loops. Links are not just unassigned, up or down. Additionally, LMR has downstream-blocked, unassigned-waiting, and awaiting-broadcast links. These statuses are assigned depending on the situation, usually as a result of link failures or link reversals. The extra status provides better control.

2.2.5 Destination-Sequenced Distance Vector (DSDV) Routing

DSDV [10] is a routing protocol for mobile wireless networks by Perkins and Bhagwat. It is adaptive, and tries to maintain shortest path information.

Each node keeps a copy of a routing table. These tables hold a list of available destinations, and the number of hops. The information in these tables is shared in the event of some major change, for example a link breakage. They are also shared periodically. Full tables or partial updates are transmitted to update information about a table.

Damping is implemented to reduce fluctuation of the routes. Updates are made but not advertised if the node feels that it can receive better information soon. Nodes remember how frequent certain routes change, and react more slowly to fast changing routes.

These tables are merged and analyzed based on the number of hops or se-
quence numbers. Newer information and routes with lower hopcounts are preferred.

Park and Corson [9] argue that DSDV is possibly inefficient due to excessive overhead as it transmits both periodic and table exchanges and announcements of major topological changes. DSDV must also wait for updates from the destination, making routing less independent and suffers from synchronization problems. DSDV is also sensitive to its parameters, including periodic update interval, maximum value of the “settling time” for a destination, and the number of update intervals that may transpire before it is considered “stale”. The behaviour of the topology may change over time, making the choice of good parameters difficult.

2.2.6 Shortest Path Algorithm

This shortest path algorithm by Merlin and Segall [6], is designed for wired networks and provides a method for all nodes to find their shortest distance, and maintain shortest distance information in the event of link failures.

The protocol has two main parts, initialization and recovery.

Every node $i$ has an estimate of its distance or weight to the base station, denoted as $d_i$. The base, by definition, has $d_{base} = 0$. The base sends a message $[d_{base}]$ to all its neighbours. A node receiving this or any other distance estimate, $d_j$, then compares its current distance $d_i$ (which intuitively begins at $d_i = \infty$). If $d_i > d_j + d_{ij}$, where $d_{ij}$ is the weight of using the link $(i, j)$, then node $j$ becomes node $i$’s preferred node and $d_i = d_j + d_{ij}$. Node $i$ transmits the new $d_i$ to all neighbours, except for $j$.

Once node $i$ receives a $d_j$ from all its neighbours, it will then transmit $d_i$ to its preferred neighbour, the transmission skipped above. By induction, eventually all neighbours of the base will transmit $d_j$ to base. The base has no preferred neighbour, but these messages confirm a successful end of the initialization.

Figure 2.6 shows the initialization of the shortest distance algorithm. The base starts in Figure 2.6.a by sending a D(0) packet. In Figure 2.6.b, both nodes $d$ and $e$, compare the distance, further propagate more D(X) packets, and then set the originator of the D packet, as their preferred node. In Figure 2.6.d, nodes $a$ and $c$ confirm their distance with their preferred neighbour. This process continues until Figure 2.6.f where the network is stabilized.

The algorithm reacts to breakages of links in the network. Assume a preferred link fails between node $i$ and node $j$. Node $i$ loses its connection to its preferred neighbour $j$, while $j$ maintains its preferred neighbour. $j$ sends a request, REQ, which is routed down to the base. If a node recently sends out a REQ, it will
ignore and not forward any subsequent REQ.

Node $i$ will instead set $d_i = \infty$, send $d_i$, keep no preferred node, and set $d_{ij} = \infty$. Any node receiving $d = \infty$ from a preferred neighbour will repeat the same process. The base, receiving a REQ, will start a new update cycle, once the previous cycle has ended.

To ensure there is no confusion, the cycle number is appended to all messages. Only new cycles are acted upon, distances from previous cycles are ignored.

In Figure 2.7.a, the link between $e$ and $f$ is broken. Detecting this, $e$ sends a REQ to the base station, while $f$ sets $d_f = \infty$, and sends out $D(\infty)$. Node $c$ in Figure 2.7.b, receiving this clears its path as well. When the base receives the REQ, it will then start another round of initialization, similar to Figure 2.6.
CHAPTER 3

Sourced Distance Link Routing

3.1 Motivation

The primary motivation for adapting Link Reversal Routing (LRR) into wireless sensor networks (WSN) is its fault tolerance. Link failure, when rare, has virtually no effect on routing; alternate paths provided in LRR are used immediately. Networks only react when the last downlink, i.e. last connection to a path to the base, fails. WSN being dense, static, and having relatively infrequent changes, should be stable and incur less cost for rerouting. When nodes do fail, LRR handles the problem locally. This design is scalable, and suitable for large WSNs.

The multiple path structure in LRR is also attractive to WSNs. Multiple path routing spreads the load on any individual node. Nodes have limited energy, and if a single node is overburdened, it will die. Spreading load ensures that the network is healthy as a whole.

Distance routing was introduced into the investigation for two reasons. The first reason was to ensure quick initialization of the network. Initial experiments with the “pure” GB algorithm did not work well. WSN are large, and some knowledge of the network is needed to maintain efficient routing. In the GB algorithm, an up or a down status on a link does not mean much; it does not indicate whether messages are getting closer or further from the base station. Distance was used to set the direction the right way.

The next motivation was to have optimal routing. Optimal routing minimizes the total number of messages sent by the nodes of the network. When routing is optimal, the number of times each node forwards messages is equal to its minimum distance to the base station.

However, optimal routing can cause an imbalance on the load of individual nodes. Optimal routes are always used while longer routes are underutilized. In Figure 3.1, node a, b and c always route through node f, and never route through node e and g. We hope that multiple-path routing will alleviate the situation.
Node $d$ in the figure will balance its load between node $c$ and $e$ as both nodes $c$ and $e$ have the same hopcount.

### 3.2 Core Concepts

The algorithm uses shortest distance estimation to set link statuses for routing. *up*, *down* and *unassigned* are attributed to links for routing. Nodes forward messages down to the base station. These link statuses are assigned based on their estimated distance to the base station. Neighbours with lower distance are downstream neighbours. Distance estimation is first constructed then maintained throughout the running of the protocol.

Each node keeps a list of its neighbours, and for each neighbour, a distance estimation, a direction, and whether or not the neighbour is a downstream neighbour.

This distance information is shared with its neighbours both actively and passively. Distance is included in every packet sent. In wireless networks, messages are broadcast. This means that all messages are heard even though the node may not be the intended recipient. These messages are ignored, but the distance estimate is taken. The estimation continually, without much cost, improves as messages are sent and received. A node may also send an UP Packet, to announce a significant change in its distance estimate.
3.2.1 Downstream Nodes

Node $i$ can become a downstream neighbour of node $j$ if node $j$ knows that there exists a path from node $i$ to the base station that does not include node $j$. In other words, node $i$ has at least one route to the base station that is independent of $j$. If $i$’s distance estimation is based on that independent route, $i$’s distance estimation is independent of $j$’s distance estimation.

If node $j$ loses all of its downstream links, it becomes untrusted, and its distance is invalid ($d_j = \infty$). Any neighbouring node $k$ is now not allowed to rely on node $j$ for a distance estimate. If node $k$ has some other downstream link, it can reevaluate its position based on the other downstream link, or else it too becomes untrusted.

A node $i$ receiving a lower distance estimation from any neighbour $j$, will trust $j$. Clearly, if $d_i < d_j$, then $j$ has a path to the base station that does not include $i$. Therefore, lower distance nodes are trusted.

Identifying downstream neighbours is important to ensure that distance estimates are reliable. Node $j$ cannot rely on node $k$, if node $k$’s estimate was based on its own, now invalid, estimate. Node $j$ must rely on an independent node.

3.2.2 Shortest Distance

The shortest distance estimate of any node is the minimum shortest distance of a downstream neighbour + 1.

$$d_j = \min(d_i : i \text{ is a downstream neighbour of } j) + 1$$

If a node has no downstream neighbors, its distance is $\infty$, and it becomes untrusted.

The base station has $d_{\text{base}} = 0$.

In distributed Bellman-Ford (DBF) type algorithms, the estimate is based on all the neighbours of the node, not just the downstream neighbours. The difference becomes apparent when the last downstream neighbour fails. In a DBF type algorithm, the new distance would automatically be reset.

$$d_j = \min(d_i : i \text{ is a neighbour of } j) + 1$$

In this algorithm however, the distance is set to $\infty$. 
3.2.3 Slack

Change of distance estimation may prompt an announcement of the new distance. This announcement is suppressed by slack, for a distance change less than the amount

\[ |\text{old distance} - \text{new distance}| > \text{slack} \]

In other words, a node does not report its distance change to its neighbours if the change is less than the slack. Use of slack reduces traffic in the network.

Slack can also be used to focus on recovery. All changes that increase the distance estimate are announced, but for changes that reduce the distance estimate, only changes more than slack are announced.

Slack could lead to distance estimates that are not the shortest distance. But the estimate can then be improved. As messages are sent and distances are piggybacked, distance estimates improve. Also, paths are based on direction, which depends on the relative distance estimation, rather than an absolute distance, so accurate distance is not critical. What is more important is that each node has several downlinks.

3.3 The Execution of the Algorithm

3.3.1 Initialization

All nodes other than the base have \( d = \infty \) and no downstream links. The base is trusted, has no downstream link and \( d_{\text{base}} = 0 \). The base initiates the network by sending a UP(Base,0), because from the base’s point of view, the links between it and its neighbours are uplinks.

Nodes receiving an UP with an \( d_{\text{UP}} < d \), will take \( \text{UP}_{\text{source}} \) as a trusted source. A node \( i \) will then set their distance to \( d_i = \min(d_{\text{UP}} + 1, d_i) \)

The distance of the neighbour is also recorded. If the neighbour has a lower distance, the link to the neighbour is set to down, if the distance is higher, link is set to up, if they are the same, link is set to unassigned.

In Figure 3.2.a, the base initiates the protocol, by sending out the first UP packet. This packet is received by \( d \), but is delayed for some reason to \( e \). In Figure 3.2.b, \( d \) then propagates the UP further, becoming a downstream neighbour to both \( a \) and \( e \). The process continues, until Figure 3.2.e, where \( e \) correctly receives UP(0). Finding a better estimate, \( e \) broadcasts this new estimate, and
Figure 3.2: SDLR Initialization Without Slack Example

Figure 3.3: SDLR Initialization With Slack Example
erases its downlink to $d$. Both $b$ and $f$, further propagate their improved estimate in Figure 3.2.f. The network stabilizes by Figure 3.2.g.

However, in Figure 3.3.e, when $e$ correctly receives UP(0), it changes its estimate to $d_e = 1$, but it does not announce it due to slack. It only erases its downlink to $d$, and creates a downlink to the base. Fewer packets are sent because of slack.

3.3.2 Recovery

A node performs recovery only when it loses its last downstream link. The node sends an Invalid (INV) packet. All nodes receiving it remove themselves as a downstream neighbour. These nodes also send out INV and set their distance to $\infty$ if, as a consequence, they lose all their downstream neighbours. Nodes receiving an INV that still have other sources will transmit an UP. Nodes that recently lost their distances, receiving an UP will update their distances and link directions, and advertise their new distances as they did during initialization.

![Figure 3.4: SDLR Recovery Example](image)

In Figure 3.4.a, node $d$ sends an INV as it has no downlinks. In Figure 2.5.b, node $a$ loses all its downlinks and sends an INV. In Figure 2.5.b, node $e$, receiving an INV from $d$, replies with an UP, because it still has downstream nodes. In Figure 2.5.c, node $b$ also sends an UP, while $d$ now advertises its now valid distance. By Figure 2.5.d, the network is reconfigured.

In Figure 3.5, there is a partition. Node $d$ sends an INV but this eventually causes all nodes to lose all their sources. By Figure 2.5.c, the network is reconfigured, and all nodes realize that they are on a partition, as neither they nor their neighbours have downlinks ($d = \infty$).
3.3.3 Partitions

Partitions can cause misleading information to be distributed. Nodes may reply with an UP packet, if they have not yet realized they they have been disconnected from their downstream neighbours. It is also possible that the downstream neighbours have lost all their downstream neighbours, and have not yet informed their upstream neighbors.

In Figure 3.6.a, node $g$ fails, and node $e$ detects it first, and send out an INV. In Figure 3.6.b, node $d$ detects it and send out an INV. Node $b$ receives the INV from $e$, and assumes that node $d$ is a valid downstream link. It then falsely replies with an UP to node $e$. Node $e$ in In Figure 3.6.c, falsely informs its neighbours that it is now trusted, and broadcast its own UP.

To prevent the above situation, a delay is held before replying to an INV. This delay allows the collection of all INVs, before UPs are sent out, making the UP more reliable. Additionally, it was observed that partitioned nodes frequently move from being trusted to untrusted. Consequently, nodes that frequently moves from being trusted to untrusted will increase their delay. While this will increase the speed of detecting a partition, this could delay network recovery.
3.4 Feature Comparisons

The GB algorithm [3] and SDLR have similar features. Both have height to regulate direction of the links. In SDLR, height is more controlled, and is equal to an estimate of the shortest distance. Both provide multiple routes for each node, and both react only when the last downlink fails.

GB does not handle partitions well. One reason is trust. Reversing links on a partition sets a downstream path to nodes with no path to the base station. In SDLR, partitions are handled because all downlinks are to trusted sources, with paths to the base station. Only nodes which are certain of the reliability of their distance estimates reply to Invalid packets.

DSDV [10] and SDLR both suppress information. In SDLR, information is suppressed because it is not critical to have an accurate shortest distance estimate. SDLR also suppresses information thought to be unreliable; if it detects that the node may be disconnected from the base station. DSDV also suppresses information in anticipation of better information. Suppression of information gives the network more stability, and increases efficiency by reducing traffic.

DSDV has a timer to periodically broadcast distance information. SDLR however relies on piggybacking on other messages to continuously advertise distance information. SDLR has only one parameter, slack, so it does not have as many issues of parameter sensitivity as DSDV.

Merlin and Segall’s [6] shortest distance algorithm (SD) and SDLR both initialize in the same way. Nodes build upon progressively improved estimates, which they share with neighbouring nodes. However SD keeps only one preferred neighbour. All neighbours with a lower distance estimate are “preferred neighbours”. This allows SDLR to initiate recovery less often as SD.

Recovery is quite different. SD informs the base station which then reinitializes the whole network. In SDLR, the request for valid information only goes up to the first trusted node. In a way, SD assumes only the base station is trustworthy.

LMR [1] has a very similar strategy for initialization and recovery of the network. In initialization, REPs (Replies) are flooded upstream in a similar way to UP. This flood creates the link status. In recovery, LMR depends on a node with at least one downstream link to give a REP after receiving an INV (Invalid), akin to a trustworthy node giving an UP. On a partition, eventually both SDLR and LMR will find no source, and clear the link status (distance estimate). SDLR does not have deadlock and loop prevention. It depends on delays for accurate partition detection. Routing is also similar, as nodes forward
messages downstream.

SDLR with a very high slack value behaves in the same way as LMR. There is no attempt to improve estimates when slack is high, so the initial UP flood is enough for initialization. As stated above, recovery and routing too is similar. SDLR can be thought of as an LMR which is parameterized by the degree of accuracy required for optimal routing.

3.5 Simulation

3.5.1 Model for Simulation

A model has been built to simulate the running of the protocol and to assess and measure its behaviour and correctness. The model primarily consists of nodes, messages, wireless links between nodes, and the topology itself.

Nodes transmit and receive packets from their neighbours. Each node has a single out buffer, \( outbuff \). A message is put into buffers of neighbouring nodes. Each node keeps a separate in buffer, \( inbuff \), for each neighbour. Neighbours are determined by their connection. Nodes are connected if they are in transmission range of each other. Nodes process the messages they receive, via a state machine model. The \( inbuff \) is managed as a ring, so that each node processes one buffer at a time. It may also choose to wait and read only from one buffer, for example, if it is waiting for a reply.

The links between the nodes copy messages from the source \( outbuff \) to the multiple \( inbuffs \) of the destination. This simulates a message transfer from one node to another. This link is updated periodically. The simulation assumes that there are no collisions and all messages sent are received, and received in the right order.

Messages are transmitted from one node to another. Messages have several addressing schemes, either to a single or multiple destinations, or to be broadcasted for everyone. For example, nodes forward \( data \) messages to a single downstream neighbor, send an \( up \) packet to all its upstream neighbors or broadcast its distance to all its neighbors.

Messages keep track of their source, destination, type, data if any, and hop count. If a message reaches it maximum hopcount, it is silently discarded. Messages are transmitted in the following way. They are placed on the \( outbuff \) of the sending node. Upon updating the link, messages are copied from the \( outbuff \) to specific \( inbuffs \) of the all neighbours. Each node has one \( inbuff \) for each of its
neighbours.

3.5.2 Experiment Description

The topology is randomly created. A fixed number of nodes are located at a random (x,y) coordinate within a 1000x1000 units square box. Each node is given a random ID between 1 and 9999. Each node is given a set range. Any two nodes within range of each other are connected. The number of links for each node is however limited to 20 per node. All links are bidirectional.

Nodes are indexed during construction. The first node, node 0, is set as the base. The base station, unlike all other nodes, has a fixed location. It is located at the middle top part of the simulation area. See Figure 3.7 for an example topology.

![Figure 3.7: Sample Topology with Base Location (nodes:500, range:70)](image)

In Figure 3.7.a, there are 500 nodes with a range of 70 units. In Figure 3.7.b, 10% of the nodes are destroyed. Destroyed nodes are marked with an X, nodes that become partitioned are marked with an O. The dots without edges are a placeholder. At these dots, information about each node can be displayed including its shortest distance, its current estimation of its distance and the number of messages it has sent or received.

In every phase, every node is updated, along with its links. In the update, each node will read its `inbuff`, process the messages it receives and possibly place
new messages in its outbuff. Updating the link moves the messages from one node to its neighbours.

Nodes accept two main commands from the application layer during the simulation, sendMessage and die.

sendMessage instructs a node to generate a data message and route the message to the base station. This simulates a node taking measurements of its surroundings, and reporting the information to the base station.

die instructs a node to stop sending or receiving any messages and disconnect itself from its neighbours. All buffers This simulates a node that has depleted its energy or become faulty.

Once everything is initialized, in the first phase the base station will initiate the protocol, and the nodes will continue to communicate creating routes for themselves.

After a period of time, the message phase begins. In this phase, nodes send messages to the base station. Nodes are globally instructed to sendMessage in a random order. The number of messages sent is distributed evenly throughout the phases, until the end of the message phase.

At some phase in the middle of the message phase, a set number of nodes are instructed to die. Nodes die as messages are being sent, and the surviving nodes must recover and find new routes if their old routes have been destroyed.

At the end of the message phase, the network is allowed to settle down and complete the transfer of the final messages still being routed.

Partitions are not investigated in these experiments. Nodes with no path to the base station are removed from the analysis. Only nodes which can participate in the protocol are accounted for.
CHAPTER 4

Topological Analysis

With global knowledge of the topology, with relative ease, optimal routes for any criteria can be established, including shortest paths, well-balanced load, highest bandwidth or lowest latency. Here a topological analysis is conducted, to find shortest distance paths. This provides a baseline for the analysis of SDLR.

The topology changes as nodes fail. Paths may change, or the network may become partitioned. The effect of these changes on the topology also are important to understanding the behaviour and fault tolerant nature of SDLR.

4.1 Experiment Description

The topology is created randomly as described in the model of simulation. Dijkstra’s algorithm is executed on the topology to obtain the minimum distance to the base station, $\delta$, on every node. The weight of each link is 1. Each node, other than the base starts with $\delta = \infty$, $\delta_{\text{base}} = 0$.

Nodes with $\delta = \infty$ upon termination of the Dijkstra’s algorithm are considered partitioned from the network.

The position of the base is set to the middle top part of the simulation area. This decreases the variation in the results, compared to having a randomly located base station. If the base station is in the centre, the network diameter would be smaller compared to if the base station was at some corner.

Nodes that are destroyed have $\delta = \infty$.

The topological analysis will find topologies that are practical and interesting. A practical topology is one where wastage is minimized; nodes deployed are useful, i.e., can communicate with the base station. Interesting topologies are topologies that are interesting in the event of node failures. Very highly connected topologies change very little as nodes fail or as nodes are added. Interesting topologies therefore are connected topologies which become disconnected after
considerable amount of damage; i.e., a large number of faulty nodes.

The experiments are run with combinations of

- 600 nodes, range 60 units
- 500 nodes, range 60 units
- 500 nodes, range 70 units

For each scenario, the experiment is run 40 times, and results are averaged.

4.2 Connectivity

A node is considered to be connected to the network, if there is a path from it to the base station. Connectivity is taken as the proportion of connected nodes. Connectivity is an important consideration. Connectivity is also a measure of how partitioned a network is. Additionally, having low connectivity means that a large number of nodes are being wasted as they cannot communicate with the base station. Only a few nodes are left to participate in the protocol. Low participation will bias any results in node efficiency as, effectively, unconnected nodes send no messages. Thus it is important to investigate the nature of highly connected topologies.

Higher traffic densities contribute to higher connectivity. Higher traffic density is gained by increasing the number of nodes, or the range of each node in a fixed area. Higher traffic density implies higher probability for a node to be connected to a single or multiple neighbours. An increase in the connectivity of nodes at an individual level pushes an increase in the connectivity of the network as a whole.

Figure 4.1 shows clearly that having more nodes or higher range increase the connectivity.

In Figure 4.1.a, the rate at which connectivity increases with respect to nodes depends on the range. Having a higher range means that the addition of each node, contributes more significantly to connectivity. To compare, at a range of 50, adding 300 nodes, from 400 nodes increases the connectivity from 10% to 80%. For a range of 60, adding only 200 nodes, from 100 nodes changes connectivity in the same way. This is also visible in Figure 4.1.b, as increasing the range increases the connectivity very quickly.
Connectivity vs. nodes  Connectivity vs. range

Figure 4.1: Connectivity on Varying Topologies

The topology with 500 nodes, range 60, seems interesting as decreasing the number of nodes causes many nodes to be partitioned. Investigating such topologies investigates the nature of the protocol with respect to partitions.

As nodes are destroyed, some other may lose their paths to the base station. At some critical level, destroying a few nodes will cause major disruptions in the network.

Figure 4.2.a shows the loss in connectivity due to node failure, while Figure 4.2.b emphasizes the nodes which become disconnected because of the failure of other nodes. It is a measure of damage; the number of nodes who were previously connected, and have not been destroyed, but are disconnected from the network.

The decline of connectivity in Figure 4.1.a is not linear. This is emphasized by the growing gap between the three lines. Less connected topologies degrade more quickly. Additionally, as more nodes are lost, connectivity degrades, increasing the rate of connectivity decline.

From Figure 4.1.b the most highly connected network (500 nodes, range 70), is not greatly damaged by the destruction of nodes. For a considerable amount of node failure, the network is undamaged. This is the strongest network, due to its resilience to node failure.

The weakest network (500 nodes, range 60), is damaged throughout, losing many nodes as the network disintegrates.
4.3 Hopcount

Hopcount refers to the number of hops to the base station. Hopcount is a view of the network size. The radius of the network is the maximum hopcount of any connected node. Apart from the radius, the average hopcount also indicates the size of the network. The larger the hopcount, the longer paths are from the nodes to the base station.

Figure 4.3: Maximum and Average Hop Count vs. Nodes

Connectivity Loss Damage in Network

(a) (b)

Figure 4.2: Change in Connectivity due to Node Failure

4.3 Hopcount

Hopcount refers to the number of hops to the base station. Hopcount is a view of the network size. The radius of the network is the maximum hopcount of any connected node. Apart from the radius, the average hopcount also indicates the size of the network. The larger the hopcount, the longer paths are from the nodes to the base station.

Figure 4.3: Maximum and Average Hop Count vs. Nodes
Figure 4.4: Maximum and Average Hop Count vs. Range

Figure 4.5: Effect of Range on Hopcount
In Figure 4.3 and Figure 4.4 both the maximum and average hopcount have the same behaviour. In the figures, at first the hopcount increases, reaches some maximum, and then declines. The increase in hopcount is related to the increase in connectivity. Nodes that are not connected are not counted. As more nodes are connected, the network becomes larger, thus increasing the maximum and average hopcount.

The decline then follows. The decline happens for two reasons. Firstly, the maximum hopcount becomes increasingly more difficult to increase. To increase the maximum hopcount, a node must connect itself only to nodes with the maximum hopcount. As the network becomes more dense, this becomes increasingly improbable. Secondly, as the network becomes more connected, there are more paths for each node to the base station. More options provide the opportunity for more shorter paths.

In Figure 4.5, the effect of increasing range is demonstrated. Nodes can now reach more distant nodes, thus allowing them to “skip” certain hopcounts causing a reduction in the average and maximum hopcount. These results show that hopcount is not a direct measure of network size. It only increases with size when connectivity is low.

The same pattern is seen as we examine the effect of node failure. Failures could cause rerouting from some shortest path, to some longer path. Failures can also cause the network to shrink as the edges are partitioned from the network.

Examining Figure 4.6.a and Figure 4.6.b, the weakest network (500 nodes, range 60), shrinks as nodes fail. This is consistent with its loss of connectivity. The strongest network (500 nodes, range 70) stays the same. Because of its high connectivity, and its high node density, the loss of some nodes does not affect it very much. The middle network (600 nodes, range 60), first increases then decreases as it loses its connectivity. Note that at around 15%, it is the same topology as (500 nodes, range 60) and disintegrates.

Figure 4.6.c shows how many nodes change their hopcount as nodes fail. This includes those who now have $d = \infty$ because they were destroyed or have become disconnected from the network. From the Figure 4.6.c, we find that shortest distance is extremely sensitive to node failure. Small changes in the network, cause many nodes to change their hopcount. This is true in all three network topologies, irrespective of their strength.
Figure 4.6: Change to Hop Count due to Node Failure
4.4 Downlinks

The number of downlinks a node has is an indication of the resilience of the network. Having multiple downlinks helps the nodes to survive in the face of faults and spread the load onto several different nodes. Node death reduces the number of downlinks, reducing the resilience of the network.

![Downlinks vs. nodes](a)

![Downlinks vs. range](b)

Figure 4.7: Average Number of Downlinks on Varying Topologies

![Downlinks vs. percentage of kills](a)

Figure 4.8: Average Number of Downlinks vs. node failure

From Figure 4.7 the graphs are simple, showing a linear increase in the number of downlinks as the graph becomes more dense. From Figure 4.8 the opposite
occurs as nodes fail as the quality of the network degrades. Disintegration of the network does not seem to affect the quality of the paths of the surviving nodes. Quality degrades at roughly constant rate.

The number of downlinks, along with the quality of the network, seems to degrade with higher slack. By not advertising a lower distance value, nodes become unaware that some of their neighbors are downlinks.
CHAPTER 5

Efficiency

A cost is incurred every time a message is sent and received. Efficiency in routing is to reduce the total cost incurred by the network to run a specific task. Efficiency also has to be seen at the node level, the load on each node should be well distributed.

The major costs of the protocol include initiation, message sending and recovery. Topological information has to be collected and distributed to initialize the network via the sending and reception of messages. Message sending involves the cost of relaying the message from one node to another. In recovery, topological change information has to be collected and distributed, along with commands for any changes to routing.

5.1 Initialization cost

The protocol requires all nodes to correctly set their distances, and the directions of all their links. Packets are sent to initialize this. To obtain initialization cost we run the simulation without sending any messages, and without destroying nodes.

From Figure 5.1 we see the relationship between slack and initialization cost. Slack is a dampening factor. The higher the slack, the fewer messages are sent. Figure 5.1.b is a measure of how many times a single estimate is repeated. In perfect flooding, a single node perfects its estimate an average of 6 times, for the given topologies. Perfect flooding of shortest distance estimates seem excessively expensive on a network. Such large cost of perfect flooding could lead it to being less efficient, even though it would provide optimal routes to the base station.

Figure 5.1, even increasing slack from 0 to 1, gives a large reduction both in the total initialization cost and cost per node. The lower bound is of course one packet per node, where each node updates its estimate once, and never announces an improved value; i.e, slack is $\infty$. 

35
Figure 5.1: Initialization cost vs. slack

(a) Total Cost

(b) Average Cost per Node

Figure 5.2: Initialization cost vs. nodes

(a) Total Cost

(b) Average Cost per Node
From Figure 5.2 and Figure 5.3, we find again that a larger slack value implies a much lower initialization cost, across many different topologies.

Figure 5.2.a shows a linear increase with respect to the number of nodes, emphasized by Figure 5.2.b, as the cost per node grows very slowly. This shows that the initialization of the protocol is scalable, making it applicable to large wireless sensor networks.

Examining the cost on each node on Figure 5.2.b, we find an initial growth in the cost of initialization. This is because, before a certain number of nodes are being included into the network, most nodes are not connected to the base station. They do not participate in the initialization, so cost is very low. As the network grows, the cost increases. After that adding more nodes to the network does little to increase the load on each node; newly added nodes participate in exactly the same manner as the nodes that were there before them.

From Figure 5.3 large increases in range seems to have very little effect on the initialization cost, apart from its initial influence on connectivity. The number of nodes is the main contributor to initialization cost.

5.2 Message sending cost

To send messages, messages are relayed from one node to another until they reach the base station. In addition to that, control messages are sent to manage the topology and routing. Here, messages refer to data messages generated by a node
to be routed to the base station. A packet refers to a datalink layer packet.

Message sending cost is measured as the number of packets sent by all nodes in the network, minus the number of packets sent for initialization of the network. This cost is effectively the cost of routing.

The costs are examined on different topologies, with different values of slack.

Figure 5.4: Cost of Message Sending vs. slack

Figure 5.4 shows the effect of slack on message sending cost. When a large number of messages are sent, as in Figure 5.4.a, and Figure 5.4.b, the cost of messages sending is not very much affected by the value of slack. This is clearly different to Figure 5.1, where changing slack from 0 to 1, cuts the initialization cost by nearly half. We find that, although routes are not optimal due to slack, they are still efficient.
However, when a small number of messages are sent, such as in Figure 5.4.c and Figure 5.4.d, there is a large difference. Firstly, routes are less optimal, so there is an increase in cost. There is also an increase in cost because as messages are being sent, and distances are being discovered, some reconfiguration of the network to improve estimates may occur. When slack is extremely high however, reconfiguration never occurs, which explains why the cost when slack is 5 is low. The number of packets sent is very sensitive to slack, especially when only a few messages are sent and most of the cost is reconfiguration cost. This explains the huge variance in Figure 5.4.c and Figure 5.4.d.

This relationship is further examined by looking at the optimality of message sending in Figure 5.5. Here the cost is first divided by the number of messages to extract the cost of each message. A message is optimally routed if the number of times it is relayed is equal to the shortest distance, or minimum number of hops, from the node to the base station. Dividing cost of each message by the average hopcount, we find a measure of the efficiency. An optimal message is routed with a measure of one.

![Figure 5.5: Efficiency of Message Sending](image)

Figure 5.5 shows the benefit of perfect flooding, i.e zero slack. When slack is low, the paths more correctly reflect the shortest path. In Figure 5.5, it can be seen that lower slack reduces the cost in routing. We also find that all perfect floods have optimality close to or equal to one. For the rest, their optimality tends towards one as the number of messages sent increases. This shows how the network slowly learns more about the network, through passive listening. Note that the difference is small, for 500 messages, the efficiency is around 1.1, which explains why slack has little contribution to message cost in Figure 5.4.

Figure 5.6 and Figure 5.7 further emphasize the relationship between hop-
Figure 5.6: Cost of Message Sending vs. nodes

Figure 5.7: Cost of Message Sending vs. range
count and cost of routing. As number of nodes increase, or range increases, hopcount reduces as seen in Figure 4.3 and Figure 4.4. This is reflected in the decrease of routing cost.

![Figure 5.8: Cost of Message Sending vs. number of messages](image)

Figure 5.8 shows the relationship between the cost of message sending and the number of messages. Clearly, sending more messages increases the cost, both on the network as a whole in Figure 5.8.a, and on each individual node as in Figure 5.8.b. The increase is linear, sending twice as many messages costs twice as much.

5.3 Cost of Node Failure

Node failure causes a change in the topology, and a reconfiguration of the topology. Cost is incurred to maintain shortest distance information, and to route through longer routes. The results for 500 nodes with range 60 is not shown. The topology disintegrates, making it useless after node failure.

Increase in cost due to node failure is affected by many factors. Firstly, an increase in hopcount would increase the cost of routing. This would reflect in a higher cost after node failure. Referring to Figure 4.6, there is roughly a 25% increase in the hopcount of the (600 nodes, range 60) network, and a roughly 15% increase for the (500 node, range 600) network.

There is also a cost to reconfigure the network. From Figure 4.6.b, we find that large number of node have to be reconfigured. However, there are partitions
in the network. This decreases the size of the network, and these nodes do not require reconfiguration.

The large change in hopcount, and more importantly the large number of nodes with incorrect distances reflect in the high cost of reconfiguration seen in Figure 5.9.a. The cost is comparable, to a complete reinitialization of the entire network. From Figure 5.9.b, it costs each node up to fifteen packets each to reconfigure, whereas it would only cost six to seven packets to reinitialize. Its advantage over reinitialization of the entire network, is that the change is handled locally, it only occurs when faults occurs, and no central control is needed. Also, the cost is dependant of the amount of damage incurred. If nodes continuously but slowly fail, reinitialization would have to be done multiple times. The high cost of reconfiguration is still somewhat justified, as the network has changed considerably.

From Figure 5.9.a, small changes have large effects on the cost. This is unacceptable, as nodes were supposed to use their alternative paths for routing. Because the networks are not dense, the number of downlinks are limited. Referring to Figure 4.8, only one of the topologies starts with more than 2 downlinks per node. Automatic rerouting through the use of multiple downlinks seems only applicable to much more dense networks.

The same figure shows that slack has little effect on the cost of reconfiguration. This is counter intuitive to the nature of reconfiguration, which is closely related to initialization. This is probably because cost is also incurred from sending Invalid (INV) packets, and also from nodes falsely replying with UP prior to
realizing that they, too, have lost their sources.

5.4 Load on Nodes

Nodes are expected to carry varying loads depending on certain factors mainly to do with their position in the topology. Nodes close to the base station will carry a larger load; all messages eventually pass through them. Conversely nodes at the periphery of the topology transmit fewer messages as no node uses them to forward messages to the base station. Also some nodes become bottlenecks; many other nodes depend on them for routing. They are links between clusters within the topology.

In profiling nodes, we chose the topology with 500 nodes, range 70. Results are averages from 40 experiments, and in each experiment, 200 nodes were profiled, the network routes 500 data messages, no nodes were destroyed, and the slack was set to 0. We remove the profile of nodes close (up to three hops away) to the base station. We also removed the peripheral nodes, i.e. nodes with no uplinks.

The normal case is good. For the bottom 90% of the population, each node sends an average about 15 packets, with a standard deviation of roughly 8 messages. The bottom 90% send at most 45 messages, three times the average.

However, the top senders send a disproportionately large number of messages. About 3% of the nodes send more than 100 messages. The top 1% send around 200 messages. The average for the top 10% is roughly 90 messages, 6 times larger than the average of the bottom 90%.

It is undesirable for 1% of the node to carry a load 5 times greater than 90% of the rest of the population. This occurs when a large number of nodes depend on a certain node for routing. These nodes are bottlenecks. Due to the large load they carry, most of them will deplete their energy very quickly.

The protocol could be modified to force routing away from these nodes. They could, having detected their excessive load, send out an INV packet, or simply increase their distance estimates. The node will then make sure that it stays on top, and not decrease it distance estimate. The other nodes will be forced to find other, possibly longer, routes. This extra load due to longer routes is bearable, because if those routes are not taken, then burdened node will die quickly, forcing a reroute anyway.

The use of multiple links have been effective in balancing the load. However, having multiple links alone is not enough. Special care must be taken to protect the top 3% of the node population.
Figure 5.10: Load Distribution on Nodes

(a) All Samples
(b) Middle 80%
(c) Bottom 10%
(d) Top 10%
Wireless sensor networks depend on disposable sensors for its operations. Nodes may deplete their limited energy source after they are deployed. Nodes may also fail as they are destroyed in their unsupervised, possibly hostile environments.

We have presented a protocol, SDLR, capable of reconfiguring the network as these faults occur, by adapting the route recovery method in Link Reversal Routing protocols. The protocol is initialized with shortest distance estimates for optimal routing.

The recovery process is also capable of handling partitions. Partitions in wireless sensor networks could be large, making them hard to detect. Partitions in wireless sensor networks are also persistent, until a time when fresh nodes are added.

We have also presented our results from our analysis of randomly deployed network topologies, and studied the results of the protocol. We found a strong relationship between the number of connected nodes, and the initialization cost. We also found that node density, and range affects initialization cost only slightly. We also found a strong relationship between the cost of routing a message from a node to the base station, and the average shortest distance of the network. Using shortest distance estimates the node can route messages through optimal or near optimal routes. We also found that both hopcount and message sending cost decrease as more nodes are added, and as range is increased. We also found that the cost of recovery most closely follows the number of nodes who change their hopcount as a result of node failure. This is unfortunate, as the nodes’ hopcount is sensitive to node failure. This causes the cost of recovery to be quite high.

The merge between link reversal routing techniques for recovery, and shortest distance for initialization has overall been successful for use in wireless sensor networks. With a slack of 2, initialization cost is between two and three, comparable to LMR. Message sending is near optimal, with efficiency between 1.3 to 1.1. The recovery process is not very efficient but the process is distributed, and the cost is dependant on the damage incurred. Finally, the use of multiple paths
has been beneficial in balancing the load on individual nodes. However, in every topology, there will be bottlenecks, and additional measures should be taken to prevent them from being overburdened.

There are a few unresolved problems with the protocol. First is the use of flooding. The simulation model did not account for any collisions between messages, and collisions are frequent in flooding. A true test of the viability of the protocol, is to see its performance under a more realistic simulation model.

Next, is the behaviour of the protocol in extremely dense networks. In these networks, there are more redundant downlinks, and they could better benefit from the link reversal routing recovery process.

Finally, partitions were not studied in depth. More experiments are needed to analyze not only the cost of recovery, but also the the cost of detecting partitions.
APPENDIX A

Original Honours Proposal

Title:    Adding Partition Management and Fault Tolerance to Link-Reversal Protocols for Wireless Sensor Networks

Author:  Ibrahim Abdul Rahim

Supervisor: Assoc. Prof. Amitava Datta

Background

Routing in wireless ad-hoc networks is a complicated issue. There are many conflicting needs, with conflicting priorities. Algorithms must be able to cope with changes in the topology. Communication links are frequently created and destroyed due to the mobility of nodes, limited broadcast range, multipath interference, and changes in shielding factor [3]. Algorithms must also be scalable, tunable, distributed and easy to deploy [8]. Other issues such as cost of deployment, capability and complexity of node, routing without the use of large conventional routing tables, adaptable in front of intermittent functioning regimes, network partitioning and survivability must be addressed by any wireless ad-hoc network routing algorithm [8].

Wireless Sensor Networks has specific needs when it comes to routing. Wireless sensor networks compose of tens of thousands of tiny devices with very limited resources (motes) [11]. The number of nodes is very large number, and it becomes impractical to have a complete and consistent view of the entire topology. Furthermore the motes are have limited resources, in memory, computational power, and energy [11]. Additionally nodes can ‘sleep’ [11], which can affect the topology of the network. In terms of routing, in wireless sensor networks, they must rely on each other to communicate their readings to a centralized collection point [11]. This means that typically there is only a single or a few ‘destinations’ for data to be routed to.
Some of these specific needs can be addressed by a Link Reversal Routing Protocol, such as the Gafni-Bersekas (GB) Algorithms \cite{3}, and Temporally-Ordered Routing Algorithm (TORA) \cite{9}. The algorithm briefly works on the following basis.

An Acyclic Directed Graph (ADG) is destination oriented if for every node there exist a directed path originating at this node and terminating at the destination. Otherwise the graph is destination disoriented. A connected ADG is destination disoriented if and only if there exist a node other than the destination that has no outgoing link. The GB Algorithm then proposes two methods to change a connected destination disoriented graph into a destination oriented graph by changing some of the links, by reversing them \cite{3}.

This algorithm has several desirable properties. It has some redundancy, some additional paths, drastically reducing the frequency of node loosing all routes to its source. It is local, i.e. it does not depend on instructions from a central station. It does not employ flooding, and routes are free of loops (for connected graphs) and it can incorporate new nodes \cite{3}.

The algorithm is useful for Wireless Sensor Networks because, wireless sensor networks have only a few, if not one, ‘destination’, they have a huge number of nodes so only ‘local’ information is practical, and the nodes have limited resources, so a ‘simple’ algorithm is preferred.

Aim

The aim of this project is to address some of the problems in Link Reversal Routing Protocol, for its application in wireless sensor networks. One problem the GB algorithm faces is partitions. The GB algorithm works on the basis that the graph is connected, i.e. no partitions exist in the network. The GB algorithm exhibits instability in portions of the network which become partitioned from the destination \cite{9}. Control and data messages run in loops as long as the partition is there.

Another issue is how to make the GB Algorithm more fault-tolerant. The project will try to embed into the implementation ways to detect faults and recover from them. The project will also try to detect and determine the location of partitions in the network.

Additionally, the project will look at how to make the GB Algorithm more efficient by using some characteristic in wireless sensor networks, either through some hierarchical partitioning, or by using techniques from other algorithms.
The project will also analyze the new protocol with respect to a few key indicators, including fault-tolerance, bandwidth, efficiency, power use, adaptability, and scalability.

To facilitate the analysis, some simulation option will be used, or a simulator will be created. The protocol will be implemented and tested on the simulator.

### Plan

<table>
<thead>
<tr>
<th>Action</th>
<th>Time</th>
<th>Deliverable/Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research problem and prepare for presentation</td>
<td>1 week</td>
<td>A fully prepared presentation [Presentation A]</td>
</tr>
<tr>
<td>Research existing protocols. Look at their implementation, problems, benefits and references</td>
<td>2 weeks</td>
<td>Compile a list of protocols with their properties, and their suitability [List A]</td>
</tr>
<tr>
<td>Research what features are needed to improve the protocol, and problems associated with them</td>
<td>1 week</td>
<td>List of features and improvements that can be implemented on List A [List B]</td>
</tr>
<tr>
<td>Research ways to implement the required features</td>
<td>3 weeks</td>
<td>A implementation description for each feature in List B [Paper A]</td>
</tr>
<tr>
<td>Prepare revised project proposal</td>
<td>1 week</td>
<td>A revised proposal [Proposal A]</td>
</tr>
<tr>
<td>Choose options for a simulator</td>
<td>1 week</td>
<td>List of simulators [List C] and a chosen simulator with reasons why it is chosen</td>
</tr>
</tbody>
</table>

[49]
<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
<th>Description</th>
<th>Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement base protocol into simulator</td>
<td>2</td>
<td>A fully implemented protocol, that is ready for simulation</td>
<td>[Paper C]</td>
</tr>
<tr>
<td>Run simulation and take measurements</td>
<td>2</td>
<td>A set of results of protocol on different benchmarks</td>
<td>[Paper D]</td>
</tr>
<tr>
<td>Find other tweaks and implement</td>
<td>2</td>
<td>An improved implementation of the protocol, that is ready for simulation</td>
<td>[Paper E]</td>
</tr>
<tr>
<td>Simulate new ideas, and take measurement</td>
<td>2</td>
<td>A set of results of protocol on different benchmarks</td>
<td>[Paper F]</td>
</tr>
<tr>
<td>Write, and edit draft dissertation</td>
<td>3</td>
<td>A good draft Honors Thesis [Thesis A]</td>
<td></td>
</tr>
<tr>
<td>Prepare, write, and edit seminar</td>
<td>2</td>
<td>A fully prepared seminar presentation [Presentation B]</td>
<td></td>
</tr>
<tr>
<td>Find other tweaks and implement</td>
<td>2</td>
<td>An improved implementation of the protocol, that is ready for simulation</td>
<td>[Paper G]</td>
</tr>
</tbody>
</table>


Bibliography


