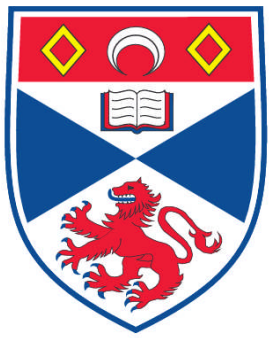
NODE RELIANCE : AN APPROACH TO EXTENDING THE LIFETIME OF WIRELESS SENSOR NETWORKS

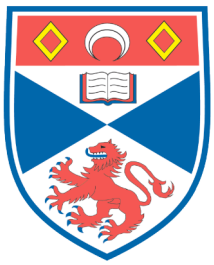
Alan W. F. Boyd

A Thesis Submitted for the Degree of PhD
at the
University of St. Andrews



2010

Node Reliance: An Approach to Extending the Lifetime of Wireless Sensor Networks



A thesis to be submitted to the

UNIVERSITY OF ST ANDREWS

for the degree of

DOCTOR OF PHILOSOPHY

by

Alan W. F. Boyd

School of Computer Science

University of St Andrews

June 2010

”And now for something completely different.” –
Monty Python

Abstract

A Wireless Sensor Network (WSN) consists of a number of nodes, each typically having a small amount of non-replenishable energy. Some of the nodes have sensors, which may be used to gather environmental data. A common network abstraction used in WSNs is the (source, sink) architecture in which data is generated at one or more sources and sent to one or more sinks using wireless communication, possibly via intermediate nodes.

In such systems, wireless communication is usually implemented using radio. Transmitting or receiving, even on a low power radio, is much more energy-expensive than other activities such as computation and consequently, the radio must be used judiciously to avoid unnecessary depletion of energy. Eventually, the loss of energy at each node will cause it to stop operating, resulting in the loss of data acquisition and data delivery. Whilst the loss of some nodes may be tolerable, albeit undesirable, the loss of certain critical nodes in a multi-hop routing environment may cause network partitions such that data may no longer be deliverable to sinks, reducing the usefulness of the network.

This thesis presents a new heuristic known as *node reliance* and demonstrates its efficacy in prolonging the useful lifetime of WSNs. The node reliance heuristic attempts to keep as many sources and sinks connected for as long as possible. It achieves this using a *reliance value* that measures the degree to which a node is relied upon in routing data from sources to sinks. By forming routes that avoid high reliance nodes, the usefulness of the network may be extended.

The *hypothesis* of this thesis is that the useful lifetime of a WSN may be improved by node reliance routing in which paths from sources to sinks avoid critical nodes where possible.

I, Alan W. F. Boyd, hereby certify that this thesis, which is approximately 78,843 words in length, has been written by me, that it is the record of work carried out by me, and that it has not been submitted in any previous application for a higher degree.

date _____ *signature of candidate* _____

I was admitted as a research student in October 2005 and as a candidate for the degree of Doctor of Philosophy in October 2005; the higher study for which this is a record was carried out in the University of St Andrews between October 2005 and June 2010.

date _____ *signature of candidate* _____

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Doctor of Philosophy in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

date _____ *signature of supervisor* _____

In submitting this thesis to the University of St. Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and abstract will be published, and that a copy of the work may be made and supplied to any *bona fide* library or research worker, that my thesis will be electronically accessible for personal or research use unless exempt by award of an embargo as requested below, and that the library has the right to migrate my thesis into new electronic forms as required to ensure continued access to the thesis. I have obtained any third-party copyright permissions that may be required in order to allow such access and migration, or have requested the appropriate embargo below.

The following is an agreed request by candidate and supervisor regarding the electronic publication of this thesis: Access to printed copy and electronic publication of thesis through the University of St Andrews.

date _____ *signature of candidate* _____ *signature of supervisor* _____

Acknowledgements

There are many people without whom this thesis may not have even got off the ground and to whom I shall owe eternal gratitude:

Dharini Balasubramaniam, my main supervisor, for being forced to put up with endless proof reads, barely comprehensible musings and endlessly correcting my use of ”in to” as two words. I am most grateful for her infinite patience, which is more than I deserved.

Alan Dearle and Ron Morrison for acting, at different times, as my second supervisor as well as having steered the DIAS project on which my studentship was based.

My parents, Ann and Ian Boyd, have provided emotional and financial assistance. Their academic backgrounds have been invaluable in seeing things from a different perspective.

Claire Loptson, my wonderful girlfriend, for tolerating my daily rants and me in general and also for making life significantly more bearable than it might otherwise be.

Frank Gunn-Moore and Ali Khajeh-Hosseini for in-car entertainment between Edinburgh and St Andrews and for bravely proof reading my thesis, despite its size.

Angus Macdonald, Jon Lewis and Jamie Smith for tolerating my daily renditions of the Wallace & Gromit theme tune.

Kris Getchell, Andrew Gray, Ben Maydon, David Forester, Dawn Fulton, Juliette Daigre, Gemma McLean, Paula Whiscombe, Keith McDonald, Monika Vorberg, Markus Tauber and Stuart Norcross for welcome distractions and advice.

Lastly, I wish to thank you, the person I inevitably forgot, who still read my thesis to see if they’re mentioned in the acknowledgements. Alas, I was time constrained during my final weeks, and I thank you in advance for your forgiveness.

Published Research

Alan W. F. Boyd, Dharini Balasubramaniam, Alan Dearle, and Ron Morrison.
An approach to extending the lifetime of wireless sensor networks.
In *The 9th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*, pages 123–128, Liverpool, UK, 2008.

Alan W. F. Boyd, Dharini Balasubramaniam, Alan Dearle, and Ron Morrison.
On the selection of connectivity-based metrics for wsns using a classification of application behaviour.
In *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, Newport Beach, California, USA (accepted for publication), 2010.

Alan W. F. Boyd, Dharini Balasubramaniam, and Alan Dearle.
A collaborative wireless sensor network routing scheme for reducing energy wastage.
In *IEEE International Conference on Communications Workshop on Energy Efficiency in Wireless Networks & Wireless Networks for Energy Efficiency*, Cape Town, South Africa, 2010.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless Sensor Networks (WSNs) consist of distributed, autonomous *nodes* that work together to observe some phenomenon of interest by taking sensor readings of factors such as temperature, humidity and radiation. WSNs have been used for many applications, including animal tracking [111][128][18][50], structural monitoring [122][71][17], environmental monitoring [9][114][119] and resource monitoring in offices and homes [62][53]. Data is collected in a WSN by transferring it from source nodes that generate data to sink nodes that collect it. The nodes are battery powered and thus can only operate for a limited period of time. Once the energy has been exhausted from a node, it ceases to function and its loss may inhibit other nodes in the network. In many applications, data is transferred by routing, which may consume the majority of the battery life and potentially render the WSN unusable.

This thesis examines the problem of keeping WSNs as useful as possible for as long as possible. It presents a new basis for routing known as *node reliance*, which assigns a cost to each node based on the degree to which it is relied upon in routing data from sources to sinks. As will be discussed in Section 1.3, in many applications, the usefulness of a WSN at a particular instant is considered to be proportional to the *source diversity* of data that it collects, where source diversity is defined as the number of sources used to provide a set of data.

It is hypothesised that by routing using node reliance, it is possible to achieve a high source diversity for longer than with other routing protocols.

1.1 Wireless Sensor Networks

Historically, WSNs have been characterised as wireless networks consisting of numerous small, energy constrained, low cost, autonomous nodes that are distributed over an area for the purpose of monitoring or sensing [48] [96] [24]. Communication or relaying of data typically occurs via wireless multi-hop routing. The majority of WSNs exhibit a (source, sink) architecture, which may include any number of:

1. *source* nodes, which generate data, usually by using sensors to measure environmental factors such as temperature, humidity or radiation,
2. *sink* nodes, which collect all data gathered by source nodes, and
3. *intermediate* nodes (which may include source nodes) that aid the transmission of data from sources to sinks.

The generation of data at source nodes occurs either proactively or in response to some request. It has been suggested that sink nodes, which are often referred to as base stations, may be high powered [19], linked to databases via satellite links [96] or have more resources than other nodes [49].

Despite the difficulties with limited energy capacities and the need to engage in multi-hop routing in order to collect data, WSNs offer a number of advantages over conventional environmental monitoring systems. In particular it is anticipated that WSNs will be rapid to deploy [49], adaptable [32][69] and self configuring [15][38][11][16], thus reducing the effort required to set up the devices and lowering costs of data gathering.

1.2 Routing Protocols

As indicated in Section 1.1, source nodes typically transmit data to sink nodes through multi-hop routing. In this thesis, routing and dissemination are differentiated by the following definitions:

- *Dissemination protocols* are a set of algorithms in which sources distribute data to every other node in the network. An example of a dissemination protocol is SPIN

[47] in which pairs of nodes negotiate the data that will be exchanged to ensure that only required data is transmitted.

- *Routing heuristics* are a set of algorithms or procedures that perform a *costing* task in which costs are assigned to available paths to indicate how desirable they are and a *selection* task in which a path is selected.

- *Routing protocols* are a set of algorithms or procedures that carry out the functionality of a routing heuristic as well as a *discovery* task in which data regarding the network is collected.

The discovery, costing and selection tasks will be discussed in further detail in Chapter 3.

The use of radio in WSN applications is known to be a heavy user of the battery, even when receiving data [94][24][92]. Given the limited energy reserves of nodes, it is important for dissemination protocols, routing protocols and heuristics to limit any wastage of energy and select paths that will extend the time for which the network remains functional.

As will be shown in Chapter 4, many routing protocols will only work under ideal conditions in which radio transmissions are never lost and the nodes always transmit in a perfectly circular shape. Furthermore, very few routing protocols avoid forming contentious paths amongst sources, i.e. the optimal path from every source may include a common subset of nodes. Those routing protocols that do avoid contention are often wasteful of network resources by e.g. expending energy to avoid contention or using paths that result in excessive battery drain on nodes. If sources do not avoid contention, it is possible that one or more nodes may be exhausted due to overuse and the network may be partitioned. When a partition occurs, sources are disconnected from sink nodes, thus reducing source diversity and potentially the total data transferred from sources to sinks.

1.3 Motivation and Problem Statement

Certain applications benefit from having a high source diversity for as long as possible, since this leads to an increased resolution of data and allows a domain expert to extract more information from the available data.

There are several examples [111][128][81][114][71][119] in the literature of WSN applications that benefit from receiving data from a variety of sources for as long as possible. For example, CenseMe [81] is a social-networking WSN application, designed to operate on mobile phones, which act as source nodes. The application infers what a user is doing (and with whom) by means of the microphone, GPS receiver, accelerometer and bluetooth receiver. For example, the bluetooth receiver and accelerometer may allow the inference that Alice is walking with Bob. The WSN remains functional even if individual sources are switched off. However, as the number of concurrently operating sources increases, it is possible to get more information from the data that is received from them. In the above example, if Bob deactivates his source, it is only known that Alice is walking. Conversely, if Jon switches his phone on, it becomes known that Jon, Alice and Bob are all walking together.

Consider the network shown in Figure 1.1. It consists of nine nodes (A-H and Z) of which two are sources (A and D) and one is a sink (Z). An edge between two nodes indicates that those nodes can communicate with each other. Within the network, it is obvious that certain nodes are more important to maintaining (source, sink) connectivity than others. For example, the loss of sink Z renders the network useless. The loss of node C makes source A useless, but allows source D to continue operating. Finally, the loss of node E or H has no effect.

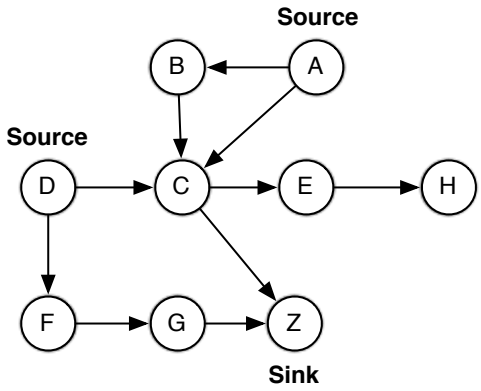


Figure 1.1: An example network with with 9 nodes, including 1 sink and 2 sources

The problem is to find a routing protocol that will create as great a source diversity for as long as possible, i.e. it will keep as many sources connected to sinks for as long as possible. In most routing heuristics, each source determines the optimal path for routing to a sink. For example, the optimal routes to the sink might be ACZ from source A and DCZ from

source D. However, both of these routes require the use of node C, a node that is essential for the operation of source A.

A cooperative scheme may require source D to use the non-optimal route DFGZ, thus reducing the energy expenditure of node C and allowing both sources to remain connected simultaneously for longer.

1.4 Assumptions

The following assumptions are made for the purpose of this thesis:

- Battery drain from computation is negligible compared to that from radio.
- Communication in the network may be imperfect.
- Nodes are immobile.
- The energy on each node is limited and cannot be replenished.

These assumptions will be examined in further detail in Chapter 2.

1.5 Approach: Node Reliance

This thesis presents a new family of routing heuristics known as *node reliance*, which aim to maintain a high source diversity for as long as possible, i.e. to keep sources connected to sinks. The node reliance heuristics are later extended into routing protocols for use in WSNs.

The premise of node reliance routing is that each node is assigned a weighting, which gives an indication of how much that node is relied upon in routing from sources to sinks. The routing protocol will preferentially avoid using routes containing nodes of high reliance. Node reliance acts cooperatively in that sources determine routes based on a holistic view of the network avoiding the use of nodes upon which others rely.

It is hypothesised that by routing using node reliance, it is possible to achieve a high source diversity for longer than with other routing protocols.

1.6 Contributions

This thesis makes four original contributions:

- A modularised view of routing protocols,
- A family of new routing heuristics and routing protocols named *node reliance*,
- A new metric named *Connectivity Weighted Transfer* (CWT) that may be used to measure source diversity over time, and,
- An experimental analysis of routing heuristics and protocols and their suitability in different operating conditions to maintaining source diversity.

The modularised view of routing protocols demonstrates the way in which the behaviour of many routing protocols can be separated into a number of different *modules* each with different objectives, features, advantages and disadvantages. Using such a view makes it possible to compare routing protocols and to construct new ones to meet particular application features by assembling modules.

Node reliance is the new family of routing heuristics and routing protocols that aim to maintain source diversity for as long as possible in a WSN.

CWT is a new metric that is introduced in order to measure source diversity over time. Source diversity relies on connectivity between sources and sinks. A common measure of connectivity is the total number of bytes transferred from sources to sinks. However, this measurement does not distinguish whether the data is received from a single source or whether it comes from many sources, the latter of which is preferred. CWT measures the total data transferred, but the score is biased. A user-defined *weighting factor* can be used to indicate whether it is preferable to have:

- many sources connected for a short time, or

- few sources connected for a long time.

The ability to bias numerous connections over a possibly shorter period of time is important, since this thesis is interested in maintaining source diversity for as long as possible.

Finally, an experimental analysis of routing heuristics and routing protocols, including the node reliance family and third party approaches is carried out. Measurements include both CWT and the more commonly used total data transfer in order to establish under which conditions each routing protocol is able to best obtain source diversity for as long as possible.

1.7 Thesis Structure

The thesis is structured as follows:

Chapter 2 discusses WSNs and justifies the assumptions made in this thesis regarding their capabilities and restrictions. Chapter 3 presents the modularised view of routing protocols. Related work, including common routing protocol paradigms and their suitability to the problem solution are discussed in Chapter 4. Chapter 5 deals with measuring source diversity in WSNs. Specifically, this chapter covers different metrics, including the new CWT metric as well as an experimental methodology for measuring the source diversity provided by using different routing protocols. The new node reliance heuristics are proposed in Chapter 6 and analysed in Chapter 7. The node reliance routing protocols are proposed in Chapter 8. The routing protocols are analysed under perfect conditions in Chapter 9 and under realistic conditions in Chapter 10. Finally, conclusions and further work are discussed in Chapter 11.

Chapter 2

Wireless Sensor Networks

This chapter examines the characteristics of WSNs and justifies the assumptions made in Chapter 1. The literature is inconsistent regarding the capabilities and characteristics of WSNs. Often, the description or assumptions of the system are unspecified. These problems make it difficult to analyse a WSN system and find solutions to many of the problems they face. In this chapter, several assumed WSN characteristics from the literature are analysed in order to develop a definition of WSNs that will be used consistently in the remainder of the thesis.

2.1 Overview

Historically, WSNs have been characterised as wireless networks consisting of numerous small, energy constrained, low cost, autonomous nodes that are distributed over an area for the purpose of monitoring or sensing [48] [96] [24]. Communication or relaying of data typically occurs via wireless multi-hop routing. The majority of WSNs described in the literature exhibit a (source, sink) architecture, which may include any number of:

1. *source* nodes, which generate data, usually by using sensors to measure environmental factors such as temperature, humidity or radiation,

2. *sink* nodes, which collect the data gathered by source nodes, and

3. *intermediate* nodes, which may include source nodes, that aid the transmission from sources to sinks.

The generation of data at source nodes may occur either proactively or in response to some request. It has been suggested that sink nodes, which are often referred to as base stations, may be high powered [19], linked to databases via satellite links [96] or have more resources than other nodes [49].

WSN nodes are typically battery powered and for reasons expounded upon in Section 2.5.4, the energy of a node is used to refer to its remaining battery power within this thesis. The energy capacity of a battery is dependent on its size and since nodes are expected to be small, the batteries are unlikely to be of high capacity. It has been suggested [33] that battery depletion is one of the key challenges experienced in developing and working with WSNs, particularly since every operation performed by the node requires expenditure of energy [31]. While other resources such as the CPU, memory or storage may be immediately re-used when released, the same is not true of the battery. Unless a node has some means of energy replenishment, which is discussed in Section 2.5.4, the capacity of batteries restricts both the maximum lifetime of nodes and the frequency with which the node can carry out particular actions.

Beyond these characteristics, it is difficult to provide a formal definition of the exact capabilities of a WSN, particularly due to the increasing number of scenarios making use of the technology [96]. It has been theorised that, with WSNs typically being application dependent, it will never be possible to create a single architecture, which can be used in all applications [59]. Without a single architecture being defined, it may be impossible to precisely define the characteristics of a WSN. Römer has created a design space [96] that discusses the large number of dimensions in WSN deployments, including:

- deployment type,
- node mobility,
- node size,
- node cost (in dollars),
- available energy resources,

- heterogeneity of nodes,
- method of wireless communication,
- infrastructure,
- network topology,
- sensor coverage,
- connectivity,
- number of nodes,
- estimated running time, and,
- quality of service.

Most of these categories are self-explanatory. The *deployment type* refers to the method by which nodes are physically deployed, i.e. whether nodes are deployed *one time* or whether they are *iteratively* replenished and whether their placement is *random* or *manual*. *Infrastructure* specifies how routing occurs within the network; possible values are *ad hoc* in which nodes may communicate with each other and *base station* in which nodes may only directly communicate with a base station. The *network topology* affects how nodes are interconnected, i.e. which nodes may communicate with each other. *Sensor coverage* reflects the density of source nodes, i.e. *sparse* or *dense*. In an extreme case, the sensor coverage may be *redundant* so that multiple sources cover the same area. The *connectivity* of a network indicates the frequency with which any two nodes may communicate. For example there is always a path between any pair of nodes in a *connected* network. In an *intermittent* network, pairs of nodes may be occasionally partitioned. Finally, if nodes are usually isolated but occasionally come into contact with each other then the network is said to have *sporadic* connectivity. *Quality of service* requirements include any constraints that may be placed on the network. Examples include the requirement that data must be received by a sink within some period of it being generated by a source, and that the network must remain operational with a certain degree of node loss.

The survey carried out by Römer shows that the majority of deployments considered involve tens of battery-powered nodes that are manually placed and communicate via radio. However, other aspects of their behaviour vary greatly.

2.2 Relationship to Wireless Ad Hoc Networks

Römer's design space [96] indicates that the infrastructure of certain WSNs may be ad hoc, i.e. nodes may freely communicate and route through each other. Since communication in a WSN also takes place wirelessly, the field of *Wireless Ad hoc Networks* may contain relevant literature to the field of WSNs. In particular, research involving wireless communication, routing and resource management is of interest to both fields and so may be relevant to either domain.

Even though the literature does not consistently attribute any other features to wireless ad hoc networks, it has been suggested [120][98] that there are substantial differences between WSNs and ad hoc networks. For example, Royer [98] suggests that WSNs are deployed for a specific task, whereas wireless ad hoc networks are a result of mobile users joining a stationary network using mobile devices such as phones. However, such a distinction is inadequate, since nodes in a WSN may be dynamically reprogrammed [32], thus changing the task for which they were deployed. Furthermore, if sources are mobile and sinks are static, then a WSN may consist of a stationary network in which mobile nodes join and leave the network. Such behaviour is also covered in Römer's design space, in which connectivity may be sporadic, sources may be mobile and sink nodes may be static.

In the literature, there are many examples of work that was designed for wireless ad hoc networks and is readily referenced in work on WSNs. For example, AODV [89] was initially designed for a wireless ad hoc network and yet is commonly referenced and used as a basis for comparison in work on WSN routing protocols [12] [103] [106] [79] [117] [10]. Similarly, the GPSR [60] or DSR [54] routing protocols, frequently referenced in the WSN literature, were initially designed for wireless ad hoc networks.

Due to the overlap of these areas, research that is aimed at wireless ad hoc networks is included alongside work on WSNs.

2.3 Advantages

Traditionally, environmental monitoring would be carried out using networks of wired sensors [69], satellites, airborne sensor systems or small numbers of devices equipped with

cameras and microphones [18]. WSNs offer a number of advantages over these systems, such as being:

- self configuring and adaptable, [18],
- quickly deployable [49],
- low cost [49][8], and
- usable in inhospitable areas [33].

These advantages are described in more detail below.

WSN nodes are envisioned as being able to configure themselves [49] in order to satisfy application goals. Once nodes are deployed, routes from sources to sinks are automatically found and used. Source nodes whose sensing areas overlap can schedule the times at which they are active to minimise data duplication and maximise operational time [15][38][11][16]. Adaptability allows the network to respond to the expiration of nodes as well as to make use of new nodes that are added to the network [49]. A user may also adjust the behaviour of a WSN. For example, the user may be able to control what data the network collects. In an extreme case, users can dynamically reprogram nodes to change the way in which they operate [32][69].

WSNs are fast to deploy, since nodes are self-configuring and need not be wired together [49]. Furthermore, since a WSN can adapt to the addition of nodes, it is possible for a network to be incrementally deployed at the convenience of the operator [49].

It has been suggested that, since nodes are small and individually have little processing power, as the number of manufactured nodes increases, the cost of nodes will drop perhaps to less than a dollar each [49][8]. Costs can also be reduced by the lack of wiring required to connect nodes and by the improved deployment time, which can reduce money spent on human effort.

The final advantage of a WSN over traditional systems is the ability to deploy the system in an inhospitable environment. Since a WSN is potentially adaptable and self-configuring, there is no need for prolonged human intervention in running the network. It may therefore be possible to deploy the network in an otherwise hostile environment such as a battlefield.

2.4 Example Deployments

In order to give examples of WSN usage, this section discusses a number of deployments that feature prominently in the literature. For each deployment, details of the number of nodes that constitute the deployment, a summary of the data collection mechanism, and the WSN's resilience to source loss, which reflects how the expiration of sources affects the system, are provided. These features can be used to express the data collection behaviour of an application and so can be used to summarise how the deployment works and under what conditions it remains useful.

Since many proposed WSN applications have been shown to be impractical or unfeasible [112], only those applications that have been physically deployed or deployed in a test environment are considered. Applications that have only been proposed or theorised are not discussed.

One of the best-known WSN deployments is the habitat monitoring of Great Duck Island (GDI) [111]. 150 source nodes were placed on the island. Two types of source were used: burrow nodes to monitor the nesting burrows and weather nodes to measure the ambient weather conditions of the island. The application did not require all sources to be alive and nodes expired at various times throughout the 120-day deployment.

ZebraNET [128] used a WSN to track the locations of zebras in Kenya. The deployment of nodes called for 30 specially built collars to be attached to zebras. Each collar acted as a source node and contained solar panels, a small battery, a radio and a GPS receiver. GPS data was periodically gathered and stored at each source. Unlike many other deployments, ZebraNET used a *dissemination protocol* in which any data collected by a source was distributed to all other sources in the network. Since the application gathered the positional data of each zebra, it remained operational even if some source nodes expired. However, the number of correlations between zebra positions would be reduced.

Wisden [122] is a system for structural monitoring in which source nodes are manually placed on a structure, such as a bridge, to monitor vibration levels using sample rates of around 100 Hz. All data is stored at source nodes and periods of interesting activity i.e. consecutive data samples that differ and lie above some threshold, are forwarded to a sink. The authors note that Wisden is intolerant to data loss and consequently all sources must remain active for Wisden to function.

CenseMe [81] is a social-networking WSN application, which is designed to operate on mobile phones (specifically, the Nokia N95). The application takes periodic readings from the accelerometer, microphone and GPS receiver. It also takes periodic random photos using the built-in camera and uses the Bluetooth receiver to detect other nearby CenseMe users. Combining all of this information, the application attempts to infer what the user is currently doing. For example, by using the microphone, the application may determine that the user is engaged in conversation. Similarly, the accelerometer may determine whether the user is standing, sitting, walking or running. In the application's test deployment, 22 people ran the CenseMe application as they went about their daily lives. Since users could simply switch their phones off at any time, it is concluded that CenseMe is capable of operating with source loss, albeit with reduced functionality.

A system for analysing the microclimates surrounding redwood trees has been produced by Tolle [114]. In Tolle's deployment, 33 source nodes were placed approximately 2m apart on a single tree between 15 metres and 70 metres off the ground. Each source generated and stored periodic data based on temperature, humidity and radiation readings before routing it to a sink at the bottom of the tree. The author notes that analysis remains possible with source loss and so hence, not all sources are required for the application to be functional.

NAWMS [62] provides fine granularity detail regarding water usage in homes. By placing a source node on each water pipe and a sink node at the water meter (an accurate device that measures water flow through the entire building), NAWMS can calibrate pipe vibrations against water flow. Once the system is calibrated, it provides real-time data regarding water usage for each device. It is estimated that a house with 3 bedrooms and 2 bathrooms may require between 17 and 21 sources. For such a system to be correctly calibrated, every source would have to be operational. If individual sources failed then the water meter's measurement would not correspond with the monitored vibration of the pipes. Thus the loss of any source would appear to render the system unusable.

PermaSense [9] aims to gather environmental data regarding permafrost in the Alps for at least 3 years. Approximately 25 sources form a wireless network in which environmental data is periodically gathered, stored and routed to a sink. The project has tight constraints regarding the loss of data; specifically, 99% of data must be recovered and no more than 10 consecutive points of data may be lost. Consequently, the loss of even a single source would violate this requirement and render the application unusable.

SASA [71] (the structure-aware self-adaptive WSN system) is an application to monitor the structural integrity of mines. One particular test deployment used 27 source nodes and a single sink to monitor a tunnel. Sources were mounted to the walls, ceiling and floor. Each source monitored its neighbours (i.e. the closest six sources) by the periodic exchange of *beacons*. If, during some time period T, two of a source's neighbours were to change, the network would infer that a collapse had taken place and send a report to the sink. Each source was pre-programmed with its location in the tunnel, and consequently it was possible to determine the size and location of any hole that was formed. SASA has a limited resilience to source loss. If too many sources are lost in one area or in a short space of time, the ability to detect collapses may be compromised.

Werner-Allen has used WSNs for monitoring the Reventador volcano in Ecuador [119]. 16 source nodes used seismometers and microphones sampling at 100 Hz to carry out measurements. If a source detected a seismic event, it would forward its readings to the sink. The sink then queried other sources for readings during that time period to gain a complete view of all sources during possible seismic events. A small number of sources were lost during the operation of the network, leading to the conclusion that the application remains operational with source node loss.

Ceriotti has deployed a WSN to monitor the health and deformation of the medieval Torre Aquila tower located in Trento, Italy [17]. The deployment consisted of 16 sources, plus a sink node. Sources measured either deformation of the structure, vibrations or temperature and forwarded readings towards the sink every period of time. Several forms of analysis were carried out such as structural deformation versus temperature. It was therefore necessary for certain numbers of each type of source node to remain active in order for all analysis to be carried out.

2.5 Assumptions

In Section 1.4, the following assumptions are made for the purpose of this thesis:

- Battery drain from computation is negligible compared to that from radio.
- Communication in the network may be imperfect.

Table 2.1: Current drawn by the TMote sky in different scenarios

Activity	Nominal Current (mA)
Radio receiving, MCU[a] on	21.8
Radio transmitting, MCU on	19.5
Radio off, MCU on	1.8
Radio off, MCU idle	0.0545
Radio off, MCU standby (low power)	0.0051

[a] Although not defined in the document, it is believed that MCU refers to the micro controller unit, i.e. the processor.

- Nodes are immobile.
- The energy on each node is limited and cannot be replenished.

These assumptions are justified in the following sections.

2.5.1 Battery Drain from Computation

This thesis assumes that the battery drain from carrying out computation is negligible when compared to that caused by usage of the radio.

A radio can be a major cause of battery depletion in nodes [94], whether it is transmitting or receiving [92]. The TMote sky [101] is a recent, generic, commercially available WSN platform. Table 2.1 shows how much current is drawn by the TMote sky in particular scenarios according to the TMote sky data sheet [101].

The table reveals two interesting facts.

Firstly, it shows that using the radio to receive and transmit draws a large current but also that the current drawn for receiving is more than the current drawn for transmitting. Thus, it is unreasonable to disregard energy expenditure in receiving transmissions.

Secondly, even if the radio's usage of the processor is disregarded (21.8 - 1.8 = 20 mA for the radio alone), the current used when receiving data on the radio is 11.1 times higher than the current used by the processor (1.8 mA). These figures also assume that the processor is running at full speed. If the processor only executes a small number of instructions, as

it would when sampling from a sensor and then forwarding the result to the radio, then the processor will be mostly idle. Thus, the current used when receiving data on the radio may be 3922 times higher than that used by the idle processor (0.0545 mA). If the processor enters a low power mode rather than merely remaining idle, the current would be expected to be even lower.

These numbers suggest that the battery depletion due to radio use is significantly higher than that of using the processor. Given that the onboard processor of nodes is relatively slow, it is more likely that a complex computation will be impractical by reason of its long running time rather than the energy it is likely to use. It is therefore reasonable to assume that the depletion of the battery caused by computation is negligible when compared to the depletion caused by using the radio.

2.5.2 Imperfect Communication

In a WSN, nodes communicate with each other wirelessly and usually via low powered radio. The use of wireless communication has a number of distinct challenges or features that are not present or less problematic in wired communication.

Firstly, communication between a pair of nodes may be unreliable, i.e. packets may be lost. Secondly, communication may be unidirectional, i.e. node A can transmit to node B, but node B cannot transmit to node A. These problems are discussed in more detail in the following sections.

2.5.2.1 Unreliability

Wireless communication is unreliable and there is always a small probability that a packet will be lost [109]. In this section, the cause of the unreliability and its immediate consequences are discussed.

In wireless communication, a radio signal is transmitted with a certain power. As the signal travels through the air, it attenuates (degrades) and may be absorbed by particles in the air, physical obstacles and the ground. For a low-lying antenna that transmits a signal across a short distance (e.g. 100 metres), the attenuation is proportional to the fourth power of the

distance travelled [92] [14]. For example, a signal that travels a distance d will degrade by a factor of d^4. If the distance is doubled, the signal's quality will decline by 16 times (2^4).

It is often assumed that the relationship between the transmission power and distance is d^2 rather than d^4 [46]. In either case, transmitting a signal across a bigger distance will cause a bigger degradation of the signal and so transmissions must be made with high power to ensure they are received. WSNs that are deployed over large areas must therefore either use very large amounts of energy or must engage in multi-hop routing in order to communicate.

The probability that a node A receives a transmission from B is proportional to the difference between the received signal strength of B's transmission at A and the sensitivity of A's radio. If the sensitivity of A's radio is exactly equal to the signal strength then the transmission is lost with probability 1. Signal strength is affected by the power with which the transmission is made, the attenuation of the signal, background noise and interference from other radio transmissions. The latter two factors are random, and thus there always remains a small probability that a transmission may be lost.

Complicating the problem is the fact that although attenuation is approximately proportional to the 2nd or 4th power of distance, it is not uniform.

If it is known that a packet has been lost, it is possible to transmit it again. However, confirming packet reception at each hop may increase the number of transmissions being made by nodes and thus drain batteries more quickly. Furthermore, in order to receive confirmation that a packet has been received, the links between nodes must be bidirectional, which may not be the case as will be discussed in Section 2.5.2.2.

It has been suggested that routing along paths of high stability may lower energy consumption [39][40][120] since fewer retransmissions might take place. However, it is debatable whether stable links are better formed using paths of few hops or many hops.

Woo [120] suggests that paths of many reliable links are better than paths of few unreliable links. Paths with few hops result in nodes that are geographically distant. Since attenuation is dependent on distance between nodes, paths with few hops are likely to involve links that are unreliable and may have to be retransmitted, expending additional energy. If nodes are geographically close then the transmission energies of nodes may be vastly reduced since there is no reason for them to transmit such great distances. However, such a reduction in transmission power would also lower the probability with which each transmission is

received.

Conversely, Haenggi [39] [40] has suggested that long hops formed as a result of an increased transmission power have a higher signal to interference and noise ratio and so are more likely to be received. Haenggi also observes that paths of fewer nodes mean that there are fewer places in which a path can fail.

In conclusion, it should not be assumed that wireless communication is always successful. It also remains unclear whether paths with geographically distant nodes are more or less reliable than paths with numerous nodes that are close together.

2.5.2.2 Unidirectional Links

Wireless communication is not necessarily bidirectional. If a node A can send a message to a node B, there is no guarantee that node B can transmit to node A. Unidirectional links may occur because one node is capable of sending a higher energy transmission. It may also be the case that the geographical location of the nodes makes communication unidirectional. For example, nodes on a hill may be able to transmit to nodes in a valley, but not vice-versa. Nodes may achieve bidirectionality by performing a limited local flood, i.e. using other neighbouring nodes to transmit from node B to node A. However, there is no guarantee that such a path would exist and energy would be drained as a result of trying to find or use such a path.

One way to encourage bidirectional links is to place nodes close together, so that the probability of successful transmission is high in both directions. However, distributing nodes in this manner will result in a highly dense network, which may not be suitable for many applications. For example, more nodes will be required to cover the same area, thus increasing the cost of the network and potentially increasing scalability problems. Each node will also be in transmission range of more nodes, thus increasing the number of messages received and overheard and therefore causing its energy reserves to be depleted more quickly.

In conclusion, it is impractical to assume that links between nodes are bidirectional, since doing so limits the networks in which an application can operate. Even if it is possible to discard unidirectional links, the application's efficacy may drop as a result of not fully utilising all links in the network. In a worst-case scenario, the network may be treated as unconnected if it is entirely made up from unidirectional links.

2.5.3 Mobility

This thesis assumes that nodes are immobile for two reasons:

1. The majority of networks use immobile nodes.
2. If mobile nodes were permitted, the problem of keeping sources connected to sinks becomes significantly harder to solve.

These reasons are explained in more detail below.

The majority of deployments studied in Section 2.4 use stationary nodes. The two major exceptions are ZebraNET [128] and CenseMe [81], in which nodes are attached to animals or people and can therefore freely move around. SASA [71] detects structural changes in mining shafts by the sudden change in position of nodes. However, the nodes are not considered to be mobile since many nodes change position once, simultaneously. Thus, the change in node positions is more like a reconfiguration of the network topology than a node position being under constant change.

The second reason for assuming immobility of nodes is that there may be no universal solution to maintaining high source diversity for long periods of time if the topology is consistently changing. Since the movement of nodes may be random, the importance of any node may change at any time. Nodes that were vital may become irrelevant and irrelevant nodes may become essential to routing. The connectivity of the network is therefore based on a largely random factor and the success of any approach at maintaining (source, sink) connectivity over time is dependent on that random movement.

2.5.4 Scavenging and Replenishment

It is assumed that replenishment of batteries by scavenging energy from the environment is not possible. Although several approaches have been suggested for energy scavenging, such as via solar cells [31][56] or nearby vibrations [97] there are two limitations, which greatly restrict the scenarios in which scavenging can take place:

1. Scavenging can only take place in particular locations and may require large nodes.

2. Nodes are restricted as to when they may expend energy.

These restrictions are summarised below.

Firstly, scavenging can only take place when there is some source of plentiful energy available. For example, if nodes are placed within buildings, pockets or away from direct sunlight, e.g. a forest, the nodes may be unable to gather sufficient energy for the applications to operate. Even indoor lighting may be insufficient, since it produces 2500 times less energy than can be achieved with direct sunlight [97]. Similarly, many applications would require a reliable, consistent and nearby source of vibrations in order to operate, which may not be available. A further problem is that the amount of energy that can be scavenged is dependent on the size of the node, with bigger nodes scavenging more energy [97]. However, as stated in Section 2.1, WSN nodes are generally perceived as being small. Thus, the amount of energy that can be scavenged is also likely to be limited.

Secondly, nodes will be greatly restricted in their ability to expend energy. For example if a node relies on scavenged energy, such as solar power, it may be unable to respond to events of interest early in the morning when its energy levels are low. Similarly, if several cloudy days pass then the node may be unable to recharge its energy supply and the node may be forced in to a low power mode until the sun comes out. Thus, the environment that supplies the energy restricts the operation of the network.

These limitations do not rule out the possibility of energy scavenging. However, there are clearly many applications where it is not appropriate and thus our assumption that energy scavenging is not present is reasonable.

Since a node's energy is limited to its battery power, this thesis uses the term *energy* to refer to energy that is provided by the battery.

2.6 Summary

This chapter presents a definition of WSNs and justifies the assumptions that are presented in Chapter 1. It is argued that WSNs consist of numerous autonomous immobile nodes of finite battery power. Source nodes use onboard sensors to sense or monitor the local environment and generate data based on sensor readings. Data is transmitted unreliably

through intermediate nodes (that might include other sources) towards a sink node which may be mains powered or may have more capabilities than other nodes in the network such as satellite links or more powerful processing capabilities.

This chapter provides a basis on which to discuss WSNs. Having justified their capabilities and limitations; it is possible to find a solution to the problem of routing to achieve high source diversity for long periods of time. Chapter 3 presents a new modularised view of routing protocols, which is used in Chapter 4 to compare different routing protocols in the literature and their ability to solve the problem of maintaining source diversity for as long as possible in a WSN.

Chapter 3

A Modularised View of Routing Protocols

In Section 1.2, a routing protocol is defined as a set of algorithms or procedures that carry out a discovery task in which available paths are discovered, a costing task in which costs are assigned to the available paths and a selection task in which a path is selected for use in routing. Similarly, a routing heuristic is defined as a set of algorithms or procedures that only carry out a costing and selection task. This chapter shows how the behaviour of a routing protocol may be represented in terms of these three tasks. The variations in the behaviour of these tasks may in turn be expressed by a number of modules, each with different options, advantages, disadvantages and requirements. Tasks and modules are not confined to any order and may even be carried out simultaneously.

The purpose of this chapter is to provide a framework for comparing the operation of different routing protocols. The framework may also be used to construct a new routing protocol, subject to user-defined criteria by assembling modules together. Unlike other categorisations of routing protocols [129] [73], the view presented in this chapter is less concerned with application specifics such as the number of sinks and mobility of nodes and more with how routing protocols work, e.g. how paths are costed and the advantages/disadvantages of particular ways of calculating path costs.

The discovery task, and the modules that are a part of the discovery task are discussed in Section 3.1. Similarly, the costing and selection tasks and the corresponding modules that

are a part of those tasks are discussed in Sections 3.2 and 3.3 respectively.

The modularised view of routing protocols that is presented in this chapter is used to discuss routing protocols from the literature in Chapter 4.

3.1 Discovery

During the discovery task, paths from sources to sinks are found. The discovery task may be further broken down into four modules. *Initiator* refers to the node or set of nodes that begin finding a (source, sink) path. *Preparation* indicates whether nodes form paths proactively or whether paths are formed in response to some demand. The *search method* module describes the way in which the initiator finds paths. Finally, *table entries* defines what routing data is stored on each node for each (source, sink) path.

3.1.1 Initiator

The initiator of a routing protocol is the node, which begins the operation to find a suitable (source, sink) path for routing.

3.1.1.1 Source

In a source initiated routing protocol, source nodes are responsible for forming and selecting paths to sink nodes.

Requirements:

- Some search methods, such as enumeration or distance vector, which are discussed in Section 3.1.3, discover paths by flooding from source to sink which causes the paths to be found by the sink nodes. It may therefore be necessary to have a second phase in which either the path is returned to the source or intermediate nodes are notified of the next hop to use, to forward messages towards a sink.

3.1.1.2 Sink

In a sink initiated routing protocol, sink nodes begin the process that forms (source, sink) paths. Such routing protocols typically discover paths by finding the (sink, source) paths and reversing them.

Advantages:

- Sink initiated routing is well suited to applications that are based on publish/subscribe mechanisms. In such systems, the sink, which may be controlled by a user, makes a query for particular data. A *subscription* request is flooded through the network, and those sources that can provide the requested data *publish* it back to the sink node.

Requirement:

- Links between nodes must be bidirectional in order for (sink, source) paths to be reversed and form valid (source, sink) paths.

3.1.1.3 Intermediate

In an intermediate initiated routing protocol, the process of finding a (source, sink) path begins with intermediate nodes, which may include source nodes.

Requirements:

- Not all intermediate nodes are sources. Therefore an intermediate node may not necessarily know when data is ready to be routed and so paths must be formed proactively rather than reactively (both terms are defined in Section 3.1.2).

3.1.2 Preparation

If a routing protocol forms a path only when it is required, the routing protocol is said to be *reactive*. Alternatively, if paths are formed before they are required, the routing protocol is said to be *proactive*.

3.1.2.1 Proactive Routing

In proactive routing, paths from sources to sinks are formed before any demand for them exists.

Advantages:

- There is no delay between requiring a path and being able to use it.

Disadvantages:

- A path may be proactively found but expire before it is used, causing the unnecessary expenditure of energy.

3.1.2.2 Reactive Routing

Reactive routing is the process by which paths are formed only when demand for a path exists.

Advantages:

- Energy is only expended in forming paths that are actually used.

Disadvantages:

- Some delay may be experienced between a path being requested and it being available.

3.1.3 Search Method

Search method is the process by which the initiator discovers the (source, sink) paths it may use in order to route data. The search may take place using *local knowledge*, i.e. only knowledge of itself or its immediate neighbours, *distance vector* in which a node may discover paths using the routing information of its neighbours, *global knowledge* in which

all nodes gather a view of the entire topology by sharing their neighbour lists with all other nodes in the network or by *enumeration* in which all acceptable paths are enumerated, starting at the initiator and being collected at the destination.

3.1.3.1 Local Knowledge

Local knowledge as a search method means that a node only uses data that originates entirely from itself or its neighbours. For example, local knowledge of node A includes the geographic position of its neighbours. However, it excludes node B's knowledge of a node C which is closer to a sink but not a neighbour of A, since that knowledge originates outside the region of A and its neighbours.

Advantages:

- Overhead due to network changes is minimal. If data regarding a node changes, only that node's neighbours must be notified, thus reducing energy consumption.

Disadvantages:

- Knowledge of distant nodes is not obtainable. For example, it is not possible to know that a neighbour's neighbours are all unsuitable for routing.

Requirements:

- For a node A to use state information of its neighbours, those neighbours must be able to transmit to A. For A to use a neighbour as a next hop to a sink, node A must be able to transmit to that neighbour. Thus, links between nodes must be bidirectional.

3.1.3.2 Distance Vector

In distance vector routing, each node shares its routing table, which is a table consisting of the next hop and total cost for each destination [41], with its neighbours. When a node's routing table changes, it is obliged to inform its neighbouring nodes. As a consequence

of this, the neighbouring nodes may also update their routing tables, causing a change to cascade through the network.

If the routing protocol is *source initiated*, the discovery often consists of two phases. In the first phase, a flood is sent from source nodes. A *backpath* is set up in which each node notes the neighbour from which it received the flood. In the second stage, routing tables are exchanged by only a subset of nodes, e.g. those that are most direct from the sink to the source that requested the path.

Advantages:

- Distance vector routing protocols are efficient, because only routing information is exchanged. There is no need to forward details of every link or node in the network.

Disadvantages:

- Only the routing information of neighbours is available to form paths. The nodes that lie on a path to a sink (for example) are not known.

Requirements:

- Distance-vector routing generally assumes bidirectionality of links so that a node may route to a neighbour from which it received routing data.

3.1.3.3 Global Knowledge (GK)

Using global knowledge, each node exchanges its neighbourhood list with all other nodes in the network. Thus, each node builds its own view of the network topology in order to determine (source, sink) paths. This method of gathering network topology is most commonly used in the link state advertisement (LSA) routing protocol [41].

Advantages:

- Alternative paths can be quickly calculated, since a copy of the network topology is available at the node.

- Bidirectionality between nodes is not required.

Disadvantages:

- The growth rate of the number of messages exchanged in global knowledge is $O(n^2)$ where n represents the number of nodes in the network. This is not scalable and thus, for very large networks, it may be inefficient.

Requirements:

- There must exist a route from any node to any other node. Otherwise it is not possible to learn about certain nodes in the network.

3.1.3.4 Enumeration

When enumeration is used to gather network topology, one message is sent along each path of interest to the routing protocol in the network from the initiator to the destination. By analysing the set of messages that arrive at the destination node, it is possible to determine the full set of (source, sink) paths.

Advantages:

- A message must be sent along a path in order for that path to be detected. It is therefore likely that messages will be lost when they are sent along paths of low stability, i.e. unstable paths are less likely to be detected than stable paths and so are less likely to be used in routing.
- In sparse networks, there is less connectivity between nodes and therefore a smaller number of paths. Thus, less overhead is experienced as a result of gathering the set of available paths compared to global knowledge.

Disadvantages:

- In large networks, there may be a prohibitive number of paths causing a large amount of energy to be expended to locate all paths.

3.1.4 Table Entries

The table entries module indicates how the available (source, sink) paths are stored in the network. There are three possible choices. *Source full path* indicates that a path is only stored by the source at the head of the path. *All full path* means that each (source, sink) path is stored on each node that lies on that path. *Next hop* means that each node on a path knows only the next hop of the path. These options are discussed in further detail in the following sections.

3.1.4.1 Source Full Path

When source full path is used for the table entries module, each (source, sink) path is held only at the corresponding source of the path. In order to route data, it is necessary to include the path as part of outgoing messages, since the path is unknown to intermediate nodes.

Advantages:

- The entire path is specified in outgoing messages and so it is not possible to form routing loops, i.e. where two nodes consider each other to be the next hop in order to reach a sink Z.

Disadvantages:

- Since the entire path is provided in outgoing messages, each message will be bigger, requiring more energy to forward it through the network.
- Topology changes might take time to filter through the network, which can result in messages being sent with invalid paths. Such messages may be lost since they rely on nodes or links that are no longer usable.

Requirements:

- Since the entire path must be known, it is necessary to use a searching method in which the entire path can be determined. Such methods include global knowledge and enumeration, which are discussed in sections 3.1.3.3 and 3.1.3.4 respectively.

3.1.4.2 All Full Path

Using all full path as an option for the table entries module causes each node on a path to store the entire (source, sink) path. This option usually occurs as a consequence of using global knowledge as the method of searching, as discussed in Section 3.1.3.3.

Advantages:

- There is no need to include the full (source, sink) path as part of outgoing messages, since intermediate nodes know how to forward a message so that it reaches the sink.
- Messages can be re-routed if the topology changes, since each intermediate node knows how to route a message. There is no reliance on possibly outdated paths supplied by a distant source node.

Disadvantages:

- Routing loops may form, causing rapid energy consumption and expiration of nodes.

Requirements:

- Since the entire path must be known by each intermediate node, it is necessary to use a searching method in which the entire path can be determined. Such methods include global knowledge and enumeration, which are discussed in sections 3.1.3.3 and 3.1.3.4 respectively.

3.1.4.3 Next Hop

When next hop is used for routing table entries, each node along a path only knows the next node that must be used in order to reach a sink.

Advantages:

- Outgoing messages need not include the full path, thus reducing their size and the energy required to transmit them.

- Since each node knows the next hop for a destination, there is no reliance on data that is held in the message itself, which may be out of date and based on a previous topology configuration.

Disadvantages:

- Routing loops are liable to form in which node A forwards messages to node B to reach a sink Z and node B forwards messages to node A to reach sink Z. Such routing loops cause nodes to expire quickly as messages become trapped in the network.

3.1.5 Summary

A summary of the modules that are part of the discovery task, including the options for each module, is presented in Figure 3.3.

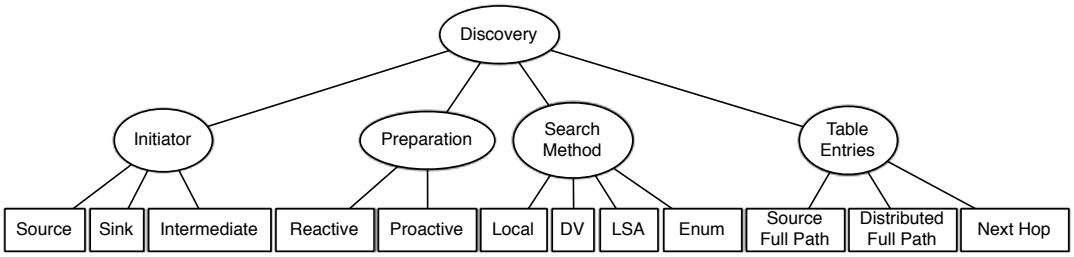


Figure 3.1: Routing protocol modules involved with the discovery task

3.2 Costing

Costing is the second task carried out by a routing protocol. During this task, (source, sink) paths are analysed based on a set of criteria defined in the routing protocol. Costing may be further broken down into two modules. *Link/Node cost* defines the set of characteristics of nodes or links on which cost is based and *path cost* defines how the individual node or link costs are aggregated in order to establish a path's cost. Options for the two modules are summarised in the following sections.

3.2.1 Link/Node Cost

Each link or node may have a cost associated with it. There are many ways of measuring the cost for a node or link. The most common approaches are outlined here and include *unit*, *geographic distance*, *TX/RX costs*, *energy reserves* and *quality of service*.

3.2.1.1 Unit

A *unit* link/node cost considers the cost of each node or link to be the same, i.e. 1. Such a measurement is almost exclusively used in *minimum hop routing* (discussed in Section 4.2) where the selected path is the one in which the fewest hops must be taken to reach the destination.

Advantages:

- Since the cost is not based on any feature of the node or link, cost assignment is trivial.

3.2.1.2 Geographic Distance

The cost of using a particular node may be based on how close that node is to the destination, i.e. a neighbouring node that is geographically closer to the destination than another neighbouring node will have a lower cost. Typically, geographic positions would be discovered by GPS. However, this can be expensive both in terms of infrastructure costs and energy and requires direct line-of-sight to the sky [99]. Other options have become available such as the Cricket [93] location system, which uses the delay between the arrival of simultaneously sent RF and ultrasound communications to calculate the approximate distance to some transmitter. Another option is the ad hoc location system (AHLoS) [99] developed by Savvides, which uses triangulation techniques to locate a node relative to several other nodes. Unlike GPS, neither of these techniques gives an absolute location, but rather a location relative to the transmitter.

Advantages:

- Nodes can use their position as a globally unique identifier; subject to the resolution

of the positioning system.

Requirements:

- Each node must know its geographic distance from any sink, which may have to be calculated after the WSN is deployed.

3.2.1.3 TX/RX Costs

TX/RX costs are based on the amount of energy required for a message of fixed size to be exchanged between two nodes. The energy requirements may include the cost of transmission (TX) and/or the cost of receiving (RX). For example, if a node A expends 20 mJ to transmit a 25 byte message to B and B expends 30 mJ to receive a 25 byte message then the cost of the link (A, B) may be 2 mJ/byte.

Advantages:

- The link cost is truly representative of the cost of routing a message, unlike other options for link/node cost which assume they will reduce the routing cost by reducing some variable (e.g. geographic distance).

Disadvantages:

- The energy cost of a link can only be accurately known after the communication has taken place, since messages may have to be retransmitted. Therefore any value given for the energy cost prior to the event is only an estimate.

Requirements:

- In order to determine whether messages must be retransmitted, nodes must acknowledge each successful transmission.
- Acknowledging transmissions requires that a receiving node can transmit back to the transmitter. Thus, links between nodes must be bidirectional.

3.2.1.4 Energy Reserves

Another commonly used link/node cost is that of energy reserves or remaining energy at a node. The cost of using a node is inversely proportional to the amount of remaining energy on that node, i.e. a node with a small remaining energy has a high cost.

Advantages:

- Nodes with low remaining energy tend to be avoided for routing, and thus remain active for longer.
- In theory, all nodes should expire at approximately same time, reducing wastage.

Requirements:

- Nodes must know the energy levels of their neighbours, which may change rapidly.

3.2.1.5 Quality of Service

When considering Quality of Service (QoS), the cost of using a node is inversely proportional to its performance, based on some user-defined measurement. For example, if it is important to prevent data loss, a node's performance may be based on the proportion of messages successfully routed to the sink. If it is important to route messages quickly, the performance may be based on time taken to forward the message.

Advantages:

- The network is encouraged to use nodes that perform well.
- The aims of the network are directly met by the routing protocol.

Disadvantages:

- Additional overhead may be encountered in calculating QoS, either in terms of time or energy expenditure.

3.2.2 Path Costs

The last module of the costing task is that of path costs and represents the way in which individual costs for each link or node are combined to determine how desirable it is to use a particular path. Several options for path costs are covered in this section, including *shortest path*, *lexicographic ordering* and *min/max element*.

3.2.2.1 Shortest Path

A path's cost is calculated by adding the individual costs of each node that lies on that path. The shortest (or minimum) path is the path whose sum of node costs is the least.

Advantages:

- Calculating shortest path in a distributed/incremental manner is easy and efficient. Each node simply forwards the smallest cost path that it learns of, adding its own node or link cost.

Disadvantages:

- It has been suggested that shortest path routing [25][105][21][20] may cause a subset of common nodes to expire quickly. Many such comments are made in reference in minimum hop routing that combines shortest path routing with a unit link/node cost or minimum energy routing that combines shortest path routing with link/node cost of TX/RX.

- Shortest path routing does not directly discourage the use of highly weighted nodes. For example, consider the network shown in Figure 3.2. The shortest path from source A to sink Z is ACGZ even though C has a higher cost than other nodes in the network. If data is sent this way, energy expenditure at node C will increase, causing it to expire quickly and disconnecting source E.

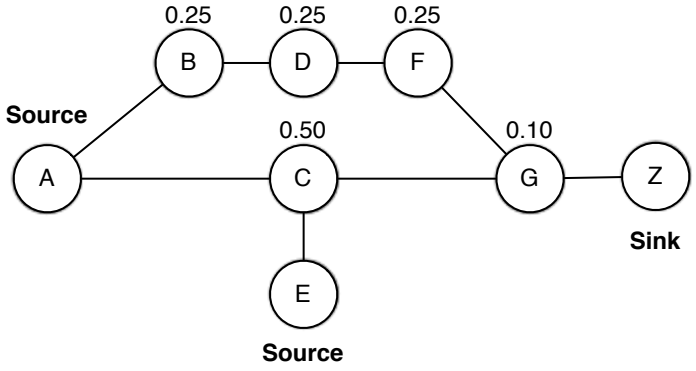


Figure 3.2: Shortest path may not prevent usage of high cost nodes

3.2.2.2 Lexicographic Ordering

Chang [20] has proposed the idea of lexicographic ordering. A path P is said to have a lower lexicographic order than a path Q if the highest cost of all the nodes in P is less than the highest cost of all nodes in Q. If the costs are the same, the second highest node cost from each path is compared, and so on. If all costs of a path have been compared, then the path with the fewest elements has the lowest lexicographic order. Lexicographic ordering closely relates to a dictionary sorting of the cost of nodes on each path, when node costs have been sorted in descending numerical order.

For example, consider node costs where A=1, B=2, C=3, ..., Z=26. ABBEY has a lower lexicographic order than OZ because the highest cost node in ABBEY (Y) is lower than the highest cost node in OZ (Z). As another example, MIME has a lower lexicographic order than MIMICK. The highest cost node in MIME and MIMICK is the same (M). The second highest cost node in MIME and MIMICK is also the same (M). However, the third highest cost node in MIME (I) is lower than the third highest cost node in MIMICK (K). Therefore, MIME has lower lexicographic order.

The lowest lexicographically ordered path can be found by using a modification of Dijkstra's algorithm where the shortest path is considered as the one with the lowest lexicographic order rather than the one with the smallest total node cost.

Advantages:

- The use of high cost nodes is more strongly discouraged than with shortest path

routing.

Disadvantages:

- It is only possible to say that a path A has a lower lexicographic order than a path B. There is no means to determine how much lower A is compared to B.
- The lowest lexicographically ordered path may still have a high total cost. For example, if path P has a single node of cost 0.5 and path Q has an infinite number of nodes of cost 0.4 then path Q has the lowest lexicographic order, even if the total cost is much higher than P.

3.2.2.3 Min/Max Element

Using min/max element, the cost of using a path is equal to the minimum (or conversely, the maximum) cost of any individual node or link that lies on that path.

Advantages:

- The use of particular paths can be encouraged or discouraged based on the features of the best or worst node on each path.

Disadvantages:

- Only the smallest (or largest) element in the path is considered. The cost or number of remaining elements is disregarded. Consequently the cheapest path may still use many nodes of high cost.

3.2.3 Summary

A summary of the modules that are part of the costing task and the options for each module are presented in Figure 3.3.

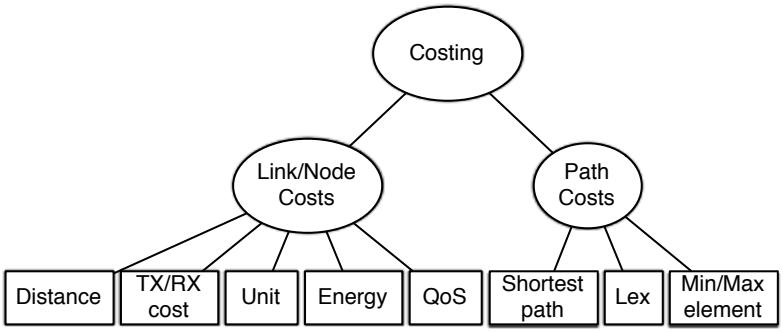


Figure 3.3: Routing protocol modules involved with the costing task

3.3 Selection

The selection task is the third and final task carried out by a routing protocol. Selection determines which path is used by the routing protocol. It may not always be the case that the cheapest path is used. The selection task is further broken down into two modules. *Topology* may impose additional restrictions on the way that data must be routed in the network from source to sink. *Multipath* is a technique in which several paths are discovered simultaneously for the purposes of distributing workload or ensuring data recovery by sending data along multiple paths to a sink. The options for these two modules are explored in the following sections.

3.3.1 Topology

Topology is the first of two modules that are part of the selection task in which a (source, sink) path is selected for routing. The topology module is used to create an additional level of abstraction over the network topology, thus causing data to be routed in a particular way from source to sink. For example, data could be routed such that it flows through a node that performs aggregation on incoming data in order to reduce the size of outgoing packets. Possible options for managing topology include:

- *hierarchy* in which nodes form clusters of nodes,
- *tree* in which each node has a single path to the sink, and,

- *role assignment* where node are dynamically assigned roles such as source, aggregator, forwarding node, etc.

These three options are discussed in the following sections.

3.3.1.1 Hierarchy

In hierarchical routing, nodes organise themselves into groups, clusters or zones. Members of a cluster may work together to distribute workload between them, or act as redundant nodes for each other. Typically, the nodes in each cluster elect a *clusterhead*, which acts as an aggregation source for the nodes of that cluster. Routing generally then occurs in one of two ways:

- All communication from a node travels through that node's cluster head after being aggregated with data from other members of that cluster. If multiple levels of a hierarchy exist, that cluster head may forward the data to its cluster head where the data may be further aggregated. The top level of the hierarchy is a sink node.
- Within each cluster are a number of *gateway* nodes which act as bridges between multiple clusters. Messages are aggregated by the cluster head and are forwarded through a sequence of clusters by using the gateway nodes until the message arrives at the sink.

Hierarchical routing facilitates data aggregation, potentially reducing the amount of data that must be transmitted to the sink. Since less communication takes place, less energy is expended and so nodes take longer to expire, although it has been suggested [65] that the possible savings from data aggregation are limited to about 9%. Clustering protocols require bidirectionality in order to form clusters and elect clusterheads. A node must be able to transmit to a clusterhead. The clusterhead can only be discovered if a node receives an announcement from that clusterhead. Thus, communication must initially be bidirectional. Once clusterheads have been elected and clusters have been formed, there is usually no need for communication to continue to be bidirectional until clusters or clusterheads change.

Advantages:

- Data aggregation can reduce the number and size of messages being transmitted.
- Not all data is routed through nodes nearest the sink.
- Nodes within a cluster may be able to lower their transmission power, as transmissions are likely to travel shorter distances.

Disadvantages:

- As shown in Section 2.5.2.1, it is debatable whether using long hops such as that from cluster heads to sinks is appropriate.

Requirements:

- If cluster heads are used, all cluster heads must be within radio range of the sink.
- Co-ordinating membership of a cluster may require bidirectionality to detect the cluster head and then send to it.

3.3.1.2 Tree

In tree-based routing, there exists a single path from any source to a sink node, which sits at the root. The tree is constructed so that the routing workload is distributed as evenly as possible. For example, sources that generate a lot of data may have (source, sink) paths that are as disjoint as possible, i.e. have few nodes in common with other sources in order to ensure that no subset of nodes is overused.

Advantages:

- Routing workload is distributed and balanced so that nodes do not expire quickly.
- Nodes with multiple children may aggregate data, reducing the size of messages being forwarded and thus saving energy.

Disadvantages:

- The loss of a single node may cause many upstream nodes to become disconnected.

- The construction of a balanced routing tree may require a lot of coordination between nodes and requires knowledge regarding the data generation rate of each source.

Requirements:

- Achieving a good balance of routing workload requires that the communication range of nodes is high. If numerous nodes generate large quantities of data, but are close together and have a small communication range, it may be impossible to balance their workload.

3.3.1.3 Role Assignment

Using role assignment as a means to enforce a topology, each node is given a specific task to aid in routing data from sources to sinks. For example, some nodes may act as aggregators to combine data from multiple sources. If an area is devoid of sources, and the nodes have suitable hardware then regular nodes may be promoted to sources in order to improve the amount of area being sensed. If too many sources lie in an area then some may be deactivated in order to reduce the quantity of superfluous data.

Advantages:

- Typically, each node performs less work:
  - Each node may only perform a subset of tasks, such as sensing or analysing, allowing nodes to spend more time in a low power state.
  - Complex role assignment may allow the network to operate for longer by placing nodes with a particular role in an optimal location, e.g. by placing aggregation nodes equidistant from sensing nodes.

Requirements:

- If one node announces that it is a routing node, bidirectionality must be present for nodes to receive the announcement and route to the announcing node.
- Complex role assignments may require large amounts of specific topology data such as locations of nodes.

- Some role assignments may require special hardware. For example promoting regular nodes to sources due to a lack of sensing resolution requires that every node is equipped with sensors.

3.3.2 Multipath Routing

This final module covers multipath routing in which a node discovers multiple paths in a single discovery process. Since each discovery process often involves a flood of messages through the network, which causes nodes to quickly expire, it has been suggested [35] that multipath routing can reduce energy expenditure by finding several paths simultaneously. Thus, when one path expires, another path is already known without the need for another flood.

However, it has been argued [36] that multipath routing does not significantly reduce the number of discovery processes that must be undertaken, since the loss of some nodes may invalidate all discovered paths. Furthermore, since additional paths are being discovered, more data must be transmitted during the discovery process, leading to an increase in energy expenditure.

In conclusion, it remains unclear whether multipath routing has any benefit. It is presented in this modularised view to represent those routing protocols that use it.

Having discovered multiple paths, a routing protocol may act in one of two ways. With *Weighted Cost*, each message is sent along a different path with cheaper paths being used most frequently. With *Backup Path*, a node maintains multiple paths in order to send data from sources to sinks with greater reliability.

3.3.2.1 Weighted Cost

When multipath routing is used with weighted costs, data is more likely to be routed along paths with lower cost. Paths may be selected in turn with the cheaper paths receiving more turns, or paths may be selected randomly with the cheaper paths being picked with higher probability.

Advantages:

- Workload from routing can be spread out, preventing a subset of nodes being overused and expiring quickly.

Disadvantages:

- Using multiple paths may involve the use of paths whose quality is poor. Thus, the performance of the network may suffer.

Requirements:

- Routing workload is spread out the most when the paths are as node-disjoint as possible. Two paths are *node-disjoint* if they have no nodes in common.

3.3.2.2 Backup Paths

When backup paths are used in multipath routing, a primary path and a number of backup paths are stored. Data is mostly routed along the primary path and the backup path is used when the primary path fails.

Another use of the backup path is to send data along it as well as the primary path in order to increase the probability that data is received at a sink node.

Advantages:

- Data of value can be sent twice, increasing the probability with which it is received.

Requirements:

- Backup paths are the most reliable when they have as few nodes in common with the primary path, i.e. they are as node-disjoint as possible. If they have nodes in common, the loss of a single node may invalidate the primary and backup paths.
- If data is sent twice, the application must be designed to handle the possibility that duplicate data may be received by the sinks.

3.3.3 Summary

A summary of the modules that are part of the selection task and the options for each module are presented in Figure 3.3.

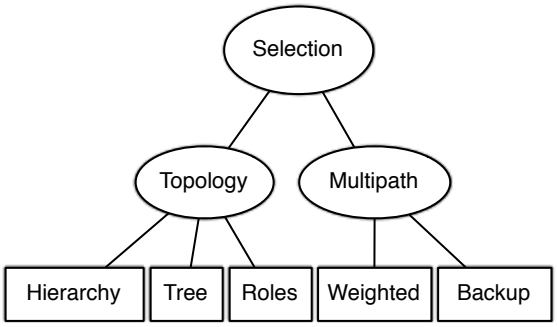


Figure 3.4: Routing protocol modules involved with the selection task

3.4 Summary

A routing protocol is an algorithm that gathers necessary topology information in order to select a (source, sink) path through which future data can be sent to sink nodes. This chapter has shown how a routing protocol's behaviour may be broken down into three tasks, discovery, costing and selection. Those three tasks may be further expressed as a series of different modules, each with different advantages, disadvantages and requirements. Each module has been explained in detail.

This modularised view of routing protocols will be used in Chapter 4 to compare different WSN based routing protocols. The behaviour of routing protocols in the literature will be explained using the modularised view in order to determine which (if any) are suitable for solving the problem of routing in order to maximise source diversity for long periods of time.

Chapter 4

Routing Protocols

This chapter reviews related work in WSN routing protocols. The modularised view of routing protocols discussed in Chapter 3 is used as a basis for comparison and discussion in this chapter.

4.1 Classifying Routing Protocols

A survey on WSN routing protocols has been presented by Royer [98] who classifies routing protocols as being one that either forms and maintains paths:

- via source nodes when needed (source initiated, reactive), or
- continually, regardless of whether they are needed (proactive).

However, as shown in Chapter 3, these categories only cover a small subset of the possible characteristics that may be exhibited by a routing protocol. Furthermore, some source initiated routing protocols such as DSDV [88], which is discussed in Section 4.2, are proactive, making their categorisation unclear.

Another survey by Akkaya [1] groups routing protocols into the following categories:

- *Data-centric*, which refers to sink initiated routing protocols whose applications use a publish/subscribe mechanism.

- *Hierarchical*, which forms clusters of nodes as discussed in Section 3.3.1.1.
- *Location based*, where data is forwarded at each hop to the neighbour that is geographically closest to a destination.
- *Quality of service*, where routes are formed based on network considerations such as latency, as discussed in Section 3.2.1.5.
- *Network flow*, in which the route taken by each message is pre-planned so as to maximise the amount of total data transferred, and,
- *Data aggregation*, in which nodes nearer the sink combine data from upstream nodes.

As shown in Chapter 3, these categories are neither all encompassing nor mutually exclusive. For example, the routing protocol MEHR [3], which is discussed in Section 4.3 is hierarchical. However, messages are forwarded to nodes that are geographically closer to the sink, thus also making it location based. Many of the proposed categories also include application details, such as the fact that the data-centric category relies on a publish/subscribe mechanism. The modularised view of routing protocols allows a precise description of routing protocols that might fit into multiple categories in a scheme such as Akkaya's.

Different routing protocols from the literature are now examined, with reference to the modularised view of routing protocols presented in Chapter 3. These protocols are grouped into the following categories, derived from the literature on WSNs:

- *Minimum hop routing*, in which data is sent through as few nodes as possible between the source and sink.
- *Hierarchical routing*, where nodes are grouped into *clusters* of nodes, each managed by a *clusterhead* that aggregates data and sends it to a sink.
- *Geographical routing*, where each node forwards data to whichever neighbour is geographically nearest to a sink.
- *Minimum energy routing*, in which data is sent along a path to a sink such that the total energy consumed by the entire network is minimised.
- *Load balancing routing*, which includes *energy aware routing*, *load balancing routing trees* and *congestion adaptive routing* and where routing workload is distributed in order to extend the time until the first node expires.

- *Flow control*, in which the path used by each message is selected such that the total data transfer is maximised.
- *Multipath routing*, where a node may search for and use multiple (source, sink) paths, either to reduce the number of times the discovery task must take place or to improve the probability that important data is received by the sink.

Where a routing protocol may belong to multiple categories, it has been included in the one that is more heavily emphasised by its authors.

Publish/subscribe based routing protocols have been examined but are not discussed in detail in this chapter. These routing protocols rely on specific application behaviour in order to function and so are not suitable for general WSN applications. Two common types of routing protocol fall into this category:

- data-centric routing protocols which have already been discussed, and
- *source initiated querying* routing protocols in which data is advertised by sources prior to the discovery task taking place in order to reduce the number of messages that must be exchanged to form a (source, sink) path.

Razzaque and Dobson have examined the use of cross-layer protocol stacks in wireless communication systems [95] in which non-adjacent layers of the network stack can exchange information. For example, the application layer could use information from the physical layer such as the received signal strength to adapt its behaviour, e.g. sending smaller packets more frequently if the signal strength is low. By allowing each layer to adapt to information provided by other layers, which would not normally be present, end-to-end communication can be optimised. Such a technique may be especially useful in a WSN that may be expected to operate efficiently and autonomously. However, since their contribution is not a routing protocol per-se, it is not considered further in this thesis.

4.2 Minimum Hop Routing

Minimum hop routing selects a path based on the minimum number of hops necessary to reach some destination (usually a sink node). In the modularised view of routing protocols,

minimum hop routing is represented by a unit cost for node/link costs and shortest path for path cost. Thus, the cheapest path is the one with the fewest hops to the sink. The typical module options for minimum hop routing are represented in Figure 4.1.

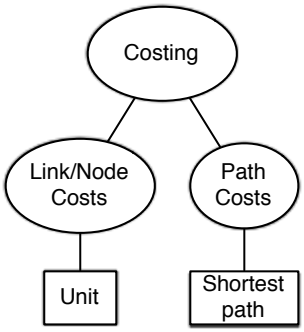


Figure 4.1: Typical module options for minimum hop routing protocols

Destination-Sequenced Distance-Vector Routing Protocol (DSDV)

DSDV [88] is perhaps the simplest minimum hop routing protocol. Distance-vector routing is efficient in that it only requires each node to know the cost and next node to use in order to reach a particular destination. However, if nodes or edges are lost and the knowledge of these losses are not correctly communicated through the network, then a routing loop can form in which two nodes try to route messages through each other in order to reach a sink, which can lead to messages being forever forwarded between two nodes, causing those nodes to quickly expire.

DSDV solves the problem of routing loops by using a network-wide sequence number for each sink during the discovery task. Each time a path to a sink node fails, the sequence number for that destination is incremented and a new discovery task for that sink is begun. Nodes will not route via any other node that uses a path with an old sequence number. When forwarding data, a node will always route data along the path of highest sequence number. Given a choice of paths with equal sequence numbers, the shortest one is used. DSDV forms paths proactively and attempts to keep paths updated, even if they are not being used.

In DSDV, each node discovers paths by periodically exchanging its routing table with its neighbours as well as whenever the node detects a change in topology. Routing table entries

include, for each sink:

- the number of hops to the sink, and,
- the sequence number associated with the sink, as discussed above.

Ad hoc On-Demand Distance Vector Routing Protocol (AODV)

AODV [89], modifies DSDV by forming paths reactively rather than proactively, as defined and discussed in Section 3.1.2. AODV discovers paths by a two-phase discovery task:

- The route request phase, in which RREQ messages are flooded from the source throughout the network in response to demand for a path to a sink.
- The route reply phase, in which a sink or a node with a path to a sink responds to an RREQ with an RREP message, which is sent along the shortest path taken by the RREQ.

Each RREQ message includes the number of hops through which the message has travelled, the source of the request, a broadcast ID, which is incremented every time the source makes a request, and a destination sequence number which specifies the required freshness of any path to the sink. When an intermediate node receives an RREQ, the node examines the broadcast ID and source node in the message, which collectively and uniquely identify each request. If the node has already seen this request, no further action is taken. The node then notes the neighbour from which it received the message, the source from where the message originated and the number of hops through which the message travelled. These details are used to establish a backpath from a sink to the source when a path is found. If the node has a path to a sink, the node examines the sequence number of its path to the sink and compares it to the destination sequence number in the RREQ in order to determine whether the node's path is of sufficient freshness. If the sequence number of the path held at the node is lower than the sequence number of the incoming RREQ then the node's path is too old and the RREQ must be retransmitted. Otherwise, the node may reply with an RREP using the path stored at the node.

In the route reply phase, a backpath is formed by sending an RREP back towards the source using the pointers that were set up at each node when they received RREQs. An RREP

message includes the source node, the destination sequence number and the hop count to a sink from the node that generates the RREP. An RREP is forwarded through the network using the backpath that was established during the flooding of RREQ messages. At each node, the hop count of the RREP is incremented, and the node establishes a *forward path* to a sink using the node that it received the RREP from and the hop count in the RREP. An intermediate node may receive multiple RREPs for a particular request from a single source. Latter RREPs will only be forwarded if they have higher destination sequence numbers than the first or if they have the same destination sequence number and a lower hop count. Thus, RREPs with the most up-to-date or efficient routing information are always forwarded.

An inherent requirement of AODV and DSDV is bidirectionality between pairs of nodes, i.e. if a node A can receive from a node B then node B can receive from a node A. Bidirectionality is required in order to reinforce the source to sink path by working from the sink to the source when the RREP message is sent during the second part of the path discovery phase. Section 2.5.2 suggests that this requirement may be unrealistic due to unreliable radio communication in WSNs.

Dynamic Source Routing (DSR)

DSR [54] also forms paths reactively. However, it does not require bidirectionality between pairs of nodes. In DSR, RREQs and RREPs contain the sequence of nodes through which the RREQ has travelled rather than just the number of nodes. The RREP is then returned to the source as part of a three-phase discovery task:

- The route request phase, in which RREQ messages are flooded from the source throughout the network in response to demand for a path.
- The route reply phase, in which a combined RREP+RREQ message is generated by either a sink or an intermediate node with a path to a sink and flooded through the network back to the source.
- A second route reply phase, in which an RREP is generated by either the source or an intermediate node and sent directly to the node that generated the RREP+RREQ message.

DSR does not rely upon backpaths in order to reinforce the (source, sink) path and so bidirectionality is not required, provided that there is some path from the intermediate node to the source. A disadvantage of DSR is its lack of scalability. Each RREQ message contains the sequence of nodes through which the message has travelled. Consequently, as the network size grows, the size and number of RREQ messages increase. Thus, more energy is expended by each node, reducing the lifetime of the network. DSR allows a node to store multiple paths to a sink. Thus, if one path fails, another may be used without the need to carry out the discovery task again.

In order for DSR or AODV to return the path of fewest hops, it is necessary to assume that nodes always take the same length of time to forward a message. However, this assumption may lead to a long path being selected when a shorter one may exist. For example, consider the network shown in Figure 4.2 in which source A sends an RREQ to find a path to sink Z. If node C is delayed in responding to the RREQ and node E receives the RREQ via ABDE first then E will discard the RREQ sent along ACE, even though that path has fewer hops. Path ACE will therefore remain unused.

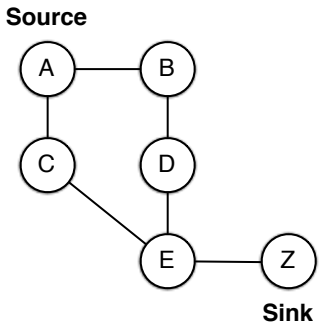


Figure 4.2: If a node is delayed in responding, the shortest path may not be returned

Temporally Ordered Routing Algorithm (TORA)

TORA [87] is a minimum hop routing protocol that aims to reduce the control messages that must be exchanged when a topology change occurs. It is assumed that links between nodes are bidirectional. However, it is also assumed that data flows in only one direction across each link. The protocol forms a minimum hop routing tree rooted on the sink node. The author does not address the possibility of multiple sinks. However, such an architecture could be represented by treating each sink node as the child of a single virtual *supersink*.

When a node failure occurs, the neighbours whose edges flowed towards the failed node systematically reverse the flow of their edges to recreate the tree. It should be noted that the process of routing tree repair does not consider network heuristics such as hop counts. Consequently a sequence of node failures may lead to a sub-optimal routing tree being used.

Energy Aware Routing Protocol (EARP)

When nodes expire, any paths making use of those nodes become invalid and a new path must be calculated, requiring the path discovery task to begin again. This can be very draining in a network, since path discovery often involves a flooding process in which every node must participate. EARP [79] is a modification of AODV, which calculates estimated path expiration times. The expiration time of a path is based on the elapsed time since the path was discovered, the number of hops involved and node mobility (if any). In AODV, a path will expire after being unused for some period of time. Rather than automatically invalidating a path after a fixed period of time, EARP estimates the duration for which a path will remain active. This period can also be extended by confirming whether a path is still functional. Each of these techniques allows the number of times the discovery task occurs to be reduced, thus lowering energy expenditure in the network.

As discussed in Section 2.5.2.1, it is hard to determine whether minimum hop routing creates paths of high or low reliability. Some routing protocols therefore aim to form minimum hop paths that are also stable.

Link Quality Estimation Routing (LQER)

In LQER [74], minimum hop paths are used to reach a sink. However, where a node has several equally distant neighbours, it uses the one whose link stability is the highest. LQER determines link stability based on a sliding window of k messages to determine which neighbour has the highest successful receive rate. Since LQER relies on receive rates, some mechanism must be in place to determine whether a transmission was successful. Consequently, links must be bidirectional. Another disadvantage of LQER is that nodes, which become temporarily unstable may stop being used long after their stability has improved. Determining that an unstable node's stability has improved would require

messages to occasionally be sent to the unstable nodes, risking message loss.

Efficient and Reliable Routing Protocol (EAR)

The efficient and reliable (EAR) routing protocol [61] considers a combination of three options for node/link costs, including:

- number of hops,
- QoS (packet loss), and,
- remaining energy.

A node with high packet loss may be used if it has a large amount of remaining energy. Thus, over time, as energy of reliable nodes drops, nodes with low reliability will be periodically attempted. Nodes with a sufficiently high packet loss may also be temporarily blacklisted and removed from the routing table. The removal is temporary in case the stability loss is also temporary and may be used in the future. As with LQER, determining packet losses requires the existence of bidirectional links, which may not be present.

Vidhyapriya's Routing Protocol

Vidhyapriya's routing protocol [117] is a modified form of AODV, which considers link stability as the major factor in selecting paths. It consists of two phases. In the first phase, a *neighbour discovery packet* is flooded through the network from sink to source, i.e. the protocol is sink initiated. The discovery packets dictate the minimum required energy and minimum received signal strength (RSSI) required to forward the packet further in the network. When a discovery packet is received, a node begins a *back off timer*. The value of the timer is inversely proportional to the received signal strength of the incoming packet. It is assumed that if a discovery packet is received with a small signal then the node is further away. When the timer fires, the node rebroadcasts the packet. Any node receiving a discovery packet with an unfired timer immediately cancels its timer. Thus, flooding is limited and the path with the smallest number of hops is encouraged. As with the AODV protocol, the path is reinforced by sending a *route reply* message back along the path, in

this case from source to sink. The minimum received signal strength field allows extremely poor links to be discarded. However, the routing protocol actively encourages the use of links that are poor quality (provided that they are above some threshold). Therefore, it is highly likely that using paths discovered in this manner would require messages to be retransmitted causing nodes to expend more energy to forward messages. The literature is also divided as to whether RSSI is a suitable measure of link stability, with different authors being both for [109] and against [120] its use.

In summary, minimum hop routing suffers from a number of problems. Firstly, it remains debatable whether such paths are stable. Some attempts have been made to form paths of minimum hops, which explicitly contain stable links. However, these routing protocols either require bidirectional edges in order to determine packet drop rates or they rely on RSSI, which may be unreliable. Secondly, as already discussed in Section 3.2.2.1, it has been well documented that shortest path routing (of which minimum hop routing is one example) can cause a small subset of nodes to be overused, leading to their expiration. The overuse of nodes typically occurs because shortest path routing protocols fail to consider the accumulated traffic from multiple sources. A node close to the sink is more likely to lie on numerous shortest paths compared to a node far from the sink that is not near any sources.

4.3 Hierarchical Routing

Estrin's Routing Protocol

Estrin [33] proposes a system of hierarchical routing in which nodes form clusters managed by cluster heads. Each clusterhead can be used to aggregate data created within its cluster to reduce the quantity of data being transmitted. Estrin's proposed clusterhead promotion system allows any node to be promoted to a clusterhead if a period of time passes during which no neighbour is promoted. The length of time is dependent on the number of local nodes and the remaining energy at a node. Thus, nodes with few neighbours and a high remaining energy are the most likely to be promoted. Estrin notes that only nodes with bidirectional links can be used in order to avoid inconsistencies in the network topology. Figure 4.3 shows the modules that make up all hierarchical based routing protocols.

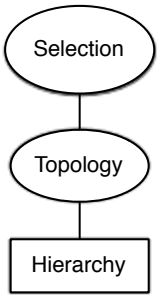


Figure 4.3: Typical module options for hierarchical routing protocols

Low Energy Adaptive Clustering Hierarchy Routing Protocol (LEACH)

LEACH [46] is an early clustering protocol in which the role of cluster heads is periodically and randomly rotated. The probability of a node being promoted to a clusterhead is affected by the desired number of cluster heads and the number of times that node has been elected as a clusterhead in the past. Under Matlab, using randomly generated networks with 5% of nodes as clusterheads LEACH was shown [46] to offer up to a factor of 8 reduction in energy compared to *minimum energy routing* which is discussed in Section 4.6. LEACH-C [45] is a derivative of LEACH in which clusterhead formation is controlled by the sink node. Cluster heads are selected based on their remaining energy and on the energy required by ordinary nodes to communicate with the cluster heads. Results demonstrate that LEACH-C is considerably more efficient than LEACH.

Kumar's Routing Protocol

Kumar [66] has proposed a system in which clusters are formed based on the remaining energy of nodes. Clusterheads are elected based on a combination of their remaining energy and the sum of the minimum power required by nearby nodes to communicate with them. A node's probability of becoming a clusterhead is improved by it being inexpensive to communicate with or having high energy reserves. This encourages the formation of clusters that will not expire quickly and consume small amounts of energy to operate. In contrast to most hierarchical routing protocols that proactively form clusters, cluster formation is reactively triggered by the reception of some data that is deemed interesting.

A disadvantage of these routing protocols is that clusterheads are expected to be able to

directly communicate with a sink, i.e. multihop communication does not take place [80]. Since any node may be elected as a clusterhead, it is necessary for every node to be in communication range of a sink. Thus, the communication range of nodes limits the deployment area. Furthermore, since clusterheads communicate directly to the sink and may be far away, transmissions may be unreliable or may require large energy expenditure.

Multi-hop Energy-aware Hierarchical Routing (MEHR)

MEHR [3] is a hierarchical routing protocol in which multihop routing is encouraged. Nodes are required to be aware of their location and capable of dynamically adjusting their transmission power to reach any node in the network. The clusterheads (which are randomly promoted) form a backbone through the network. Clusterheads use this backbone in order to reduce the number of long-range transmissions they must make, thus reducing energy expenditure. A clusterhead may also use a node within its cluster to further reduce the distance of any single transmission.

Hybrid Energy-Efficient Distributed Clustering Routing Protocol (HEED)

Changing clusterheads requires numerous exchanges of messages, which further reduce the energy of nodes in the network. In HEED [125], nodes with the most remaining energy are selected as clusterheads so that long periods of time will elapse before a new clusterhead must be chosen, thus reducing energy wastage. Having selected clusterheads, nodes join a cluster such that the transmission energy required to transmit to the clusterhead is minimised. As with MEHR, communication costs can be further reduced by the use of inter-cluster communication rather than requiring each clusterhead to directly transmit to a sink node.

Amis' Load Balancing Clusterhead Selection

Amis [2] has suggested a load balancing approach to clusterhead selection such that each node has the opportunity to become a clusterhead before any nodes expire. The advantage of such an approach is that the time for which all nodes are alive should be maximised. However, by increasing the number of clusterhead rotations, additional control messages

must be exchanged which further increases energy expenditure of the network and thus decreases its total capacity.

Hierarchical routing protocols offer the advantage of scalability in large networks containing many nodes. Since each cluster operates independently, there is no need for any regular node to be aware of the network outside its cluster and so nodes are only required to find a route to their local clusterhead in a smaller collection of nodes. Many routing protocols can achieve the scalability benefits of hierarchical routing by making use of the hierarchy module discussed in Section 3.3.1.1. For example, as will be discussed in Section 4.7, the MLDA [57] routing protocol's poor scalability can be improved by adding a hierarchical topology, and performing the MLDA algorithm on a small set of clusters rather than a large set of nodes. This new routing protocol is known as CMLDA [26].

The disadvantage to hierarchical routing protocols, and similarly the hierarchy module, is that links must be bidirectional so that a node may receive an advertisement from a clusterhead and consequently transmit to that clusterhead. As discussed in Section 2.5.2.2, links between nodes may be unidirectional and so this assumption may be unreasonable.

4.4 Geographical Routing

At each hop in geographical routing, messages are sent to nodes that are geographically closest to the destination. It is theorised that by routing in this manner, messages will be sent along the most direct path from a source to the sink. Since the energy expended in transmitting is a function of the transmission distance, the energy consumed by nodes in the network should therefore be minimised. It is common for no path discovery to occur in a geographical routing protocol, since nodes need only be aware of the location of their neighbours and the location of the end destination. Typical module selections for geographical routing protocols are shown in Figure 4.4.

Location Aided Routing (LAR)

LAR-1 and LAR-2 [63] are two protocols for routing to a mobile node using location information. Each protocol operates by estimating the location of a destination node and

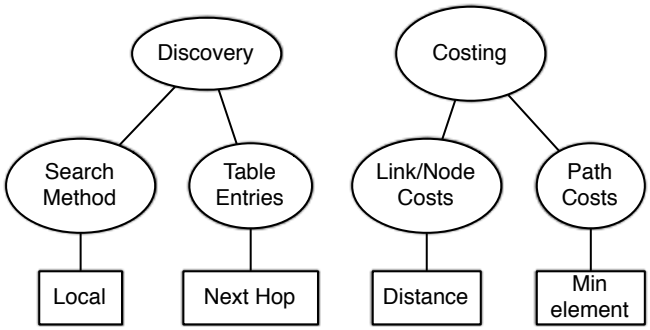


Figure 4.4: Typical module options for geographical routing protocols

routing data towards that location. In LAR-1, messages are forwarded towards a node's last known location and then flooded throughout the area where the node might be, based on its speed and the time since the node's position was last known. If the node's last position is unknown, the message is simply flooded throughout the network. In LAR-2, each node forwards data towards a node's last known position if they are closer to the destination or if they are not much further from the destination than their predecessor. Thus, the message is not routed along the most direct route but is slightly spread out. In both of these protocols, nodes must know their precise location. The authors suggest the use of GPS. However, this may be too imprecise and too energy intensive to be practical. A further problem with LAR is that there is no solution to the local minima problem [60] in which a message is routed to the edge of a *void* containing no nodes. The node at which the message arrives may have no neighbours that are closer to the destination. If a local minima is present, messages may be lost in the network as they cannot be forwarded further.

Greedy Perimeter Stateless Routing (GPSR)

GPSR [60] also uses geographic location information as a basis for routing. Unlike LAR, GPSR makes the assumption that nodes are stationary and only considers the use of *greedy* forwarding, i.e. routing a message to the neighbour that is geographically closest to the destination. A node periodically sends out a beacon message indicating its ID and position. Thus, all of the node's neighbours know where it is. GPSR is able to resolve the local minima problem. When a message is routed to a node on the edge of a *void* in which no neighbour is geographically closer to the destination node, GPSR uses a *right-hand rule* to route around the perimeter of the void. Once the void has been negotiated, geographical

routing can resume. GPSR requires that node connectivity is bidirectional, thus allowing a node to determine the geographic locations of its neighbours.

Power-Efficient Gathering in Sensor Information Systems (PEGASIS)

PEGASIS [78] uses the geographical positions of nodes to form a chain. Whenever a source receives any incoming data, it is aggregated with the source's own data and forwarded to the next source in the chain, i.e. whichever source is geographically closest. One source in the chain is designated as the *leader* and forwards any incoming data to a sink, which may be a long distance away. The leader is rotated in order to balance the energy expended from long distance communications. The chain is formed by linking nearby nodes together so that transmission powers may be dropped and energy expenditure lowered. However, since data may be forwarded through many sources before it arrives at a sink, the latency between data being generated and data arriving at the sink may be high.

Geographical and Energy Aware Routing (GEAR)

GEAR [126] is a geographical routing protocol that also considers the energy reserves at each neighbouring node. Nodes estimate their neighbours' energy reserves by tracking the data sent to each neighbour since that neighbour last announced its energy reserves. The suitability of a neighbour for the next hop for a message is based on a weighting of two factors:

- the distance of the neighbour to the destination, and
- the estimate of the remaining energy of that neighbour compared to other neighbouring nodes.

The weighting is such that when all nodes have an equal estimated remaining energy, the next hop is the node that is nearer to the destination. Conversely when all nodes are equidistant from the destination, the next hop is the node with the greatest estimated remaining energy. When the message reaches the target geographic region, the message is disseminated across all the nodes in that area. One difficulty with this protocol is in knowing or estimating the remaining energy on each neighbour node. Activities such as

processing or sensing may affect the energy reserves of a neighbour. Furthermore, as the number of sources is increased, estimates become harder to make. For example, if sources A and B each use a node C they are only aware of their own contributions to the loss of C's energy unless they are both involved in routing for each other. It is implied that bidirectionality is required for GEAR to work, as nodes must update their neighbours regarding how much energy they have remaining.

Weighted Energy Aware Routing Protocol (WEAR)

WEAR [102] is a proposed improvement of the GEAR protocol. In WEAR, each node is assigned a cost. The cost is weighted on four different factors:

- the proximity of the node to the sink, with closer nodes having a higher cost;
- the proximity of the node to a hole (void), i.e. a region with no nodes, with a node's cost being increased by being near a void;
- the node's energy reserves; and
- the distance of the node from the target, with nearer nodes having a lower cost.

These four factors are weighted and combined to provide a node cost. In considering all these factors, WEAR aims to improve two perceived flaws of GEAR. Firstly, the overuse of nodes nearer the sink and secondly the expansion of routing voids by continually taking the same path to avoid a void. Factors 1 and 4 appear to be contradictory, in that a node near the sink is both encouraged and discouraged. However, the former is used to discourage the unnecessary use of nodes near the sink in order to achieve load balancing while the latter is used to encourage direct paths between a source and sink. WEAR assumes the presence of bidirectional links.

Geographic routing causes messages to be routed along the most direct path to a destination. At each hop, the node nearest the destination is selected to be the next hop. However, this behaviour would appear to encourage the use of long hops. As shown in Section 2.5.2.1, there is conflicting evidence as to whether this is a sensible approach or not.

4.5 Load Balancing Routing

Load Balancing Routing aims to distribute the workload experienced from routing across as many nodes as possible. If workload is perfectly balanced, then all nodes in the network should die almost simultaneously. There are three common approaches to achieving load balancing in routing protocols:

- *energy aware routing* refers to the use of nodes with the most energy when forwarding data from sources to sinks,
- *balanced routing trees* involves the creation of load balanced trees such that the total data forwarded by any node is as even as possible,
- *congestion aware routing* allows nodes to send control messages to their upstream nodes in order to reduce the amount of data being sent to them.

These three options for load balancing are discussed in the following sections.

4.5.1 Energy Aware Routing

In energy aware routing, paths from sources to sinks are formed such that the nodes with the most remaining energy are used where possible. The selection of modules used in energy aware routing protocols is shown in Figure 4.5.

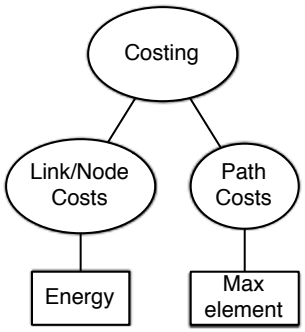


Figure 4.5: Typical module options for energy aware routing protocols

Singh's Routing Heuristics

Singh [105] presents several energy aware routing heuristics. These approaches are:

- *minimise energy expenditure*, which will be discussed in Section 4.6,
- *minimise max node cost*, which is also discussed as part of minimum energy routing in Section 4.6,
- *maximise time until network partition*, which aims to maximise the time until there is no path from a source to a sink; Singh indicates this problem is NP-complete and therefore impractical,
- *minimise variance in node power levels*, which Singh also indicates is NP-complete and therefore impractical,
- *minimise cost/packet*, where link/node cost is the inverse of a node's energy reserves and the path cost is shortest path.

This final remaining approach (minimise cost/packet) is used as the basis for most energy aware load balancing routing protocols. Chang and Tassiulas [20][21] have experimented with routing strategies to extend a WSNs lifetime, considered to be the time until the first node expires, and have studied this approach in more detail. In their experiments, the cost of a link was based on two factors: the remaining energy at the transmitting node and the energy required to transmit. By trying different weightings of these factors, Chang and Tassiulus determined that the dominant factor in extending lifetime is the remaining energy of a node. Consequently, paths should be selected to avoid the use of nodes with little remaining energy. One problem with the experiments of Chang and Tassiulas is their assumption that nodes only expend energy in transmitting. As discussed in Section 2.5.1, a node may expend more energy in receiving than transmitting and so the experiments of Chang and Tassiulas may not be entirely accurate.

There are also problems with the minimise cost/packet heuristic. Firstly, if cost is based on a node's energy reserves then a node's cost may be constantly changing. Determining the remaining energy at a node requires energy expenditure. It therefore becomes difficult or expensive to accurately assess which route is cheapest in the network. In order to provide an accurate approximation to the amount of energy stored in a node, Lin [76] proposes

representing the remaining energy of nodes using discrete *energy levels*. Using this technique, a four-level logarithmic scale is proposed with level 0 corresponding to full energy, level 1 between half and full energy, level 2 between quarter and half energy, and level 3 between one eighth and zero energy. Whilst this reduces the overhead associated with load balancing, routes are required to be recalculated when the energy level of any node changes. The second problem is that minimising cost/packet only involves the distribution of routing workload and does not consistently use paths of lower cost. Thus, the capacity of the network may be lessened by the use of sub-optimal paths even if more time elapses before the first node expires. The remainder of routing protocols in this section aim to use the minimise cost/packet heuristic while reducing the cost of each routed message.

max-min zP$_{min}$

In max-min zP$_{min}$ [72], the transmission power of nodes is reduced such that the network remains connected. By reducing transmission power, the energy consumption of the network is reduced. In order to carry out load balancing, the routing protocol then selects the (source, sink) path whose node of least energy is the highest, i.e. the maximum minimum remaining energy. Paths containing nodes of little energy are therefore avoided. However, in reducing the transmission power of nodes, the number of connections between nodes will drop. As the number of connections drops, the amount of load balancing that can take place will also drop. For example, if reducing the transmission power causes more bottleneck nodes to form then it may not be possible to carry out load balancing, since all data must travel through the bottlenecks.

Chang's Routing Heuristic

Chang [22] has proposed a new link/node cost measurement in which the cost of transmission becomes more important as the node's remaining energy drops. Therefore, at first, it is acceptable to use suboptimal paths to achieve load balancing. However, as the energy of the network declines, it becomes more important to route such that energy is preserved. There are two difficulties with Chang's approach. Firstly specific implementation details are absent, making it difficult to comment on the approach. Secondly, the solution involves the use of sub-optimal paths in order to carry out load balancing. The use of these paths

may reduce the capacity of the network. A better solution would be to carry out load balancing on the optimal paths, thus not wasting energy and carrying out load balancing simultaneously.

Capacity Maximisation Routing Protocol (CMAX)

CMAX [58] operates by firstly disregarding all links where the cost of transmission is higher than some fraction of the remaining energy of a node. Thus, links will not be used if they result in a node's expiration. Link costs are based on both the remaining energy of a transmitting node and the cost of transmitting from that node to the receiver such that a link's cost is lower if the node has more energy or if it the transmission cost is low. Path cost is calculated based on shortest path routing with the additional requirement that the path cost must be below some constant. The author of CMAX notes that some of the routing protocol's behaviour is unacceptable in a WSN. For example, if the use of a particular path is deemed too high, or if it involves the use of nodes that are close to expiring, the routing protocol will refuse to route the message. This behaviour will lead to a network where all nodes have a small amount of remaining energy and cannot engage in routing, which is of little practical use.

HEAP

HEAP [127] is a modification of AODV in which each message contains a transmission power p and remaining energy r. The protocol operates similarly to AODV except that on receiving a message, a node may only react if it has more than r remaining energy. Furthermore, any message transmitted by the node must be sent using a transmission power of p. The routing protocol carries out route requests using increasing values of p and decreasing values of r until a path is found. A disadvantage with this routing protocol is the extra energy expenditure that occurs in carrying out multiple route requests. Furthermore, more route requests must be carried out if there are only poor paths available, i.e. those in which r is at a minimum and p is maximum.

The majority of these routing protocols are problematic in that they involve additional energy expenditure in order to search for a low cost path that may be load balanced. There is no guarantee that any path will be found, thus causing additional energy expenditure for

no benefit. Another limitation of energy aware routing is that it extends the time until any node expires. However, the loss of some nodes may have no impact on the connectivity of the network. On the contrary, it may even be beneficial to overuse certain nodes to prevent certain critical nodes from expiring and partitioning the network.

4.5.2 Load Balanced Routing Trees

In some WSNs, all nodes act as sources and efficient routing is achieved by the construction of a *routing tree*. The idea of such a tree is that the total traffic handled by each source is as distributed as possible so that the majority of traffic does not flow through a single source. Such a situation is common when a small number of nodes surround the sink and all data must travel through one of those node to achieve (source, sink) routing. Routing protocols that form load balanced routing trees generally use the selection of modules shown in Figure 4.6.

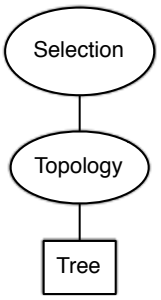


Figure 4.6: Typical module options for routing protocols that form load balanced trees

Dai's Routing Protocol

Dai [25] proposes the iterative construction of a balanced routing tree. The tree begins with the sink. At each iteration, the node that is not on the tree and has the heaviest load is added to the branch of the tree with the lowest load. This technique produces a routing tree that is roughly balanced according to that routing load that each node exerts on the network. However, Dai's approach is only able to balance the routing load to a certain extent and is limited by the physical connections that nodes can form. For example, if only two nodes are directly connected to the sink node then Dai's balanced routing tree could attempt to

balance the network such that 50% of the network traffic travels through each of those two nodes. However, a situation in which four nodes were directly connected to the sink node and where 25% of traffic travels through each of those four nodes would result in a more balanced tree.

Wu's Routing Protocol

Wu's approach [121] is to limit the number of bottlenecks in the network by adding and removing edges to the routing tree to spread the number of children at each node. Wu defines a bottleneck as any node whose ratio of energy reserves to node *degree* is above some constant, where the *degree* of a node refers to the number of upstream neighbours at a node (i.e. those nodes further from the sink). Thus, a node is considered a bottleneck if it no longer has the energy to support its children. Wu's approach solves this problem by removing children from a node so that it has less incoming data and assigning those children to another node.

Jung's Repair Mechanism

A problem with routing trees is that each node has only a single path to the sink, and consequently if a single link fails, one or more nodes will be disconnected from the tree. If the lost node is near to the sink then more nodes will be disconnected. Jung's repair mechanism [55] allows trees to be dynamically reformed. When a node expires, its children send out broadcasts on their radios to search for new parent nodes to connect them to the sink. However, in order for a node to request a new parent and receive the parent's reply, links must be bidirectional.

There are two further difficulties in forming routing trees. Firstly, it is necessary to know how much data a given source will generate in order to determine where in the network it should be (in order to balance routing). It may be impractical or even impossible in some applications to preemptively determine how much data a given source may generate, since it may be dependent on the environment being sensed. Secondly, there is a limit to how flexible the tree may be given that two nodes must be in communication range of each other in order to be a child/parent pair in the routing tree. For example, if the nodes that generate the most data are all clustered together and if the radio range is small then it may not be

possible to distribute their routing loads.

Energy-Aware Data-Centric Routing Algorithm (EAD)

EAD [12] solves both of these problems. Firstly, rather than relying on how much data a node will generate, the routing protocol uses a node's remaining energy as a metric to connect it to the tree. Secondly, routing load can be better distributed by assigning roles to nodes. Nodes may become either aggregators or leaf nodes. The former carry out routing and aggregation of data while the latter sense the environment. Since routing involves more energy expenditure than sensing, the roles last only a fixed period of time in order to allow rotation and balancing. The algorithm waves out from the sink. When a node announces that it is connected to the sink, its unconnected neighbours compete to determine which node will act as an aggregator for the others. Each node waits for a period of time inversely proportional to its remaining energy. The node that awakens first becomes a non-leaf (aggregator) node and other nodes become leaf nodes. The process then repeats, with the aggregator announcing that it is connected to the sink node. Since a node may announce its state as an aggregator and other nearby nodes will transmit to that node, bidirectional links are required.

None of these approaches seek to minimise the energy consumed in routing data from sources and sinks, and thus, the capacity of the network may be greatly reduced even if the time until first node expiration is extended.

4.5.3 Congestion Adaptive Routing

In congestion adaptive routing, the aim is to achieve load balancing by reacting to the congestion of particular nodes. Such a reaction may be to use a different path, change roles or to limit the data being sent along a particular path. However, these approaches assume that data is being continually routed from sources to sinks. If data generation is sporadic, e.g. it occurs in relation to some unpredictable event, then a node may never receive a large number of messages in a short space of time and so may never be considered to be congested. Consequently, a node may expire without ever having been considered to be congested. Figure 4.7 shows the modules that are typically present in congestion adaptive routing protocols.

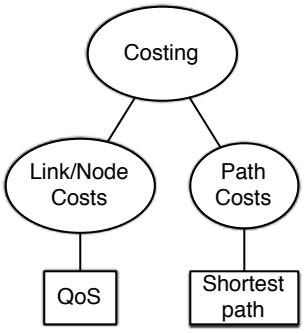


Figure 4.7: Typical module options for congestion adaptive routing protocols

Lee's Routing Protocol

Lee [67] presents a routing protocol in which a node's cost is proportional to the workload that it currently experiences from routing. When a source forms a path to a sink, it selects the shortest path, i.e. the sequence of nodes whose sum of workloads is the smallest. As data packets are routed from sources to sinks, an intermediate node may attach its current workload. When the packet arrives at a sink, the sink may determine that the workload of certain nodes is too high and notify the source that it should search for another path. In this manner, the network can react to nodes that become congested at a later time. One disadvantage to this approach is that it may not handle cases where congestion is short-term. For example, in an event-based system, a node may only become congested very rarely and for brief periods of time. This may cause source to select new paths, by which time the congestion may have ended, causing the source to select the same path or a path which is prone to the same periodic congestion.

Tran's Congestion Adaptive Routing Protocol

Another disadvantage to Lee's routing protocol is that is requires the discovery task to begin again whenever a node is considered congested. Since the discovery task involves the extra transfer of messages without transferring additional data from sources to sinks, Lee's protocol may result in large additional overhead. Tran's Congestion Adaptive Routing protocol [115] allows nodes to route around a heavily loaded downstream neighbour by forming a bypass around that node. Since a node only needs to route around its neighbour, the amount of information required about the topology is limited. Furthermore, if a node

can be routed around, it is not necessary for the discovery task to begin again, and thus, the overhead of the routing protocol may be reduced.

SPEED

SPEED [44] is a congestion aware routing protocol that is also geographically greedy. At each node, the next hop is calculated by selecting the lightest loaded node that is geographically closer to the sink. Nodes can apply a *backpressure* to indicate to their upstream nodes that they are congested and must be sent fewer messages in order to allow the congestion to clear. A problem with SPEED is that if all nodes along a path are congested, then a backpressure request may have to be forwarded through all those nodes in order to get a reaction. For example, if each node on the path ABCDE is congested, then a backpressure request must be sent through each node in order to get node A to route messages elsewhere. However, if each of those nodes is congested, it becomes difficult for a request to be received. Furthermore, if messages are routed from A to E then the backpressure request must be forwarded from E to A, thus requiring bidirectional links between nodes.

Unlike the node reliance routing protocols presented in this thesis, these congestion adaptive routing protocols are not cooperative. For example, if several sources collectively use a node X then X may become congested. If each routing protocol simultaneously switches to using a node Y then the situation is not corrected, since a different node becomes congested. Furthermore, encouraging the use of underused nodes may still be to the detriment of the network. For example, consider a node M, which acts as a bottleneck to source A. If only source A is routing through M at a particular time, then M may be uncongested. However, it would be unsuitable for other sources to route through M if they had other paths available since the loss of M would disconnect A from the network.

4.6 Minimum Energy Routing

Several routing protocols have been discussed that aim to reduce the distance that nodes transmit in an effort to reduce energy expenditure. However, distance is not the only factor that affects the RX or TX costs. If a link is particularly unreliable, each message may have to be sent several times in order to ensure that it is received, even if the nodes are

closely placed. More generally, routing protocols that send data along paths such that energy expenditure is at a minimum are known as *minimum energy routing* protocols. Such protocols use a link/node cost of *TX/RX cost* and path cost of *shortest path* in order to minimise expended energy. Typical modules used by minimum energy routing protocols are shown in Figure 4.8.

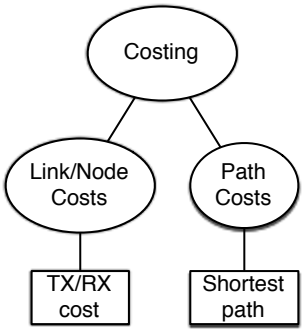


Figure 4.8: Typical module options for minimum energy routing protocols

Min-max Link Power Routing Protocol (MLRP)

MLRP [127] is a minimum energy routing protocol that behaves similarly to AODV. When route request (RREQ) messages are flooded through the network, they indicate a maximum transmission power for each node to use when forwarding the message. By starting with a low transmission power and repeating the route request phase with increasingly large transmission powers until a RREP is received, a shortest path using the least transmission power at each hop can be determined. However, this solution is not necessarily the path of minimum energy since increasing the transmission power slightly may result in a path with significantly fewer hops. Another disadvantage with MLRP is its reliance on bidirectional links between pairs of nodes due to the creation of a backpath. Since such links may not exist, the routing protocol may perform poorly.

As discussed in Section 3.2.2.1, shortest path routing may lead to the overuse of a subset of nodes, which may cause them to quickly expire. The majority of minimum energy routing protocols therefore select paths that distribute workload as well as reduce the total energy expenditure of the network. Raghunathan [94] suggests that routing protocols should avoid using the same path repeatedly and avoid using nodes with low energy reserves in order to reduce node expiration and make more nodes available in the future for critical work.

The preferred approach is to distribute workload as uniformly as possible over the network. However, it is observed that the optimal distribution of workload is only possible if future network activity is known. It may therefore be argued that without knowing data generation rates, optimal load balancing cannot generally be achieved. The remainder of routing protocols discussed in this section carry out minimum energy routing while distributing workload.

Shah's Routing Protocol

Shah [103] proposes calculating a *set* of energetically efficient paths between sources and sinks and randomly selecting a path to use. In Shah's proposal, an energetically efficient path is one whose nodes are:

- close to the sink,
- have high remaining energy, and,
- have low transmit and receive costs.

By additionally using multiple paths, Shah's routing protocol performs load balancing while simultaneously reducing energy expenditure in the network, thus extending the time until the first node expires.

Ye's Routing Protocol

Ye [124] introduces a minimum cost routing solution in which each node estimates the minimum energy required for it to route a message to the sink. Each data message includes two fields:

- the energy budget, which is equal to the amount of energy required to route a message from the source node to the sink, and,
- the expended energy, which is an updated field reflecting how much energy has been expended so far in routing the message to the sink.

The difference between the two values indicates how much energy is still to be spent in routing the message to the sink. When a node receives a message, it determines whether it is capable of routing to the sink within the remaining budget. If not, the message is ignored. If it can, it updates the expended energy field and rebroadcasts it. The advantage of this scheme is that it is not necessary to directly form any paths. The energy budget is equal to the minimum energy required to route a message to the sink and a node will only forward the message if it lies on that minimum energy path. A disadvantage of Ye's approach is that there may be several minimum energy cost paths from a node to a sink. In such a situation, every packet broadcast by the node will be routed multiple times, once along each path.

Minimum energy routing protocols make use of the node/link cost module by examining TX/RX costs as discussed in Section 3.2.1.3. For TX/RX costs to vary, either the transmission power or the number of transmission attempts must vary. However, in each case it is necessary to confirm that a message has been successfully received. Otherwise, the total cost of transmission and receipt is unknown. If the transmission power is not varied and all messages are assumed to be successfully received, the TX/RX costs will always be the same and may not represent the costs incurred by the node.

4.7 Flow Control

Network flow problems are a class of problem in graph theory that aim to maximise the flow of some *commodity* from source nodes where flow is generated to sink nodes where it is collected. The network is made up of nodes and directed, weighted edges that indicate the flow capacity across each edge. With the exception of the source and sink nodes, any incoming flow to a node must be equal to the outgoing flow from that node. The solution to a network flow problem reveals which paths to use in order to maximise the commodity received at the sinks.

The routing protocols in this section optimise the lifetime of nodes by treating the network as a network flow problem. Nodes in the graph represent nodes in the network and a directed edge from a node A to a node B in the graph indicates that node A can transmit to node B in the network. The capacity of an edge (A, B) represents the number of bytes that node A can transmit to node B before running out of energy and the commodity of the network is data. The solution to the corresponding network flow diagram indicates how

data should be routed from sources to sinks such that the number of bytes received by the sink is maximised while not causing any node to expire. Sending additional data through the network will cause the expiration of nodes.

Flow control routing protocols are not routing protocols in the conventional sense; paths are selected based on the solution to the network flow based problem rather than through the assignment of costs to individual links or nodes. Consequently, the modularised view of routing protocols described in Chapter 3 cannot be used to accurately represent the general behaviour of these protocols.

Maximum Lifetime Data Aggregation (MLDA), Maximum Lifetime Data Routing (MLDR) and Clustering-Based MLDA (CMLDA)

Kalpakis presents two routing heuristics, MLDA and MLDR [57]. MLDA is used in networks in which aggregation of data takes place and MLDR is used where it does not take place. The heuristics take the form of linear programming equations whose solution provides a set of routing trees that indicate how each individual future message should be routed in order to maximise the number of bytes received at the sink without any node expiring. A problem with Kalpakis' solution is the length of time required to solve the network flow problem for a given WSN. Dasgupta [26] notes that the growth of the number of variables in the linear programming problems is of the order O(n^3) where n represents the number of nodes in the network and that it takes 60 seconds to produce a solution in a network with 20 nodes and up to 5 hours to produce a solution in a network of 100 nodes. Dasgupta therefore proposes a modified routing protocol known as CMLDA in which the network is firstly separated into clusters and MLDA is carried out on the clusters themselves. Since there are fewer clusters than nodes, the linear programming problem contains fewer variables and is faster to solve.

Pham's Distributed Algorithm for Network Flow

Rather than solving the network flow problem on a single centralised machine, it may be more efficient to distribute the task throughout the network. Pham [91] has presented a distributed algorithm for solving the network flow problem. However the growth of the number of message exchanges is O(n^{2m}) where n is the number of nodes and m is the

number of edges in the network. Such a growth rate is impractical for a WSN in which energy must be expended in order to transmit or receive every message, since the energy expenditure of the network will grow rapidly with respect to the size of the network.

Li's Routing Heuristic

A major disadvantage with network flow routing protocols is the requirement that data generation rates of sources are known. If the data rates, and hence the output of each source, are not known, it is not possible to formulate a network flow problem since it is not known how much data each downstream node must forward. Li [75] attempts to resolve this requirement by the use of a *traffic matrix*, which represents the average total data transferred between each (source, sink) pair over a period of time. However, even this requirement is impractical if data is formed in response to random environmental factors.

Flow control routing protocols are not common in WSNs. They require knowledge of the data generation rate of sources in order to control where that data should be routed. The capacity of each node must also be known, which requires knowledge of both a node's remaining energy and the energy it must expend in transmitting and receiving. However, as noted in Section 4.6, determining how much energy is involved in transmitting or receiving requires bidirectionality between pairs of nodes, which may not exist.

4.8 Multipath Routing

As discussed in Section 3.3.2, multipath routing protocols discover several paths in each discovery task rather than finding a single path and using that path until it expires. The aim of multipath routing is to reduce the number of discovery tasks that must take place, which often involve flooding messages through the network and are therefore energetically expensive. Multipath routing also provides the advantage of having multiple paths available that may be used for load balancing or for ensuring important data arrives at a sink by sending it along multiple paths. The typical modules involved in multipath routing protocols are shown in Figure 4.9.

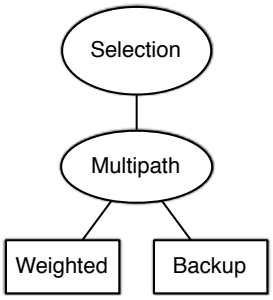


Figure 4.9: Typical module options for multipath routing protocols

Random Walk Routing

Random walk routing [113] is a simple multipath routing protocol. When a node receives a message, it selects a random neighbour and forwards the message to that node. A variant of the protocol prevents a node from forwarding messages back to the neighbour that sent them. An advantage of random walk routing is that it has minimal overhead. The discovery task of each node operates locally, i.e. it only needs to know the identity of its neighbours. However, random walk routing suffers from a number of disadvantages. Firstly, as it makes no effort to send a packet towards its destination, a message may be repeatedly passed between the same set of nodes causing unnecessary energy expenditure. Secondly, it is unclear how random walk routing scales with network size. As the number of nodes increases, the probability that the correct sequence of nodes will be randomly selected such that a packet arrives at the sink must rapidly decrease. Thus, random walk routing would not be expected to work well with large networks.

Split Multipath Routing (SMR)

SMR [68] is a reactive routing protocol that selects a primary path based on a node cost of QoS. Specifically, the first path to be discovered becomes the primary path. A secondary path is formed which is selected to be as disjoint as possible from the primary, i.e. as few nodes as possible appear on both the primary and secondary path. The routing protocol alternates between the primary and secondary path, thus achieving load balancing and reducing the frequency with which the discovery task must be carried out.

Ganesan's Routing Heuristic

A disadvantage of SMR is that it can only handle the failure of one node on each of the primary and backup paths. Thus, if one node on each path fails, routing must stop until the discovery task can be carried out again. Ganesan [35] offers a different approach in which a single (source, sink) path is braided, i.e. there exists a path around every single node on the primary path. Thus, many node failures must occur before routing becomes impossible.

Multipath On-demand Routing Protocol (MOR)

MOR [10] is proposed to make use of all shortest paths between a source and sink. It is a modification of AODV, which was discussed in Section 4.2. Ordinarily in AODV, a node will examine the sequence number in an RREP and disregard the packet if the sequence number is one that the node has already forwarded. In MOR, if the sequence number is the same then MOR will record which neighbour the RREP came from and how many hops were required. Thus, as a node receives multiple RREP packets with the same sequence number it will learn which neighbours have different paths to the sink. MOR is vulnerable to the same disadvantages of AODV. It relies on bidirectionality of links in order to form (source, sink) paths and it uses shortest path routing, which may cause a subset of nodes to be overused. Furthermore, as discussed in Section 4.2, it remains unclear whether minimum hop routing may lead to the use of unreliable links in the network.

Reliable Energy Aware Routing (REAR)

A potential disadvantage of multipath routing is the overuse of important nodes. Since each source forms multiple paths from itself to a sink, it is more likely that an important node, i.e. one on which sources rely upon to remain connected to the sink, will be overused. REAR [42] resolves this problem by only allowing nodes to appear on a certain number of paths. When a path is formed, each node on that path reserves an amount of its remaining energy for routing data. Only nodes with sufficient unreserved energy can take part in future routing requests. Consequently a single node cannot be over-used as it can only offer what it has available.

Although REAR prevents a path from being overused, the order in which paths are formed

is important. For example, if source A forms a path through X first, then X may refuse to appear on a path from source B. However, if X acts as a bottleneck to B then B is unable to participate in routing. The node reliance heuristics presented in Chapter 6 are superior in this respect, since node reliance allows bottlenecks to be used by those sources that must use them, while discouraging their use by sources that do not require them. Node overuse is not directly prevented. Instead, the overuse of nodes that are essential to routing is discouraged.

4.9 Comparison

The routing protocols that have been discussed in this chapter are compared in the following tables. The behaviour of each routing protocol has been described using the modularised view of routing protocols that was presented in Chapter 3. In some cases, the way in which a routing protocol works is unclear or is too complex to summarise. Where the author has neglected to include implementation details, a best guess has been made based on description provided by the author. Where it is impossible to guess what an author intended, the details have been omitted.

Protocol/Author	Discovery				Costing		Selection	
	Initiator	Prep	Search	Table	Link/Node Cost	Path Cost	Topology	Multipath
DSDV [88]	Source	Proactive	DV	Next	Unit	SP	None	None
AODV [89]	Source	Reactive	DV	Next	Unit	SP	None	None
DSR [54]	Source	Reactive	Enum	Source, full	Unit	SP	None	Backup
TORA [87]	Source	Reactive	DV	Next	Unit	SP	Tree	None
EARP [79]	Source	Reactive	DV	Next	Unit	SP	Tree	None
LQER [74]	Sink	Proactive	DV	Next	Unit/QoS	SP/Min	None	None
EAR [61]	Source	Reactive	DV	Next	Unit/QoS/Energy	SP/Min/Max	None	? [a]
Vidhyapriya [117]	Sink	Proactive	DV	Next	QoS	SP	None	None
Estrin [33]	Intermediate	Proactive	Local	Next	None [b]	None	Hierarchy	None
LEACH [46]	Intermediate	Proactive	Local	Next	None [c]	None	Hierarchy	None
LEACH-C [45]	Sink	Proactive	GK	Next	Energy/TX	Max/Min	Hierarchy	None
Kumar [66]	Source	Reactive	Local	Next	Energy/TX	Max/Min	Hierarchy	None
MEHR [3]	Intermediate	Proactive	Local	?	[d]	[d]	Hierarchy	None
HEED [125]	Intermediate	Proactive	Local	?	[e]	[e]	Hierarchy	None
Amis [2]	?	?	?	?	None [f]	None	Hierarchy	None
LAR [63]	Source	Reactive	Local	Next	Distance		None	None
GPSR [60]	Intermediate	Reactive	Local	Next	Distance	Min	None	None
PEGASIS [78]	Intermediate	Proactive	? [g]	All, full	Distance	Min	Tree	None
GEAR [126]	Intermediate	Reactive	Local	Next	Distance/Energy	Min/Max	None	None

Protocol/Author	Discovery				Costing		Selection	
	Initiator	Prep	Search	Table	Link/Node cost	Path cost	Topology	Multipath
WEAR [102]	Source	Reactive	Local	Next	Multiple [h]	Min	None	None
minimise cost/packet [105]	Source	Reactive	DV	Next	Energy	Max	None	None
Max-Min zP$_{min}$ [72]	?	?	?	?	Unit [i]	SP	None	None
Chang [22]	?	?	?	?	Energy/TX	SP	None	None
CMAX [58]	?	?	?	Next	TX/Energy	SP	None	None
HEAP	Source	Reactive	DV	Next	Unit [j]	SP	None	None
Dai [25]	Sink	Proactive	?	?	QoS	Min	Tree	None
Wu [121]	Sink	Proactive	GK	?	Energy/Degree	?	Tree	None
Jung [55]	Sink	Proactive	DV	Next	Unit	SP	Tree	None
EAD [12]	Sink	Proactive	DV	Next	Energy	Max	Role	None
Lee [67]	Source	Reactive	DV	Next	QoS	SP	None	None
Tran [115]	Source	Reactive	DV	Next	QoS	SP	None	Multipath [k]
SPEED [44]	Intermediate	Reactive	Local	Next	Distance/QoS	Min	None	None
MLRP [127]	Source	Reactive	DV	Next	Unit [l]	SP	None	None
Shah [103]	Sink	Proactive	DV	Next	Distance/Energy	Min/Max	None	Multipath
Ye [124]	Sink	Proactive	DV	None [m]	TX	SP	None	None
MLDA/MLDR [57]	Sink	Proactive	?	?	None [n]	None	None	None
CMLDA [26]	Sink	Proactive	?	?	None [n]	None	Hierarchy	None
Li [75]	?	Proactive	?	?	None [n]	None	None	None

Protocol/Author	Discovery				Costing		Selection	
	Initiator	Prep	Search	Table	Link/Node cost	Path cost	Topology	Multipath
Random walk [113]	None [o]	Reactive	Local	Next	None	None	None	Multipath
SMR [68]	Source	Reactive	Enum	Distributed	QoS	Min	None	Backup
Ganesan [35]	Source	?	?	Next	?	?	None	Backup
MOR [10]	Source	Reactive	DV	Next	Unit	SP	None	Multipath
REAR [42]	Source	Reactive	DV	Next	Unit [p]	SP	None	Multipath

[a] The authors state that a node will store "more than one route" to a sink. However, the nature of these multiple paths is not explained.

[b] Sources route to their clusterheads, which are promoted if a period of time elapses in which no neighbour is promoted.

[c] Sources route to their clusterheads, which are promoted based on the number of times the node has previously been promoted.

[d] Clusterheads are promoted randomly. Routing occurs based on shortest geographic distance to the sink.

[e] Clusterheads are promoted based on remaining energy and nodes join the cluster that will reduce their TX costs.

[f] Clusterheads are promoted in round-robin fashion and are swapped if they lose nodes or expend too much energy.

[g] It is not explained how, but each node is expected to know the positions of other nodes in the network.

[h] Node cost is based on a weighting of distance to sink, distance to routing voids and remaining energy.

[i] Node transmission powers are firstly dropped as much as possible such that the network remains connected.

[j] Only nodes with a certain amount of remaining energy may engage in discovery, and only using a certain transmission power.

[k] A secondary path is formed to avoid a node if that node becomes congested, in order to permit load balancing.

[l] RREQs dictate the power to be used when they are forwarded, thus, the routing protocol also minimises TX costs.

[m] Messages have an energy budget which is reduced at each hop. Nodes rebroadcast a message if they can route to the sink within its budget.

[n] Routes are not selected based on cost, they are selected in order to maximise total data transfer.

[o] No search for a (source, sink) path is carried out.

[p] Nodes must budget their energy and so may only appear on a certain number of (source, sink) paths.

4.10 Summary

A number of routing protocols are discussed in this chapter. However, none of them are obvious solutions to the problem of maintaining high source diversity for as long as possible.

Minimum hop routing, geographical routing and minimum energy routing all aim to reduce the energy expended by the network in routing a message from source to sink. However, it remains unclear whether minimum hop routing is a suitable tactic due to the possible use of unreliable links. Geographical routing relies on the ability to determine node positions, which may be energetically expensive. Minimum energy routing requires knowledge of transmission costs, which can only be determined after the event and only if edges are bidirectional. Furthermore, none of these approaches are cooperative. As discussed in Section 1.3, a source's optimal path may cause the overuse of a node that acts as a bottleneck to another sources.

Hierarchical routing and multipath routing can be applied to any routing protocol but do not themselves indicate which path is most suitable for each source.

Flow control routing protocols only maximise the capacity of the network, i.e. the total data transferred from sources to sinks. They do not take source diversity into account and so are not suitable.

Finally, load balancing routing protocols aim to keep all nodes active for as long as possible. However, they require knowledge of the remaining energy of nodes, which may be hard to collect and to maintain, since the energy reserves of nodes will continually change. Furthermore, postponing the time until the first node fails may cause every other node in the network to expire more quickly since load balancing inherently involves the use of sub-optimal paths. Thus, a network that may have lasted for many weeks after the expiration of one source could instead only last for a few days with all sources before the entire network fails.

Having determined that none of the routing protocols examined in this chapter are suitable for solving the source diversity problem, a new routing heuristic, known as node reliance, is presented in Chapter 6. It operates by determining how important each node is to maintaining source diversity. It is hypothesised that by forming (source, sink) paths through

unimportant nodes, the source diversity of the network may be maintained for longer. The node reliance heuristic is extended into a routing protocol in Chapter 8.

Chapter 5

Measuring Source Diversity

As stated in Section 1.3, this thesis addresses the issue of routing in a WSN such that a high source diversity is maintained for as long as possible. In order to determine how good a routing protocol is at solving this problem, it is necessary to have some metric that measures the level of source diversity over some period of time. This chapter examines the measurement of source diversity.

Section 5.1 examines pertinent metrics from a recent comprehensive survey of WSN metrics carried out by Dietrich and Dressler [28]. A new metric known as Connectivity Weighted Throughput (CWT), which is used to gauge the efficacy of applications that benefit from high source diversity for long period of time, is presented in Section 5.2. Section 5.3 discusses methods for comparing routing heuristics and protocols for maintaining high source diversity. Finally, Section 5.4 explains the simulation configuration that was used for the experiments in this thesis.

5.1 Common WSN Metrics

This section examines WSN metrics from the literature and discusses their suitability for determining whether a high source diversity over time has been achieved. Dietrich and Dressler [28] have recently carried out a comprehensive review of metrics, which are categorised into one of the following groups:

1. *number of live nodes*,
2. *sensor coverage*,
3. *connectivity*,
4. *sensor coverage and connectivity*,
5. *application quality of service requirements*, and,
6. *triple of (connectivity, number of live nodes, coverage)*, which is specified as "the definition provided by Blough and Santi" in [28].

Source diversity is entirely dependent on connectivity between sources and sinks. Thus, metrics that do not consider connectivity are unable to measure source diversity and can be immediately disregarded. Such categories include number of live nodes, sensor coverage and application quality of service requirements. The results of any metric falling into one of these categories may not be indicative of source diversity. For example, if only one node has expired, source diversity may be high. However, if that node partitions all sources from sinks, then all connectivity is lost, routing cannot take place between sources and sinks and source diversity drops to 0. Thus, the result has little meaning.

The remaining categories of metric are connectivity, sensor coverage and connectivity and the triple of (connectivity, number of live nodes, coverage). Specific metrics that fall into these categories are:

1. total data transfer, which is included in the connectivity category in [28],
2. k-of-n lifetime, which is covered in both the connectivity category and the number of live nodes category in [28],
3. sink connectivity, which is included in the connectivity category in [28],
4. the triple of (connectivity, number of live nodes, coverage), and,
5. sensor coverage and connectivity.

The suitability of each of these metrics for measuring source diversity is discussed in the following sections:

5.1.1 Total Data Transfer

Several authors have considered measuring lifetime in terms of the maximum amount of data transferred by the network. Baydere [7] considers network lifetime "in terms of total messages transmitted". Yu [126] considers the "number of data packets sent and successfully delivered before network partition". Other authors consider a slightly more general measurement, such as Giridhar [37] who measures lifetime in terms of "the maximum number of times a certain data collection function or task can be carried out without any node running out of energy" and Olariu [83] who measures "the number of successful data gathering trips (or cycles) that are possible until connectivity and/or coverage are lost".

More generally, these metrics measure the total data transferred during the operation of the network, which is considered to terminate at some time - at first node death, on network partition or when all nodes expire. An advantage of this metric is that it is possible to measure the total data transferred across a single node or in response to some event of the network such as a node expiring or some data of interest being generated. Thus it is possible to measure the effect of an event or how specific nodes are used.

When measuring total data transfer, only the quantity of received data is considered. The quantity of information that can be inferred from the data has no effect on the metric. Consequently, as reported by Dietrich and Dressler [28], the total data transfer metric may be ineffective where data aggregation is employed. Additionally, it is noted that the metric may be of limited use where sources forward data on behalf of other sources. These situations are discussed below.

Data aggregation is a technique intended to reduce energy expenditure in routing protocols. It aims to combine data from multiple sources into a single, smaller piece of data. A well known routing protocol that performs data aggregation is LEACH [46]. In LEACH, cluster heads aggregate data from their neighbours and forward the smaller, aggregated data to the sink. Since the total data transfer metric only measures the quantity of data received by sink nodes and data aggregation reduces the amount of data transmitted, the metric does not accurately reflect the quantity of information received.

When sources forward data on behalf of other sources, they may expend more energy than they would if they had generated the data themselves. Consequently, to reduce the energy consumption of sources, the optimal solution is for sources to refuse to forward data that

originates elsewhere. However, this approach may lead to a loss of source diversity, since some sources may be unable to send their data towards a sink. This is referred to as the *source-forwarding problem*. Attempting to increase source diversity by forcing sources to forward all data may be insufficient. A poor application or routing protocol may cause sources that rely on source-forwarding to expire. Thus, the application may appear to be near optimal according to the total data transfer metric even though the source diversity would be reduced.

To illustrate the source-forwarding problem, consider a scenario in which the inefficient *maximum-hop routing* is employed within the network shown in Figure 5.1. The network consists of nine nodes, including two sinks and seven sources (A-G). A subset of sources B-F is referred to as group Z for convenience. The source nodes generate data and send it towards one of the sink nodes. A directed edge from a node X to a node Y indicates that Y receives every transmission made by X.

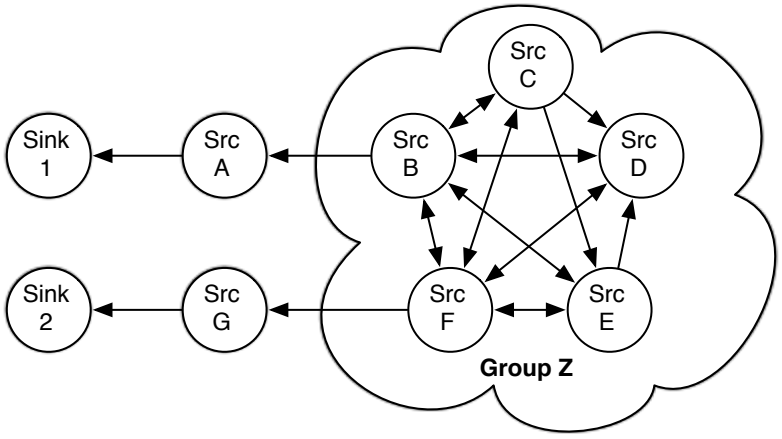


Figure 5.1: An example network with two sinks and seven sources. Sources B-F are referred to as group Z for convenience

In maximum-hop routing, data is routed from a source node to a sink node through as many distinct intermediate nodes as possible. Maximum-hop routing is used here as an example of a poor routing protocol. Note that in the network of Figure 5.1, there exists a path from every source in Z that travels through all the other sources of group Z. Consequently, any data generated by a node in Z will be transmitted and received five times before it reaches either A or G. Every transmission made by a source in Z is overheard by sources B and F, which connect Z to the sinks. Thus, every piece of data generated by a node in Z will

be received/overheard four times and transmitted once by both B and F. Consequently, sources B and F expend significantly more energy than nodes A and G, and will quickly expire, disconnecting group Z from the sinks. The optimal total data transfer solution then remains, since the sources that are still connected to the sinks (A and G) need only expend energy to produce and transmit their own data. Thus, a high total data transfer can be achieved. In this example, no source has refused to forward data on behalf of another source. Due to the use of maximum-hop routing the optimal total data transfer solution quickly emerges due to the inhibition of source-forwarding. Thus, as measured by the total data transfer metric, the inefficient maximum-hop routing protocol performs well, even though the received data is from a small variety of sources.

5.1.2 k-of-n Lifetime

k-of-n lifetime measures "the time during which at least k out of n nodes are alive". However, the fact that a proportion of nodes are functional has no bearing on the source diversity of the network. Even if the metric was to be changed to represent the time for which k out of n sources were alive, a live node may not be capable of forwarding messages to a sink if it does not have a (source, sink) path. Thus, the metric is hard to use as a measurement of source diversity over a period of time.

Similarly, the *n-of-n lifetime*, which is the time during which the network functions "without any node running out of energy" [37] or "the time until the first node depletes its energy" [121] is of little use, since the loss of a node does not necessarily disconnect any sources from sinks. Even if the metric was changed to instead consider sources rather than simply nodes, a node may be active and not connected to a sink. Therefore, the metric cannot measure source diversity.

5.1.3 Sink Connectivity

Carbunar [16] measures connectivity lifetime "as the percentage of nodes able to route to the collection point" (i.e. the sink). Carbunar assumes that all nodes act as source nodes and that only a single sink exists. However, by trivially modifying the metric, it is possible to measure the percentage of source nodes with a path to any sink. Carbunar's approach very

precisely reflects source diversity. However, there are two limitations to sink connectivity as a metric. Firstly, it does not provide a numeric output. Thus it is difficult to compare routing protocols. Secondly, in some scenarios the sink connectivity metric simplifies to total data transfer, with the consequent limitations discussed earlier.

It is not clear how two different scenarios or executions could be compared based on sink connectivity. For example, consider two scenarios, A and B. In scenario A, 100% of source nodes have a path to a sink for 10 seconds. After that time, 50% of source nodes have a path to a sink for 10 seconds before the network expires. In scenario B, 100% of sources have a path to a sink for 15 seconds. After that time, no source has any path to a sink. It is not possible to determine which scenario maintains the better connectivity without specific goals. For example, if the aim is to keep 100% connectivity for as long as possible then scenario B is superior. But if the aim is to maintain some degree of sensor coverage for as long as possible, then scenario A is better. It is also possible to consider the average connectivity of the network. In the above example, both scenarios maintain an average connectivity of 75% for 20 seconds. However, considering the average connectivity implies that a good average connectivity is the goal. Hence, the sink connectivity metric is ideal for measuring the connectivity of a specific scenario whose specific connectivity goals are known. However, it is not suitable for evaluating a generic feature, such as a routing protocol, which may not have been specifically designed for a particular goal.

The sink connectivity metric is directly related to total data transfer in certain application scenarios. For example, in an application that periodically routes data from sources to sinks, any source having sink connectivity increases total data transfer. Consequently, the limitations that were discussed in Section 5.1.1 may also apply to sink connectivity.

5.1.4 Triple of (connectivity, number of functional nodes, coverage)

Blough and Santi [11] consider three factors in measuring network lifetime. Specifically, their metric returns the first time until one of the following conditions drops below user-defined thresholds:

1. the number of active nodes in the network,

2. the volume being sensed by the sources (coverage), and,

3. the largest number of connected nodes (connectivity).

Adjusting each of the thresholds may customise the metric. Such a metric is ideal for applications whose usefulness can be precisely specified in terms of active nodes, sensor coverage or node interconnectivity (or some subset of these factors). Blough and Santi's metric is especially useful where different source nodes may have different coverage. For example, if some source nodes have more powerful sensors, they may have a greater coverage than other sources. Consequently, their loss is likely to have a greater impact on the network.

By measuring the number of active nodes in the network, the metric rewards applications in which a variety of source nodes are available. As noted by Dietrich and Dressler [28], considering the largest number of connected nodes is not a good measurement of connectivity for (source, sink) architectures since many nodes (or sources) being connected to each other is orthogonal to sources being connected to sinks. As the networks under consideration involve data being sent from sources to sinks, this measurement of connectivity is inappropriate.

5.1.5 Sensor Coverage and Connectivity

Sensor coverage and connectivity considers the duration for which a network is both connected and has coverage (as previously defined). Specific metrics that consider both coverage and connectivity can be trivially emulated by the sink connectivity metrics discussed in Section 5.1.3 or Blough and Santi's lifetime triple discussed in Section 5.1.4. Sensor coverage and connectivity metrics are therefore not examined any further.

5.1.6 Conclusions

In order to determine which routing protocol is best at achieving a high source diversity for as long as possible, it is necessary to provide some metric to measure source diversity over time. However, none of the metrics that have been discussed are suitable.

A suitable metric must reward high source connectivity for long periods of time. However, the metric must also compensate for the source-forwarding problem and the data aggreg-

ation problem as discussed in Section 5.1.1. Based on these requirements, a new metric known as Connectivity Weighted Transfer is introduced.

5.2 Connectivity Weighted Transfer

Connectivity Weighted Transfer (CWT) is a new metric that may be used to measure source diversity over time. It rewards the connection of as many sources to sinks for as long as possible and is able to compensate for the source-forwarding and data aggregation problems explained in Section 5.1.1

5.2.1 Definition of Connection

A source is connected while it is active and a sink is receiving the data it generates. WSNs typically transmit discrete data packets rather than continual streams of data. Therefore, an abstraction must be used to represent the discrete data packets as streams of data. The operational time of the network is split up into non-contiguous *frames*. A source is connected to a sink for the entirety of a frame F if any sink receives a packet from that source during frame F.

When one frame ends, another does not begin until a sink receives a packet from a source. Delaying the creation of a frame until it is required makes it less likely that a source will connect at the end of an otherwise empty frame, reducing the perceived connectivity of the sources. For example, consider a network with a frame length of five seconds. If data is received by source A and source B at times 4 and 6 respectively then they will be perceived to be simultaneously connected if the frame starts at time 4, but not simultaneously connected if the frame begins at time 0.

5.2.2 Metric Theory

The CWT metric combines the quantity of transferred data and the number of sources that have provided the data. The metric does not consider the length of time for which a source has been connected, since this is application dependent, e.g. an application that generates

data every one second will expend energy more quickly than an application that generates data every five seconds and so will remain connected for less time. By avoiding any application dependent variables from the metric, two application scenarios can be compared using the CWT result.

In order to represent the connectivity of sources, the metric should reflect the number of sources connected within a single frame. However, to compensate for the source-forwarding problem discussed in Section 5.1.1 there should be a non-linear relationship between the number of sources connected in a frame and the result of the metric. For example, having 10 sources connected for one second must be more highly rewarded than having 1 node connected for 10 seconds.

5.2.3 Formal Description

Formally, the CWT metric for a network may be expressed as:

$$\sum_{i=1}^{f} n_i^x (b_i \cdot n_i) \tag{5.1}$$

where:

- f is the number of frames from activation of the network until all sources expire
- i is the frame number
- n_i is the number of connected sources in frame i
- b_i is the average number of bytes transferred per source in frame i
- x is a weighting factor, which is discussed further in the next section

The metric multiplies the total data transfer during a frame by an enhancement function, i.e. the number of connected sources raised to the power of x. This total is then added across all frames to produce the CWT value for the application. Many choices were available for the enhancement function. The function n_i^x was selected due to its versatility and scalability.

The n_i^x function is versatile. Depending on the choice of weighting factor x, the function could be made to encourage connectivity to any required extent. Thus, a large range of application behaviours can be examined, including those for which improved connectivity is irrelevant, desirable, essential or even undesirable.

The proposed enhancement function is also scalable, since its growth rate is relatively slow. Consequently, a single weighting factor may allow the comparison of both large and small networks for a single application. Conversely, functions such as x^n grow very quickly and it may be impossible to find a weighting factor that would allow the analysis of an application in networks whose size varied.

5.2.4 Weighting Factor

The weighting factor x is used to bias whether it is desirable to have more sources connected for shorter periods of time or fewer sources connected for longer. Depending on the value assigned to it, the metric can be made to operate in different ways.

For values of $x > 0$, the bias is in favour of having many sources connected for a short period.

For values of $x < 0$, the bias is in favour of having a small number of sources connected for a long period.

For $x = 0$, no bias is applied. In this case, CWT simplifies to the *total data transfer* metric.

For an example of weight assignment, consider a scenario in which the number of connected sources and the total data transfer are consistent across all frames. The user indicates that it is c times more preferable to have n sources connected for one second rather than one source connected for n seconds. With such a value, c, the user can determine the weighting factor as follows:

$$\sum_{i=1}^{1} n^x(b \cdot n) = c \sum_{i=1}^{n} 1^x(b \cdot 1) \tag{5.2}$$

The sigma operators can be trivially expanded, since the frame number does not affect any of the variables. Thus:

$$n^x(b \cdot n) = cn \cdot 1^x \cdot (b \cdot 1) \tag{5.3}$$

By combining and simplifying:

$$n^x = c \tag{5.4}$$

Finally, it is possible to solve for x by taking the natural logarithm of both sides of the equation:

$$x = \frac{ln(c)}{ln(n)} \tag{5.5}$$

By substituting Equation 5.5 into Equation 5.1, it is possible to derive an equation that allows the user to quantitatively express how much more desirable it is to have multiple sources connected over longer periods of time. For example, if the user wishes to set the weighting such that having 10 sources connected for one second is 50% more desirable than having 1 source connected for 10 seconds, then the value of x = ln(1.5)/ln(10) = 0.18. A domain expert for whom the data is being collected may best determine the weighting.

5.2.5 Operational Concerns

Some operational concerns of using the CWT metric are now considered. Firstly, the frame length, which so far has not been specified, is addressed. Secondly, the operation of the metric in a delay tolerant network is considered. In a delay tolerant network, the source and sink may not be continually connected. Instead, data may be forwarded from a source to an intermediate node where it accumulates before the intermediate node and sink become connected. At that time, the data can be forwarded along the remainder of the path to a sink. Thus, the connectivity of a source during some time frame cannot be measured by the arrival of data from that source at a sink. Thirdly, a variant in which sources route streams of data to the sink node, rather than discrete packets that were assumed in Section 5.2.1 is considered. It is also shown how the CWT metric may be modified for the use of query or event-based applications. Finally, the issues of data aggregation are discussed and a possible modification of the CWT metric is presented to handle networks in which an

intermediate node may aggregate the incoming data from other nodes in an effort to reduce the size of data that it forwards towards the sink.

These concerns are discussed in more detail in the following sections.

5.2.5.1 Effect of Frame Length

The length of each frame dictates the granularity of the connectivity calculation. Large frame lengths potentially introduce errors, since a source may be considered connected for a long period even if a source immediately expires after sending one message at the start of a frame.

However, setting very low values for frame lengths is also unsuitable. If the frame length is particularly small, it is unlikely that many sources will be perceived as being connected simultaneously. For example, consider that a WSN may only be capable of receiving a single transmission at a time. If the frame length is set to the length of time required to receive a transmission, then it becomes impossible for two sources to be considered as simultaneously connected.

Consequently, a frame length that matches the frequency with which data is sent by the source nodes is recommended. If sources continue to send data at this rate, then (discounting propagation delays) they would be expected to remain connected from activation until expiration.

If sources send data at different rates, the greatest common factor is used as the frame size. Sources that generate data at slower rates are considered to be connected for the number of frame lengths that represent their data generation rate. For example, consider sources A and B that send data at 1 packet of x bytes every second and 1 packet of y bytes every five seconds, respectively. The greatest common factor (i.e. 1 second) is used for the frame length. Source A is considered connected during any frame in which a packet from A is received at a sink. Source B is considered connected for any frame in which a sink receives a packet from B, as well as the subsequent 4 frames. Source B is therefore considered connected for 5 seconds (its data generation rate) whenever a sink receives a packet from B. The number of bytes received from a source must remain the same. Therefore, in each of the five frames that source B is considered connected for, it is treated as having delivered 0.2y bytes.

5.2.5.2 Delay Tolerant Networks

In a delay tolerant network, the path between source nodes and sink nodes may only be intermittently available. Data may accumulate at an intermediate node, which is only periodically connected to a sink. Thus, data may be delayed in being forwarded from sources to sinks. An example of such an application is ZebraNET [128]. In ZebraNET, data is disseminated throughout the network to all nodes. When one of the nodes eventually comes in contact with the sink, all the data is downloaded. Therefore, a sink may instantaneously receive several frames worth of data all at once. Thus, the use of frames, as previously defined, is not appropriate.

To modify the CWT metric for such an application, it is necessary to adjust the definition of connection to the following: A source A is connected to a sink for the entirety of a frame F if source A generates any data during frame F that is subsequently received by a sink. Thus, each source must be able to attach a timestamp to the data it generates. Essentially, the network disregards any propagation delay experienced in routing data from sources to sinks.

5.2.5.3 Continuous Data Streams

In Section 5.2.1 it was assumed that sources would send discrete packets of data rather than continuous streams of data. Consequently, a source A is considered connected for the entire length of a frame if a sink receives a packet of data from source A during that frame. However, it may be the case that a continuous stream of data is sent from sources to sinks, i.e. the source continually generates data and forwards it to an intermediate node, each intermediate node continually receives and transmits data to the next hop in the path and the sink continually receives data. If sources send continuous streams of data, the duration for which each source is connected is precisely known. For example, in an application that sends live video streams across a WSN, data may be continuously sent from sources to sinks. As such, a more precise measurement of source connectivity can be made. To handle this variant, frames are permitted to have different durations. The beginning of a new frame is used to represent a change in the number of simultaneous (source, sink) streams in the network.

If the network uses multiple sinks, they must coordinate among themselves to determine

when new frames begin or end. A simple solution to this is for each source to log the times at which each of their incoming streams starts and ends so that the metric calculation can be performed offline. The CWT equation otherwise remains identical.

5.2.5.4 Event and Query-based Systems

In an event or query-based system, data is only sent either in response to a user request or in response to receiving some data of interest. Consequently, a number of frames may pass in which data is not received from a particular source and so according to the connection definition provided in Section 5.2.1, that source will not be considered to be connected. However, it is assumed is that sources must periodically notify a sink that they are still active. If the sources do not give some indication that they remain functional, there is no means to determine that the network is still operational. It is therefore proposed that these periodic notifications are used to measure the CWT rather than the data produced as the result of an event occurring.

5.2.5.5 Data Aggregation

It is assumed that a network utilising data aggregation contains a number of *aggregator* nodes that forward data either to additional aggregator nodes or to the sink.

Modifying CWT to handle data aggregation requires an adjustment to the connection definition: A source A is connected to a sink for the entirety of a frame F is source A generates any data during frame F that is subsequently received by:

1. a sink, or,
2. an aggregator that uses the data to generate a packet that is received by a node covered by one of these two categories.

It is also important to address the issues surrounding variable b_i, which refers to the average number of bytes transferred per source in frame i. Due to data aggregation, the number of bytes sent by each source may differ from the number of bytes received by the sink. Since it is desirable for the CWT metric to measure the quantity of information rather than the

quantity of data, variable b refers to the average number of bytes transferred per source *before aggregation*.

5.2.6 Example

This section uses an example to illustrate the limitations of common metrics as well as demonstrate the effectiveness of the CWT metric. The network shown in Figure 5.2a is used as an example, and contains one sink node (Z) and nine source nodes (A-I). For convenience, it is assumed that communication between nodes is perfect and bidirectional. Thus, an edge between two nodes in the diagram indicates that those nodes can communicate with each other.

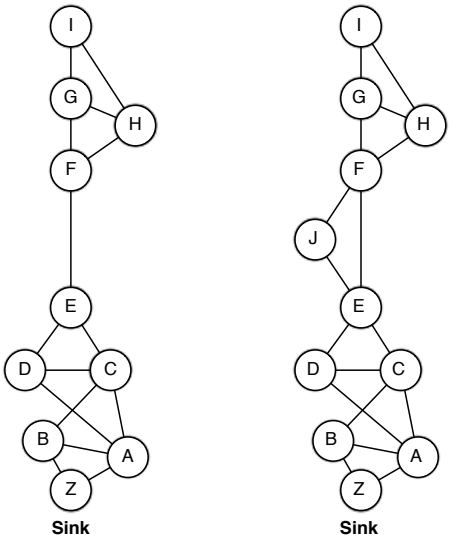


Figure 5.2: Two example networks with and without source J

The proposed application mimics that of Tolle's redwoods microclimate [114] project in which the sink node is placed at the bottom of a tree and the sources are placed at different heights. Every five seconds, sources generate a piece of data, and send it via the minimum hop path towards the sink. Since Tolle does not specify a data packet size and sources may produce data packets of different sizes, particularly in different deployments, the simulation randomly determines each packet's data size between 2 and 100 bytes. The aim of the application is to calculate a temperature gradient of the tree for as long as possible.

Thus, the application benefits from having more sources (to produce a higher resolution temperature gradient) for longer.

The application is simulated using the Castalia 1.3 simulator, whose use is justified in Section 5.3. The nodes are based on the TMote sky [101]. However, in order to reduce the simulation time, each node is given only 14.58 J of energy as opposed to the 29160 J that would be provided by a pair of AA batteries. The sink is given as much energy as the simulator would allow. Experimental observations are shown in Table 5.2.6.

Time	Event
32539 (9 hours)	Source C expires
33905 (9.4 hours)	Source D expires
103735 (28.8 hours)	Source A expires
125923 (35 hours)	Source B expires

Table 5.1: Simulation of experiment 1

Source C is the first node to expire. However, its loss is unlikely to render the network unusable since source D can be used in place of source C for routing. When source D expires, only sources A and B remain connected to the sink. However, it may be possible to estimate the temperature gradient at future times using only A and B. If the network remains usable with only sources A and B, then its useful network lifetime (35 hours) is almost four times greater than n-of-n lifetime suggests.

The network can be modified to that shown in Figure 5.2b by inserting an additional source J to the void between sources E and F in order to obtain a higher resolution temperature gradient. The observations are shown in Table 5.2.6.

Time	Event
28511 (8 hours)	Source C expires
29450 (8.2 hours)	Source D expires
105538 (29.3 hours)	Source A expires
122628 (34 hours)	Source B expires

Table 5.2: Simulation of experiment 2

By adding source J, source C is required to forward additional data originating from other

sources (the source-forwarding problem). Sources C and D expire 13% more quickly than in the initial scenario. However, during that time the temperature gradient is more precise due to the presence of source J. Over the entire experiment, the introduction of J causes the total data transfer to drop from 4.39 MB to 4.35 MB. Thus, the total data transfer metric may be ineffective at representing the usefulness of the network. Conversely, the CWT metric with $x = 2$ increases from 2.45×10^8 to 2.93×10^8 (19.6%).

Thus, the CWT metric correctly reflects the improved temperature gradient that can be achieved by the introduction of source J whereas the total data transfer metric incorrectly suggests that the addition of source J has a negative effect on the application, despite the increased temperature gradient resolution that can be achieved.

5.2.7 Conclusion

Existing metrics are unsuitable for comparing two WSNs to determine which is better at maintaining a high source diversity for a long period of time. Classical approaches such as total data transfer or sink connectivity do not compensate for the increased energy expenditure caused by sources routing data on behalf of other sources (the source-forwarding problem). A new metric, known as CWT has been introduced, which can be used to reward networks that maintain high source connectivity. By utilising a user defined weighting, an application's performance can be measured according to its ability to keep numerous sources connected.

It has been shown how the metric may be simply adapted to be used in several varieties of WSN application, including those that use continual data streams, discrete packets and delay tolerant networking.

A user can design a network that maximises the CWT by constructing their network such that no sources have any intermediate nodes in common (for a positive weighting factor) or such that each source must route through as many other sources as possible (for a negative weighting factor) and vice-versa for minimising CWT. For example, a network in which no sources have intermediate nodes in common can be achieved in a star topology in which each source is a direct neighbour of a sink. Conversely, an example network in which sources must route through many other sources is a linear network.

5.3 Experimental Methodology

This section explores the different options available for experimentally analysing the effectiveness of routing heuristics or protocols at maintaining high source diversity for long periods of time. The different options include:

- algebraic models,
- a physical deployment of nodes, and,
- a simulator.

The following sections examine each of these options to determine which is most suitable.

5.3.1 Algebraic Models

In an algebraic model [77] [107] [37] [76] [75], a network is expressed as a series of mathematical equations. Typically, the equations are combined to form a linear programming problem where the objective function is to maximise some variable that reflects the network's lifetime. By adjusting the variables to reflect the behaviour of a routing heuristic, it is possible to see how the lifetime variable changes.

Algebraic models typically make assumptions regarding the network or the knowledge available. For example, Li [75] assumes that the cost of forwarding data from one node to another is fixed. In practice, as discussed in Section 2.5.2, wireless communication is unreliable and it is therefore unreasonable to generalise its behaviour by the use of equations. For example, with random probability, messages may be lost or may have to be retransmitted at additional cost. Neighbouring nodes may also overhear the message and expend energy. Each of these actions may lead to randomness in the rate at which nodes exhaust their batteries. In a random network, it is very difficult to predict the effect of node expiration.

Another disadvantage of an algebraic model is that modelling a routing protocol is significantly harder. As discussed in Chapter 3, a routing protocol includes a discovery task in which available paths are collected. It is extremely difficult to algebraically express the

messages that are exchanged during the discovery task, since it may constitute multiple phases. Furthermore, each message may cause different behaviour at different nodes depending on the content of the message and the state of the node (i.e. messages that it has previously received, its ID, any timers that it has set up, etc.). Furthermore, the topology of the network may be random, making it difficult to determine which nodes might receive each message. Thus, modelling a routing protocol becomes difficult if not impossible.

5.3.2 Physical Deployments

Physical deployments of nodes are not commonly used for examining the effectiveness of routing protocols, due to difficulties with keeping conditions constant. Since radio communications can be affected by the physical environment, node positions and battery voltages, it would be almost impossible to keep these parameters constant across a series of executions of an experiment.

5.3.3 Simulation

Simulation has the advantage of allowing experiments to be carried out in specific scenarios in reproducible circumstances. Even the *random seeds* that govern the unreliable nature of radio communications or the random readings from node sensors can be reproduced. This permits one routing protocol to be compared to another in identical circumstances, allowing a fair comparison.

Another advantage of simulation is that the virtual environment can be fully controlled by the user. Simulations may also be easily carried out in a virtual environment that would be impractical to run in real life. For example, it would be infeasible to determine how a WSN responds to detecting the boundary of a real forest fire. However, in a virtual environment, the forest fire can be created many times over and controlled in any way the user desires. An unlimited number of nodes can be created and placed, either randomly or in a fixed pattern whereas in a real deployment, budgetary constraints would limit these factors.

It is important to ensure that the simulations are executed on a single architecture in order to precisely reproduce each experiment. Running the same experiment on different architectures may produce unknown results, even if random seeds remain the same.

5.3.4 Chosen Methodology

Since physical deployments make reproducibility of experiments too difficult and an algebraic analysis of routing protocols may require unrealistic assumptions regarding the network, this thesis has carried out the experimental analysis of routing heuristics and protocols by the use of simulation.

Several choices of simulator were available. These included:

- TOSSIM [70] and the PowerTOSSIM extension [104] for power analysis,
- Atemu [52],
- ns-2 [34], and
- Castalia [90], an extension of the OMNeT++ platform [116].

TOSSIM [70] is a simulator for the TinyOS [49] operating system. The analysis of power usage is not possible with TOSSIM. However, the analysis can be accomplished with the use of the PowerTOSSIM extension [104]. Performing analysis of a program's power usage using the PowerTOSSIM extension requires the program to be compiled for both the PC architecture and the architecture being simulated, i.e. a node. PowerTOSSIM then cross references the code produced for the WSN architecture with that generated for the PC and thereby calculates the number of processor cycles on the node required for each instruction block being executed on the PC. If the energy expenditure per processor cycle is known for the simulated architecture, the cost of processing can be calculated as the program is simulated, allowing a power analysis of the program to be carried out.

PowerTOSSIM also provides a number of plug-in replacements for TinyOS components that allow the program to be run on PC rather than a node. For example, PowerTOSSIM provides replacement radio and sensor modules. These replacement modules permit power analysis by recording the number of times each module is accessed. As with computation, if the power consumption of each hardware component is known, the energy expended by the program as a result of using the simulated hardware can be calculated. The radio layer provided by TOSSIM seems to be limited to a probabilistic bit error model where the link between each pair of nodes must have a specified probability of error.

Atemu [52] is an instruction level simulator, i.e. an emulator, which supports the AVR processor and several peripheral devices. As with TOSSIM, it allows the simulation of a WSN application without requiring the application to be rewritten or ported to another language. The two simulators differ in that TOSSIM emulates TinyOS applications in which node behaviour is bundled together with the TinyOS operating system. Conversely, code that is emulated by Atemu only includes an operating system if the programmer manually links it in. Atemu emulates each individual compiled instruction as it would be executed on a hardware platform. This is useful for the verification of code which is due to be installed on a WSN deployment, but as each instruction must be interpreted by the simulator, it can be up to 20 times slower than the code cross referencing approach of PowerTOSSIM [104].

ns-2 [34] is a discrete event simulator which is written in C++. It is widely used in the simulation of routing protocols. Unlike TOSSIM and Atemu, which only execute a piece of code, ns-2 also allows the user to define *simulation parameters*. The parameters allow the user to specify what happens to the network over a period of time. For example, the parameters might indicate the initial positions of nodes, the occurrence of an event, the movement of nodes or the sudden death of nodes. By separately defining the parameters, the programmer can examine the effect of a routing protocol under particular circumstances without having to hard-code this information in the application code being run. In ns-2, parameter files are written in TCL [85].

Castalia [90] is an extension to the OMNeT++ platform [116]. As with ns-2, the application code and simulation configurations are separated. Simulated code is written in C++ and Castalia provides a number of text-based configuration files, which can be modified to specify simulation parameters. The communications model of Castalia is flexible, and can be used to represent both ideal radios and realistic radios. The communications model was validated against real deployments in order to verify its correctness. Finally, Castalia has the advantage that it was specifically created with WSNs in mind.

Having examined these possible simulators, Castalia was chosen to carry out the simulations, for the reasons given below.

Firstly, the simulator was designed for the domain of WSNs. Consequently, it is likely that any assumptions made by the authors of Castalia would be appropriate for the simulations being executed. Furthermore, the simulator comes with a number of configuration files which are suitable for WSNs such as the TMote hardware and the CC2420 radio.

Secondly, Castalia provides a realistic radio model, which has been based on empirical data from WSN deployments. Consequently, the realistic radio model would be expected to be accurate and reflect real-life deployments. This latter point is particularly important, since routing protocols that perform well when communication is perfect often perform poorly when used with a more realistic mode of communication [109]. Consequently, it is desirable to examine routing protocols using radio models that are as realistic as possible. Finally, any results that are attained by using Castalia can be considered to be reliable since the simulator has been validated by its creators.

The experiments in this thesis were carried out using Castalia 1.3 [5] which extends OMNeT++ 3.3 [84] and was the most recent version available at the time.

5.4 Simulator Configuration

The following sections discuss the configuration of Castalia used in this thesis.

5.4.1 Node Connectivity

Node or network *connectivity* refers to the ability of nodes to communicate with one another. It reflects the communication network that exists, as opposed to the physical *placement* of nodes or the forced network *topology* of nodes, which might dictate how nodes are ordered to communicate with one another. A connectivity map can be used to represent the connectivities of nodes throughout the network.

In all experiments, nodes were configured into a single, large, unpartitioned network. To properly analyse the effect of an increasing network size, it was important that the deployed nodes acted as a single network rather than a collection of smaller independent networks. Nodes were placed in such a manner that each node was indirectly connected to every other node in the network, assuming an ideal model of communication existed between the nodes.

The Castalia 1.3 User Manual [4] states that the realistic wireless channel model used in Castalia can be expressed by Equation 5.6 where $PL(d)$ is the path (signal) loss at a distance d metres and η is the path (signal) loss exponent. $PL(d_0)$ is the known path loss

at a known distance d_0 metres away from the transmission source and X_σ is a zero-mean random variable with a standard deviation of σ.

$$PL(d) = PL(d_0) + 10\eta \log(\frac{d}{d_0}) + X_\sigma \tag{5.6}$$

The average effect of the zero-mean random variable across all radio communications is zero and so shall be disregarded here. For the CC2420 radio, which is part of the TMote Sky [101], Castalia states a path loss of 55 dBm at a range of 1m and a path loss exponent of 2.4. Castalia further defines the receiver sensitivity of the CC2420 radio to be -95 dBm and the maximum transmission power to be 0 dBm. Therefore, for a node to receive another node's transmission sent at maximum power, the path loss must be less than 95 dBm. Using these values and Equation 5.6 to solve for d, the distance at which a signal cannot be received, provides Equation 5.7. The solution to the equation is that d = $10^{\frac{5}{3}}$ which is approximately 46.42m.

$$95 = 55 + 24 \log(\frac{d}{1}) \tag{5.7}$$

In each of the experiments in this thesis, each node was placed no greater than 46.42 metres from another node in order to theoretically ensure network connectivity. In practice, the nature of radio communication is largely random and prone to interference from other nodes as already discussed in Section 2.5.2. Therefore, this requirement may not ensure network connectivity in a realistic scenario.

5.4.2 MAC Protocol

The MAC protocol has a large effect on both the probability of successful radio communication and also the amount of energy consumed by the radio. In particular, a WSN MAC protocol may control the *duty cycle*, which is defined as the proportion of time that a node spends listening or receiving transmissions rather than in a low power state. A MAC protocol may also carry out *carrier sensing*, which attempts to reduce radio interference by waiting for an absence of local radio traffic before beginning a transmission. Other factors such as time or channel division, retransmission attempts and retransmission times, among

others may all contribute towards both the probability with which transmissions are successfully received and the amount of energy consumed.

The choice of MAC protocol is heavily dependent on the WSN application. For example, carrier sensing can only take place if bidirectionality is ensured, which may not be the case (as discussed in Section 2.5.2). In order to avoid selecting some arbitrary MAC protocol that might only be applicable to a small number of applications, the simulations were run without the benefit of a MAC protocol, i.e. no carrier sensing or message retransmission took place. Duty cycling was unnecessary and not provided because nodes were configured to not expend energy listening to the radio unless they were receiving or overhearing something. This latter decision is explained in more detail and justified in Section 5.4.3.

In some of the experiments that were carried out, a perfect communication model was required. In order to provide this, the simulator was configured to deliver each message in a unique network-wide time slot during which no other communication could occur. Consequently, it was not possible for a receiving node to be transmitting, thus preventing message collision. Furthermore, Castalia's perfect communication model was used in which the probability of success for receiving a message on a node in range was 100%.

5.4.3 Communications Energy

In order to maximise the area being sensed, it was assumed that the transmission power of node radios would be at maximum power. As stated in Section 2.5, the computation costs are assumed to be negligible and consequently this thesis only considers the energy expended by using the radio. The TMote sky [90] uses a CC2420 [23] radio. When it operates at 3.3 V, which is assumed by the CC2420 data sheet, the power consumption is 57.42 mW for transmission, 62 mW for receiving or listening and 1.4 mW for sleeping.

A radio is considered to be listening when it is searching for incoming transmissions, but nothing is being received. Receiving differs from listening in that the radio detects a transmission and then decodes it. If the radio is not transmitting, receiving or listening it is considered to be sleeping.

In order to better measure the effects of using different routing protocols, the sleeping energy was reduced to 0 mW. Without this modification, the majority of energy in each

simulation would be expended in sleeping. Even though the energy consumption of radio transmission is approximately 44 times higher than that of sleeping, the transmission lasts only a small period of time. For example, the CC2420 [23] transmits data at 250 kbps. The radio therefore takes 0.004 seconds to transmit 125 bytes of data. Even if the node were to transmit 125 bytes of data every second and spend the remainder of the time sleeping, 85.4% of a node's energy would be expended while asleep. It would be very difficult to analyse the effect of using different routing protocols, since minimal energy savings would be available. To make the differences in routing protocols more obvious, the sleep energy was reduced to 0 mW.

The other modification that was made was to reduce the listening energy to 0 mW. As already justified in Section 5.4.2, no MAC layer was provided. Consequently, no duty cycling took place and so the radio would constantly listen for incoming transmissions, eliminating any possibility for energy savings. Counter-intuitively, it would be beneficial for a routing protocol to cause a node to continuously transmit rather than listen/receive, since the energy expenditure per second is lower. The listening energy was therefore reduced to 0 mW so that energy would only be expended in receiving data.

These modifications are the smallest set of changes from a realistic physical deployment that are necessary in order to accurately measure the effect of using different routing protocols. Although the modifications are idealised, they are not unrealistic and may be provided by a well designed duty cycling system and a node that is capable of a very low power sleep mode.

5.4.4 Handling Multiple Networks

In order to determine a routing protocol's overall ability at maintaining (source, sink) diversity, it is necessary to examine it in a variety of networks, each with different parameters. Furthermore, since some parameters are random, such as the seeds that govern the random radio communication or the positions of nodes (which may be randomly placed), it is necessary to repeat the experiment several times, each time varying the choice of random parameter. If the experiment is not repeated with different random parameters, the randomly generated network may happen to be the only one in which a particular routing protocol performs well. Thus, that protocol would appear to perform well when on average it may perform poorly.

The following sections indicate how the results of routing protocols are compared in multiple networks in order to determine which routing protocol is the best, overall. Section 5.4.4.1 discusses the need for score normalisation, which is when the top performing routing protocol for a single network is given a score of 1 and other routing protocols are given scores based on their relative performance when used in the same network. Section 5.4.4.2 discusses standard deviation as a means to measure uncertainty in the overall assessment of a routing protocol.

5.4.4.1 Normalising Scores

In each network, CWT (weighting factor 2) and total data transfer were measured for each routing protocol. The results were then normalised so that the best performing routing protocol was given a normalised measurement of 1 and other routing protocols were given a normalised measurement equal to their performance relative to the best. For example, if the highest CWT was 1000 and the next best routing protocol achieved a CWT of 900 then the normalised CWT of the first routing protocol would be 1 and the second routing protocol would achieve a normalised score of 0.9. This normalisation solved two problems:

1. It allowed all networks to be treated as equally important.
2. It preventing a very high performance in one network from heavily influencing the results.

In the first case, it was necessary to treat all networks as equally important. Networks with small numbers of sources with achieve a higher CWT than those networks with many sources. By normalising the results, the scores themselves become irrelevant and only the relative performance of each routing protocol matters.

In the second case, a routing protocol that performs extraordinarily well in a tiny number of cases might (on average) appear to be the best if, e.g. the raw CWT or total data transfer scores were added up across all networks. However, such behaviour should not lead to the conclusion that the high performing routing protocol is the best overall. On the contrary, the routing protocol should be described as performing poorly on average, but particularly well in a number of special cases. Again, normalisation removes the influence of a single very high score in one network by scoring each routing protocol proportionally.

5.4.4.2 Standard Deviation

The need to repeat the experiment with different random factors has already been discussed. However, since it is impossible to examine each routing protocol in every single possible network with every possible random factor, the set of networks that are analysed only represent a *sample* of all possible networks that could be examined. Consequently, the sample mean is likely to differ from the true *universe* mean. Measurements shown in this thesis therefore have an associated confidence value indicating how likely it is that the universe value differs from the measured value.

From [6] it is possible to calculate the standard deviation of sample means, which is given by:

$$S_m = \sqrt{\frac{\sum((\bar{x} - x_i)^2)}{(N-1)(N)}} \tag{5.8}$$

Given a sample of measurements, $\bar{x}$ represents the mean of that sample, x_i represents measurement number i and N is the size of the sample. For normally distributed data, 68% of measurements fall within the range of one standard deviation and 95% of data is within the range of two standard deviations. Therefore, there is a 68% probability that the (true) universal mean lies within $m \pm S_m$ and a 95% probability that it lies within $m \pm 2S_m$.

5.5 Summary

This chapter discusses how to measure a routing protocol's ability to maintain a high source diversity for as long as possible. It justifies the creation of a new metric, known as CWT, which can measure the total data transferred from sources to sinks with a user adjustable bias towards whether the data comes from a variety of sources or not. The chapter presents different methodologies for examining routing protocols, including algebraic analysis, physical deployment and simulation and concludes that simulation (and particularly the Castalia 1.3 simulator) is the most suitable tool with which to carry out the analysis. Finally, the chapter discusses the best way to configure the simulator and how to analyse measurements (including CWT and total data transfer) in order to reach an analysis.

The information from this chapter will be used in Chapters 7, 9 and 10 which describe the experiments that were carried out to compare the new node reliance heuristics and protocols with third party contributions in their ability to maintain high source diversity over time.

Chapter 6

Node Reliance Heuristic

Several varieties of routing protocol are examined in Chapter 4 for solving the problem of maintaining high source diversity for as long as possible. Very few of the routing protocols avoid contention on nodes, i.e. paths selected by different sources may cause a subset of nodes to be overused, potentially partitioning the network [100] and lowering source diversity. Routing protocols that avoid contention, such as energy aware routing protocols, tend to be wasteful of network resources and thus reduce the maximum total data transferred from sources to sinks.

This chapter presents a new family of routing heuristics known as node reliance. Node reliance heuristics measure the degree to which each node is relied upon when routing messages from source nodes to sink nodes. Paths are formed from sources to sinks such that the heavily relied upon nodes are avoided where possible.

It is hypothesised that by using node reliance to route messages from sources to sinks, it is possible to achieve a high source diversity for longer than with other routing protocols.

As discussed in Chapter 3, a routing heuristic is made up of two tasks. The first of these is costing and deals with assigning costs to nodes/links and paths and will be explored in this chapter in Section 6.1 and Section 6.2 respectively. The second task is selection and involves choosing paths to route data from sources to sinks. The selection task contains modules whose use is optional. In order to explore the importance of selecting nodes based on costs, rather than other factors, the selection task is not considered here. More accurately, the selection task does not apply any topology or multipath routing.

6.1 Link/Node Costs

This section examines a number of models for assigning costs to links or nodes. The aim is to assign costs to nodes in proportion to how relied upon each node is in routing from sources to sinks. When routing data from a set of sources to a set of sinks

- A *bottleneck* is defined as a node through which all data must travel.
- An *unused node* is defined as a node through which no data travels.

Figure 6.1 shows examples of bottlenecks and unused nodes in a network where source A routes to sink Z.

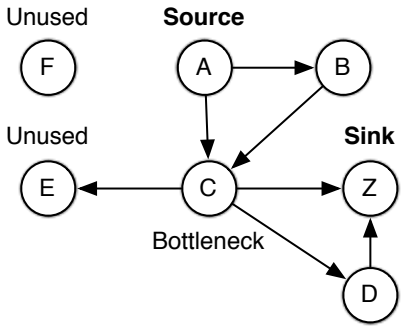


Figure 6.1: An example network showing bottlenecks and unused nodes

Bottlenecks and unused nodes are most commonly created in random node deployments in which the network topology may not be known ahead of time. However, such nodes may also be inadvertently created in fixed network topologies when nodes expire. Furthermore, as discussed in Section 2.5.2, the communication range of nodes is not uniform and is often subject to random environmental factors, which may vary over time. Therefore it may be very difficult, if not impossible, to prevent the creation of bottlenecks or unused nodes for the lifetime of a network.

The term *node reliance* (also referred to as reliance) is used to represent how relied upon a node is for routing and therefore the cost of that node. The two types of node illustrated above represent the extremes. A bottleneck node is essential to routing and hence will be given a very high reliance value, whereas an unused node has no relevance to routing and will be given a minimal reliance value. Between these two extremes, a *node reliance model* must be applied to determine the reliance value of a particular node.

Each source may rely on a single node to a different degree. For example, consider the network diagram shown in Figure 6.2. Node D acts as a bottleneck to source A and source B, whilst node E acts as a bottleneck to source B and is unused by source A. A node that acts as a bottleneck to multiple sources is more relied upon than a node that acts as a bottleneck to a single source and so should be given a higher reliance value. Although not relevant in this example network, it is also important to consider the reliance values of sources, since it may be beneficial to the entire network for one source to route through another. For example, it may be acceptable to route through one source if the alternative is to route through several sources.

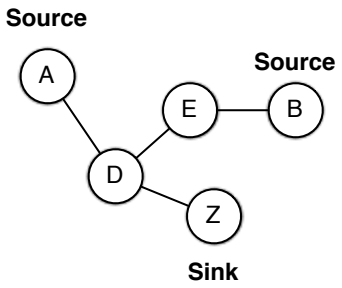


Figure 6.2: The loss of node D has more impact on the network than the loss of node E

Any model of node reliance must be able to distinguish between how relied upon a node is to a particular source and how relied upon a node is in the context of the entire network. To distinguish between these two concepts, the following definitions are made:

- The *relative reliance* of a node B to a (source, sink) pair (A, Z) is the degree to which node B is relied upon in routing data from A to Z.
- The *absolute reliance* of a node B is the degree to which node B is relied upon in the entire network.

The remainder of this section is structured as follows: Section 6.1.1 discusses models of node reliance that may be found in the literature and are shown to be unsuitable. Sections 6.1.2, 6.1.3 and 6.1.4 present three new models for expressing a node's reliance. Finally, Section 6.1.5 discusses how a realistic physical layer, in which messages may be lost and links may be unidirectional, may affect node reliance.

6.1.1 Unsuitable Models

Several network characteristics may reflect how much a node is relied upon in routing between sources and sinks:

- The *degree* of a node [27], which is the number of nodes that are directly connected to that node.
- The *clustering coefficient* [118] of a node, which is a means of expressing the local connectivity between the neighbours of a node.
- The effect on the set of shortest paths in the network of removing a node [43][30].
- The *pagerank* [86] of a node, which is an algorithm by Google to determine the importance of nodes (or web pages) in a graph.
- The *criticality* [64] of a link, which reflects how the bandwidth from sources to sinks changes due to the use of that link for routing.
- The number of *node-disjoint* paths that exist between a given (source, sink) pair.

Each of these models was rejected as a basis for calculating node reliance for reasons that are outlined in the following sections:

6.1.1.1 Node Degree

In graph theory, a node's *degree* refers to the number of edges incident to that node [27], regardless of the edge's directionality (if any). It could be argued that a node with a high degree is well connected and is more likely to be used in routing, thus making it highly relied upon. In practice, a node's degree only measures its immediate connectivity and does not reflect how well connected that node is between a particular (source, sink) pair. For example, in the network shown in Figure 6.3, node D has the highest degree of six, but is unused when routing from source A to sink Z. Node B has the second highest degree of four. Although node B is not an unused node, it is less relied upon than the bottleneck node H which only has a degree of two.

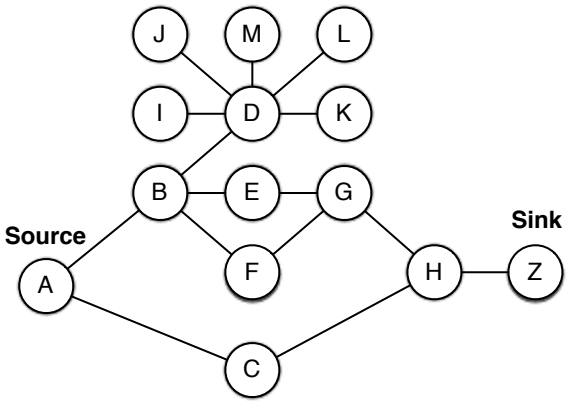


Figure 6.3: Node degree is not an indicator of how relied upon a node is in routing

The example network of Figure 6.3 demonstrates that a node's degree does not represent the degree to which that node is relied upon in routing between a given (source, sink) pair. Even if unused portions of the network (such as node D) can be removed, the node degree only measures the immediate connectivity of a node without reference to any source or sink nodes.

6.1.1.2 Clustering Coefficient

The clustering coefficient [118] C is defined as follows: "Suppose that a vertex v has k_v neighbours; then at most $k_v(k_v - 1)/2$ edges can exist between them [...]. Let C_v denote the fraction of these allowable edges that actually exist. Define C as the average C_v over all v". Informally, the clustering coefficient of a node reflects how well the neighbours of that node are connected to one another. The clustering coefficient of the network is the average clustering coefficient of each node.

For example, in Figure 6.4a, node A has four neighbours, but none of those nodes are connected to each other, giving a clustering coefficient of 0/6 = 0. Nodes B, C, D and E have one neighbour and so have a clustering coefficient of 0/0 = 0. Thus, the clustering coefficient of the network is 0. In Figure 6.4b, every node has four neighbours and all four neighbours are connected to each other. Thus, each node has a clustering coefficient of 6/6 = 1 and so the network clustering coefficient is 1.

In the definition, it is assumed that edges are bidirectional. If links are unidirectional,

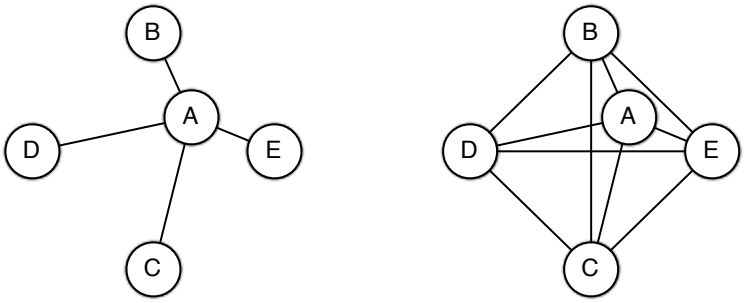


Figure 6.4: Two example networks with minimum and maximum clustering coefficient

as was assumed in Section 2.5.2.2, then the maximum number of edges that are allowed between the k neighbours of a node increases to k(k-1). The clustering coefficient of a node remains as the fraction of those allowable edges that exist.

The reason for considering the clustering coefficient as a measurement of node importance is that a node with high connectivity coefficient is well connected within the network and therefore may be more relied upon in keeping nodes connected to each other. However, as with node degree, a particular node may have a high connectivity coefficient and be unused for routing between a particular (source, sink) pair. Even if unused portions of the network can be disregarded, connectivity coefficient is still not indicative of how relied upon a particular node is. For example, consider the network shown in Figure 6.5. In this network, a bottleneck and a non-bottleneck (nodes H and C respectively) both have a clustering coefficient of 0 since none of their neighbours are connected to each other. Thus, clustering coefficient is not indicative of how much a node is relied upon in routing.

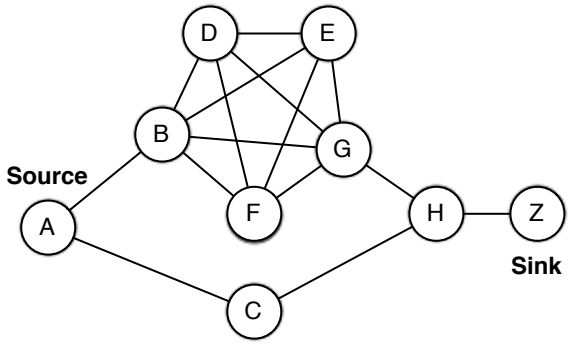


Figure 6.5: The clustering coefficient is unusable

6.1.1.3 Shortest Paths

The literature includes two approaches to calculating the importance of a node based on shortest paths:

Hawick [43] calculates the importance of a node in a peer-to-peer network based on how much the *Djikstra all-pairs distance* (DAPD) changes when that node is removed from the network. The DAPD is the average shortest path length between every pair of nodes in the network. Thus, a node is important if its loss would cause the shortest path between node pairs to increase. Hawick's approach is not conducive to a (source, sink) architecture, since it assumes that routing may take place between any pair of nodes. However, even if the calculation instead only considers the DAPD between (source, sink) pairs, the metric still does not accurately reflect a node's significance in many cases. For example, in Figure 6.6, the sum of shortest paths between every (source, sink) pair is 8, giving an average DAPD of 2. This value is unchanged if any of nodes F-I are removed and consequently the importance of each node would be 0. However, nodes F and G are clearly more relied upon by H and I, which can only be used by source D. Consequently, nodes F and G should be considered more important than H and I.

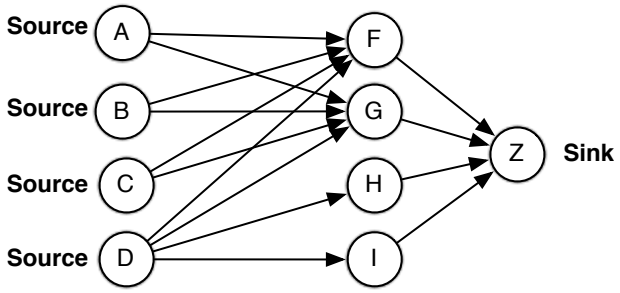


Figure 6.6: Removing any intermediate node does not cause the shortest path length to increase between any source and sink

Dimokas [30] suggests a model in which a node's importance is determined by the proportion of shortest paths that the node lies on between each pair of nodes within a cluster. Again, this measurement can be trivially changed to consider only the proportion of shortest paths between sources and sinks. However, it does not accurately reflect the importance of nodes in many cases. For example, consider Figure 6.7. The shortest path from A is AFZ, from B is BFZ and from C is CEZ. With the exception of the sink, node F is the most im-

portant with a value of 2/3. In order to avoid using node F, source A would therefore route along the only remaining path, ADEZ. Since node D has no importance and node E is of lower importance than F, the path is better than AFZ. However, the path contains node E, which is relied upon by source C. Consequently, the overuse of node E will cause source C to become quickly disconnected, lowering source diversity.

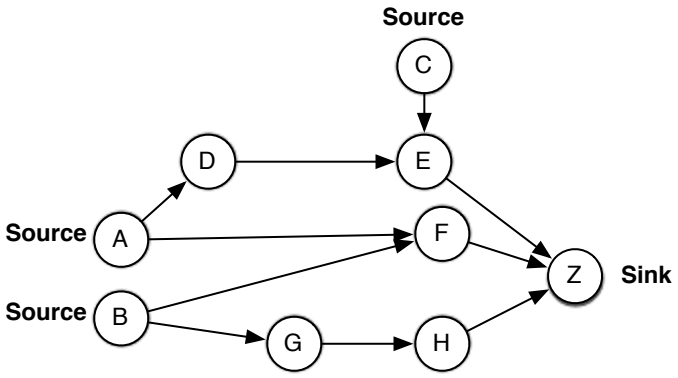


Figure 6.7: Being on the minimum hop path is not indicative of a node's importance in routing

In this example, the optimal paths to take would be AFZ, BGHZ and CEZ. None of these paths have any nodes in common besides the sink, and so would be expected to keep sources and sinks connected for the longest period of time.

None of these approaches justifies the significance of shortest paths for node importance, unless the intention is to always route along the shortest path. However, as discussed in Section 3.2.2.1, shortest path routing may cause a subset of important nodes to quickly expire. Furthermore, it is not obvious what the link/node costs should be. If the intent is to use shortest path routing with a unit edge weight, then the approaches may suffer from additional problems associated with minimum hop routing, as discussed in Section 4.2.

6.1.1.4 PageRank

PageRank [86] is a ranking system used to determine the importance of web pages in the Google [13] search engine. In PageRank, the web is modelled as a graph, with nodes representing web pages and a directed edge AB representing a hyperlink from page A to page B. The PageRank algorithm then iteratively calculates how important each node is. In

the context of Google, the importance of a node represents the probability of a random web surfer landing on a particular page.

It is tempting to consider the use of PageRank in a WSN context. However, the random movement through the graph does not accurately represent the directed movement of data that takes place in a (source, sink) architecture. PageRank may therefore be a suitable measurement of node importance in a WSN where any pair of nodes can communicate, but is not considered to be relevant for the purposes of this thesis.

6.1.1.5 Link Criticality

Rather than consider the importance of a node, one might instead consider the importance of the links that a node provides. A link may be considered *critical* if its use causes the maximum achievable bandwidth between any (source, sink) pair to drop. The *criticality* of a link may be defined as the number of (source, sink) pairs for which that link is critical.

Link criticality is used in the Minimum Interference Routing Algorithm [64] (MIRA), which is used to form bandwidth guaranteed tunnels between (source, sink) pairs in wired networks. A tunnel, i.e. a path, can only be formed if the links that make up that path have sufficient unreserved bandwidth remaining. The aim of MIRA is to refuse as few tunnel requests as possible.

One requirement of MIRA is that the current bandwidth used by each link must be known in order to estimate how the maximum achievable bandwidth of the entire network will be affected by sending additional data along a (source, sink) path. An alternative approach involves pre-computing the criticality of nodes by considering their *criticality threshold*, i.e. the degree to which a link's capacity must drop in order that the link becomes critical. In this way, it is unnecessary to know the bandwidth used by a link. If the criticality threshold of the link is low, it should be avoided since it will become critical more quickly that those with a higher criticality threshold.

There appears to have been no attempt to use minimum interference routing in WSNs. Indeed, the term *minimum interference routing* in WSNs tends to concern itself with routing protocols that minimise radio interference between nodes. A possible explanation for this lack of use in WSNs may be that bandwidth guarantees are of a lesser concern than the limited energy of nodes. It is not obvious how MIRA might be modified for use in a WSN.

Considering the maximum capacity of the network, rather than maximum bandwidth, is insufficient because criticality would lose any meaning. When routing between a source and sink stops, the bandwidth used by the intermediate nodes is released. However, the capacity of nodes in a WSN is not restored when an intermediate node is no longer used for (source, sink) routing. Furthermore, unless two sources have disjoint paths to the sink, there will always exist a path whose use will cause the maximum (source, sink) capacity to drop. Thus, it may be very rare that non-critical links could be used.

6.1.1.6 Node-Disjoint Paths

The final rejected model is the number of *node-disjoint* paths that can be taken from a source to a sink. Two paths are said to be *node-disjoint* if they do not have a single node in a common. For the purposes of a node-disjoint model, sources and sinks are not considered and may appear on multiple paths. Thus, the value of a node may be expressed as the inverse of the number of node-disjoint paths between a given (source, sink) pair. If there are many node-disjoint paths from a source to a sink, then no path (and so no node on those paths) is heavily relied upon. For example, if there are four node-disjoint paths between a source and a sink, each node on those paths may be given a reliance value of 0.25.

However, it is not obvious how the set of node-disjoint paths should be calculated. For example, in the network shown in Figure 6.8, there can only be a single node disjoint path from source A to sink Z, since every path must include the node F. However, it is not clear whether the node-disjoint path is ABEFZ, ABDFZ or ACFZ. All paths are equally valid.

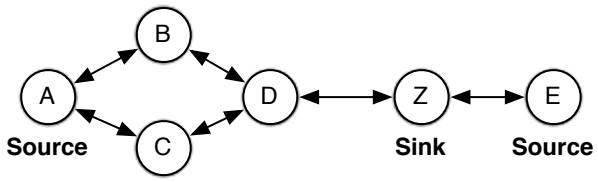


Figure 6.8: It is not clear which path should be selected

It may be tempting to consider the set of shortest node-disjoint paths. However, such a model would suffer from the problems outlined in Section 3.2.2.1 with respect to shortest path routing.

6.1.1.7 Conclusion

None of the approaches discussed here are suitable for representing node importance in routing from sources to sinks. Having rejected these approaches, this thesis now presents a family of models that may be used to represent node reliance. These approaches are the *simple paths model*, the *simplest paths model* and the *contraction model* and are studied in the following sections.

6.1.2 Simple Paths Model

The simple paths model is the first of three models that are proposed in this thesis for calculating node reliance values. A *simple* path is defined as a sequence of nodes in which no node appears more than once. For example, the path ABCD is simple, but the path ABCB is not simple because the node B is repeated. The simple paths model considers the set of simple paths between a (source, sink) pair and assigns node reliance values based on the proportion of paths containing each node.

Paths containing more than one sink are not considered since the aim of a (source, sink) architecture is to route the data from a source to any sink. There is no value in using a sink as an intermediate node since the use of additional nodes on the path can only increase energy expenditure in the network.

The intuition of the simple paths model is that data may be sent along any simple path from a source to a sink. There is no point in routing along a non-simple path, since it involves sending data through extraneous nodes, consuming the batteries of nodes for no benefit. If a node appears on many simple paths, then its loss will greatly reduce the number of options for routing and so is heavily relied upon. Conversely if a node appears on very few paths, then it is not greatly relied upon, since many paths will remain available if it expires. For example, in Figure 6.9, there are two simple paths from source A to sink Z: ABDZ and ACDZ. Any other (source, sink) path is non-simple. The loss of node D removes all paths from A to Z. Consequently it is highly relied upon and so should have a high reliance value. Conversely, nodes B and C each appear on half the paths from A to Z. The loss of either one causes half of the (A, Z) paths to become invalid. Thus, their reliance value is 0.5. Source E does not appear on any paths between A and Z and so its loss is inconsequential to (A, Z) routing.

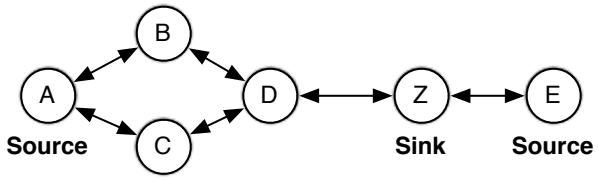


Figure 6.9: An example network with two sources, A and D

The *relative reliance* of a node B to (source, sink) pair (A, Z) is equal to the proportion of simple paths between A and Z that contain node B. A node that lies on all simple paths between A and Z is a bottleneck and has the maximum relative reliance of 1.0. A node that does not lie on any simple paths between A and Z is an unused node, and therefore has the minimum relative reliance of 0.0. Between the extremes, a node B's reliance value at A reflects the proportion of paths from A to Z that require node B.

The *absolute reliance* value for a node is the average relative reliance of that node across all (source, sink) pairs for which at least one simple path exists. Thus, a node that is a bottleneck to all sources will have an absolute reliance value of 1.0. A node that acts as a bottleneck to half the sources and is unused by the other half will have an absolute reliance value of 0.5. By calculating the absolute node reliance value based on the average relative value of a node across all (source, sink) pairs, the nodes with highest absolute reliance will be those nodes that act as a bottleneck to the most sources in the network.

6.1.2.1 Example

Consider the network shown in Figure 6.10 in which sources A and D route data towards sink Z.

Performing a depth first search on the graph, beginning at the source nodes and terminating whenever a sink node is reached, can determine the set of simple paths. The formation of non-simple paths is prevented by limiting the depth first search to nodes that have not already been visited based on the search's current state, i.e. the partially formed path. The set of simple paths from source nodes A and D to the sink node Z in the network in Figure 6.10 is shown in tree form in Figure 6.11 and enumerated in Table 6.1.

The relative reliance values of each node can be calculated by examining the number of

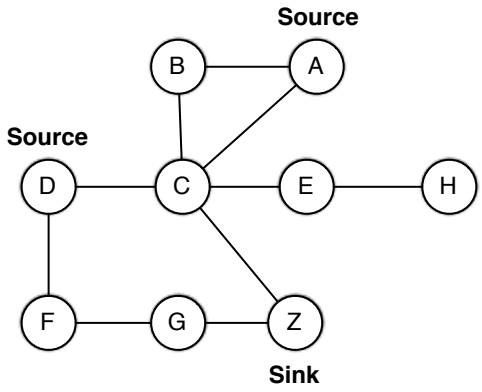


Figure 6.10: An example network with two sources, A and D

Simple Paths Originating from A	Simple Paths Originating from D
ABCDFGZ	DCZ
ABCZ	DFGZ
ACDFGZ	
ACZ	

Table 6.1: Simple paths from sources A and D to the sink Z from the network of Figure 6.10

simple paths on which each node lies.

In the example network of Figure 6.10 there are four simple paths from A to Z. Nodes A, C and Z appear on all of these paths, i.e. they are bottlenecks and so have a relative reliance value to (A, Z) of 1.0. Nodes E and H do not appear on any of the paths, i.e. they are unused and so have a relative reliance value to (A, Z) of 0.0. Node B lies on half the (A, Z) paths (ABCDFGZ and ABCZ) and so its relative reliance is 0.50. Node D lies on half the paths (ABCDFGZ and ACDFGZ) and so has a relative reliance of 0.50. Node F lies on half the (A, Z) paths (ABCDFGZ and ACDFGZ) and so has a relative reliance of 0.50. Finally node G lies on half the paths (ABCDFGZ and ACDFGZ) and also has a relative reliance to (A, Z) of 0.50.

There are also two simple (D, Z) paths. Nodes D and Z lie on both of these paths and so have a relative reliance value to (D, Z) of 1.0. Nodes A, B, E and H are absent from all the paths, i.e. they are unused in routing from D to Z and so are given a relative reliance of 0.00 to (D, Z). Node C lies on half the paths (DCZ) and nodes F and G also lie on half the

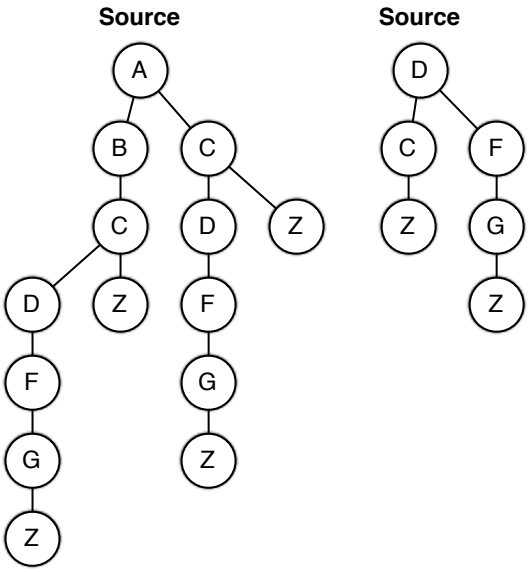


Figure 6.11: A tree showing simple paths from sources A and D to sink Z

paths (DFGZ in both cases) and so are also given a relative reliance value of 0.50 to (D, Z).

The absolute reliance values are determined by calculating the average relative reliance value across each source. For example, source A has a relative reliance of 1.0 to (A, Z) and a relative reliance of 0.0 to (D, Z). The absolute reliance is the average of these, i.e. 0.50. Similarly, node D has a relative reliance of 0.50 to (A, Z) and a relative reliance of 1.00 to (D, Z). The absolute reliance of node D is therefore equal to 0.75, i.e. the average of 0.50 and 1.00. Table 6.2 shows the relative and absolute values for all nodes in the network.

Visual inspection reveals the absolute and relative reliance values follow intuition. The most valuable node is sink Z whose loss would cause the entire network to fail (if the network had multiple sinks, then this value would be lower). The next highest reliance value nodes are C and D. Node C is a bottleneck to source A and therefore if C were to expire, A would be disconnected from the network. Furthermore, the loss of node C causes half D's (source, sink) paths to become unusable. Thus, node C is given a high absolute value. Node D acts as a source and obviously its loss results in the disconnection of a source from the network. Additionally, the loss of node D causes half of the (A, Z) paths to become unusable. Thus, node D has a high absolute value. Nodes A, F and G have the next highest reliance values of 0.50. Node A acts as a source whose loss would result in a source being disconnected from the network, which is undesirable. Sources F and G each lie on

Node	Relative Reliance to (A, Z)	Relative Reliance to (D, Z)	Absolute Reliance
A	1.00	0.00	0.50
B	0.50	0.00	0.25
C	1.00	0.50	0.75
D	0.50	1.00	0.75
E	0.00	0.00	0.00
F	0.50	0.50	0.50
G	0.50	0.50	0.50
H	0.00	0.00	0.00
Z	1.00	1.00	1.00

Table 6.2: Reliance values calculated from the simple paths shown in Table 6.1

half the paths between (A, Z) and between (D, Z). Thus, the loss of either node causes half of the paths in the network to become unusable. Of the remaining nodes, node B has a low reliance value because it is unused by source D and only appears on half of the paths from source A. Finally, nodes E and H are unused. There is no simple path from any source to any sink that uses either of these nodes. Thus, their reliance value is 0 and their loss has no impact on (source, sink) connectivity.

6.1.2.2 Worst-Case Growth Rate

Despite an extensive literature search, no method for finding the proportion of simple (source, sink) paths that a node lies on has been found except by enumerating all the paths and examining them as was shown in the earlier example. It is therefore desirable to analyse the worst-case scenario of how the number of simple paths grows with respect to the number of nodes in the network in order to determine under what circumstances such an enumeration is tractable.

The number of simple paths in a network depends on the topology of the network. Given a partially formed (source, sink) path from a source A to an intermediate node N, the number of simple paths is affected by the nodes already visited and the links from node N. For example, consider the network shown in Figure 6.12, which was used earlier in the example of the simple paths model.

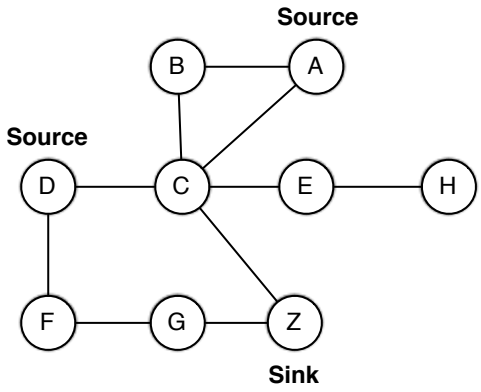


Figure 6.12: An example network with two sources, A and D

Figure 6.13 shows a tree of simple (source, sink) paths that originate from source A and travel through node C. The topology affects which nodes are connected to node C, i.e. nodes A, B, D, E and Z. However, not all of these nodes can be used. For example, if the path taken from the source to node C is AC then nodes B, D, E and Z are all candidates for the next hop. However, if the path taken is ABC then B is not a candidate because it has already been visited and paths must be simple. Note that although node E is a candidate for the next hop in the tree, it is impossible to form a simple path from A to Z that travels through node E.

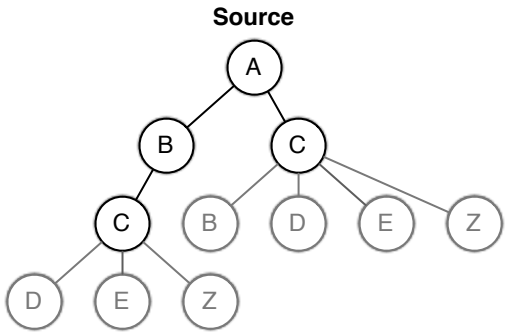


Figure 6.13: A tree showing simple (source, sink) paths from source A

The number of leaves on the tree, and therefore the number of simple paths in the network, is increased by increasing the height of the tree and the number of children of each node. The number of children of each node is improved by improving the connectivity of each node. Thus, the worst-case growth rate of the number of simple paths in terms of the number of nodes in the network is achieved by having all nodes at maximum connectivity.

A *fully connected* network is defined as a network in which every node is connected to every other node and results in a maximisation of the number of (source, sink) simple paths. By considering fully connected networks, it is possible to analyse the growth of the number of simple paths in the network with respect to the number of nodes and thus determine the point at which the calculation of node reliance may become intractable in a worst-case scenario.

A fully connected network of n nodes $(n \geq 2)$ containing one source (A) and one sink (Z) such as that shown in Figure 6.14 includes all of the following:

- One simple path containing 2 nodes, i.e. the path AZ.
- $(n-2)$ simple paths containing 3 nodes, i.e. A, followed by any of the $(n-2)$ intermediate nodes, followed by Z.
- $(n-2)(n-3)$ simple paths containing 4 nodes, i.e. A, followed by any of the $(n-2)$ intermediate nodes, then any of the $(n-3)$ remaining intermediate nodes and finally Z.
- $(n-2)(n-3)(n-4)$ simple paths containing 5 nodes, etc.

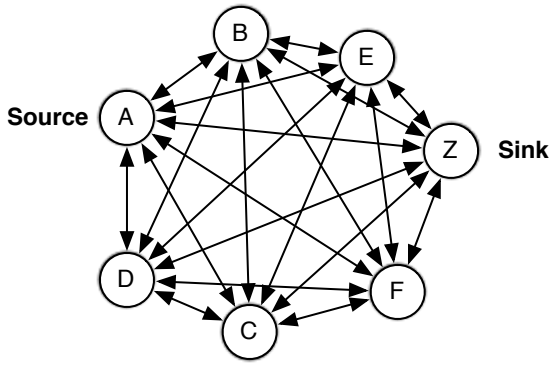


Figure 6.14: A fully connected network of 7 nodes, including 1 source and 1 sink

In each case, a simple path containing i nodes $(i \geq 2)$ consists of the source A, followed by $(i-2)$ intermediate nodes, finally followed by the sink node Z. There are $(n-2)$ possibilities for the first intermediate node, i.e. every node except the source or sink, and once chosen a node cannot be reselected. Mathematically, this is known as a permutation.

If $(i-2)$ choices must be made from a population of $(n-2)$ entities, then there are said to be ${}^{(n-2)}P_{(i-2)}$ possible permutations, i.e. numbers of simple paths.

$$ {}^{(n-2)}P_{(i-2)} = \frac{(n-2)!}{((n-2)-(i-2))!} = \frac{(n-2)!}{(n-i)!} \tag{6.1} $$

The variable i is dependent on the number of nodes on the path, which may be between 2 (in the case AZ) to n in which every node lies on the simple path. Therefore, the total number of simple paths is:

$$ \sum_{i=2}^{n} \frac{(n-2)!}{(n-i)!} = (n-2)! \sum_{i=2}^{n} \frac{1}{(n-i)!} = (n-2)! \sum_{i=0}^{n-2} \frac{1}{i!} \tag{6.2} $$

It is known that the mathematical constant e can be represented by the infinite series:

$$ \sum_{i=0}^{\infty} \frac{1}{i!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} ... \tag{6.3} $$

Therefore, Equation 6.2 approximates $e(n-2)!$ where $(n \geq 2)$.

Since the number of simple paths grows at a factorial rate, it only requires a small number of nodes to be fully connected in some part of the network for the enumeration of simple paths to become intractable. For example, a network of 12 fully connected nodes creates 9864101 simple paths, each of which must be analysed and the occurrences of each node counted. Given that the processing power of nodes is severely limited (8 MHz on the TMote sky [101]), a full enumeration of all paths is unlikely to be practical. Furthermore, each successive node added to the network dramatically increases the number of simple paths. For example, a network of 13 fully connected nodes has 108505112 simple paths.

6.1.3 Simplest Paths Model

The *simplest paths model* is a modification of the simple paths model in which node reliance values are calculated based on the proportion of *simplest* paths containing a particular node rather than the number of simple paths containing that node.

A path P from A to Z is defined as being simplest if it is not possible to form another (A, Z) path by removing nodes from P. For example, in Figure 6.15 the path ACGZ is simplest, but the path ACDFGZ is not simplest because the removal of nodes D and F results in another path from A to Z (ACGZ).

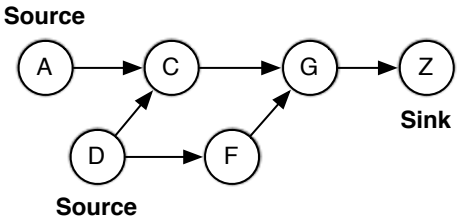


Figure 6.15: An example network showing simplest paths

A *subpath* of a path P is said to be any path Q with the same source and sink as P that can be created by removing nodes from P. For example, in Figure 6.15 the path ACGZ is a subpath of ACDFGZ because ACGZ can be formed by removing nodes D and F from ACDFGZ.

Finally a *superpath* of a path P is said to be any path Q with the same source and sink as P that is formed by adding nodes to P. For example, in Figure 6.15 the path ACDFGZ is a superpath of ACGZ since ACDFGZ is formed by adding nodes D and F to ACGZ.

As with the simple paths model, the intuition with the simplest paths model is that if a node lies on multiple paths, it is more likely to be used in routing from sources to sinks. By considering the set of simplest paths rather than simple paths, only those paths that are likely to be used in routing can contribute a node's value. It is unlikely that a non-simplest path would ever be used, since the path contains nodes whose presence is not essential. Therefore, these paths should not contribute towards a node's value.

As in the simple paths model, it is assumed that a path will never contain more than one sink. Consequently, the set of paths under consideration is the set of single-sink, simplest paths. As in the simple paths model, it is assumed that no path may contain more than one sink.

6.1.3.1 Example

The network shown in Figure 6.16 is used to demonstrate the simplest paths model.

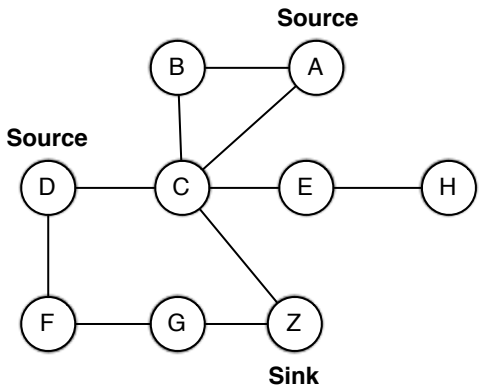


Figure 6.16: An example network with two sources, A and D

The network consists of two sources (A and D) and one sink (Z). Having already calculated the simple paths of the network in Section 6.1.2.1, it is possible to simply eliminate those paths that are not simplest. Table 6.3 shows the set of simple paths of the network. For each non-simplest path, the simplest subpath is also shown.

Simple Path	Simplest Subpath
ABCDFGZ	ACZ
ABCZ	ACZ
ACDFGZ	ACZ
ACZ	
DCZ	
DFGZ	

Table 6.3: Simple and simplest paths from the network of Figure 6.16

The relative reliance values of each node can be calculated by examining the number of simplest paths that each node lies on. The pair (A, Z) has one simplest path (ACZ). Consequently, all three nodes, A, C and Z have a relative reliance of 1.0 to the pair (A, Z). Other nodes that do not appear on this path, i.e. B, D, E, F and G all have a relative reliance of 0.0 to (A, Z). The pair (D, Z) has two simplest paths (DCZ and DFGZ). Since nodes C, F and G each appear on one of these two paths, they have a relative reliance of 0.5 to the pair

(D, Z). Nodes D and Z appear on both paths and so have a relative reliance of 1.0 to (D, Z). Nodes A, B and E do not appear on either path and so have a relative reliance of 0.0 to (D, Z). A node's absolute reliance is equal to the mean of its relative reliances. For example, node D has a relative reliance of 0 to (A, Z) and a relative reliance of 1 to (D, Z), giving it an absolute reliance of 0.5. The relative and absolute reliance values for each node are shown in Table 6.4.

Node	Relative Reliance to (A, Z)	Relative Reliance to (D, Z)	Absolute Reliance
A	1.00	0.00	0.50
B	0.00	0.00	0.00
C	1.00	0.50	0.75
D	0.00	1.00	0.50
E	0.00	0.00	0.00
F	0.00	0.50	0.25
G	0.00	0.50	0.25
Z	1.00	1.00	1.00

Table 6.4: Reliance values calculated from the simple paths shown in Table 6.3

The assignment of absolute and relative reliance values follows intuition. Sink Z is the most valuable node, since it is the only sink in the network and its loss would cause the entire network to fail. The next highest reliance value node is C. Node C is a bottleneck to source A and therefore if C were to expire, A would be disconnected from the network. Furthermore, the loss of node C causes half of D's (source, sink) paths to become unusable. Thus, node C is given a high absolute value. The loss of a source node prevents (source, sink) routing from that node and consequently the source nodes have the next highest reliance value. Of the remaining nodes, F and G have a low reliance because they are unused by source A and appear on only half the simplest paths of source D. Finally, nodes B, E and H are all unused. There is no simplest path from any source to any sink that uses any of these nodes. Thus, their reliance value is 0 and their loss has no impact on (source, sink) connectivity.

6.1.3.2 Worst-Case Growth Rate

As is the case with the simple paths model, no means of determining node reliance values based on the number of simplest paths has been found, except to enumerate each path and count the number of paths on which each node lies. Therefore, to determine whether enumerating all simplest paths is tractable or not, it is necessary to determine the worst-case growth rate for the number of simplest paths in the network with respect to the number of nodes.

As discussed in Section 6.1.2.2, for any given number of nodes, the number of simple paths is maximised when the network is fully connected. However, in a fully connected network, the source (A) and sink (Z) are directly connected and so only one simplest path exists. Any other path from the same (source, sink) pair must be a superpath.

It is therefore necessary to consider under what situations the number of simplest paths in a network is maximised for a given number of nodes. Figure 6.17a shows a tree of simplest paths for the network shown in Figure 6.17b.

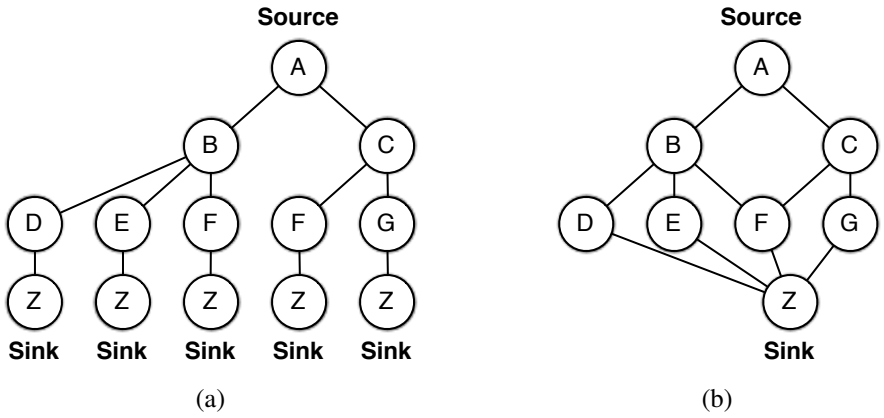


Figure 6.17: A tree of simplest paths that is formed from an example network

Each of the five leaf nodes in the routing tree of Figure 6.17a represents a different simplest path from A to Z in the network of Figure 6.17b. The non-leaf nodes in the tree represent intermediate nodes in the network. Thus, to determine the worst-case growth rate for the number of simplest paths in a network, we wish to determine how to maximise the number of leaves for a fixed number of intermediate nodes in the corresponding tree.

Note that some intermediate nodes appear more than once in the tree of Figure 6.17a. For

example, node F is treated as a child of both node B and C. It is advantageous to include a node more than once in the tree because although it acts as several intermediate nodes in the tree, it is only a single intermediate node in the network. The number of leaves (and therefore the number of simplest paths) depends on the height of the tree and and the number of children of each node. Thus, by including nodes more than once in the tree, the same number of intermediate nodes produces more simplest paths. The optimal solution occurs when all nodes of a particular depth have the same children. The *depth* of a node is defined as the number of hops to reach that node from the root or source of the tree. This configuration is optimal because it maximises the number of intermediate nodes in the tree of simplest paths while minimising the number of intermediate nodes in the network.

For example, Figure 6.18a shows the tree of simplest paths that is formed from the network of Figure 6.18b in which all nodes of a particular depth have the same children.

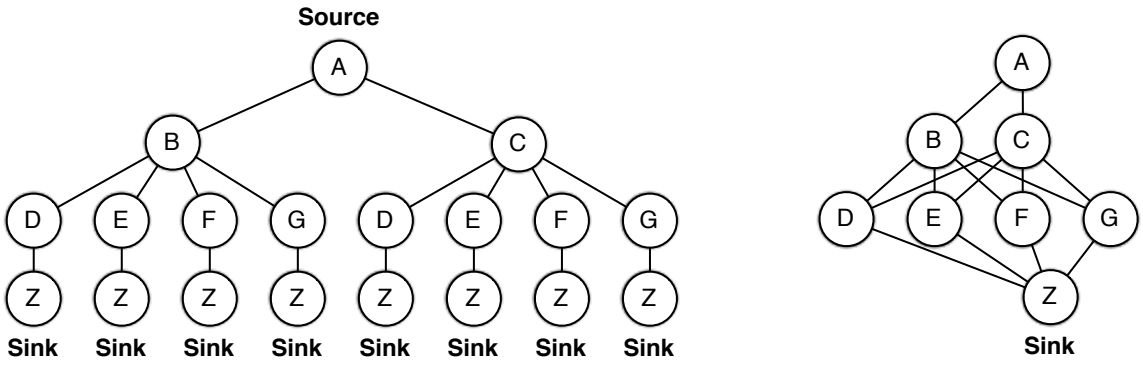


Figure 6.18: A tree of simplest paths that is formed from an example network

An important restriction is that if all nodes of a particular depth have the same children, then all parents of a node must be at the same depth too. If this restriction is not met, then it results in subpaths, and so the number of simplest paths in the network will drop. For example, Figure 6.19 shows a network in which all nodes of a certain depth have the same children. However, node G has parents of depth 0 (A) and depth 1 (B and C). Consequently, paths ACGZ and ABGZ are no longer simplest paths because of the subpath AGZ. Thus, the number of simplest paths is reduced.

Thus, in order to maximise the number of simplest paths, all nodes of depth n should have the same children, which should have exactly the same parents. However, given a network with a single source and single sink and m intermediate nodes, it is unclear how many

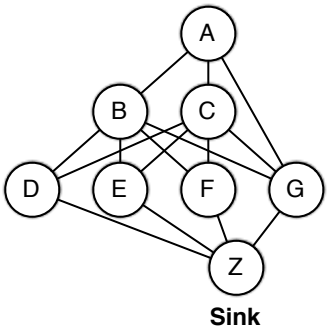


Figure 6.19: Adding link AG causes simplest paths to become non-simplest

children each node should have, and what the maximum depth of the tree should be in order to maximise the number of simplest paths. These questions are answered algebraically below.

The tree of simplest paths provides a simplest (source, sink) path for each leaf in the network. The number of simplest paths, s, is equal to the number of leaves. In a tree, the number of leaves is equal to c^d where d represents the maximum depth of the tree and c represents the number of children for each parent. The maximum depth of the tree and the number of children available at each node are limited by the number of intermediate nodes m and $m = cd(m \geq 0)$. Combining these two equations gives:

$$s = c^{\frac{m}{c}} \tag{6.4}$$

By calculating the derivative of s with respect to c, it is possible to determine the values of c for which the growth of s is at an extreme. Once the extremes are found, the second derivative indicates whether those extremes are maximum or minimum growth rates and it is possible to find the value of c for which s grows the fastest. Before deriving the equation, the natural logarithm of each side is taken, thus making the derivative easier to calculate.

$$ln(s) = \frac{m}{c} ln(c) = mc^{-1} ln(c) \tag{6.5}$$

Since s is a function of c, we can use implicit differentiation to calculate the derivative.

$$\frac{d}{dc}(ln(s)) = \frac{d}{dc}(mc^{-1}ln(c)) \tag{6.6}$$

Differentiating the right side of the equation involves the chain rule, and yields the equation:

$$\frac{1}{s}\frac{ds}{dc} = \frac{mc^{-1}}{c} - mc^{-2}ln(c) = mc^{-2} - mc^{-2}ln(c) \tag{6.7}$$

$$\frac{ds}{dc} = smc^{-2}(1 - ln(c)) \tag{6.8}$$

Finally, by substituting the value of s from equation 6.4 and simplifying, the following equation can be formed:

$$\frac{ds}{dc} = mc^{\frac{m}{c}-2}(1 - ln(c)) \tag{6.9}$$

The maximum and minimum growths of a function occur where the derivative of that function is equal to 0.

$$mc^{\frac{m}{c}-2}(1 - ln(c)) = 0 \tag{6.10}$$

The above equation can be solved by determining the values of c for which $mc^{\frac{n}{c}-2} = 0$ or $(1 - ln(c)) = 0$. In the former case, there is no solution since there is no value of y for which $x^y = 0$. In the latter case $1 - ln(c) = 0$ where $c = e$. Therefore, only one extreme for the function exists, where $c = e$.

The second derivative can be calculated to determine whether the extreme is a maximum or minimum when $c = e$. The extreme is a maximum if the second derivative of s is less than zero. Conversely, the extreme is a minimum if the second derivative is greater than zero. The second derivative is calculated by using implicit differentiation on Equation 6.7 to give:

$$\frac{1}{s}\frac{d^2s}{dc^2} - s^{-2} = -2mc^{-3} + (2mc^{-3}ln(c) - mc^{-3}) \tag{6.11}$$

This equation is combined with $c = e$:

$$\frac{1}{s}\frac{d^2s}{dc^2} - s^{-2} = -2me^{-3} + (2me^{-3} - me^{-3}) = -me^{-3} \tag{6.12}$$

$$\frac{d^2s}{dc^2} = -sme^{-3} + s^{-1} \tag{6.13}$$

$$\frac{d^2s}{dc^2} = -e^{\frac{m}{e}}me^{-3} + e^{\frac{-m}{e}} = e^{-e}(e^{-m} - e^{m-3}) \tag{6.14}$$

The second derivative indicates that the extreme is a maximum or minimum depending on the value of m. To determine the point at which the number of simplest paths grows the fastest, it is necessary to find the point at which the extreme is a maximum, i.e. when $\frac{d^2s}{dc^2} < 0$

$$e^{-e}(e^{-m} - e^{m-3}) < 0 \tag{6.15}$$

$$e^{-m} < e^{m-3} \tag{6.16}$$

$$-m < m - 3 \tag{6.17}$$

Therefore, when c = e (2.72), the growth of the number of simplest paths is at a maximum provided that $m > 1.5$. Obviously it is not possible to have 1.5 intermediate nodes, so the number of simplest paths is at a maximum provided that $m \geq 2$ and there are e (2.72) children at each node. It is not possible to have 2.72 children at each node, since the number of children must be an integer. Therefore it is necessary to determine whether the growth of s, i.e. the number of simplest paths, is greatest when $c = 2$ or $c = 3$. Determining which growth rate is faster can be done by inserting these two different values into Equation 6.4:

$$s_1 = 3^{\frac{m}{3}} \tag{6.18}$$

$$s_2 = 2^{\frac{m}{2}} \tag{6.19}$$

Equations 6.18 and 6.19 are raised to the power of $\frac{6}{m}$ to produce:

$$s_1^{\frac{6}{m}} = (3^{\frac{m}{3}})^{\frac{6}{m}} = 3^2 = 9 \tag{6.20}$$

$$s_2^{\frac{6}{m}} = (2^{\frac{m}{2}})^{\frac{6}{m}} = 2^3 = 8 \tag{6.21}$$

Comparing these equations, we can see that $s_1^{\frac{6}{m}}$ is greater than $s_2^{\frac{6}{m}}$. Therefore, having three children at each node maximises the number of simplest paths such that $s = 3^{\frac{m}{3}}$. Previously it had been stated that $m \geq 2$. However, for there to be 3 children at each node, it is necessary to have at least 3 intermediate nodes. Thus, $m \geq 3$.

Finally, if n represents the number of nodes in the network containing m intermediate nodes, one source and one sink then $s = 3^{\frac{n-2}{3}}$.

Figure 6.20 shows an example network in which a total of 14 nodes are placed in such a manner as to maximise the number of simplest paths. Each node (except the sink) has three children of equal depth, each of which has the same parents. The network contains 3^4 = 81 simplest paths.

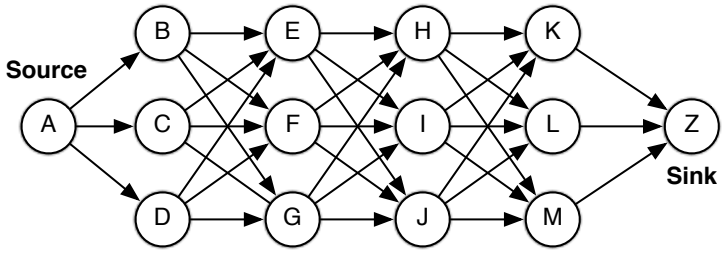


Figure 6.20: The worst-case scenario layout for generating simplest paths

A worst-case growth rate in the number of paths of $O(3^{\frac{n-2}{3}})$ may be considered tractable for networks with fewer than 50 nodes. This restriction includes the majority of deployments studied in Section 2.4.

6.1.3.3 Advantages

The simplest paths model offers two advantages over the simple paths model. Firstly, there are a smaller number of simplest paths than simple paths and therefore, the simplest paths model may be more tractable in larger networks. Secondly, the model only assigns value to nodes based on paths that may be used for routing rather than all possible paths. These issues are discussed in further detail, below:

Comparing the worst-case growth rate of each of the models, as derived in Sections 6.1.2.2 and 6.1.3.2, it can be seen that the growth of the number of paths with respect to the number of nodes is $O(n!)$ for the simple paths model and $O(3^{nk})$ for the simplest paths model where n is the number of nodes in the network and k is a constant. Since it is necessary to enumerate all paths in order to determine node reliance values in the network, the simplest paths model may prove to be more tractable in larger networks.

In the simple paths model, a large number of paths may contribute to the reliance value of a node. However, many of those paths may be non-simplest, thus containing nodes that can be removed, and so would never be used for routing. Therefore, a node may achieve a higher reliance value than they deserves if it lies on many simple paths and very few simplest paths. By only considering the simplest paths in the network, the assignment of reliance values more accurately represents the degree to which a node is relied upon since only those paths that could actually be used will contribute to a node's reliance value.

6.1.4 Contraction Model

The final model to be discussed is the *contraction model*, which is an extension to the simplest paths model. It is often the case that the use of one node requires the presence of another node. For example, consider the network shown in 6.21.

In the diagram, it can be seen that node F can only be included in a path if it is preceded by node D and succeeded by node G. Consequently, one might consider *contracting* the network to eliminate node F. Thus, the incoming and outgoing edges of F are combined to form a single edge from D to G as shown in Figure 6.22.

Note that further contractions in the network are not possible, since none of the remaining

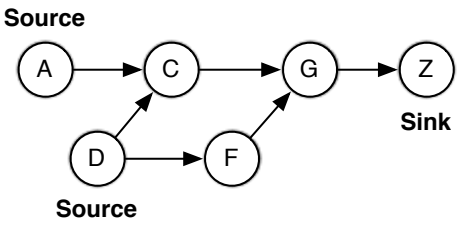


Figure 6.21: An example network

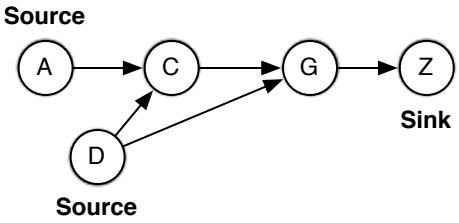


Figure 6.22: A contraction of the network shown in Figure 6.21

nodes have a single incoming and single outgoing edge. Thus, their use is not confined to a single case and the nodes cannot be contracted. For example, node G cannot be contracted by connecting nodes C and D to sink Z, since such a modification results in the loss of information that node G is required independently by nodes C and D. The rules for network contraction can be generalised to the following:

- A node may only be contracted if it has exactly one incoming and one outgoing edge.
- Contractions should be carried out on a network made up from simplest paths.
- The network does not become multi-edge, i.e. there may only be one edge from a node A to a node B.
- Sources and sinks may not be contracted.

The rules are explained below.

A node must have exactly one incoming and outgoing edge; otherwise the contraction of the node may result in a loss of information. As long as data can only enter and exit from known neighbours, the node's use in routing is entirely deterministic.

It may not be sufficient to contract a network based on the connectivity of nodes alone, since the fact that a node A can transmit to a node B does not mean that the edge AB is suitable to route from source to sink. For example, consider the network shown in Figure 6.16. Although node C can transmit to node B, it is not possible to form a simple path from source to sink that uses edge CB. Furthermore, contractions in the network are impossible, since every non-sink node has more than one incoming and more than one outgoing edge. Contractions can only be carried out by reducing the number of edges to those that may be used in routing from sources to sinks, i.e. the set of simple or simplest paths. The contraction algorithm considers the set of simplest paths due to the advantages already discussed in Section 6.1.3.3.

A *multi-edge* network is defined as a network in which there may be more than one edge from a node A to a node B. For example, in Figure 6.23, AB is multi-edge because there are two edges from node A to node B. AC is not multi-edge because there is only one link from A to C. The other edge is from C to A. When nodes are contracted, the network does not become multi-edge since the metric is only concerned with revealing those nodes whose use is contentious, i.e. relied upon by multiple sources. Making the network multi-edge does not help in finding nodes with contention.

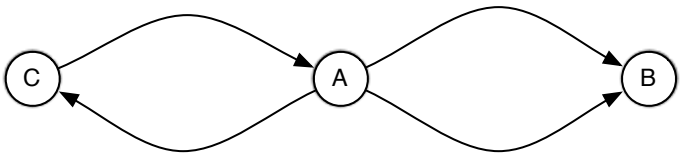


Figure 6.23: Multi-edge examples

Finally, sources and sinks may not be contracted since they are automatically potential sources of contention. If a source has no incoming edges (or a sink has no outgoing edges) then the node cannot be contracted due to the requirement that the node has exactly one incoming and one outgoing edge. If the source does have an incoming edge (or the sink has an outgoing edge) then the node is being used for routing as well as being a producer (or receiver) of data. Thus, the node has contention and cannot be contracted.

Having contracted the network and removed as many nodes as possible, the set of simplest paths from sources to sinks can be enumerated. As with the simplest paths model, the relative reliance value for a node X to a (source, sink) pair (A, Z) is the proportion of simplest paths on which the node lies. The absolute reliance value for node X is the average

relative value of X across all (source, sink) pairs for which there exists at least one path. As with the other models, it is assumed that a path cannot contain more than one sink node.

6.1.4.1 Example

Consider the network shown in Figure 6.24 which features two sources (A and D) and a sink (Z).

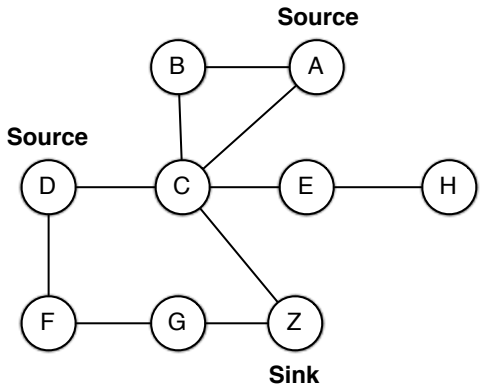


Figure 6.24: An example network with two sources (A and D) and one sink (Z)

As discussed in Section 6.1.3.1, the simplest paths in this network are ACZ, DCZ and DFGZ. Figure 6.25 shows the tree of simplest paths for this network.

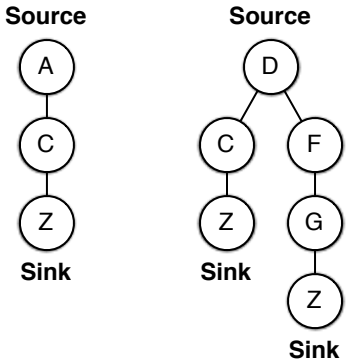


Figure 6.25: A tree showing simplest paths from sources A and D to sink Z

In this tree it can be seen that node F has only a single incoming (D) and outgoing (G) edge and similarly, node G has a single incoming (F) and outgoing (Z) edge. Thus, nodes F and G can be contracted. No other nodes may be contracted since nodes A and D are sources,

node Z is a sink and node C has two incoming edges from A and D. The resulting tree is shown in Figure 6.26.

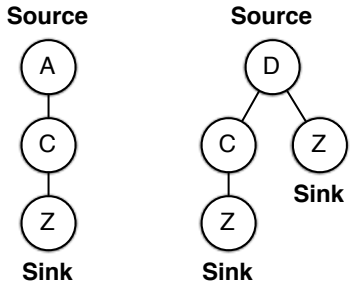


Figure 6.26: A tree showing simplest paths from sources A and D to sink Z

However, these contractions result in the formation of a subpath DZ and thus, the path DCZ is no longer a simplest path. Therefore, the two remaining contracted simplest paths are DZ and ACZ. The reliance values can now be calculated by examining the number of contracted simplest paths on which each node lies. Each source only has a single (source, sink) path. Nodes A, C and Z all have a relative reliance of 1.0 to the pair (A, Z). Other nodes do not appear on the path and so have a relative reliance of 0.0 to (A, Z). Similarly, nodes D and Z have a relative reliance of 1.0 to (D, Z) and other nodes have a relative reliance of 0.0 to (D, Z). The absolute reliance is calculated by averaging the relative reliances. For example, node Z has an relative reliance of 1.0 to (A, Z) and (D, Z) giving it an absolute reliance of 1.0. The relative and absolute reliance values for each node are shown in Table 6.5.

Node	Relative Reliance to (A, Z)	Relative Reliance to (D, Z)	Absolute Reliance
A	1.00	0.00	0.50
B	0.00	0.00	0.00
C	1.00	0.00	0.50
D	0.00	1.00	0.50
E	0.00	0.00	0.00
F	0.00	0.00	0.00
G	0.00	0.00	0.00
Z	1.00	1.00	1.00

Table 6.5: Relative and absolute reliance values calculated from the contracted simple paths of Figure 6.24

Having calculated the reliance values, the network shown in Figure 6.24 is used to select a path. The contracted simplest paths cannot be directly used, because they do not represent how nodes are actually connected. For example, node D is not in reality directly connected to node Z.

6.1.4.2 Worst-Case Growth Rate

As with the other node reliance models, there is no means of determining node reliance based on contracted simplest paths without enumerating all the paths and determining the proportion of paths on which each node lies. The tractability of the contraction model is dependent on the number of paths that may have to be enumerated in a worst-case scenario.

The worst-case growth rate of the contraction model is the same as for the simplest paths model, which is repeated here for convenience. Each node should have 3 children of the same depth, each of which should have the same parents. For example, Figure 6.27 shows an example network in which a total of 14 nodes are placed in such a manner as to maximise the number of simplest paths. The network contains 3^4 = 81 simplest paths.

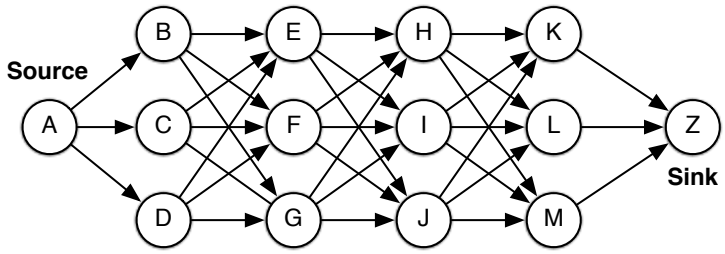


Figure 6.27: The worst-case scenario layout for generating simplest paths

The worst growth rates are the same because the contraction model seeks to reduce the number of simplest paths in the network and none of the paths in the worst-case growth rate of the simplest paths model can be contracted. Each node is immune to contraction by virtue of being a source, a sink or by having more than one incoming or outgoing edge. Thus the worst-case growth rate of the contraction model is the same as for the simplest paths model.

6.1.4.3 Advantages

The contraction model provides node reliance values of higher accuracy than the simplest paths model. Only those nodes whose use is contentious are assigned reliance values. For example, consider the network shown in Figure 6.28a and the contracted network shown in Figure 6.28b

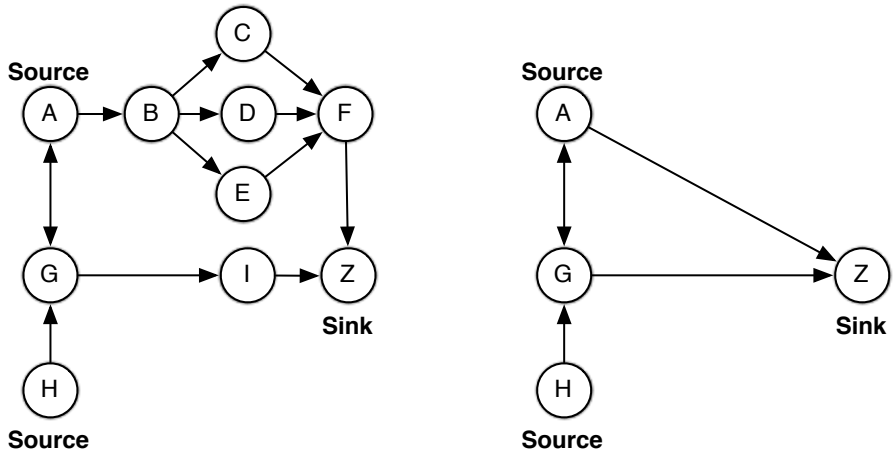


Figure 6.28: Contracting a network may allow a more accurate measurement of node reliance values

The contracted network shows only those nodes that cause contention, i.e. nodes that are directly relied upon by multiple sources in routing to a sink node. The use of other nodes (such as B) is dependent on other nodes whose use causes contention. In this example, node B can only be used by source H after node G has been used, whose use causes contention with source A. Consequently, the use of node B is not directly contentious. By considering only the contentious nodes, reliance values are not inflated due to numerous possible (but non-contentious) nodes being present.

For example, in Figure 6.28a there are three simplest paths from source A to sink Z through node B and only one through node I. Therefore, node I would be used in preference to node B due to its lower reliance. However, if source A were to avoid node B, it would be required to use the contentious node G which is a bottleneck to source H. In this example, by allowing non contentious nodes to affect node reliance values, an undesirable node is selected for use in a path.

6.1.5 Reliance Values for a Realistic Physical Layer

As discussed in Section 2.5.2, wireless communication may be unreliable, unidirectional and temporary. Each of the models shown in this chapter relies upon the notion of two nodes being *connected*. However, the concept of two nodes being connected is undefined for a realistic physical layer and it is not obvious when two nodes should be considered connected. Communication from a node A to a node B is more accurately represented by a probability $p_{a,b}$ that a bit transmitted by node A is received by node B.

The node reliance calculations require a representation of the network in order to determine the number of paths between sources and sinks. The assumption is made that if an edge can be detected in the process of determining the network topology, then that edge is likely to have a high probability of successfully transmitting data. If the edge was unreliable, then it is unlikely that it would have been detected. Thus, the reliability of an edge is directly proportional to the probability with which that edge will be reported when the topology is examined, since a communication must take place for an edge to be detected.

It is therefore hypothesised that these heuristics are suitable for use in a realistic physical layer without modification and are capable of producing paths whose edges are sufficiently reliable for a realistic physical layer.

6.1.6 Conclusion

Three models have been presented for calculating node reliance in a network. Each model bases the reliance value of a node on the proportion of paths that the node lies on between sources and sinks. In the *simple* paths model, the entire set of simple paths is considered. The *simplest* paths model considers only the set of paths from which no nodes can be removed, i.e. no paths that contain unnecessary nodes are considered. Finally, the *contraction* model considers the set of paths containing nodes whose use may be contentious between sources.

These three models can be used to assign link/node costs, which is part of the costing task of a heuristic. The next section addresses how the link/cost costs should be combined to assign costs to paths.

6.2 Path Costs

As discussed in Chapter 3, the second part of the costing task that a routing heuristic undergoes is the process of assigning costs to paths based on the costs of individual elements, i.e. the link/node costs. Different options include *shortest path* routing, *lexicographic ordering* and *min/max element*. The advantages and disadvantages have already been discussed in Chapter 3. This section discusses the key points of using these path cost modules with one of the node reliance heuristics that have been discussed earlier in this chapter. The network shown in Figure 6.29 is used as an example, together with node reliance values shown in Table 6.6, which are calculated based on the simple paths model. The following sections discuss each of the path cost models.

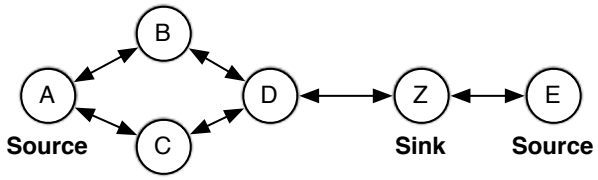


Figure 6.29: An example network with two sources, A and D

Node	Relative Reliance to (A, Z)	Relative Reliance to (D, Z)	Absolute Reliance
A	1.00	0.00	0.50
B	0.50	0.00	0.25
C	1.00	0.50	0.75
D	0.50	1.00	0.75
E	0.00	0.00	0.00
F	0.50	0.50	0.50
G	0.50	0.50	0.50
H	0.00	0.00	0.00
Z	1.00	1.00	1.00

Table 6.6: Reliance values calculated from Figure 6.29 using the simple paths model

6.2.1 Shortest Path

Shortest path was previously discussed in Section 3.2.2.1. A path's cost is calculated by adding the individual costs of each node that lies on that path and can be calculated using Dijkstra's algorithm [29].

In the example network, the shortest path from source A to sink Z is ACZ and has a cost of 2.25 (0.50 + 0.75 + 1.00) and the shortest path from source D is DCZ with a cost of 2.50 (0.75 + 0.75 + 1.00). However, note that source D uses a single high cost node (C) rather than two low cost nodes (F and G) which results in contention between source A and source D. Both sources make use of node C, which may cause that node's batteries to drain quickly. If the batteries of node C drain quickly then source A will be disconnected from the network.

As discussed in Chapter 3.2.2.1, a disadvantage to shortest path routing is that it may cause a subset of nodes to be overused, thus causing those nodes to quickly expire. A routing heuristic that combines shortest path routing with node reliance should not suffer this problem because nodes are encouraged to use nodes with low values. Since nodes with low values appear on few (source, sink) paths, each source is encouraged to use nodes that other sources are unlikely to be able to use. Thus, no common subset of nodes should be overused.

6.2.2 Lexicographic Ordering

Lexicographic ordering was discussed in Section 3.2.2.2. In lexicographic ordering, a path is not assigned a numeric cost. However, a set of paths may be compared to determine which has the lowest order. Specifically, a path P is said to have a lower lexicographic order than a path Q if the highest cost of all the nodes in P is less than the highest cost of all nodes in Q. The lowest lexicographic order path can be found using Dijkstra's algorithm in which the cost to reach each node is calculated lexicographically rather than being based on shortest path.

In the example network above, the lowest lexicographic order path from source A to sink Z is ACZ and the lowest lexicographic order path from source D is DFGZ even though the total cost is higher than that of the path DCZ. In this network, the paths selected by sources

A and D do not form any contention.

6.2.3 Min/Max Element

Min/Max elements were discussed in Section 3.2.2.3. In the context of finding the cheapest path, an appropriate tactic would be to select the path whose node of maximum node reliance was the least.

In the example network above, all paths contain the sink, which has the highest node reliance of 1.00. If we were to then consider the second highest node reliance on each path then the algorithm being used is lexicographic ordering.

6.2.4 Conclusions

Min/Max element has shown that it may be insufficiently precise in determining which path is the cheapest to use. Lexicographic ordering may be considered as an extension of min/max element, which reduces the number of equal cost paths. Consequently, the use of min/max element to determine the cheapest path with respect to node reliance is not considered any further.

The remaining two methods have had their advantages and disadvantages outlined in Section 3.2.2 and are both considered to be valid means for determining the cheapest path.

6.3 Summary

The aim of a node reliance heuristic is to identify those nodes that are essential in enabling (source, sink) routing. It is hypothesised that by identifying those nodes and avoiding their use where possible, the connectivity of sources and sinks can be kept high for as long as possible.

This chapter discusses how a node reliance heuristic may work. Chapter 3 demonstrates how a heuristic may be broken up into two tasks. These tasks include the costing task in which node/link costs and path costs are assigned, and a selection task in which a path is

selected for routing.

The selection task is eschewed, due to its presence overcomplicating the analysis of node reliance. Instead, this chapter focuses on the possibilities for link/node costs and path costs.

Several link/node cost options in the literature for node reliance are analysed. However, many of them are unsuitable for WSNs and many are unsuitable for use in a (source, sink) architecture. Therefore, three new models are introduced, known as the simple paths model, the simplest paths model and the contraction model. Each operates by assigning reliance values to nodes based on the proportion of (source, sink) paths on which each node lies. A node that lies on more paths is deemed to be more important, since its loss invalidates a greater number of routing possibilities between sources and sinks.

Finally, the different options for path costs are analysed. These are discussed in Chapter 3 and include shortest path, lexicographic ordering and min/max element. Of these, min/max element is shown to be too imprecise for selecting paths, while the other approaches both remain plausible for use with a node reliance heuristic.

Chapter 7 proposes and presents the results to an experiment designed to compare the different possible heuristics for node reliance with other approaches of third parties. The results are used to determine which heuristic is best at maintaining (source, sink) diversity over time.

Chapter 7

Routing Heuristic Experiment

This thesis addresses the issue of maintaining a high source diversity in WSNs for as long as possible. Chapter 4 discusses a number of routing heuristics and protocols from the literature. However, none of these are suitable for addressing the above issue. In Chapter 6, a family of heuristics known as node reliance is presented to address this issue. These heuristics work by assigning a value to each node, indicating how important that node is in routing data from sources to sinks. It is theorised that by forming routes that avoid important nodes, a higher source diversity can be achieved for longer.

Chapter 5 discusses various issues involved with measuring source diversity over time, including a new metric known as CWT, the need to normalise measurements across multiple routing heuristics and a justification and suitable configuration for using the Castalia-1.3 simulator [90] to compare routing heuristics and protocols.

This chapter describes experiment that analyse the effectiveness of various routing heuristics in keeping source diversity as high as possible for as long as possible. The heuristics being examined include the new node reliance heuristics described in Chapter 6 as well as several heuristics that are described in Chapter 4.

Section 7.1 describes the parameters that are used for the experiment, including the number of nodes, number of sources and number of sinks. Section 7.2 describes the experimental procedure for using the simulator. Section 7.4 describes the requirements for this particular experiment. Finally the results are presented and analysed in Section 7.5.

7.1 Parameters

The aim of the experiment is to determine how well each routing heuristic managed to maintain high source diversity over time. In order to make generalisations about each routing heuristic's performance, it is necessary to measure the performance of each routing heuristic across of a variety of parameters, as discussed below:

- Number of nodes (n)
  - 2, 5, 10, 20, 40
- Number of sources
  - 1, 2, 5, 10, 20, $n-10$, $n-5$, $n-2$, $n-1$
- Number of sinks
  - 1, 2, 5, 10, 20, $n-10$, $n-5$, $n-2$, $n-1$
- Node placement:
  - Regular triangular grid
  - Regular square grid
  - Regular hexagonal grid
  - Uniform random
- Data rate:
  - Static
  - Random

The number of nodes in the network (n) ranges from 2 to 40. At least 2 nodes are required, since a (source, sink) architecture must include a source node and a sink node. The maximum value of 40 is selected based on the example topologies summarised in Section 2.4. In that section, it can be seen that the majority of WSN deployments use fewer than 40 sources. Between these two extremes, values are chosen to allow the analysis of routing heuristic efficacy with an exponentially increasing number of nodes. An advantage of

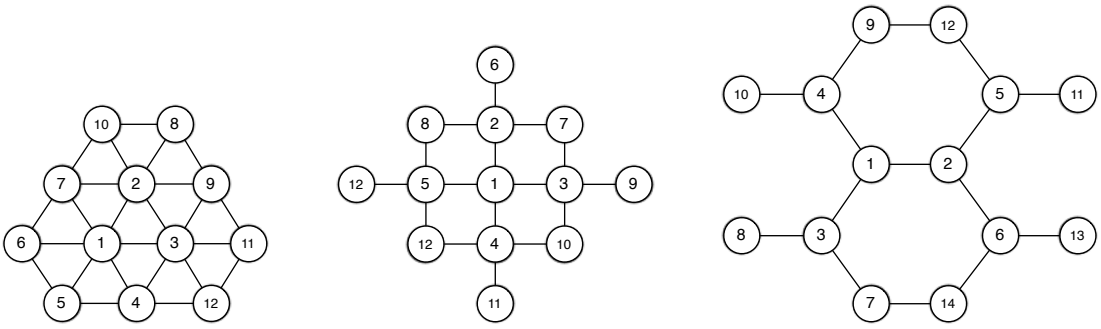
choosing an exponential growth is that trends can be discovered with a smaller range of parameters.

A (source, sink) architecture demands the presence of at least one source and one sink. A node is not permitted to be both a source and sink simultaneously, since doing so would remove the need for routing, thus defeating the purpose of the experiment. Consequently, a maximum of $(n - 1)$ sources or sinks can be present in any network. The example deployments discussed in Section 2.4 suggest that the number of sources should generally be $(n - 1)$. However, this may not always be the case, since some nodes may produce uninteresting or duplicate data. The experiment therefore includes networks with non-source intermediate nodes.

Node *Placement* refers to the physical shape resulting from the deployment of nodes. By controlling the degree of each node, it is possible to control the overall connectivity and number of paths of each node in the network. For example, if nodes are deployed next to each other then the connectivity is higher than if the nodes are placed in a straight line such that their separation distance is equal to their maximum communication range. Since node reliance heuristics are heavily dependent on numbers of paths between sources and sinks, this parameter allows an analysis of the effect of node connectivity on routing heuristics. The nodes are all placed into a 500 metre by 500 metre area using the node placement algorithms described in Appendix A, which is briefly summarised here.

In the uniform random deployment, nodes are added to the network with random (x, y) coordinates such that they are connected to at least one node that has already been placed. Thus, the network always remains connected. Since the coordinates of each node are uniformly random, it is expected that this will produce a random graph. However, regardless of whether a random graph is produced or not, this node placement strategy mimics the way in which a network may be placed in real life, i.e. by placing nodes one at a time and adjusting them until they connect to other nodes in the network.

In the case of triangular, square and hexagonal placements, each node has a target degree of six, four and three respectively. New nodes are added to the network such that they increase the degree of the oldest node whose degree is below the target. Neighbours of a node are evenly spaced at a fixed distance, i.e. the node's maximum communication range, away from the node and are added in clockwise order. Figure 7.1 shows the order in which nodes are added to the network in triangular, square and hexagonal networks, respectively.



(a) (b) (c)

Figure 7.1: Node positions for triangular, square and hexagonal node placements

Periodic data generation is used, since the majority of deployments studied in section 2.4 operate in this manner. The data rate parameter makes it possible to judge each routing heuristic's ability to handle even and uneven node usage when the generation rate is static or random across each node. Random data rates are integers between 1 and 10 seconds. The static data rate is always 5 seconds.

7.2 Procedure

Each heuristic operates on a collection of networks. During the simulation, an *oracle* provides the data required for the operation of each heuristic, so that each source can route without having to expend energy to find or maintain any network data. The experiment is therefore able to examine the ability of each heuristic to maintain (source, sink) diversity over time without considering whether it can efficiently gather available paths or other network information required for routing.

The following heuristics are tested as part of this experiment:

- *SimpleLex*: The simple paths model for link/node costs as discussed in Section 6.1.2 in combination with a path costs module of lexicographic ordering.
- *SimplestLex*: The simplest paths model for link/node costs as discussed in Section 6.1.3 in combination with a path costs module of lexicographic ordering.
- *EnergyAware*: Singh's minimize cost/packet [105] heuristic in which the remaining

energy of nodes is represented as a discrete set of energy levels which increase in granularity as the remaining energy approaches zero as proposed by Lin [76]. Both of these are discussed in Section 4.5.1.

- *MinHop*: The minimum hop routing heuristic discussed in Section 4.2.
- *ContractedLex*: The contraction model for link/node costs in combination with a path costs module of lexicographic ordering.
- *SimpleShortest*: The simple paths model for link/node costs in combination with a path costs module of shortest path.
- *SimplestShortest*: The simplest paths model for link/node costs in combination with a path costs module of shortest path.
- *ContractedShortest*: The contraction model for link/node costs in combination with a path costs module of shortest path.

Elements of the EnergyAware heuristic are examined in Section 4.5.1 and are briefly summarised here. The link/node cost of a node is based on Singh's minimize cost/packet heuristic in which a node's cost is inversely proportional to the remaining energy of the node, i.e. nodes with little energy are expensive to use. The path cost is established using shortest path. Although Singh's work is not particularly recent (1998), modern routing protocols frequently use remaining or consumed energy as part of their routing decisions [110][82][127][102]. It also achieves load balancing, since nodes that have consumed more energy are less likely to be selected for routing. The EnergyAware heuristic also makes use of Lin's [76] proposal in which the remaining energy of a node is represented by the use of four discrete energy levels that map to a logarithmic scale. Level 1 corresponds to between half and full energy, level 2 between a quarter and half energy, level 3 between an eighth and a quarter energy and level 4 below an eighth energy.

MinHop selects the (source, sink) path with the smallest number of nodes in it. It is used as a basis for comparison here due to the frequency with which it is referenced in the literature.

The experiment proceeds as follows:

1. The Castalia-1.3 simulator is configured according to the guidelines presented in Section 5.4.

2. Each node in the network provides data to the *oracle*, depending on the heuristic being carried out. Nodes determine their connectivity by the exchange of *beacon* messages.

3. The data generated by a source is random (between 0 and 100 bytes). When data is ready to be routed, the oracle provides the necessary network data for the heuristic to operate and select a path to a sink.

4. The data is routed along the selected path.

5. Nodes are required to keep the oracle updated with any information required by the heuristic such as remaining energy or connectivity. Nodes that expire are also required to notify the oracle that they are no longer functioning.

6. The process continues until all source nodes have expired.

7. Measurements discussed in Section 7.3 are made during the simulation.

8. No energy is expended in calculating or maintaining paths, as information produced by the oracle is always accurate.

As discussed in Section 5.4.4, the experiment is repeated for each combination of parameters. Furthermore, since several parameters are random, such as the random seeds that govern radio communications, the positions of nodes, the selections of sources and sinks and the data generation rates (when a random rather than static rate is selected), the experiment is repeated for each combination of parameters, each time changing the random selections.

The experiment is repeated 10 times. This is based on third party work that examines trends in routing protocols, in particular, the work of Broch [14] whose experiments were repeated 10 times and Xu [123] whose experiments were repeated 30 times. The lower rate is selected here due to the larger set of experimental parameters dealt with in this thesis. The higher repeat rate is reserved for later experiments that are discussed in Chapters 9 and 10 which deal with routing protocols. Since the routing protocols must collect their own topology data, which may fail to arrive, or may arrive late for routing decisions to be made, the performance is expected to vary more than the experiment in this chapter. Thus, the higher rate of repeat experiments is used for them.

7.3 Measurements

For each heuristic, two measurements are made:

- Total data transfer, and
- CWT (weighting factor 2).

The total data transfer is measured in order to provide an indication of how many sensor readings can be transferred to the sink nodes during the operation of the network. However, as discussed in Chapter 5, this measurement is prone to the source-forwarding problem. Therefore, a high total data transfer does not necessarily reflect ideal application behaviour.

The CWT (as discussed in Chapter 5) is also measured in order to measure the extent to which sources remain connected to sink nodes during the operation of the application. A weighting factor of two is selected for the CWT. A factor of 0 would generate to total data transfer. A value of 2 is selected to significantly encourage a high simultaneous connectivity of sources. The value is also chosen to be consistent so that the levels of connectivity in networks with different numbers of sources could be compared. Consequently, it has to be independent of the number of sources. The final value of the weighting factor is arbitrarily selected based on these restrictions.

The node reliance heuristics are expected to perform well with respect to CWT, since they are specifically designed to maintain (source, sink) connectivity over time. By additionally measuring the total data transfer, it is possible to determine the type of tradeoffs in total data transfer (if any) as a result of achieving high source diversity for long periods of time.

7.4 Requirements

The experiment has three requirements:

- Communication is perfect,
- Sinks have a near-infinite supply of energy, and,

- Other nodes have only 14.58 J of energy.

These requirements are explained in more detail below.

As stated in Section 2.5.2, communication in a WSN is unreliable and links between nodes may be unidirectional. However, in this experiment it is necessary to assume perfect communication so that the oracle can provide accurate details regarding the network. If links between nodes are realistic, then there is no longer a certainty as to whether two nodes are connected. Consequently, any data provided by the oracle regarding the network may be inaccurate, leading to poor routing decisions. In order to ensure that the routing heuristics themselves are being examined, rather than the ability of the oracle to provide network data or the routing heuristic's ability to handle inaccurate data, it is necessary to ensure that the oracle's data is always accurate. Thus, the network required perfect communication.

Sink nodes are given an infinite supply of energy since it is possible that any sink may be required to deal with incoming messages from every source node. As discussed in Section 2.1, sink nodes are commonly high powered. Thus, this requirement is not deemed to be unreasonable.

Non-sink nodes are given an initial energy of 14.58J, which is 1/200th of a TMote sky's pair of AA batteries. This requirement allows each simulation to terminate in a reasonable period of time. In a real deployment, the initial energy of each node would vary randomly. However, in the simulations, the initial energy of each non-sink node is kept consistent so as to allow trends in network configurations to be better analysed.

7.5 Results

Table 7.1 shows the results of the experiment for each routing heuristic, including the name of the heuristic, the means of associating link/node costs, the system used for establishing path costs, the results of the CWT metric, the standard deviation of the CWT metric across all simulations, the total data transfer metric and the standard deviation of the total data transfer metric across all simulations.

Several observations can immediately be made from these results.

Name	Node Cost	Path Cost	CWT		Transfer	
			Value	SD	Value	SD
EnergyAware	Energy	Shortest	0.998	0.02%	0.992	0.04%
SimpleShortest	Simple	Shortest	0.991	0.07%	0.990	0.07%
SimplestShortest	Simplest	Shortest	0.978	0.08%	0.984	0.07%
ContractedShortest	Contracted	Shortest	0.978	0.08%	0.984	0.07%
SimpleLex	Simple	Lex	0.968	0.15%	0.979	0.12%
MinHop	Unit	Shortest	0.966	0.10%	0.982	0.08%
ContractedLex	Contracted	Lex	0.950	0.14%	0.972	0.09%
SimplestLex	Simplest	Lex	0.949	0.14%	0.971	0.09%

Table 7.1: Results of the experiment when used with static data rates.

Firstly, the standard deviation of each routing heuristic is at most 0.15% of the CWT and 0.12% of the total data transfer, suggesting that the uncertainty of the measurements is low and that the number of repeat experiments is sufficient.

Secondly, all of the heuristics perform similarly with respect to both metrics. The best performing routing heuristic (EnergyAware) is only 5.16% better than the worst performing routing heuristic (SimplestLex) when averaged over all simulations.

Thirdly, the highest performing heuristic is EnergyAware. The measurements of 0.998 and 0.992 for CWT and total data transfer respectively indicate that the heuristic performs close to optimal under most circumstances. However, this experiment does not consider the costs associated with updating the network with the remaining energy on each node. A routing protocol based on the EnergyAware heuristic may expend a significant amount of energy in sending update messages regarding the remaining energy of nodes. The second highest performing heuristic for both CWT and total data transfer is the SimpleShortest heuristic followed by the SimplestShortest heuristic, both of which are original contributions of this thesis. Since these heuristics do not continually expend energy to update other nodes on their status, one might expect routing protocols based on those heuristics to achieve a higher CWT or total data transfer score than one based on the EnergyAware heuristic.

The remainder of this section makes a number of other observations regarding the experiment. Section 7.5.1 discusses the time taken for the simulations to run. Section 7.5.2 discusses the use of shortest path over lexicographic ordering for determining path costs.

The effects of random data rates are explained in Section 7.5.3, the effect of path availability is discussed in Section 7.5.4. The consequences on CWT and data transfer on increasing the proportion of sources are discussed in Sections 7.5.5 and 7.5.6 respectively. Finally, the effect of increasing the number of sources is discussed in Section 7.5.7.

7.5.1 Computation Time of Simple Paths Heuristic

The *simulation time* of an experiment refers to the time taken for the simulation to run and produce output. Several heuristics take a long time to compute even before networks with 40 nodes can be carried out. Table 7.2 shows the maximum computation time required to simulate any network containing 20 or fewer nodes, according to the routing heuristic used.

Name	Node Cost	Path Cost	Max Computation Time (s)
SimpleLex	Simple	Lex	456704
SimpleShortest	Simple	Shortest	455797
EnergyAware	Energy	Shortest	290
MinHop	Unit	Shortest	227
SimplestShortest	Simplest	Shortest	170
ContractedLex	Contracted	Lex	140
ContractedShortest	Contracted	Shortest	118
SimplestLex	Simplest	Lex	117

Table 7.2: Maximum simulation time for simulating a network of up to 20 nodes for each heuristic

The table clearly shows that those routing heuristics relying on the set of simple paths require several orders of magnitude more computation time than other routing heuristics. For example, in SimpleLex, at least one network requires over 126 hours to simulate. As discussed in Section 6.1.2.2, the worst-case growth of the number of simple paths is proportional to the factorial of the number of nodes in the network. Since 20 nodes require simulation times in excess of 126 hours, examining these heuristics in larger networks is infeasible. Furthermore, given that calculating the set of simple paths of networks containing 20 nodes is shown to be practically intractable, the SimpleLex and SimpleShortest heuristics are not examined in further detail.

Table 7.1 shows SimpleShortest to be the second most effective heuristic, with respect to CWT and total data transfer, of all heuristics under examination. One explanation for this effectiveness is that well-connected nodes, which are likely to transmit to many other nodes, may lie on many simple paths. Consequently, nodes that are likely to cause large amounts of overhearing are given high node reliance values and are avoided. By avoiding nodes that are likely to cause overhearing, the overall energy consumption of the network as a result of routing from sources to sinks can be lowered.

Given the SimpleShortest heuristic's apparent success, it may be worth exploring a faster way of calculating node reliance values. As it stands, the heuristic is unusable for networks of even moderate size.

7.5.2 Shortest Path vs. Lexicographic Ordering

According to Table 7.1, for both the CWT and total data transfer metrics, every routing heuristic's performance is improved by the use of shortest path ordering rather than lexicographic ordering. Since nothing else is changed between each version of the routing heuristic, it seems reasonable to conclude that shortest path ordering maintains connectivity and improves total data transfer better than lexicographic ordering.

Section 3.2.2.2 explains that in lexicographic ordering, there is no limit to the number of nodes that may lie on a path. For example, if the node of highest reliance value on path A is 1.0 and the node of highest reliance on path B is 0.9 then path B has the lowest lexicographic order, regardless of the number of nodes on path B. Consequently, the drain on batteries may be bigger when using path B than path A.

This hypothesis can be verified by considering the number of successful transmissions made per byte received by the sink. A routing heuristic that uses long paths will have a high ratio, since one successful transmission is made per hop that a message travels through on its path to the sink. As with the CWT and total data transfer metrics, this metric is normalised in order to provide a relative comparison between the routing heuristics. Table 7.3 shows, for each heuristic, the average ratio between normalised number of transmissions and normalised total data transfer. Since SimpleLex and SimpleShortest are only practical in networks containing 20 or fewer nodes, the averages are only calculated for those networks. The table clearly shows that those heuristics using lexicographic ordering

have higher ratios than those that use shortest path ordering. Consequently, more energy is expended by the entire network in order to route messages from sources to sinks.

Name	Node Cost	Path Cost	Transmissions/Msg	SD
SimpleLex	Simple	Lex	0.987	0.10%
ContractedLex	Contracted	Lex	0.981	0.12%
SimplestLex	Simplest	Lex	0.981	0.13%
ContractedShortest	Contracted	Shortest	0.964	0.16%
SimplestShortest	Simplest	Shortest	0.958	0.19%
EnergyAware	Energy	Shortest	0.957	0.19%
MinHop	Unit	Shortest	0.955	0.20%
SimpleShortest	Simple	Shortest	0.955	0.19%

Table 7.3: Normalised average transmissions per message received at the sink for networks of 20 or fewer nodes

If the paths used in lexicographic ordering are longer, the number of transmissions is higher. Consequently, the number of overheard messages could also be higher.

Table 7.4 indicates that heuristics that use lexicographic ordering do not necessary cause nodes to overhear more messages than heuristics that use shortest path. Consequently, the additional transmissions per transferred message shown in Table 7.3 can only be attributed to longer paths being used.

Name	Node Cost	Path Cost	Overheard/Msg	SD
SimplestLex	Simplest	Lex	0.976	0.14%
ContractedLex	Contracted	Lex	0.975	0.14%
MinHop	Unit	Shortest	0.967	0.16%
SimpleShortest	Simple	Shortest	0.966	0.21%
ContractedShortest	Contracted	Shortest	0.959	0.17%
SimplestShortest	Simplest	Shortest	0.954	0.19%
EnergyAware	Energy	Shortest	0.950	0.19%
SimpleLex	Simple	Lex	0.944	0.18%

Table 7.4: Normalised average overheard messages per message received at the sink for networks of 20 or fewer nodes

7.5.3 Effect of Random Data Rates

The results of the experiment with random data rates are shown in Table 7.5.

Name	Node Cost	Path Cost	CWT		Transfer	
			Value	SD	Value	SD
EnergyAware	Energy	Shortest	0.997	0.02%	0.991	0.02%
SimpleShortest	Simple	Shortest	0.992	0.06%	0.990	0.07%
SimplestShortest	Simplest	Shortest	0.980	0.07%	0.985	0.06%
ContractedShortest	Contracted	Shortest	0.980	0.07%	0.984	0.06%
SimpleLex	Simple	Lex	0.973	0.13%	0.979	0.11%
MinHop	Unit	Shortest	0.971	0.09%	0.982	0.07%
SimplestLex	Simplest	Lex	0.955	0.13%	0.972	0.09%
ContractedLex	Contracted	Lex	0.955	0.13%	0.972	0.09%

Table 7.5: Results of experiment when used with random data rates.

The differences exhibited by each heuristic between the use of static and random data rates are shown in Table 7.6. The standard deviations are high. Consequently, there is a large uncertainty regarding the effect of random data rates over static data rates and it is not possible to form a solid conclusion regarding the effects.

Name	Node Cost	Path Cost	CWT		Transfer	
			Value	SD	Value	SD
SimplestLex	Simplest	Lex	+0.006	4579%	+0.001	250%
SimpleLex	Simple	Lex	+0.005	6282%	+0.001	432%
ContractedLex	Contracted	Lex	+0.006	4761%	+0.001	199%
MinHop	Unit	Shortest	+0.005	3791%	0.000	434%
SimplestShortest	Simplest	Shortest	+0.002	5976%	+0.001	164%
ContractedShortest	Contracted	Shortest	+0.002	7151%	+0.001	176%
SimpleShortest	Simple	Shortest	+0.001	16079%	0.000	3882%
EnergyAware	Energy	Shortest	0.000	7684%	-0.001	76%

Table 7.6: Difference of using static and random data rates

There is, however, a general trend towards the total data transfer metric remaining un-

changed. This is because the maximum data that can be transferred in the network is entirely dependent on the network topology and the maximum data that any node can transfer. For example, if node A alone connects the sinks to the rest of the network, the maximum total data transfer depends on how much data node A can transfer before it expires. In the experiment, a node expends no energy while it is merely alive. Thus, all energy is expended in sending messages. Furthermore, since communication is perfect, energy is only expended for a successful transmission. Altering the data rates of nodes therefore had no effect on the capacity of the network and thus, the data rates remain largely unchanged.

Due to the similarity of results between random and static data rates, the effects of random data rates are not examined in any further detail in the remainder of this chapter.

7.5.4 Effect of Path Availability

It may be argued that an increase in path availability improves the performance of the routing protocol. As the number of paths increase, the set of (source, sink) node-disjoint paths should also increase, i.e. the set of paths in which no node appears on more than one path. A larger number of node-disjoint paths means that sources should be able to send data to sinks without affecting the connectivity of other sources. Consequently, the length of time that each source remains connected for remains maximised.

This theory is tested here by plotting number of simple/simplest paths in a network versus the CWT of a routing heuristic in that network. The number of nodes is varied in order to provide a range of networks with different numbers of simple/simplest paths. The number of sources and sinks remains constant, since these factors can also affect the CWT. The relative performance of the routing heuristic is unimportant and so it is sufficient to plot the raw CWT value rather than the normalised value. Figures 7.2 and 7.3 show the relevant graphs for increasing numbers of simple and simplest paths respectively. The number of sinks is fixed at 1 and the number of sources is fixed at 5. The probability of node-disjoint paths being present in small networks (10 nodes) is very small, and increased in the larger networks (20 - 40 nodes). As previously discussed, it is impractical to enumerate or count all simple paths for networks of greater than 20 nodes. Consequently, it is not possible to plot those points in the graph.

These graphs consider the set of *triangular* shaped networks that are produced using the

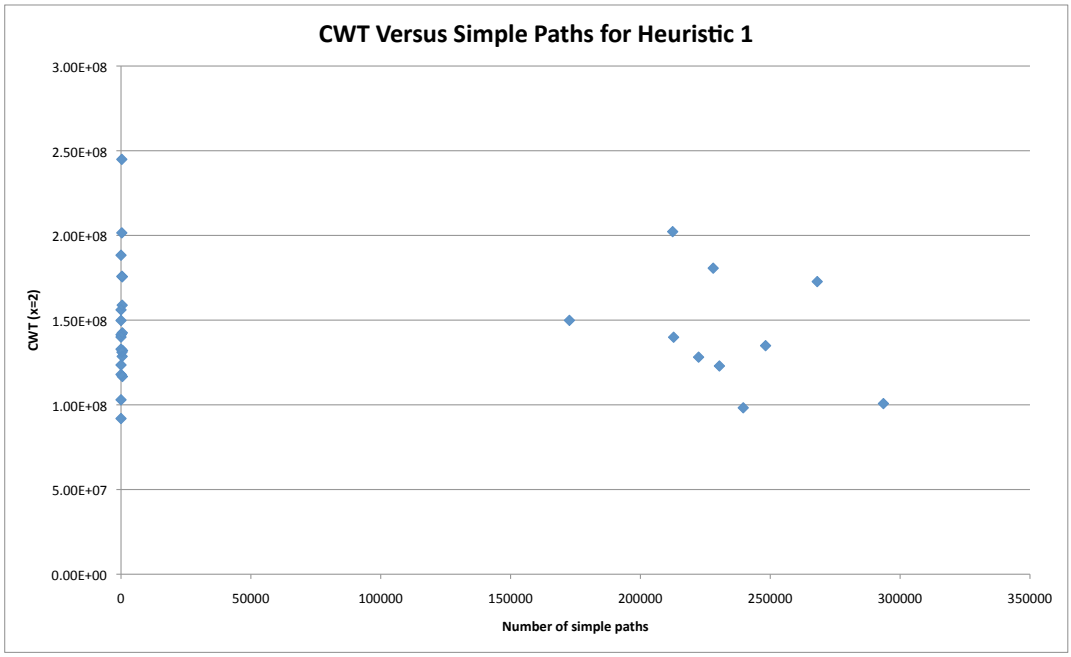


Figure 7.2: SimpleLex CWT vs. number of simple paths

algorithm described in Appendix A.2. Only triangular networks are considered for two reasons. Firstly, the addition of nodes in a triangular, square or hexagonal network will increase the number of paths at a steady and relatively predictable rate. Conversely, in a random network, the addition of a single node can have no effect on the number of paths or can increase the number of paths by orders of magnitude. Secondly, in a triangular network, the degree of each node is higher than in a square or hexagonal network. Thus, the number of possible paths grows more quickly with respect to the number of nodes, making the relationship between CWT and number of simple/simplest paths more obvious.

None of the graphs show a correlation between the number of paths (simple or simplest) and routing heuristic performance. There are two explanations for this outcome:

- overhearing messages, and
- the lack of relationship between simple/simplest paths and node-disjoint paths.

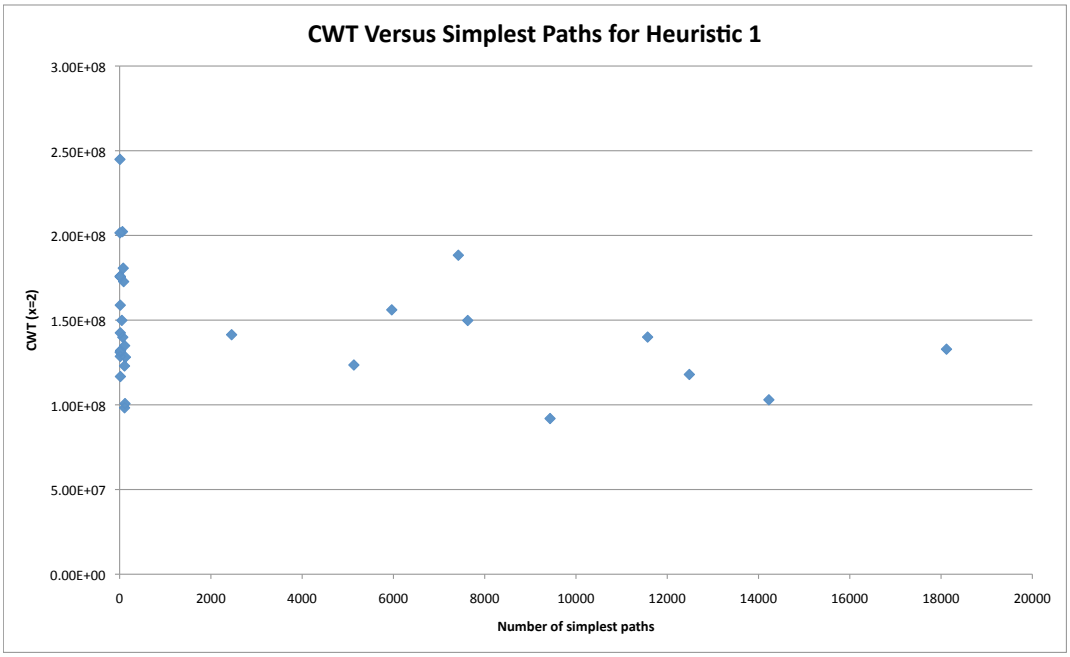


Figure 7.3: SimplestLex CWT vs. number of simplest paths

Firstly, the use of one particular path may cause nodes not on that path to suffer energy expenditure as a consequence of overhearing messages intended for other nodes, as previously discussed.

Secondly, keeping multiple sources connected for long periods of time requires node-disjoint paths, i.e. the paths from sources to sinks should have as few nodes in common as possible. If all (source, sink) paths have a single bottleneck then all data must flow through that bottleneck and it expires quickly, disconnecting all sources. Conversely, if every source has a node-disjoint path, then it may be possible to keep every source connected until it expires. There is no obvious relationship between the number of simple/simplest paths and the number of node-disjoint paths. Thus, the number of such paths is not indicative of the ability to keep sources connected to sinks and consequently there is no relationship between the number of simple/simplest paths and the CWT that can be achieved.

7.5.5 CWT and Increasing Proportions of Sources

This section examines the effect on CWT of increasing the proportion of sources in a network with a fixed number of nodes.

The graphs in Figures 7.4, 7.5 and 7.6 show the change in CWT caused by increasing the number of sources in networks of 10, 20 and 40 nodes respectively. Each graph considers the average CWT across all topologies (random, triangular, square and hexagonal).

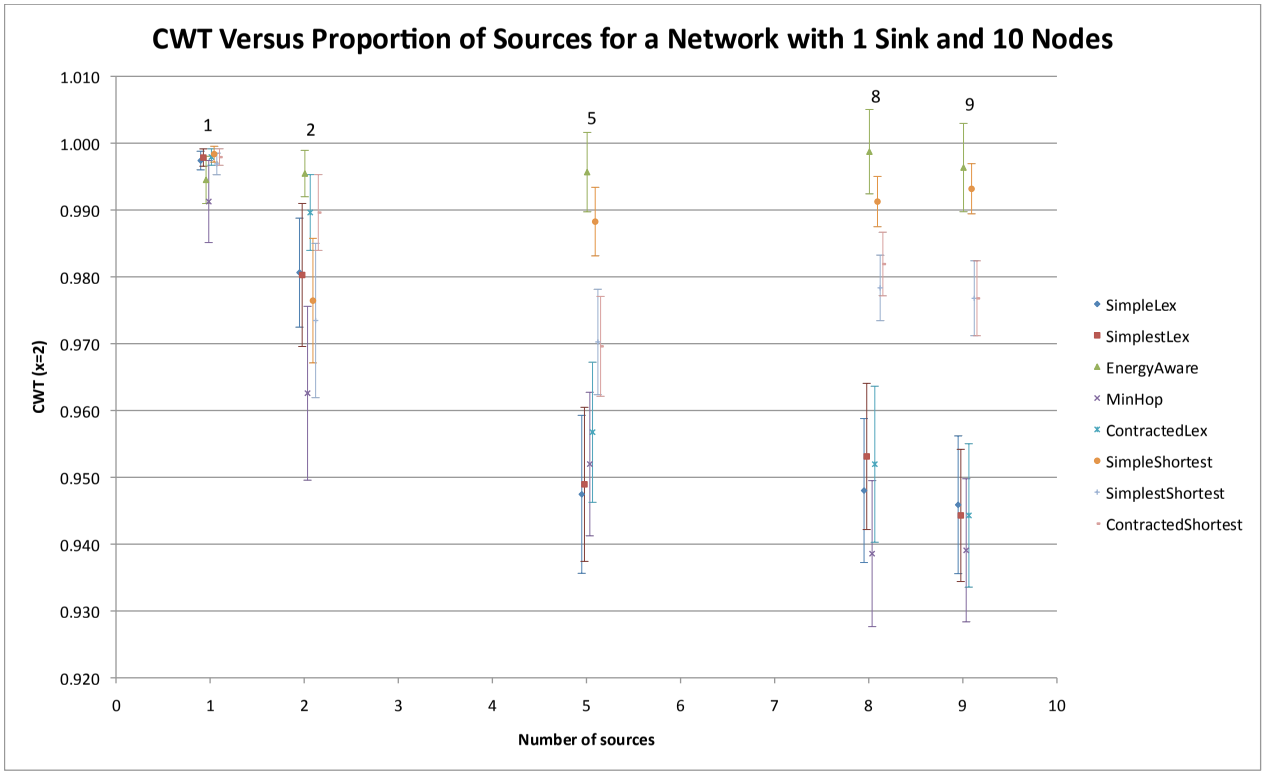


Figure 7.4: CWT vs. proportion of sources for networks with 10 nodes and 1 sink. Error bars represent one standard deviation

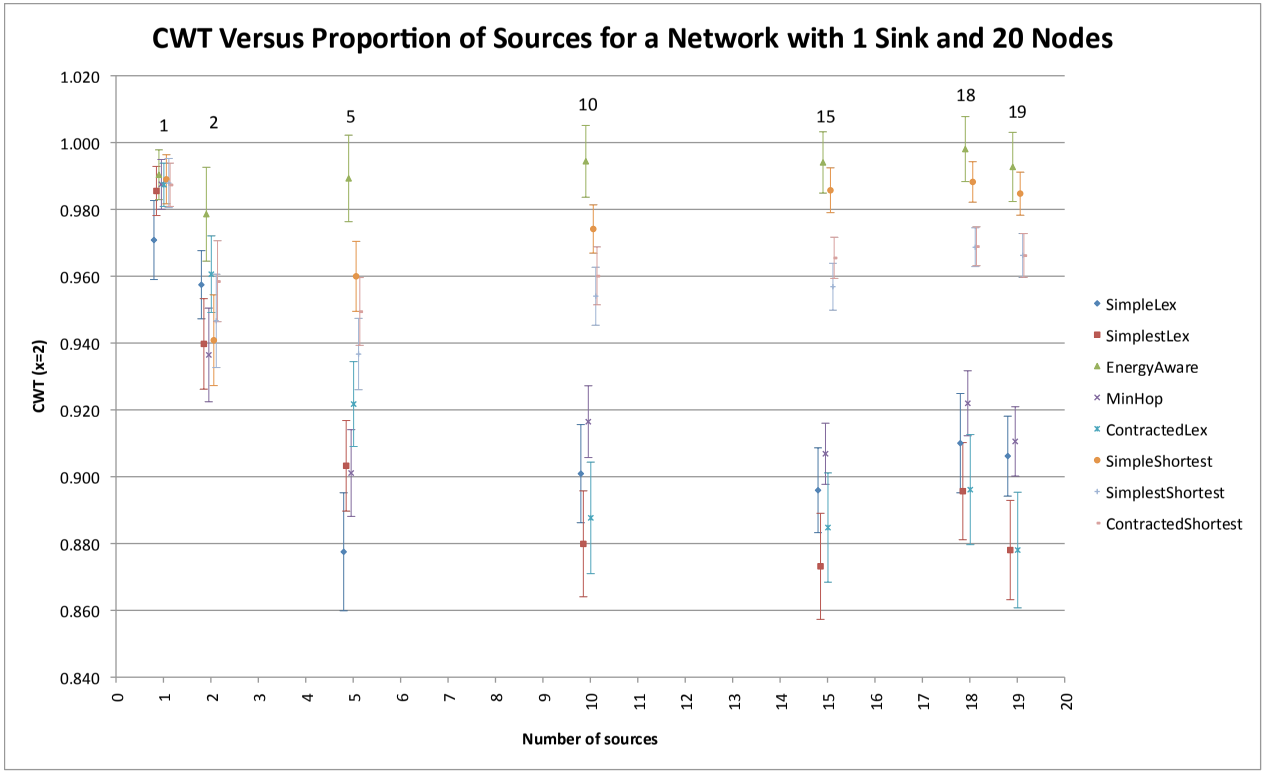


Figure 7.5: CWT vs. proportion of sources for networks with 20 nodes and 1 sink. Error bars represent one standard deviation

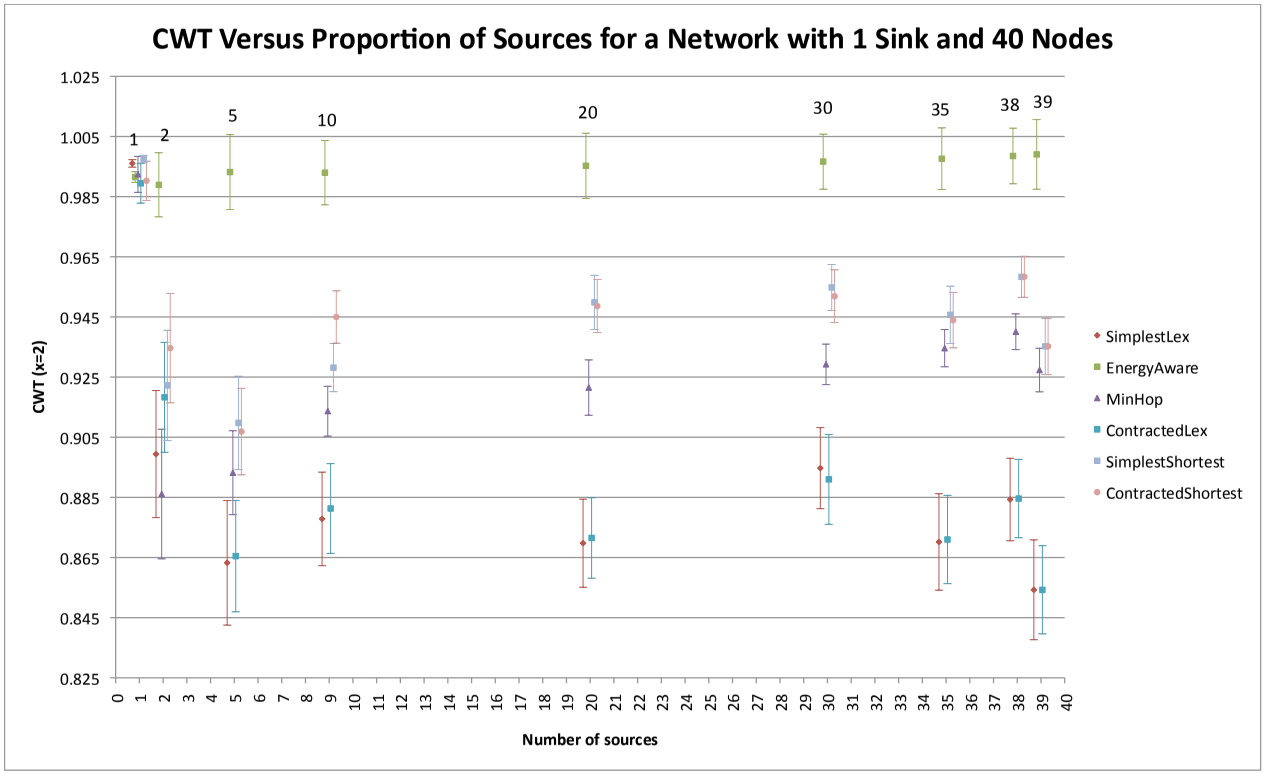


Figure 7.6: CWT vs. proportion of sources for networks with 40 nodes and 1 sink. Error bars represent one standard deviation

The performance of each routing heuristic remains approximately the same in each of the three graphs, with only two significant differences:

- The graph of Figure 7.6 considers 40 node networks and so does not show the performance of simple path based heuristics.
- In comparison to the other routing heuristics, MinHop performs worse in 10 node networks than in 20 or 40 node networks.

By examination of the results, it is possible to determine that the MinHop heuristic performs poorly due to simultaneously consuming energy on multiple nodes that could be used to form different routes to a sink. For example, consider the network shown in Figure 7.7, which features source 9, sink 5 and other intermediate nodes. Note that communication between pairs of nodes is bidirectional. The path used by the MinHop heuristic is shown in bold.

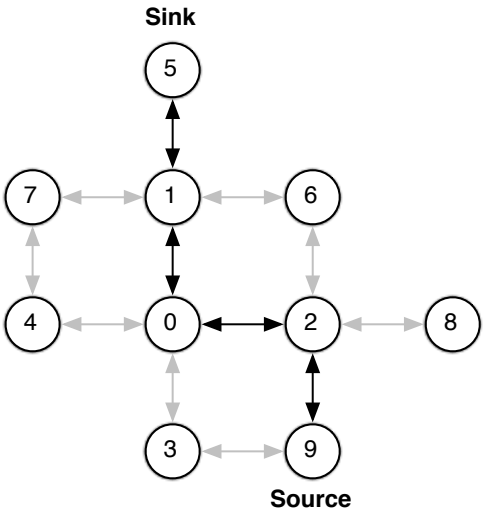


Figure 7.7: MinHop route from source 9 to sink 5

For each message sent from source to sink:

- the source transmits the message once and overhears it when node 2 forwards the message to node 0, and,
- node 2 receives the message from source 9, transmits the message to node 0 and overhears the message when it is forwarded from node 0 to node 1, and,

- node 0 receives the message from node 2, transmits the message to node 1 and overhears the message when it is forwarded from node 1 to sink 5, and,
- node 1 receives the message from node 0 and transmits the message to sink 5.

Thus, nodes 2 and 0 expend more energy than any other node and will be the first to expire, causing source 9 to become disconnected from the sink.

In the same network, the path used by SimplestShortest is via nodes 3, 0 and 1 to sink 5. Using this path, nodes 3 and 0 overhear, receive and transmit the message once. However, when those nodes expire, node 2 remains available for routing to the sink. Consequently, source 9 is able to remain connected for longer. Node reliance heuristics specifically avoid nodes that lie on multiple paths to the sink. Similarly, the use of EnergyAware causes paths to be dynamically changed based on the remaining energy at each node. However MinHop does not consider such approaches and simply routes based on the path with the least hops to the destination. Consequently, MinHop may select a path that is incompatible with the aim of encouraging (source, sink) connectivity.

In a larger network, there are more and better connected nodes. Consequently, there are more paths from sources to sinks and a recovery from the loss of an individual node is more likely. Additionally, it is more likely that the source and sink will be better connected. Consequently, there are fewer nodes whose loss would disconnect the source and sink. Thus, the MinHop heuristic is likely to be less problematic in larger networks.

In all the three graphs shown, the highest performing routing heuristic is EnergyAware, which is also the only heuristic in which the performance remains approximately constant as the proportion of source nodes increases. The performance of the EnergyAware heuristic is not unexpected, since it is the only one that instantly responds to changes in the remaining energy of nodes in the network. By extending the time until the first source node expires, the time for which all sources are connected to sink nodes is extended, regardless of what proportion of nodes are sources. Thus, EnergyAware may be expected to perform the best.

The improved performance of shortest path routing over lexicographic routing is discussed in Section 7.5.2. The three graphs seem to indicate that the performance of the node reliance heuristics and MinHop initially drops as the proportion of sources increases and then either levels out or increases slightly. A logical explanation for this behaviour is that the increased number of sources causes an increase in the contention for intermediate nodes.

Whereas EnergyAware can adapt to the increased usage of intermediate nodes, other heuristics cannot. The node reliance heuristic causes sources to use less relied upon nodes when routing from sources to sinks. However, there is no explicit coordination between the sources, and it is possible that every source simultaneously routes messages through a single node, causing it to quickly expire and disconnect part of the network.

A final observation is that the separation between the performances of each type of routing heuristic (EnergyAware, simplest node reliance, MinHop and lexicographic node reliance) increases as the size of the network increases. For example, the graph in Figure 7.4 represents networks with 10 nodes. However, the separation in performance (i.e. the lines) is much greater in the graph in Figure 7.6 representing networks with 40 nodes. This difference suggests that the relative performance of the MinHop and node reliance heuristics with respect to EnergyAware drops as the size of the network increases.

7.5.6 Total Data Transfer and Increasing Proportions of Sources

Another consideration is how the choice of routing heuristic affects the total data transferred from sources to sinks. Graphs representing networks of 10, 20 and 40 nodes in all network topologies were calculated.

The graphs representing networks of 10 and 20 nodes are somewhat indistinct and are difficult to interpret. The most intelligible of the graphs, representing 40 node networks, is shown in Figure 7.8.

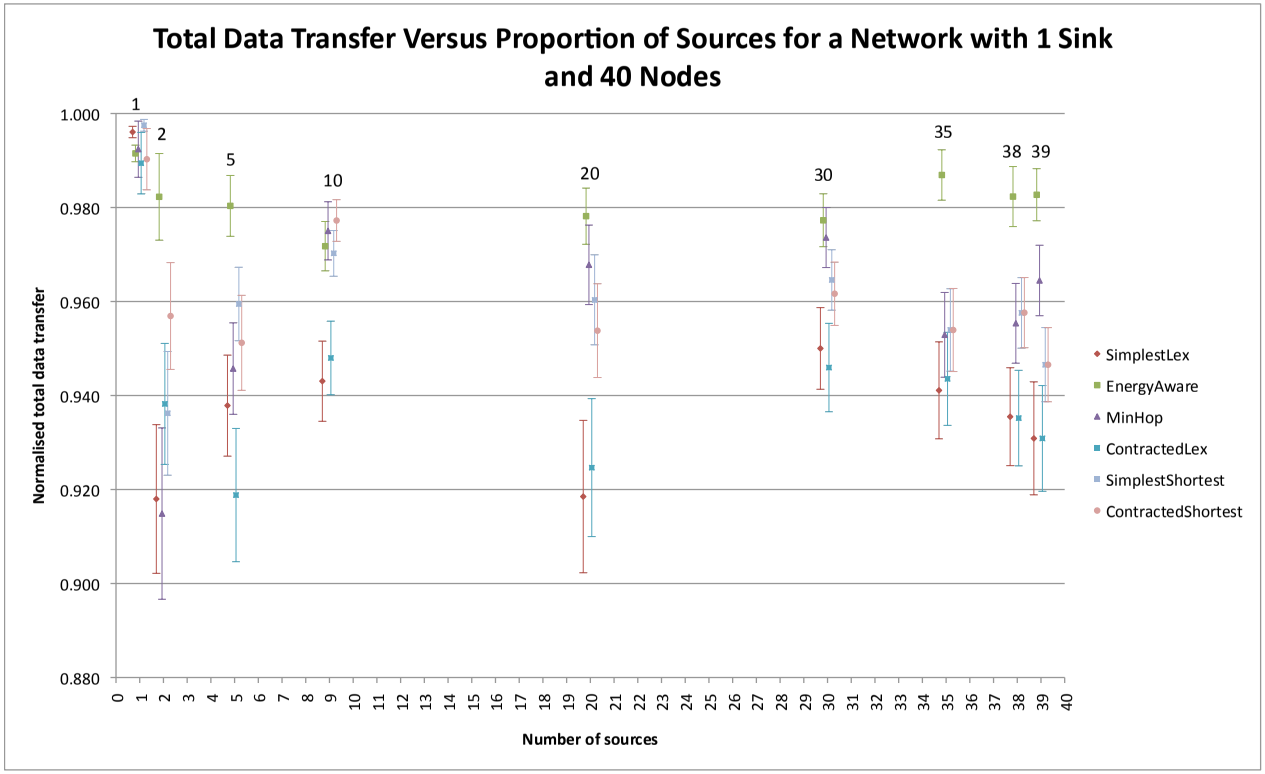


Figure 7.8: Total data transfer vs. proportion of sources for networks with 40 nodes and 1 sink. Error bars represent one standard deviation

As with the graph showing CWT versus proportion of sources for networks of 40 nodes and one sink, the performance of each heuristic is similar. EnergyAware performs the best. The heuristics that use shortest path (MinHop, SimplestShortest, ContractedShortest) come next and are followed by the heuristics that use lexicographic ordering (SimplestLex, ContractedLex). However, the difference in relative performance seems to be much smaller. For example, the total data transfer of EnergyAware, SimplestShortest, ContractedShortest are relative close whereas the CWT of EnergyAware is significantly higher than SimplestShortest and ContractedShortest.

As previously discussed, the total data transfer in the network is mostly dependent on the topology of the network. Each node is capable of transmitting a certain number of bytes before it expires and some nodes must be used, possibly in a particular combination, to route from a source to a sink. Consequently, the choice of routing heuristic cannot affect the maximum data that can be routed from sources to sinks and can only ensure that unnecessary expenditure of energy is minimised.

The graph indicates that the routing heuristics using lexicographic ordering perform worse than those relying on shortest path ordering due to the tendency to use more nodes. Since more nodes are used, the total energy expenditure in the network (and therefore the total capacity of the network) drops more quickly. This observation is supported by Table 7.3, which shows that lexicographic heuristics cause more transmissions (per byte received by the sink) than shortest path routing.

In contrast to its CWT measurement, MinHop performs nearly as well as the EnergyAware heuristic with respect to total data transferred. This observation seems reasonable. Reducing the number of nodes that a message must travel through decreases the total energy expended by the network, allowing more messages to be sent. As already discussed, however, this technique is prone to reducing the connectivity of sources to sinks, even though the total transfer capacity remains the same.

The performance of the EnergyAware heuristic may be highest due to its ability to coordinate path selection among the different sources. Consider, for example, the use of a path P by source A and path Q by source B. If nodes on P and Q are neighbours, they will overhear each other's messages, increasing energy expenditure and decreasing capacity. The EnergyAware heuristic is the only one that will actively avoid the use of nodes whose energy expenditure is high, thus encouraging the use of alternative paths if possible. Con-

sequently, the energy expenditure may drop, causing the capacity to increase. Note that there is no easy means to verify this theory, since the number of overheard transmissions does not necessarily decrease. Transmissions may be overheard on nodes that are not relied upon in routing.

The final observation regarding the effect on total data transfer of increasing the proportion of source nodes concerns Figure 7.9, which represents 20 node networks.

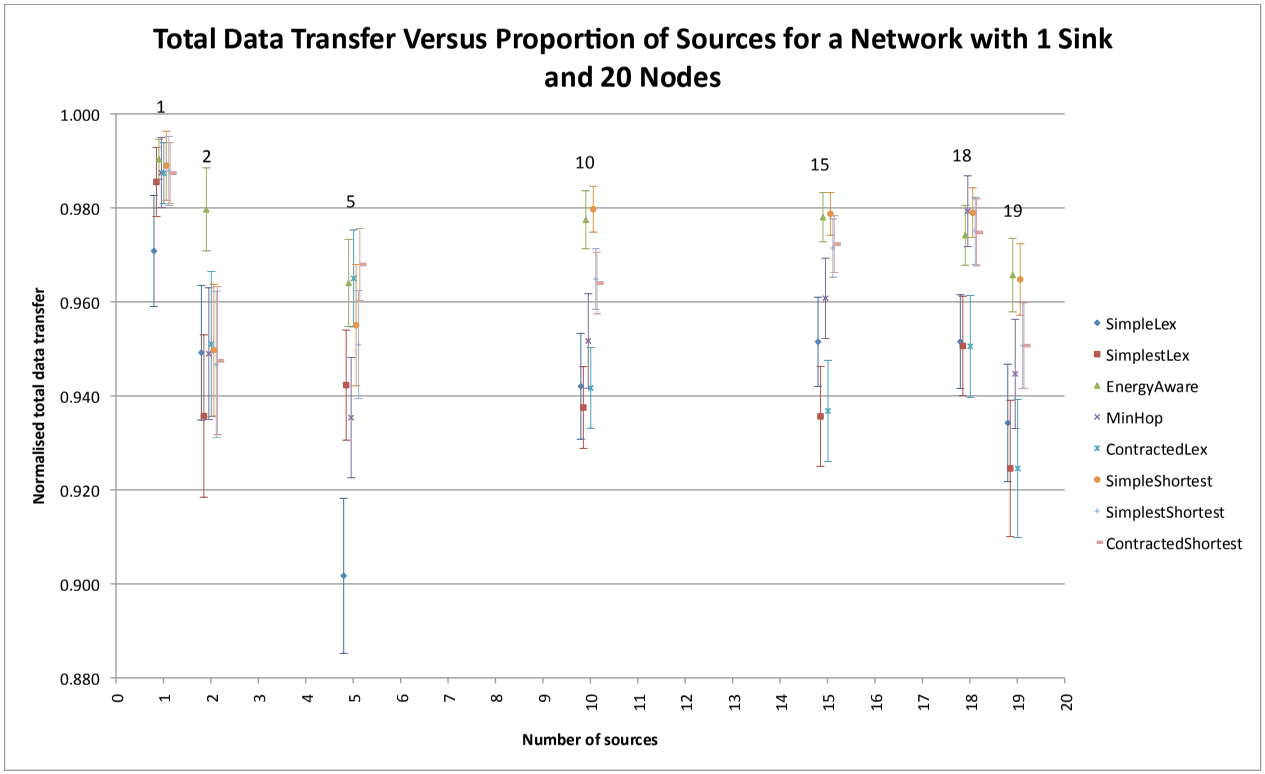


Figure 7.9: Total data transfer vs. proportion of sources for networks with 20 nodes and 1 sink. Error bars represent one standard deviation

The graph is significantly more difficult to understand than that shown for the 40 node networks in Figure 7.9. However, it is worth noting that as with CWT, the performance of MinHop improves in larger networks. The two total data transfer graphs seem to suggest that MinHop surpasses shortest path node reliance when the network has somewhere between 20 and 40 sources. As before, the simplest explanation for this behaviour is that larger networks provide more and better connected nodes, thus reducing the number of nodes whose loss would disconnect sources from sinks.

7.5.7 Effect of Increasing Numbers of Sources

The effects of increasing the number of sources in the network are now considered. In contrast to the previous section where the number of nodes is kept constant and the proportion of those nodes acting as sources are increased, this section considers that sources are added to the network - thus increasing both the number of nodes and the number of sources simultaneously. The number of sinks remains constant at one. Such a network configuration, in which the network contains a single sink and all other nodes act as sources is shown to be the most prevalent in the WSN deployments studied in Section 2.4.

Figures 7.10 and 7.11 show the effect on total data transfer and CWT respectively as a result of increasing source numbers. The performance of each contracted heuristic is exactly equal to the corresponding simplest path heuristic. The reason for the equality is the restriction (discussed in Section 6.1.4) that source and sink nodes may not be contracted from the network. In each of the networks represented by the graphs, every node is either a source or a sink. Consequently, no contraction can take place and so the contraction heuristic becomes the simplest path heuristic.

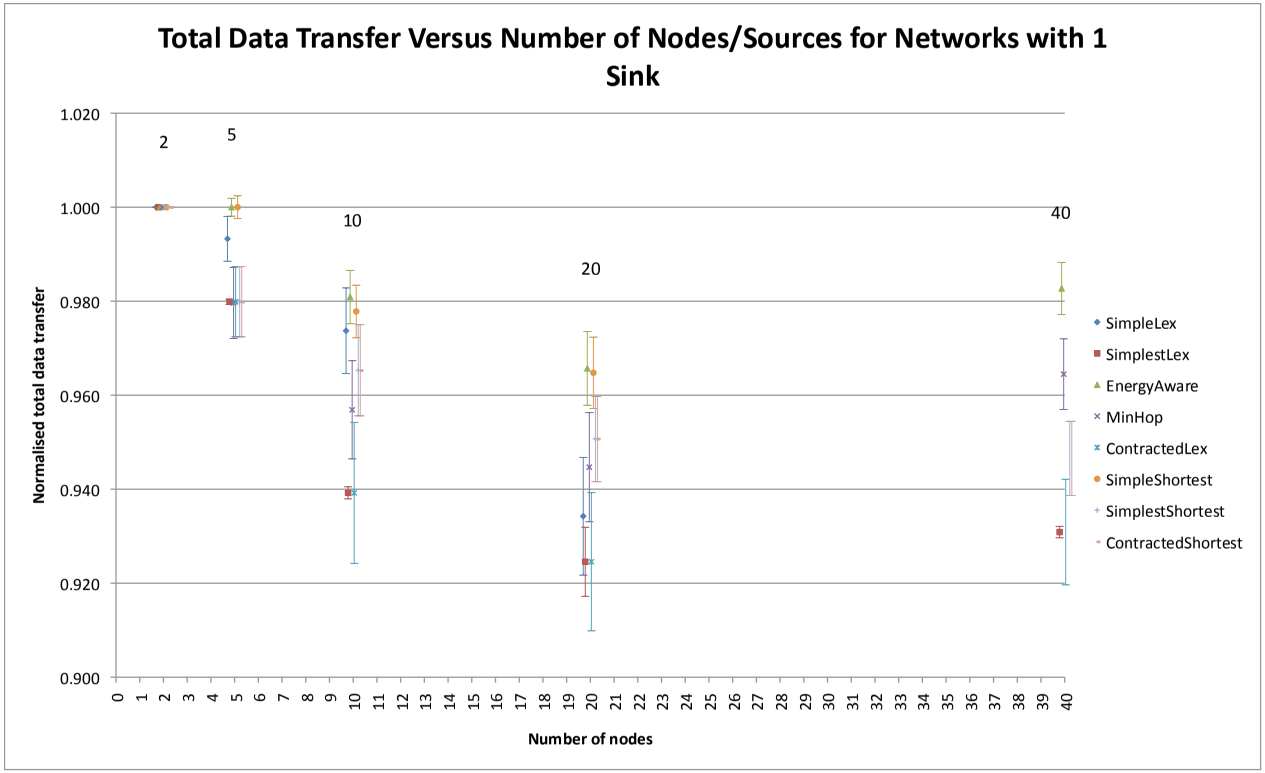


Figure 7.10: Total data transfer vs. number of nodes/sources for networks with 1 sink. Error bars represent one standard deviation

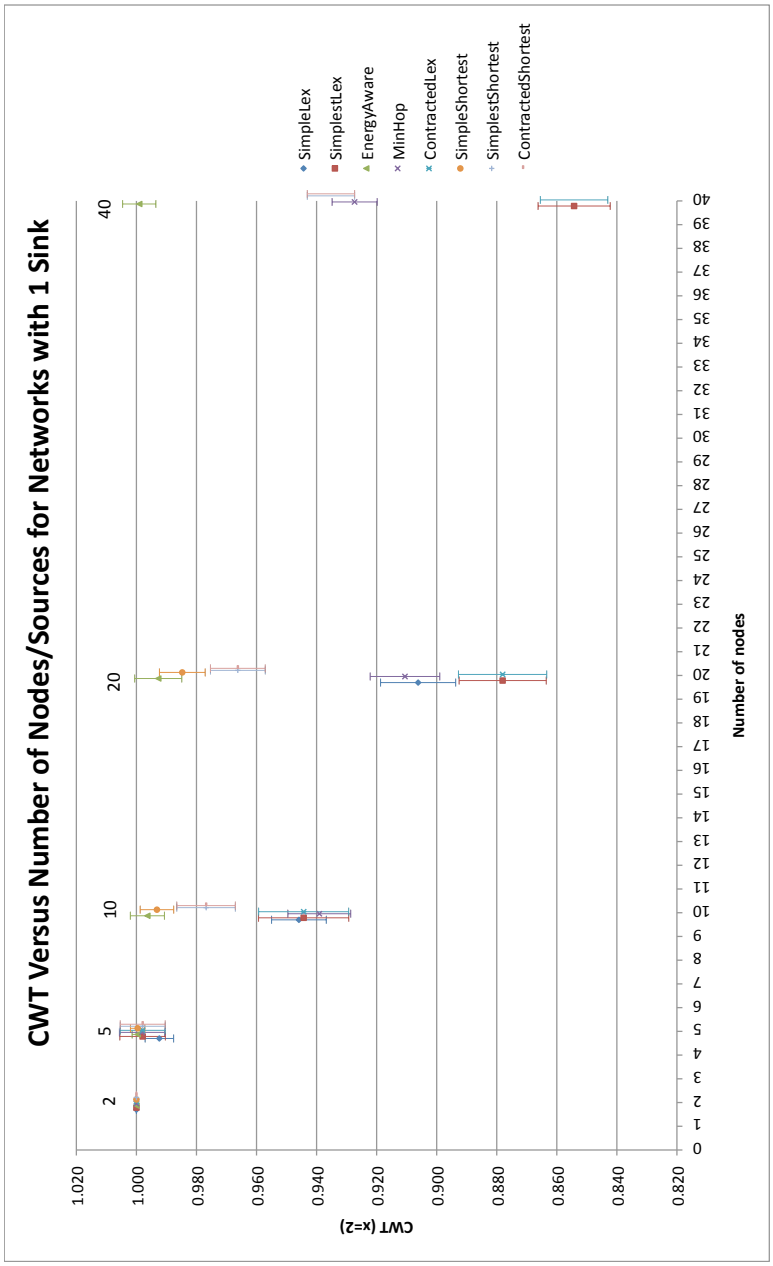


Figure 7.11: CWT vs. number of nodes/sources for networks with 1 sink. Error bars represent one standard deviation

The total data transfer graph indicates that all routing heuristics decline in performance until the network reaches some point between 20 and 40 sources. At that point, the performance of the heuristic levels off (in the case of the shortest path node reliance heuristic) or increases (in the case of the other heuristics). Since the measurements are normalised and the normalised score of all heuristics drops, the performance becomes more unpredictable.

Since there appears to be no increase in the number of overheard or received messages, the only explanation for the loss of performance is that node loss is occurring in such a way that the capacity of the network is reduced. A similar explanation is suggested in Section 7.5.6 for the apparent improvement in MinHop (with respect to total data transfer) between networks containing 20 nodes and 40 nodes. It seems reasonable that all routing heuristics would be prone to the same problem. However, since the EnergyAware and node reliance heuristics actively seek to avoid using contentious nodes, MinHop would be the worst affected.

It is worth noting that in the same graph, the performance of MinHop increases more than any other heuristic between the use of 20 nodes and 40 nodes. Furthermore, the graph shows that the total data transfer achieved by MinHop exceeds that of SimplestShortest and ContractedShortest at approximately 25 nodes (24 sources), which would explain the apparent improvement discussed in Section 7.5.6. A further question remains as to why SimplestShortest and ContractedShortest continue to perform more poorly as the network size is increased beyond 20 nodes. There are two possible explanations for this behaviour:

- The improved performance of the other heuristics causes the relative performance of the node reliance shortest path heuristic to drop.
- There may be an uncertainty with respect to the plotted point, particularly since the drop in CWT between 20 and 40 nodes is small.

A final remark regarding the total data transfer graph is that it supports the findings discussed earlier with respect to the relative performance of each routing heuristic. Specifically, EnergyAware is the highest performing heuristic, followed by SimplestShortest, ContractedShortest and MinHop and lastly SimplestLex and ContractedLex.

In the CWT graph, the performance of every heuristic decreases until the network contains at least 20 sources. Such a result is not unexpected given that the total data transfer is known

to decrease during this time and since total data transfer affects the CWT metric. After 20 nodes, the performance of both EnergyAware and MinHop increase. Again, this increase in performance could be due to the increased total data transfer experienced. The cause of the decrease in performance of the two node reliance heuristics is not immediately obvious. The number of overheard messages does not increase, nor does the number of transmissions per received message. Once again, the only remaining explanation is that important nodes expire quickly and an examination of the time until first node death supports this theory.

Figure 7.12 plots average normalised CWT against average normalised total data transfer for networks with 40 nodes, 39 sources and 1 sink and networks with 20 nodes, 19 sources and 1 sink. There appears to be a correlation between CWT and the time at which the first node expires, suggesting that in these networks, CWT is heavily affected by the time at which the first node expires.

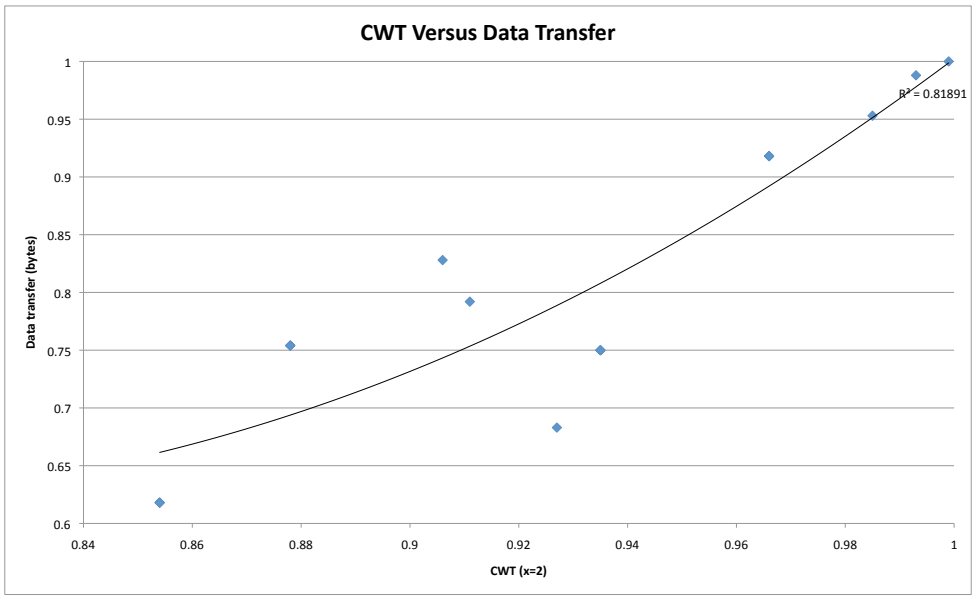


Figure 7.12: CWT vs. total data transfer for 40 node/39 source/1 sink networks and 20 node/19 source/1 sink networks

However, the graph does not indicate the importance of the first node to expire. Furthermore, the only means to measure the effect of the disconnection of a node would be the

CWT metric itself. An unanswered question is what causes the important nodes to become disconnected, since the node reliance heuristics are specifically designed to prevent such an event from occurring. The most likely explanation as to the cause of those nodes expiring is the fact that the node reliance heuristics take no measures to avoid messages being overheard by important nodes. Such a theory cannot be confirmed by an analysis of the number of overheard messages, since that statistic provides no information regarding the distribution of overheard messages. For example, there is a significant difference between 1000 messages being overheard by a single node or 1000 nodes.

7.6 Summary

The EnergyAware heuristic is consistently shown to be the most efficient of the routing heuristics with respect to both CWT and total data transfer. However, a routing protocol based on this heuristic would expend additional energy in gathering the available energy at each node, which is likely to be continually changing. Consequently such a routing protocol may not remain efficient.

SimpleShortest is generally the second most efficient of the heuristics. However, the computation time required to calculate the node reliance values is prohibitive for a WSN of even 20 nodes. Consequently, it is not examined in any further detail in this thesis. However, if a more efficient means of calculating the node reliance values based on simple paths could be found, the heuristic might be worth re-examining.

The third most efficient routing heuristics (in terms of both CWT and total data transfer) are SimplestShortest and ContractedShortest. The performance of each of these heuristics is the same. Unlike EnergyAware, these heuristics only require a view of the connectivity of the network and so require less periodic updates.

The experiment suggests that at some point between 20 and 40 nodes, the relative efficiency of routing heuristics, particularly MinHop, changes. For example, Figure 7.10 indicates that when the network size reaches 25 nodes (24 sources), MinHop surpasses SimplestShortest and ContractedShortest in terms of total data transfer. It is hypothesised that in smaller networks, MinHop is more prone to expending unnecessary energy on critical nodes, since it does not take steps to avoid using such nodes.

This chapter has analysed the effectiveness of routing heuristics in keeping sources connected to sinks for as long as possible. In a WSN, the operation of a routing heuristic requires the gathering of data, which may drain the batteries of nodes and thus reduce the time for which sources and sinks can remain connected. Chapter 8 discusses how the node reliance heuristics may be modified to form routing protocols that additionally gather the required data. Chapters 9 and 10 analyse the effectiveness of these and other routing protocols in the literature.

Chapter 8

Node Reliance Routing Protocols

Chapter 6 presents node reliance heuristics, which calculate the usefulness of nodes in routing data from sources to sinks. In this chapter, the heuristics are transformed into routing protocols by adding a discovery task. As discussed in Section 3.1, the discovery task gathers the set of available paths in the network. As stated in Sections 6.1.2.2, 6.1.3.2 and 6.1.4.2, it is necessary to enumerate all paths in the network in order to determine the node reliance values. Consequently, the routing protocols must gather the entire set of paths.

The node reliance protocols make no assumptions about the quality of communication. Specifically, messages may be lost during transmission, communication between pairs of nodes may not be bidirectional and transmissions may take an unknown and possibly variable period of time. This is consistent with the assumptions made in Section 2.5 regarding wireless communication.

Section 8.1 lists the requirements of the node reliance routing protocols. Three routing protocols are discussed in this Chapter. They are referred to as *enumeration*, *global knowledge* (GK) and *partial data*.

The *enumeration* protocol is discussed in Section 8.2 and is based on the enumeration search method, discussed in Section 3.1.3.4. In the routing protocol, one message is sent along each simple or simplest path. Messages contain the sequence of nodes through which they have travelled and these paths are collected at the sink nodes. The sink nodes flood these paths through the network and thus each source gains a view of all (source, sink)

paths in the entire network.

The *global knowledge* protocol is discussed in Section 8.3 and is based on the global knowledge search method, discussed in Section 3.1.3.3. In the protocol, each node floods its set of immediate neighbours to every other node in the network. Consequently, every node learns the entire network topology.

Partial data is a hybrid routing protocol in which topology data is gathered at intermediate nodes and incrementally collected by the sink nodes. It is discussed in Section 8.4.

Finally, the chapter is summarised in Section 8.5.

8.1 Assumptions

The routing protocols described in this chapter operate with the following assumptions in addition to those discussed in Section 2.5:

1. All nodes are activated simultaneously.
2. There are a finite number of nodes.
3. Every node has a unique ID.
4. Any node may be required to participate in routing.

These assumptions are justified in the remainder of this section.

If nodes are not activated simultaneously, then the discovery task may begin before some nodes have been activated, leading to sub-optimal paths being used for routing. It is possible to simulate all nodes being activated simultaneously by requiring the user to inject a *begin discovery* signal through the network after all nodes have been switched on.

The number of nodes in the network must be finite, since the routing protocols are designed for real network implementations, which cannot operate with an infinite number of nodes.

Each node is required to have a unique ID so that it is possible to differentiate between different nodes. Unique IDs are particularly important to node reliance routing so that high

reliance nodes may be avoided where possible. It is not possible to avoid particular nodes unless each node can be uniquely identified. Unique node IDs may be uploaded to nodes during software installation, negotiated by the network or else based on unique hardware IDs such as the MAC address of a ZigBee radio which is a unique 64-bit identifier [51].

Finally, any node may be required to participate in routing, in order to maximise the probability that sources can communicate to sink nodes. Consequently, no node is considered to be exempt from routing on account of its hardware or configuration, such as the presence of a special sensor or because it is a source. It may be the case that it is undesirable for a particular node to be used in routing. However, such a situation is determined by the node reliance heuristic.

8.2 Enumeration

The enumeration routing protocol consists of three phases, which are the *exploration phase*, the *reply phase* and the *enforcement and data phase*, in order:

The exploration phase collects all available paths from sources to sinks at the sink nodes. The collected paths are dependent on the heuristic used.

In the reply phase, sinks flood the set of paths through the network. The paths are received by source nodes, which combine the received messages and thus acquire a set of all (source, sink) paths in the network.

In the enforcement and data phase, a path is selected by each source and nodes along that path are informed of their next neighbour in forwarding data to a sink.

The routing protocol has been presented as three phases for the sake of improving the clarity of the routing protocol's behaviour. The first and second phases correspond to the discovery task, discussed in Section 3.1. The costing task, discussed in Section 3.2 takes place between the second and third phases. This routing protocol has no modules from the selection task; it simply uses the cheapest path.

During this process, data may be lost due to the expiration of nodes or the repeated loss of data on nodes with poor connectivity. If a sink stops receiving data for a period of time then the path is declared to have failed. The constant *data-loss-tolerance* refers to the number

of seconds that may pass without receiving data from some source node before the sink determines that the path in use by that source has failed. If a path has failed, the sink node resets all routing data in the entire network and the routing protocol begins again from the exploration phase. A reset is necessary, since it may not be possible to easily determine which nodes have expired and how reliance values may have changed as a result. In order to ensure that old routing data is not kept, a network-wide *sequence number* is incremented by the sink that requests the reset. The sequence number is included as part of every message sent in the network. Nodes with old sequence numbers cannot be used in routing and cannot participate in route discovery.

The three phases of the routing protocol are discussed in the following sections.

8.2.1 Exploration Phase

During the exploration phase, Exploration messages (discussed below) are flooded through the network from sources to sinks. Each Exploration message contains the sequence of nodes through which that message has travelled. Each Exploration message is either collected by a sink, or is discarded by intermediate nodes. A receiving node discards a message if the path is not one required by the heuristic. For example, a receiving node would discard a message containing a non-simplest path if the simplest paths heuristic was in use.

A source will assume that its attempt to form a path has failed if it does not receive a response to its Exploration message after some period of time. The exploration phase may fail if all Exploration messages are lost in the network. The constant *enum-path-formation-time* refers to the number of seconds that the source will wait for a reply before it concludes that the process has failed. If a reply is not received, then the source begins its exploration phase again. However, the number of times that the exploration can take place is limited to ensure that the batteries of nodes are not being continually drained trying to unsuccessfully form paths.

8.2.1.1 Messages

During the exploration phase, nodes exchange only one type of message:

Exploration messages are used to discover paths and one such message is sent along each path required by the routing heuristic, e.g. simplest paths. An Exploration message has three fields:

- The network-wide sequence number.
- A rerequest sequence number.
- The sequence of nodes through which the message instance has travelled.

The network-wide sequence number is incremented whenever a node failure occurs during routing of data from sources to sinks. It is used to ensure that only nodes with fresh routing data can be used to form routes or to route data. The second field is a rerequest number, which allows sources to rerequest routing data that is stored at sink nodes, if they do not receive a response to their Exploration messages.

8.2.1.2 Data Structures

The exploration phase uses the following data structures:

KnownSources refers to the set of sources that have been discovered in the network. The set is built up at the sink nodes on receipt of each Exploration message and is sent out as part of an ExplorationReply message.

KnownSinks refers to the set of sinks that have been discovered in the network. The set is built up at the sink nodes and is sent out as part of the ExplorationReply.

RoutingMatrix is an n-by-n boolean *adjacency matrix* that represents known connections between the n nodes in the network. It is formed at sink nodes from the paths contained in the Exploration messages and is flooded out to the source nodes in the ExplorationReply messages.

EnumerationMatrix is necessary if all simplest paths are being enumerated. This matrix stores the set of paths that have been learned by a node during the exploration phase. It is used to determine whether incoming paths may be rejected due to being a subpath of a previously forwarded path, or whether the path must be forwarded in an outgoing Exploration message.

8.2.1.3 Pseudo Code

Below is the pseudo code for the exploration phase of the enumeration algorithm. In this code, it is assumed that all simplest paths are being enumerated. The same code runs on each node. As well as the data structures and messages shown above, the following functions are used within the code:

id() returns the unique identifier for a node.

role() returns the role of a node (i.e. whether it is a source or a sink).

delay(f, t) executes function f after t seconds. Calling delay on a function that is already delayed changes the time until that function is executed.

delayed(f) returns a boolean indicating whether function f is due to be executed at some future time (via the *delay(f, t)* function).

```
1  On_Startup() {
2     if (role() == source) {
3        delay(routingRequest, RREQ-DELAY-PER-NODE * id())
4        delay(checkPathExists, ENUM-PATH-FORMATION-TIME)
5     }
6  }
7
8  void routingRequest() {
9     sendExplorationMsg(SeqNum, ReqNum, id())
10 }
11
12 void checkPathExists() {
13    if (RoutingMatrix.isEmpty()
14    && ReqNum < ENUM-RREQ-RESEND-ATTEMPTS) {
15       ReqNum++
16       sendExplorationMsg(SeqNum, ReqNum, id())
17       delay(checkPathExists, ENUM-PATH-FORMATION-TIME)
18    }
19 }
20
21 On_Receive_ExplorationMsg(int seq, int req, Node[] path) {
```

```
22    if (!checkSeqNum(seq))
23       return
24
25    Node[] newPath = path.add(id())
26
27    if (path.contains(id())) {
28       EnumerationMatrix.add(newPath)
29       return
30    }
31
32    if (role() == sink) {
33       RoutingMatrix.add(newPath)
34       KnownSources.add(newPath.front())
35       KnownSinks.add(id())
36
37       if (req > ReqNum)
38          ReqNum = req
39
40       if (!delayed(outputRoutingMatrix)
41          delay(outputRoutingMatrix, ENUM-EXPLORE-REPLY-DELAY)
42    } else {
43       EnumerationMatrix.add(newPath)
44
45       if (!EnumerationMatrix.containsSubpathOf(path))
46          sendExplorationMsg(seq, req, newPath)
47    }
48 }
49
50 boolean checkSeqNum(int seq) {
51    if (seq > SeqNum) {
52       SeqNum = seq
53       resetRoutingData()
54       sendErrorMsg(SeqNum)
55       if (role() == source && !delayed(routingRequest))
56          delay(routingRequest, ENUM-PATH-RETRY-DELAY)
57    } else if (seq < SeqNum)
58       return false
```

```
59
60      return true
61  }
```

On_Startup() deals with the startup of each node (which is assumed to be simultaneous). Exploration messages are staggered. Each source calls *routingRequest()*, which sends the Exploration message, after waiting for a period of time based on the node's ID. Staggering Exploration messages reduces the number of message collisions that occur. After scheduling an Exploration message to be sent, the function also starts a timer to check that a valid path has been formed after *enum-path-formation-time* seconds.

routingRequest() sends an Exploration message.

checkPathExists() checks whether a (source, sink) path exists. It is called after the node starts up and the delay determined by the constant *enum-path-formation-time*. If no path exists, the route request number (ReqNum) is incremented and an Exploration message is sent again. If no (source, sink) path is formed after *enum-rreq-resend-attempts* attempts, the node stops. The limit is to ensure that intermediate nodes do not continuously expend energy trying to form routes for a source to a possibly unreachable sink.

On_Receive_ExplorationMsg() parses Exploration Messages. To begin with, the sequence number of the message is checked via *checkSeqNum()*. If the message's sequence number is older than that stored at the node, the message is discarded. Next, the path is checked to ensure it is simple, i.e. the node's ID is not already on the path. If the path is non-simple, it is added to the EnumerationMatrix to aid in the detection of non-simplest paths and then discarded. If the node is a sink, it stores the path in its RoutingMatrix and adds the source and sink to the KnownSources and KnownSinks set. The sink also updates the local ReqNum value if it is smaller than that in the incoming Exploration message. A timer is started to begin the reply phase via the *outputRoutingMatrix()* function after *enum-explore-reply-delay* seconds. If the node is not a sink, the path is added to the EnumerationMatrix and forwarded if it is simplest.

checkSeqNum() checks the validity of sequence numbers and is called whenever a message arrives. The function returns a boolean indicating whether the incoming message is valid, i.e. whether it should be acted upon. An old sequence number should be discarded and causes the function to return a value of false. Such messages should be ignored since they

Constant	Value
ENUM-PATH-FORMATION-TIME	30
ENUM-RREQ-RESEND-ATTEMPTS	1
ENUM-EXPLORE-REPLY-DELAY	5
ENUM-ROUTE-REPLY-COLLECTION-TIME	5
RREQ-DELAY-PER-NODE	1
ENUM-PATH-RETRY-DELAY	30

Table 8.1: Constant values for the enumeration algorithm examples

potentially carry obsolete information. New sequence numbers cause the node's sequence number to be updated and all of its routing data to be reset. If the node is a source and is not currently waiting to initiate a path discovery process, then path rediscovery will be scheduled for *enum-path-retry-delay* seconds time. The node then forwards an Error message containing the new sequence number throughout the network. If the sequence number in the message matches that on the node, then a value of true is returned, indicating that the message should be retained.

8.2.1.4 Example

This section presents an example of the exploration phase of the enumeration algorithm. The example network contains two sources (nodes 1 and 2) and two sinks (nodes 6 and 7). A directed arrow from a node A to a node B indicates that messages from node A can be received by node B. In this example, all but two messages are reliably delivered, which are shown in red on the diagrams. The failed messages are used to demonstrate the algorithm's resilience to possible transmission loss. Table 8.1 shows the values of various constants for the enumeration algorithm.

It is assumed that the nodes all simultaneously activate at time 0. Source nodes 1 and 2 delay for a number of seconds equal to *rreq-delay-per-node* (1) multiplied by their node ID before beginning the first phase of the algorithm. In practice, therefore, each node begins phase one at a time equal to its ID.

Thus, at time unit 1, source 1 begins the first phase by broadcasting an Exploration message, which is received by nodes 2 and 3 as shown in Figure 8.1.

At time 2, a number of actions take place as shown in Figure 8.2. Firstly, node 3 forwards the Exploration message originating from source 1, which is received by sink 7. The message is also received back by source 1, which ignores the message due to its non-simplicity (source 1 already appears on the path). Source 2 also forwards the Exploration message originating from source 1. It is received by node 4 and sink 6. It is also received by node 3, which rejects the message due to being non-simplest. Node 3 has already received the path 1:3 (as shown in matrix 1) which is a subpath of the path 1:2:3. Finally, source 2 sends out its own Exploration message, which is received by nodes 3, 4 and 6.

At time 3, node 3 broadcasts the Exploration message from source 2, which is received by node 1 and sink 7. Node 4 forwards Exploration messages from sources 1 and 2, both of which are received by sink 7. The network state at time 3 is shown in Figure 8.3.

Finally at time 4, source 1 forwards the Exploration message originating from source 2. The message is received by nodes 2 and 3. However, both of these nodes reject the message since the paths 2:3:1:2 and 2:3:1:3 are not simple. The network at time 4 is shown in Figure 8.4.

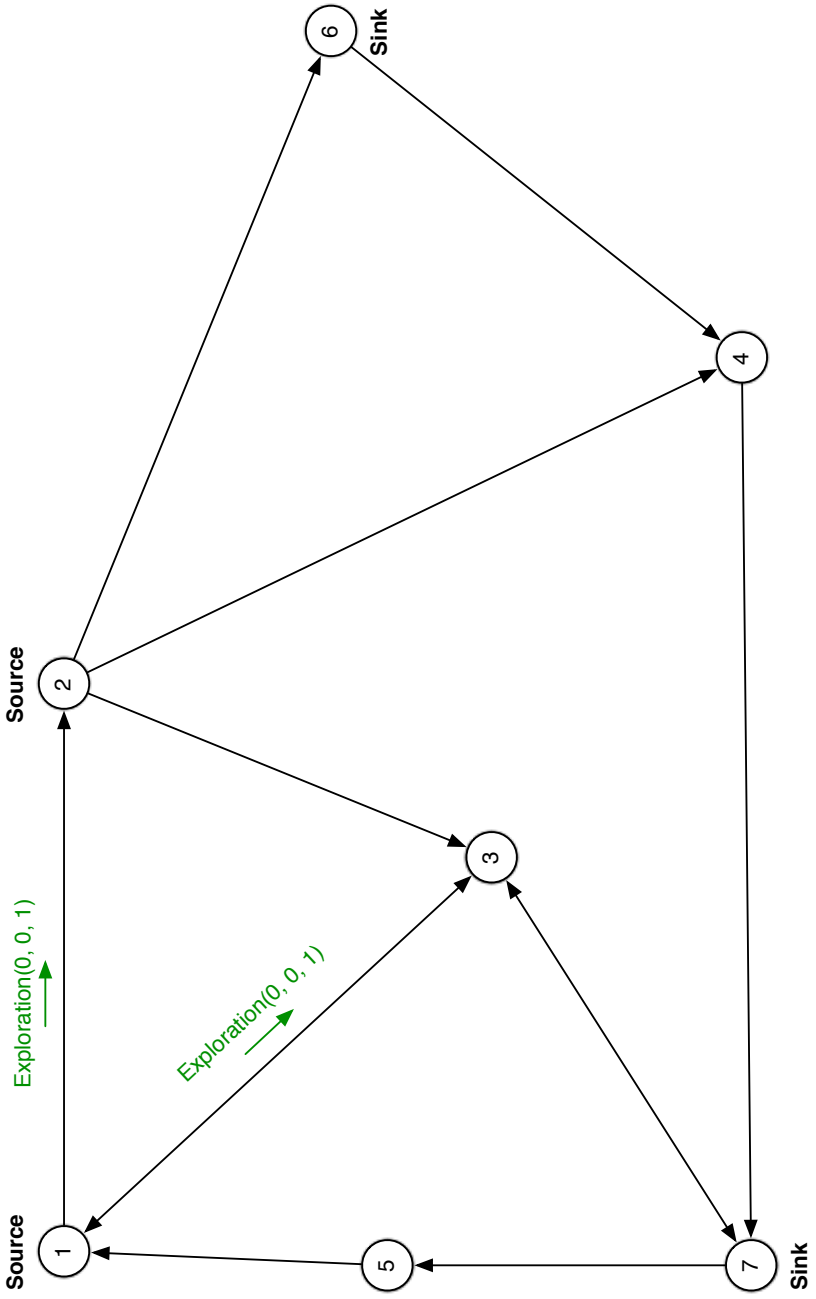


Figure 8.1: State of the network at time 1

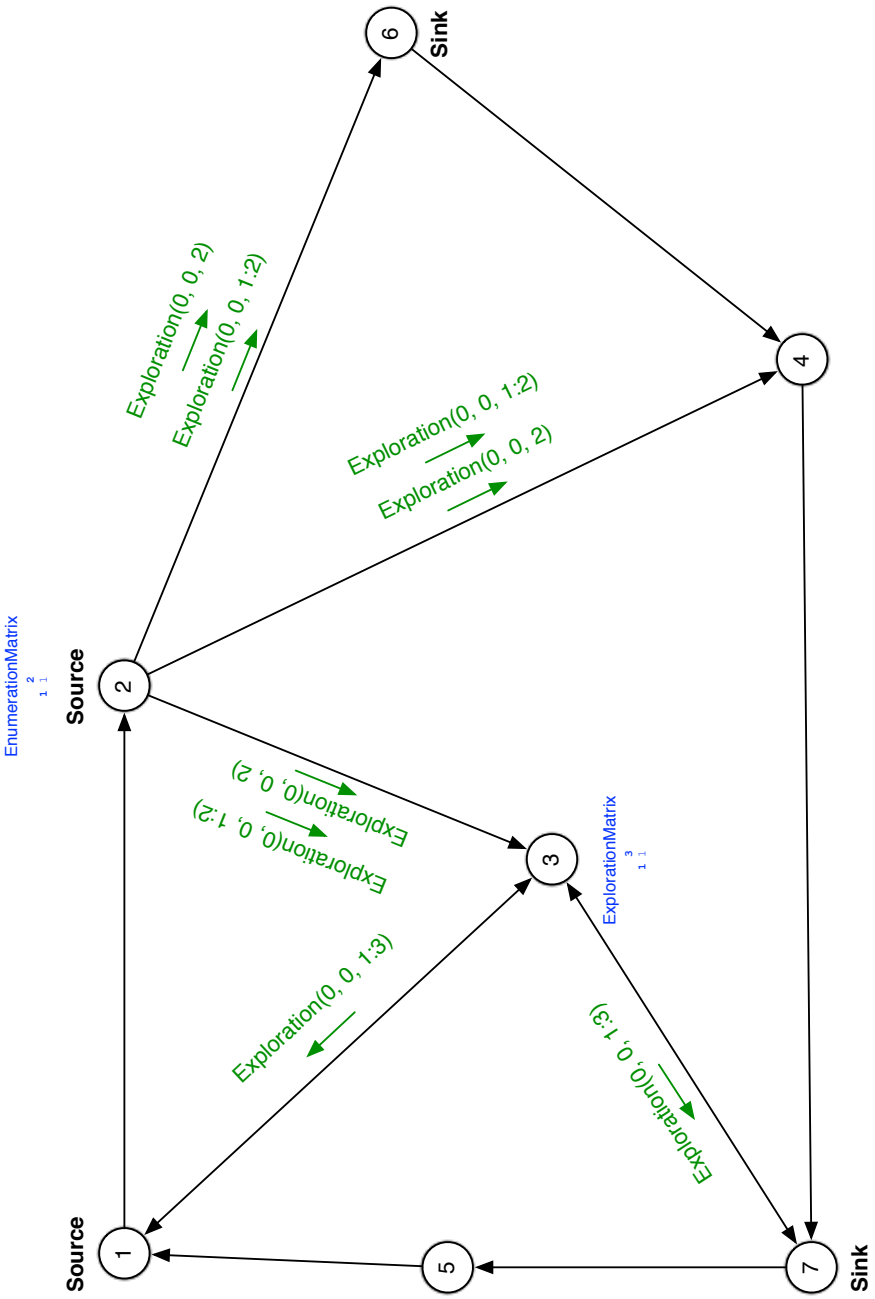


Figure 8.2: State of the network at time 2

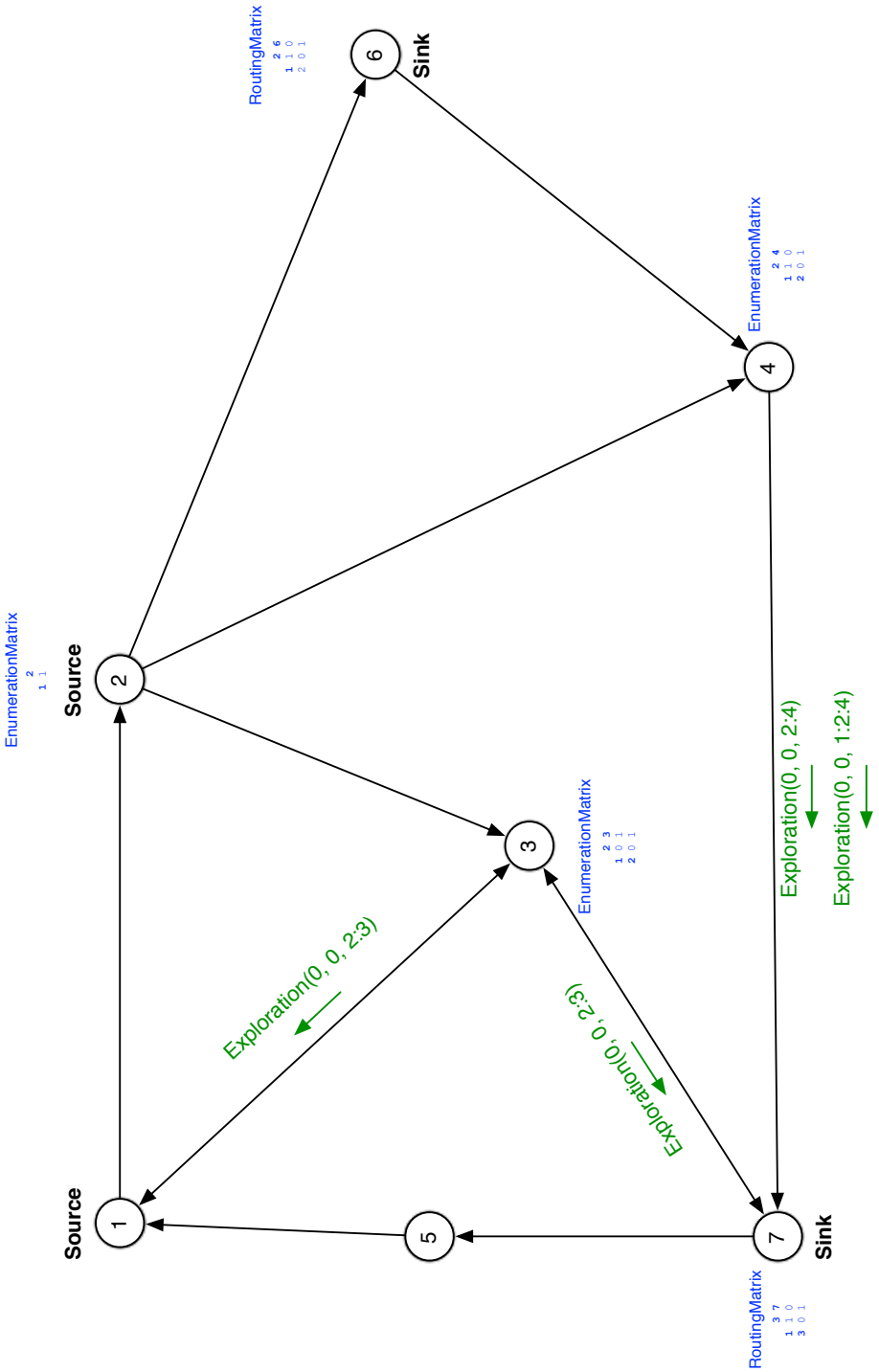


Figure 8.3: State of the network at time 3

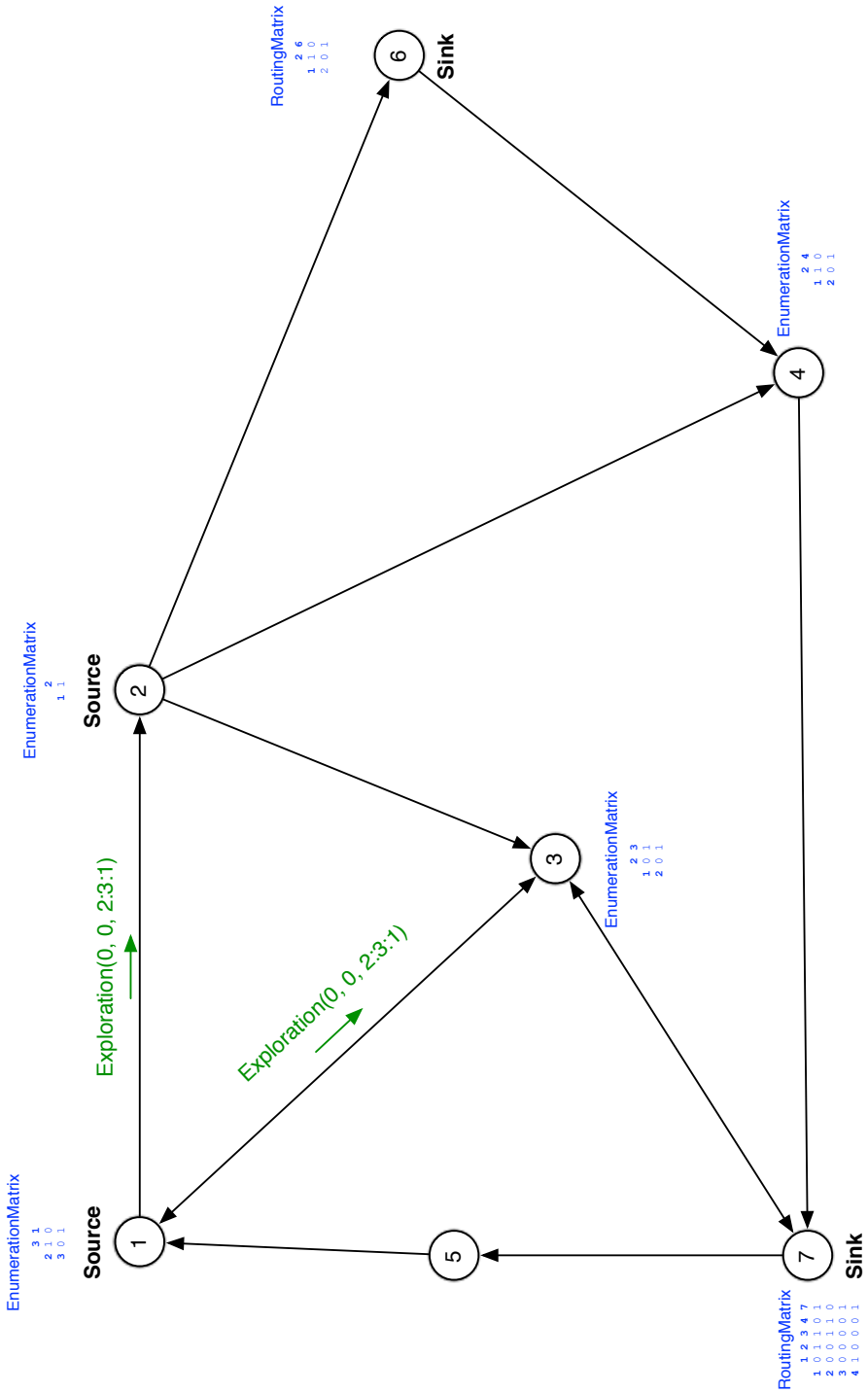


Figure 8.4: State of the network at time 4

8.2.2 Reply Phase

The second phase in the protocol is the reply phase. A sink enters this phase *enum-explore-reply-delay* seconds after receiving a path in the exploration phase. In this phase, the paths that were received by the sinks during the exploration phase are flooded through the network and eventually received by source nodes.

8.2.2.1 Messages

During the reply phase, nodes exchange only one type of message:

ExplorationReply messages are used to send collected network topology data back to the source nodes. The message consists of six fields:

- The network-wide sequence number.
- A rerequest sequence number.
- The sink from which the ExplorationReply originates.
- The *adjacencyMatrix* at the sink that the message originates from.
- The set of known sources at the sink that the message originates from.
- The set of known sinks at the sink that the message originates from.

For convenience, this description assumes that ExplorationReply messages are sent in one part. In practice, the probability of successful communication is improved by splitting a large ExplorationReply message into several parts. A further advantage is that if further Exploration messages are received after the ExplorationReply message has been sent, a node need only send the new topology information that it has learned.

In order to limit the effect of lost ExplorationReply messages, when a node rebroadcasts an ExplorationReply message, it includes all routing data stored at that node. Thus, even if ExplorationReply messages are lost in the network, the data that they contained may be transmitted in a different ExplorationReply message originating from a different sink.

8.2.2.2 Data Structures

The reply phase uses the following data structures:

KnownReplies refers to the set of sinks from which ExplorationReply messages have been received. An ExplorationReply message is only forwarded if the reply is not listed in the KnownReplies data structure. Thus, the reply phase will terminate once every node has received the ExplorationReply messages once. In practice, KnownReplies must track each part of each ExplorationReply message that has been received. For the purposes of this description, it is assumed that ExplorationReply messages are sent in one discrete packet and contain all the network data, i.e. no new network data is learned after the ExplorationReply message is sent.

RoutingMatrix is an n-by-n boolean matrix that represents known connections between the n nodes in the network. It is formed at sink nodes from the paths contained in the Exploration messages and is flooded out to the source nodes in the ExplorationReply messages.

LastRoutingMatrix represents the last RoutingMatrix transmitted by a sink node. It allows sink nodes to send incremental changes to the RoutingMatrix if more Exploration messages are received after the ExplorationReply is sent. Again, this description assumes that such an event does not occur.

ReqNum refers to the number of times the RoutingMatrix has been rerequested by sources in the network, since the last network-wide reset. ReqNum is initially 0.

8.2.2.3 Pseudo code

Below is the pseudo code for the reply phase of the enumeration algorithm. As well as the data structures and messages shown above, the following functions are used within the code:

id() returns the unique identifier for a node.

role() returns the role of a node (i.e. whether it is a source or a sink).

delay(f, t) executes function f after t seconds. Calling delay on a function that is already delayed changes the time until that function is executed.

delayed(f) returns a boolean indicating whether function f is due to be executed at some future time (via the *delay(f, t)* function).

```
1  void outputRoutingMatrix() {
2     sendExplorationReply(SeqNum, ReqNum, id(), RoutingMatrix,
3     KnownSources, KnownSinks)
4  }
5
6  boolean checkSeqNum(int seq) {
7     if (seq > SeqNum) {
8        SeqNum = seq
9        resetRoutingData()
10       sendErrorMsg(SeqNum)
11       if (role() == source && !delayed(routingRequest))
12          delay(routingRequest, ENUM-PATH-RETRY-DELAY)
13    } else if (seq < SeqNum)
14       return false
15
16    return true
17 }
18
19 On_Receive_ExplorationReplyMsg(int seq, int req, Node sink,
20 Matrix matrix, Set sources, Set sinks) {
21    if (!checkSeqNum(seq))
22       return
23
24    RoutingMatrix.merge(matrix)
25    KnownSources.merge(sources)
26    KnownSinks.merge(sinks)
27
28    if (ReqNum > req)
29       return
30    else if (ReqNum < req) {
31       ReqNum = req
32       KnownReplies.erase()
33    }
34
```

```
35     if (KnownReplies.contains(sink))
36        return
37
38     sendExplorationReply(seq, req, sink, RoutingMatrix,
39     KnownSources, KnownSinks)
40
41     if (role() == source)
42        delay(formPath, ENUM-ROUTE-REPLY-COLLECTION-TIME)
43  }
```

outputRoutingMatrix() is the first function to be called in this phase of the routing protocol and is called enum-explore-reply-delay seconds after an Exploration message is received by a sink.

checkSeqNum() checks the validity of sequence numbers and is called whenever a message arrives. It behaves similarly to its counterpart in the exploration phase and is summarised here for convenience. The function returns a boolean indicating whether the incoming message is valid, based on its sequence number compared to the sequence number held at the node. An invalid message is one with an old sequence number and should be discarded. If the sequence number is new, then the node resets all routing data and if the sequence number is current then no action is taken.

The function returns a boolean indicating whether the incoming message is valid, i.e. whether it should be acted upon. An old sequence number should be discarded and causes the function to return a value of false. Such messages should be ignored since they potentially carry obsolete information. New sequence numbers cause the node's sequence number to be updated and all of its routing data to be reset. If the node is a source and is not currently waiting to initiate a path discovery process, then path rediscovery will be scheduled for *enum-path-retry-delay* seconds time. The node then forwards an Error message containing the new sequence number throughout the network. If the sequence number in the message matches that on the node, then a value of true is returned, indicating that the message should be retained.

On_Receive_ExplorationReplyMsg() parses ExplorationReply messages. As with other incoming messages, the sequence number of the message is first checked and the message is discarded if the sequence number is old. On receiving an ExplorationReply message,

the node merges the contained matrix, sources and sinks with RoutingMatrix, KnownSources and KnownSinks that are stored on the node. If the ExplorationReply contains a new request number, the node updates its local version of the number, erases its set of KnownReplies and forwards the message. By eliminating the set of KnownReplies, the ExplorationReply messages from sink nodes can be propagated through the network again. Conversely, if the ExplorationReply message has already been received, it is discarded. Finally, the ExplorationReply message is forwarded and the a timer is started to call the *formPath()* function after *enum-route-reply-collection-time* seconds. The timer is reset whenever an ExplorationReply message is received. The *formPath()* function represents the start of the enforcement and data phase.

8.2.2.4 Example

This example is a continuation of the example from Section 8.2.1.4 and addresses the reply phase of the routing protocol. The same constants, assumptions and notations are used. The second phase begins when the *outputRoutingMatrix()* function is called at time 7, i.e. enum-explore-reply-delay (5) seconds after the first Exploration message was received by a sink.

At time 7, sinks 6 and 7 broadcast ExplorationReply messages. The messages contain the SeqNum, ReqNum, RoutingMatrix, KnownSources and KnownSinks sets at each sink along with the sink's ID. Initially, the ExplorationReply message broadcast by sink 7 is received by nodes 3 and 5 and the ExplorationReply message broadcast by sink 6 is received by node 4. Each of those nodes records the ID of the node that the ExplorationReply originated from in their KnownReplies set. The receiving nodes also merge the matrix, source and sink sets of the incoming message with their own RoutingMatrix, KnownSources and KnownSinks sets. These interactions are shown in Figure 8.5.

At time 8, nodes 3 and 5 broadcast the ExplorationReply message originating from sink 7 and node 4 broadcasts the ExplorationReply originating from sink 6 and node 4 broadcasts the ExplorationReply received from sink 7. The outgoing messages consist of the RoutingMatrix, KnownSources and KnownSinks sets of the sending nodes as shown in Figure 8.6. Source 1 receives the ExplorationReply messages from nodes 3 and 5. When the first message arrives, it is added to the node's KnownReplies set, so that when the second message arrives, it is discarded as being a reply that has already been received. The KnownSources,

KnownSinks and RoutingMatrix structures are also all merged at source 1. Sink 7 receives ExplorationReply messages from nodes 4 and 3. The ExplorationReply originating from sink 7 is already known by sink 7 and so is ignored. However, the data structures from the ExplorationReply originating from sink 6 are merged with the RoutingMatrix, KnownSources and KnownSinks data structures that are held at sink 7.

At time 9, source 1 broadcasts an ExplorationReply message to nodes 2 and 3 and sink 7 broadcasts an ExplorationReply to nodes 3 and 5 as shown in Figure 8.7. In order to demonstrate the effect of unreliable communication, it is arbitrarily decided that the transmission to node 2 (shown in red) is lost. Node 3 has also already received the ExplorationReply from node 6. Thus, the ExplorationReply broadcast by source 1 has no effect. The ExplorationReply message from sink 7 uses the new, merged, KnownSources, KnownSinks and RoutingMatrix data structures that were held at sink 7. However, the origin sink of the message remains as sink 6. Since neither node 3 or 5 has received an ExplorationReply from sink 6, the data structures are merged and the nodes rebroadcast the message.

At time 10, nodes 5 and 3 rebroadcast the ExplorationReply message originating from sink 6. The message is received by sink 7, which has received the message already and ignores it. The message is also received twice by source 1. The first time the message is received, the data structures are merged and the message is rebroadcast. The second time it is received, the message is discarded. Figure 8.8 shows the transactions that occur during time 10.

At time 11, source 1 broadcasts the ExplorationReply that originated from sink 6 as shown in Figure 8.9. Node 3 has already received the ExplorationReply from sink 6 and so discards the message. Source 2 fails to receive the transmission (shown in red).

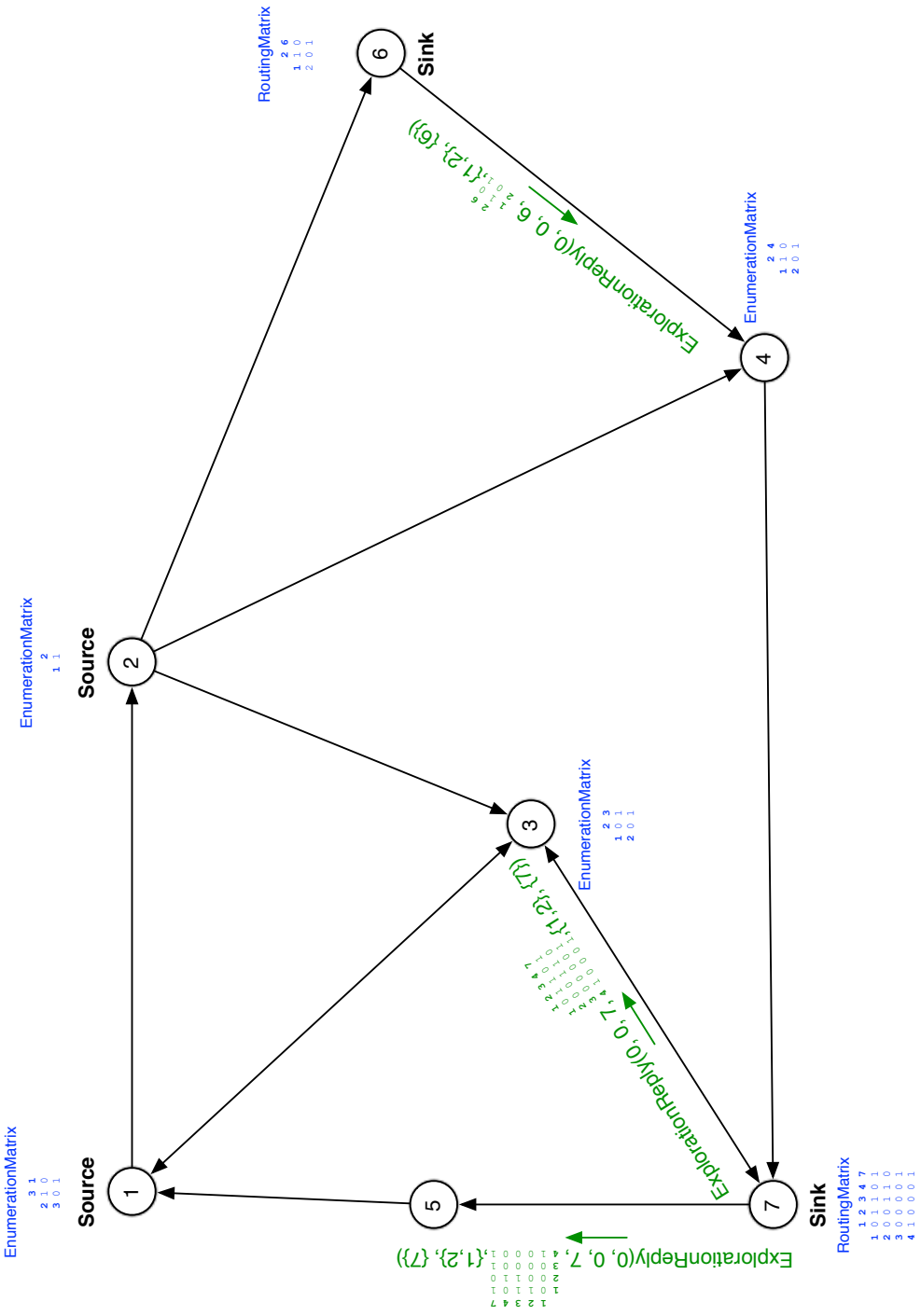


Figure 8.5: State of the network at time 7

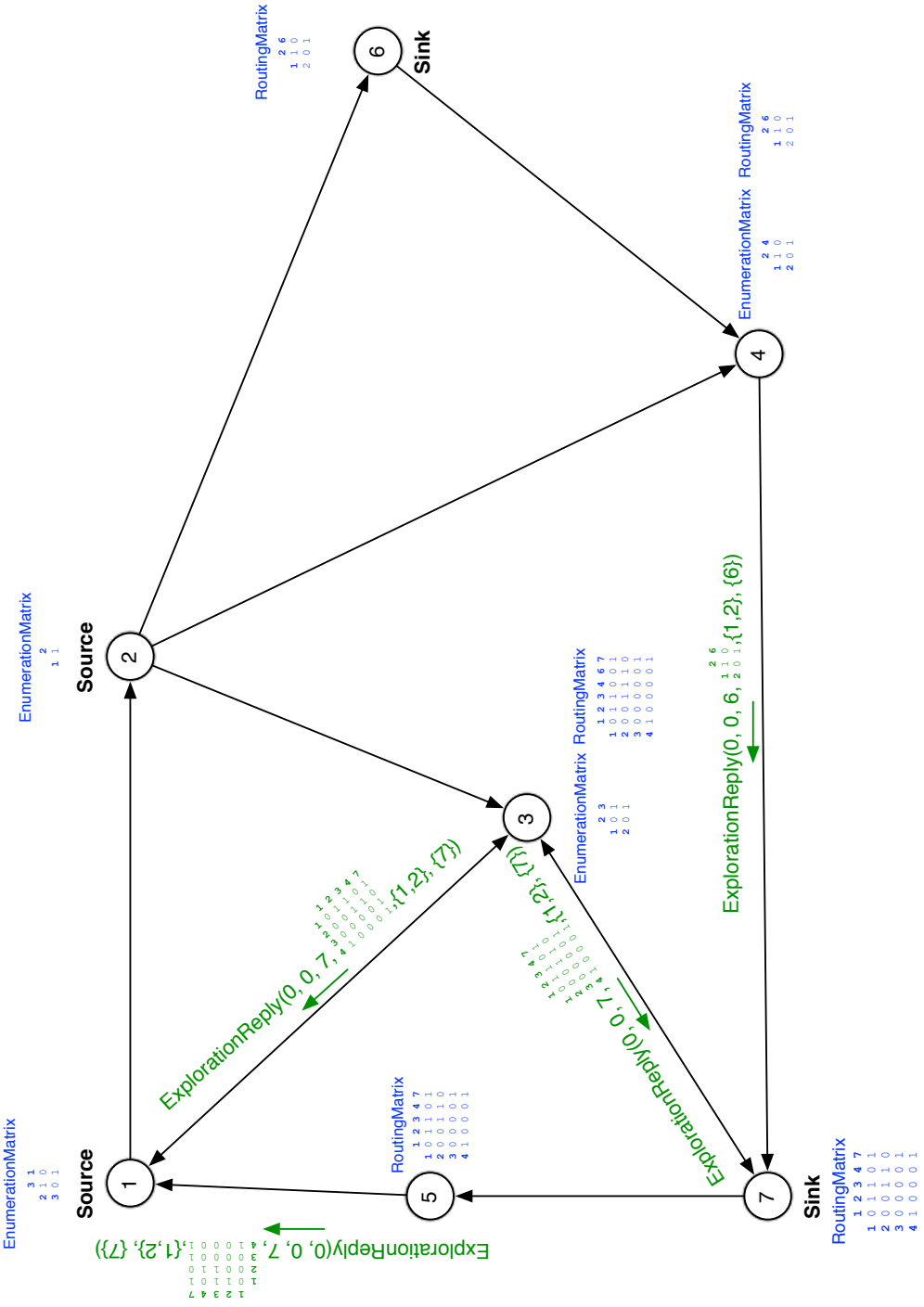


Figure 8.6: State of the network at time 8

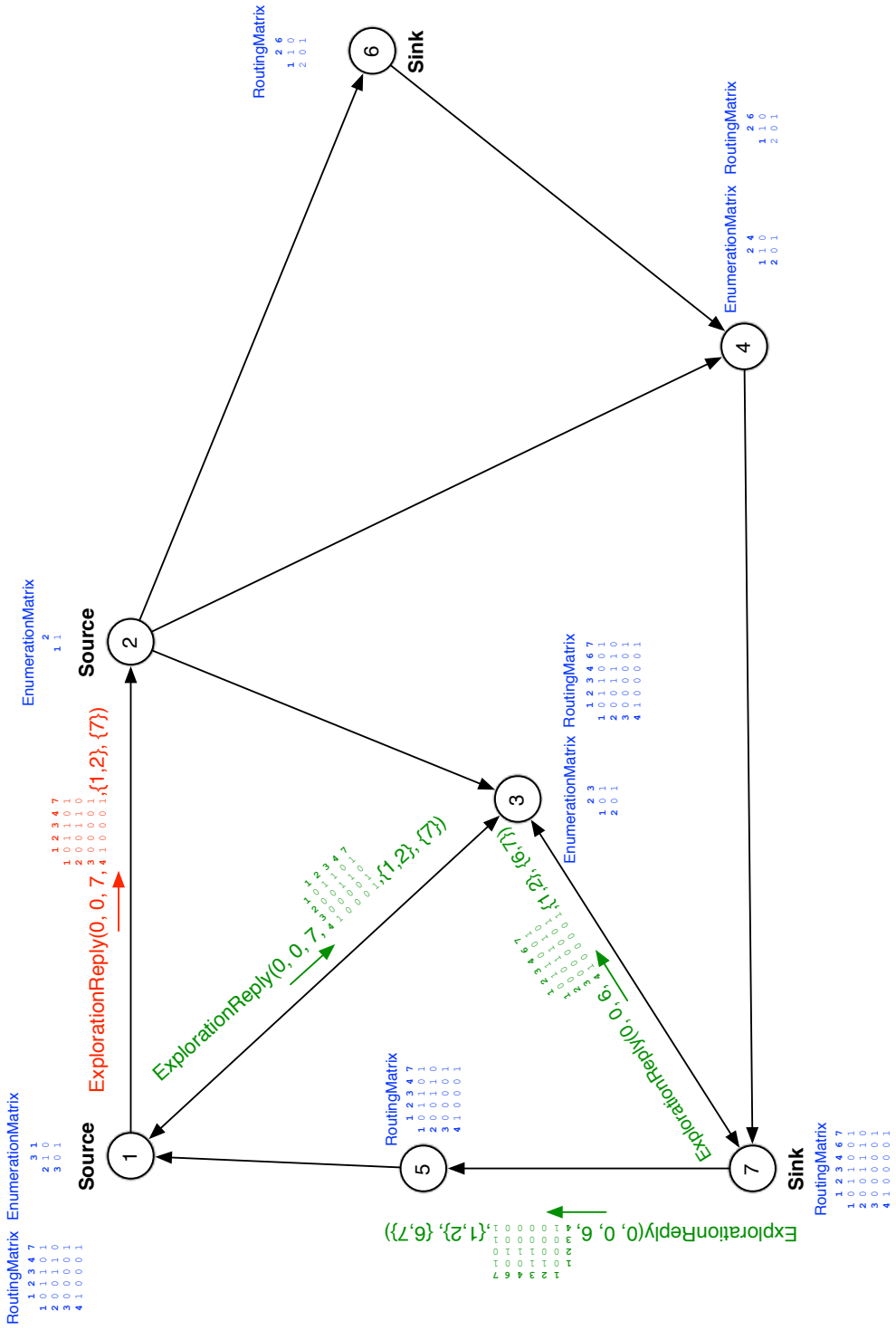


Figure 8.7: State of the network at time 9

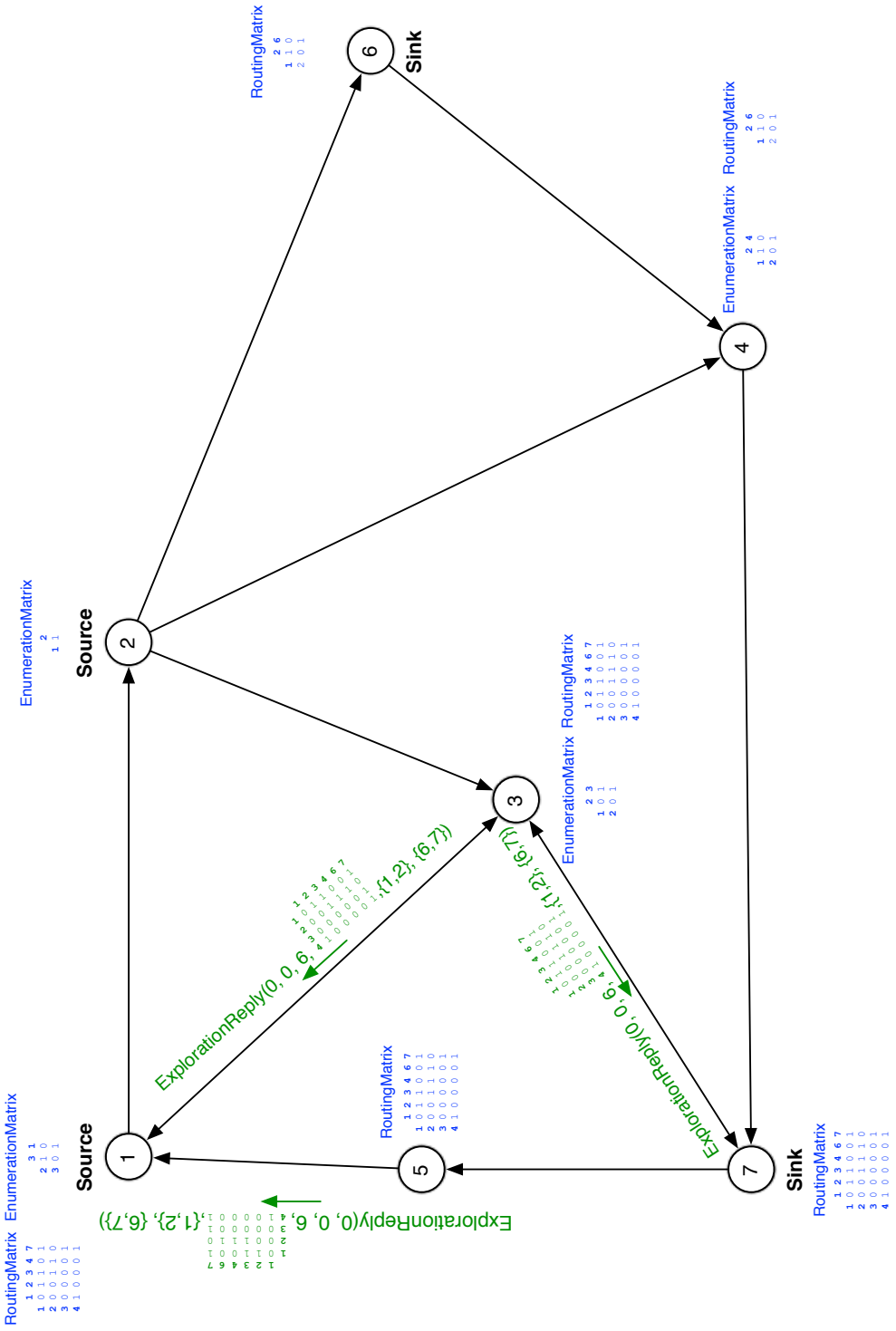


Figure 8.8: State of the network at time 10

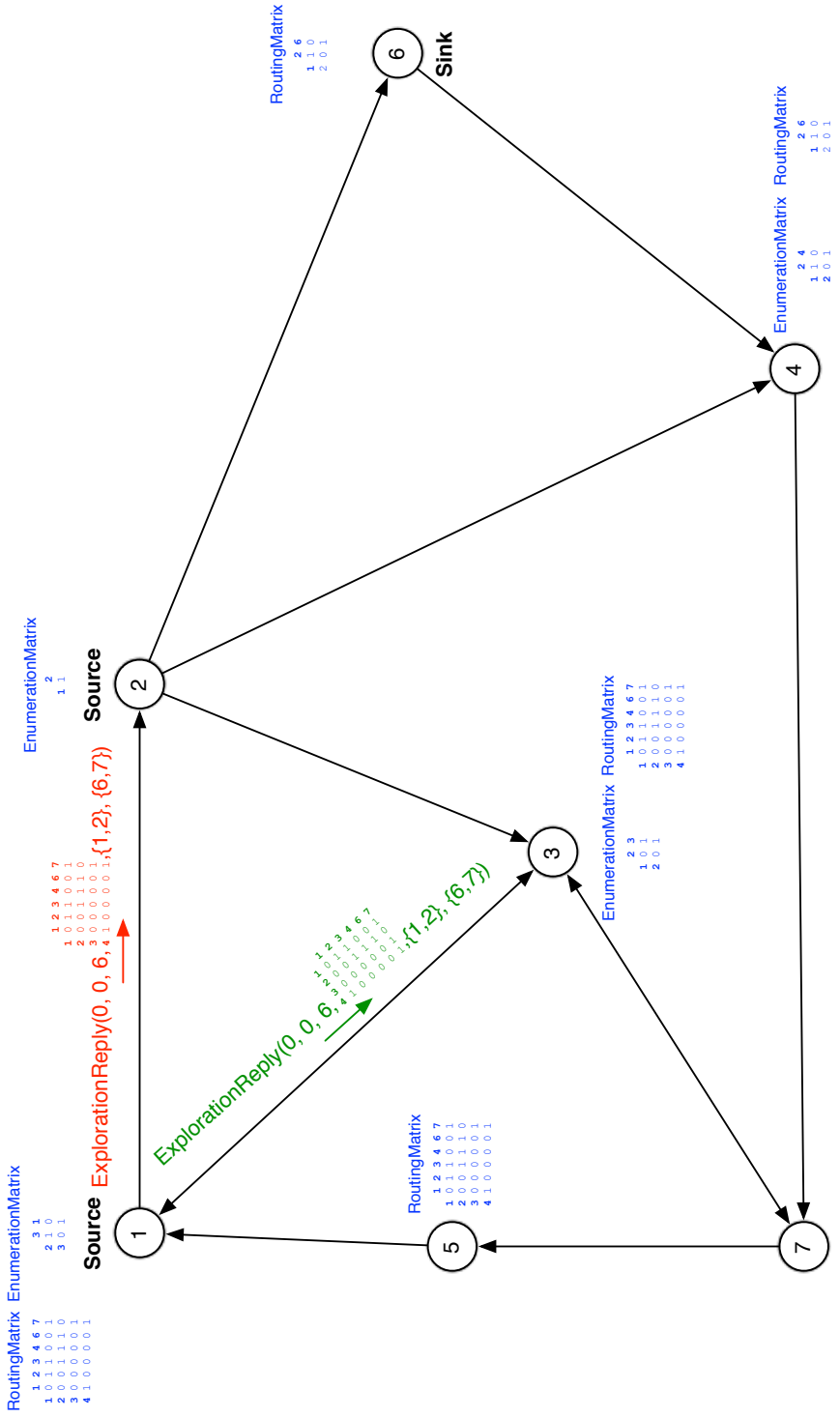


Figure 8.9: State of the network at time 11

8.2.3 Enforcement and Data Phase

The third phase of the protocol is the path enforcement and data phase. Each source enters this phase if *enum-route-reply-collection-time* seconds pass in which no further routing matrices are received. During the enforcement and data phase, a source node sends a message along the path that is to be used for sending data. As the message passes along the path, each intermediate node notes its next hop. In this manner, the path is enforced, i.e. future messages need not contain the entire path to be used for transmission. Consequently, less data is communicated by radio, reducing energy expenditure. Once the path has been enforced, the process of sending periodic data towards a sink node can begin.

8.2.3.1 Messages

The enforcement and data phase involves the exchange of *Path*, *Data* and *Error* messages:

Path messages are sent along a path that is to be used for (source, sink) routing. Each node along the path notes its next hop. Path messages consist of three fields:

- The network-wide sequence number, as known by the source.
- The origin source of the message.
- The remaining sequence of nodes that the message must travel through, to reach the sink.

Data messages contain the data sensed by the source nodes. The messages are made up of four fields:

- The network-wide sequence number, as known by the source.
- Origin source of the message.
- The next hop for the data to be sent to.
- The data.

Sink nodes broadcast *Error* messages when a node failure is suspected. They contain a single field, which is the new universal sequence number. When a node receives an Error message with a new sequence number, it updates its local copy of the universal sequence number to that of the message, resets all its routing data and forwards the message. As with all other messages, Error messages containing an old sequence number are ignored. Error messages with the same sequence number as the node have no effect.

8.2.3.2 Data Structures

The enforcement and data phase uses the following data structures:

SeqNum is the universal sequence number for the network. Every time it is incremented, the routing information held in the network resets and the exploration phase begins anew. SeqNum is initially 0.

NextHops is a mapping of sources to nodes. Given a source, the NextHops data structure indicates the node to which a Data message should be forwarded, in order to reach the sink.

8.2.3.3 Pseudo code

Below is the pseudo code for the enforcement and data phase of the enumeration algorithm. As well as the data structures and messages shown above, the following functions are used within the code:

id() returns the unique identifier for a node.

role() returns the role of a node (i.e. whether it is a source or a sink).

delay(f, t) executes function f after t seconds. Calling delay on a function that is already delayed changes the time until that function is executed.

delayed(f) returns a boolean indicating whether function f is due to be executed at some future time (via the *delay(f, t)* function).

sourceIsActive(n) updates the internal record that source n is active. If a node has not been active for some period of time, the sink will reset the network by sending an Error message

and path formation will begin again (code not shown).

```
1  void formPath() {
2     Node[] path = RoutingMatrix.cheapestPath().removeIndex(0)
3     NextHops[id()] = path[0]
4     sendPathMsg(seqNum, path, id())
5  }
6
7  boolean checkSeqNum(int seq) {
8     if (seq > SeqNum) {
9        SeqNum = seq
10       resetRoutingData()
11       sendErrorMsg(SeqNum)
12       if (role() == source && !delayed(routingRequest))
13          delay(routingRequest, ENUM-PATH-RETRY-DELAY)
14    } else if (seq < SeqNum)
15       return false
16
17    return true
18 }
19
20 On_Receive_PathMsg(int seq, Node[] path, int source) {
21    if (!checkSeqNum(seq) || path[0] != id())
22       return
23
24    Node[] newPath = path.removeIndex(0)
25    if (newPath.length > 0){
26       NextHops[source] = newPath[0]
27       sendPathMsg(seq, newPath, source)
28    }
29    else if (newPath.length == 0 && role() == sink)
30       sourceIsActive(source)
31 }
32
33 On_Receive_DataMsg(int seq, int source, int target, Data data) {
34    if (!checkSeqNum(seq) || target != id())
35       return
```

```
36
37      if (role() == sink)
38          sourceIsActive(source)
39      else
40          sendDataMessage(seq, source, NextHops[source], data)
41  }
```

checkSeqNum() checks the validity of sequence numbers and is called whenever a message arrives. It behaves similarly to its counterpart in the exploration phase and is summarised here for convenience. The function returns a boolean indicating whether the incoming message is valid, based on its sequence number compared to the sequence number held at the node. An invalid message is one with an old sequence number and should be discarded. If the sequence number is new, then the node resets all routing data and if the sequence number is current then no action is taken.

The *formPath()* method is executed by a source node after an ExplorationReply message is received and if *enum-route-reply-collection-time* seconds elapse without any further ExplorationReply messages being received. The method determines the cheapest path to a sink and sends a Path message, thus enforcing that path.

On_Receive_PathMsg() deals with receiving a Path message. As with all messages, Path messages are discarded if they contain an old sequence number. However, they may also be discarded if the receiving node is not the intended recipient of the message. Otherwise, the first element of the path is removed. If additional nodes lie on the path, the next hop for this source is stored in the node's NextHops data structure, which indicates the next node to use when forwarding for a particular source. The Path message is then rebroadcast. In this manner, the Path message is forwarded through the network with the first element being removed at each hop. When the Path message arrives at the destination sink, the sink notes that the source node from where the Path message originates is active.

On_Receive_DataMsg() handles the Data messages. As with Path messages, the Data messages are intended for a specific node. Consequently, if other nodes receive the message, they discard it. The message may also be discarded if it contains an old sequence number. If the receiving node is a sink, it notes that the origin of the message (the source) is still active. Otherwise, the node determines what the next hop is, in order to reach the destination for the particular source and forwards the message onwards.

8.2.3.4 Example

This example is a further continuation of the example from Sections 8.2.1.4 and 8.2.2.4. This section addresses the third and final enforcement and data phase of the routing protocol. The same constants, assumptions and notations are used. The third phase begins when the *formPath()* function is called at time 15, i.e. *enum-route-reply-collection-time* (5) seconds after the last ExplorationReply message was received by a source.

At time 15, source 1 calculates the cheapest path to a sink, using its RoutingMatrix. The cheapest path is calculated as being 1:3:7. Source 1 removes its ID from the head of the path and stores NextHop[1] = 3, indicating that the next hop to reach the sink required by itself (node 1) is node 3. Source 1 then broadcasts a Path message containing the path 3:7. The message is received by source 2 and node 3. The state of the network is shown in Figure 8.10.

At time 16, source 2 and node 3 receive the Path message that originated from source 1. By examining the path contained within the message, both nodes determine that the intended recipient is node 3. Source 2 therefore discards the message. Node 3 removes its ID from the head of the path and records the mapping NextHops[1] = 7, indicating that the next hop to reach source 1's sink is node 7. The Path message is then broadcast by node 3, containing the truncated path, which is received by sink 7. These transactions are shown in Figure 8.11. Being the destination sink, the message is not forwarded any further. However, sink 7 notes that it is now receiving data from source 1. When data is to be forwarded by the application, each node need only send the data itself and the source of the data. Each node is able to determine the next hop and forward the data one hop closer to the sink.

Data messages are not shown in this example, since they are specific to the underlying application and not the routing protocol.

At time 31, i.e. *enum-path-formation-time* (30) seconds after the node sends its Exploration message, source 1 will check whether it has a non-empty RoutingMatrix. Since the RoutingMatrix is not empty, no further action is taken.

At time 32, i.e. *enum-path-formation-time* (30) seconds after the node sends its Exploration message, source 2 will check whether it has a non-empty RoutingMatrix. Since the RoutingMatrix is empty, and since the maximum number of resend requests (*enum-rreq-*

resend-attempts) has not been reached, the source increments its ReqNum by 1, and begins the first phase of the algorithm again. As part of this process, the node schedules the *check-PathExists()* function to execute again after an additional *enum-path-formation-time* (30) seconds.

The sequences of message transactions that occur during the repeated phases are not expounded upon, since they are virtually identical to those that have already been discussed. All simplest paths from source 2 to the sinks will be re-enumerated and stored at the sink nodes. The sink nodes will then flood ExplorationReply messages through the network again. The ExplorationReply messages contain a new ReqNum (1 rather than 0), and thus each node will forward the new ExplorationReply messages exactly once. The description of message transactions begins again at time 47, which is the point at which source 1 would begin its enforcement and data phase for the second time.

At time 47, source 1 again uses its RoutingMatrix and determines that the shortest path to the sink is the path 1:3:7. It removes its ID from the head of the path, notes that NextHop[1] is node 3 and rebroadcasts the Path message which will be received and ignored by source 2 and received by node 3. These message transactions are shown in Figure 8.12.

At time 48, source 2 determines its shortest path to the sink is the path 2:6. It removes its ID from the path, notes NextHop[2] = 6 and forwards a Path message. The message will be received and ignored by nodes 4 and 3. Sink 6 will note that it is the intended recipient of the message and that source 2 is active. Similarly, the Path message originating from source 1 will be forwarded by node 3. This message will be received and ignored by node 1 and received by node 7. As the intended recipient and the last node on the path, sink 7 will note that source 1 is active. These transactions are shown in Figure 8.13

Had the new ExplorationReply message not been received by source 2, then that source would have been unable to engage in (source, sink) routing. It has already made the maximum number of resend requests (*enum-rreq-resend-attempts*, i.e. 1) and so could not make more.

After time 48, sources can freely send data towards sink nodes. No further activity takes place with respect to the routing protocol except for path maintenance which is triggered if no data is received from a particular source in a period of *data-loss-tolerance* seconds. If a source failure is suspected by a sink node, the sink increments its SeqNum value,

and erases all locally stored routing data (including its RoutingMatrix, ExplorationMatrix, KnownSources, KnownSinks and KnownReplies). The sink then broadcasts an Error message containing the new SeqNum value.

When a node receives a message with a higher SeqNum value than the one it has stored, it erases its routing data structures, updates its SeqNum value and broadcasts a new Error message. The new sequence number is therefore flooded throughout the entire network. When source nodes receive the Error message, they additionally schedule the exploration phase to begin again after *enum-path-retry-delay* seconds. The use of a persistent sequence number ensures that nodes with potentially old routing data cannot participate in routing.

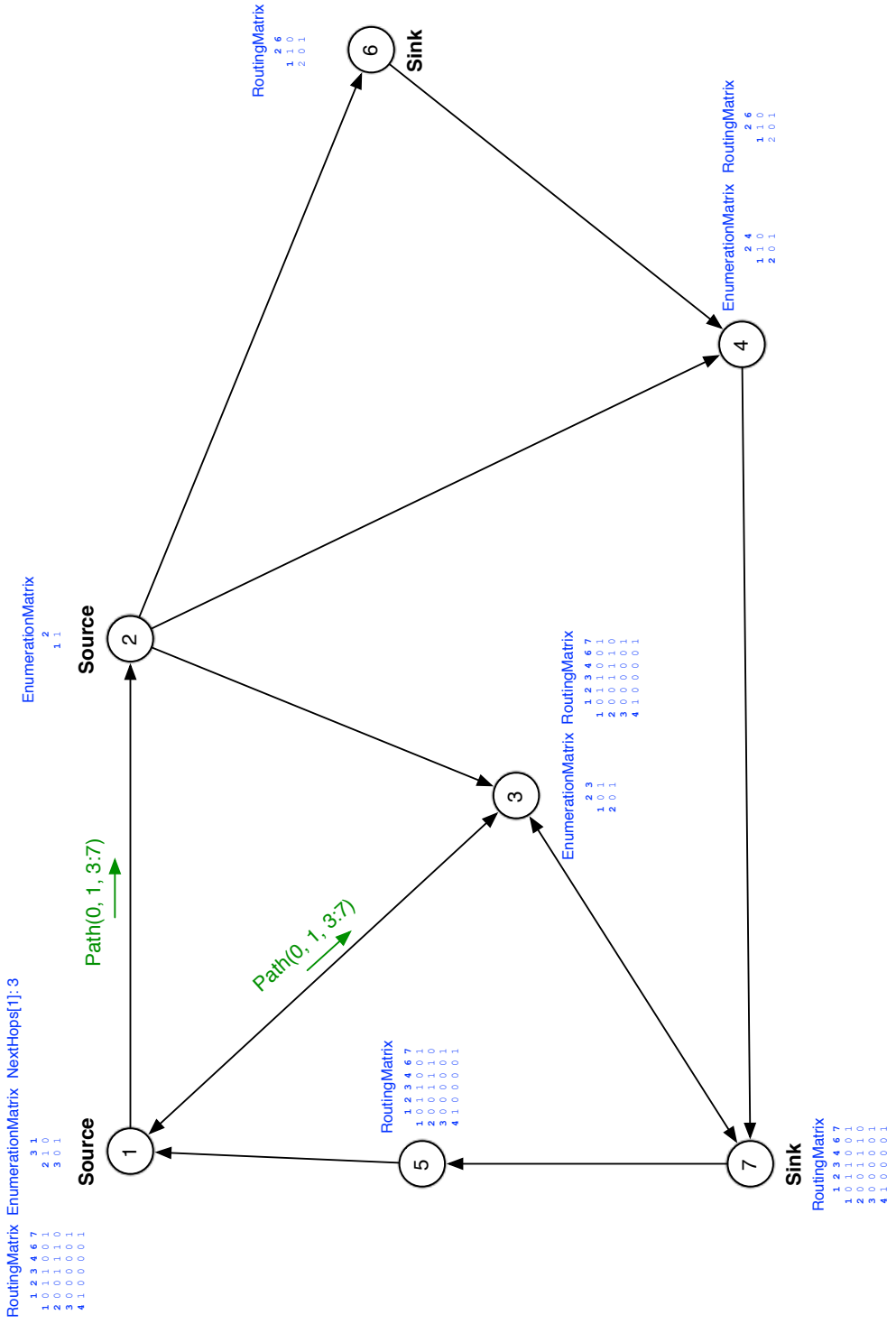


Figure 8.10: State of the network at time 15

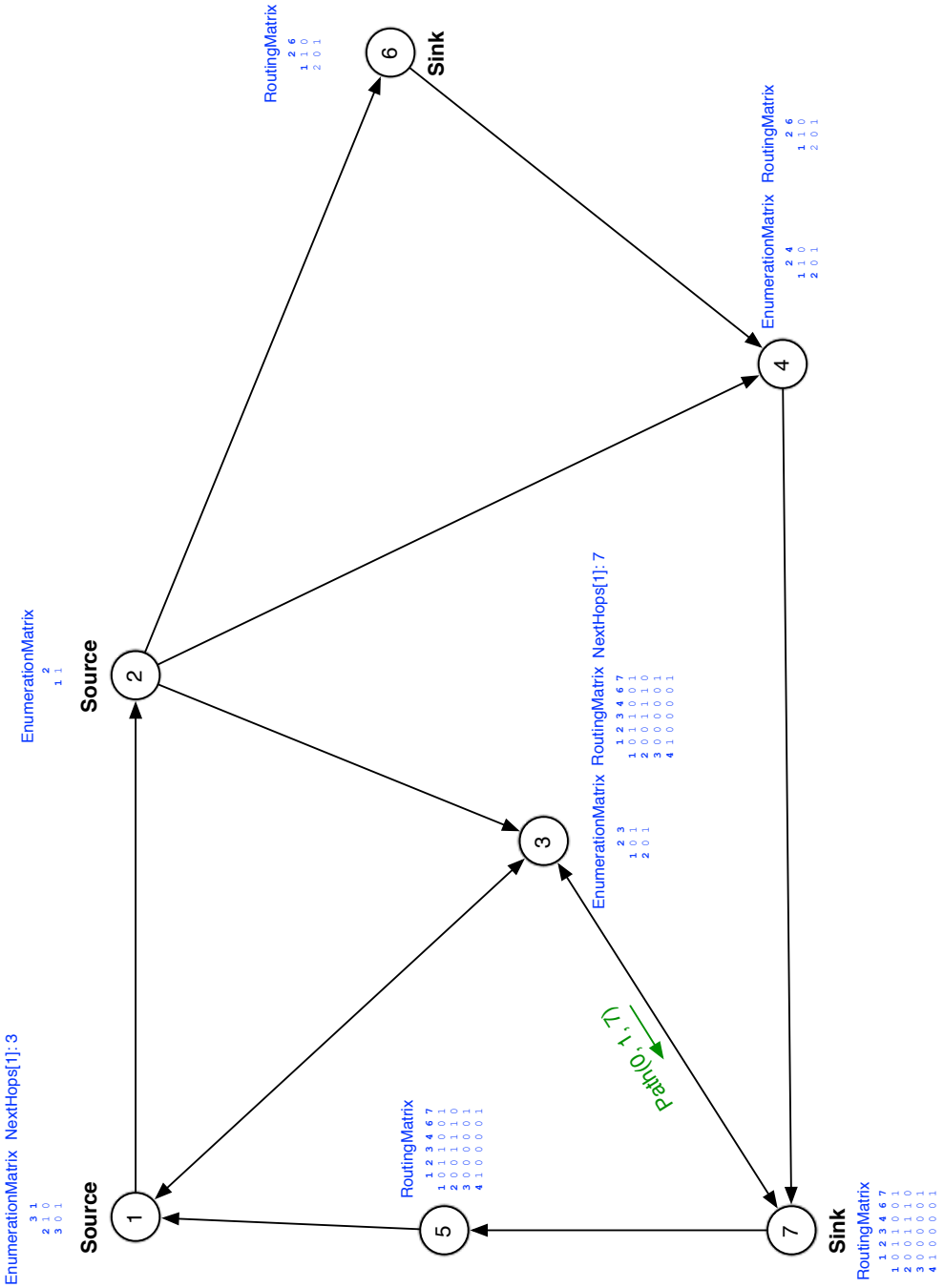


Figure 8.11: State of the network at time 16

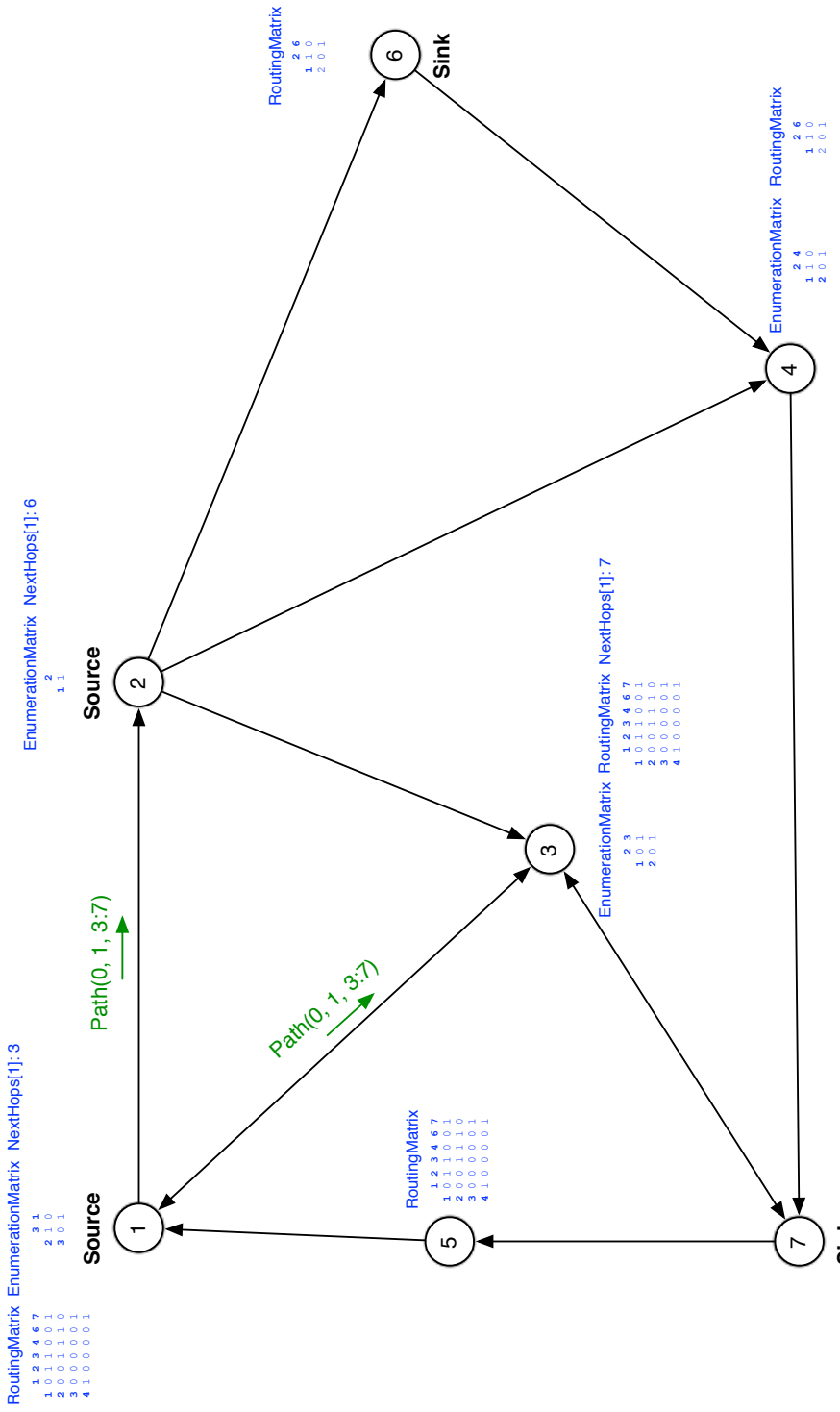


Figure 8.12: State of the network at time 47

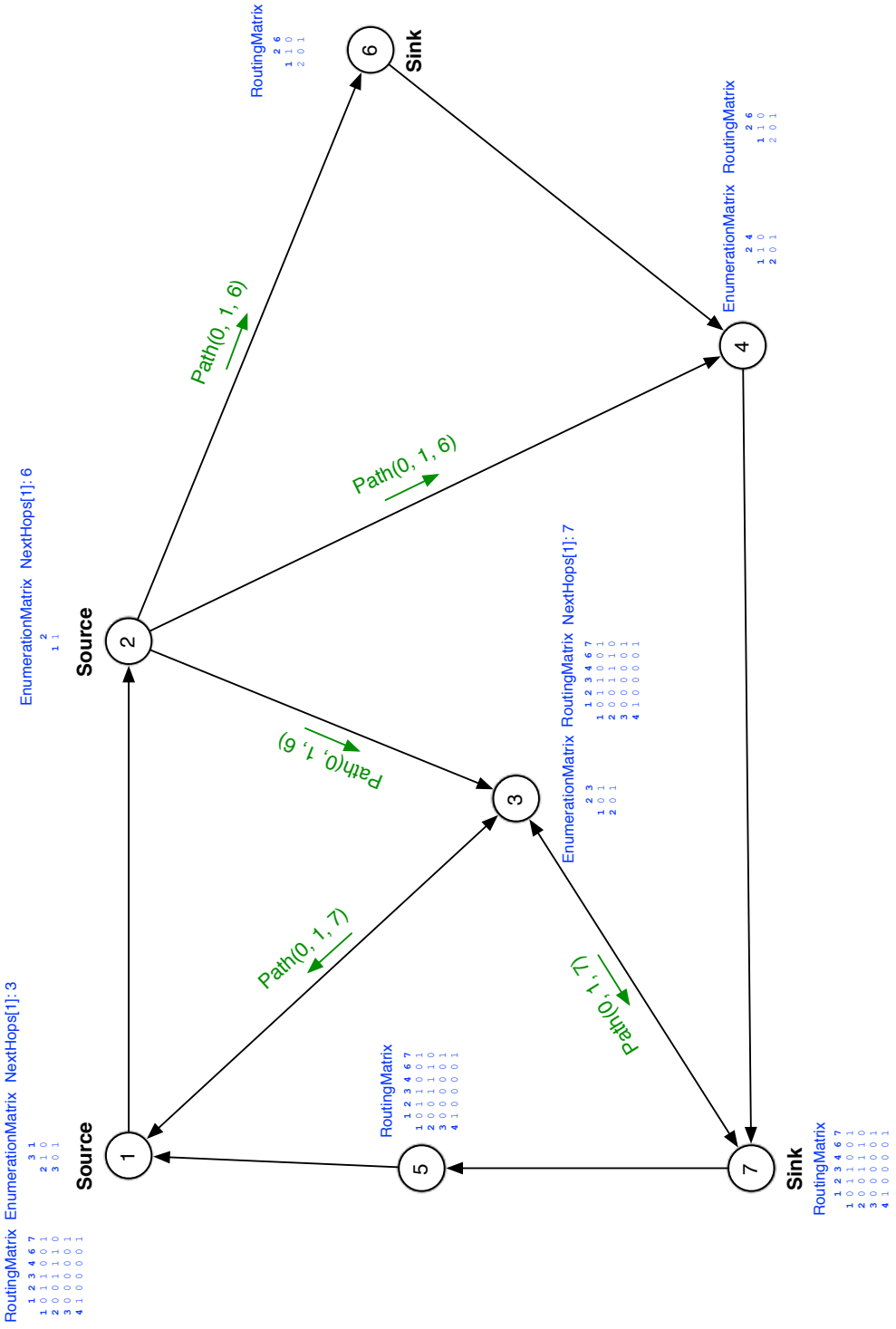


Figure 8.13: State of the network at time 48

8.3 Global Knowledge

In the global knowledge routing protocol, each node gathers a view of the entire network. The sources use their network views to calculate the set of required paths. As well as determining the neighbours of each node in the network, nodes must also gather the set of source and sink nodes so as to determine the relative reliance values to all (source, sink) pairs in the network, and thus calculate the absolute reliance values of nodes. Since the approach requires every source to gather data from every other node in the network, the number of messages that must be exchanged has order O(n^2) where n represents the number of nodes in the network.

The routing protocol consists of two phases, which are the *topology-sharing phase* and the *enforcement and data phase*.

The first phase is the topology-sharing phase, in which each node progressively builds up a view of the entire network by sharing network knowledge with its neighbours, including known sources and sinks.

The second phase is the path enforcement and data phase, in which a path is selected by each source and nodes along that path are informed of their next neighbour when forwarding data to a sink.

The routing protocol has been presented as two phases for the sake of improving the clarity of the routing protocol's behaviour. The first phase corresponds to the discovery task, discussed in Section 3.1. The costing task, discussed in Section 3.2 takes place between the first and second phases. This routing protocol has no modules from the selection task; it simply uses the cheapest path.

As with the enumeration routing protocol, nodes may fail during the operation of the network. If a period of *data-loss-tolerance* seconds elapses in which no data is received from a previously transmitting source, it is assumed that source has failed. If a source loss is suspected, each node in the network erases its routing data and the topology-sharing phase begins again. It is necessary to erase all routing data in the network to ensure that old, invalid data is not held by any nodes. Since it is non-trivial to determine which node(s) have failed, all routing data in the entire network must be completely reset. In order to ensure that old routing data is not kept, a network wide *sequence number* is incremented

by the sink when a reset is requested. The sequence number is included in all messages sent through the network and nodes with old sequence numbers cannot be used to route and cannot participate in route discovery.

The two phases of the network are discussed in the following sections.

8.3.1 Topology-Sharing Phase

The topology-sharing phase is the first phase of the global knowledge routing protocol, in which nodes exchange topology data with one another in order to gather the topology of the entire network. The process begins by each node sending a single message, so that any neighbouring nodes can detect its presence. Whenever a node discovers new parts of the network, it sends those newly discovered parts to its neighbours. Rather than send each new section of topology immediately, a node waits for up to *gk-path-formation-time* seconds, amalgamating new parts together before sending them out in a single message.

A source will assume that its attempt to form a path to a sink has failed if it has not gathered sufficient topology data to form a (source, sink) path after *gk-path-formation-time* seconds elapse. Such failure may occur if, for example, topology-sharing messages are lost in the network. If a source fails to form a path, it sends a message that requires each node in the network to resend their entire gathered topology data. However, the number of times that this request can be made is limited to ensure that the batteries of nodes are not being continually drained trying to unsuccessfully form paths.

8.3.1.1 Messages

During the topology-sharing phase, two types of message are exchanged, namely *Matrix* messages and *Resend* messages.

Matrix messages contain views of the network that are exchanged between nodes, and consist of five fields:

- The network-wide sequence number, as known by the sending node.
- The sender of the Matrix message.

- An adjacency matrix representing known links between nodes (as previously discussed and shown).
- The IDs of known sources.
- The IDs of known sinks.

Matrix messages may be sent in multiple parts by splitting the matrix data structure (the second field). Since the probability of receiving a smaller message is greater than that of receiving a bigger message, it is more likely that some network information will arrive at the neighbouring nodes. The full sets of sources and sinks are included in every part of every outgoing Matrix message.

A *Resend* message is flooded throughout the network by a source node if it fails to establish any path to the sink after *gk-path-formation-time* seconds. This message causes all nodes to wait a defined period of time before beginning the first phase of the algorithm again. However, unlike a network-wide reset, each node maintains its routing data. Thus, nodes gain an opportunity to resend the routing data that they have learned. Resend messages contain two fields:

- The network-wide sequence number as known by the source.
- The resend sequence number (ReqNum, defined below).

8.3.1.2 Data Structures

The topology-sharing phase of the link state advertising protocol uses the following data structures:

KnownSources refers to the set of sources that have been discovered in the network. The set is built up at the same time that the network topology data is exchanged by nodes. Each source is responsible for announcing its status as a source node when it makes its presence known to its neighbours.

KnownSinks refers to the set of sinks that have been discovered in the network. As with KnownSources, it is built up at the time that network topology data is exchanged by nodes.

Each sink is responsible for announcing its status as a sink node when it makes its presence known to neighbours.

Matrix is an n-by-n boolean (adjacency) matrix that represents a node's current view of the network. A boolean value in cell (i, j) indicates whether an edge exists from node i to node j in the network. By merging matrices between nodes through the use of Matrix messages, each node is able to iteratively gather a view of the entire network.

LastSentMatrix is an n-by-n boolean (adjacency) matrix that represents the node's view of the network when it last sent a Matrix message. By examining the differences between the LastSentMatrix and Matrix, the node is able to send only those network connections that have been newly discovered, rather than its entire network view, in every Matrix message, allowing it to reduce its energy consumption.

SeqNum is the universal sequence number for the network. Every time it is incremented, the routing information held by the nodes in the network is reset and the first phase of the routing protocol begins again. SeqNum is initially 0.

ReqNum refers to the number of times nodes have been requested to send their entire view of the network, rather than incremental changes. ReqNum is initially 0.

8.3.1.3 Pseudo code

Below is the pseudo code for the two phases of the global knowledge algorithm. With the exception of the code that determines node and path costs, the algorithm does not change depending on the reliance heuristic being used. The same code runs on each node. As well as the data structures and messages discussed above, the following functions are used within the code:

id() returns the unique identifier for the node.

role() returns the role of a node (i.e. if it is a source, or a sink).

delay(f, t) executes function f after t seconds. Calling delay on a function that is already delayed changes the time until that function is executed.

delayed(f) returns a boolean indicating whether function f is currently the subject of a

delay() request.

sourceIsActive(n) is a function used by sink nodes to indicate that a source is currently active and routing data to the sink. If some period of time passes without a source being marked as active, the sink will reset the entire network by incrementing its value of SeqNum and broadcasting a new Matrix message.

```
1  On_Startup() {
2     beginPropagation()
3  }
4
5  void beginPropagation() {
6     if (role() == source) {
7        KnownSources.add(id())
8        delay(checkPathExists, GK-PATH-FORMATION-TIME)
9     }
10    if (role() == sink)
11       KnownSinks.add(id())
12
13    if (!delayed(amalgamate)) {
14       LastSentMatrix = Matrix
15       sendMatrixMsg(SeqNum, id(), Matrix, KnownSources, KnownSinks)
16    }
17 }
18
19 void checkPathExists() {
20    Node[] cheapestPath = Matrix.cheapestPath()
21    cheapestPath = = cheapestPath.removeIndex(0)
22    if (cheapestPath.length <= 0
23    && ReqNum < GK-MATRIX-RESEND-ATTEMPTS) {
24       ReqNum++
25       LastSentMatrix = null
26       delay(beginPropagation, GK-RESEND-DELAY)
27       sendResendMsg(SeqNum, ReqNum)
28    }
29    else if (cheapestPath > 0) {
30       NextHops[id()] = cheapestPath[0]
```

```
31        sendPathMsg(SeqNum, cheapestPath, id())
32     }
33  }
34
35  On_Receive_MatrixMsg(int seq, int from, Matrix mat,
36  Set sources, Set sinks) {
37     if (!checkSeqNum(seq))
38        return
39
40     KnownSources |= sources
41     KnownSinks |= sinks
42     Matrix |= mat
43     Matrix.add(from, id())
44
45     if (role() == source)
46        KnownSources.add(id())
47     if (role() == sink)
48        KnownSinks.add(id())
49
50     if (!delayed(amalgamate))
51        delay(amalgamate, GK-BEACON-DELAY)
52  }
53
54  boolean checkSeqNum(int seq) {
55     if (seq > SeqNum) {
56        SeqNum = seq
57        resetRoutingData()
58        beginPropagation()
59     } else if (seq < SeqNum)
60        return false
61
62     return true
63  }
64
65  void amalgamate() {
66     Matrix diff = Matrix ^ LastSentMatrix
67     LastSentMatrix = Matrix
```

```
68
69      if (!diff.isZero())
70          sendMatrixMsg(SeqNum, id(), diff, KnownSources, KnownSinks)
71  }
72
73  On_Receive_ResendMsg(int seq, int req) {
74      if (!checkSeqNum(seq))
75          return
76
77      if (req <= ReqNum)
78          return
79
80      ReqNum = req
81      if (!delayed(beginPropagation)) {
82          MatrixLastSent = null
83          delay(beginPropagation, GK-RESEND-DELAY)
84      }
85
86      sendResendMsg(SeqNum, ReqNum)
87  }
```

On_Startup() handles the startup of the nodes, which is assumed to be simultaneous. The startup logic itself is handled in the *beginPropagation()* function, so that it may be called in the event of resend requests.

The *beginPropagation()* function causes each source node to store its own ID in its set of KnownSources and each sink node to store its own ID in its set of KnownSinks. Sources also set a timer for *gk-path-formation-time* seconds to call checkPathExists, which checks whether a path to a sink exists and sends the resend request if no path to the sink is found. If the node is not currently amalgamating incoming topology data, it sends a Matrix message containing the entire known topology, known sources and known sinks. The node then updates its value of MatrixLastSent to reflect the Matrix message that was just transmitted.

checkPathExists() is timed to be called *gk-path-formation-time* seconds after a source begins propagating network data. If the source has found a path to a sink, the source enters the second (enforcement and data) phase, which is discussed in Section 8.3.2. If no path is found, the source increments its resend request number and propagates a Resend messages

through the network, which causes each node to resend its known topology.

On_Receive_MatrixMsg() handles the parsing of Matrix messages. Firstly the network-wide sequence number within the incoming message is checked. If the sequence number is old then the message is dropped. Otherwise, the data structures of the incoming message are merged with those held on the node. If the node is not already amalgamating incoming data from other nodes, then the *amalgamate()* function is scheduled to run after *gk-beacon-delay* seconds elapse, causing the node to send out any newly discovered parts of the network topology.

checkSeqNum() behaves similarly to its counterpart in the enumeration routing protocol and is called for any incoming message. The function returns a boolean indicating whether the incoming message should be discarded or not, based on the sequence number. If the sequence number of the incoming message is higher (newer) then the node updates its own sequence number to that of the message, erases all its routing data and immediately restarts the topology-sharing phase again with the new sequence number by calling the *beginPropagation()* function.

amalgamate() is used to determine what new parts of the network topology have been discovered since the node last sent a Matrix message and to distribute those new discoveries in a Matrix message. The *amalgamate()* function allows a node to delay sending every new topology change immediately and instead combine several changes into a single message.

On_Receive_ResendMsg() handles the parsing of Resend messages. Firstly, as with other messages, *checkSeqNum()* is called to determine whether the message should be discarded based on its sequence number. If the message is not discarded, the node examines the request number of the incoming message. If it is not new, the message is discarded, since the request has already been responded to. Otherwise, the node updates its locally stored version of the resend request number and broadcasts the Resend message to notify other nodes. The node then erases its record of the last matrix to be sent. If the node is not already due to resend its matrix then it will wait for *gk-resend-delay* seconds and resend all its routing data.

Constant	Value
GK-PATH-FORMATION-TIME	10
GK-MATRIX-RESEND-ATTEMPTS	1
GK-BEACON-DELAY	1
RREQ-DELAY-PER-NODE	0
GK-RESEND-DELAY	30

Table 8.2: Constant values for the enumeration algorithm examples

8.3.1.4 Example

This section presents an example of the topology-sharing phase of the global knowledge algorithm. The example network contains two sources (nodes 1 and 2) and two sinks (nodes 5 and 6). A directed arrow from a node A to a node B indicates that messages from node A can be received by node B. In this example, all but seven messages, shown in red on the diagrams, are reliably delivered. The failed messages are used to demonstrate the algorithm's resilience to possible transmission loss. Table 8.2 shows the values of various constants for the global knowledge algorithm.

It is assumed that the nodes all simultaneously activate at time 0. Every node broadcasts a Matrix message containing the Matrix held at that node (which is null at this time in the example), KnownSources and KnownSinks. KnownSources will contain the node's ID if it is a source, and KnownSinks will contain the node's ID if it is a sink. Otherwise, these structures are null. Figure 8.14 shows the message transactions that take place.

At time 1, every node amalgamates the data that it received at time 0. Since no node has yet broadcast a Matrix message, each will broadcast its entire view of the network topology as well as all known sources and known sinks. Figure 8.15 shows the sequence of message transactions that take place. At this time, two messages to source 1 (one from node 3 and one from node 4) are lost.

At time 2, the process repeats. However, since nodes have now sent Matrix messages, each node only retransmits new topology data that it has previously not discovered. The transactions are shown in Figure 8.16 and once again, two messages to source 1 are lost (one from node 3 and one from node 4).

At time 3, some nodes have successfully gathered the entire network topology as well as the complete set of sources and sinks. Consequently, any further Matrix messages they receive after this time will not be sent on. The transactions that take place are shown in Figure 8.17. One message from node 3 to source 1 and one from node 4 to source 1 are lost.

At time 4, nodes send fewer Matrix messages, as each node fails to learn any new topological features. The messages that are sent are shown in Figure 8.18 and another message is lost between node 4 and source 1.

At time 5, every node except source 1 has gathered a view of the entire network topology, including all known sources and known sinks. Node 2 sends out the latest topology data that it has learned. However, the Matrix message transmitted by source 2 is ignored, since it does not include any topology data that nodes 3, 5 or 6 do not know.

At time 10, *gk-path-formation-time* seconds have elapsed since each source began exchanging topology data. Therefore, each source attempts to form a (source, sink) path. Source 1 fails, whereas source 2 succeeds in forming the cheapest path 2:5. Since the path formation phase is not being discussed at this time, the messages that are relevant to that process will not be discussed here. However, they are shown alongside the other messages that are transmitted in Figure 8.20. Having failed to form a (source, sink) path, source 1 increments its resend request counter to 1 and schedules a new attempt to form a path after *gk-path-formation-time* seconds, i.e. at time 50. Source 1 then broadcasts a Resend message, which is received by source 2. Since the resend request number in the incoming message (1) is higher than that held at source 2 (0), the node must react to it.

At time 11, source 2 receives the Resend message from source 1. Source 2 updates its resend request number and schedules to broadcast its Matrix after *gk-resend-delay* seconds, i.e. at time 41. The node rebroadcasts the Resend message, which is received by nodes 3, 5 and 6. Additionally, since the node is a source, it schedules to attempt path formation again after *gk-path-formation-time* seconds, i.e. at time 51. The message transactions are shown in Figure 8.21.

At time 12, nodes 3, 5 and 6 receive the Resend message. Since the resend request number at those nodes (0) is lower than that in the Resend message (1), the message is not discarded. Those nodes rebroadcast the message and schedule to broadcast their Matrix after

gk-resend-delay seconds, i.e. at time 42. The message transactions are shown in Figure 8.22.

At time 13, nodes 1, 2, 3, 5 and 6 receive the Resend message. These nodes ignore the message because their resend request number is the same as that in the Resend message (1). The message is also received by node 4 which increments its resend request number to 1 and schedules to broadcast its Matrix after *gk-resend-delay* seconds, i.e. at time 43. The message transactions are shown in Figure 8.23.

Finally, as indicated previously, nodes rebroadcast their matrices at times 40, 41, 42 and 43 as shown in Figures 8.24, 8.25, 8.26 and 8.27 respectively. At time 42, source 1 learns new topology data from node 3. Consequently, this new data is rebroadcast from source 1 at time 43 in a Matrix message. The Matrix message is received by source 2. However, since the topology data is new to source 2, the message is disregarded.

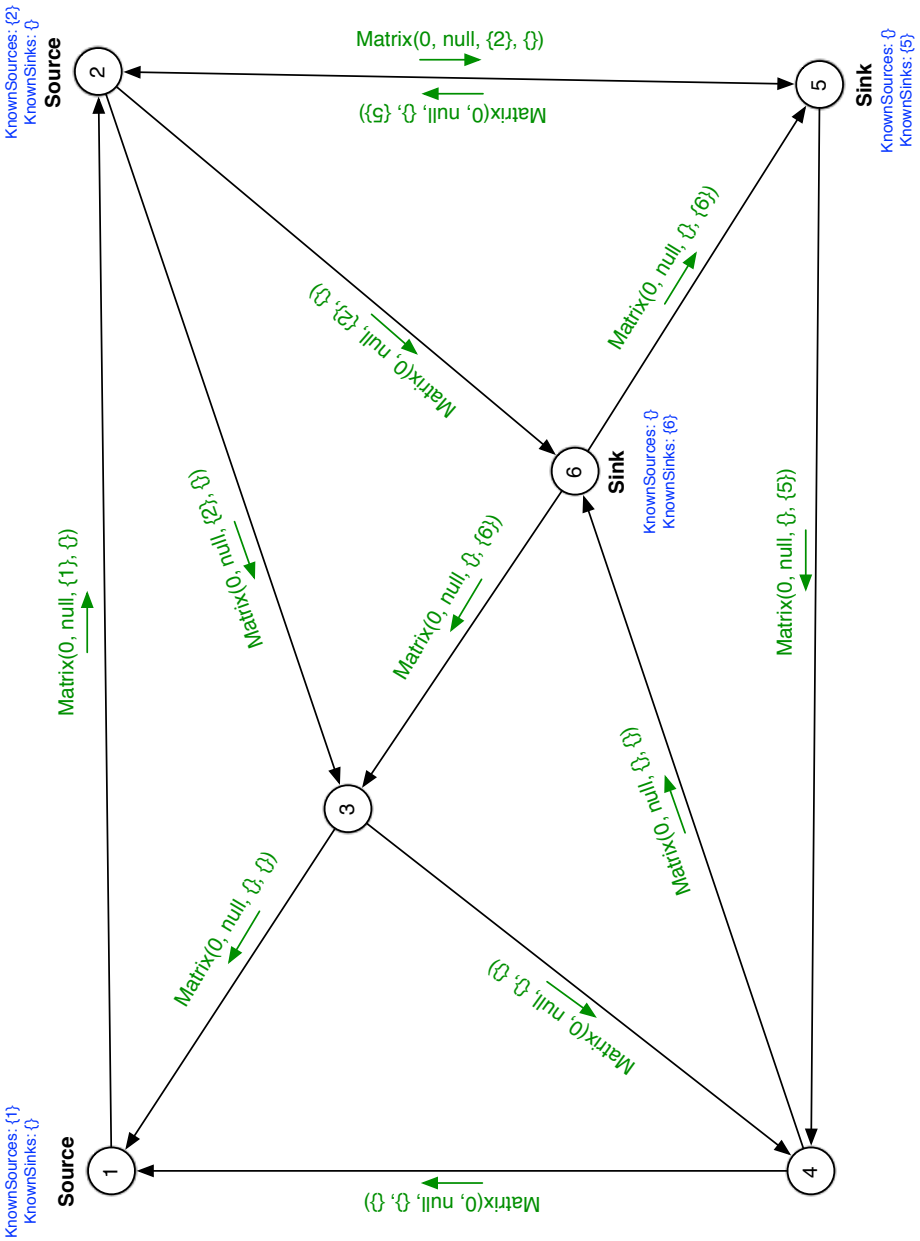


Figure 8.14: State of the network at time 0

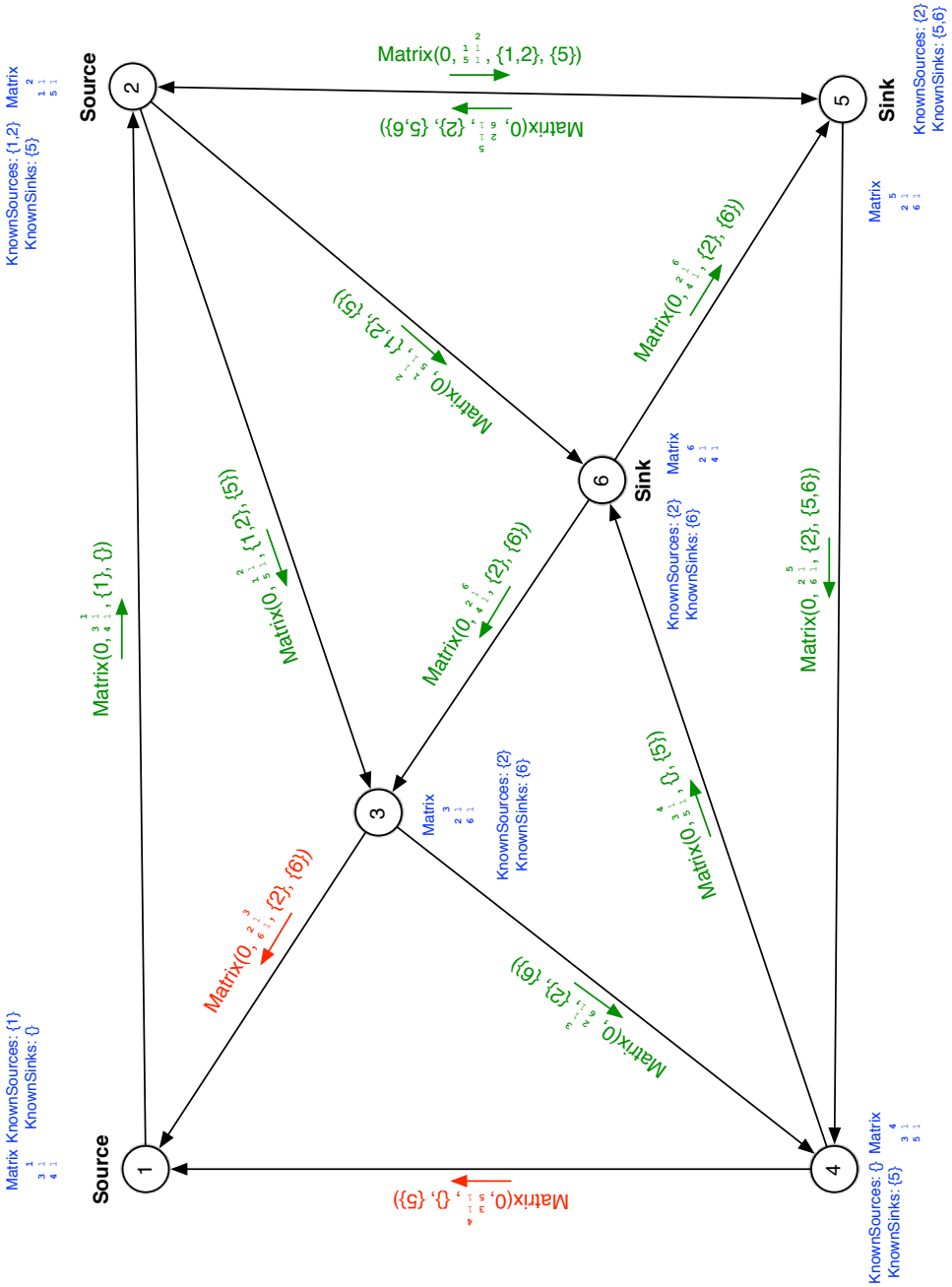


Figure 8.15: State of the network at time 1

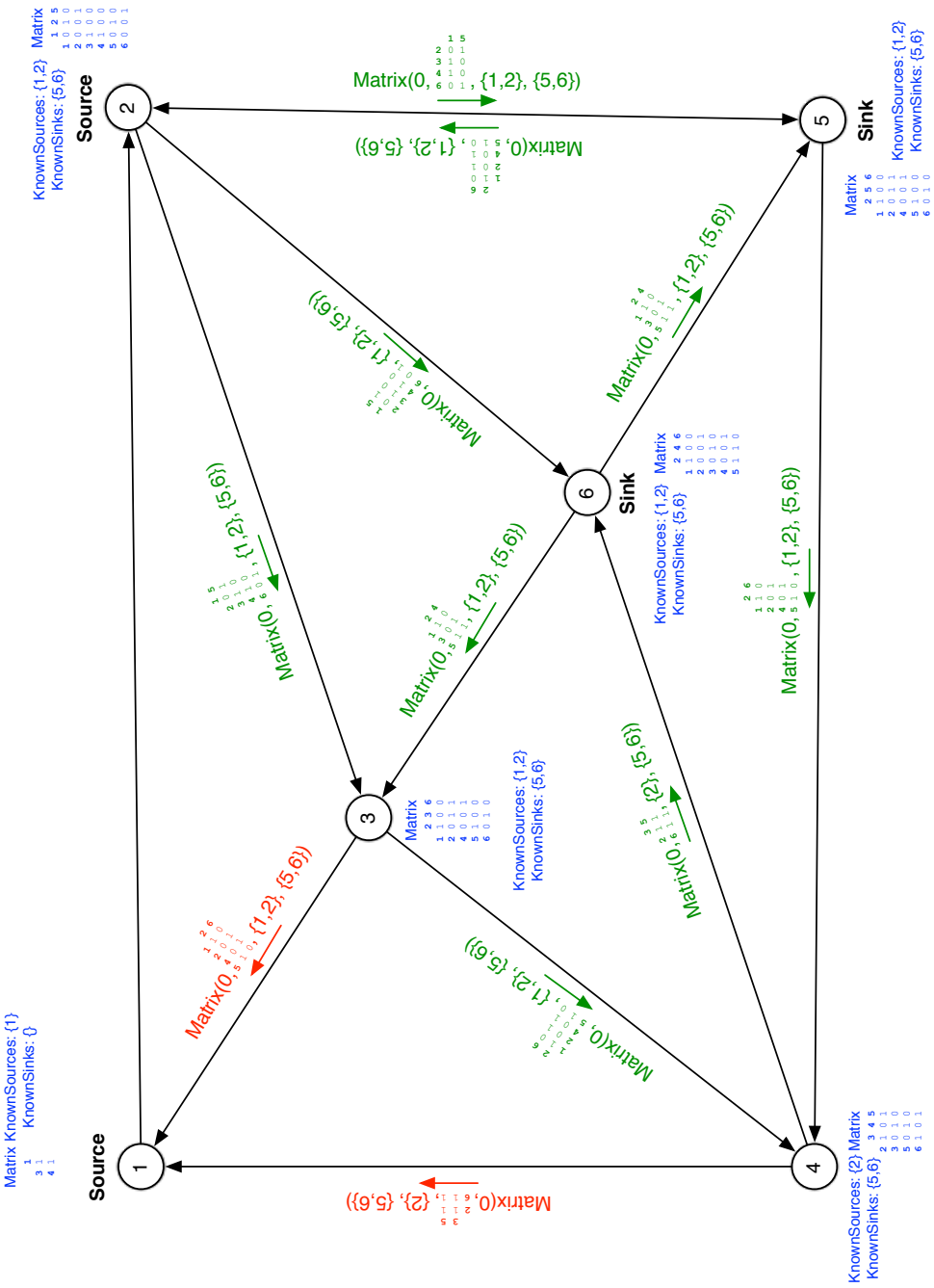


Figure 8.16: State of the network at time 2

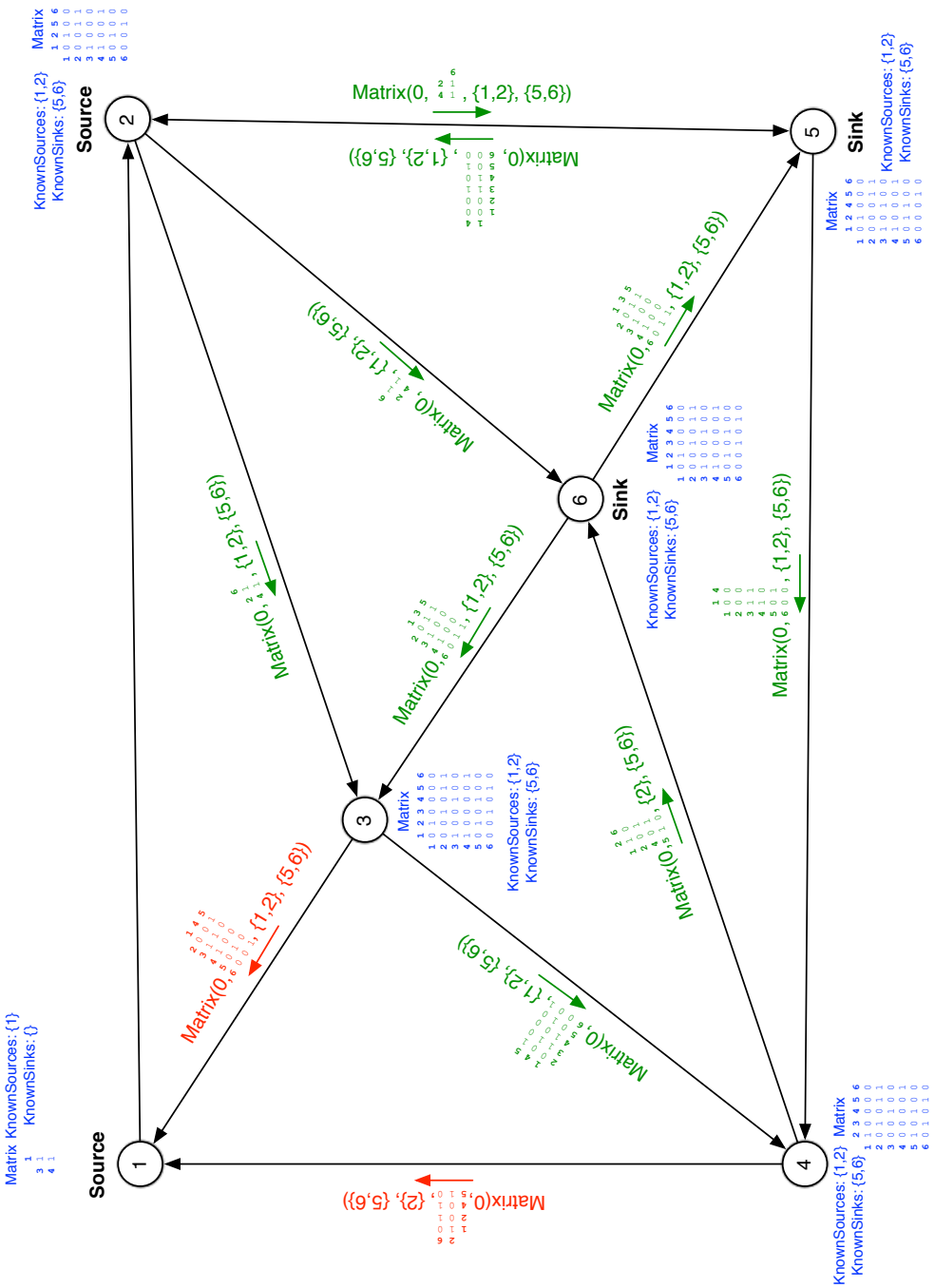


Figure 8.17: State of the network at time 3

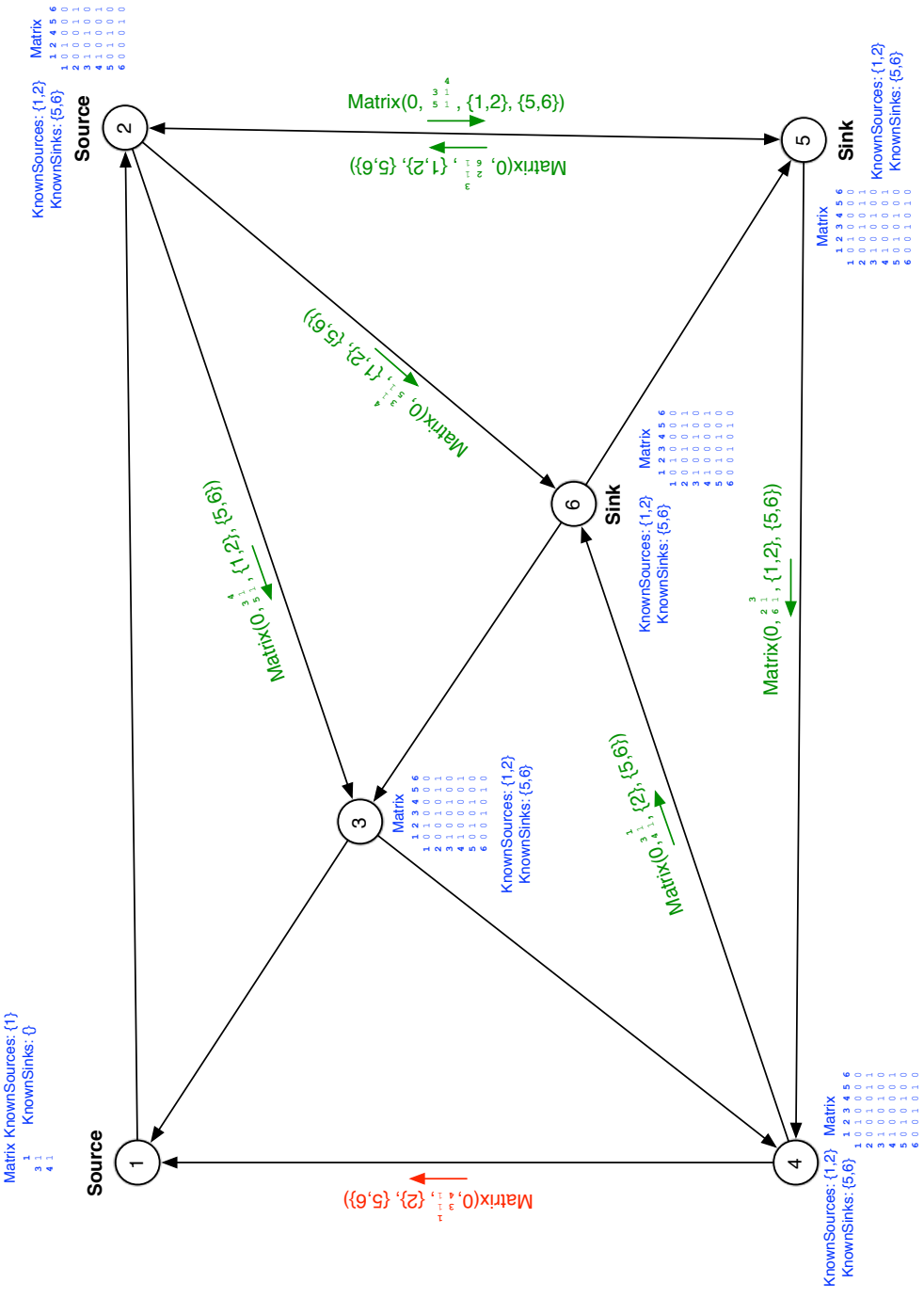


Figure 8.18: State of the network at time 4

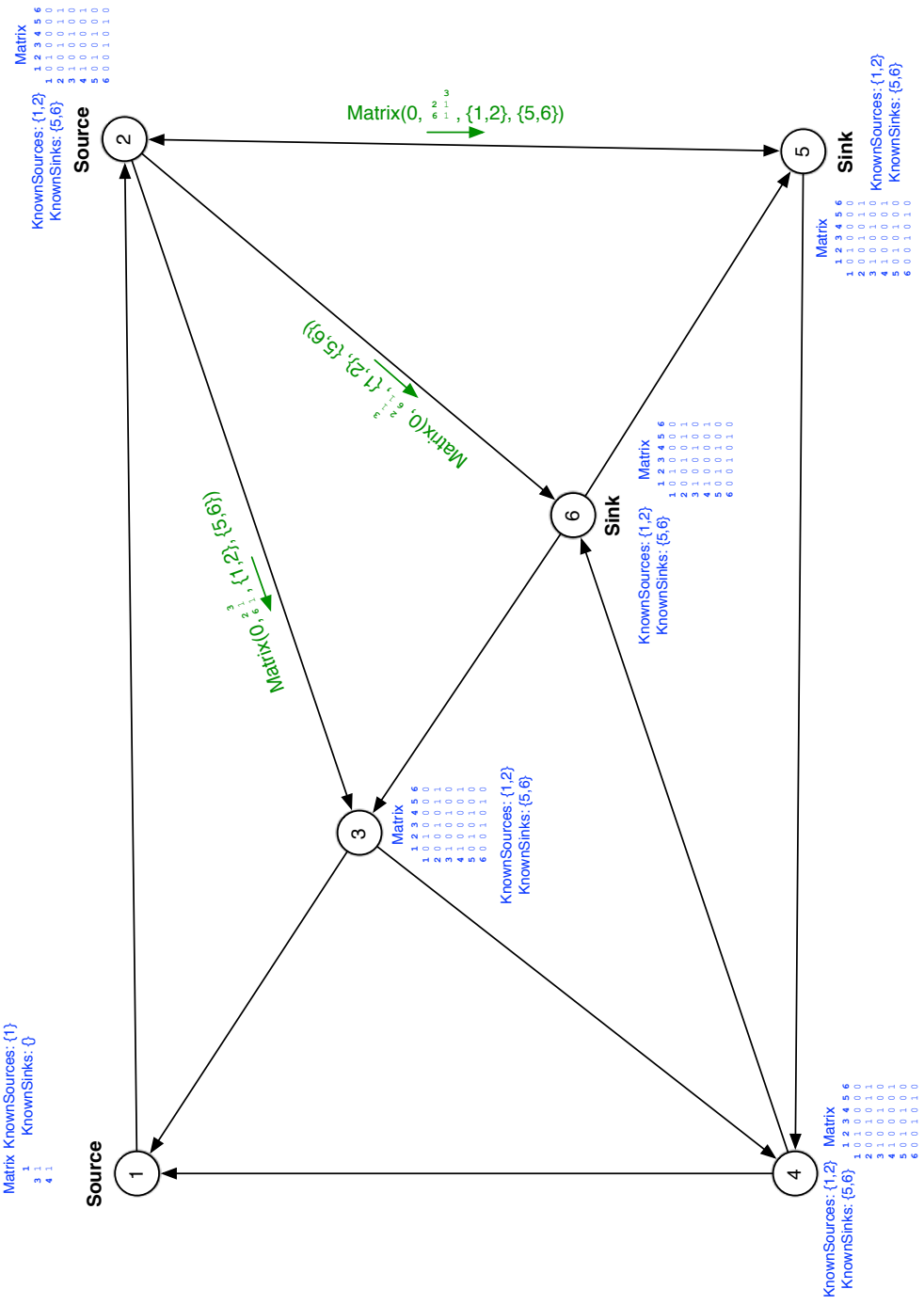


Figure 8.19: State of the network at time 5

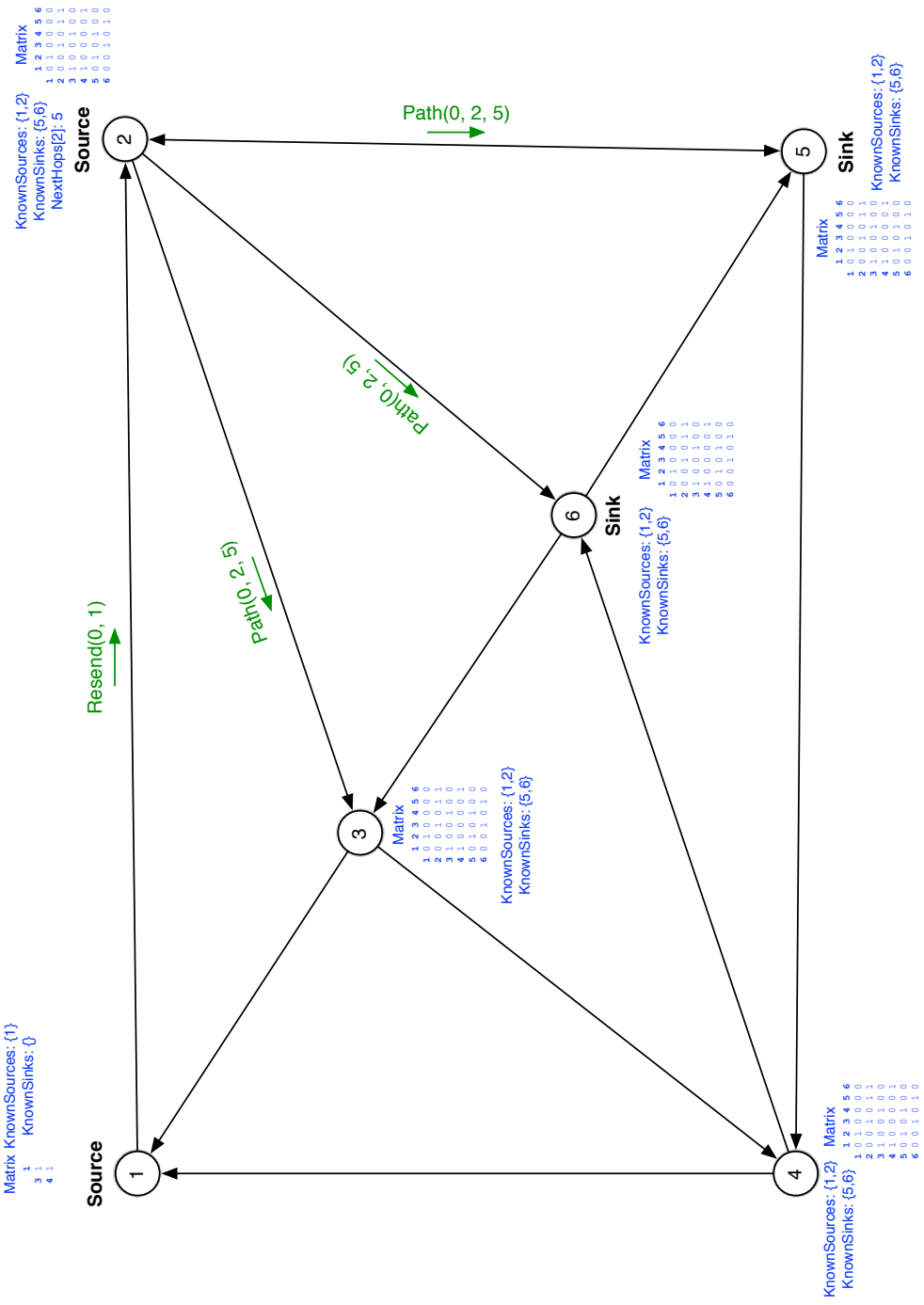


Figure 8.20: State of the network at time 10

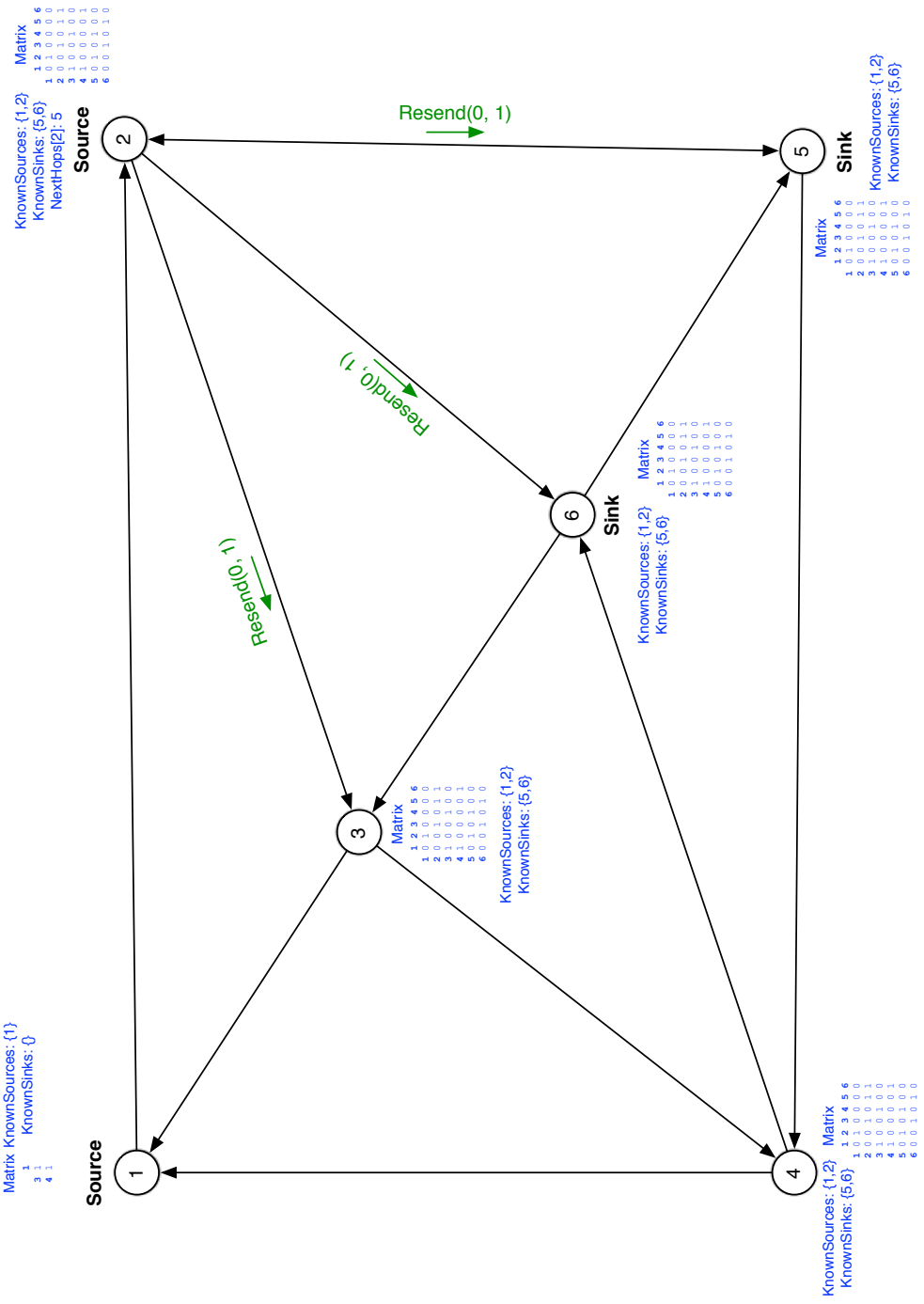


Figure 8.21: State of the network at time 11

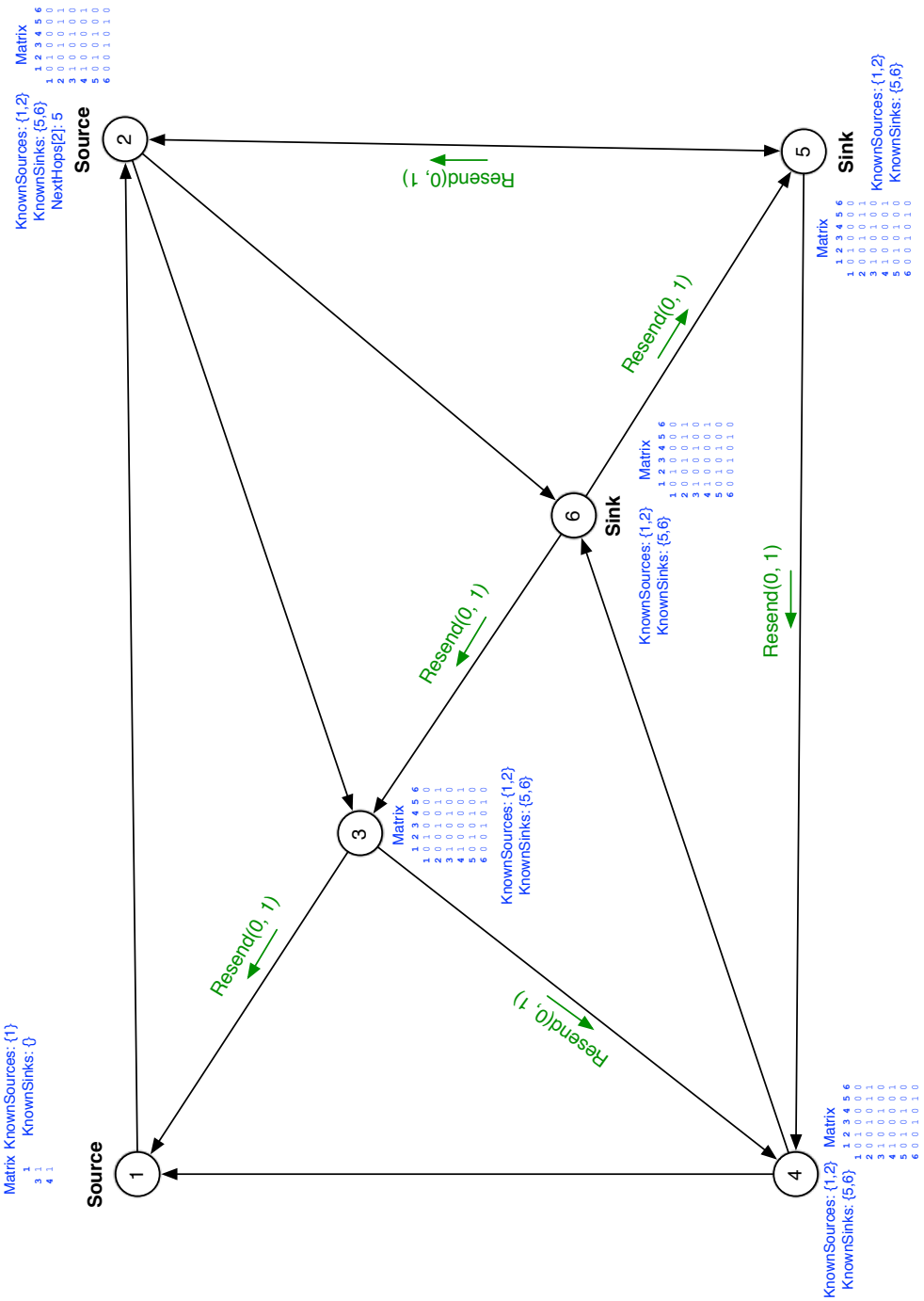


Figure 8.22: State of the network at time 12

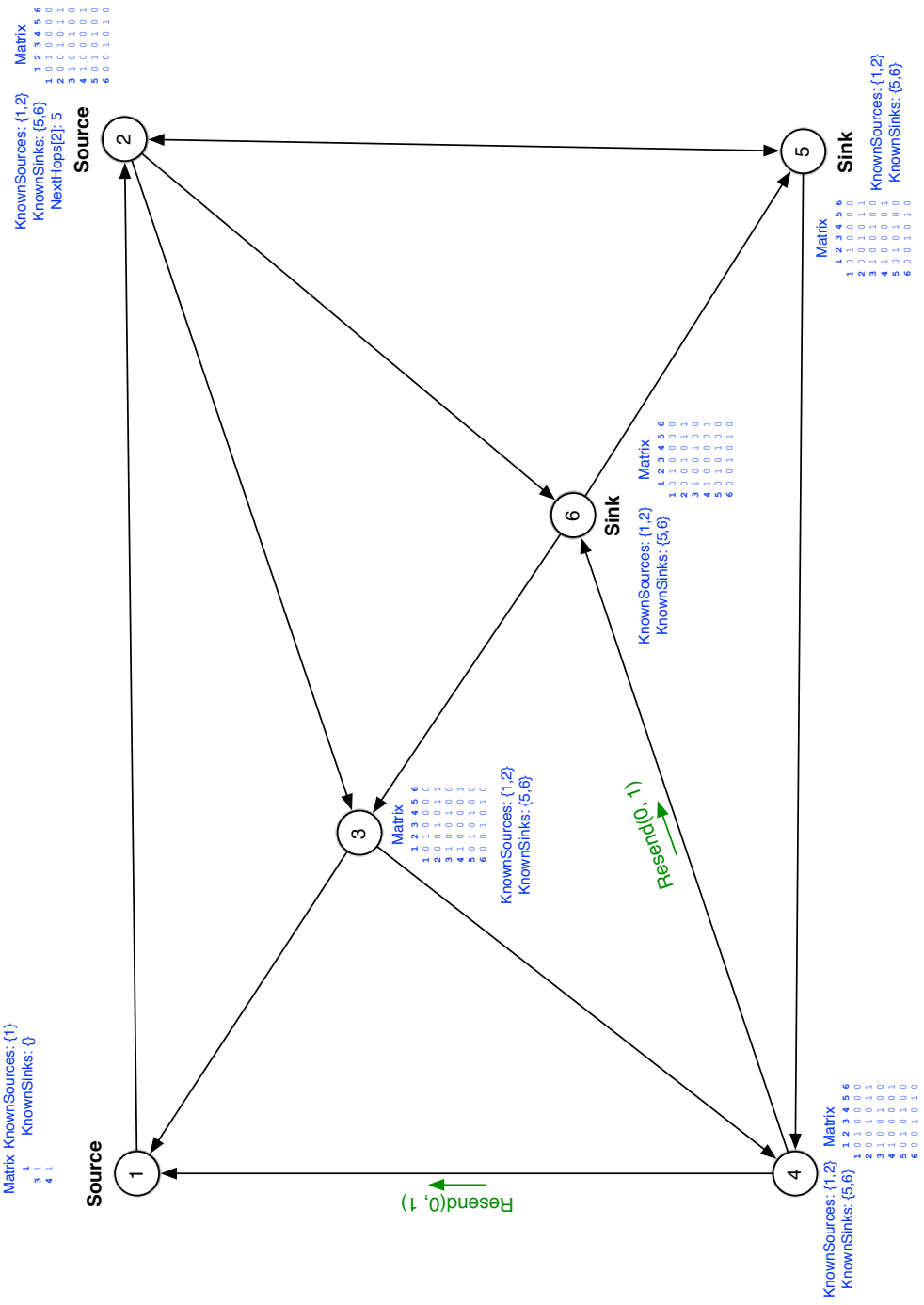


Figure 8.23: State of the network at time 13

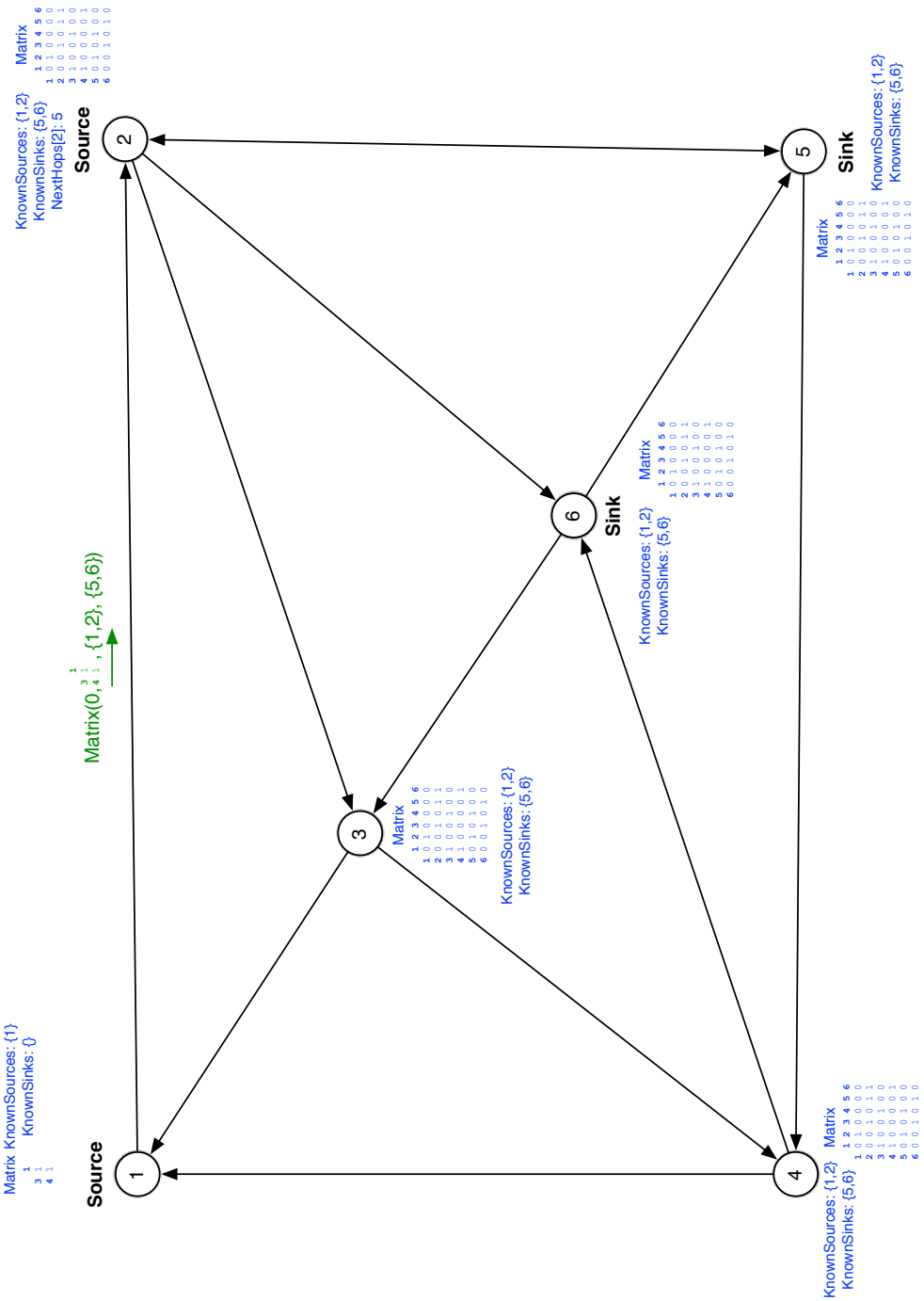


Figure 8.24: State of the network at time 40

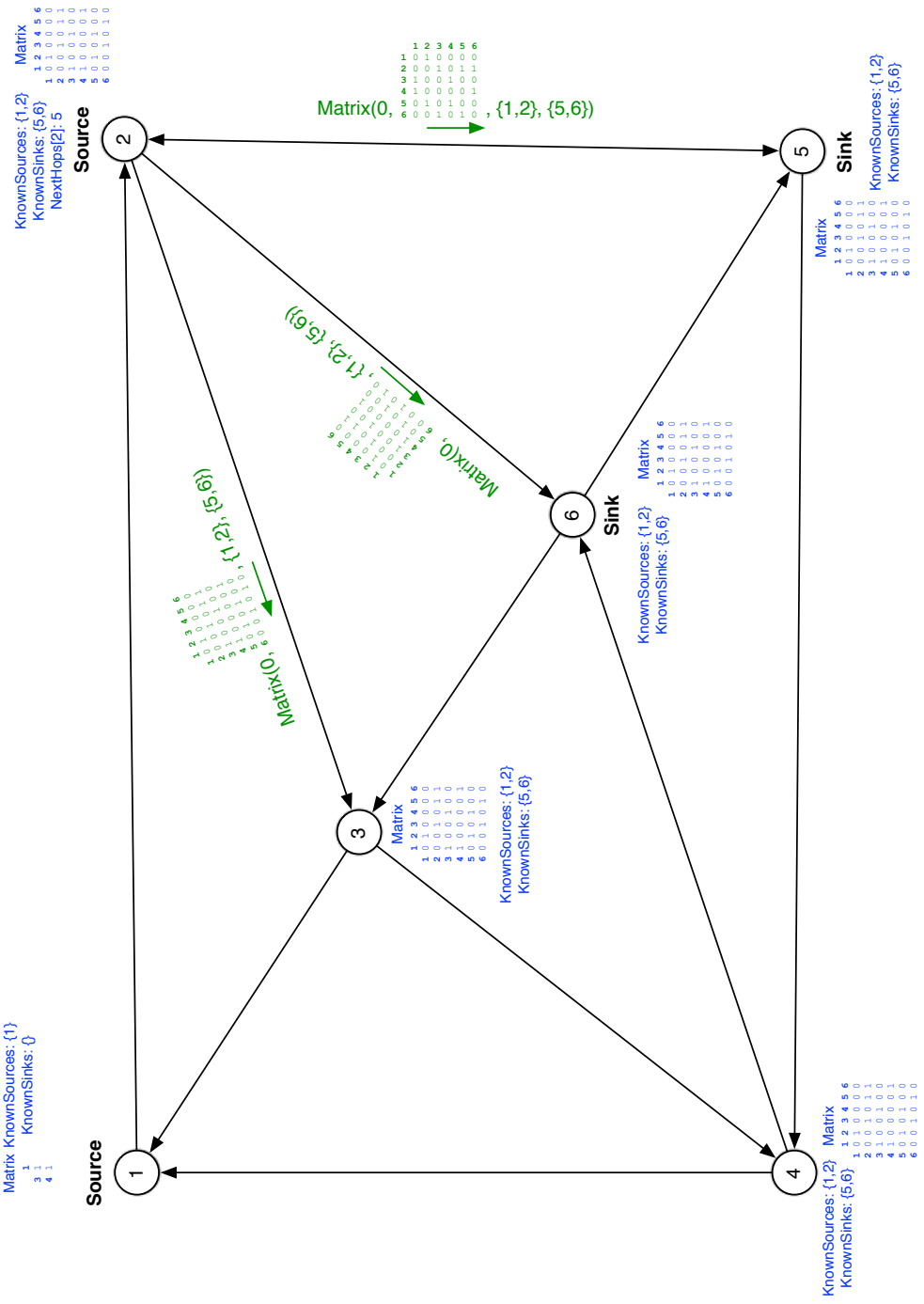


Figure 8.25: State of the network at time 41

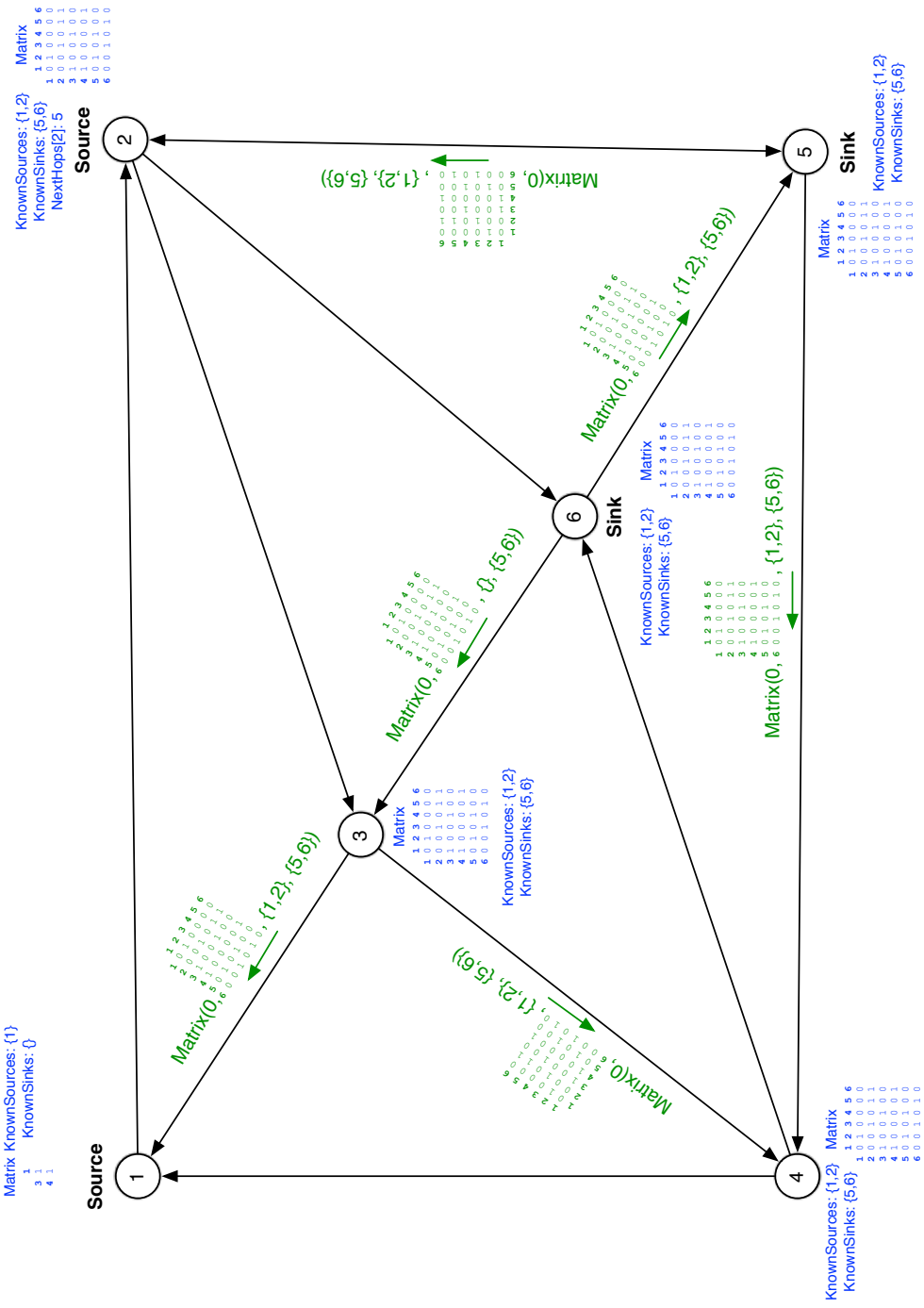


Figure 8.26: State of the network at time 42

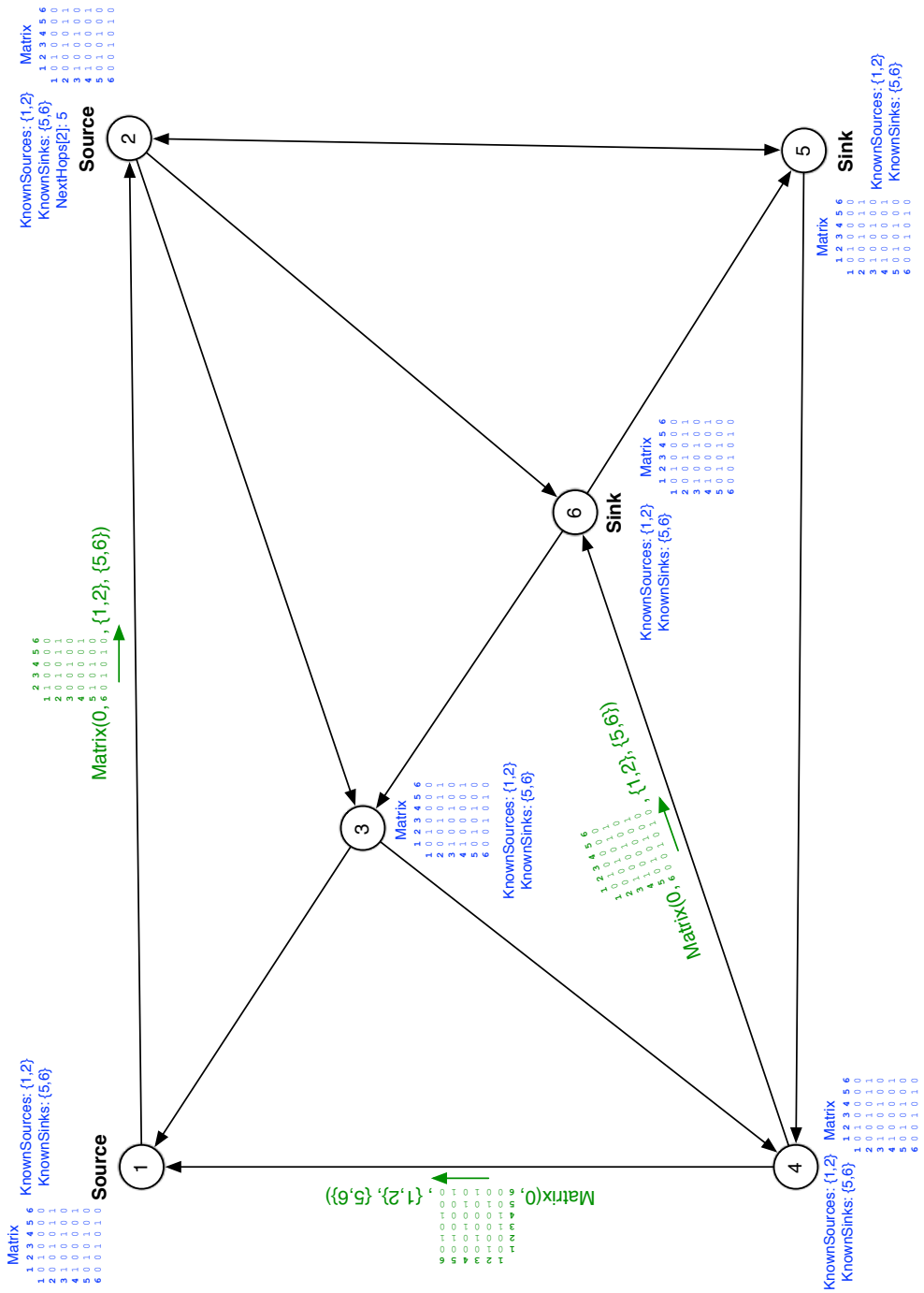


Figure 8.27: State of the network at time 43

8.3.2 Enforcement and Data Phase

The second phase of the global knowledge algorithm, the enforcement and data phase, begins after *gk-path-formation-time* elapse from when the node last began the topology-sharing phase and only if the node is able to form a path to a sink. During this phase, a source sends a message along the path that is to be used for sending data. At each node along the path, the node notes its next hop. Thus, when future Data messages are to be routed to a sink, it is only necessary to indicate the source from where the message originated. Each node along the path can forward the data closer to its destination.

8.3.2.1 Messages

During the enforcement and data phase of the global knowledge protocol, two types of message are exchanged, namely *Path* messages and *Data* messages:

Path messages are sent along the path to be used by a particular source. Each node on the path notes its next hop. Thus, when Data messages are to be sent, it is not necessary for the full routing path to be sent along with the data. Consequently, less data is transmitted and so the energy consumption of nodes is reduced. Path messages contain three fields:

- The network-wide sequence number as known by the source.
- The source from which the Path message originates.
- The remaining sequence of nodes that the Path message must travel through to reach the sink.

Data messages contain the data that has been collected by the source nodes. The messages are made up of four fields:

- The network-wide sequence number as known by the source.
- The source from which the Data message originates.
- The next node to be used in order to reach the sink.
- The data, as generated by the source.

8.3.2.2 Data Structures

The following data structures are used during the enforcement and data phase of the global knowledge routing protocol:

NextHops is a mapping of sources to nodes. Given a source, the NextHops data structure indicates the next node to use in order to reach the sink requested by that source.

8.3.2.3 Pseudo code

Below is the pseudo code for the enforcement and data phase of the routing protocol. The same code runs on each node. As well as the data structures and messages discussed above, the following functions are used within the code:

id() returns the unique identifier for the node.

role() returns the role of a node (i.e. if it is a source, or a sink).

delay(f, t) executes function f after t seconds. Calling delay on a function that is already delayed changes the time until that function is executed.

delayed(f) returns a boolean indicating whether function f is currently the subject of a delay() request.

sourceIsActive(n) is a function used by sink nodes to indicate that a source is currently active and routing data to the sink. If some period of time passes without a source being marked as active, the sink will reset the entire network by incrementing its value of SeqNum and broadcasting a new Matrix message.

```
1 boolean checkSeqNum(int seq) {
2    if (seq > SeqNum) {
3       SeqNum = seq
4       resetRoutingData()
5       beginPropagation()
6    } else if (seq < SeqNum)
7       return false
8
```

```
9     return true
10 }
11
12 On_Receive_PathMsg(int seq, Node[] path, int source) {
13    if (!checkSeqNum(seq) || path[0] != id())
14       return
15
16    Node[] newPath = path.removeIndex(0)
17    if (newPath.length > 0){
18       NextHops[source] = newPath[0]
19       sendPathMsg(seq, newPath, source)
20    }
21    else if (newPath.length == 0 && role() == sink)
22       sourceIsActive(source)
23 }
24
25 On_Receive_DataMsg(int seq, int source, int target, Data data) {
26    if (!checkSeqNum(seq) || target != id())
27       return
28
29    if (role() == sink)
30       sourceIsActive(source)
31    else
32       sendDataMessage(seq, source, NextHops[source], data)
33 }
```

checkSeqNum() behaves similarly to the function of the same name in the enumeration routing protocol and is called for any incoming message. The function returns a boolean indicating whether the incoming message should be discarded or not, depending on the sequence number. If the sequence number of the incoming message is higher (newer) then the node updates its own sequence number to that of the message, erases all its routing data and immediately restarts the topology-sharing phase again with the new sequence number by calling the *beginPropagation()* function.

On_Receive_PathMsg() deals with receiving a Path message. As with all messages, Path messages are discarded if they contain an old sequence number. However, they may also

be discarded if the receiving node is not the intended recipient of the message. The first element of the path is removed. If additional nodes lie on the path, the next hop for this source is stored in the node's NextHops data structure. The Path message is then rebroadcast and forwarded through the network with the first element being removed at each hop. When it arrives at the destination sink, the sink notes that the source node from where the Path message originates is active.

On_Receive_DataMsg() handles Data messages. As with Path messages, the Data messages are intended for a specific node. Consequently, if other nodes receive the message, they discard it. The message may also be discarded if it contains an old sequence number. If the receiving node is a sink, it notes that the origin of the message (the source) is still active. Otherwise, the node determines what the next hop is, in order to reach the destination for the particular source and forwards the message onwards.

8.3.2.4 Example

This section shows an example of the enforcement and data phase of the global knowledge algorithm. It is a continuation of the example from Section 8.3.1.4 and makes use of the same constants that were defined there. The example begins at time 50, which is *gk-path-formation-time*, i.e. 10 seconds after source 1 last began its topology-sharing phase.

At time 50, source 1 attempts to determine the cheapest path to the sink. The node is able to form the path 1:2:5. It removes its own ID from the path and stores the next hop for itself as node 2. It then broadcasts a Path message containing this truncated path, which is received by source 2 as shown in Figure 8.28.

At time 51, the Path message originating from source 1 is received by source 2. Since source 2 is the intended recipient, the message is kept. Source 2 removes its own ID from the front of the path contained within the message and records that the next hop for source 1 is node 5. Source 2 then rebroadcasts the Path message with the truncated path. Nodes 3, 5 and 6 receive the message. However, since node 5 is the intended recipient, nodes 3 and 6 will discard it. Being the last node on the path, node 5 notes that source 1 is active. Simultaneously, at time 51, source 1 determines that the cheapest path to the sink is the path 2:6. The node removes its own ID from the front of the path and notes that the next hop for itself is node 6. Source 2 then broadcasts a Path message containing the truncated

path. Again, nodes 3, 5 and 6 receive this Path message. However, nodes 3 and 6 disregard the message as they are not the intended recipients. Node 5 receives the message and notes that source 2 is active.

At the end of time 51, all the sources have an enforced path to a sink. Future Data messages can be sent, and the entire path need not be contained in the message, thus reducing energy consumption. If at some future time, a node expires and causes a loss of messages from a source for *data-loss-tolerance* seconds, the sink will erase any locally stored routing data, increment its SeqNum and transmit an empty Matrix message. Thus, all routing data in the network is erased and the first topology-sharing phase begins again.

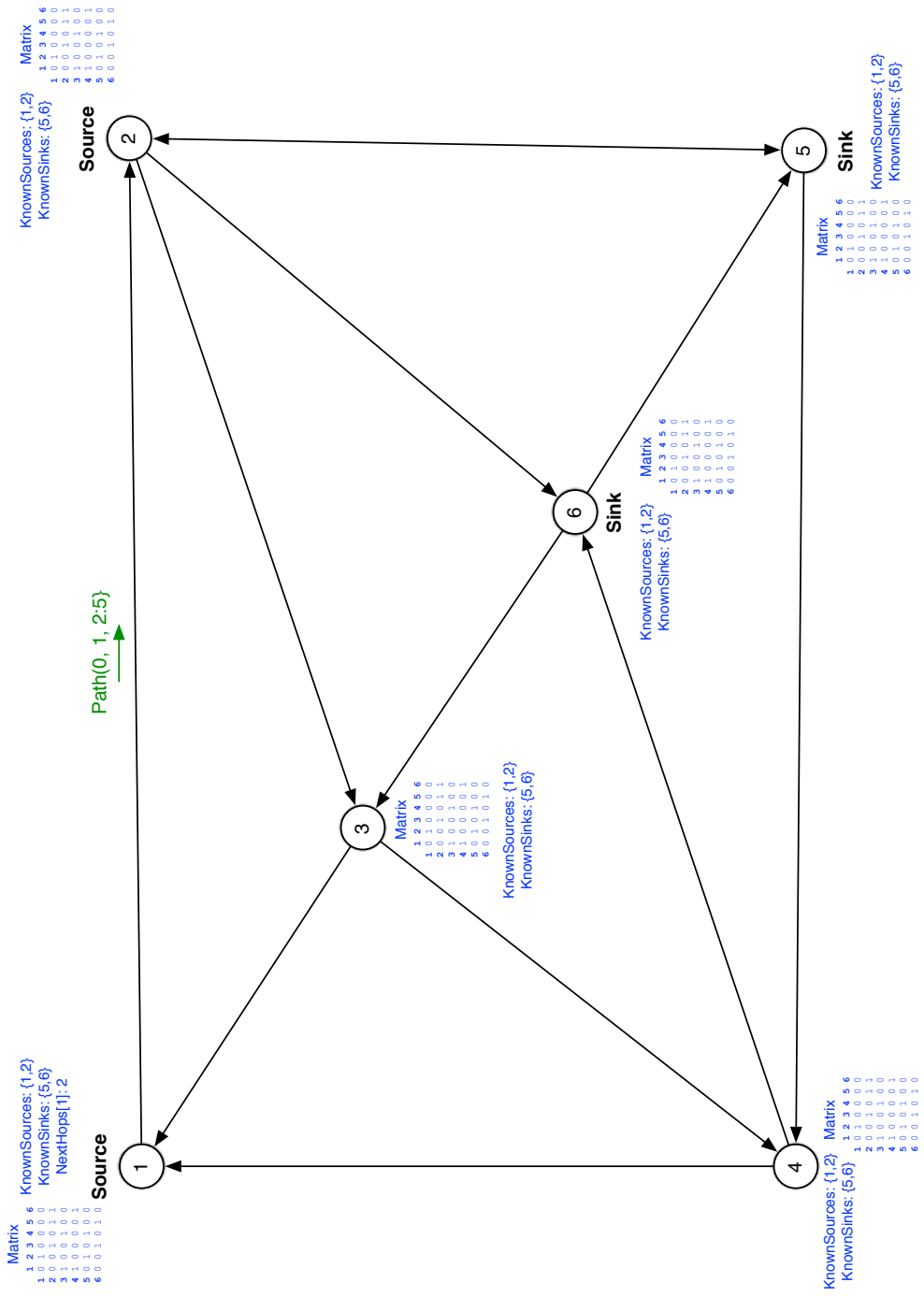


Figure 8.28: State of the network at time 50

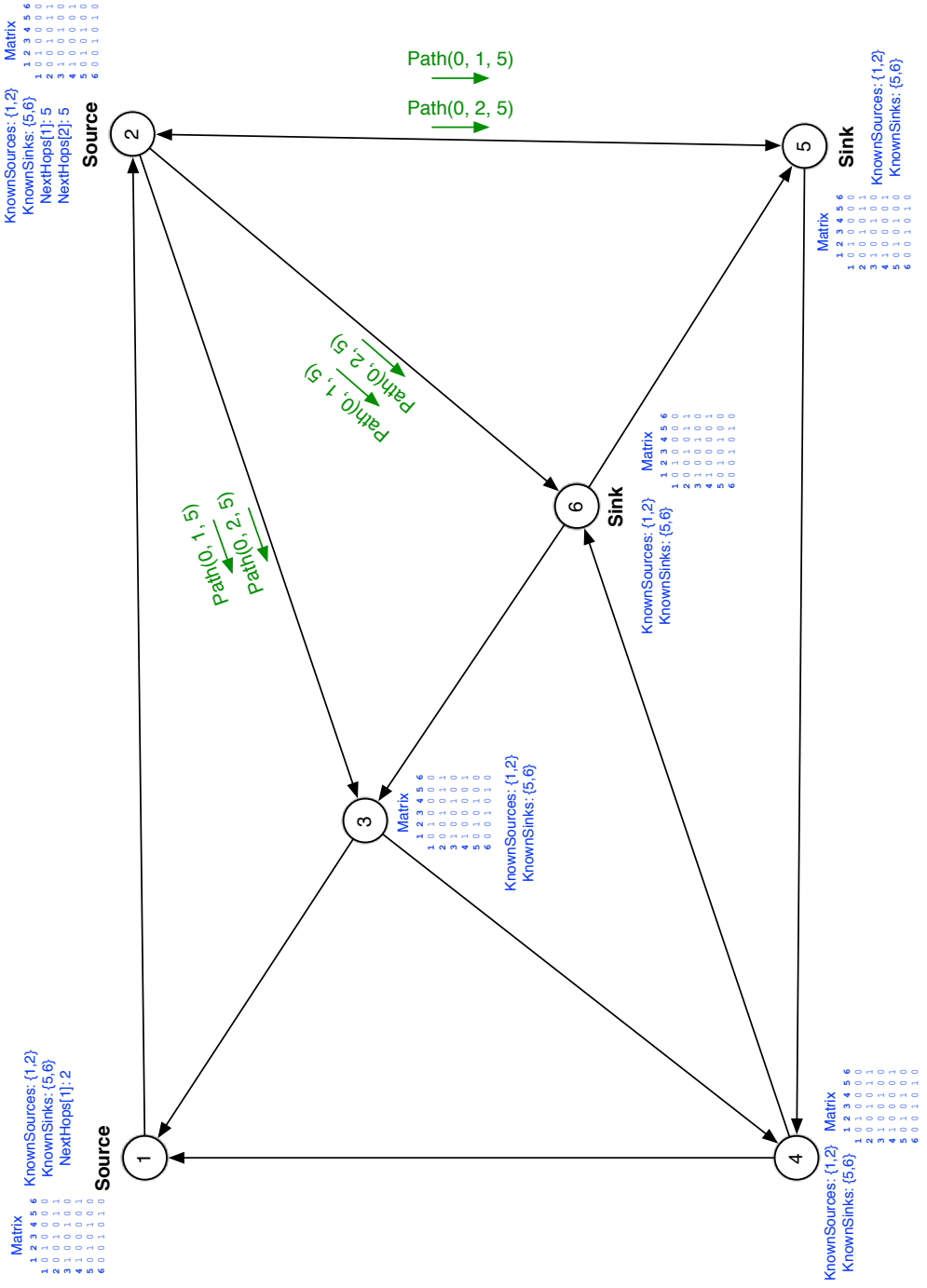


Figure 8.29: State of the network at time 51

8.4 Partial data

In the partial data routing protocol, paths are stored at intermediate nodes. When a large number of paths have been collected, they are amalgamated and forwarded along to the sink nodes. The sink nodes collect this data and then distribute it to the source nodes.

It is observed that since reliance values are based on the proportion of paths that a node lies on, it should be sufficient to calculate reliance values on a randomly selected subset of the full set of paths. For example, if a node lies on 50% of all paths S, then it should also lie on 50% of paths in C where $C \subset S$. Initial experiments were carried out using a purpose built Java simulator in which the number of known simple paths between a (source, sink) pair was incrementally increased and used to determine node reliance values. The graph in Figure 8.30 shows how the average error in node reliance value varies with the proportion of simple paths known.

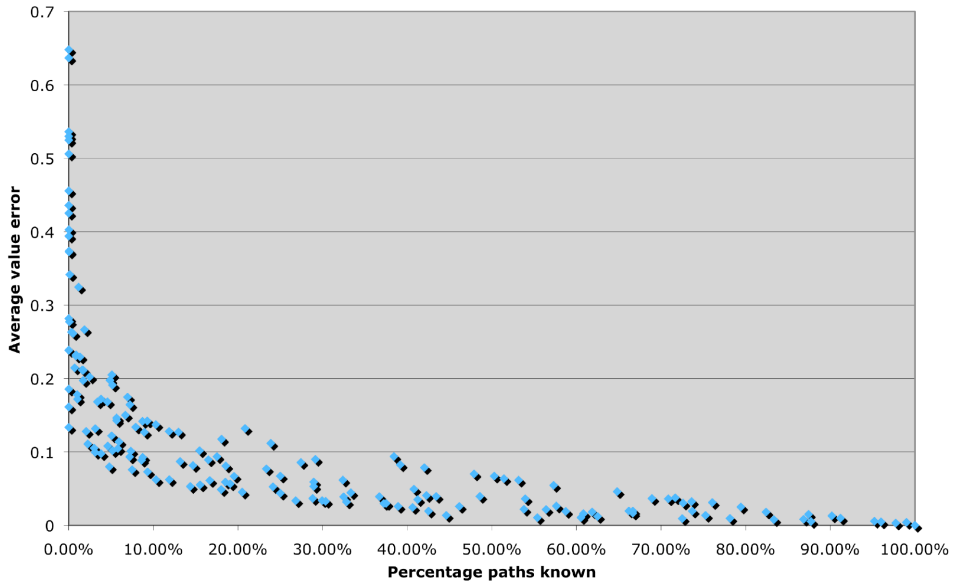


Figure 8.30: Mean node reliance error varies with the proportion of simple paths known

The graph shows that beyond approximately 20%, an increase in the number of simple paths known does not have an appreciable effect on the accuracy with which node reliance values can be estimated. Once approximately 50% of all simple paths are known, the average error in node reliance value is at most 0.05. These results demonstrate that while an increased set of simple paths provides more accurate estimates of node reliance, acceptable

values can be produced using a small random subset of simple paths.

However, there are difficulties in exploiting this fact. Firstly, the nodes that are most desirable to use for routing are those nodes that lie on very few paths. However, the probability that they will appear in a random sampling of all paths is also very small and so it is unlikely that these desirable nodes can be used. It is more likely that higher reliance nodes that lie on many paths will be discovered instead. Secondly, no method could be devised for selecting a path with a uniform degree of randomness (i.e. so that any path had an equal probability of being selected) without first calculating the set of paths. For example, Figure 8.31 shows a network with four simple paths. Source A has two next hops, B and C. However, the majority of paths are connected to node B rather than node C. To randomly select paths such that every path is equally likely to be selected, source A must be aware that 3/4 of all simple paths use node B and consequently send 3/4 of all Exploration messages via node B. This knowledge can only be achieved by firstly enumerating and analysing all paths.

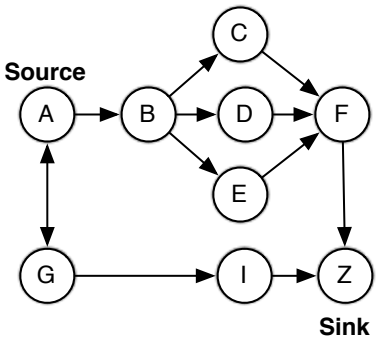


Figure 8.31: Many simple paths may travel through a single node

As a result of these difficulties, the concept of calculating reliance values using a limited set of paths was abandoned.

8.5 Summary

This chapter discusses three routing protocols for calculating node reliance, of which enumeration and global knowledge may be described as being functional.

The first is the enumeration routing protocol, which calculates the set of paths by sending

one message along each path of interest. The messages are accumulated at the sink nodes and flooded throughout the network, thus providing each source with the set of required paths between each (source, sink) pair.

The second algorithm is the global knowledge routing protocol. Rather than determine the set of required paths by sending one message along each path, the network topology data is gathered at each node and paths are enumerated there.

The third algorithm is the partial data routing protocol. Its aim is to store fragments of network topology in the network. During the course of sending data from sources to sinks, the topology data can be incrementally gathered. It is theorised that even if a small proportion of nodes were known, it is be possible to calculate reasonable estimates of the node reliance values. However, preliminary experiments indicate that the approach does not work as expected.

The remaining two routing protocols have been discussed and demonstrated in some depth. In Chapter 9, these routing protocols are compared with third party routing protocols with a perfect communications model to determine which one is more effective at maintaining (source, sink) connectivity for as long as possible. Similarly, in Chapter 10, the same routing protocols are compared in a realistic communications model.

Chapter 9

Routing Protocol Experiment (Perfect Physical Layer)

The experiment described in this chapter is used to analyse the effectiveness of the node reliance routing protocols in comparison with other published routing protocols in a network with a perfect model of communication. Unlike the experiment proposed in Chapter 7, nodes are required to expend energy in order to collect and maintain any network data that they require. The experiment reflects the result of the tradeoff between superior routing decisions made with increased knowledge versus the energy expended in order to get the required knowledge for the routing protocol to operate.

The experiment considers the following routing protocols:

- Lexicographically ordered simplest paths gathered by global knowledge
- Shortest path ordered simplest paths gathered by global knowledge
- Lexicographically ordered simplest paths gathered by enumeration
- Shortest path ordered simplest paths gathered by enumeration
- Lexicographically ordered contracted simplest paths gathered by global knowledge
- Shortest path ordered contracted simplest paths gathered by global knowledge
- Minimum hop path gathered by global knowledge

- The EnergyAware heuristic combined with global knowledge
- The EnergyAware heuristic combined with a modification of DSR [54]
- Random walk routing [113]
- Multipath on-demand routing (MOR) [10]

Minimum hop routing is included due to the frequency with which it is analysed in the literature.

The EnergyAware heuristic can be modified into a routing protocol by considering two data collection mechanisms. The first is Global Knowledge (GK), as discussed in Chapter 8. The second is based on Dynamic Source Routing (DSR) [54], which is an efficient means of finding shortest cost paths from sources to sinks [14] and does not rely on bidirectional links between pairs of nodes. This DSR-like mechanism operates in two phases. In the first phase, messages are flooded from sources to sinks. Each message includes the ID of each node through which it has passed. For a particular request, the request message is rebroadcast by intermediate nodes provided that it is the cheapest (or only) path encountered by that node. When the request is received by a sink, or an intermediate node with a path to a sink, the (source, sink) path is flooded throughout the network. For implementation specifics, the reader is directed to the source code of the routing protocol, which may be found as part of the archive at http://sourceforge.net/projects/wsnnodereliance.

In random walk routing [113], as a message is routed from a source to a sink, a random neighbour is selected at each hop. This routing protocol is included for comparison since it achieves load balancing with a very small overhead; each node only requires knowledge of the immediate neighbourhood.

The MOR protocol [10] achieves load balancing by making use of all shortest paths between a source and sink. This differs from random walk routing in that each node only forwards data to a random neighbour on a shortest path to a sink rather than any random neighbour. However, the additional knowledge increases the overhead of the routing protocol, as paths must be actively discovered.

9.1 Parameters

The parameters of the experiment are the same as for those carried out on routing heuristics, described in Section 7.1, with the exception of the data generation rate, which is fixed at 5 seconds. They are summarised below:

- Number of nodes (n)
  - 2, 5, 10, 20, 40
- Number of sources
  - 1, 2, 5, 10, 20, $n-10$, $n-5$, $n-2$, $n-1$
- Number of sinks
  - 1, 2, 5, 10, 20, $n-10$, $n-5$, $n-2$, $n-1$
- Node placement:
  - Regular triangular grid
  - Regular square grid
  - Regular hexagonal grid
  - Uniform random

For an explanation and justification of these parameters, the reader is directed to Section 7.1. A randomised data rate is not considered in this experiment, since it is shown to have little effect on the performance of routing heuristics in Chapter 7.

9.2 Procedure

The experiment proceeds as follows:

1. A network is generated by using a purpose built Java program, subject to the parameters discussed in Section 9.1.

2. Sources and sinks are randomly selected and are stored as part of a network configuration file. Each network configuration operates with a different set of random seeds.

3. The simulation is run once on each network configuration for each routing protocol.

4. Data generation starts 60 seconds after the activation of nodes, thus allowing each source a period of time with which to form routes to sinks.

5. Sources generate data of random size (between 0 and 100 bytes) every period of time and route it to one of the sink nodes.

6. The process continues until all source nodes have expired.

7. Sources are permitted to expire at any time if they determine that routing to a sink is impossible.

8. Measurements discussed in Section 9.3 are taken during the simulation.

Nodes generate data periodically with a constant data rate of 5 seconds. A periodic data rate is selected since this is common in most deployments studied in Section 2.4. The data rate is considered to be as fast as a typical WSN application is likely to generate and route data. Faster data rates are likely to be handled by causing the node to buffer data for a period of time, compress it and send it in chunks rather than streaming it.

The *data-loss-tolerance*, i.e. the number of seconds that can elapse without hearing from a source before a sink assumes the source has been disconnected, is set to 120 seconds. This value is arbitrarily selected such that it is bigger than the estimated maximum time required to form a route between a source and sink and is kept constant in all routing protocols.

9.3 Measurements

The same measurements specified in Chapter 7 are also used in this experiment.

The node reliance routing protocols are expected to perform well as measured by the CWT metric. However, since they are designed with realistic communications in mind, they are not able to take advantage of network features, such as bidirectional edges, that are present

Constant	Value
GK-PATH-FORMATION-TIME	45
GK-MATRIX-RESEND-ATTEMPTS	3
GK-BEACON-DELAY	1
GK-RESEND-DELAY	15

Table 9.1: Values of constants in the global knowledge routing protocols

in a perfect network. Thus, other routing protocols that rely on these features, such as MOR, may perform better.

9.4 Requirements

As with the experiment in Chapter 7, sinks are given a near infinite supply of energy and other nodes are given 14.58J of energy.

Node reliance routing protocols are able to split up large messages that contain adjacency matrices into several smaller messages to help with transmission. In this experiment, a large message is considered to be any message whose size exceeds 30 bytes. This value is determined based on the results of preliminary experiments.

The variables shown in Table 9.1 are used for routing protocols that gather topological data using global knowledge.

Based on the results of preliminary experiments, GK-BEACON-DELAY is set to one second, which is sufficiently long for amalgamation of data to take place without large packets being consistently generated.

The worst-case scenario for GK-PATH-FORMATION-TIME is that n transmissions must be made by each node in order for the topology data to be transmitted through the network, where n represents the number of nodes in the network. Since each node has a beacon delay of one second, a path formation time of 45 seconds seems appropriate. The extra five seconds are added in order to provide network synchronisation, for example, to ensure that all nodes reset or are activated before path formation begins.

Constant	Value
ENUM-PATH-FORMATION-TIME	45
ENUM-EXPLORE-REPLY-DELAY	25
ENUM-ROUTE-REPLY-COLLECTION-TIME	45
ENUM-PATH-RETRY-DELAY	15
ENUM-MATRIX-RESEND-ATTEMPTS	3

Table 9.2: Values of constants in the enumeration routing protocols

GK-RESEND-DELAY is set to 15 based on the results of preliminary experiments. Too high a value will result in large amounts of data being lost while the source waits to re-request a route. Too low a value will increase the probability of collisions and will not give the network a chance to reduce its traffic before another request is made.

The GK-MATRIX-RESEND-ATTEMPTS value is arbitrarily selected in order to emulate a particular application scenario. The application will attempt to form a route three times. If no route can be formed during this time, any route that might exist is deemed to be too unreliable to expend energy on forming. For example, even if a route can be formed after 100 attempts, the route is likely to result in high message loss if used. Consequently, it would not be long before the sink lost contact with the source and the path had to be reformed, especially considering that a message containing data is likely to be bigger than a message used in path formation and so is less likely to be successfully transmitted.

Routing protocols that enumerate paths use the variables shown in Table 9.2

In order to keep the enumeration based algorithms consistent with the global knowledge based algorithms, ENUM-PATH-FORMATION-TIME is set to 45 seconds, ENUM-PATH-RETRY-DELAY is set to 15 seconds and ENUM-MATRIX-RESEND-ATTEMPTS is set to 3.

The ENUM-EXPLORE-REPLY-DELAY value is set to 25 seconds based on the results of preliminary experiments.

ENUM-ROUTE-REPLY-COLLECTION-TIME is set to 45 seconds and is based on the fact that a source with ID 40 will not begin path discovery for 20 seconds (0.5 x 40) after source 0. Since a sink must also wait for at least 25 seconds before sending an ExplorationReply message, there could be at least 25 + 20 = 45 seconds between the first and last

Constant	Value
ENERGYAWARE-PATH-FORMATION-DELAY	5
ENERGYAWARE-RREQ-PERIOD	15

Table 9.3: Values of constants in the EnergyAware routing protocol

Constant	Value
MOR-PATH-FORMATION-DELAY	5
MOR-RREQ-PERIOD	15
MOR-MAX-FAILURES	3
MOR-MSG-CONFIRMATION-TIME	2

Table 9.4: Values of constants in the MOR routing protocol

ExplorationReply messages from sinks. It is possible that the delay will be greater than this value. However, path formation cannot begin until the data from all sinks has been collected. Consequently, setting this value higher could result in data loss.

The EnergyAware routing protocol uses the constants shown in Table 9.3

ENERGYAWARE-PATH-FORMATION-DELAY is equal to the data generation period of 5 seconds. Consequently, path formation is given as long as possible while trying to avoid data loss. It is possible to set such a low value because of the small number of messages that are exchanged in the routing protocol.

ENERGYAWARE-RREQ-PERIOD refers to the delay experienced before path formation begins after failing and is 15 seconds. The value is derived based on the time that it would take every other source (maximum 38) to send a message that is flooded through every node (40 in total) and is flooded back. Assuming that each message requires 0.0004 seconds to transmit as discussed in Section 5.4.3, the total time required would be 12.16 seconds. The value is rounded up to 15 seconds.

In MOR, the constants in Table 9.4 are used.

MOR-PATH-FORMATION-DELAY and MOR-RREQ-PERIOD are set the same as for the EnergyAware routing protocol.

MOR-MAX-FAILURES represents the number of times a node may fail to forward a message and is set to 3, as used in the description of MOR [10].

MOR-MSG-CONFIRMATION-TIME refers to the length of time permitted for a node to acknowledge receipt of a message. The value is set to 2 seconds based on preliminary experiments.

Finally, the only variable in random walk routing is the period with which each node sends a beacon message in order to alert its neighbours to its presence. The value is set to 120 to coincide with the data loss tolerance.

9.5 Results

Table 9.5 shows the results of the experiment for each routing protocol. The CWT and total data transfer columns show the average normalised CWT and total data transfer measurements across all simulations. Since the experiment in Chapter 7 indicates that heuristics using shortest path achieve a higher CWT and total data transfer than those that use lexicographic ordering, the lexicographically ordered routing protocols are not discussed here. However, the experiment that is carried out confirms the findings of the earlier chapter, i.e. the CWT and total data transfer scores are lower with lexicographic ordering than with shortest path. The results that are achieved with those routing protocols support the explanations of results that are given in this chapter. The reader is directed to the archive at http://sourceforge.net/projects/wsnnodereliance for verification or further details.

It is possible to immediately make certain observations regarding the overall results. Firstly, the standard deviation of each routing protocol is at most 0.55% of the measured value for CWT and 0.41% of the measured value for total data transfer, suggesting that the measurements are accurate.

Secondly, routing protocol performances are more spread out in this experiment than they are in the experiment of Chapter 7. For example, the best performing routing protocol (EnergyAwareGK) performs 95.6% better than the worst routing protocol (RandomWalk).

The following subsections deal with a few further observations made in this experiment. Section 9.5.1 discusses the poor performance of RandomWalk which under-performs every

Name	Heuristic	Discovery	CWT		Transfer	
			Value	SD	Value	SD
EnergyAwareGK	EnergyAware	GK	0.988	0.02%	0.983	0.03%
ContractedGK	ContractedShortest	GK	0.985	0.04%	0.986	0.03%
SimplestGK	SimplestShortest	GK	0.984	0.04%	0.985	0.03%
MinHopGK	MinHop	GK	0.972	0.05%	0.983	0.04%
EnergyAwareDSR	EnergyAware	DSR-like	0.961	0.04%	0.961	0.04%
MOR	MinHop	DV	0.946	0.07%	0.963	0.05%
SimplestEnum	SimplestShortest	Enum	0.932	0.18%	0.956	0.13%
RandomWalk	RandomWalk	Local	0.505	0.55%	0.591	0.41%

Table 9.5: Normalised CWT and total data transfer metrics for each routing protocol with a perfect radio model

other routing protocol by a significant margin. Sections 9.5.2 and 9.5.3 discuss the effect on CWT and total data transfer of changing the proportion of sources in the network. Section 9.5.4 discusses the effect of altering the number of sources in the network.

9.5.1 Poor Performance of Tian's RandomWalk Routing Protocol

RandomWalk [113] performs significantly more poorly than every other routing protocol. The most likely explanation for this poor performance is that there are no limits to the number of hops that a message might travel through and a message may even travel through the same node more than once. Consequently the energy expended by the network for routing a single message may be unlimited.

This theory can be verified by considering the number of successful transmissions made per message received by the sink. If such a ratio is particularly high for RandomWalk, it indicates that a disproportionately high number of transmissions take place in the network for each message received. Consequently, the capacity of the network will be smaller, due to the increased energy expended in transmitting extra messages, and the time for which sources can remain connected to sinks will also be reduced.

Table 9.6 shows, for each routing protocol, the normalised number of successful transmis-

sion made for each message received by the sinks. RandomWalk has a significantly higher ratio than all but one of the other routing protocols. Thus, for each message routed to the sink, RandomWalk expends significantly more energy than most of the other routing protocols.

Name	Heuristic	Discovery	Transmissions/Msg	SD
MOR	Min Hop	DV	0.893	0.22%
RandomWalk	RandomWalk	Local	0.667	0.39%
SimplestEnum	SimplestShortest	GK	0.333	0.60%
EnergyAwareDSR	EnergyAware	DSR-like	0.301	0.61%
SimplestGK	SimplestShortest	GK	0.285	0.61%
ContractedGK	ContractedShortest	GK	0.280	0.62%
SimplestGK	SimplestShortest	GK	0.278	0.62%
MinHopGK	Minimum hop	GK	0.277	0.63%
EnergyAwareGK	EnergyAware	GK	0.277	0.63%

Table 9.6: Normalised average transmissions per message received at the sink

The high ratio of the MOR protocol is due to an inflation of the number of successfully received transmissions. A transmission is considered to be successfully received if the intended recipient receives the node. However, in MOR, a message is sent from a node A with two intended recipients. The first is the next hop from A along the path to the sink. The second is the node from which A received the message. By using two recipients, MOR is able to confirm that a message has been successfully received and forwarded to the next hop with a single message. Thus, each node is the receiver of twice as many messages.

RandomWalk under-performs other routing protocols by such a margin that it is impractical to include it on many of the graphs shown in this section. Where possible, the protocol has been included. In other cases, the inclusion of RandomWalk makes other points unreadable and is therefore omitted.

9.5.2 CWT and Increasing Proportions of Sources

The graphs shown in Figures 9.1, 9.2 and 9.3 show the effect on CWT of increasing the number of sources in networks of 10, 20 and 40 nodes respectively. Each graph considers

the average CWT across all topologies (random, triangular, square and hexagonal).

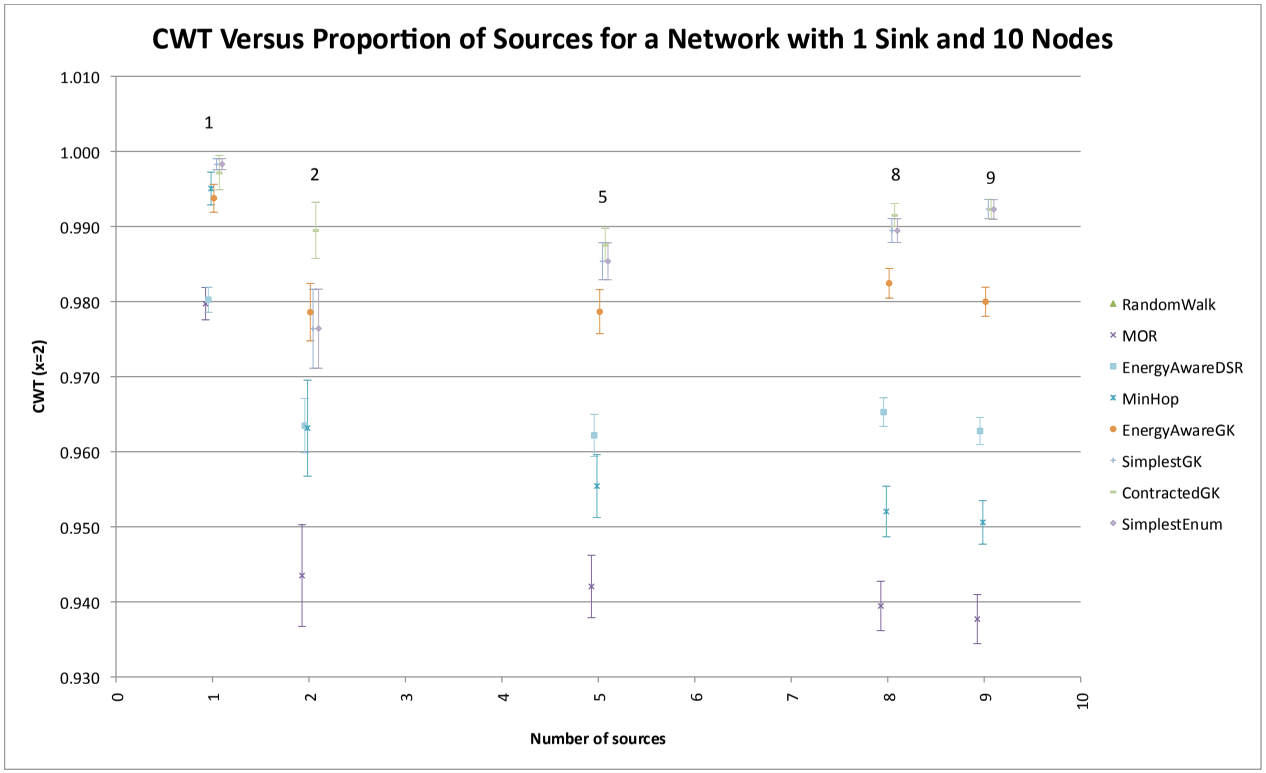


Figure 9.1: CWT vs. proportion of sources for networks with 10 nodes and 1 sink

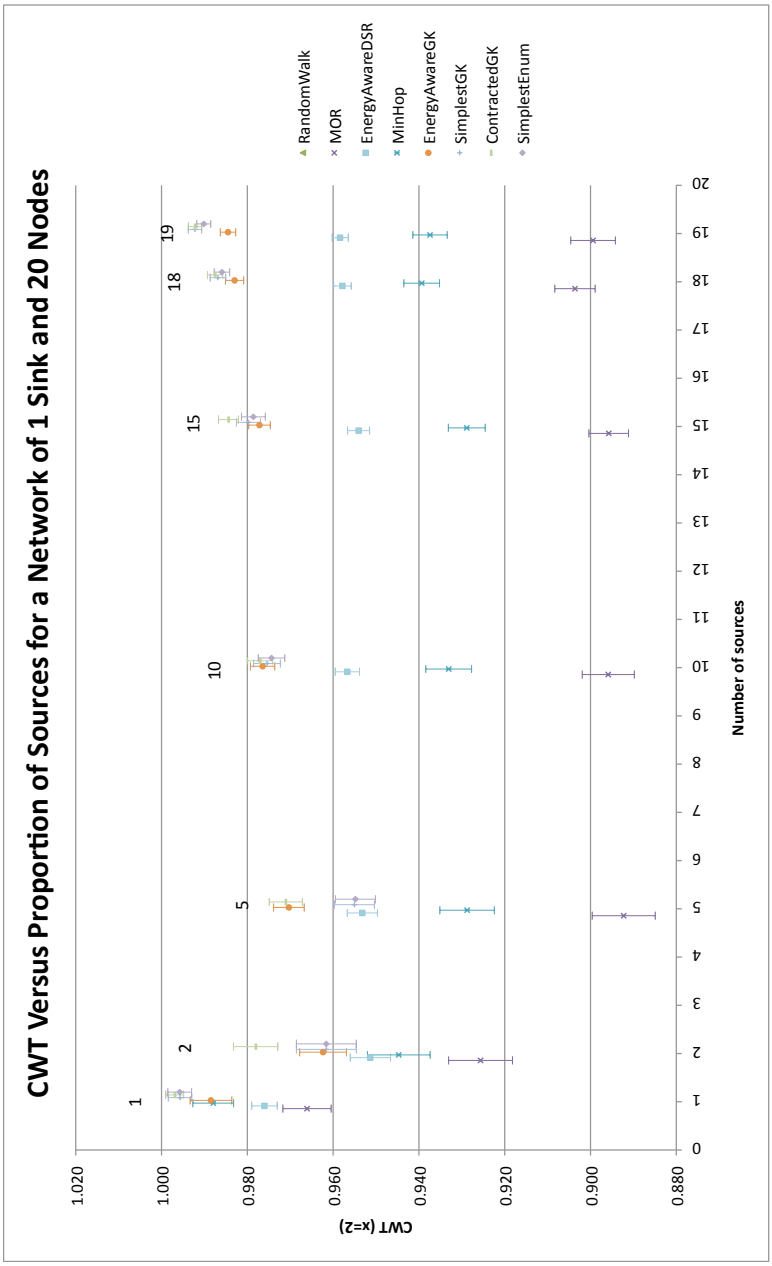


Figure 9.2: CWT vs. proportion of sources for networks with 20 nodes and 1 sink

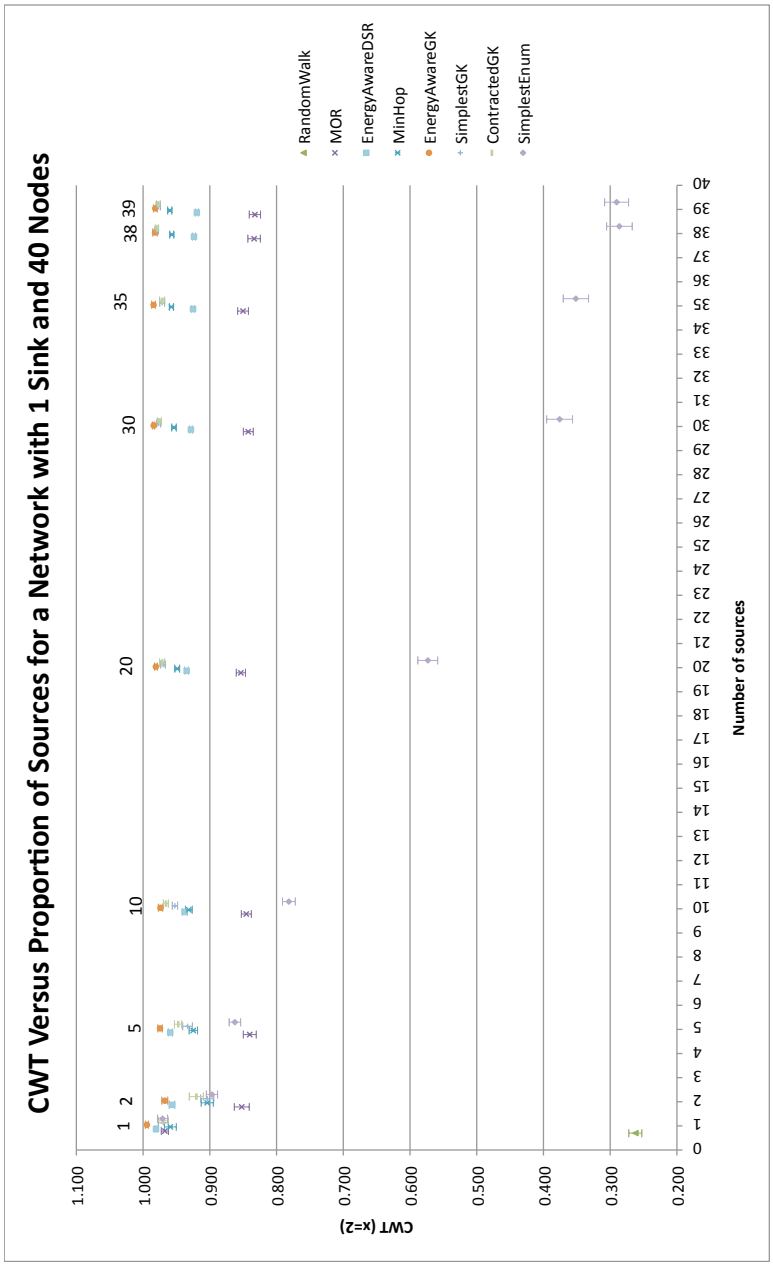


Figure 9.3: CWT vs. proportion of sources for networks with 40 nodes and 1 sink

Two effects of increasing the proportion of sources in a network have been observed in this experiment. The first is the performance convergence between ContractedGK and SimplestGK as the proportion of sources increases. For certain proportions of sources, the points even overlap. The second is the decline in performance of routing protocols that enumerate paths, particularly in large networks. Each of these three effects is described in more detail below:

As the proportion of sources increases, ContractedGK converges on SimplestGK. The reason for this convergence is that as the proportion of sources increases, fewer nodes can be contracted, since only non-source, non-sink nodes with a single incoming and outgoing edge can be contracted. Thus, the networks become similar and so the node reliance values match each other more closely in each model. Finally, when the number of sources is equal to (n-1), i.e. every node is either a source or a sink, no nodes can be contracted and so the node reliance values are calculated in the same way as for the simplest paths model.

The performance of SimplestEnum is shown to decline as the proportion of sources increases in networks of 40 nodes (Figure 9.3). The decline in performance is not present in the graphs representing smaller networks. Routing protocols that discover network topology based on enumeration send one Exploration message along each simplest path. Consequently, increasing the number of source nodes may be expected to produce a proportional increase in the number of Exploration messages being sent, causing more energy to be expended on nodes and sources and sinks to be connected for less time. Examining the graph of Figure 9.3 confirms this theory. The segment of the line between 2 and 30 sources is almost straight, suggesting that the performance of the routing protocol is proportional to the proportion of sources. For example, increasing the proportion of sources by 5 causes the normalised CWT score to drop by approximately 0.1. The relationship does not hold with one source or with more than 30 sources due to the selection of nodes with very few simplest paths to a sink. Where there is only one source, the average number of Exploration messages per source is extremely small. Where there are more than 30 sources, it is more probable that a node with very few simplest paths to a sink will be selected as a source.

An increase in the use of nodes with small numbers of simplest paths causes the energy expended for network discovery to drop, thus causing overall (source, sink) connectivity to increase. Hence, with one source or more than 30 sources, the points on the graph are higher than would be expected from the linear portion of the graph.

9.5.3 Total Data Transfer and Increasing Proportions of Sources

Figures 9.4, 9.5 and 9.6 show the change in total data transfer when the proportion of sources is increased in networks of 10, 20 and 40 nodes respectively. The graphs show the average number of bytes received by sink nodes across all topologies (random, triangular, square and hexagonal).

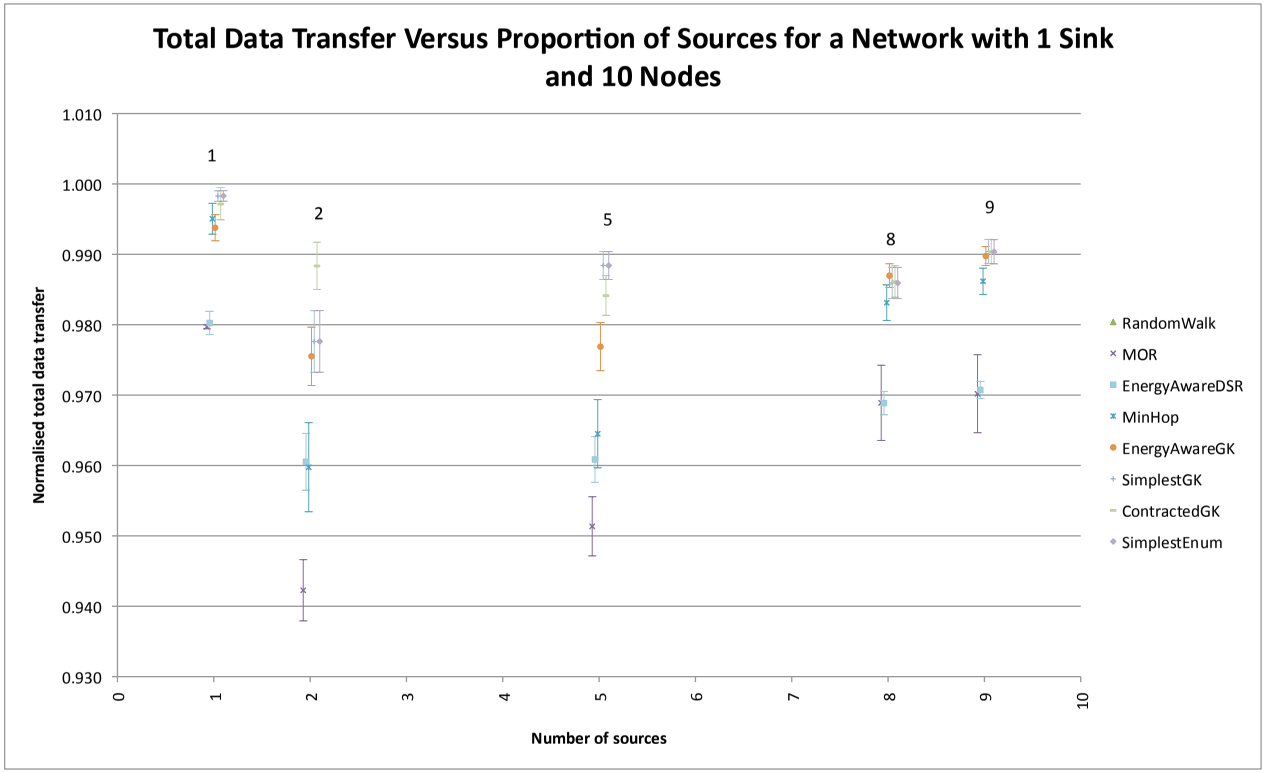


Figure 9.4: Total data transfer vs. proportion of sources for networks with 10 nodes and 1 sink

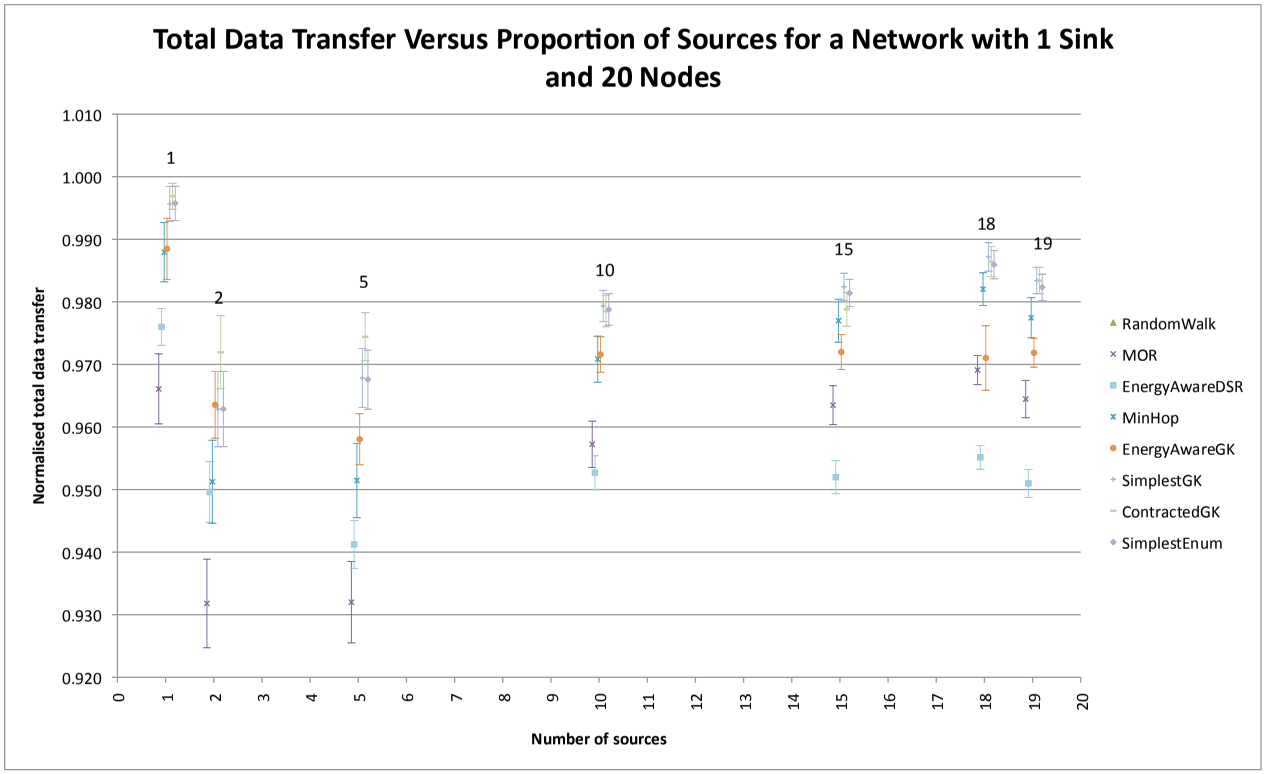


Figure 9.5: Total data transfer vs. proportion of sources for networks with 20 nodes and 1 sink

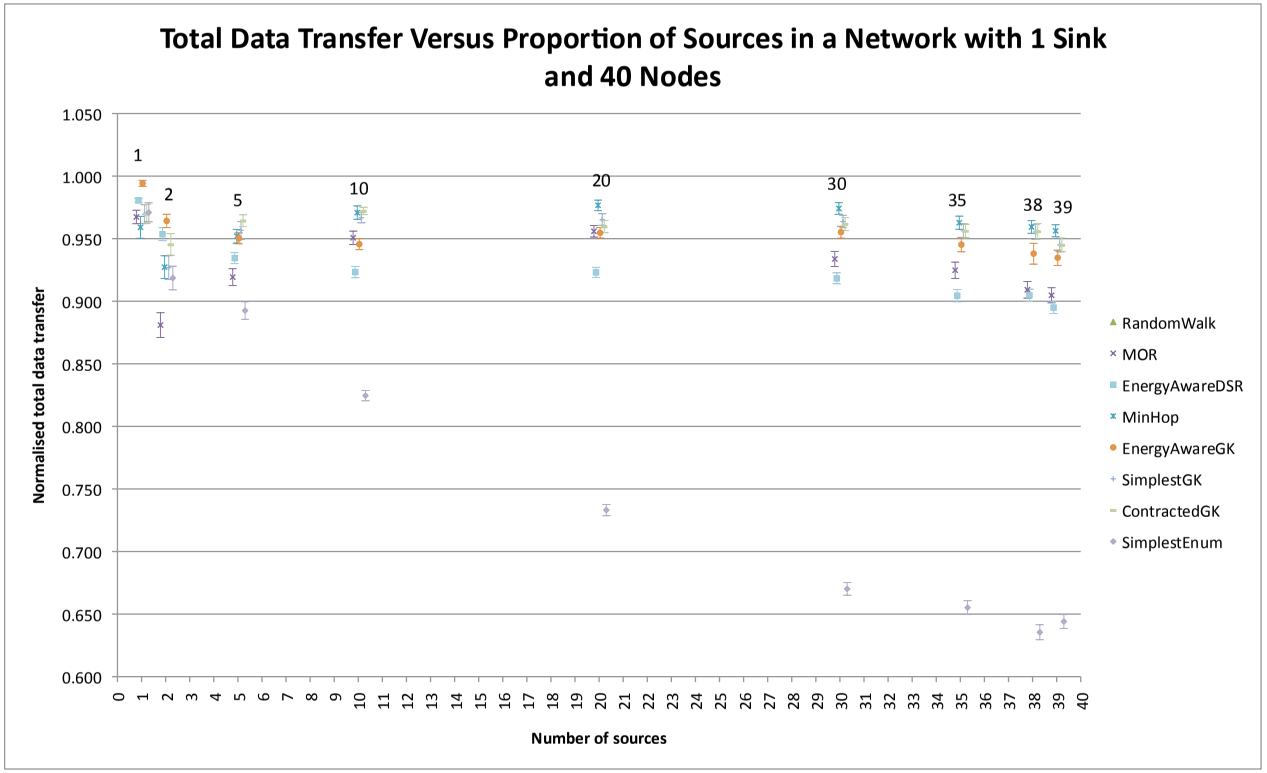


Figure 9.6: Total data transfer vs. proportion of sources for networks with 40 nodes and 1 sink

These graphs can be used to verify many of the statements made in the previous section, and make it possible to distinguish why certain routing protocols perform better or worse than others. In particular, if a lot of energy is expended in determining the network topology, the total data transfer of the network will substantially drop. Conversely, if (source, sink) connectivity drops for some other reason, such as distant nodes being disconnected from the network, CWT will drop, but the total data transfer will not.

The figures show that there is no substantial drop in total data transfer, except for SimplestEnum in networks with 40 nodes, which is shown by Figure 9.6. At 40 nodes, the number of possible paths has dramatically increased and so nodes expend more energy enumerating all possible paths than routing data. It is therefore reasonable to conclude that the only instance of routing protocols performing poorly due to enumerating all simplest paths is in the case of 40 node networks. In other cases, the poor performance of a routing protocol is due to another factor.

As previously discussed, the contracted simplest paths model more closely resembles the simplest paths model as the number of sources increases. Thus, as with CWT, the total data transfer of ContractedGK and SimplestGK converge.

9.5.4 Effect of Increasing Numbers of Sources

This section considers the effect of increasing the number of sources in the network, rather than just the proportion of nodes that are sources. Figures 9.7 and 9.8 show the effect on total data transfer and CWT respectively as a result of increasing source numbers. Since every node in these networks is either a source or a sink, it is not possible for network contractions to take place. Consequently, the performance of ContractedGK is identical to SimplestGK.

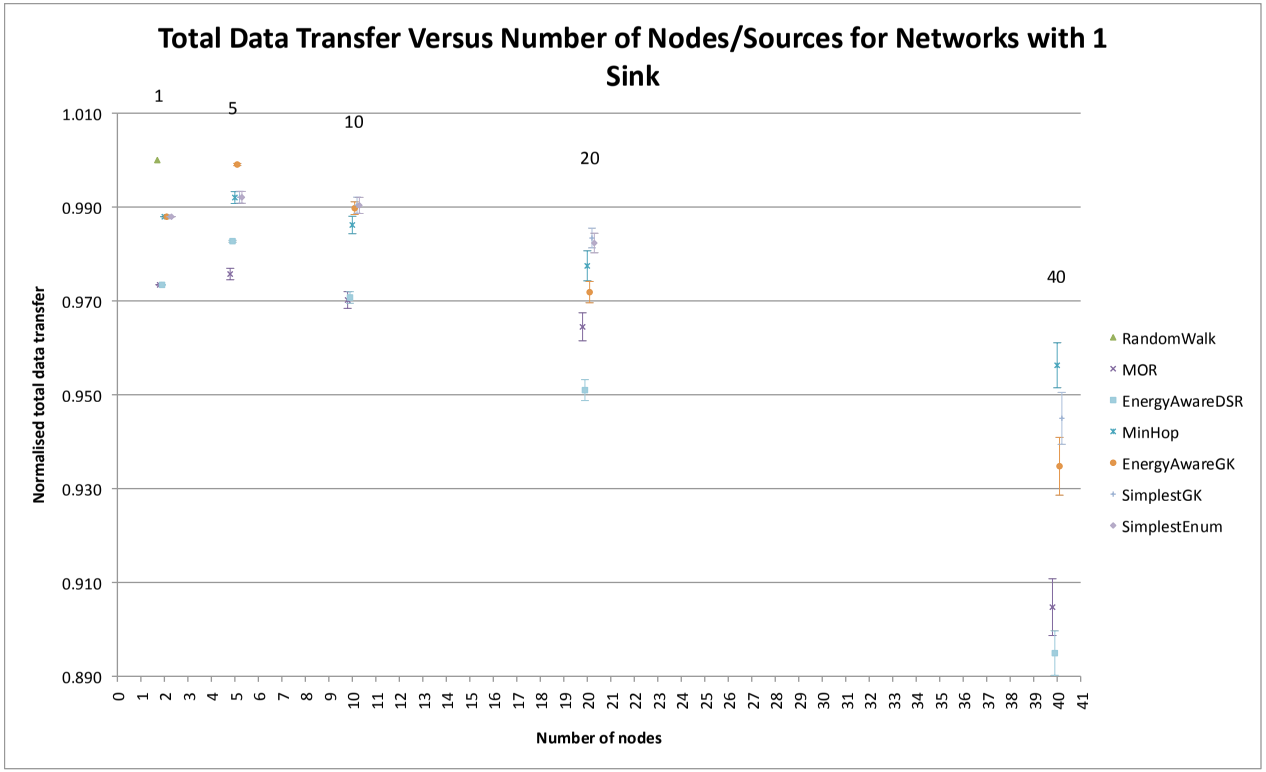


Figure 9.7: Total data transfer vs. number of nodes/sources for networks with 1 sink

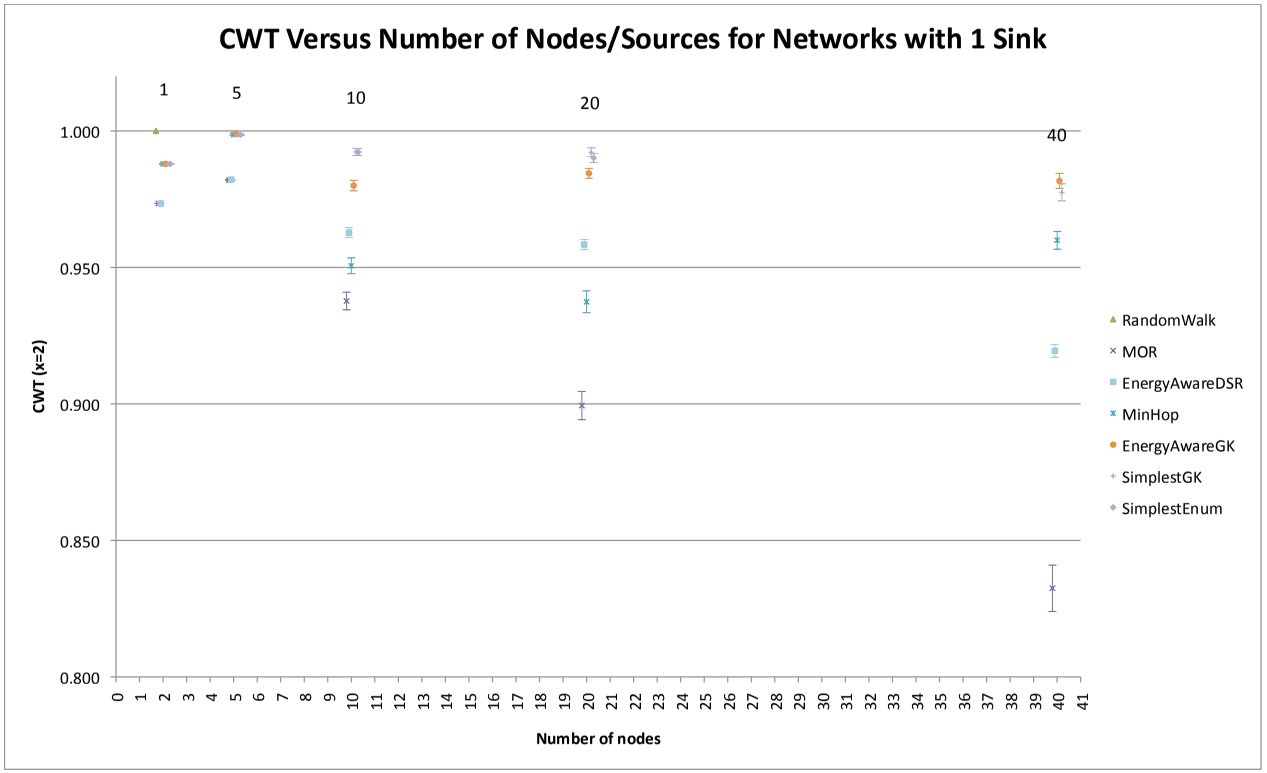


Figure 9.8: CWT vs. number of nodes/sources for networks with 1 sink

The total data transfer of all routing protocols drops as the number of sources increases. In general, adding a node to the network does not increase the capacity of the network. The capacity is limited by the *minimum cut* between the sources and sinks, i.e. the smallest number of nodes whose removal separates all sources from all sinks. Unless the addition of a node to the network causes that cut to increase, the maximum data that can flow between the sources and sinks does not change. Furthermore, the introduction of an additional source to the network increases the energy expended by the network in finding a path from that source to the sink. Consequently, less energy is available for routing data between sources and sinks and so the total data that can be transferred is reduced.

Figure 9.7 shows that SimplestEnum responds poorly to increased network size. As the number of nodes/sources increases beyond 20, total data transfer for this routing protocol dramatically decreases. This decrease in total data transfer occurs as a result of more energy being expended to determine the network topology by enumerating all simplest paths. As discussed in Chapter 6, the number of simplest paths is $3^{\frac{n-2}{3}}$ where n represents the number of nodes in the network. Hence, the number of simplest paths grows very quickly with respect to the number of nodes in the network in a worst-case scenario. Since SimplestEnum sends one message along each simplest path, an increase in the number of nodes causes the energy expended to rapidly increase. Furthermore, in SimplestEnum, each Exploration message stores the sequence of nodes through which it has travelled. A larger network therefore results in larger Exploration messages, which require more energy to transmit, again increasing energy expenditure of the network. RandomWalk continues to perform the worst, and its performance drops so quickly that only a single data point can be shown on the graph in Figure 9.7. An increasing network size dramatically increases the number of routes available. Additionally, on average, the number of hops that a message must travel through to reach the sink increases and so the energy expended by the network per message also increases.

The CWT for each routing protocol is shown in Figure 9.8. The increased energy expenditure of RandomWalk as well as SimplestEnum cause them to perform similarly poorly with respect to CWT. MOR and EnergyAwareDSR also perform poorly with increasing network size. Possible reasons for the poor performance of these routing protocols will now be discussed.

The declining performance of EnergyAwareDSR is not due to its routing decisions. The same routing decisions are made by routing protocol EnergyAwareGK whose CWT score

does not drop with increasing network size. Thus, the declining performance must be due to the way in which EnergyAwareDSR gathers the topology. It may also be argued that the decreased total data transfer is not the sole cause of the reduced CWT of the network. As shown by Figure 9.7, all routing protocols experience a drop in total data transfer as network size increases. However, some routing protocols such as MinHopGK and EnergyAwareGK do not experience a drop in CWT. It may therefore be argued that the declining performance of EnergyAwareDSR is due to the loss of high reliance nodes caused by an increase in the messages exchanged in order to determine the network topology. Consider that there may be a small number of nodes neighbouring a sink. All data travelling to that sink must travel through one of those nodes. Consequently, the loss of those nodes will cause the CWT of the network to fall. However, being further away from the sources, those nodes are also more likely to lie on more paths between the source and sink. Thus, when a route request takes place, those nodes will receive and transmit more messages than other nodes. As the number of sources increases, the capacity of the network is unlikely to change. However, the number of route requests will increase, causing those highly relied upon nodes to fail more quickly. A similar situation may explain the declining performance of MOR.

Routing protocols that use a global knowledge collection mechanism are not subject to the same problem, since the energy expended by the network in determining the network topology is distributed through the network. Since all topology data is distributed to the network, no subset of nodes expends more energy than other any other node.

9.6 Summary

This experiment is designed to show the efficacy of different routing protocols with respect to total data transfer and (source, sink) connectivity as measured by CWT when a perfect model of communication is used. Unlike the experiment discussed in Chapter 7, which only considers routing heuristics, the experiment in this chapter also takes into account the costs associated with gathering the network topology and maintaining routes between sources and sinks.

The results show that although the EnergyAwareGK heuristic performed best, as a routing protocol it is only marginally better at maintaining (source, sink) connectivity than other

routing protocols that are tested. Furthermore, other protocols such as ContractedGK that use shortest path ordering on node reliance values and gather the network topology via global knowledge are better than EnergyAwareGK at getting a high total data transfer.

Other results that are observed in Chapter 7 remain valid in the experiment of this chapter. For example, lexicographic ordering performs worse in terms of both CWT and total data transfer than shortest path routing.

Various data collection mechanisms are examined, including global knowledge, enumeration and a DSR-like mechanism. Enumeration is shown to perform more poorly than global knowledge in networks of more than 20 nodes due to the rapid growth of the number of simplest paths, each of which has a message sent along it in an enumeration-based routing protocol. Although all routing protocols suffer a drop in total data transfer as a result of increasing network size, some manage to maintain good CWT. MOR and routing protocols that collect data in a DSR-like manner are shown to scale poorly with an increasing network size. It is hypothesised that this is due to the increased expenditure of energy on key nodes that keep the network connected.

Finally, RandomWalk consistently performs poorly in almost all simulations.

Having examined the performance of routing protocols with a perfect communications layer, Chapter 10 explores the same set of routing protocols in a realistic communications layer.

Chapter 10

Routing Protocol Experiment (Realistic Physical Layer)

This experiment analyses the effectiveness of the node reliance routing protocols in comparison with other published routing protocols in a network with a realistic model of communication in which messages may be lost due to interference or collisions (two nodes transmitting simultaneously). The experiment reflects how a routing protocol may perform in a real network of nodes.

The following routing protocols are compared as part of this experiment:

- Lexicographically ordered simplest paths gathered by global knowledge
- Shortest path ordered simplest paths gathered by global knowledge
- Lexicographically ordered simplest paths gathered by enumeration
- Shortest path ordered simplest paths gathered by enumeration
- Lexicographically ordered contracted simplest paths gathered by global knowledge
- Shortest path ordered contracted simplest paths gathered by global knowledge
- Minimum hop path gathered by global knowledge
- The EnergyAware heuristic combined with global knowledge

- The EnergyAware heuristic combined with a modification of DSR [54]
- Random walk routing [113]
- MOR [10]

The configuration of the experiment is exactly the same as that for the experiment carried out with a perfect communications model as discussed in Chapter 9.

10.1 Results

Of the 11880 different networks that each routing protocol operated on, there are 2383 networks in which the connectivity is too poor for any routing protocol to route any data from a source to a sink. In these networks, the normalised scores are undefined because the optimal measurement and the measurement for a particular routing protocol are both zero. For example, if the optimal CWT score for a given network is 0 then the normalised score for any routing protocol must be 0/0, which is undefined. Consequently, these networks are disregarded from any calculations.

Table 10.1 shows the results of the experiment that operated under a realistic physical layer.

Name	Heuristic	Discovery	CWT		Transfer	
			Value	SD	Value	SD
MinHopGK	Min Hop	GK	0.551	0.80%	0.601	0.72%
ContractedGK	ContractedShortest	GK	0.522	0.84%	0.582	0.74%
SimplestGK	SimplestShortest	GK	0.521	0.84%	0.583	0.74%
EnergyAwareGK	EnergyAware	GK	0.487	0.87%	0.548	0.62%
MOR	Min Hop	DV	0.486	0.87%	0.559	0.69%
SimplestEnum	SimplestShortest	Enum	0.337	1.22%	0.457	0.97%
RandomWalk	RandomWalk	Local	0.298	1.08%	0.368	0.83%
EnergyAwareDSR	EnergyAware	DSR-like	0.215	1.59%	0.336	1.19%

Table 10.1: Normalised CWT and transfer metrics for each routing protocol with a realistic radio model

The results in Table 10.1 vary greatly. For example, with respect to CWT, the best performing routing protocol (MinHopGK) performed 156% better than the worst (EnergyAwareDSR). The spread of results suggests that the choice of routing protocol makes a big difference in a realistic physical layer and that choosing the wrong one could result in very poor application performance. The spread of results with respect to total data transfer is lower at 79%. However, the difference is still significantly large that the correct choice of routing protocol is vital.

The standard deviation of the results is at most 1.59% of the CWT and 1.19% of the total data transfer, indicating that the results are reliable but that there is a bigger uncertainty than when a perfect physical layer is present. This decrease in reliability is not unexpected since successful communication between two nodes is random in this experiment whereas it is guaranteed with perfect physical layer. In general, each of the routing protocols operating with a realistic physical layer had a significantly larger error than the same routing protocol operating in a perfect physical layer. It is therefore more difficult to make generalisations or conclusions regarding the routing protocols in a realistic physical layer.

The results in this chapter and that of Chapter 9, which considered a perfect physical layer, are achieved by examining the same set of networks. It is therefore possible to determine the average consequences of applying the realistic physical layer on both the CWT and total data transfer metrics, as summarised in Table 10.2. As before, networks in which routing is impossible with a realistic physical layer have not been included in these calculations.

The introduction of a realistic physical layer has the biggest effect on the routing protocols that discover the topology by DSR or Enumeration. Each of these collection methods relies on sending exploration messages from sources to sinks. Each exploration message contains the sequence of nodes through which it has travelled. Discovering the optimal path in this manner has poor scalability. As the size of the network increases, the number of exploration messages increases and the number of nodes each message may have passed through increases. Both of these factors decrease the probability of message receipt, since more messages increases the probability of collision and bigger messages are less likely to be successfully transmitted.

DSR performs more poorly than enumeration for two reasons. Firstly, in enumeration, the entire set of (source, sink) paths is collected before an exploration reply is flooded. Therefore, regardless of the number of sources, only one exploration reply is flooded per

Name	Heuristic	Discovery	CWT		Transfer	
			Value	SD	Value	SD
EnergyAwareDSR	EnergyAware	DSR-like	-0.744	0.53%	-0.623	0.72%
SimplestEnum	SimplestShortest	Enum	-0.590	1.10%	-0.496	1.18%
EnergyAwareGK	EnergyAware	GK	-0.502	0.90%	-0.435	1.06%
MOR	Min Hop	DV	-0.461	1.06%	-0.406	1.08%
SimplestGK	SimplestShortest	GK	-0.463	1.01%	-0.403	1.16%
ContractedGK	ContractedShortest	GK	-0.463	1.02%	-0.404	1.16%
MinHopGK	Min Hop	GK	-0.422	1.13%	-0.384	1.34%
RandomWalk	RandomWalk	Local	-0.206	3.07%	-0.229	2.51%

Table 10.2: Difference between the normalised CWT and total data transfer metrics in perfect and realistic physical models

sink. Conversely, in DSR, each sink floods one route reply whenever a better route request is received from a source. The number of floods is therefore much higher which leads to message collisions. Secondly, the ExplorationReply message in enumeration can be split up and sent in parts. Each part is small and is therefore more likely to be received. However, in DSR no message splitting occurs and so the probability of the route reply being received by the sources is smaller.

All routing protocols perform less well with a realistic physical layer than with a perfect physical layer. However, the performance of RandomWalk does not decline as much as other protocols. In the experiment with a perfect physical layer, the poor performance is due to messages expending lots of energy in the network through the random and often indirect path that they took. However, with a realistic physical layer, it is less likely that a message will be successfully sent through multiple nodes. Consequently, although the message may not arrive at a sink, it will not expend large amounts of energy in the network, thus allowing nodes to operate for longer.

It is expected that MOR would perform poorly in a realistic physical layer, since it relies upon bidirectional links in order to confirm reception of a packet. However, MOR does not perform significantly worse than other routing protocols. MOR is a multipath routing protocol and so if a node A considers its neighbouring node B to have failed (i.e. if an acknowledgement is not received from B) then node A may select another node to route future

messages. Therefore, the impact on the network as a result of assuming bidirectionality is limited.

10.1.1 CWT and Increasing Proportions of Sources

Increasing the proportion of sources in the network had very little effect with respect to the CWT measurement. Figure 10.1 shows the change in CWT as the number of sources in networks of 40 nodes is increased. The graph considers the average CWT across all topologies (random, triangular, square and hexagonal).

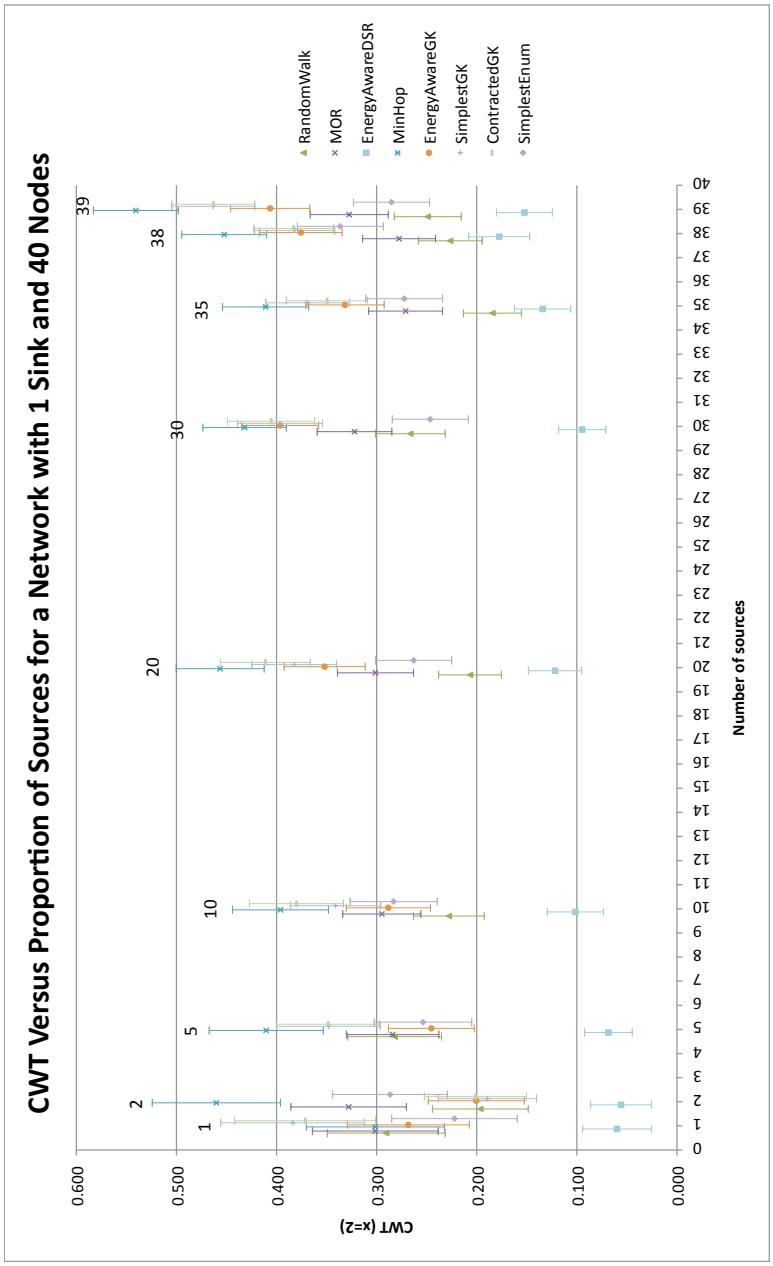


Figure 10.1: CWT vs. proportion of sources for networks with 40 nodes and 1 sink

Only the 40 node network is considered for two reasons:

Firstly, as already stated, there is little effect from increasing the proportion of sources. In routing protocols that used global knowledge as a discovery mechanism, each node exchanges its network topology regardless of whether it is a source or not. In other routing protocols that flood exploration messages, an increase in the proportion of sources increases the probability with which two messages will collide. Thus, the large number of messages that might be exchanged in a perfect network are not exchanged in a realistic network.

Secondly, the graph representing 40 node networks has been shown here as it provides the greatest variance in the proportion of sources in the network. Although the performances of different routing protocols vary between 10, 20 and 40 node networks, these differences are covered in a later section discussing the effect of increasing network size.

Figure 10.1 also confirms many of the results shown in Table 10.1. EnergyAwareDSR and RandomWalk perform the worst. These are followed by SimplestEnum and then by MOR. The best performing routing protocol is MinHopGK.

10.1.2 Total Data Transfer and Increasing Proportions of Sources

Figure 10.2 shows the effect on total data transfer of increasing the proportion of sources in 40 node networks of triangular, square, hexagonal and random node placements.

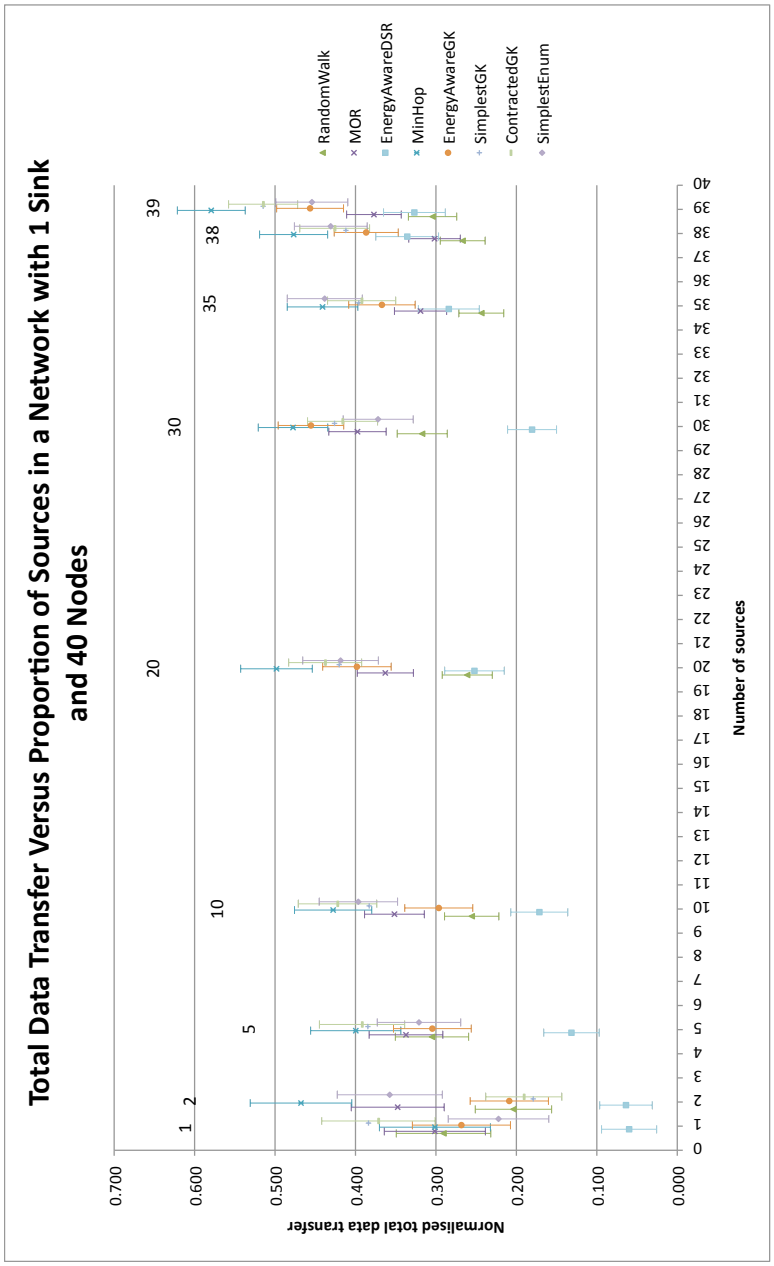


Figure 10.2: Total data transfer vs. proportion of sources for networks with 40 nodes and 1 sink

As with the CWT measurement, in the majority of networks, an increase in the proportion of sources has no significant effect. In a 40 node network, however, there is a large improvement in the total data transfer performance of EnergyAwareDSR as the proportion of source nodes increases.

Since Figure 10.1 does not show a corresponding improvement in CWT with increasing source proportion, it is reasonable to conclude that the improved total data transfer that occurs in all routing protocols as a result of increasing proportion of sources is not as a result of more sources connecting to sink nodes. However, as the proportion of sources increases, those sources that are closer to a sink node are more likely to discover a path to the sink. Since any message sent as part of the discovery mechanism has a small number of hops to travel through to reach a sink, it is not given the opportunity to grow to a size where transmission becomes improbable. Similarly, if a response to an exploration message is made, it will be smaller, is given less opportunity to grow and only needs to travel through a small number of hops to reach the source again. Therefore, it is more likely that (source, sink) paths can be established. In a network of few sources, it is less likely that a nearby node will be selected to be a source. Thus, the total data transfer is likely to be low.

With respect to Figure 10.2, the graph continues to follow the trend shown with CWT in the previous section. However, the difference in performance between each routing protocol is smaller with many routing protocols clustered together in the middle and the difference in performance between the best and worst routing protocols being smaller.

10.1.3 Effect of Increasing Numbers of Sources

Figures 10.3 and 10.4 show the effect on total data transfer and CWT of increasing the number of sources in the network rather than the proportion of nodes that are sources.

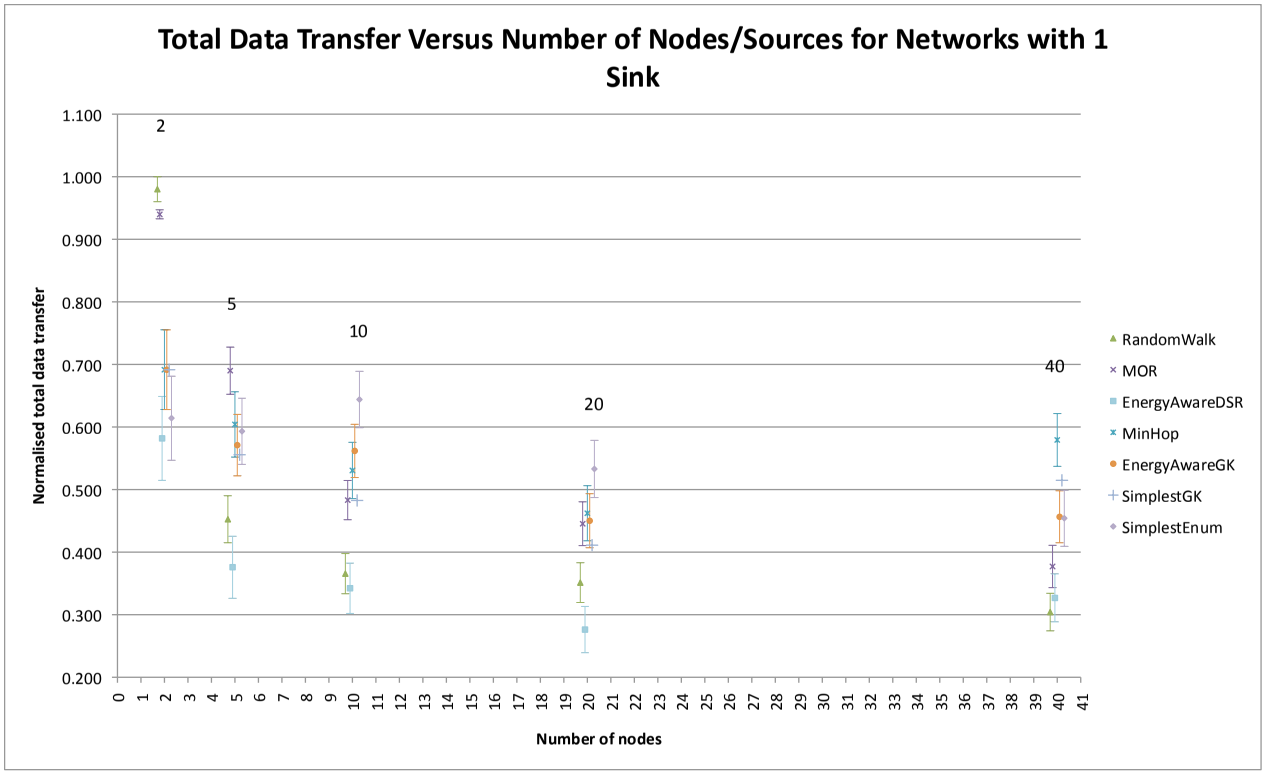


Figure 10.3: Total data transfer vs. number of nodes/sources for networks with 1 sink

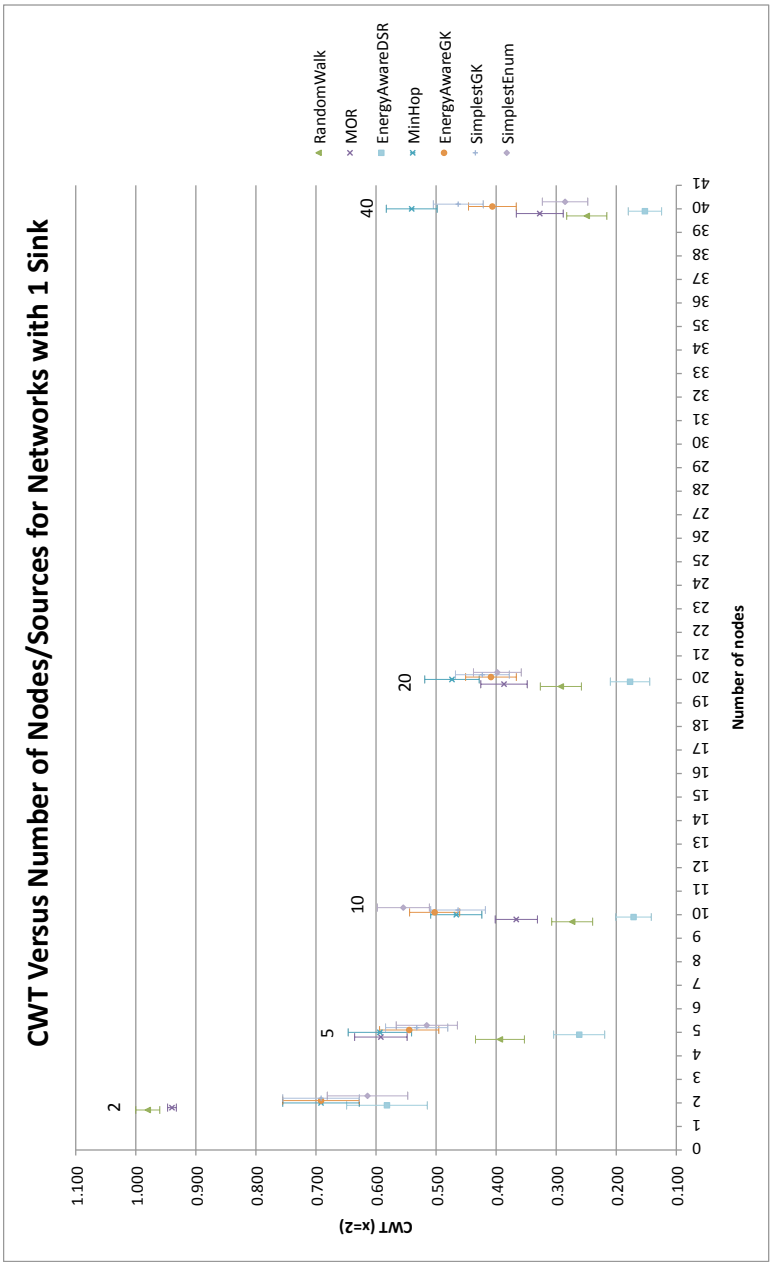


Figure 10.4: CWT vs. number of nodes/sources for networks with 1 sink

Since all nodes in these networks are either sources or sinks, a contraction of the network is not possible. Therefore, ContractedGK behaves identically to SimplestGK.

Figure 10.3 shows that RandomWalk is the best in networks containing two nodes. Since a network containing two nodes has only a single link (and therefore a single path), the most efficient routing protocol would be the one with the lowest overhead, i.e. RandomWalk. Where other routing protocols must engage in a sequence of message exchanges in order to establish a (source, sink) path, RandomWalk may simply forward a message along the only link in the network in order to route.

For networks of five nodes, the number of links (and number of possible paths) quickly grows, and the efficacy of RandomWalk drops. In such networks, the routing protocol producing the highest total data transfer is MOR. In a network of five nodes, of which one must be the sink and another must be a source, there are only three other nodes that may engage in routing. Consequently the number of paths is small and so the overhead experienced with MOR establishing multiple paths is limited. Since MOR also sends small packets, it is more capable of finding a (source, sink) path than other routing protocols whose large packets may be lost in transmission. MOR assumes bidirectional edges are present and so its RREQ and RREP messages do not contain full paths but simply the number of nodes through which each message has travelled.

As the network size increases to 10 and 20 nodes, SimplestEnum performs the best in terms of total data transfer. MOR's performance may drop due to the increase in network size, which increases the overhead associated with forming multiple paths. Furthermore, it becomes more probable that a given (source, sink) path will contain an edge that is not bidirectional and therefore cannot be used. Conversely, SimplestEnum does not require edges to be bidirectional. As shown in the experiment that is carried out with a perfect communication model, the network size does not cause a rapid increase in the number of Exploration messages sent until it exceeds 20 nodes. It is easy for sources that are close to the sink to form paths, since the Exploration messages are small and are forwarded through few nodes. The presence of unreliable links and only a moderate network size means that a large amount of energy is not expended by sending Exploration messages through the network. This hypothesis can be verified to some extent in that a heuristic using enumeration to discover the topology performs more poorly when it instead discovers the topology via global knowledge. Consequently, the performance of these routing protocols is due to their data gathering methodology rather than their routing heuristic.

However, as the network size increases to 40 nodes, Exploration messages sent by SimplestEnum may cause a substantial increase in overhead in determining (source, sink) paths and those routing protocols rapidly decline in performance. The best routing protocol at this network size is MinHopGK. A possible explanation for this result is that, at this network size, many messages travel through many hops, increasing the number of hops that a message must travel through increases the probability with which a message may collide or else be sent along an unreliable link and causing its loss. Minimum hop routing discourages the use of paths with many nodes, and thus, these dangers are negated. It had been expected that minimum hop routing might encourage the use of geographically longer, less reliable links in order to minimise the number of hops. However, the results indicate that if this is the case, it is overshadowed by sending messages through few nodes.

The worst performing routing protocol is EnergyAwareDSR. Since EnergyAwareGK performs acceptably well and uses the same heuristic, it is reasonable to conclude that the poor performance of EnergyAwareDSR is due to its use of the DSR-like discovery mechanism rather than its routing heuristic.

The graph in Figure 10.4 represents CWT with increasing number of sources and the graph closely matches that representing total data transfer. The only significant difference is that (with respect to CWT) MinHopGK performs better than SimplestEnum in networks of 20 nodes. Since SimplestEnum achieves a higher total data transfer in networks of this size, it is reasonable to conclude that MinHopGK causes more sources to be simultaneously connected than SimplestEnum. Again, the most likely explanation is that the use of numerous hops (in SimplestEnum) causes message loss. By minimising the number of hops through which a message must travel, MinHopGK successfully keeps multiple sources connected to sinks simultaneously.

The higher total data transfer and lower CWT scores in 20 node networks for SimplestEnum (compared to MinHopGK) also suggest that the source-forwarding problem may occur. The drain of batteries is higher when a source forwards data from other sources than when that source generates its own data and routes it to a sink. Therefore, if SimplestEnum transfers more data but keeps fewer sources connected than the MinHopGK, the high total data transfer may be caused by sources nearer the sink failing to forward data from other sources. As already suggested, one explanation for this is that sources further away require longer Exploration messages to be routed through the network, which increases their chances of being lost. Consequently, the use of SimplestEnum may prevent far away sources from

routing.

10.2 Summary

The use of a realistic physical layer has a very significant effect on the efficacy of a routing protocol, both in terms of total data transfer and CWT. The results suggest that for larger networks, MinHopGK provides a higher total data transfer and connectivity than other routing protocols. In is hypothesised that this is due to the routing protocol's tendency to send messages through multiple hops, increasing the probability that messages may be lost. In smaller networks (those with 20 or fewer nodes), SimplestEnum may be better at achieving data throughput, but is only marginally better at maintaining connectivity. It has been hypothesised that nodes nearer the sink are able to engage in routing due to the small number of small Exploration messages sent in order to determine (source, sink) paths. However, sources that are more distant from the sink are more likely to lose Exploration messages and so would be unable to take part in routing, resulting in lower source diversity.

Overall, in a realistic physical layer, MinHopGK performs the best both in terms of total data transfer and connectivity (CWT). These results are closely followed by ContractedGK and SimplestGK, again emphasising the preference for shorter paths over longer paths. The worst routing protocol is EnergyAwareDSR, indicating that a capability of dealing with unidirectional links is not sufficient. It is necessary to cope with unreliable links by minimising routing protocol message sizes.

Chapter 11

Conclusions and Further Work

A number of WSN applications benefit from having data from a diversity of sources for as long as possible. However, it is not immediately obvious which routing protocols from the WSN literature are best disposed towards improving this diversity. Common approaches to routing in WSNs focus on minimising energy expenditure, balancing energy expenditure or routing through as few nodes as possible in order to maximise the time for which the network can function. None of the routing protocols examined claims to improve (source, sink) diversity over time.

This thesis proposes a new approach to routing, known as node reliance, and examines a number of routing heuristics and protocols based on node reliance to keep as many sources connected to sinks for as long as possible in a WSN. Nodes are assigned values indicating their importance to the network in routing data from sources to sinks, based on the proportion of (source, sink) paths on which they appear. A node that appears on many paths is considered important because it is heavily relied upon in routing messages from sources to sinks. Paths from sources to sinks are formed such that as few high value nodes are used as possible. It is hypothesised that by avoiding those nodes that are highly relied upon, sources and sinks can remain connected for longer, even though the total data transferred may remain the same.

To determine the efficacy of node reliance routing and other third party routing protocols at maintaining high source diversity for long periods of time, several simulation experiments are proposed in which sources regularly generate data and route it towards a sink. The new CWT metric and total data transferred are measured to determine which routing protocols

best achieve high source diversity. Details of the experiments and the results are covered in Chapters 7, 9 and 10. This chapter draws some conclusions based on those results and proposes further work where necessary.

11.1 Intuition of Node Reliance

Load balancing routing protocols (discussed in Section 4.5) extend the time until the first node expires and so may also be used to improve (source, sink) connectivity over time. They operate by requiring knowledge of either the data generation rates of sources, which may be unknown if sources produce data in response to external phenomena, or the remaining energy of nodes in the network, which causes an increase in overhead and therefore faster expiration of nodes.

Node reliance requires no such knowledge. Instead, a node should be avoided where possible if other nodes rely on it and may be used if it is not heavily relied upon. The degree to which the node is relied upon depends on the proportion of paths it lies on between sources and sinks. For example, if a node lies on all paths between a source and a sink then that node is essential in allowing that source to route to a sink and so should be avoided by other sources.

The node reliance routing protocol is compared with third party routing protocols to determine the circumstances in which each protocol keeps sources connected to sinks for the longest.

11.2 Summary of Results

Chapter 7 describes an experiment to analyse routing heuristics. The results consistently indicate that if no energy is expended in obtaining network topology data, load balancing routing is the best way to maintain (source, sink) connectivity. Load balancing routing heuristics distribute routing workload so that the time until the first node expires is extended as much as possible. The second best routing heuristic (again, assuming that the topology data costs no energy to get) is node reliance, calculated using a simple paths heuristic and ordered by shortest path routing. However, the set of simple paths grows very quickly and

enumerating all paths to calculate node reliance values takes too great a time to be practical.

In practice, it is impossible to gather the required network knowledge without expending energy. Nodes may randomly become unusable due to changes in environmental conditions and energy sources may vary in their capacities. A WSN may be deployed in such a manner that it is less prone to failure and has significantly more energy available than the sensing task requires. However, such a deployment would achieve a perfect (source, sink) connectivity since all sources would remain continually connected and so the choice of routing protocol becomes irrelevant in the majority of cases. It is important to consider the associated costs in gathering the network topology. The experiments discussed in Chapters 9 and 10 analyse the effectiveness of routing protocols and take such costs into account.

As discussed in Chapter 2, radio communication in a WSN is often unreliable. However, there exist various techniques that might be used to make transmissions more reliable or to make links between nodes bidirectional. It is therefore necessary to examine each routing protocol in both environments, i.e. under a realistic and a perfect communications layer.

The results summaries that follow are concerned with networks in which one node is a sink and all others are sources. Such a setup shown to be common in the deployments studied in Chapter 2.

In the case of a perfect network, total data transfer is maximised using load balancing in small networks of 10 or fewer nodes. In networks of 10 or 20 nodes, the highest total data transfer is achieved by using a simplest paths or contracted simplest paths node reliance heuristic and selecting the path of least total node cost. Finally, in the case of 40 node networks, minimum hop routing achieves the highest total data transfer. In each of these routing protocols, the network topology collection is achieved by global knowledge. A near optimal connectivity score (as measured by the CWT metric described in Section 5.2) is achieved in all examined networks by using simplest paths or contracted simplest paths node reliance. However, at 40 node networks, load balancing very slightly outperforms node reliance. All of these routing protocols also collect topology data via global knowledge. However, for networks of 20 or fewer nodes, the optimal CWT score may additionally be achieved by using a node reliance enumeration-based routing protocol. Past 20 nodes, the performance of enumeration-based routing protocols rapidly declines.

In a realistic network, total data transfer is optimal for networks of five nodes when MOR is

used. For networks of 10 or 20 nodes, enumeration-based node reliance routing protocols provide the highest total data transfer. For networks of 40 nodes, the highest total data transfer is achieved by using minimum hop routing and gathering the network topology via global knowledge. Similarly, for optimising connectivity, MOR provides an optimal data transfer in networks of five nodes and the enumeration-based node reliance routing protocols are best in networks of 10 nodes. Minimum hop routing (gathering topology by global knowledge) produces the best connectivity for networks of 20 and 40 nodes and is jointly optimal (with MOR) in networks of five nodes.

All routing protocol experiments show that minimum hop routing is the best at maintaining (source, sink) connectivity when considered over all possible parameters of sources, sinks, nodes and placements. Minimum hop routing is also the best at achieving total data transfer in realistic networks. However, in a perfect network, the highest total data transfer is achieved by simplest path or contracted simplest path node reliance routing protocols that select the paths of least total weight and gather the topology via global knowledge.

11.3 Conclusions From Analysis of Results

11.3.1 Minimum Hop Routing

As discussed in Chapter 4, many authors discourage the use of minimum hop routing due to potentially unreliable links and the overuse of certain key nodes, which may cause the network to partition.

The results of the experiments carried out as part of this thesis do not corroborate such a viewpoint. The results suggest that minimum hop routing regularly achieves a high data transfer in both perfect and realistic deployments. Furthermore, in terms of keeping source nodes connected to sink nodes, minimum hop routing is, on average, the optimal routing heuristic.

Minimum hop routing may cause a subset of nodes to expire more quickly than they would under a load balancing routing protocol. However, as discussed in Section 4.5, most load balancing routing protocols only distribute routing load and do not reduce it. Thus, the nodes may expire more quickly than with those routing protocols that actively reduce en-

ergy consumption. Furthermore, some load balancing routing protocols such as those that are energy aware incur additional overhead, since the remaining energy of nodes must be transmitted through the network, further draining the batteries of nodes.

11.3.2 Node Reliance Routing

Node reliance routing is shown to be a good approach for optimising (source, sink) connectivity as measured by CWT.

The analysis of results summarised in Section 11.2 show that for several of the deployments studied in Chapter 2, node reliance routing may produce the optimal (source, sink) connectivity and the optimal data transfer. Those deployments may therefore benefit from the use of node reliance routing.

11.3.3 Lexicographic Routing

Lexicographic ordering is used to form paths that avoid nodes of high cost by using several nodes of lower cost. In theory, by avoiding the highest cost nodes and instead expending energy on the low cost nodes, the important nodes could be prevented from expiring.

The results have shown that when used with node reliance routing heuristics, lexicographic routing results in poorer connectivity and total data transfer than shortest path routing provides on average. This behaviour occurs for two reasons, both of which stem from the fact that the lowest lexicographically ordered path will contain more nodes that the path of least total cost (the shortest path).

Firstly, since the lowest lexicographically ordered path contains more nodes, a message sent from source to sink must be transmitted and received more times than it would if it is sent along the shortest path. Therefore, the energy expenditure of the entire network is increased and so its total capacity drops.

Secondly, since the message travels through more nodes, it is broadcast more times. Consequently, the message will be overheard by more nodes and so more energy will be wasted by unnecessarily receiving and decoding the message.

There may be situations in which lexicographic routing is better than shortest path routing. However, shortest path routing always seems to be superior when considering node reliance heuristics and maximising total data transfer or connectivity.

11.3.4 Scalability

Calculating node reliance values requires a view of the entire network topology. As the network size grows, more data must be gathered in order to calculate the values. Consequently, node reliance routing protocols are of limited scalability.

One way to solve the scalability problem is to consider the use of hierarchical routing, which is discussed as a routing protocol module in Chapter 3. In hierarchical routing, the network is split into clusters. Since clusters are independent, calculating reliance values for nodes in a cluster would only require the topology of that cluster rather than the entire network.

Routing in a hierarchical network can proceed in two ways, either via clusterheads or by border nodes.

In the first case, each node belongs to exactly one cluster, which is managed by a clusterhead node. Each source forwards its data (possibly through multiple hops) to its clusterhead, which may amalgamate the data from multiple sources and forward it to a sink node.

If the network uses border nodes, it may be a necessary for a source to route messages through multiple clusters in order to reach a sink. A node that belongs to more than one cluster is known as a border node and is used to facilitate inter-cluster routing. A (source, sink) path in such a network is formed thusly:

- By representing clusters as sources, the set of (cluster, sink) paths can be enumerated and *cluster* reliance values can be calculated.
- Using the *cluster* reliance values, it is possible to form a shortest path P indicating the order in which clusters should be used, in order to reach a sink.
- For each cluster in P, the node reliance values of nodes within that cluster are calculated. Each node is treated as being a source. Border nodes that connect the cluster to the next cluster in P are treated as sink nodes.

- When data is routed from a source to a sink, it is sent along P. As the message enters each cluster, it is sent along the shortest path to the border node that connects the current cluster to the next in the sequence.

For example, consider the hierarchical network shown in Figure 11.1, which contains sink (Z) and 57 source nodes and has been split up into eight clusters. The sink forms a cluster by itself.

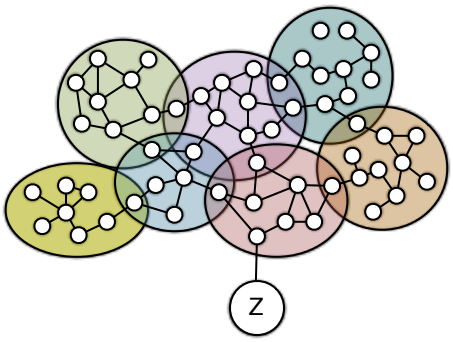


Figure 11.1: An example network split into clusters

First, the cluster reliance values must be calculated by treating each cluster as a source node. Figure 11.2 is a representation of Figure 11.1 in which the clusters have been replaced by sources. Each source has the same connectivity of the cluster that it replaced. For example, the yellow cluster is replaced by source A, which is connected to the green cluster (B) and the blue cluster (C).

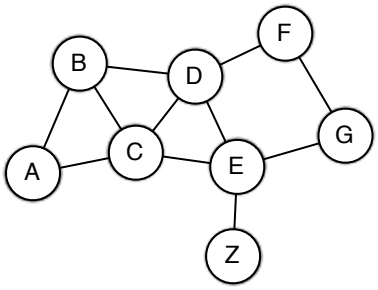


Figure 11.2: A representation of Figure 11.1 with a source replacing each cluster

Using the simplified network, it is possible to calculate the set of (cluster, sink) simplest paths and therefore the cluster reliance values. For example, cluster A has two simplest

paths (ACEZ and ABDEZ). Cluster B has two simplest paths (BCEZ and BDEZ). There is only one simplest path for each of clusters C, D and E (CEZ, DEZ and EZ). Finally, there are two simplest paths for cluster F (FDEZ and FGEZ) and one simplest path for cluster G (GEZ). The cluster reliance values are shown in Table 11.1.

Node	Relative Reliance							Absolute Reliance
	(A, Z)	(B, Z)	(C, Z)	(D, Z)	(E, Z)	(F, Z)	(G, Z)	
A	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.14
B	0.50	1.00	0.00	0.00	0.00	0.00	0.00	0.21
C	0.50	0.50	1.00	0.00	0.00	0.00	0.00	0.29
D	0.50	0.50	0.00	1.00	0.00	0.50	0.00	0.36
E	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.14
G	0.00	0.00	0.00	0.00	0.00	0.50	1.00	0.21
Z	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 11.1: Reliance values for the network of Figure 11.2

Using these values, it is possible to determine which sequence of clusters should be used in order to route a message from one cluster to a sink. For example, the shortest path from A to Z is ACEZ. Therefore, any message originating from a node in cluster A must pass through clusters C and E before reaching sink Z. It remains to be shown how the message should be routed within each cluster. Consider the yellow cluster (A) shown in detail in Figure 11.3.

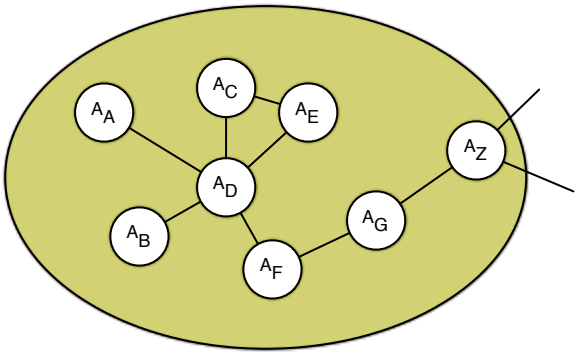


Figure 11.3: A detailed representation of cluster A from Figure 11.2

A message is to be routed from node A_A to sink Z. As previously stated, any message

destined for the sink that is forwarded to cluster A will be forwarded to cluster E. Border node A_Z is the only node that connects cluster A to cluster E. Therefore, node A_Z acts as the sink for cluster A. There is only one simplest path from A_A to A_Z, therefore the node reliance values are irrelevant. The path taken is $A_A A_D A_F A_G A_Z$.

The process is repeated for cluster C, which is shown in Figure 11.4.

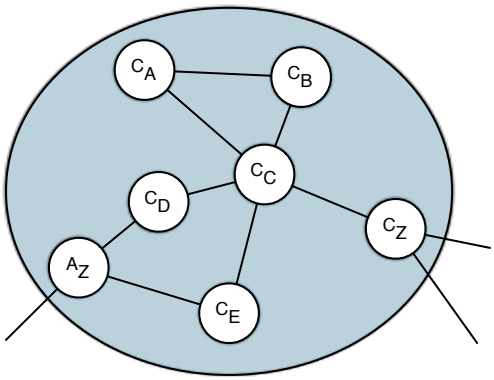


Figure 11.4: A detailed representation of cluster C from Figure 11.2

Within cluster C, each node is a source (since it may be the origin of a piece of data). Any message passing through cluster C to sink Z must be passed to cluster E. Therefore, border node C_Z is the only sink. There is only one simplest path from each node in cluster C to sink C_Z ($C_A C_C C_Z$, $C_B C_C C_Z$, $C_C C_Z$, $C_D C_C C_Z$, $C_E C_C C_Z$) with the exception of node A_Z which has two simplest paths ($A_Z C_D C_C C_Z$ and $A_Z C_E C_C C_Z$). The reliance values for the nodes in cluster C are shown in Table 11.2.

Node	Relative						Absolute
	(A_Z,C_Z)	(C_A,C_Z)	(C_B,C_Z)	(C_C,C_Z)	(C_D,C_Z)	(C_E,C_Z)	
A_Z	1.00	0.00	0.00	0.00	0.00	0.00	0.17
C_A	0.00	1.00	0.00	0.00	0.00	0.00	0.17
C_B	0.00	0.00	1.00	0.00	0.00	0.00	0.17
C_C	1.00	1.00	1.00	1.00	1.00	1.00	1.00
C_D	0.50	0.00	0.00	0.00	1.00	0.00	0.25
C_E	0.50	0.00	0.00	0.00	0.00	1.00	0.25
C_Z	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 11.2: Reliance values for the network of Figure 11.4

Using the node reliance values for cluster C, the two paths from A_Z to C_Z are of equal cost. Therefore, either path may be used. When a message reaches node C_Z, it enters cluster E, which is shown in Figure 11.5.

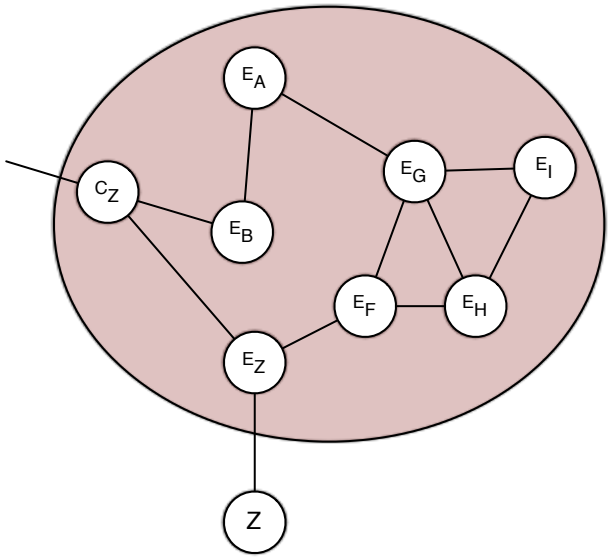


Figure 11.5: A detailed representation of cluster E from Figure 11.2

Only one border node (E_Z) in cluster E is connected to the sink, therefore E_Z is treated as the sink for this cluster. Only one simplest path exists from node C_Z to border node E_Z ($C_Z E_Z$) therefore the node reliance values are irrelevant.

Pooling the routing information from each cluster together, the path taken for a message from node A_A to sink Z is $A_A A_D A_F A_G A_Z C_D C_C C_Z E_Z Z$ (although C_D may be substituted for C_E).

If all nodes are treated as a single network, approximately $58^2 = 3364$ messages may have to be exchanged. By splitting the network into 8 clusters containing 8, 10, 7, 12, 8, 12 and 11 nodes (some nodes appear in more than one cluster), the number of messages that must be exchanged is reduced to $8^2 + 8^2 + 10^2 + 7^2 + 12^2 + 8^2 + 12^2 + 11^2 = 750$. Thus, by splitting up the network, the scalability of the heuristic can be improved.

11.4 Further Work

There are several possibilities for extending the work presented in this thesis:

- The complementary (source, sink) vertex cut of a node is a different model of node reliance and reflects the smallest number of nodes that must be removed to make that node a bottleneck.
- By considering the extent to which each node overhears transmissions of other nodes it may be possible to further extend the lifetime of nodes.
- By not recalculating node reliance values when nodes expire, it may be possible to reduce the energy expenditure of nodes in recalculating the network topology.
- It may be possible to use node reliance for other things, such as pinpointing points of failure and using that information to design better networks.
- Finally, node reliance may be combined with other schemes such as load balancing or minimum energy in order to further extend the lifetime of the network or to handle networks in which the initial energy of nodes is not homogeneous.

These possibilities are examined in further detail in the following sections.

11.4.1 Minimum Complementary (source, sink) Vertex Cut

The minimum complementary (source, sink) vertex cut of a node N is another model of node importance and is defined as being the smallest number of nodes that must be removed from the network in order that all data must travel through node N from sources to sinks. It can therefore be used to represent how much node N is relied upon in routing from sources to sinks. A *cut* is a set of nodes (or alternatively edges) whose removal partitions the network. In this case, the cut partitions the sources and sinks. If the cut contains many nodes, then there are many other nodes that may be used in place of N, and so it has little value. Conversely, if the cut contains few nodes, then N is very important since there are few other options available to using N.

The advantage of using the minimum complementary (source, sink) vertex cut is that it directly represents the level of redundancy that is available for a particular node. Conversely, the models presented in Chapter 6 represent the degree to which each node appears on paths from sources to sinks, which may indirectly represent the level of redundancy available.

The minimum complementary (source, sink) vertex cut is closely related to the minimum (s, t) cut problem [108], which determines either the set of nodes or the set of edges that must be removed in order to partition a source s from a sink t. The problem may be solved in polynomial time. It differs from the minimum complementary (source, sink) vertex cut in that the latter requires the solution to make a specific node a bottleneck rather than partitioning sources and sinks.

The obvious solution to calculating the minimum complementary (source, sink) vertex cut of a node N is to remove N from the network, and then calculate C, the minimum (s, t) cut of the resulting graph. However, this solution is invalid because C might partition the sources from sinks rather than making node N a bottleneck.

For example, Figure 11.6a shows a simple network with a source A and sink Z. The aim is to find the minimum complementary (source, sink) vertex cut of node D. Using the above solution, the first step is to remove node D, thus forming graph G' shown in Figure 11.6b. By observation, it can be seen that the minimum (s, t) cut in G' is node F and has size 1. However, this answer is incorrect because the removal of node F does not cause node D to become a bottleneck in Figure 11.6a.

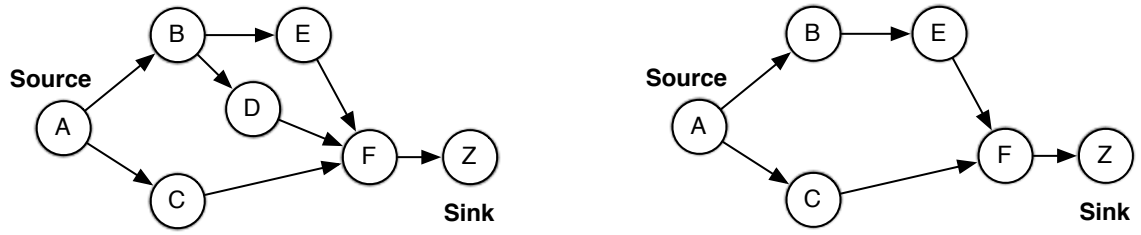


Figure 11.6: Removing D does not help to find the minimum complementary (source, sink) vertex cut of D

In Figure 11.6a, the correct answer is 2. By removing nodes C and E, node D becomes a bottleneck. There is no smaller number of nodes that can be removed to cause D to become a bottleneck.

It has not been possible so far to develop an algorithm to efficiently find the minimum complementary (source, sink) vertex cut, nor has there been any success in proving that an efficient time algorithm even exists. Since a cut may consist of any combination of any number of nodes in the network, the number of cuts has a factorial growth with respect to the number of nodes in the network. It is possible, therefore, that such a problem is non-polynomial.

The development of an algorithm for finding the minimum complementary (source, sink) vertex cut, or a proof that such an algorithm would be impractical has been left as further work.

11.4.2 Avoiding Overhearing

Since radios do not usually have the ability to unicast, a node may expend energy receiving a message and decoding it before determining that it is not the intended recipient. Overhearing can be avoided in hardware by the use of multiple frequencies or scheduling the communications of each node so that radios can be switched off when not in use. However, using multiple frequencies may increase the cost of the hardware and scheduling communication between nodes requires bidirectional links. Therefore, if neither of these options are suitable, taking overhearing into account when calculating a node's reliance value may improve the accuracy of node reliance values appreciably and further increase the time for which sources are connected to sinks.

In such a scheme, a node's reliance is composed from two components:

- an absolute reliance value as described in one of the heuristics of Chapter 6, and,
- and an inherited component from the node's neighbours.

The inherited component is derived from the sum of the absolute reliance values of those neighbours from which a node overhears messages. The intuition is that a well-connected node will overhear many messages and will consequently have a high reliance value reflecting the additional energy expenditure used as a consequence of overhearing. Similarly, a high reliance node that might overhear messages from a neighbour will cause that neighbour's reliance value to rise, thus discouraging its use in routing.

For example, consider the network shown in Figure 11.7. For simplicity, all edges between nodes are considered to be bidirectional. Reliance values are calculated based on simplest paths and are shown in Table 11.3.

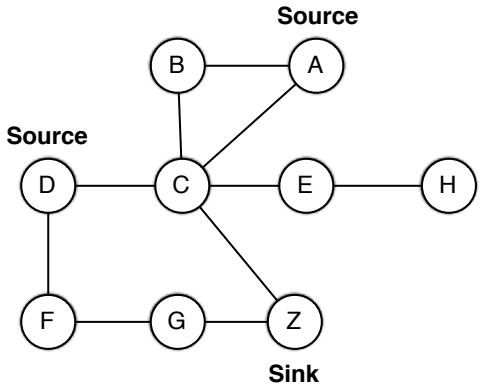


Figure 11.7: An example network with two sources, A and D

The inherited component of a node is the half of the sum of absolute reliance values of nodes whose transmissions may be received by that node. The value of a half is used under the assumption that the energy expended in receiving is approximately equal to that of transmitting, which is a fair assumption according to the numbers shown in Section 2.5.1. For example, node B may receive transmissions from source A and from node C. Consequently, the inherited component of B is equal to the half of the sum of the absolute reliance values of source A and node C $0.5(0.50 + 0.75) = 0.75$. The value of 0.5 is used because nodes that overhear a message only receive it whereas nodes that are used in routing must receive and transmit the message. Thus, the cost of overhearing is half of that encountered by nodes that forward the message.

Table 11.3 shows the inherited component of each node, together with the final reliance value, which is the sum of the inherited component and the absolute node reliance of that node.

Considering an inherited component changes the importance of nodes. For example, node C's importance increases significantly, since a transmission made by C will expend energy on nodes A, B, D, E, F and G. Thus, the use of node C must be avoided where possible in order to preserve other nodes.

Under this routing heuristic, the performance of lexicographic routing may improve. As already discussed, lexicographic routing can cause the use of longer paths in which more

Node	Absolute Reliance	Inherited Component	Final Reliance
A	0.50	0.375	0.875
B	0.00	0.625	0.625
C	0.75	1.00	1.75
D	0.50	0.50	1.00
E	0.00	0.375	0.375
F	0.25	0.375	0.625
G	0.25	0.625	0.875
Z	1.00	0.50	1.50

Table 11.3: Components of node reliance calculated from the network of Figure 11.7

transmissions are made and more messages are overheard, leading to increased energy expenditure. However, if node costs are adjusted to take overhearing into account, then longer paths of low reliance nodes should cause few messages to be overheard, thus reducing the energy expenditure of the network.

Analysis of this routing heuristic (possibly with lexicographic routing) is left to further work.

11.4.3 Determining Node Reliance Values Once

The routing protocols discussed in Chapter 8 make use of a network-wide sequence number, which is incremented whenever a sink stops receiving data from a source. The increase in sequence number is used to represent the possible expiration of a node in the network and causes each source to gather the network topology again and recalculate node reliance values.

Calculating the reliance values requires the expenditure of energy, since messages must be routed through the network in order to gather the new network topology. An alternative approach to this problem would be to simply determine which node had expired and use calculate the new cheapest path using the old node reliance values. The reliance values would no longer reflect the new network topology, but would instead reflect the importance of nodes when the network was first activated.

A future analysis could determine whether this strategy results in poorer routing decisions being made and whether the approach saved significant energy by reducing the number of times the topology must be gathered.

11.4.4 Other Uses for Node Reliance

There are other possible uses for node reliance. For example, since node reliance gives an importance value for each node in (source, sink) routing, it is possible to determine which parts of the network are heavily relied upon. Such information can be used in network design in order to increase the stability of the network.

For example, by using a simulator and determining the reliance value of each node in the network, it is possible to determine which nodes act as a point of failure in the network. A network designer may use this information to place additional nodes, thus reducing the extent to which any part of the network is relied upon and therefore increase the source diversity of the network for longer.

Another possible use for node reliance is in quality of service based routing protocols. Nodes with high reliance values are more relied upon than other nodes in the network and so are more likely to be congested. Such information could be used by sources in order to limit the quantity of data they send through high reliance nodes in order to improve the quality of service of the network.

11.4.5 Considering Initial Energy

The experiments discussed in Chapters 7, 9 and 10 make the assumption that all non-sink nodes in the network have the same initial energy level. In practice, the quality of energy cells can vary greatly. Some energy cells may also deteriorate faster than others, particularly due to environmental conditions such as weather or pressure.

If the available energy of nodes is likely to vary, one might consider a variation of the node reliance heuristic in which a node's reliance value is modified based on the relative proportion of energy that it has. For example, consider that the average initial node energy is 1. A node's reliance value can then be divided by the quantity of energy it has, relative

to the average. For example, a node whose initial energy is double the average would have its reliance value halved. A node whose initial energy is quarter the average would have its reliance value divided by 0.25 (i.e. multiplied by 4). The result of these modifications is that a node's use is encouraged if it has a lot of initial energy (even if it is of high importance) and discouraged if it initially has little energy (even if it is of low importance).

Further analysis needs to be carried out in order to determine how this heuristic modification copes with maintaining (source, sink) connectivity compared to other routing heuristics.

11.5 Finally

This thesis analyses the ability of different routing protocols to maintain (source, sink) connectivity. Minimum hop routing, whose use is commonly discouraged in the WSN literature is shown to be good at achieving a high connectivity and data transfer in perfect and realistic networks when combined with a global knowledge topology gathering scheme. A new routing family of heuristics and routing protocols known as node reliance is presented. In certain circumstances, node reliance has been shown to be the best routing protocol for maintaining (source, sink) connectivity and achieving a high total data transfer. Approaches for dealing with its possible scalability problems are addressed and avenues for further research are suggested whose implementation may increase its performance.

Glossary

Backpath

A path that is formed by reversing another path. Formation of a backpath inherently requires that links between nodes are *bidirectional*.

Backward path

A path from a sink to a source. Opposite is a *forward path*.

Bidirectional

Communication between a pair of nodes is the same in both directions, i.e. node A can receive the transmissions of node B if and only if node B can receive the transmissions of node A.

Contracted simplest paths

The set of paths formed by firstly finding the set of *simplest paths* from a source to a sink, then by removing those nodes whose presence is deterministic and finally by recalculating the set of *simplest paths*. Used in the *contraction model*.

Contraction model

A *node reliance* heuristic in which reliance values of nodes are calculated from the set of *contracted simplest paths*. Discussed in Section 6.1.4.

Costing

The second *routing protocol* task in which the cost of using each path is determined.

Dijkstra's algorithm

A greedy algorithm used for solving the *shortest path* problem [29].

Discovery

One of three *routing protocol* tasks, which determines how and when possible paths are found.

Enumeration

An option for the search module during the *discovery* task of a *routing protocol*. Each path between the sources and sinks are enumerated and collected at the sink nodes. General discussion in Section 3.1.3.4. The enumeration routing protocol for *node reliance* is discussed in Section 8.2.

Forward path

A path from a source to a sink. Opposite is a *backward path*.

Global Knowledge (GK)

An option for the search module during the *discovery* task of a *routing protocol*. Each node shares its topology data with every other node in the network so that all nodes gather a view of the entire network. Based on the link state advertisement (LSA) routing protocol. Discussed in Section 3.1.3.3.

Lexicographic ordering

A means of calculating *path cost* in which elements are arranged in descending numerical order and then dictionary sorted. Discussed in Section 3.2.2.2.

Max element

A *path cost* mechanism in which the cost of a path is based on the maximum cost of any node that lies on the path. Opposite is *Min element*.

Min element

A *path cost* mechanism in which the cost of a path is based on the minimum cost of any node that lies on the path. Opposite is *Max element*.

Minimum hop

A classification of *routing protocol* in which the optimal path contains the fewest number of intermediate nodes. May be calculated by considering the *path cost* as *shortest path* and *node cost* as unit weight or alternatively by considering *path cost* as *max element* and *node cost* as *number of hops*. Discussed in Section 4.2.

Link/Node cost

A module of the *costing* task that determines the cost associated with using a particular link or node. For example, *unit weight* considers each node as being of equal cost.

Node reliance

A new *routing heuristic* that represents the importance of a node in routing. May be either relative to a particular (source, sink) pair or absolute (to the entire network). Node reliance uses different models to represent the cost of each node, including the *simple paths model*, *simplest paths model* and *contraction model*.

Number of hops

A means of representing *link/node cost* based on the smallest number of nodes a message must travel through from that node to a sink node.

Path cost

A module of the *costing* task that determines the cost associated with a path. Typically a path's cost is a function of the *link/node cost* of the nodes or links that lie on that path. Discussed in Section 3.2.2.

Perfect communication

A model of communication between nodes in which links are *bidirectional* and transmissions are always received by nodes in range.

Realistic communication

A model of communication between nodes in which links may not be bidirectional and in which a transmission is only received by a node with some probability p. The model is said

to be realistic because it reflects the features of real wireless communication.

Routing protocol

A specific set of message exchanges and data structures that are used to collect data regarding the network in order to select paths for routing from sources to sinks. A routing protocol's operation may be separated into three *tasks*: *discovery*, *costing* and *selection*.

Routing heuristic

An algorithm for selecting which path to use in routing from sources to sinks. Is not concerned with collecting the data necessary to carry out the heuristic. A routing heuristic's operation may be separated into two *tasks*: *costing* and *selection*.

Selection

The third *routing protocol* task, which covers how paths are selected for routing based on the requirements of the application.

Shortest path

An option for the *path cost* module in which the cost of a path is equal to the sum of the cost of nodes that lie on that path. The cheapest path is the one with the smallest cost. The shortest path can be calculated using *Dijkstra's algorithm* if all link weights are non-negative. Discussed in Section 3.2.2.1.

Simple path

A sequence of nodes in which no node is repeated.

Simple paths model

A *node reliance* heuristic in which reliance values of nodes are calculated from the set of *simple paths* between sources and sinks. Discussed in Section 6.1.2.

Simplest path

A path P is simplest if it is not possible to form another path between the same (source, sink) pair by removing nodes from P.

Simplest paths model

A *node reliance* heuristic in which reliance values of nodes are calculated from the set of *simplest paths* between sources and sinks. Discussed in Section 6.1.3.

Unit disk graph (UDG)

A representation of possible communication between nodes in a network, assuming *perfect communication*. Each node has a transmission radius. A line is drawn between two nodes if one falls within the transmission radius of another. A line between two nodes indicates that direct communication between those nodes is possible. Communication between unlinked nodes always fails.

Unit weight

An option for the *link/node cost* module in which each node has the same cost (i.e. 1).

Appendix A

Node Placement Algorithms

This appendix provides details of the algorithms used to construct networks of particular shapes. Four different algorithms are considered:

- Uniform random node placement, in which nodes are placed in random locations such that the network is connected.
- Triangular node placement, where nodes are placed in a triangular formation in an aim to give each node six neighbours.
- Square node placement, where nodes are placed in a square formation in an aim to give each node four neighbours.
- Hexagonal node placement, in which nodes are placed in a hexagonal formation and the aim is to give each node three neighbours.

In order to define two nodes as being connected, it is assumed that communication between nodes is perfect. As previously discussed in Section 2.5.2, communication between nodes is unreliable. However, the concept of two nodes being connected is undefined when a transmission sent by one node is only received by the second node with some probability. Thus, it is necessary to assume that the nodes communicate perfectly when discussing node placement algorithms.

The algorithms used for each of these node placements are given in the following sections.

A.1 Uniform Random Node Placement

In a *uniform random node placement*, nodes are equally likely to be placed at any point in space, subject to certain constraints. In particular, the network remains connected. Thus, a new node must be added such that it is connected to at least one other node. Uniformly random placement is a common scenario in the analysis of routing protocols, since it allows for a wide range of node connectivities.

The intuition behind the algorithm is that (x, y) coordinates are repeatedly selected until a location is selected, which is within radio range of another node. Thus, the network remains connected as each node is added. To reduce the number of times that random (x, y) coordinates are selected, the coordinated must fall within radio range of the region given by the maximum and minimum coordinates of any node in the network. If a point outside this region is selected, it is guaranteed that the network will not be connected. Thus, there is no point considering coordinates outside that area.

The algorithm used to construct of a uniformly random, connected network of nodes in a square deployment area is as follows:

```
1  Node[] nodes;
2
3  for (i = 0; i < numberNodes; i++){
4     // First node is placed in the centre of the area
5     if (i == 0){
6        nodes[i].x = AREA-SIZE/2
7        nodes[i].y = AREA-SIZE/2
8        continue
9     }
10
11    // Determine the min/max (x,y) coordinates of all nodes
12    int minx, miny = AREA-SIZE
13    int maxx, maxy = 0
14
15    for (j = 0; j < i; j++){
16       if (nodes[j].x - MAX_RANGE < minx)
17          minx = nodes[j].x - MAX-RANGE
```

```
18        if (nodes[j].x + MAX_RANGE > maxx)
19           maxx = nodes[j].x + MAX-RANGE
20        if (nodes[j].y - MAX_RANGE < miny)
21           miny = nodes[j].y - MAX-RANGE
22        if (nodes[j].y + MAX_RANGE > maxy)
23           maxy = nodes[j].y + MAX-RANGE
24     }
25
26     while (true){
27        // Keep picking random (x, y) coordinates
28        nodes[i].x = random(minx, maxx)
29        nodes[i].y = random(miny, maxy)
30
31        // Stop if these coordinates are in range of another node
32        boolean done = false;
33        for (k = 0; k < i; k++){
34           if (distance(nodes[k], nodes[i]) <= MAX-RANGE){
35              done = true
36              break
37           }
38        }
39
40        if (done)
41           break
42     }
43  }
```

area-size refers to one of the dimensions of the square deployment area.

max-range is used to indicate the theoretical radio range of nodes. For example, Section 5.4.1 has suggested that on the TMote sky, this value should be 46 metres.

An example network is shown in Figure A.1a. The dashed lines show the region which is within radio range of the min/max (x, y) coordinates of all nodes. By selecting coordinates within this region, it is less likely that a node will be placed such that the network is unconnected. Thus, fewer sets of random coordinates will be required and the network can be quickly generated. However, in Figure A.1b it is shown that it is still possible to select

coordinates that would result in an unconnected network. For example, the coordinates selected for node 6 lie within the dashed region. However, that location is not within range of any nodes.

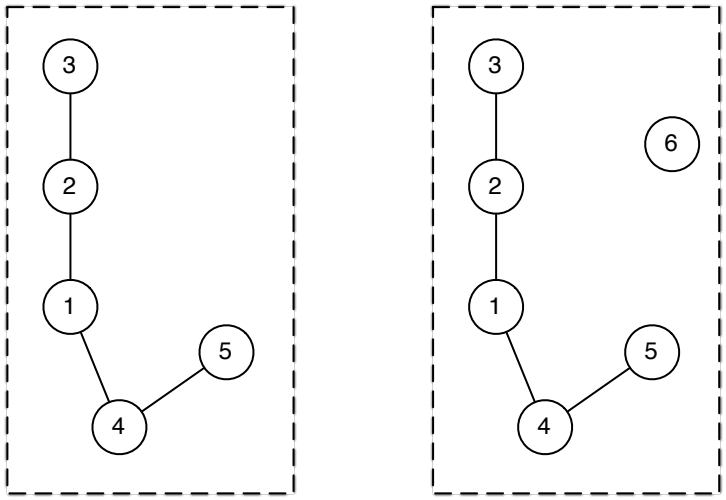


Figure A.1: The dashed lines indicate the region in which new nodes may be placed

Another option for the algorithm shown above was to randomly select an already placed node and to select a random point up to *max-range* metres away from that node. In practice, however, the probability of picking a node in a dense area of the partially constructed network was significantly higher than picking a node from a sparse section of the network. Consequently, nodes were more likely to be added to the network where the density was already high, resulting in a non-uniform distribution of nodes.

A.2 Triangular Node Placement

In a triangular node placement, nodes are placed in such a manner that the positions of the nodes form a tessellation of triangles. For large numbers of nodes, the majority of nodes will have a degree of six, the exception being those nodes at the edges of the deployment area.

In the algorithm, new nodes are added to the network such that the degree of the oldest node without six (*max-neighbours*) neighbours is increased. Neighbours are added to a

node in clockwise order starting at *bearing1* which is 30 degrees, then *bearing2* which is 90 degrees, then *bearing3* at 150 degrees and so on up until *bearing6* at 330 degrees. Thus, nodes are added in clockwise order every 60 degrees.

The algorithm for the construction of a triangular, connected network of nodes is as follows:

```
1  Node[] nodes
2
3  for (i = 0; i < numberNodes; i++){
4     // The first node is placed in the centre of the area
5     if (i == 0){
6        nodes[i].x = AREA-SIZE/2
7        nodes[i].y = AREA-SIZE/2
8        continue
9     }
10
11    // workingNode is the oldest node without six neighbours
12    Node workingNode
13    for (j = 0; j < i; j++){
14       if (nodes[j].neighbours < MAX_NEIGHBOURS){
15          workingNode = nodes[j]
16          break
17       }
18    }
19
20    // Find first bearing at which workingNode has no neighbour
21    // Add new node to that position
22    if (!workingNode.hasNodeAtBearing(BEARING1)){
23       nodes[i].x = workingNode.x + sin(60) * MAX-RANGE
24       nodes[i].y = workingNode.y - cos(60) * MAX-RANGE
25    } else if (!workingNode.hasNodeAtBearing(BEARING2)){
26       nodes[i].x = workingNode.x + MAX-RANGE
27       nodes[i].y = workingNode.y
28    } else if (!workingNode.hasNodeAtBearing(BEARING3)){
29       nodes[i].x = workingNode.x + sin(60) * MAX-RANGE
30       nodes[i].y = workingNode.y + cos(60) * MAX-RANGE
31    } else if (!workingNode.hasNodeAtBearing(BEARING4)){
```

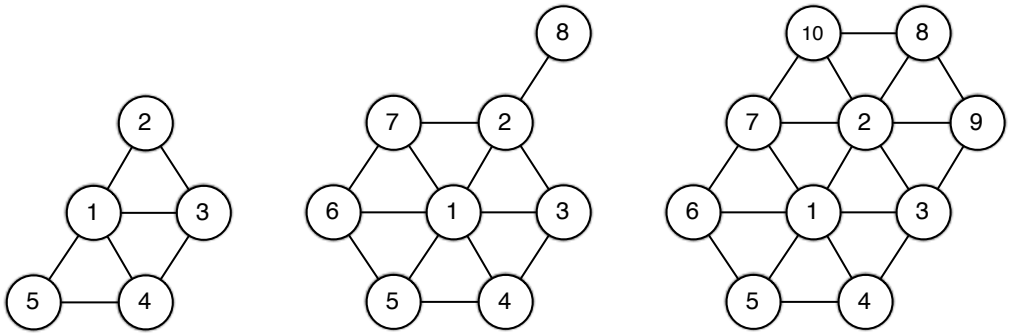
```
32        nodes[i].x = workingNode.x - sin(60) * MAX-RANGE
33        nodes[i].y = workingNode.y + cos(60) * MAX-RANGE
34     } else if (!workingNode.hasNodeAtBearing(BEARING5)){
35        nodes[i].x = workingNode.x - MAX-RANGE
36        nodes[i].y = workingNode.y
37     } else if (!workingNode.hasNodeAtBearing(BEARING6)){
38        nodes[i].x = workingNode.x - sin(60) * MAX-RANGE
39        nodes[i].y = workingNode.y - cos(60) * MAX-RANGE
40     }
41  }
```

As before, *area-size* refers to one of the dimensions of the square deployment area and *max-range* is used to indicate the theoretical radio range of nodes.

Figure A.2a shows the addition of nodes to a network. Node 1 is added to the centre of the area. When node 2 is due to be added, the oldest node without a degree of six is node 1, which currently has no neighbours. Thus, node 2 is added at a bearing of 30 degrees to node 1. Similarly, when node 3 is due to be added, node 1 remains the oldest node without a degree of six and so node 2 is added at a bearing of 90 degrees to node 1. Eventually, the addition of node 7, at a bearing of 330 degrees to node 1, gives node 1 a degree of six.

When node 8 is to be added, the oldest node without a degree of six is node 2. Node 2 has no node at a bearing of 30 degrees, and thus, node 8 is added to that position as shown in Figure A.2b. Similarly, node 9 is added at a bearing of 90 degrees. When node 10 is due to be added, it must be added at a bearing of 330 degrees since node 2 already has nodes in the other positions. Thus, with the addition of node 10, node 2 achieves a degree of six as shown in Figure A.2c. At this point, the next oldest node without a degree of six is node 3 and future nodes will be added there.

Another option for the algorithm was to add neighbours to nodes with the highest node degrees, excluding those nodes with a degree of six. In practice, however, this resulted in a small number of nodes with a degree of six and a very large number of nodes with much smaller degrees. It was therefore unreasonable to claim that the majority of the network was triangular. By using the algorithm shown above, the network expands in all directions and each node eventually achieves a degree of six. Furthermore, the entire formation resembles the shape being formed. In the case of the triangular node placement, the entire formation



(a) (b) (c)

Figure A.2: Diagrams showing the formation of a triangular network

of nodes resembles a single large triangle.

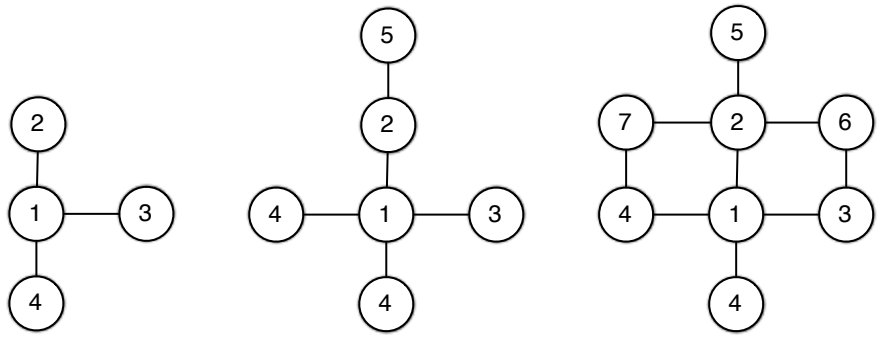
A.3 Square Node Placement

A square node placement behaves similarly to a triangular node placement. The differences are that nodes form a tessellation of squares, the majority of nodes in a large square node placement have a degree of four and nodes are spaced by 90 degree angles starting from a bearing of 0 degrees. Consequently, *max-neighbours* is 4 and *bearing1*, *bearing2*, *bearing3* and *bearing4* are 0, 90, 180 and 270 degrees respectively. The shape of the overall placement also resembles a square.

The corresponding algorithm is a trivial modification of that shown for Triangular Node Placements in Section A.2 and is not repeated here. Figures A.3a, A.3b and A.3c show how the network shape forms as nodes are added to the network.

A.4 Hexagonal Node Placement

The hexagonal node placement differs slightly from that of the triangular and square node placements. Nodes form a tessellation of hexagons, the majority of nodes have a degree of three and are spaced by 120 degree angles as might be expected. However, the positions of the neighbouring nodes alternates. On some nodes, the neighbours are at bearings of 30,



(a) (b) (c)

Figure A.3: Diagrams showing the formation of a square network

150 and 270 degrees and at other nodes, the neighbours are at bearings of 90, 210 and 330 degrees.

As in the other algorithms, new nodes are added in such a manner to increase the degree of the oldest node with fewer than three neighbours. Nodes are still added in a clockwise manner. However, the bearings of the neighbours is dictated by those neighbours that are already present. For example, if the oldest node without a degree of three has a neighbour at 150 degrees, then its other neighbours should be at 30 and 270 degrees. Conversely if it has a neighbour at 330 degrees then its other neighbours should be at 90 and 210 degrees. If a node has no neighbours, which is the special case after node 1 has been placed, then the next node is placed at a bearing of 90 degrees.

The algorithm for this placement is shown here:

```
1  Node[] nodes;
2
3  for (i = 0; i < numberNodes; i++){
4     // The initial node is placed in the centre
5     if (i == 0){
6        nodes[i].x = AREA-SIZE/2
7        nodes[i].y = AREA-SIZE/2
8        continue;
9     }
10
11    // workingNode is the oldest node without a degree of three
```

```
12    Node workingNode;
13    for (j = 0; j < i; j++){
14       if (nodes[j].neighbours < MAX_NEIGHBOURS){
15          workingNode = nodes[j];
16          break;
17       }
18    }
19
20    if (workingNode.hasNoNeighbours()){
21       // Special case: Place node at a bearing of 90 degrees
22       nodes[i].x = workingNode.x + MAX-RANGE;
23       nodes[i].y = workingNode.y;
24    } else if (workingNode.hasNodeAtBearing(BEARING1A)
25    || workingNode.hasNodeAtBearing(BEARING1B)
26    || workingNode.hasNodeAtBearing(BEARING1C)){
27       // Neighbours are at bearings of 90, 210 and 330 degrees
28       if (!workingNode.hasNodeAtBearing(BEARING1A)){
29          // Add neighbour at bearing of 90 degrees
30          nodes[i].x = workingNode.x + MAX-RANGE;
31          nodes[i].y = workingNode.y;
32       } else if (!workingNode.hasNodeAtBearing(BEARING1B)){
33          // Add neighbour at bearing of 210 degrees
34          nodes[i].x = workingNode.x - sin(30) * MAX-RANGE;
35          nodes[i].y = workingNode.y + cos(30) * MAX-RANGE;
36       } else if (!workingNode.hasNodeAtBearing(BEARING1C)){
37          // Add neighbour at bearing of 330 degrees
38          nodes[i].x = workingNode.x - sin(30) * MAX-RANGE;
39          nodes[i].y = workingNode.y - cos(30) * MAX-RANGE;
40       }
41    } else if (workingNode.hasNodeAtBearing(BEARING2A)
42    || workingNode.hasNodeAtBearing(BEARING2B)
43    || workingNode.hasNodeAtBearing(BEARING2C)){
44       // Neighbours are at bearings of 30, 150 and 270 degrees
45       if (!workingNode.hasNodeAtBearing(BEARING2A)){
46          // Add neighbour at bearing of 30 degrees
47          nodes[i].x = workingNode.x + sin(30) * MAX-RANGE;
48          nodes[i].y = workingNode.y - cos(30) * MAX-RANGE;
```

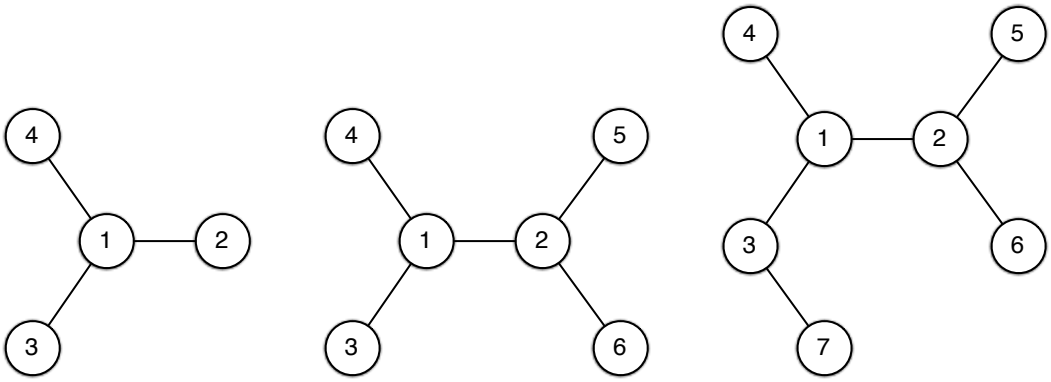
```
49        } else if (!workingNode.hasNodeAtBearing(BEARING2B)){
50           // Add neighbour at bearing of 150 degrees
51           nodes[i].x = workingNode.x + sin(30) * MAX-RANGE;
52           nodes[i].y = workingNode.y + cos(30) * MAX-RANGE;
53        } else if (!workingNode.hasNodeAtBearing(BEARING2C)){
54           // Add neighbour at bearing of 270 degrees
55           nodes[i].x = workingNode.x - MAX-RANGE;
56           nodes[i].y = workingNode.y;
57        }
58     }
59}
```

As before, *area-size* refers to one of the dimensions of the square deployment area and *max-range* is used to indicate the theoretical radio range of nodes. In this algorithm, *max-neighbours* is 3.

Figure A.4a shows the addition of nodes to a network. Node 1 is added to the centre of the area. When node 2 is due to be added, the oldest node without a degree of three is node 1, which currently has no neighbours. Thus, node 2 is added at a bearing of 90 degrees to node 1. When node 3 is due to be added, node 1 remains the oldest node without a degree of three. Since node 1 already has a neighbour at a bearing of 90 degrees, it indicates that the next neighbour should be added at 210 degrees. The addition of node 4 at a bearing of 330 degrees to node 1, gives node 1 a degree of three.

When node 5 is to be added, the oldest node without a degree of three is node 2. Node 2 already has a neighbour, i.e. node 1, at a bearing of 270 degrees. Thus, the neighbours of node 2 should be at 30, 150 and 270 degrees. Thus, node 5 is placed at a bearing of 30 degrees to node 1 and node 6 is placed at a bearing of 150 degrees, thus forming the network shown in Figure A.4b.

The next oldest node without a degree of three is node 3. Node 3 already has a neighbour at a bearing of 30 degrees, indicating that its other neighbours should be at 150 and 270 degrees. Thus, node 7 is added at a bearing of 150 degrees as shown in Figure A.4c.



(a) (b) (c)

Figure A.4: Diagrams showing the formation of a hexagonal network

Bibliography

[1] Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3):325–349, 2005.

[2] Alan D. Amis and Ravi Prakash. Load-balancing clusters in wireless ad hoc networks. In *3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'00)*, pages 25–32, Washington, DC, USA, 2000. IEEE Computer Society.

[3] Abolfazl Asudeh, Zainab Asudeh, and Ali Movaghar. Mehr: Multi-hop energy-aware hierarchical routing for wireless sensor networks. In *New Technologies, Mobility and Security*, pages 1–6, Tangier, Morocco, 2008.

[4] Athanassios Boulis. Castalia User Manual (http://tinyurl.com/25c3t9e). Retrieved 3rd March 2008.

[5] Australia, National ICT. Castlia Simulator (http://tinyurl.com/cfqo5o). Retrieved 6th August 2010.

[6] D. C. Baird. *Experimentation: An Introduction to Measurement Theory and Experiment Design*. Prentice Hall, third edition, 1995.

[7] Sebnem Baydere, Yasar Safkan, and Ozlem Durmaz. Lifetime analysis of reliable wireless sensor networks. *IEICE Transactions on Communications*, E88-B(6):2465–2472, 2005.

[8] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Power efficient monitoring management in sensor networks. In *IEEE Wireless Communiation and Networking Conference*, pages 2329–2334, Atlanta, 2004.

[9] Jan Beutel, Stephan Gruber, Andreas Hasler, Roman Lim, Andreas Meier, Christian Plessl, Igor Talzi, Lothar Thiele, Christian Tschudin, Matthias Woehrle, and Mustafa Yuecel. Permadaq: A scientific instrument for precision sensing and data recovery in environmental extremes. In *The 8th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 265–276, San Francisco, USA, 2009.

[10] Edoardo Biagioni and Shu Hui Chen. A reliability layer for ad-hoc wireless sensor network routing. In *37th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii, 2004. IEEE Computer Society.

[11] Douglas M. Blough and Paolo Santi. Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks. In *The 8th Internatonal Conference on Mobile Computing and Networking*, pages 183–192, Atlanta, Georgia, US, 2002.

[12] Azzedine Boukerche, Xuzhen Cheng, and Joseph Linus. A performance evaluation of a novel energy-aware data-centric routing algorithm in wireless sensor networks. *Wireless Networks*, 11(5):619–635, 2005.

[13] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[14] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc networking routing protocols. In *International Conference on Mobile Computing and Networking*, pages 85–97, Dallas, Texas, United States, 1998.

[15] Qing Cao, Tarek F. Abdelzaher, Tian He, and John A. Stankovic. Towards optimal sleep scheduling in sensor networks for rare event detection. In *The Fourth International Conference on Information Processing in Sensor Networks*, pages 20 – 27, Los Angeles, California, USA, 2005.

[16] Bogdan Carbunar, Ananth Grama, and Jan Vitek. Redundancy and coverage detection in sensor networks. *ACM Transactions on Sensor Networks*, 2(1):94–128, 2006.

[17] Matteo Ceriotti, Luca Mottola, Gian Pietro, Amy L. Murphy, and Stefan Gun. Monitoring heritage buildings with wireless sensor networks: The torre aquila deployment. In *ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 277–288, San Francisco, California, USA, 2009.

[18] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. Habitat monitoring: Application driver for wireless communications technology. In *2001 ACM SIGCOMM Workshop on Data Communications*, 2001.

[19] Anantha Chandrakasan, Rajeevan Amirtharajah, SeongHwan Cho, James Goodman, Gangadhar Konduri, Joanna Kulik, Wendi Rabiner, and Alice Wang. Design considerations for distributed microsensor systems. In *IEEE 1999 Custom Integrated Circuits Conference (CICC '99)*, pages 279 – 286, San Diego, CA, USA, 1999. IEEE.

[20] Jae-Hwan Chang and Leandros Tassiulas. Routing for maximum system lifetime in wireless ad-hoc networks. In *37th Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, 1999.

[21] Jae-Hwan Chang and Leandros Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *INFOCOMM*, pages 22–31, Tel Aviv, Israel, 2000. IEEE Computer Society.

[22] Jae-Hwan Chang and Leandros Tassiulas. Maximum lifetime routing in wireless sensor networks. *IEEE/ACM Transactions on Networking*, 12(4):609–619, 2004.

[23] Chipcon. CC2420 data sheet (http://tinyurl.com/2c3oscp). Retrieved 6th August 2010.

[24] David Culler, Deborah Estrin, and Mani Srivastava. Overview of sensor networks. *Computer*, 37(8):41–49, 2004.

[25] Hui Dai and Richard Han. A node-centric load balancing algorithm for wireless sensor networks. In *IEEE GLOBECOM - Wireless Communications*, volume 1, pages 548 – 552, San Francisco, USA, 2003. IEEE Communications Society.

[26] Koustuv Dasgupta, Konstantinos Kalpakis, and Parag Namjoshi. An efficient clustering-based heuristic for data gathering and aggregation in sensor networks.

In *IEEE Wireless Communications and Networking Conference*, pages 1948–1953, New Oelans, Louisiana, USA, 2003.

[27] Reinhard Diestel. *Graph Theory*. Springer-Verlag Heidelberg, electronic edition 3 edition, 2005.

[28] Isabel Dietrich and Falko Dressler. On the lifetime of wireless sensor networks. *ACM Transactions on Sensor Networks*, 5(1):5:1 – 5:39, 2009.

[29] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[30] Nikos Dimokas, Dimitrios Katsaros, and Yannis Manolopoulos. Node clustering in wireless sensor networks by considering structural characteristics of the network graph. In *International Conference on Information Technology*, pages 122–127, Las Vegas, Nevada, USA, 2007. IEEE Computer Society.

[31] L. Doherty, B. A. Warneke, B. E. Boser, and K. S. J. Pister. Energy and performance considerations for smart dust. *International Journal of Parallel Distributed Systems and Networks*, 4(3):121 – 133, 2001.

[32] Adam Dunkels, Niclas Finne, Joakim Eriksson, and Thiemo Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *Conference On Embedded Networked Sensor Systems*, pages 15–28, Boulder, Colorado, USA, 2006. ACM Press.

[33] Deborah Estrin, Ramesh Govindan, John S. Heidemann, and Satish Kumar. Next century challenges: Scalable coordination in sensor networks. In *Mobile Computing and Networking*, pages 263–270, Seattle, Washington, USA, 1999. ACM.

[34] Kevin Fall and Kanna Varadhan. The ns Manual (http://tinyurl.com/2csora3). Retrieved 6th August 2010.

[35] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *Mobile Computing and Communication Review*, 4(5), 2001.

[36] Yashar Ganjali and Abtin Keshavarzian. Load balancing in ad hoc networks: Single-path routing vs. multi-path routing. In *Twenty third Annual Joint Conference of the*

IEEE Computer and Communications Societies, volume 2, pages 1120–1125, Hong Kong, China, 2004.

[37] Arvind Giridhar and P. R. Kumar. Maximizing the functional lifetime of sensor networks. In *Fourth International Symposium on Information Processing in Sensor Networks*, pages 5–12, Los Angeles, California, USA, 2005.

[38] Himanshu Gupta, Zongheng Zhou, Samir R. Das, and Quinyi Gu. Connected sensor cover: Self-organisation of sensor networks for efficient query execution. *IEEE/ACM Transactions on Networking*, 14(1):55–67, 2006.

[39] Martin Haenggi. Routing in ad hoc networks - a wireless perspective. In *First International Conference on Broadband Networks*, pages 652–660, San Jos, California, USA, 2004. IEEE Computer Society.

[40] Martin Haenggi. On routing in random rayleigh fading networks. *IEEE Transactions on Wireless Communications*, 4(4):1553 – 1562, 2005.

[41] Fred Halsall. *Computer Networking and the Internet*. Addison Wesley, 2005.

[42] Hossam Hassanein and Jing Luo. Reliable energy aware routing in wireless sensor networks. In *Second IEEE Workshop on Dependability and Security in Sensor Networks and Systems*, pages 54–64, Columbia, Maryland, USA, 2006. IEEE.

[43] K. A. Hawick and H. A. James. Node importance ranking and scaling properties of some complex road networks, 2005.

[44] Tian He, John A. Stankovic, Chenyang Lu, and Tarek Abdelzaher. Speed: a stateless protocol for real-time communication in sensor networks. In *23rd International Conference on Distributed Computing Systems*, pages 46–55, Providence, Rhode Island, USA, 2003.

[45] Wendi B. Heinzelman, Anantha P. Chandrakasan, and Hari Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, 2002.

[46] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *33rd International Conference on System Sciences*, volume 8, pages 8020–8029, Big Island, Hawaii, USA, 2000. IEEE Computer Society.

[47] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *International Conference on Mobile Computing and Networking*, pages 174–185, Seattle, Washington, United States, 1999. ACM.

[48] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Boston, MA, USA, 2000. ACM.

[49] Jason Lester Hill. *System Architecture for Wireless Sensor Networks*. PhD thesis, Computer Science, 2003.

[50] Wen Hu, Nirupama Bulusu, Chun Tung Chou, Sanjay Jha, Andrew Taylor, and Van Nghia Tran. Design and evaluation of a hybrid sensor network for cane toad monitoring. *ACM Transactions on Sensor Networks*, 5(1):4:1 – 4:29, 2009.

[51] IEEE. Guidelines for 64-Bit Global Identifier (EUI-64) Registration Authority (http://tinyurl.com/2mqvbd). Retrieved 6th August 2010.

[52] Institute for Systems Research, University of Maryland. atemu - Sensor Network Emulator / Simulator (http://tinyurl.com/24pounu). Retrieved 6th August 2010.

[53] Xiaofan Jiang, Stephen Dawson-Haggerty, Prabal Dutta, and David Culler. Design and implementation of a high-fidelity ac metering network. In *The 8th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 253–264, San Francisco, California, USA, 2009.

[54] David B. Johnson, David A. Maltz, and Josh Broch. Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks. In Charles E. Perkins, editor, *Ad-Hoc Networking*, pages 139–172. Addison-Wesley, 2000.

[55] Eunjae Jung and D. M. H. Walker. Reliable energy efficient routing in wireless sensor networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 266–273, Washington, DC, USA, 2005.

[56] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for ”smart dust”. In *International Conference on Mobile Computing and Networking (MOBICOM)*, pages 271–278, Seattle, Washington, USA, 1999. ACM.

[57] Konstantinos Kalpakis, Koustuv Dasgupta, and Parag Namjoshi. Maximum lifetime data gathering and aggregation in wireless sensor networks. In *IEEE International Conference on Networking*, pages 685–696, Atlanta, Georgia, US, 2002. IEEE.

[58] Koushik Kar, Murali Kodialam, T. V. Lakshman, and Leandros Tassiulas. Routing for network capacity maximization in energy-constrained ad-hoc networks. In *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 673–681, San Francisco, USA, 2003.

[59] Holger Karl and Andreas Willig. A short survey of wireless sensor networks, 2003.

[60] Brad Karp and H. T. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *International Conference on Mobile Computing and Networking*, pages 243–254, Boston, Massachusetts, US, 2000.

[61] Loh Keong, Long Huan, and Pan Yi. An efficient and reliable routing protocol for wireless sensor networks. In *Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*, volume 2, pages 512–516, Messina, Italy, 2005. IEEE Computer Society.

[62] Younghun Kim, Thomas Schmid, Zainul M. Charbiwala, Jonathan Friedman, and Mani B. Srivastava. Nawms: Nonintrusive autonomous water monitoring system. In *6th ACM Conference on Embedded Networked Sensor Systems*, pages 309–322, Raleigh, North Carolina, USA, 2008.

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

[64] Murali Kodialam and T. V. Lakshman. Minimum interference routing with applications to mpls traffic engineering. In *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 884–893, Tel Aviv, Israel, 2000. IEEE.

[65] Santosh Kumar, Anish Arora, and Ten H. Lai. On the lifetime analysis of always-on wireless sensor network applications. In *IEEE International Conference on Mobile Adhoc and Sensor Systems*, Washington, D.C., USA, 2005.

[66] Sunil Kumar, Kashyap K. R. Hambhalta, Bin Zan, Fei Hu, and Yang Xiao. An energy-aware and intelligent cluster-based event detection scheme in wireless sensor networks. *International Journal of Sensor Networks*, 3(2):123–133, 2008.

[67] Sung-Ju Lee and Mario Gerla. Dynamic load-aware routing in ad hoc networks. In *IEEE International Conference on Communications*, volume 10, pages 3206–3210, Helsinki, Finland, 2000. IEEE.

[68] Sung-Ju Lee and Mario Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. In *IEEE International Conference on Communications*, volume 10, pages 3201–3205, Helsinki, Finland, 2001.

[69] Philip Levis and David Culler. Mat: A tiny virtual machine for sensor networks. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 85–95, New York, NY, USA, 2002. ACM Press.

[70] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Conference On Embedded Networked Sensor Systems*, pages 126–137, Los Angeles, California, USA, 2003. ACM.

[71] Mo Li and Yunhao Liu. Underground coal mine monitoring with wireless sensor networks. *ACM Transactions on Sensor Networks*, 5(2):10:1–10:29, 2009.

[72] Qun Li, Javed Aslam, and Daniela Rus. Online power-aware routing in wireless ad-hoc networks. In *International Conference on Mobile Computing and Networking*, pages 97–107, Rome, Italy, 2001. ACM.

[73] Y. Li and T. Newe. Wireless sensor networks - selection of routing protocol for applications. In *The Australian Telecommunication Networks and Applications Conference for 2006*, pages 334–338, Melbourne, Australia, 2006.

[74] Yanjun Li, Jiming Chen, Ruizhong Lin, and Zhi Wang. A reliable routing protocol design for wireless sensor networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*, pages 58–61, Washington, DC, USA, 2005.

[75] Yuxi Li, Janelle Harms, and Robert Holte. Optimal traffic-oblivious energy-aware routing for multihop wireless networks. In *25th IEEE International Conference on Computer Communications*, Barcelona, Spain, 2006. IEEE.

[76] Longbi Lin, Ness B. Shroff, and R. Srikant. Asymptotically optimal power-aware routing for multihop wireless networks with renewable energy sources. In *24th Joint*

Annual Conference of the IEEE Computer and Communications Societies, volume 2, pages 1262 – 1272, Miami, FL, USA, 2005. IEEE.

[77] Longbi Lin, Ness B. Shroff, and R. Srikant. Energy-aware routing in sensor networks: A large system approach. *Ad Hoc Networks*, 5(6):818–831, 2007.

[78] Stephanie Lindsey and Cauligi S. Raghavendra. Pegasis: Power-efficient gathering in sensor information systems. In *IEEE Aerospace Conference*, pages 1125–1130, Big Sky, MT, USA, 2002. IEEE.

[79] Raminder P. Mann, Kamesh R. Namuduri, and Ravi Pendse. Energy-aware routing protocol for ad hoc wireless sensor networks. *Journal on Wireless Communications and Networking*, 5(5):635–644, 2005.

[80] Vivek Mhatre, Catherine Rosenberg, Daniel Kofman, Ravi Mazumdar, and Ness Shroff. A minimum cost heterogeneous sensor network with a lifetime constraint. *IEEE Transactions on Mobile Computing*, 4(1):4 – 15, 2005.

[81] Emiliano Miluzzo, Nicholas D. Lane, Kristf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng, and Andrew T. Campbell. Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application. In *6th ACM Conference on Embedded Networked Sensor Systems*, pages 337–350, Raleigh, North Carolina, USA, 2008.

[82] Yoshitsugu Obashi, Huifang Chen, Hiroshi Mineno, and Tadanori Mizuno. An energy-aware routing scheme with node relay willingness in wireless sensor networks. *International Journal of Innovative Computing, Information and Control*, 3(3):565–574, 2007.

[83] Stephan Olariu and Ivan Stojmenovic. Design guidelines for maximizing lifetime and avoiding energy holes in sensor networks with uniform distribution and uniform reporting. In *25th IEEE International Conference on Computer Communications*, pages 1–12, Barcelona, Spain, 2006.

[84] OMNeT++ Community. OMNeT++ (http://tinyurl.com/yve96j). Retrieved 6th August 2010.

[85] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Professional, 1994.

[86] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web, 1998.

[87] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1405–1413, Kobe, Japan, 1997.

[88] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM Computer Communication Review*, 24(4):234–244, 1994.

[89] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, pages 90–100, New Orleans, LA, USA, 1999. IEEE Computer Society.

[90] Hai N. Pham, Dimosthenis Pediaditakis, and Athanassios Boulis. From simulation to real deployments in wsn and back. In Ieee, editor, *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–6, Espoo, Finland, 2007. IEEE Computer Society.

[91] Thuy Lien Pham, I. Lavallee, M. Bui, and Si Hoang Do. A distributed algorithm for the maximum flow problem. In *The 4th International Symposium on Parallel and Distributed Computing*, pages 131–138, Lille, 2005. IEEE Computer Society.

[92] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[93] Nissanka Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *Proceedings of the 6th Annual ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 32–43, Boston, Massachusetts, United States, 2000. ACM Press.

[94] Vijay Raghunathan, Curt Schurgers, Sung Park, and Mani Srivastava. Energy-aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2):40 – 50, 2002.

[95] Mohammad Abdur Razzaque, Simon Dobson, and Paddy Nixon. Cross-layer architectures for autonomic communications. *Journal of Network and Systems Management*, 15(1):13–27, 2007.

[96] K. Romer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, 2004.

[97] Shad Roundy, Paul K. Wright, and Jan Rabaey. A study of low level vibrations as a power source for wireless sensor nodes. *Computer Communications*, 26(11):11311144, 2003.

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

[99] Andreas Savvides, Chih-Chieh Han, and Mani B. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Mobile Computing and Networking*, pages 166–179, Rome, Italy, 2001.

[100] Curt Schurgers and Mani B. Srivastava. Energy efficient routing in wireless sensor networks. In *Military Communications Conference, 2001*, volume 1, pages 357–361, Virginia, USA, 2001. IEEE.

[101] Sentilla. Tmote Sky Datasheet (http://tinyurl.com/27vpw9w). Accessed: 10th May 2010.

[102] Kewei Sha, Junzhao Du, and Weisong Shi. Wear: a balanced, fault-tolerant, energy-aware routing protocol in wsns. *International Journal of Sensor Networks*, 1(3/4):156–168, 2006.

[103] Rahul C. Shah and Jan M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *IEEE Wireless Communications and Networking Conference*, volume 1, pages 350 – 355, Orlando, Florida, USA, 2002. IEEE.

[104] Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *2nd International Conference on Embedded Networked Sensor Systems*, pages 188 – 200, Baltimore, Marland, USA, 2004. ACM.

[105] Suresh Singh, Mike Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Mobile Computing and Networking*, pages 181–190, Dallas, Texas, United States, 1998. ACM.

[106] Katayoun Sohrabi, Jay Gao, Vishal Ailawadhi, and Gregory J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, 2000.

[107] Vikram Srinivasan, Carla F. Chiasserini, Pavan Nuggehalli, and Ramesh R. Rao. Optimal rate allocation and traffic splits for energy efficient routing in ad hoc networks. In *INFOCOM 2001*, 2002.

[108] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997.

[109] Ivan Stojmenovic, Amiya Nayak, and Johnson Kuruvila. Design guidelines for routing protocols in ad hoc and sensor networks with a realistic physical layer. *IEEE Communications Magazine (Ad Hoc and Sensor Networks Series)*, 43(3):101–106, 2005.

[110] R. Sumathi, M. G. Srinivasa, and R. Srinivasan. An approach to load balancing and network longevity using dynamic adaptive routing in wireless sensor networks. In *Pervasive Computing and Communications*, pages 366–371, Hong Kong, 2008.

[111] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An analysis of a large scale habitat monitoring application. In *2nd international conference on Embedded networked sensor systems*, pages 214–226, Baltimore, MD, USA, 2004. ACM.

[112] Andrew S. Tanenbaum, Chandana Gamage, and Bruno Crispo. Taking sensor networks from the lab to the jungle. *Computer*, 39(8):98–100, 2006.

[113] Hui Tian, Hong Shen, and Teruo Matsuzawa. Random walk routing for wireless sensor networks. In *Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 196–200, Dalian, China, 2005.

[114] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong.

A macroscope in the redwoods. In *3rd International Conference on Embedded Networked Sensor Systems*, pages 51–63, San Diego, California, USA, 2005.

[115] Duc A. Tran and Harish Raghavendra. Congestion adaptive routing in mobile ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(11):1294–1305, 2006.

[116] Andrs Varga. The omnet++ discrete event simulation system. In *European Simulation Multiconference*, Prague, Czech Republic, 2001.

[117] R. Vidhyapriya and P. T. Vanathi. Energy aware routing for wireless sensor networks. In *International Conference on Signal Processing, Communications and Networking*, pages 545 – 550, Chennai, India, 2007.

[118] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(June):440 – 442, 1998.

[119] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Lonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *The 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 381–396, Seattle, Washington, 2006.

[120] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Los Angeles, California, USA*, pages 14–27, Conference On Embedded Networked Sensor Systems, 2003. ACM.

[121] Yan Wu, Sonia Fahmy, and Ness B. Shroff. On the construction of a maximum-lifetime data gathering tree in sensor networks: Np-completeness and approximation algorithm. In *27th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 356–360, Phoenix, AZ, USA, 2008.

[122] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, and Deepak Ganesan. A wireless sensor network for structural monitoring. In *Conference on Embedded Networked Sensor Systems*, pages 13–24, Baltimore, MD, USA, 2004.

[123] Ya Xu, John Heidemann, and Deborah Estrin. Geography-informed energy conservation for ad hoc routing. In *International Conference on Mobile Computing and Networking*, pages 70–84, Rome, Italy, 2001.

[124] Fan Ye, Alvin Chen, Songwu Lu, and Lixia Zhang. A scalable solution to minimum cost forwarding in large sensor networks. In *Tenth International Conference on Computer Communications and Networks*, pages 304–309, Scottsdale, AZ, USA, 2001.

[125] Osama Younis and Sonia Fahmy. Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 629–640, Hong Kong, 2004.

[126] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, UCLA, 2001.

[127] Baoxian Zhang and Hussein T. Mouftah. Energy-aware on-demand routing protocols for wireless ad hoc networks. *Wireless Networks*, 12(4):481–494, 2006.

[128] Pei Zhang, Christopher M. Sadler, Stephen A. Lyon, and Margaret Martonosi. Hardware design experiences in zebranet. In *Conference on Embedded Networked Sensor Systems*, pages 227–238, Baltimore, MD, USA, 2004.

[129] Gergely cs and Levente Buttyn. A taxonomy of routing protocols for wireless sensor networks. *Hradstechnika*, LXII(1):32–40, 2007.