An Approach to Extending the Lifetime of Wireless Sensor Networks

Alan W. F. Boyd, Dharini Balasubramaniam, Alan Dearle, Ron Morrison
School of Computer Science, University of St Andrews, Fife KY16 9SX, Scotland
Email: {alanb, dharini, al, ron}@cs.st-and.ac.uk

***Abstract*—Wireless Sensor Networks (WSNs) are energy constrained. Every operation, particularly the use of radio, reduces the energy reserves of nodes. The aim of this work is to improve the *useful network lifetime* of a WSN, which we define as being from the time when the network is activated to the time at which it is no longer able to carry out its assigned task. Routing protocols try to preserve lifetime by being energy-aware or load-balancing. However, these approaches tend to behave non-cooperative across multiple sources. We present a heuristic known as node reliance, which indicates the degree to which nodes are relied upon in routing messages from sources to sinks. We hypothesise that by using this heuristic and avoiding nodes of high reliance, the useful network lifetime of the network can be extended.**

***Index Terms*—ad-hoc routing, networking, sensor networks, resource allocation techniques**

I. Introduction

We define a *Wireless Sensor Network* (WSN) to be a network of autonomous, battery-powered devices in which data is generated by one or more *source* nodes and routed to one or more *sink* nodes. It is envisioned that these networks may be deployed in hostile or remote environments, making redeployment of an expired network infeasible. The software installed on wireless sensor networks, therefore, should aim to preserve the lifetime of the network. This is especially true of routing protocols as radio use is several orders of magnitude more expensive than processor use [1].

Every action taken by a node uses some proportion of the node's energy reserves. The type of actions carried out by each node, and the frequency with which those actions are executed therefore affect the lifetime of a node. We consider route selection to be the most significant factor in determining a node's lifetime, because radio usage is expensive and must occur in order to effect routing between sources and sinks.

The vast majority of routing protocols in WSNs aim to extend the lifetime of a network through the intelligent selection of a path or the enforcement of some topology, typically to aid in data aggregation and compression. However, many of these protocols measure network lifetime in ways that are meaningless in real world applications. Furthermore, where paths are intelligently selected, they are typically optimal only for a single source and do not consider the requirements of other sources, nor the fact that one node may lie on the optimal path for multiple sources and be overused.

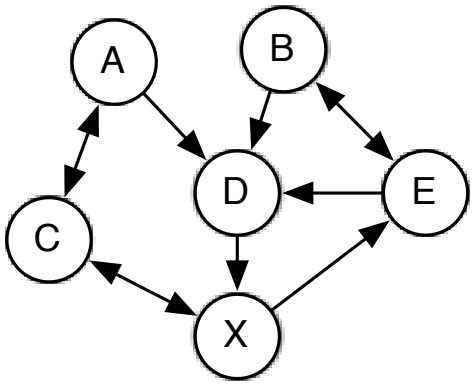
The rest of this paper is organised as follows: Section II presents the problem statement. Section III examines the literature and demonstrates the limit of current approaches. Section IV presents a new heuristic for solving the problem and provides an implementation of an algorithm that makes use of the new heuristic. Section V gives an example of how the algorithm operates under a particular network topology. Section VI discusses how the algorithm may be affected by certain properties of WSNs. Finally, Section VII explores future work.

II. Problem Statement

We consider a WSN such as that shown in Fig. 1. The task of this network is to route generated data from the source nodes (A and B) to the sink node (X). The generation of data at the sources is assumed to be random.

Routing protocols typically identify the optimal path from a source to a sink, based on some heuristic with the ultimate aim of maximising the lifetime of the network. For example, an energy-aware routing protocol might determine which path uses the smallest amount of energy in routing a message from a source to a sink. However, these route selections are rarely (if ever) co-operative. Taking the example of Fig. 1, the path ADX might be optimal for source A and the path BDX might be optimal for source B when used independently. When used simultaneously, node D may experience twice as much traffic as any other node. Being energy constrained and using its energetically expensive radio more than other nodes, node D is likely to expire quickly and the previously optimal routes are no longer available for use. Furthermore, with node D lost, source B is unable to route messages to the sink and resources

Fig. 1. An example network. Circles represent nodes and a directed edge indicates that communication can occur in the direction of the arrow.



are wasted.

A better solution would be for source A to use the path ACX, even though it may not be optimal. In this case, node D will not expire as quickly, allowing source B to remain connected for longer, reducing wastage. We identify the problem as being to extend the useful network lifetime, by which we mean the length of time the network can complete its task for. In this case the task is to route data that is generated from source nodes to sink nodes. Individual nodes are permitted to expire as long as this task can continue.

Solving this problem requires a means of identifying how important each node is in meeting the aims of the network. Any energy expenditure that takes place should then be carried out on the less important nodes in preference to the important nodes as this will allow the network to continue to function. We introduce a heuristic known as *Node Reliance* which identifies how important each node is in meeting the aims of the network. We demonstrate how this can be calculated and used in route selection to maximise the useful network lifetime.

III. Literature Review

A. Energy-Aware Routing

In explaining the difficulties of shortest path routing, Singh [2] presents five energy-aware heuristics that optimise for node and network lifetime. However, Singh does not address the difficulty of collecting energy heuristics and maintaining them. Singh's solution makes use of techniques such as "minimising variance in node power levels". However, the heuristics are hard to synchronise as they constantly fluctuate; energy must be expended to collect the heuristics. It is also hard to predict how this data changes over time due to the non-deterministic cost of radio transmissions. External factors that cannot be accounted for also make such predictions difficult, such as several sources using the same node for routing.

B. Load Balanced Routing

Dai [3] proposes a system in which a balanced routing tree is iteratively constructed, starting at the sink. Once complete, workload experienced by each node from routing should be balanced. However, this solution assumes that each node shares information about how much data it generates, which may not be known. Lee [4] presents a solution in which routes are formed on demand from sources to sinks, based on the path of minimum workload. As intermediate nodes route data from sources to sinks, they periodically attach their current workload. If the sink considers one node to be overloaded, it can cause path discovery to begin again from the source. If a path of lower workload exists, then it may be selected. Lee appears to assume that a path of lower routing load is always available, which may not be true. Tran's congestion adaptive routing [5] allows nodes to bypass a downstream neighbour that is heavily loaded by forming a bypass around that node. A proportion of all data is then sent down the bypass rather than the original path. This protocol has the advantage that global data is not required. However, as with other protocols, this solution does not anticipate how a downstream node may be used in the future.

C. Flow Control

Chang [6], [7] examines how energy aware routing might be better carried out by modelling the network as a network flow problem, with edges representing the capacity of links between each pair of nodes. Lin [8] demonstrates a static routing system which uses traffic patterns and energy replenishment statistics, rather than instantaneous energy heuristics of nodes. The authors claim that this solution "outperforms leading dynamic routing algorithms in the literature, and is close to the optimal solution when the energy claimed by each packet is relatively small compared to the battery capacity". However, it may not even be feasible for energy scavenging to take place, let alone have statistics on the process. Kalpakis [9] allows the use of data aggregation from multiple sources to reduce the amount of communication required in a network flow diagram. Once a suitable network flow has been established, a schedule can be produced. The schedule dictates how every future packet should be routed. The author's measurement of network lifetime (time until the first node expires) would not seem to reflect any particular goal of the network and so may not be appropriate.

IV. Node Reliance

A. Assumptions

- The network toplogy is initially unknown.
- At least one forward and one reverse path connect the sources and the sinks.
- Each node has a unique identifier.
- Each node can determine whether it acts as a source and/or a sink or neither.

The random nature of radio transmission ranges makes it unreasonable to statically configure a moderately sized network such that the connectivity can be exactly predefined. Environmental conditions mean that radio transmission ranges fluctuate over time, causing new links to form and old links to vanish. Dense networks have a larger number of potential links and so are more prone to this randomness in topology. Furthermore, it seems unreasonable to expect a large number of preconfigured nodes to be placed by hand in exact locations, particularly when so many applications of WSNs involve the placement of nodes in hostile or inaccessible locations.

Without a forward path, routing is impossible from sources to the sinks and the network is incapable of operating. Without a reverse path, no reinforcement is possible and the only mechanism for reliably routing messages from the sources to the sinks is flooding.

Unique identifiers can be based on the MAC address of the radio interface. Alternatively, they may be based on a pseudo-random number generator on each node, but the probability of collision must be negligible.

Source nodes and sink nodes have different behaviour to regular nodes, therefore it seems reasonable that any node can determine what its role is in the network.

B. Overview

We define:

- the *relative reliance* of a node B to a node A as being an indication of how important node B is to node A in routing messages to the sink; and
- the *absolute reliance* of a node to be an indication of how important that node is in routing messages to the sink.

A *simple path* is a sequence of nodes in which no node is repeated. The measure of a source A's relative reliance on a node B is the proportion of simple paths from A to any sink that B lies on. The maximum relative reliance of 1 indicates that a node is on all simple paths between a source and the sinks and is therefore a bottleneck for that source. The absolute reliance of a node is the average relative reliance for that node across all sources.

To determine which paths are used to route messages from sources to sinks, we use a modification of Chang's lexicographic ordering [7]. A path P is said to have a higher *lexicographic order* than a path Q if the reliance of the highest reliance node in P is greater than the reliance of the highest reliance node in Q. If the reliance values are the same, then the second highest reliance node from each path is compared, and so on.

A source selects the path to the sink with the lowest lexicographic order. The nodes on the selected path will cause the least disruption should they expire. This moves the focus away from keeping workload balanced across all nodes and means potentially flawed techniques involving comparing only energy levels or predicting future data generation are avoided.

Table I shows the absolute reliance and relative reliance of each node at each of the sources A and B. Node D lies on all paths from source B to the sink and on 50% of simple paths from source A to the sink. It is therefore given an absolute reliance value of 0.75, which is higher than any other non-sink node. In determining routes, sources A and B should ideally select paths with the lowest lexicographic order. In this case, ACX and BDX.

C. Algorithm

We present an algorithm capable of routing messages from sources to sinks along paths of lowest lexicographic order. In describing our algorithm, we define a *subpath* of a path P to be any valid path Q with the same origin and destination as P that can be produced by removing nodes from P. For example, LMN is a subpath of LEMON.

	Relative Reliance		
Node	Source A	Source B	Absolute Reliance
A	1.00	0.00	0.50
B	0.00	1.00	0.50
C	0.50	0.00	0.25
D	0.50	1.00	0.75
E	0.00	0.50	0.25
X	1.00	1.00	1.00

TABLE I
ABSOLUTE AND RELATIVE RELIANCES FOR EACH NODE FROM FIG. 1

1) Overview:

- The first phase is the path discovery phase. Messages are flooded through the network from sources to sinks. Each message keeps track of the nodes it has travelled through. If a node has already forwarded a message containing a subpath of a received path P, then path P is stored and not forwarded. This leaves small fragments of network topology data scattered throughout the network.
- The second phase is the path response phase and behaves similarly to the path discovery phase, except that messages are flooded from sinks to sources and each message also contains paths from the path discovery phase which arrived at the sinks.
- During the third (data) phase, data is generated at sources and routed to sinks. En route, any stored topology data may be attached to messages and collected at the sink nodes. In this manner, the sinks iteratively learn all simple paths between sources and sinks.

2) Messages: Three types of message are exchanged:

Route messages are used in the path discovery phase. They contain the sequence of nodes through which that particular message instance has travelled.

RouteReply messages are used in the path response phase. They are sent in response to a *Route* message arriving at a sink. They contain the sequence of nodes through which that particular message instance has travelled and the *adjacencyMatrix* (described below) of the sink at which the *Route* message arrived.

Data messages are exchanged during the data phase of the algorithm. They are used to send generated data from a source to a sink along a predefined route. Each messages contains;

- the route to use to send the message from the source to the sink;
- the generated data from the source;
- an *adjacencyMatrix* field which stores topology data that is collected as the message is routed from the source to the sink and;
- an *updateNeeded* field which is used to request that the sink sends the source an updated *adjacencyMatrix*.

3) Data Structures: Each node maintains a *paths* table consisting *Route* messages that have been forwarded.

Each node also maintains an *adjacencyMatrix*, which represents fragments of network topology and any known node roles. At the sinks, this matrix is iteratively grown as more topology data is gathered. Eventually it will represent all simple paths between sources and sinks, and it can be used to calculate the proportion of simple paths between any pair of nodes that a node lies on.

Sources also maintain a *usedNodes* array, which lists nodes that *Data* messages have been sent through by that source.

4) Description: Below we show the pseudo code for the first two phases of the algorithm. The function id() returns the unique identifier of a node and role() returns the role of the

node, represented as an integer. The same code runs on each node.

```
1  On_Startup() {
2     if (role() == source)
3        sendRouteMsg(id())
4  }
5
6  On_Receive_RouteMsg(Node[] path) {
7     if (path contains id())
8        return
9
10    Node[] newPath = path.add(id())
11
12    if (role() != sink
13    && !paths.containsSubpathOf(newPath)) {
14       if (role() == source)
15          adjacencyMatrix.add(newPath, source)
16
17       paths.add(newPath)
18       broadcastRouteMsg(newPath)
19       return
20    }
21
22    adjacencyMatrix.add(newPath, source)
23    if (role() == sink)
24       broadcastRouteReplyMsg(id(),
25       adjacencyMatrix)
26 }
27
28 On_Receive_RouteReplyMsg(Node[] path,
29 AdjacencyMatrix data) {
30    if (path contains id())
31       return
32
33    Node[] newPath = path.add(id())
34
35    if (role() == source || role() == sink) {
36       adjacencyMatrix.add(newPath, sink)
37       adjacencyMatrix.merge(data)
38       if (role() == sink)
39          return
40       unicastDataMsg(adjacencyMatrix.pathTo(
41       path.origin()), null,
42       false, adjacencyMatrix)
43    }
44    if (paths.containsSubpathOf(newPath)
45       adjacencyMatrix.add(newPath, sink)
46
47    paths.add(newPath)
48   broadcastRouteReplyMsg(newPath, data)
49 }
```

Route messages are flooded from the source nodes. Nodes discard incoming messages if the path contains their ID (lines 7-8). A path P is stored in a node's *adjacencyMatrix* if the node has forwarded a message containing a subpath of P, or if the node is a sink (line 22) or if a subpath of P has not been forwarded *and* the node is a source (lines 14-15). If none of these conditions are met, P' is formed by adding the node's ID and P' is then stored in the *paths* table and rebroadcast (lines 10, 17, 18). Whenever a sink receives a *Route* message, it broadcasts a *RouteReply* message. The *RouteReply* message operates in a similar manner to the *Route* message. The contents of *RouteReply* messages are always stored by sources and sinks (lines 36-37) and nodes that have already forwarded a message containing a subpath of the path in the incoming *RouteReply* message (lines 44-45). Sources respond to *RouteReply* messages by sending a *Data* message back to the origin of the *RouteReply* (lines 40-42).

The remaining code is used for the third phase of the algorithm.

```
50 On_DataGenerated(int data) {
51    if (adjacencyMatrix.numPathsToSinks == 0)
52       return
53
54    Matrix unused =
55       adjacencyMatrix.removeNodes(usedNodes)
56
57    if (unused.numPathsToSink() == 0)
58       useMinReliancePath(data)
59    elsif (unused.numPathsToSink == 1)
60       useRandomPath(data, unused, true)
61    else
62       useRandomPath(data, unused, false)
63 }
64
65 On_useMinReliancePath(int data) {
66    Node[] route =
67       adjacencyMatrix.getPathMinReliance()
68
69    unicastDataMsg(route, data, false, null)
70 }
71
72 On_useRandomPath(int data, AdjacencyMatrix
73 unused, bool update) {
74    Node[] route =
75       unused.getRandomPath()
76
77    unicastDataMsg(route, data, update, null)
78    usedNodes.add(route)
79 }
80
81 On_ReceiveDataMsg(Node[] route, int data,
82 bool updatedNeeded, AdjacencyMatrix matrix) {
83    matrix.add(adjacencyMatrix)
84
85    if (role() == source || role() == sink)
86       adjacencyMatrix.merge(matrix)
87   else
88      adjacencyMatrix.erase()
89    if (role() == sink && updateNeeded)
90       unicastDataMsg(adjacencyMatrix.pathTo(
91       path.getOrigin()), null,
92       false, adjacencyMatrix)
93    elsif (role() != sink)
94       unicastDataMsg(route, data,
95       updateNeeded, matrix)
96 }
```

Data messages are sent from sources to sinks when data is generated. An initial *Data* message is also sent when a source receives a *RouteReply*. Sources select a random path containing nodes that they have not used for *Data* messages before. If they have exactly one such path, they request an update from the sink (line 60), if they have no such paths they select the path of minimum node reliance (line 58) as considered by lexicographic ordering. *Data* messages are unicast along the predefined route. Nodes not on that route do not receive the message. At each hop, nodes copy their

adjacencyMatrices in to the message (line 83). If a node is not a source or sink, it then deletes its *adjacencyMatrix* (line 88). Sinks send back a *Data* message containing their *adjacencyMatrix* if the *updateNeeded* flag is set (lines 89-92).

V. Example

This example considers the network represented by Fig. 1. For the purposes of this example in demonstrating the transactions between nodes we assume that communication is perfect and always takes exactly 1 unit of time. Table II shows the sequence of actions that takes place at each time unit during the path discovery and path response phases.

When execution of the algorithm begins, *Route* messages are broadcast by sources A and B containing their node IDs and roles. As the messages are received at each node Z, the node checks the path, P. If it finds id(Z) then the path is not simple and the message is therefore discarded. This happens in time period 2 when node A receives the path "AC" from node C. A node Z then forms a new path P' by adding id(Z) to P. It searches its *paths* table for a subpath of P'.

- If a subpath is found, the incoming path is stored in the node's *adjacencyMatrix* and it is not forwarded.
- If a subpath is not found, the incoming path is stored in the node's *paths* table and it is forwarded.

In time period 2, node D received the path "BE" so it forms the path "BED" by adding its id. Node D already has the path "BD" which is a subpath of "BED". Therefore, the path "BED" is stored in the *adjacencyMatrix* at D and not forwarded. When the *Route* messages are received at a sink, it responds by sending a *RouteReply* message. This operates in exactly the same way as the *Route* message but operates from sinks to sources and contains the *adjacencyMatrix* of the sink from where the message originated. As the number of nodes in the network is finite and each node can only appear on each path once, the first and second phases eventually terminate. A source can begin its second (data) phase once it receives a *RouteReply* message. Table III shows the sequence of transactions that occurs each time unit during the data phase, starting at time period 4, immediately after sources A and B receive *RouteReply* messages. Note that the first *Data* message is sent by a source in response to a *RouteReply* message, it contains null data and the *adjacencyMatrix* stored at the source. Also, node D is the only intermediate node to have any data stored in its *adjacencyMatrix*.

Fig. 2 shows the *adjacencyMatrices* matrix1, matrix2, etc., which are stored and transferred during the three phases.

In time unit 4, the sources respond with their *adjacencyMatrices*. Source B knows of only one path to the sink, so sets the *updateNeeded* flag in its *Data* message. Both A and B randomly select a path (ADX and BDX respectively). In time unit 5, these messages travel through D. We have arbitrarily indicated that ADX arrives first and node D adds its *adjacencyMatrix* to the message. D's *adjacencyMatrix* is consequently deleted. The messages are then forwarded to X, which merges the *adjacencyMatrices* with its own. In the case of the *Data* message from source B, an update was requested. Node X therefore sends a *Data* message back to B, which is routed during time periods 6 and 7.

At the end of time period 7, all simple paths between sources and sinks are known at the sink and source B, although they do not know this fact. Future transactions depend on data being randomly generated at the sources. Each source will attempt to route a message through previously unused nodes. When only one path with unused nodes remains, the outgoing *Data* message has the *updateNeeded* flag set and the sink responds by sending its *adjacencyMatrix*. When no unused nodes remain, the source sends future *Data* messages along the path of minimum reliance using the previously mentioned lexicographic ordering.

Fig. 2. Matrices exchanged during the three phases of the example.

matrix1

	C	D	X
A	1	1	0
B	0	1	0
C	0	0	1
D	0	0	1

matrix2

	B	D	E
B	0	1	1
E	1	1	0
X	0	0	1

matrix3

	A	C	D	X
A	0	1	1	0
B	0	0	1	0
C	1	0	0	1
D	0	0	0	1
X	0	1	0	0

matrix4

	B	C	D	E	X
A	0	1	1	0	0
B	0	0	1	0	0
C	0	0	0	0	1
D	0	0	0	0	1
E	1	0	0	0	0
X	0	0	0	1	0

matrix5

	A	B	C	D	E	X
A	0	0	1	1	0	0
B	0	0	0	1	1	0
C	1	0	0	0	0	1
D	0	0	0	0	0	1
E	0	1	0	1	0	0
X	0	0	1	0	1	0

VI. Discussion

Stojmenovic [10] examines the packet reception probability between pairs of nodes as a function of distance. The work questions what it means for two nodes to be "neighbours" if the link between them is largely random. In the node reliance algorithm described in Section IV, the simple paths between sources and sinks are determined. It is possible that during this stage, one or more transmissions will be lost, resulting in an inaccurate estimate of the simple paths.

This feature of wireless communications is not detrimental to our algorithm. The user may specify the number of retransmission attempts (x) made by the MAC layer. During the path discovery phase, each message is rebroadcast x times. This method reduces the problem to finding all *stable* simple paths between the sources and sinks, where the stability of a link is specified by the user in terms of the number of retransmission attempts required for the link to be used.

VII. Status and Further Work

The algorithm has been implemented in Java, using a unit disk graph (UDG) model. Although it is not realistic for a wireless physical layer [10], we have used it to determine a proof of correctness for this algorithm, specifically that all topology data is collected, simple paths can be enumerated and that node reliance values are correctly calculated. Work is currently ongoing to port the algorithm to C++ for use in the Castalia [11] simulator which claims to have the most realistic radio models for WSN simulations. Once complete, it will be possible to fully compare this algorithm against energy aware and network flow based routing protocols to determine whether the node reliance heuristic is usable and whether an appreciable saving in terms of network lifetime can be made.

Work is also ongoing to determine whether the algorithm is optimal in collecting network topology data.

T	From	To	Message	Receiver Action
1	A	C	Route(A)	AC stored in paths
1	A	D	Route(A)	AD stored in paths
1	B	D	Route(B)	BD stored in paths
1	B	E	Route(B)	BE stored in paths
2	C	X	Route(AC)	ACX stored in adjacencyMatrix (matrix1)
2	C	A	Route(AC)	Ignored (ACA not simple path)
2	D	X	Route(AD)	ADX stored in adjacencyMatrix (matrix1)
2	D	X	Route(BD)	BDX stored in adjacencyMatrix (matrix1)
2	E	D	Route(BE)	BED stored in adjacencyMatrix (matrix2) Not forwarded due to subpath BD
3	E	B	Route(BE)	Ignored (BEB not simple path)
3	X	C	RouteReply(X, matrix1)	XC stored in paths
3	X	E	RouteReply(X, matrix1)	XE stored in paths
3	C	A	RouteReply(XC, matrix1)	XCA stored in adjacencyMatrix (matrix3) and matrix1 added to matrix3
3	E	D	RouteReply(XE, matrix1)	XED stored in paths
3	E	B	RouteReply(XE, matrix1)	XEB stored in adjacencyMatrix (matrix4) and matrix1 added to matrix4
4	A	D	RouteReply(XCA, matrix1)	XCAD stored in paths
4	D	X	RouteReply(XED, matrix1)	Ignored (XEDX not simple path)
4	B	D	RouteReply(XEB, matrix1)	XEBD stored in adjacencyMatrix (matrix2). Not forwarded due to subpath XED
5	D	X	RouteReply(XCAD, matrix1)	Ignored (XCADX not simple path)

TABLE II
MESSAGE TRANSACTIONS DURING THE FIRST TWO PHASES OF THE ALGORITHM.

T	From	To	Message	Receiver Action
4	A	D	Data(ADX,null,matrix3,false)	D's adjacencyMatrix (matrix2) is merged with matrix3 and deleted.
4	B	D	Data(BDX,null,matrix4,true)	
5	D	X	Data(ADX,null,matrix5,false)	X merges its adjacencyMatix with matrix5 to give matrix5. X does not reply.
5	D	X	Data(BDX,null,matrix4,true)	X merges its adjacencyMatrix with matrix4 to give matrix5. X is asked to reply.
6	X	E	Data(XEB,null,matrix5,false)	
7	E	B	Data(XEB,null,matrix5,false)	B merges its adjacencyMatrix with matrix5 to give matrix5. B ignores the update flag.

TABLE III
MESSAGE TRANSACTIONS DURING THE THIRD PHASE OF THE ALGORITHM.

Finally, it might be possible to consider other factors in determining path selection. For example, consider two paths P and Q. If path P is ten times more energy efficient than path Q, but the highest reliance node of path P is only very slightly more than that of the highest reliance node in path Q then path P may be the better choice. Similarly the highest reliance node on path P has an infinite energy supply then it may be perfectly acceptable to use path P. Some cost function or weighting might be appropriate to balance these two measurements of path suitability.

REFERENCES

[1] L. Doherty, B. Warneke, B. Boser, and K. Pister, "Energy and performance considerations for smart dust," *International Journal of Parallel Distributed Systems and Networks*, vol. 4, no. 3, pp. 121 – 133, 2001.

[2] S. Singh, M. Woo, and C. S. Raghavendra, "Power-aware routing in mobile ad hoc networks," in *Mobile Computing and Networking*. Dallas, Texas, United States: ACM, 1998, pp. 181–190.

[3] H. Dai and R. Han, "A node-centric load balancing algorithm for wireless sensor networks," in *IEEE GLOBECOM - Wireless Communications*, vol. 1. San Francisco, USA: IEEE Communications Society, 2003, pp. 548 – 552.

[4] S.-J. Lee and M. Gerla, "Dynamic load-aware routing in ad hoc networks," in *IEEE International Conference on Communications*, vol. 10. Helsinki, Finland: IEEE, 2000, pp. 3206–3210.

[5] D. A. Tran and H. Raghavendra, "Congestion adaptive routing in mobile ad hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 11, pp. 1294–1305, 2006.

[6] J.-H. Chang and L. Tassiulas, "Energy conserving routing in wireless ad-hoc networks," in *INFOCOMM*. Tel Aviv, Israel: IEEE Computer Society, 2000, pp. 22–31.

[7] ——, "Routing for maximum system lifetime in wireless ad-hoc networks," in *37th Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, 1999.

[8] L. Lin, N. B. Shroff, and R. Srikant, "Energy-aware routing in sensor networks: A large system approach," *Ad Hoc Networks*, vol. 5, no. 6, pp. 818–831, 2007.

[9] K. Kalpakis, K. Dasgupta, and P. Namjoshi, "Maximum lifetime data gathering and aggregation in wireless sensor networks," in *IEEE International Conference on Networking*. Atlanta, Georgia, US: IEEE, 2002, pp. 685–696.

[10] I. Stojmenovic, A. Nayak, and J. 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)*, vol. 43, no. 3, pp. 101–106, 2005.

[11] H. N. Pham, D. Pediaditakis, and A. Boulis, "From simulation to real deployments in wsn and back," in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, IEEE, Ed. Espoo, Finland: IEEE Computer Society, 2007, pp. 1–6.