Abstract—Wireless networks that combine a many-to-one traffic flow with multi-hop communication suffer from a funnelling effect that inevitably overloads nodes closer to the base station. In sensor networks this is known as the energy hole problem. A routing protocol that constructs a shortest path tree exacerbates this problem by leaving some of the most critical nodes with more descendants than others and hence more relaying work to do.

In this paper we propose a novel, fully distributed tree construction algorithm for building a load-balanced tree. Our algorithm, Degree Constrained Routing (DECOR), trades off latency for load balancing. We simulated the algorithm using a sensor network as an example. The results show that a small increase in latency, of less than 10%, can be sacrificed for a larger increase in balance of up to 80%. The lifetime of a sensor network using DECOR can be extended by up to 150% compared to the next best algorithm.

Index Terms—latency, load balancing, routing, sensor networks, wireless networks

I. INTRODUCTION

Wireless networks that combine a many-to-one traffic flow with multi-hop communication suffer from an inherent load imbalance because the nodes closest to the base station or sink must relay more packets than those further away. This problem causes a bottleneck to form around the base station which can result in poor quality of service. When the network devices have limited energy reserves, for example in sensor networks, the nodes around the sink run out of energy much sooner than other nodes leaving the sink unable to communicate with the rest of the network. This is known as the energy hole problem in sensor networks [1].

While the imbalance between nodes close to and far away from the sink is inherent to the network, it can be exacerbated by the network’s routing protocol. If a routing protocol favours some nodes over others, perhaps because they offer higher quality links, then the resultant routing tree may have more routes running through some of the sink’s neighbours than others.

Even in situations when all links are the same, it is difficult to create a properly balanced routing tree such that every one of the sink’s neighbours, $A$, has one child and another, $B$, has two, then if all of the descendants of $A$ and $B$ adopt exactly the same number of children as each other, node $B$ still has twice the workload of node $A$.

Since the sink’s neighbours are the most heavily burdened nodes in the network, balancing the load among them is most important. In battery-powered networks, the lifetime of the network is limited by the lifetime of these nodes and therefore for lifetime maximisation only these nodes need be considered.

In this paper we propose a novel tree construction algorithm that creates a load-balanced tree. Our algorithm is called Degree Constrained Routing (DECOR) and is fully distributed. We show that attempting to build a routing tree of minimum depth precludes the possibility of achieving load balance. Our algorithm therefore trades off an increase in latency (tree depth) for improved balance among the most critical nodes.

II. RELATED WORK

Li and Mohapatra introduce the important corona model for discussing the energy hole problem in sensor networks [1]. The model is for a network with a single, central sink in which all nodes share the same transmission range, $r$. In that case the network can be
viewed as being a series of concentric coronas or rings with the sink occupying ring \( r_0 \) as illustrated in Fig. 1. All the nodes in ring \( r_i \) can communicate with at least one node in the next inner ring \( r_{i-1} \) and forward their packets through a node in that ring. Because of this, a shortest path tree constructed for such a network has the property that a node in ring \( r_i \) will join the tree in level \( l_i \).

Fig. 1. A circular network can be viewed as a series of concentric coronas or rings. The square in the centre is the sink. The shaded ring contains the most critical nodes that will die first, cutting off the sink from the rest of the network.

In an early work on constructing load-balanced routing trees, Hsiao et al. defined the important terms top subtree and top load-balanced tree [2]. Given a routing tree rooted at the sink, a top subtree is a subtree whose root is directly connected to the sink. A top load-balanced tree is a routing tree in which all the top subtrees have the same weight.

They also defined a balance index, \( \beta \), given in equation (1) where \( w_x \) is the weight of top subtree \( x \) and \( k \) is the number of top subtrees. The balance index provides a measure of the fairness of the spread of the load among the top subtrees and has the desirable property that it tends to one as the loads become more equal.

\[
\beta = \left( \frac{\sum_{x=1}^{k} w_x}{k \sum_{x=1}^{k} w_x^2} \right)^2
\] (1)

Hsiao et al. propose an iterative method for amending an existing tree to increase its balance. In each iteration, one node from a top subtree “defects” to another subtree if the move increases the balance. In a distributed implementation, the authors suggest that the sink must control the process and therefore gathers the necessary information from the network before ordering a specific node to move to another tree.

Huang et al. present a distributed algorithm designed for monitoring networks, in which all nodes have the same weight, that only considers the degree of a node’s neighbours, i.e. the number of children each neighbour has [3]. Periodically during tree construction every node queries its neighbours to find their distance from the sink and their degree. After each consultation they select the node with the shortest distance to the sink that has the fewest children. The resultant tree is called a minimum balanced tree (MBT) and its depth is provably minimal.

A different approach is adopted by Andreou et al. [4]. Their algorithm, energy-driven tree construction (ETC), relies on an optimal branching factor, \( BF \), defined in equation (2) where \( N \) is the total number of nodes in the network and \( d \) is the depth of the tree. The optimal branching factor is the number of children that each node would have in a completely balanced tree of minimal depth.

\[
BF = \sqrt[2]{N}
\] (2)

An initial shortest path tree is constructed and then the sink queries it to discover the number of nodes in the network and the depth of the tree. It calculates the branching factor and floods it through the network. Every node that has more children than the calculated factor attempts to switch some of its children to new parents. To achieve this, parent nodes must gather the required information from their children, including information about the potential parents of each child, and ask a child to switch to a new parent. Some switches will be rejected if, for example, the new parent already has too many children. The process of asking children to move continues until the number of children is \( \lceil BF \rceil \) or until no more switches are available.

Following the same underlying idea of controlling the number of children per parent, Chatzimilioudis et al. propose a fully distributed algorithm for constructing a load balanced tree that they call minimum hot-spot query routing tree (MHS) [5]. The principle is that all the nodes in the same ring sequentially choose their parent, selecting the one with the fewest children. Initially the choices will be poor as the nodes will have incomplete information. Therefore, the authors propose that the first nodes to choose should be those with the fewest options which are in any case unable to make refined decisions.

To ensure this, nodes start a timer after first hearing from
potential parents. The length of the timer increases with the number of potential parents and nodes only respond after the timer finishes. While the timer is running down, the nodes listen to the control packets to determine how many children each of their potential parents has already adopted. In this way when they come to make their decision they have more information available.

Chen et al. also propose a fully distributed algorithm for data gathering networks which they call the adjustable convergecast tree (ACT) [6]. The algorithm starts by constructing a simple shortest path tree. The unbalanced tree is then adjusted relying on the notion of a load-balanced factor (LBF) defined by the authors as in equation (3), where \( n_{st_{chi_j}} \) is the number of nodes in the subtree rooted at one of node \( j \)'s children.

\[
LBF = \frac{\text{Min}(n_{st_{chi_j}})}{\text{Max}(n_{st_{chi_j}})} \tag{3}
\]

The process starts at the bottom of the tree and works upwards towards the sink. Nodes gather information from their children about subtree sizes and the node’s grandchildren. The node then calculates its LBF. If the result is less than one and the difference in size between the largest and smallest of its children’s subtrees is also greater than one, it initiates rebalancing. This works by searching among the grandchildren in the largest subtree for a node that can be moved to another subtree such that the load-balanced factor is improved. If it finds one, the node informs the relevant children and the change is made. The process of searching and adjusting continues until the LBF is maximised. The grandparent then informs its own parent that it has completed its rebalancing and the process continues at each level of the tree ending with the root.

All these solutions aim to maximise the balance while retaining a shortest path tree. We show in the next section that perfect balance cannot be achieved with such a tree.

III. System Model, Assumptions and Preliminaries

A network can be modelled as an undirected graph \( G(V, E) \) where \( V \) is the set of nodes and \( E \) is the set of wireless links between them. In this paper we follow the assumptions used by Li and Mohapatra in their corona model [1]. That is, we assume a circular network of radius \( R \) with a single, central sink \( v_0 \in V \). The sink has a much larger battery capacity than the other motes and can be viewed as having infinite energy. Each node, \( v_i \in V(i > 0) \), has the same initial energy, \( e \), and the same transmission range \( r \). The network lifetime is divided into discrete rounds and every node generates one new data packet per round. The nodes are uniformly and randomly distributed with uniform density, \( \rho \), throughout the network area. There is a MAC (Medium Access Control) layer that minimises collisions and interference.

The network can be viewed as a series of concentric coronas or rings each of width \( r \) as described above in section II. Macedo analysed the nature of a shortest path routing tree constructed for this kind of network [7]. He found that the average number of children per parent in level \( i \) of the tree, \( C(i) \), is equal to:

\[
C(i) = \frac{2i + 1}{2i - 1} \tag{4}
\]

This shows that it is impossible to achieve perfect balance for a shortest path tree. Consider, for example, the case for the third level of the tree, \( i = 3 \). The average number of children adopted by each node in level three is 1.4. Since nodes obviously cannot adopt a part of a node, this creates imbalance with some nodes adopting two children and some adopting only one.

Our algorithm, DECOR, recognises this inevitability and therefore accepts that to achieve better balance the tree cannot be of minimum depth. It therefore trades off latency for increased balance. As a preliminary to discussing the algorithm in more detail, we note that the ratio between the number of nodes in ring \( r_1 \) and the number of nodes in a given ring \( r_i \) is \( 1 : \theta(i) \), where \( \theta(i) = 2i - 1 \). This follows directly from the result shown in equation (4). This allows us to make the following definition:

**Definition:** For the corona model described above, base \( n \) rings are those rings of the network, \( i \), where \( \theta(i - 1) < n^x \leq \theta(i) \) \( \{x \in \mathbb{Z}^+ \} \). That is, a base \( n \) ring is the most central ring containing at least \( n^x \) times as many nodes as are in the inner-most ring. When \( n = 2 \), for example, the rings \( i = \{2, 3, 5, 9, 17, 33 \ldots \} \) are the base 2 rings.

IV. DECOR: Degree Constrained Routing

In this section we present our novel tree construction algorithm. The aim of DECOR is to maximise balance by minimising the variance in top subtree sizes. The variance is caused by some nodes in a level of the routing tree adopting more children than others in the same level. Wherever this happens in the tree, imbalance is caused in the top level because one node in that level ends up with more descendants than another and therefore has more work to do. Ideally, the nodes in each level would evenly
divide the children in the next level amongst themselves and each would adopt exactly the average number of children per parent for their ring, \( C(i) \) as defined in equation (4). However, apart from the first level, \( C(i) \) is always a fraction. Since nodes obviously cannot adopt a fraction of a child, without any control some nodes would adopt more than the average and some less leading to imbalance.

DECOR places a limit on the number of children a node can adopt to prevent imbalance. The starting point is to place a simple limit of \( \lfloor C(i) \rfloor \) which would prevent some nodes adopting more nodes than others. However, it also prevents lots of nodes joining the tree because \( \lfloor C(i) \rfloor = 1 \) for \( i > 1 \) while the number of nodes contained in each ring increases.

To mitigate much of this effect, DECOR uses the concept of base \( n \) rings defined above. If the number of nodes per parent were limited to just one for all nodes, then the number of nodes connected to the tree at every level would be the same as the number of nodes in ring \( r_1 \). In that case the base \( n \) rings would be the first rings in which there are at least \( n^x \) times as many nodes as there are connected to the tree from the the preceding ring. This would allow for the maximum number of children per parent to be increased because now the floor of the average number of children per parent would be greater than one.

In DECOR we use base 2 rings because this allows the maximum number of children per parent to be increased earlier and more often during tree construction. Therefore, the maximum number of children that a node in level \( i \) may adopt, \( M(i) \), is:

\[
M(i) = \begin{cases} 
\infty & \text{if } i = 0 \text{ i.e. the sink} \\
2 & \text{if ring } i+1 \text{ is a base 2 ring} \\
1 & \text{otherwise}
\end{cases}
\]

To illustrate, consider a network with 12 nodes in the first ring. Table I shows the number of nodes that would be in each ring, the maximum number of children allowed per parent and the number of connected and unconnected nodes. If the maximum number of children that a node may adopt was kept constantly to just one, then only 12 nodes in ring four could be adopted leaving 72 unable to connect to the tree. On the other hand, if all nodes (except the sink) are allowed to adopt up to two children each, then the 48 nodes in ring three that were able to connect to the tree would share the 84 nodes in ring four between themselves, leaving some with one child and others with two. DECOR ensures that all nodes in the same level of the tree adopt the same number of children by placing a limit on the maximum number that each node may adopt. By varying the limit according to a node’s level in the tree the number of nodes that are unable to attach to the tree is much lower than it would be if the limit was a constant one node per parent.

<table>
<thead>
<tr>
<th>Ring No.</th>
<th>No. of nodes</th>
<th>Max. No. of children / node</th>
<th>No. connected (unconnected)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>( \infty )</td>
<td>1 (0)</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>2</td>
<td>12 (0)</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>2</td>
<td>24 (12)</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>1</td>
<td>48 (12)</td>
</tr>
<tr>
<td>4</td>
<td>84</td>
<td>2</td>
<td>48 (36)</td>
</tr>
<tr>
<td>5</td>
<td>108</td>
<td>1</td>
<td>96 (12)</td>
</tr>
<tr>
<td>6</td>
<td>132</td>
<td>1</td>
<td>96 (36)</td>
</tr>
</tbody>
</table>

Nevertheless, as the results in Table I show, many nodes cannot connect to the tree. Preliminary simulations showed that the number unable to connect to the tree was much lower than the theory suggests. This is because the routing tree was able to turn back on itself and attach to previously unconnected nodes as illustrated in Fig. 2. This results in almost all the nodes connecting to the tree. However, latency is greatly increased by this process because nodes that are physically only a small number of hops from the sink connect to the tree many topological hops further away.

Fig. 2. A top subtree after phase one of the DECOR algorithm may contain many turns that increase latency.

DECOR therefore has a second phase which guar-
antees full connectivity and reduces the extra latency. Whereas in the first phase there was a limit to the maximum number of children a node could adopt, in the second phase this limit is removed. Removing it allows any node that has a path to the sink to join the tree, guaranteeing full connectivity. However, in order to retain close to the balance achieved in the first phase, nodes are not allowed to switch subtrees. That is, any node that was in the routing tree after phase one must remain a descendant of the same level one node during phase two. This is achieved by assigning every node a subtree number during phase one which they retain in phase two. Nodes may only adopt a child in phase two that either has the same subtree number as itself or has no number at all (meaning that the adopted node did not connect during phase one).

Removing the limit on the number of children that may be adopted also removes most of the turns by creating “shortcuts” between nodes in the same subtree. If a node within a subtree can connect to another in the same subtree that is topologically closer to the sink, then in phase two it will do so. Fig. 3 shows the same subtree as Fig. 2 after phase two illustrating the replacing of turns with shortcuts. Although this phase creates imbalance between nodes in most levels of the tree, it has little effect on the balance among the most critical nodes because the subtree sizes remain roughly the same size, only altering because of the small number of previously unconnected nodes.

A. Phase One

Both phases of the DECOR algorithm build a routing tree iteratively, starting at the sink and growing outwards. Each round of tree construction starts with the new leaf nodes, i.e. the nodes that were added to the tree in the previous round. In the first round, only the sink is a leaf node. The leaf nodes broadcast an advert, \( \text{ADV} \), packet containing their ID, the number of hops they are from the sink, \( h_c \), and their subtree number, \( s_i \). During the first round the sink generates and assigns a unique subtree number to each of its children, the level one nodes. Thereafter, nodes are assigned the same subtree number as their parent. In this way every node in the tree can be identified as a member of a specific top subtree. After broadcasting its \( \text{ADV} \) packet, each leaf node waits for a time, \( t_{\text{req}} \), to gather responses from non-leaf nodes.

Non-leaf nodes that receive an \( \text{ADV} \) packet become active for that round and wait for a time \( t_{\text{adv}} \). The details of all \( \text{ADV} \) packets heard during that time are recorded and used to select the best parent. Nodes utilise RSSI (Received Signal Strength Indication) information to determine the potential parent that is the furthest from them but still inside reliable communication range. By choosing parents that are far away, the latency and average hop count are reduced and the corona model is better matched. Having selected the best parent, non-leaf nodes transmit a request, \( \text{REQ} \), packet to the chosen parent containing their ID and the number of nodes that could serve as their parent.

After waiting \( t_{\text{req}} \), leaf nodes choose which child to adopt. The primary criterion is the number of potential parents. Leaf nodes should choose the child with the fewest options to reduce the chance that a non-leaf node is prevented from finding a parent when one could have been available. Distance, as measured using RSSI, can be used as a secondary criterion. Leaf nodes broadcast an adopt, \( \text{ADPT} \), packet containing their own ID, \( C_{ID} \), and a variable, \( \text{space} \), indicating how many more children they may adopt.

Non-leaf nodes that receive an \( \text{ADPT} \) packet can judge whether they have been adopted or rejected. If the leaf node has no space for more children, then the rejected nodes delete the leaf node from their list of potential parents. Rejected nodes select the next best parent (which may be the same node that has just rejected them) and transmit another \( \text{REQ} \) packet.

This process, of non-leaf nodes transmitting \( \text{REQ} \) packets and leaf nodes responding with \( \text{ADPT} \) packets, continues until non-leaf nodes have all been either

![Fig. 3. The top subtree from Fig. 2 after phase two. The turns have been removed through “shortcuts”, decreasing latency.](image)
adopted or rejected by all potential parents. Since leaf nodes will have to receive a maximum of two groups of \textsc{req} packets, a round time of $3t_{adv}$ is sufficient to ensure that the process is not cut short. At the end of each round the leaf nodes instruct their children to start the next round.

B. Phase Two

The process of the second phase of the DECOR algorithm is the same as the first phase; namely that leaf motes broadcast \textsc{adv} packets, wait for \textsc{req} replies and respond with \textsc{adpt} packets. However, in phase two, non-leaf nodes only record the details of \textsc{adv} packets if either the subtree number, $s_i$, contained in the packet matches the non-leaf node’s own subtree number as assigned in phase one or else the non-leaf node has no subtree number. The other difference between phases one and two is that leaf nodes are not limited in the number of children they may adopt. Therefore, in phase two, they respond with an \textsc{adpt} packet to every \textsc{req} packet.

Algorithms 1 and 2 give the pseudocode for a leaf node and non-leaf node respectively for one round of the DECOR algorithm.

**Algorithm 1** DECOR pseudocode for leaf nodes

\textbf{Phase One}

- Broadcast $\textsc{adv}(\text{my.id},hc,s_i)$
- while $|\text{children}| < \max$ do
  - for all $\textsc{req}(\text{id})$ do
    - $\text{childList}.\text{add}(\textsc{req}.\text{id})$
  - end for
  - $C_{ID} = \text{childList}.\text{select}()$
  - $\text{children}.\text{add}(C_{ID})$
  - $\text{space} = \max - |\text{children}|$
- Broadcast $\textsc{adpt}(\text{my.id},C_{ID},\text{space})$
- end while

\textbf{Phase Two}

- Broadcast $\textsc{adv}(\text{my.id},hc,s_i)$
- for all $\textsc{req}(\text{id})$ do
  - $\text{children}.\text{add}(\text{id})$
- Broadcast $\textsc{adpt}(\text{my.id},\text{id},\infty)$
- end for

**Algorithm 2** DECOR pseudocode for non-leaf nodes

\textbf{Phase One}

- for all $\textsc{adv}(P_{ID},hc,s_i)$ do
  - $\text{parentList}.\text{add}(\text{adv})$
- end for
- while parent == null do
  - $P_{ID} = \text{parentList}.\text{select}()$
  - Send $\textsc{req}(\text{my.id}) \rightarrow P_{ID}$
  - for all $\textsc{adpt}(ID,C_{ID},\text{space})$ do
    - if $C_{ID}==\text{my.id}$ then
      - parent = $\text{ADPT}.\text{id}$
      - my.$hc = \text{parent}.hc + 1$
      - my.$s_i = \text{parent}.s_i$
    - else if $\text{space} == 0$ then
      - parentList.delete(ADPT.$\text{id}$)
    - end if
  - end for
- end while

\textbf{Phase Two}

- for all $\textsc{adv}(P_{ID},hc,s_i)$ do
  - if $\text{ADV}.s_i==\text{my}.s_i || \text{my}.s_i==\infty$ then
    - parentList.add(ADV)
  - end if
- end for
- $P_{ID} = \text{parentList}.\text{select}()$
- Send $\textsc{req}(\text{my.id}) \rightarrow P_{ID}$
- parent = $\text{id}$
- my.$hc = \text{parent}.hc + 1$

V. SIMULATION RESULTS

A. Setup and Metrics

In order to evaluate the performance of our proposed algorithm we ran a number of simulations of a sensor network used for monitoring. We compared our algorithm, DECOR, to MBT [3], ETC [4], MHS [5] and ACT [6].

Our simulations are of circular networks with a single, central sink in which nodes are randomly and uniformly distributed such that the density is approximately constant across the network area. All nodes are equipped with CC1000 [8] transceivers operating at their maximum output power. Thus the data rate is 19.2kbps, the power consumed while transmitting and receiving is 80.1mW and 22.2mW respectively.

For ease of analysis we adopt the unit-disk transmission model and assume that the maximum transmission range of every node is 10m. We only consider the energy consumed as a result of communication as this is the dominant energy consumer.

All packets are 50 bytes and nodes start with 50J of energy. It should be noted that these values for packet size and starting energy are arbitrary and chosen for
simplicity. The relative performance of the algorithms are unaffected by changing these values. The results presented in section V-B are the average of 25 runs.

During the first set of simulations we maintained a constant deployment density of 0.1 nodes per square metre and varied the radius of the network. For the second set, the radius was fixed at 70m and the deployment density was varied.

We are interested in the following metrics: balance, network lifetime and average latency. For balance we use the measure defined by Hsiao et al. given in equation (1) [2]. As all nodes generate data packets at the same rate they all have the same weight and so we take the number of nodes in a top-subtree to be its weight. We are also interested in network lifetime for our example of a battery powered sensor network. Although balance gives a measure of the similarity of the burdens on each critical node it doesn’t accurately predict lifetime. The limiting factor for lifetime is the load on the most heavily burdened node and this is not measured by the balance metric. Therefore we measure network lifetime as the time until the first node depletes its battery. Latency is measured in terms of the number of hops between a node and the sink.

B. Results

Figs. 4(a)-4(c) show the results for balance, network lifetime and latency. The balance in all cases falls as the radius is increased but DECOR outperforms the alternative algorithms. Not only is its balance always higher but it degrades slower, decreasing by just under 15% over the range of radii. Compared to the next best performing algorithm, MHS, balance is improved by between 14% and 80%.

The results for lifetime follow a similar pattern with lifetime decreasing in all cases as the radius increases. Again DECOR outperforms the others with MHS being the closest alternative. DECOR offers an improved lifetime of between 66% and 130%.

The trade-off that DECOR accepts for improved balance and lifetime is increased latency. All the other algorithms create a shortest path tree and therefore all have the same average latency. The divergence between DECOR and the others increases linearly with radius ranging from just under 6% to 7.75%. It should be noted, though, that even when the difference is largest it still amounts to less than one hop’s latency.

Figs. 5(a)-5(c) show the results for balance, lifetime and latency for different deployment densities. The results for balance show that different approaches produce quite different results. For most of the algorithms, the balance falls as the density increases whereas it remains largely constant for MHS and actually improves under DECOR. Indeed, at the lowest density DECOR produces a balance just over 2% lower than MHS but at the highest density its balance is more than 50% higher.

The lifetime results show that while all the other algorithms have shorter lifetime when there are more nodes in the network, DECOR takes advantage of the extra nodes to increase lifetime, resulting, in the best case, with a 150% improvement.

Again the trade-off of latency is evident but the results
show that any increase in latency is independent of the deployment density.

(a) Balance

(b) Lifetime

(c) Latency

![Graphs comparing DECOR to proposed algorithms for varying density](image)

VI. DISCUSSION

The DECOR algorithm is based on the recognition that a shortest path tree is incompatible with a well balanced one. It therefore introduces the trade off between increased depth (and hence latency) and balance. DECOR also incorporates some global knowledge into the tree construction algorithm whilst remaining completely distributed. Additionally, unlike MBT, ETC and ACT, DECOR does not require numerous iterations to produce a balanced tree.

Of the algorithms simulated only DECOR and MHS do not rely on first constructing a shortest path tree and then attempting to make it more balanced. It is probably no coincidence, then, that these two algorithms performed the best in our simulations.

Our results show that the trade off of latency for balance (and with balance, increased lifetime) can often be extremely advantageous. In the worst case the increase in latency was less than 10% which, for the network sizes we examined, amounted to less than a single hop increase on average. Except in networks required for emergency situations, this small increase is most likely acceptable. However, the increase in latency is proportional to the radius and in very large networks may become unacceptable for some non-emergency applications.

VII. CONCLUSION

Wireless networks that combine a many-to-one traffic flow with multi-hop communication suffer from an inherent load imbalance. The problem can be exacerbated by a routing protocol that places more load on some of the base station’s neighbours than on others. In this paper we have proposed a novel, fully distributed algorithm for constructing a load-balanced routing tree which we call Degree Constrained Routing (DECOR). DECOR offers greatly improved balance and network lifetime in exchange for a small increase in latency. By noting that a tree of minimum depth is incompatible with high balance, DECOR controls the number of children per parent during the construction of the tree. Our simulations of a sensor network using DECOR show that an increase in latency of less than 10% provides an 80% increase in balance and up 150% extra lifetime.

In this initial analysis we have made two important simplifying assumptions. The first is that the density of the nodes is uniform across the network and the second is that all nodes generate the same traffic. These serve to simplify the traffic flow and make it regular and predictable. In reality neither assumption is likely to hold perfectly in all situations and we must investigate how DECOR performs when the traffic flow is less predictable.

Our work so far shows that a new trade-off can be made between latency and load balancing. The initial simulation results, albeit in favourable scenarios, indicates that a large benefit in terms of balance can be achieved with only a small sacrifice in latency.
REFERENCES


