Minimizing Power Consumption in Wireless Sensor Networks

John R Hott

Advanced Analysis of Algorithms

Spring 2006

Abstract

Power consumption among sensors has always and will continue to be an issue among developers and deployers of such devices. Wireless sensor networks are even more power draining because they utilize a wireless radio communications device which requires more power than any other items on the sensor. This paper discusses some of the previous schemes to reduce power consumption through planning or setting up topologies of the wireless sensors. The main question that this paper seeks to answer is how to set up randomly distributed nodes into a network that is useable for current routing techniques such as the approximate shortest path utilizing a method to account for energy consumption at each hop. The paper presents algorithms for setting up this system, one that works and one that appears better but has unwanted side effects. Adapting a random network to a more structured scheme is difficult, but can be done. This paper gives some samples of the algorithm using a simulation of 10 nodes, looks at the complexity of the algorithm, and concludes that even though this algorithm is not optimal or efficient, it is a good step towards utilizing these methods over randomized networks.

1 Preintro

2 Introduction

With the availability and usability of wireless sensor networks increasing, as well as many wireless devices being used that do not have unlimited power supplies, we must look at energy costs of these sensors and nodes. Most of the communication costs come from the cost of communication between nodes. As an example, from the AT90S8535 wireless sensor [4], the processor takes $5mA$ while active versus the radio, which takes $12mA$ to send and $4.5mA$ to receive. Even when inactive, the radio takes $5\mu A$ in each mode while the processor only takes $1\mu A$. Therefore, we can see that attention needs to be paid to the amount and strength that our sensors must send through the radio.

Also, some networks of sensors may be deployed in areas where administrators do not want to go or are unable to go, and therefore cannot place the sensors in a predefined grid to which their algorithm can automatically run. These environments include battlefield applications (mentioned in [1]), nature applications (habitat monitoring [5]), among others.

This paper will go into the background of some topologies, look into related works, and then focus on the algorithm proposed and conclude.

3 Background

3.1 Powering the Radio

Let us first look at the costs of powering the radio and see exactly what issues we must look at. Consider a very simple radio model, as described in [2], where the base energy is $E_{elec} = 50nJ/bit$ to transmit or receive one bit and the energy to amplify for a receiver to hear $\epsilon_{amp} = 100pJ/bit/m^2$. With this model, to send a k-bit message a distance of d meters, it would take

$$\begin{aligned} E_{Tx}(k,d) &= E_{Tx-elec}(k) + E_{Tx-amp}(k,d) \\ &= E_{elec}k + \epsilon_{amp} \cdot 10^{-3} k d^2 \\ &= 50k + 100 \cdot 10^{-3} k d^2 \end{aligned}$$

nanoJoules of energy. Similarly, to receive the same k-bit message, it would take

$$\begin{aligned} E_{Rx}(k) &= E_{Rx-elec}(k) \\ &= E_{elec}k \\ &= 50k \end{aligned}$$

nonoJoules of energy. Therefore, each k-bit message would consume at least

$$E = h(100k + 0.1kd^2)nJ,$$

if all the nodes are equidistant. However, if our message is picked up by all the neighbors of each node along the path, we are really consuming a total of

$$E = h(50k + 0.1kd^2 + 50n_i k)nJ,$$

where n_i is the number of neighbors of node i with a distance of less than d from i. This leaves us with a few options to reduce the energy consumed.

Firstly, we could reduce the size of the message, k, but that is only feasible to a certain extent because it would require the sensors to have the processing power to compress data. Even if we did allow compressing of some data, the power used to compress the data would have to be balanced out with the power to send or their combined energy uses would outweigh the cost of sending the uncompressed data.

Another option is to reduce the number of neighbors along the way. Salhieh, in [7], addresses this by sending all communications toward the edges of the network first, then towards the destination. They show improvement on energy costs, but it requires longer travels through the network to reach the destination and it wears out the nodes along the edges of the network.

The option addressed in this paper is to reduce the distance d each node must send the data, and therefore reduce the cost to amplify each bit. This can be very significant if we are sending to nodes quite far away. We will look at how to reduce this distance by considering different topologies and choosing one that is more promising.

3.2 Topologies

Wireless Sensor Networks have typically been arranged into a few different topologies. Each of them has advantages and disadvantages, which we will discuss in detail.

The most naive and simple way to set up a network is to have every node communicate directly to the base station, as seen in Figure 1. This setup provides the worst power efficiency, but it allows the base station to stay up to date consistently. Having each node communicate directly to the base station requires a lot of power to amplify the radio signal to reach the base station with enough intensity for the base to read in the message. It also introduces a need to control or avoid congestion between the nodes in case they all decide to communicate at once. It provides much better updating of the base station and much less communication time for queries and responses from the base station. Early wireless networks were this type of topology [6].

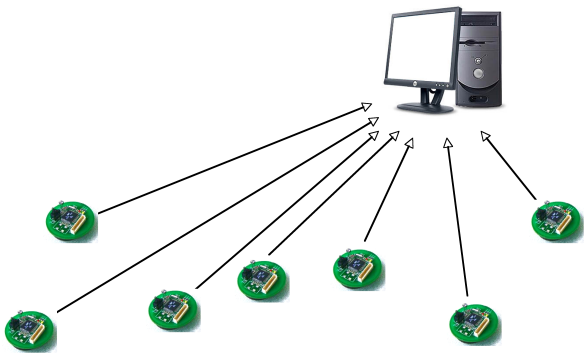


Figure 1: Naive simple approach: let all nodes connect directly to the base station.

We can extend the simple approach into an Hierarchical approach, seen in Figure 2. This approach is similar to the simple approach, except in this approach the nodes will talk to the node that is closer to the base station than itself. This approach leads to a tree-like structure with the base station as the root node. It provides many advantages and disadvantages over the simple approach. In this scheme, the communication costs for leaf nodes are drastically reduced because they do not have to amplify their signal to reach the base station, but just a neighbor. It also reduces the cost and probability of congestion in the network. For disadvantages, this system requires nodes to be able to forward data and puts more burden on nodes in the path, especially close to the base station, leaving those nodes to quickly burn out. Also, it increases communication – query and data – costs because we now have to wait for queries or data packets to be propagated through the network before reaching the base station.

A different spin on the Hierarchical approach leads us to the Clustering approach, seen in Figure 3. This scheme is similar to the Hierarchical scheme, but here we divide up the nodes into clusters based on distances between nodes. Once we have found a good cluster, one node will be the cluster head, responsible for gathering the data of the cluster and forwarding it to the base station. This system is advantageous because nodes only have to talk to a very close neighbor and there is generally only one hop between a node and the base station. The disadvantage, along with slightly increased communication cost due to propagation, is that the cluster heads will have a significant power drain with their forwarding communications duties. The biggest advantage of this approach is that it allows us, if we know the cluster head, to replace that node with a node that contains extended battery life or that would be equipped with memory to store some data and forward it later to the base station, either when the base station queries for it or at a pre-determined interval, saving some communications and energy costs of the cluster head node.

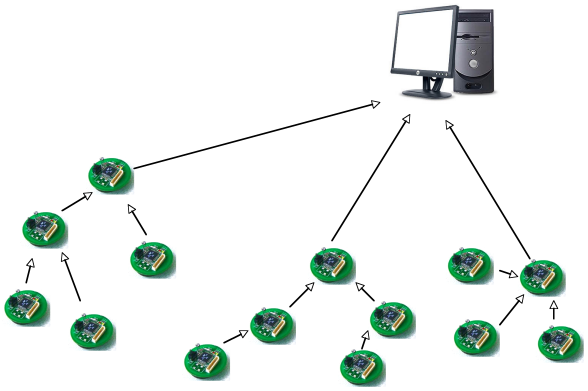


Figure 2: Hierarchical approach: each node will connect to another node closer to the base station (tree-like with base as root.

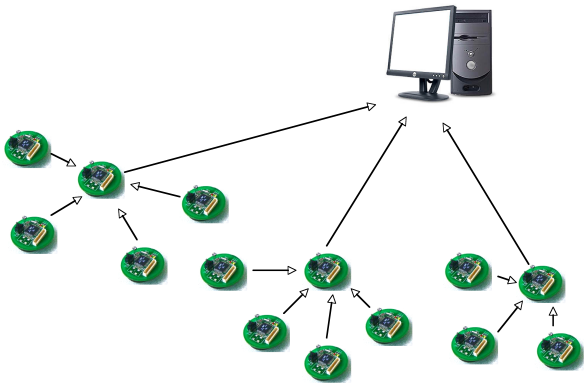


Figure 3: Clustering approach: group close nodes with one connecting to the base station.

LEACH, in [2], seeks to describe a way to get the benefits of a clustered approach while reducing the disadvantages. In their approach, they divide the nodes by distance and one node per cluster is elected as cluster head. Then, among cluster deliberation, an order is determined where each node will spend time as the cluster head, see Figure 4. This reduces the power drain per node so that no one node must bear the cost of forwarding data to a host of nodes. However, it still has the same overall drain of energy. But, it carries all the advantages of the clustering approach, except that nodes with extra batteries or store and send memory cannot be used as the cluster heads since the head is changed over time.

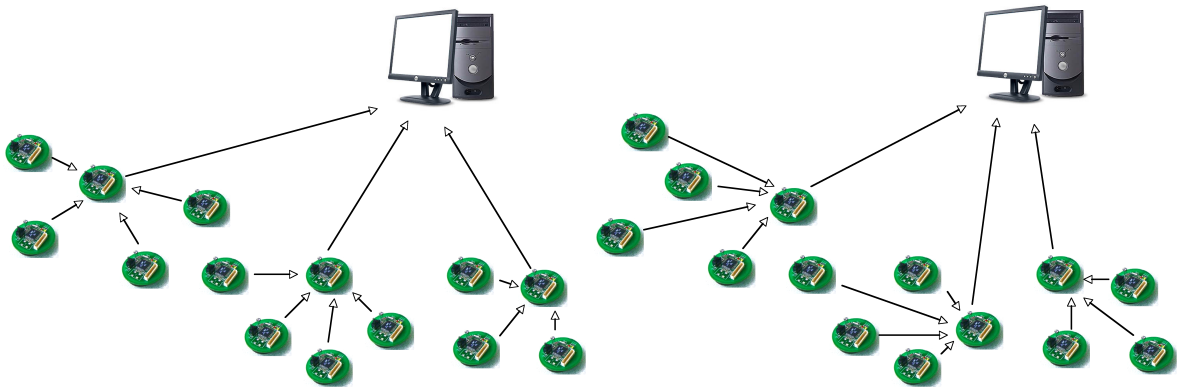


Figure 4: LEACH approach: use clustering approach but rotate cluster-head status.

Manges, in [6], discusses using traditional wired schemes on wireless sensor networks and while he covers the simple approach and a form of the hierarchical approach presented above, he also goes into a peer-to-peer system typical of web networks. He claims that this type of topology will become more prevalent, however this type of system that requires the sensors to be active more often with the state of the network constantly changing is highly energy draining.

4 Related Work

The closest work to this paper comes from Salhieh, et al, in [7]. They propose using traditional routing algorithms through a network of nodes by deploying them in predetermined patterns. Once the nodes are deployed in a set pattern, they use a distributed version of the shortest path algorithm to send messages. Each node is defined by a tuple defined by the distance to the edge in all directions. For example, a four-neigbor graph would have an example tuple of $(1, 3, 5, 2)$ in which 1 is the distance to the west end, 3 hops to the top edge, 5 hops to the right edge, and 2 hops to the edge below. Destination tuples are subtracted from the sender's tuple to determine direction, then each of the neighbors' tuples are subtracted from the destination to determine the right direction to send the data in the graph. This is repeated at every node until the data arrives at its destination. They also take into account the energy level and choose a node with higher energy.

They claim that the most energy efficient method is to send data to the edges and then propagate it along the edge to the destination. This scheme saves energy because each node that sends data to its neighbor is heard by all the other neighbors, which must choose whether to listen to the

message or discard it and return to sleep. Energy is lost in all neighbors for each send that occurs, so sending the data to a place with fewer neighbors saves more overall energy.

One discussion that they avoid since their scheme assumes that the nodes are laid out by hand beforehand is the usability of this scheme in randomized networks.

5 Problem Definition

This paper addresses one idea for an algorithm based on the work in [7] as described above, but instead of using fixed sensor positioning, we will attempt to consider random distributions of nodes transformed into a grid-like structure to see if we can increase the time energy efficiency of nodes. This approach would allow us to use multiple paths to send data to the base station, unlike most current tree-like models. All of the topologies we discussed in the Background section used a tree-like structure, which required the data from each node to traverse the same path to the base station.

Along with this idea, we will consider using a Shortest Path Scheme similar to that in [7], but adding a dynamic cost-per-edge assignment as we send data along paths through the sensors.

6 Proposed Solution

The algorithm proposed will out of necessity be a distributed one. The most cost inefficient part of this algorithm will be the initial setup of the system into a grid-like structure. Each node must send out a small radio signal, and increase until it gets at least 4 responses from neighbors. Upon receipt, it will chose the 4 closest neighbors and make specific connections with them. If multiple neighbors are equidistant, we will choose the most spread out nodes, as seen in Figure 5. If a neighbor choice already has stopped accepting neighbors, the node will choose the next candidate in its candidate pool or send out another broadcast message to acquire more candidates for its pool.

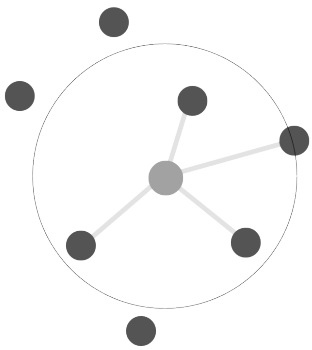


Figure 5: Our node must choose either the 4 closest neighbors or the 4 most spread out.

Once the makeshift graph is formed, the base station must send out a message to all the nodes as to its location in the grid. It will broadcast to its neighbors that it is 0 hops to the base station,

whom will formulate that they are 1 hop away, and this will propagate to all the nodes.

At this point, each node knows that it wants to send its data to a node with a lower or at least equal distance to the base station than itself. Each node will also keep track of energy cost to each neighbor. If a node sends a message to a neighboring node, it will increase the cost of that edge by 2. Likewise, all the nodes that are in hearing distance, the costs of those edges will increase by 1.

Therefore, for each node to send a message, it must choose a neighbor with the lowest energy cost and the lowest distance to the base station or make a compromise between the two.

6.1 Algorithm

To make this description clearer, the algorithm, in pseudocode, can be found in Figures 9, 11.

6.2 Simulation

A simulation was written in C++ to randomly generate nodes in a 10x10 space. Then the algorithm was carried out on these nodes. This algorithm is incomplete, however, because after running a simulation and human testing it, it does not provide an optimal grid of the nodes and has the possibility to leave out nodes that choose neighbors later. The optimal path is not chosen, and moreover, the paths are different depending on the interleaving of the distributed algorithm. Differing paths is not the main problem, though, because even though the optimal path is not chosen, in some cases the resulting graph would either be a disconnected graph or have one edge between otherwise disconnected graphs, resulting in either lost traffic or a bottleneck and energy drain. The problem with this algorithm is starvation. The first nodes that pick their edges and get replies from other nodes effectively keep the nodes they communicate with from choosing their optimal edges and they keep other nodes from forming edges where they are needed. They do this by all choosing a "hot spot" node that is needed by later nodes, but that node is filled up before the later nodes have a turn to ask for connections, resulting in nodes starved for connections.

A sample graph created under this algorithm can be seen in Figure 6.

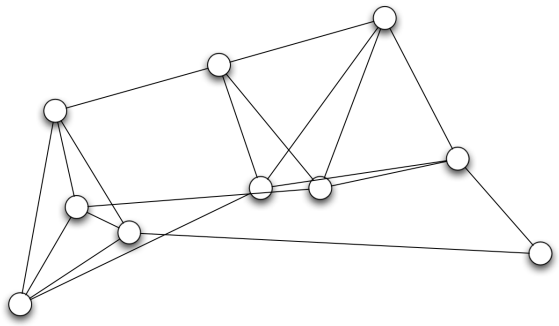


Figure 6: Sample 10-node graph from the simulation of the first algorithm.

6.3 Revised Algorithm

Since we cannot provide an optimal gridding of the nodes, we will consider an algorithm that provides a graph. This algorithm is an approximation algorithm based on the prior one, but does

not try to create any ordered grid.

The base station will initiate the neighbor linking by connecting to its four closest neighbors. They will then remember that they have been tapped by the base station and will then connect to the three closest neighbors from them (except from which they were initially tapped).

Each node, in turn, when it receives the "get your neighbors" message will connect with the three closest neighbors to it and make them part of its neighborhood. Any node may have more than 4 edges into it if another node connects back to it in later execution. Basically a node will always accept an incoming neighbor request, but will only make three neighbor requests of its own.

If a node does not get the "get your neighbors" message after a predetermined timeout, it will connect directly to its four closest neighbors.

The rest of the algorithm will continue as the previous one did, with the base station sending out a message and the nodes becoming aware of the number of hops from each neighbor to the base station. Messages will be sent out the same way as well. This version of the setup algorithm can be found in Figure 10.

6.4 Revised Simulation

Using the simulation method from above, the new algorithm was simulated by generation and by hand, and in the randomized cases considered, worked to connect the graph and to give each node at least four neighbors. Running this simulation will give us a graph similar to that in Figure 7.

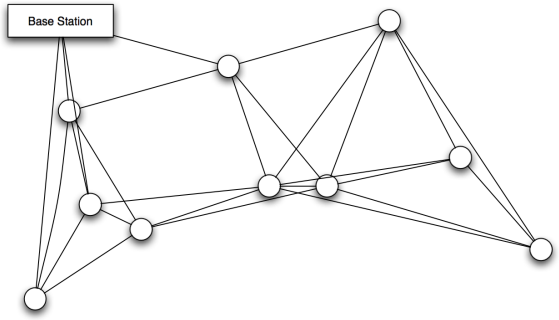


Figure 7: Sample 10-node graph from the simulation of the second algorithm.

6.5 Data Propagation

Now that we have a graph with a sufficient number of edges to perform routing techniques, let us run through an example. First, consider Figure 8 which displays the hop counts per edge that are set at the end of the setup algorithm.

After this setup, we can trace the data path coming from the node at the top right of the graphs as follows. It will always pick the path with the lowest number of hops to the base station, however, when it has already traversed a path too many times and wants to save energy, it will choose a slightly longer, yet less travelled path. This diffuses the energy spent among more nodes, keeping certain nodes from burn out. Figure 12 shows this in motion. Communicating in this way always works and always gets data to the base station, but it does not necessarily take the shortest

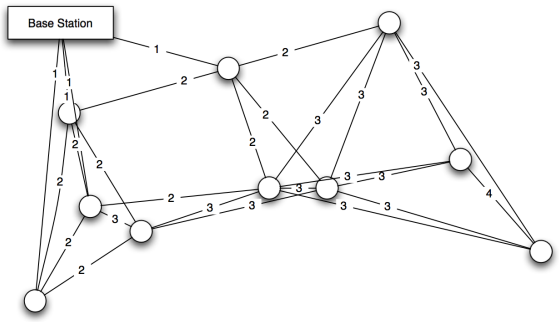


Figure 8: Sample 10-node graph with hop counts included for each edge.

path, which is acceptable since we only wish to reduce power and are okay with taking a slightly longer path to do so. We also effectively shift the workload around, moreso than LEACH since we can take multiple paths around the nodes, and no node must support extensive forwarding because there are multiple paths to the base station.

7 Discussion

7.1 Complexity

This algorithm is interesting in its complexity. We will look at the complexity per node by looking at the general complexity of each algorithm of the setup and communication.

The setup algorithm appears to be on surface level a $O(n^2)$ algorithm because of nested `for` and `while` loops, however, it is dependent on the initial choices of `MIN_DISTANCE` and `MIN_INCREASE`. In the best case, it is $O(n \log n)$ because `MIN_DISTANCE` will be large enough to get at least 3 new neighbors, have to sort once, and we will traverse over that list twice: once to add them to the perspectives list, a second to move them to the neighbors list. In the worst case, we will get something ridiculous, by setting `MIN_DISTANCE` too small and `MIN_INCREASE` small as well, we may be checking the same node hundreds of times before adding a second or third. This is worse than polynomial time, however, we are guaranteed to eventually get other nodes for neighbors if `MIN_INCREASE` > 1. On average, we will never actually hit the worst case because it deals with human choice of initial signal strength. We will also hopefully pick up on average at least one node per signal strength increase (assuming good human choice), and will communicate with nodes, adding them to our perspectives for an average of $O(n^2 \log n)$ complexity; traversing our lists of increasing length up to n times and sorting n times. Remember, though, that $n = 4$ on average.

The sending algorithm is just over linear because we have to pick a decent sorting algorithm, and we sort once and then run through our list once, giving us a general complexity of $O(n \log n)$.

7.2 Differences from a Similar Approach

This algorithm uses the same principles as that in [7], however it is remarkably different as well. Since the nodes are not hand placed, we must adapt them into a structure that can use routing schemes to get the message through the network of nodes, which turned out to be a harder problem

than initially considered. However, using an approximation of a grid-like structure, we were able to develop a graph in a decent time.

Also, this algorithm cannot benefit from the scheme of tuples to solve an approximation of the shortest path problem distributively through the nodes. Therefore, we must rely that all communication will be directed towards the base station, and rely on picking the minimum hop distance to the base station at each point along the path to the base station. This could result in cyclic movement of data in the most problematic case, but even that case will eventually be solved because the weight of the path will grow until the data leaves that cycle.

8 Conclusion

In conclusion, this algorithm is not optimal, but it gets the job done. It requires a significant amount of communication cost up front, and if the sensors are deployed and then never changed or moved, that cost will only be accrued once. However, if shifting or dying nodes become a problem and reinitialization is required, this cost could become too much. Other algorithms avoid this cost because they utilize either specific sensor placement [7] or tree like structures where only minimal setup is needed to find the closest neighbor [9], but LEACH seems to use a comperable amount of communication to set up its clusters in its initialization phase [2, 3].

After initial setup, the algorithm significantly reduces the amount of amplification that must be done to transmit the signal to the base station. Each node only has to talk to its closest neighbors and it does not wear out any neighbor by using the same path repetitively. It also seeks to take a minimal path to reach the base station, while keeping energy in consideration to not choose the minimum path every time. It balances minimizing power consumption through sending to a closest more direct node and utilizing the shortest path but not too frequently to cause burn out.

This approach also leaves many considerations out that could be included for tremendous affect on power. Including nodes that have extra battery life and routing through those nodes more often could increase the overall system power. Also, having the outside edge of the network powered would increase system energy efficiency as well. There is also discussions about placing sensors with data storage capabilities into a tree structured system [8]. Those nodes could be invaluable here at certain hot spots in the graph and could be highly used in the nodes surrounding the base station to reduce significant traffic flow until the base queries the nodes for the data. It would make a more interesting design because the queries would have to go to nodes that could answer and would have to be above the general routing scheme. The possibilities of combining this idea with other existing ones is amazing and will be interesting to consider how data flow is considered in future papers.

Bibliography

1. Akyildiz, Su, Sankarasubramaniam, Cayirci. "A Survey on Sensor Networks." IEEE Communications Magazine, August 2002.

2. Heinzelman, Chandrakasan, Balakrishnan. "Energy-Efficient Communication Protocol for Wireless Microsensor Networks." IEEE, 2000.

3. Heinzelman, Sinha, Wang, Chandrakasan. "Energy-Scalable Algorithms and Protocols for Wireless Microsensor Networks."

4. Hill, Szewczyk, Woo, Hollar, Culler, Pister. "System Architecture Directions for Networked Sensors." PSPLOS-IX, November 2000.

5. Mainwaring, Polastre, Szewczyk, Culler, Anderson. "Wireless Sensor Networks for Habitat Monitoring." WSNA, September 28, 2002.

6. Manges. "Wireless Sensor Network Topologies." Sensors, May 1, 2000.

7. Salhieh, Weinmann, Kochhal, Schwiebert. "Power Efficient Topologies for Wireless Sensor Networks."

8. Sheng, Li, Mao. "Data Storage Placement in Sensor Networks." MobiHoc, May 2006.

9. Wang, Heinzelman, Chandrakasan. "Energy-Scalable Protocols for Battery-Operated MicroSensor Networks."

Appendix

```
void setup() {
   distance = MIN_DISTANCE
   while (neighbors.count() < 4) {
      while (perspectives.count() < 4) {
         send(bcast_msg, distance)
         for (each received message rec_msg_i) {
            perspectives.add(rec_msg_i.sender, rec_msg_i.strength)
         }
         distance += MIN_INCREASE
      }

      sort(perspectives, strength)

      for (i = 1; i <= perspectives.count(); i++) {
         send_neighbor("be my neighbor", perspectives[i])
         receive(ack, perspectives[i])
         if (ack == 1) {
            neighbors.add(perspectives[i])
            if (neighbors.count() == 4)
               break
         }
      }
   }
}
```

Figure 9: Initial version of the pseudocode for the set up of each node in the graph.

```
void setup() {
   distance = MIN_DISTANCE
   while (true) {
      if (receive("get your neighbors", neighbor1) || timeout.isExpired())
         break;
   }
   neighbors.add(neighbor1)

   while (neighbors.count() < 4) {
      while (perspectives.count() < 3) {
         send(bcast_msg, distance)
         for (each received message rec_msg_i) {
            perspectives.add(rec_msg_i.sender, rec_msg_i.strength)
         }
         distance += MIN_INCREASE
      }

      sort(perspectives, strength)

      for (i = 1; i <= perspectives.count(); i++) {
         if (!neighbors.contains(perspectives[i])) {
            neighbors.add(perspectives[i])
            if (neighbors.count() == 4)
               break
         }
      }
   }
}
```

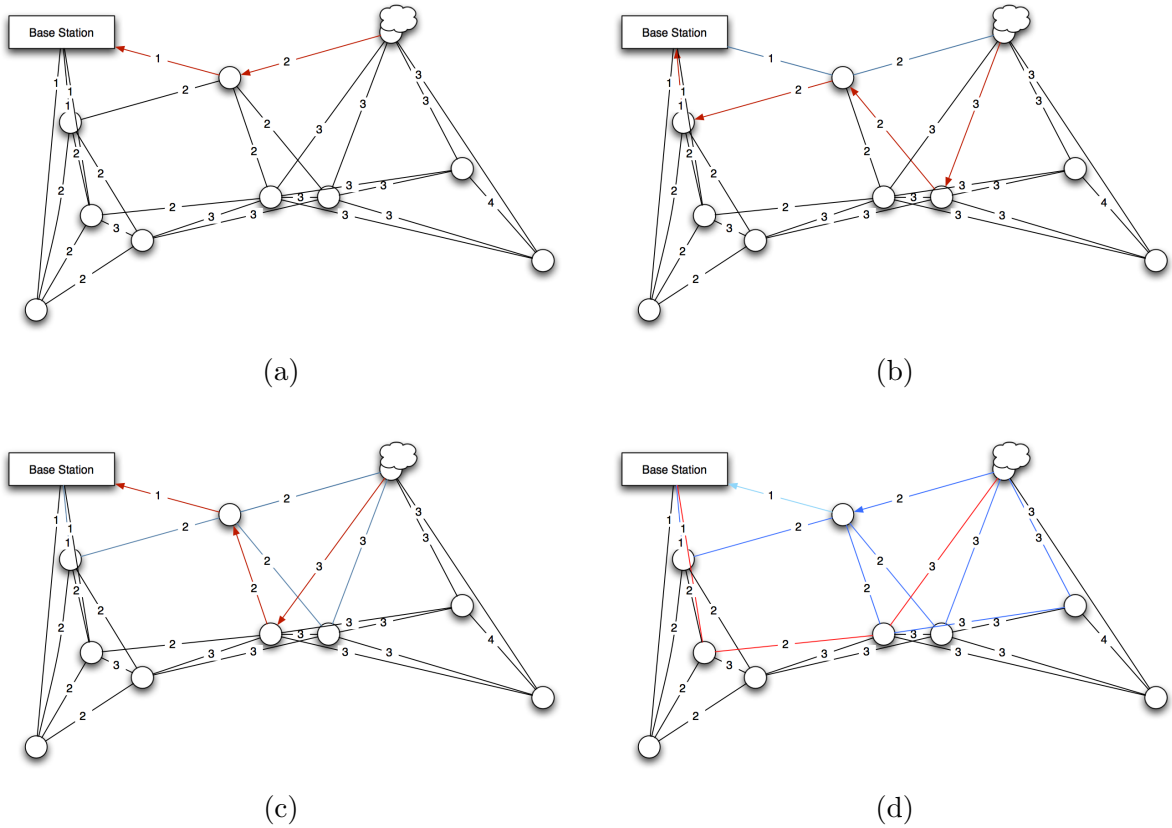
Figure 10: Revised version of the pseudocode for the set up of each node in the graph.

```
void send_data(data) {
   sort(neighbors, hops to destination)
   min_weight = neighbors.get_min_weight()
   neigbors.increase_all_weights_by(1)

   foreach (sorted_neighbor in neighbors) {
      if (sorted_neighbor.edge_weight < THRESHOLD * min_weight) {
         send(data, sorted_neighbor)
         sorted_neighber.increase_weight_by(1)
         return
      }
   }

   send(data, neighbors[LAST])
   neighbors[LAST].increase_weight_by(1)
   return
}
```

Figure 11: Pseudocode for the sending of data.



(a) (b)

(c) (d)

Figure 12: This is a sample step through of the sending portion of the algorithm. We trace data sent from the top right node to the base station. Red lines indicate the current travel path and blue lines indicate already travelled paths, with lighter shades for more travelled paths. We will also assume the threshold is low so we hit it immediately. Illustration (a) shows our first communication, which takes the shortest path. After the threshold is met, as in (b), we will take another path with the smallest hop count. It does not matter which we choose, so the node will pick a random one. (c) and (d) show more subsequent paths, but note in (c) that the shortest path is chosen from the node closest to the base station because all of its paths have been travelled upon and it then selects the shortest route.