Energy-aware On-demand Scatternet Formation and Routing

Tomas Johansson and Lenka Carr-Motyčková
{tomasjo, lenka}@sm.luth.se

Department of Computer Science and Electrical Engineering
Luleå University of Technology
SE-971 87 Luleå, Sweden

Abstract

Power-aware routing algorithms aim to extend the lifetime of a wireless network by avoiding to route over nodes that are close to being depleted of energy. Previous work in such algorithms often require each node in the network to be aware of the topology of the entire network.

In this paper we investigate power-aware routing in Bluetooth networks. Routing in a Bluetooth network involves creating a scatternet structure in order to facilitate the communication between nodes in different piconets. We propose a new routing algorithm that does not require each node to have any knowledge of the network except for its neighbors. This is done by collecting information regarding potential routing paths in the routing messages themselves.

We also present a simplified version of the algorithm for general wireless networks, which does not involve scatternet formation and therefore requires less communication overhead.

1 Introduction

A wireless ad hoc network differs in many ways from traditional wired networks. In an ad hoc network there is no fixed infrastructure, and the nodes may not have knowledge of the entire network topology. The topology might also be constantly changing if the nodes are mobile. Also, nodes in a wireless network usually use a battery as power source, and they will therefore only have a limited amount of energy. As nodes run out of energy, the network will eventually become disconnected. Therefore, when developing routing algorithms for wireless networks, it is imperative to take the energy levels of the nodes into account in order to avoid routing over nodes that are close to becoming depleted of energy.

Routing in Bluetooth networks present more challenges than routing in general ad hoc networks. For every commu-

nication between two Bluetooth devices, they must form a master-slave relationship. A Bluetooth node can only take part in one such relationship at a time. Therefore, creating a path between two Bluetooth nodes is not a trivial problem, especially if there are several simultaneous communications in the network that intersect each other.

In this paper, we propose a power-aware routing algorithm for ad hoc networks in general and Bluetooth networks in particular. Previous work on power-aware routing that we are aware of has either required each node to know the topology of the entire network, or used a simplistic model to decide whether a path is acceptable to use. In our algorithm, each node only needs to be aware of its neighbors. Information regarding potential routing paths is collected by and stored in routing messages, allowing individual nodes to select the best path in the network with only local knowledge.

The rest of the paper is structured as follows: Section 2 introduces how nodes in a Bluetooth network communicate with each other, as well as some of the major issues to consider. Section 3 provides a short survey of routing algorithms for wireless ad hoc networks, with focus on energy-aware routing algorithms. Section 4 presents a brief survey on different existing approaches to scatternet formation, including an energy-aware algorithm. We present our algorithm in Section 5. In Section 6, we present a version of the algorithm for use in general ad hoc networks, without the restrictions imposed by the Bluetooth specification. We evaluate our algorithm through simulations in Section 7. Finally, Section 8 contains a discussion of our research as well as directions for our future work.

2 Bluetooth networks

Whenever two Bluetooth devices communicate, one must assume the role as *master* and the other the role as *slave*. Together they form a *piconet*. Each piconet consists of one master and an unlimited number of slaves, but

there can only be at most 7 active slaves simultaneously. If there are more than 7 slaves in a piconet, some of them must therefore be *parked*, i.e. inactive.

A collection of piconets is called a *scatternet*. In order to communicate between piconets, nodes can be members of several piconets, even if they only can be members of one piconet at a time. A node can have different roles in different piconets: it can be slave in one piconet and master in another. A node that is a member of two or more piconets is referred to as a *bridge node*. To switch from one piconet to another, a node changes to the hopping frequency sequence of the other piconet. Therefore, one important issue in scatternet formation is to minimize the necessary piconet changes in order to reduce the delays.

3 Routing in ad hoc networks

In an ad hoc network, there is no fixed infrastructure that is responsible for the routing. Instead, every node may function as a router, and must therefore be able to make routing decisions based on its possibly limited knowledge of the network.

Routing algorithms for ad hoc networks can be divided into two main groups: table-driven and on-demand algorithms. In table-driven algorithms, each node maintains a routing table with entries for each destination and proactively finds routing paths. In on-demand algorithms the nodes do not store the same amount of information, but instead have to discover the route every time they need to send a message. Examples of on-demand algorithms are the Ad hoc On-demand Distance Vector routing (AODV) protocol [9], and the Temporally Ordered Routing Algorithm (TORA) protocol [8].

The AODV algorithm focuses on finding the single shortest path to the destination, while the TORA algorithm finds several possible paths to the destination: each node's outgoing links is divided into "upstream" and "downstream" links based on whether they lead to the destination or not.

3.1. Energy-aware routing algorithms

Recently, there have been an increased interest in routing algorithms that take the energy levels of the nodes into account. Nodes in wireless ad hoc networks typically use batteries as their power source. These batteries often have a limited capacity, which is depleted with every transmission. It is important to consider this when designing routing protocols for ad hoc networks. Only finding the paths that require the least amount of power is not enough; some nodes might still be used much more than others, and will therefore be prematurely depleted.

One example of an energy-aware routing algorithm is the CMAX algorithm [1], where each link in the network is assigned a weight. The weight of a link (i, j) when sending the message k is defined as

$$w_{ij} = e_{ij}(\lambda^{\alpha_i(k)} - 1),$$

where e_{ij} is the energy consumed by sending a unit message over the link (i, j) , and $\alpha_i(k)$ is the fraction of the energy of node i that has been used at the time when message k is generated (referred to as the *energy utilization* of a node):

$$\alpha_i(k) = 1 - \frac{E_i(k)}{E_i}. \quad (1)$$

$E_i(k)$ denotes the residual energy of node i at the time message k is generated, and E_i denotes the original energy of node i . λ is a parameter used to control how much the energy utilization of the nodes influence the link weights. As can be seen, the weight of a link increases with the energy needed to traverse it as well as with the energy already spent by the transmitting node. The shortest path, with respect to link weights, is selected by the CMAX algorithm. This algorithm requires that each node knows the energy utilization of the other nodes in the network. In order to spread the power information through the network, each node periodically broadcasts its power information to the nearest nodes.¹ This is referred to as the limited flooding approach. Using this method, a node is likely to have better information of the power level of its nearest nodes compared to the more distant ones.

The max-min zP_{min} algorithm [3] finds paths that avoid nodes with only a small fraction of their battery power remaining. The goal is to find the path with the maximal minimal fraction of remaining power after the message is transmitted. This value, called the *residual power fraction*, is similar to the *energy utilization* described above, and can be written as

$$1 - \alpha_i(k)$$

using the same notation as in Equation (1).

However, the path must also not use more power than a constant factor $z \geq 1$ of the smallest possible power consumption P_{min} . This results in finding a path that avoids using up the power of individual nodes, while still maintaining an upper bound on the total power consumption of the path.

¹Exactly how far the power information should be broadcast is a parameter of the algorithm. Simulations in [1] indicate that the lifetime of the network decreases when the broadcast distance grows higher than a certain threshold, since too much energy is spent on broadcasting energy information.

Both of the above algorithms require that each node has full knowledge of the topology of the network, an assumption that is not realistic in larger networks.

Another approach is to extend existing on-demand algorithms in order to compute energy-efficient paths. In Reference [6], the authors enhance the AODV protocol by adding energy information in the route request (RREQ) messages. In this way, the destination node can decide on the path with the lowest energy cost, using a timeout value in order to receive several RREQ messages before deciding on a path and generating a route reply (RREP) message. Simulations in the paper indicate that the effective reliable throughput is significantly increased using the extended AODV algorithm. Unlike the algorithm presented in this paper, the extended AODV algorithm only considers total energy cost and estimated bit error rates in order to select the path; it does not consider the amount of energy that remains in the nodes.

Wang et al. [11] presents a power-aware on-demand routing protocol that, like the one presented in this paper, takes the remaining energy of the nodes into account. The two main differences between the algorithm presented in this paper and Wang's algorithm is that the latter does not take the total power cost of paths into account, and the cost metric does not change over time to reflect the changing amount of energy in the nodes.

4 Scatternet formation

There are two main approaches for scatternet formation: the *pro-active* and the *on-demand*, or *reactive*, approach. Pro-active algorithms are advantageous when data transfer occur frequently, since the algorithm will produce a scatternet that covers the entire network. However, if traffic is less frequent, creating and maintaining all the Bluetooth connections can be considered to be a waste of energy. Instead, on-demand algorithms only establish temporary connections when they are needed.

4.1 Proactive algorithms

One of the earlier proactive formation algorithms is described in Reference [14], where a single root node initiates the construction of a *Bluetree*. The root node assumes the role of master and selects all its neighbors as slaves, and the slaves in turn assign themselves as masters in new piconets and searches for unassigned neighbors to be their slaves. This procedure is repeated until the entire network is covered by the bluetree.

In [4] the authors use location information in order to produce a planar subgraph, which can improve the performance of routing algorithms. This optional step is followed by assigning roles to the bluetooth nodes, a process that

takes several iterations. In each iteration, every node that has higher degree than any undecided neighbors assigns itself to be a master, and chooses no more than 7 neighbors as slaves by applying the Yao structure [13] to all its neighbors. Using the Yao structure limits the number of slaves in each piconet while still maintaining connectivity, but it also means that the nodes need to know their exact location in order to run the algorithm.

The BlueNet formation algorithm [12] aims to create a scatternet with an upper bound of the number of slaves in each piconet, while at the same time avoiding the bottlenecks that can occur in a tree-based scatternet structure. The maximum traffic flow metric (MTF), a measure of all possible maximum multi-commodity flows in the network, is defined and used to evaluate the scatternets produced by the algorithm.

4.2 On-demand algorithms

In Reference [2], a hybrid solution is presented. A permanent control scatternet is constructed in order to support routing messages, while an on-demand scatternet is created between the sending and receiving node whenever data communication is needed. The idea is that the underlying control scatternet will facilitate path discovery, while not consume too much energy itself. When constructing the control scatternet, there is no limitation on the number of slaves in each piconet since reducing the number of piconets is seen as more important than reducing communication latency.

Another on-demand algorithm is specified in [5]. In order to create the scatternet, the authors use the single role approach: instead of using nodes that are slaves in one piconet and masters in another, communication between piconets are achieved by nodes that are slaves in several networks in order to reduce the number of piconets and indirectly the communication delay as well.

The On-Demand Bluetooth scatternet formation algorithm (ODBT) [7] constructs a scatternet with a tree topology, starting with one root node. When a connection that is included in the tree is lost, the entire disconnected subtree is *flushed*, so that all nodes in that subtree can rejoin the tree. Even though this is an on-demand algorithm, the created scatternet connects all the nodes in the network, regardless of which nodes that are trying to communicate.

4.3 Energy-aware Routing in Scatternets

Recently, there has been work that aims to extend the lifetime² of scatternets by considering the amount of power remaining for each node, such as in [15]: when a node wants

²In this paper, the "lifetime" of a network is defined as the amount of time until the first node in the network has been depleted of energy.

to initiate a communication, it sends out a route request to the destination (the sink node) by flooding the network. When the destination receives the request, it responds with a route reply. To form the temporary scatternet between the source and the destination, the destination sets its role to master, and for the rest of the nodes in the path every other node is a master while the other nodes are slaves in two different piconets. Unlike the algorithms described in Section 2, no node needs to know about the topology of the entire network using this approach.

This approach has a relatively simple mechanism to avoid low-powered nodes: either a node has enough energy to forward a message, or it has not. As a node is reached by route requests that have travelled different paths, there is no way for the node to distinguish between the different paths based on the power levels of the nodes in the paths.

We propose a different approach, where we introduce a metric to evaluate how “good” a given path is with respect to power consumption. Nodes then rank a number of alternative paths in order to find the “best” one. We will explore this idea further in the next section.

5 Proposed algorithm

In this paper, we propose an on-demand scatternet formation and routing algorithm which creates a local, temporary scatternet that forms a path between the sending and receiving node for the duration of the data transfer. In this section, we present an overview of the proposed algorithm. Detailed information about the different phases of the algorithm will be presented in the following subsections.

In order for the nodes to be able to communicate with each other, there must first exist an underlying, permanent, scatternet that covers the entire network. This permanent scatternet, or *control scatternet*, is used to be able to forward the routing messages through the network.

Before a source node u can initiate a transmission to a destination node v , it must first find the most suitable path to use given the current energy situation in the network. We create several alternative paths between the source and the destination, while measuring the energy parameters for each path. The idea is to choose a path that avoids nodes that have a low amount of energy left, while still keeping the total energy cost of the path as low as possible. Since the energy parameters are collected as the alternative paths are created, we always have access to the current energy status of each node. Using this method, the nodes do not have to spend energy on repeatedly broadcasting their energy status to their neighbors.

From the different paths, the optimal one (according to the metrics described below) is selected. The final step is to construct a temporary scatternet between nodes u and v in order to support data transmission over that path. When

the temporary scatternet has been constructed, node u can initiate the transmission.

In the rest of this section, the three steps of the algorithm are explained in more detail.

5.1 Control scatternet

As stated previously, the permanent control scatternet is used to propagate the routing messages through the network. This paper does not present any algorithm for producing a control scatternet. However, in general it is desirable to have larger piconets. This might increase the delays since slave nodes might have to switch between active and passive mode, but the reduced number of piconets will simplify the inter-piconet communication.

The existing BlueStars algorithm [10] is one algorithm that may be used to produce the control scatternet. The BlueStars algorithm produces a scatternet that is a connected mesh with multiple paths between any pair of nodes. The algorithm does not limit the piconet size; there can be more than 7 slaves in a piconet. As discussed above, this is a desirable property for our algorithm.

During the formation of the scatternet, each node collects information about its neighbors. When the scatternet has been formed, each node knows the energy cost to send a message to each of its neighbors, and it also knows what piconet each neighbor belongs to, even if the node and the neighbor are not connected in the scatternet. In this context, a “neighbor” is defined as a node within communication distance to the original node, regardless of whether the nodes are connected in the control scatternet. Each master node knows of all the piconets that connects to its own piconet, as well as the identity of all the slave nodes in its piconet. The master also knows of all *possible* gateway nodes: all the nodes that have at least one neighbor belonging to another piconet.

The topology of the network can change over time, and therefore the information may need to be updated. This is done at previously scheduled times, where all the Bluetooth nodes participate in the information exchange.

5.2 Finding a path

Finding a path is done in two steps: the first step is finding piconet sequences between the sending and the destination node, and the second step is finding paths within the previously found piconet sequences. The sending node starts by broadcasting a *route request*, and in turn receives *route replies* containing information of the found piconet sequences. For each piconet sequence, the sending node then sends a *path request* in order to find paths within that sequence. These steps will be described in more detail after a description of our proposed metric.

5.2.1 Metrics

The receiving node must choose the best path in terms of the energy depletion in the nodes. To do that, it is first necessary to come up with a way to assign a numerical value to each path, representing how desirable it is to use in terms of the energy depletion in the nodes in that path. The tradeoff is that a more detailed metric gives a more realistic picture of a path's properties, but such a solution does not scale well as the length of the paths increases.

When evaluating the “quality” of a path (with respect to the energy levels in the nodes), the following definitions will be used: we define the potential remaining power of every link (i, j) in the path as $rp_{ij} = p_i - e(m)_{ij}$, where p_i is the power level at node i , and $e(m)_{ij}$ is the power needed to send the message m directly from i to j . (We assume that the nodes can adjust the power used for sending messages.) This means that rp_{ij} is not the same as rp_{ji} . Note that the power level p_i of a node is not necessarily the current power level, since some power may be reserved for future transmissions. How the power level is estimated is explained in more detail in Section 5.2.4.

For the path $P = (l_1, l_2, \dots, l_n)$, where $(l_1, \dots, l_n)$ represents the links in the path, the metric that represents the minimum potential remaining power for P is defined as

$$Min(P) = \min rp_l | \forall (l) \in P$$

In order to calculate the $Min(P)$ value for a path P , it is necessary to know the power levels of all nodes included in that path. There are two ways for a node in the path to calculate its power level: assume that it will be included in the eventual path, or assume that it will not. If it assumes that it will be included, it reports its power level as the current power level minus the amount of energy needed to forward the data to the next node in the path. This “pessimistic” approach guarantees that no node is accidentally drained of energy, but it might also lead to erroneous assumptions if the traffic density is low. In the other, “optimistic”, approach, the node always report its power level to be the same as its current power level. This way, if a node is simultaneously included in several paths before data transmission has begun in any of the paths, it could be drained of energy before all data has been forwarded.

Another important factor is the amount of energy needed to send the message m over the path P : spending too much energy trying to route around nodes with little remaining energy will also lead to premature depletion of individual nodes. The metric that represents the energy consumption of the path P is defined as:

$$Pow(P) = \sum_{i=1}^n e(m)_{l_i}$$

5.2.2 Route request

As the control scatternet consists of several interconnected piconets, the first step of the algorithm is to find a number of sender-receiver paths where each path can be described as a sequence of piconets. This is done by having the sender broadcast a route request over the network and receiving route replies.

The sending node broadcasts a route request (RREQ) message over the control scatternet. The route request message is identified by the sending node's id n_{id} in combination with the sequence number $SeqNum$ of the message, which both are included in the message. All RREQs concerning the same data transmission have the same $SeqNum$. A node assigns sequence numbers to its data transmissions in the order they occur. The RREQ also contains the id of the destination node.

If the sending node is a slave, it sends the RREQ message to its master. When the master has received the RREQ, it sends a copy of the RREQ to all the neighboring piconets. If the sending node is a master, it sends the RREQ to the neighboring piconets directly.

When the RREQ arrives in another piconet, it is forwarded to the master. The master records where the RREQ came from, and forwards the message to all other neighboring piconets. If a piconet receives several RREQs with identical n_{id} and $SeqNum$, it knows that they all concern the same transmission, and it is only necessary to forward the first message it receives. This reduces the total number of messages sent, which reduces the energy consumption.

The RREQ messages have a limited lifetime, in terms of number of hops, in order to limit the amount of forwarding.

5.2.3 Route reply

When a RREQ message reaches the destination piconet, the master node in that piconet forwards the message to the destination node. In order to receive several possible paths, the destination node waits for a specified time period t_{rreq} in order to receive several route request messages, that all have travelled unique paths.

At the end of t_{rreq} , the destination node sends a route reply message (RREP) to the first k piconets it received a RREQ from, where k is a constant. If the node has received fewer than k RREQs, it replies to all piconets it received a RREQ from. As with the route request messages, the route reply messages is forwarded using the control scatternet. The $(n_{id}, SeqNum)$ pair included in the RREQ is included in the RREP as well. As a piconet master receives a RREP, it forwards it to all neighboring piconets it recieved the corresponding RREQ from.

As the RREP is forwarded through the network, it stores the ID of each piconet it encounters. This way, a RREP

always contains a list of the piconets it has travelled. This information will be used in the next step of the algorithm.

5.2.4 Path request

When the sender receives the first RREP, it waits for a specified time period t_{rrep} in order to receive several route replies. Every RREP the sender receives represents one piconet sequence from the sender to the destination, and this information is stored in the sender.

When the time period has elapsed, the next step is to find paths in each piconet sequence that the sending node has learned from the RREPs. Exactly one path request (PREQ) message is sent along each piconet sequence reported by a RREP in the previous phase. Each PREQ message builds paths as it travels through the piconets and stores information about them.

Even though the PREQ message uses the underlying scatternet to travel through the piconets, the paths it finds might use links that are not part of the control structure. The paths might even skip over some of the piconets in the piconet sequence, depending on the connectivity of the nodes. However, the paths will never include nodes that are not part of any piconet in the sequence.

The PREQ will build and store information on several paths from the sender to the destination. Every PREQ message contains the sender node id n_{id} , the sequence number of the data transmission $SeqNum$, the piconet sequence P_{seq} it will travel, the size of the data message m , and a set consisting of the paths that has been found. The set is designated S . A PREQ can therefore be written as $PREQ(n_{id}, SeqNum, P_{seq}, |m|, S)$.

For each path P stored in S , the following information is kept: the node sequence N_{seq} of the path, and the metrics $Min(P)$ and $Pow(P)$ of the path. The paths are built incrementally: in each piconet, the master node finds the best possible way to extend the paths through that piconet.

In order to describe how the paths are found, we will focus on a single PREQ message as it travels from the sender to the destination. Assume that the PREQ message will travel through the piconet sequence $P_{seq} = (PN_1, PN_2, \dots, PN_n)$. There are three cases to consider: The piconet PN_1 that contains the sending node, the piconets PN_k where $1 < k < n$ that contains neither the sending nor the receiving node, and PN_n that contains the destination node. The following parts will describe each of those cases in more detail.

In PN_1 , the master node first finds the set of potential gateway nodes. A node u is a gateway node iff $u \in PN_1$ and u is within communication range of at least one node v such that $v \in PN_j$, where $2 \leq j \leq n$. For each gateway node u , the best path in PN_1 from the sending node to u is found. The best path P is defined as the path with the

highest $Min(P)$. The set of all such paths is stored in S . The size of S is thus equal to the size of the set of gateway nodes. Note that since a gateway node does not need to connect to a node in PN_2 specifically but rather to a node in any piconet in P_{seq} , some of the paths in S might skip one or several piconets in P_{seq} .

For any piconet PN between PN_1 and PN_n , the procedure is the same when the piconet receives a PREQ. All the paths in S that currently connects to PN are appended so that they connect to a piconet further ahead in the piconet sequence. It will not be possible to use the paths that cannot connect to a node in PN . The set S is therefore first divided into S_A and S_B , where the paths that ends in nodes reachable from PN is stored in S_A and the rest in S_B . The task is to extend the paths in S_A through PN .

The median $Min(P)$ value of all paths $P \in S_A$ is designated *threshold*. The threshold value will determine whether a path can be considered for inclusion in S . As in the first piconet, the master node first finds all potential gateway nodes in PN . Then, for each potential gateway node u the best path that extends one of the paths in S_A to u is found. To do this, each path in S_A is extended to u such that the $Min(P)$ value of the extended path P is maximized. If there are several ways to extend a path that results in the same $Min(P)$, the path with the lowest $Pow(P)$ is chosen.

For each gateway node u the algorithm will find $|S_A|$ potential paths. The algorithm then chooses the path P with $Min(P) > threshold$ and the lowest $Pow(P)$. The set of new paths (one for each gateway node in PN) is designated S_C . All paths in S_C are added to S . All paths in S_A that cannot connect to any piconet further along in P_{seq} are discarded. The new set S now only consists of paths that can connect to piconets further along in the sequence, and the PREQ is forwarded to the next piconet in that sequence.

For the piconet PN_n the process is different; for each path $p \in S$, the best path that extends p to the destination node is found. The definition of “best” path is similar to the previous case, with the difference that *threshold* is defined as the median $Min(P)$ value for all paths $P \in S$. All the found path are stored in S (and all the old paths in S are deleted) as the PREQ message is forwarded to the destination node.

If the node, after being included in a proposed path, does not receive a notification that the path has been selected for a specified time period t_{prop} , it assumes that it was not chosen and adjusts its power level accordingly.

5.3 Scatternet formation

5.3.1 Path reply

When the destination node v receives the first path request, it waits the specified time period t_{preq} in order to receive information about alternative paths. The set of paths for v

to choose from is written as S . Then, v chooses the best paths based on the metrics presented in Section 5.2.1.

The basic idea is to avoid nodes which would be drained of power if they were to forward the data message. In order to achieve that, we define a threshold value $rp_{threshold}$ so that all paths $P \in S$ such that $Min(P) < rp_{threshold}$ are removed from S . Of the remaining paths in S , the path P with the lowest $Pow(P)$ is chosen by v .

Since all nodes in the network will slowly be drained by energy as messages are being sent, we cannot have a static threshold value as it would not reflect the current power situation in the network. Instead, the information stored in the PREQ messages is used to calculate the threshold value: it is defined as the median of $Min(P)$ for all paths in S .

The receiving node sends a path reply message (PREP) back to the sender, and the chosen path is built as the PREP travels through the network. When the PREP message reaches the sending node, the temporary scatternet is completed and the transmission of the data message itself can commence.

6 Power-aware routing in general ad hoc networks

The algorithm described above can be adapted for general ad hoc networks. Since we no longer are bound to follow the Bluetooth specifications, we only need one round of communication: a route request message from the sending node, that arrives in several copies at the destination. The destination node then selects the best path and sends a route reply message along that path. Also, we do not need to create an underlying network to facilitate communication between nodes.

6.1 Route request

The sending node floods the network with a route request message in order to find paths to the destination node. The route request in the general version of our algorithm is similar to the PREQ described previously, in that the metrics that represents the path are stored in the message. Unlike in a scatternet, there is no restriction on the inter-node communication, so we can transmit messages directly between any node pair that is within communication distance of each other. Rather than finding paths on a piconet-level, we find paths on the individual node-level. Each path request stores information about the path it has travelled.

When the RREQ is forwarded, the two metrics stored in the message are updated. When the node i forwards the message to node j , $Pow(P)$ is increased with $e(m)_{ij}$, and if rp_{ij} is smaller than $Min(P)$, $Min(P)$ is replaced with rp_{ij} .

For every initiated route request, a node can receive several copies of the same RREQ message. In order to reduce the number of RREQ messages in the system (and thereby reducing the power consumption), only the first such copy the node receives is forwarded.

For all such unique pairs a node encounters, it stores the lowest $Min(P)$ and $Pow(P)$ it has seen on the RREQ messages it has forwarded. When the node receives another copy of the same RREQ, it compares the metrics included in that message with the two metrics it has stored. If neither metric in the new message is better than the stored metrics, the message is not forwarded. If one or both metrics are better, the node forwards the message and updates its stored metrics.

If the node has not stored any metrics for that specific node ID and sequence number, the RREQ is the first one that the node has received. In that case, the message is always forwarded.

A node calculates its reported power level in the same way as described in Section 5.2.4.

6.2 Route reply

When the destination node receives the first route request message, it waits for a specified time period in order to receive other copies of the message that has travelled alternative paths. It then chooses the best path in the same way as previously described in Section 5.3.1, and sends a route reply message back to the sending node along the chosen path. Unlike in the scatternet, the route request message can travel along the chosen path back to the sender, and inform all nodes included in the path that they were chosen.

7 Simulations

In order to evaluate our proposed algorithm, we performed simulations to compare it with a routing algorithm that always selects the path that consumes the least amount of energy. We generated a topology by placing 25 nodes randomly in a 100*100 unit area, where each node had a range of 32 units. The energy cost c to send a message of a given size between two nodes was modelled as proportional to the square of the distance d , in other words $c = Cd^2$, where C is a constant. Each node started out with the same amount of energy.

Each algorithm was run 20 times. Every time, data transmissions between randomly selected sender and receiver nodes were generated. This was repeated until the first node in the network had depleted all its power, when the simulation was interrupted. Each data transmission started after the previous was completed, which means that the discussions in section 5.2.1 regarding optimistic and pessimistic estimation of the power level is not applicable in this case.

Figure 1. Remaining energy in nodes.

The results from the simulations showed that our algorithm does extend the lifetime of the network compared to choosing energy-optimal paths without taking the power level of the nodes into account. The latter approach resulted in a total of 1185 transmissions over the 20 simulation rounds, while our algorithm ended after 1440 transmissions. Figure 1 show the average distribution of energy in the network when the first node had depleted its power. Our algorithm, represented by the dashed line in the graph, had consumed more energy (at least partly due to the fact that it had routed more messages), but the load had been spread somewhat more evenly through the network.

8 Discussion

Several timeout periods are discussed in this paper, but their exact sizes are not determined here. We believe that they depend on other parameters such as the diameter of the network, which influence the expected distance between the sending and the receiving node.

The algorithm described here will require substantial traffic overhead compared to an algorithm that does not take the power values of the nodes into account. The increased energy consumption due to the routing messages will reduce the lifetime of the network, but this is expected to be outweighed by distributing the data traffic as evenly as possible among all nodes in the network. This is based on the implicit assumption that the amount of data traffic in the application will be much larger than the traffic amount used by the routing algorithm itself.

The next step in our research will be to implement the algorithm described in this paper in a real-life Bluetooth network in order to evaluate its performance. One of the goals is to get a better understanding of the performance

of inter-piconet communication, which will help us in the theoretical evaluation of Bluetooth algorithms.

References

- [1] K. Kar, M. Kodialam, T. V. Lakshman, and L. Tassiulas. Routing for network capacity maximization in energy-constrained ad-hoc networks. In *Mobile Computing and Networking*, pages 673–681, 2003.
- [2] Y. Kawamoto, V. W. Wong, and V. C. Leung. A two-phase scatternet formation protocol for bluetooth wireless personal area networks. In *Proc. IEEE Wireless Commun. and Net. Conf.*, pages 1453–1458, 2003.
- [3] Q. Li, J. Aslam, and D. Rus. On-line power-aware routing in wireless ad-hoc networks. In *Mobile Computing and Networking*, pages 97–107, 2001.
- [4] X. Li and I. Stojmenovic. Partial delaunay triangulation and degree-limited localized bluetooth scatternet formation. In *Proceedings of AD-HOC Networks and Wireless (ADHOC-NOW)*, 2002.
- [5] Y. Liu, M. J. Lee, and T. N. Saadawi. A bluetooth scatternet route structure for multihop ad hoc networks. In *IEEE JSAC*, pages 229–239, 2003.
- [6] T. Nadeem, S. Banerjee, A. Misra, and A. Agrawala. Energy-efficient reliable paths for on-demand routing protocols. In *Proceedings of Sixth IFIP IEEE International Conference on Mobile and Wireless Communication Networks*, 2004.
- [7] E. Pagani, G. P. Rossi, and S. Tebaldi. An on-demand bluetooth scatternet formation algorithm. In *Proc. 1st IFIP Wkg. Conf. Wireless On-demand Net. Sys.*, pages 130–143, 2004.
- [8] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of the IEEE INFOCOM '97*, pages 1405–1413, 1997.
- [9] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of IEEE WMCSA*, 1999.
- [10] C. Petrioli, S. Basagni, and I. Chlamtac. Configuring blues-tars: Multihop scatternet formation for bluetooth networks. In *IEEE Transactions on Computers* 52(6), *Special Issue on Wireless Internet*, pages 779–790, 2003.
- [11] K. Wang, Y. Xu, G. Chen, and Y. Wu. Power-aware on-demand routing protocol for manet. In *Proceedings of the 24th International Conference on Distributed Computing Workshops*, 2004.
- [12] Z. Wang, R. Thomas, and Z. Haas. Bluenet - a new scatternet formation scheme. In *Proceedings of the 35th Hawaii International Conference on System Science (HICSS-35)*, 2002.
- [13] A. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. In *SIAM Journal on Computing*, pages 721–736, 1982.
- [14] G. V. Zaruba, S. Basagni, and I. Chlamtac. Bluetrees - scatternet formation to enable bluetooth based ad hoc networks. In *Proc. IEEE ICC*, 2001.
- [15] X. Zhang and G. F. Riley. Energy-aware on-demand scatternet formation and routing for bluetooth-based wireless sensor networks, 2005.