Learning-based Routing in Mobile Wireless Sensor Networks*

Fatemeh Kazemeyni[1,3], Mario A. Nascimento[2], Ilangko Balasingham[3], Olaf Owe[1], and Einar Broch Johnsen[1]

[1] Department of Informatics, University of Oslo, Norway
{fatemehk,einarj,olaf}@ifi.uio.no
[2] University of Alberta, Canada
mn@cs.ualberta.ca
[3] The Intervention Center, Oslo University Hospital, Institute of Hospital Medicine, University of Oslo, Norway
ilangkob@medisin.uio.no

Abstract. Limited energy supply is a chief concern when dealing with wireless sensor networks (WSNs). Thus, among other issues, routing protocols for WSNs should be designed with the goal of being energy efficient in the first place. For static networks this is already a challenge, given that different domains and application requirements lead to different constraints, and it becomes an even more complex problem when nodes in the WSNs are mobile. In this paper we address this very problem and propose centralized and decentralized routing techniques that are help prolong the nodes lifespan. The main idea is to explore the movement patterns of the nodes, through simple but effective learning, in order to use the most effective and less costly routing paths. Our experiments, using a real dataset, show that our proposed decentralized approach is twice as more energy-efficient than the centralized one, while being only marginally less effective. In addition, both outperform the well-known AODV protocol for ad-hoc mobile networks.

1 Introduction

Wireless sensor networks (WSNs) consist of wireless sensor nodes which collaborate with each other to gather data from the environment and transmit the data to base-stations or sink nodes. Sensor nodes are usually small in size, and consequently they have limited resources, especially regarding their energy supply.

Wireless sensor nodes use routing protocols to communicate with each other and to find the path to transmit the data that is sensed to a designated sink node (or nodes). Message transmission is one of the most energy expensive activities in WSNs. So, an energy efficient routing protocol has an important role reducing the energy consumption of the WSNs. In most of these protocols, the nodes

* Technical Report TR 12-01. January 2012. Dept. of Computing Science. University of Alberta. Canada.

broadcast their messages to all nodes that are within the range of their data transmission. This range is determined by the power used for the transmission.

For example, in the standard AODV protocol [1], each time a source node needs to send a data message, it requests a routing path to the destination. This request is rebroadcasted through the neighbors until a path is found. The resulting path should then be sent to the source node, which uses this path to send the data messages to the destination. The routing paths are stored in routing tables. A node may need a new routing path when it moves to a new location or when the routing path changes due to the movement of any of nodes in the routing path. This kind of routing technique is not always feasible. Especially in large-scale ad-hoc networks, nodes cannot always find and store all the routing paths efficiently because of lack of memory and high energy cost. In ad-hoc wireless sensor networks, nodes can move and change their location. Therefore, they may often enter or exit the signal ranges of each other. Consequently, the previous routing paths may not be valid any more. If nodes move frequently, the cost of finding the routing path increases dramatically. The energy cost is not the only problem of WSNs with moving nodes. The frequent disconnection between the nodes causes increasing the rate of the packet loss.

In this paper, we propose routing protocols which find the most probable routing path for mobile WSNs, considering the frequent movement of nodes. The idea of the proposed protocol is based on the nodes' movement patterns. Each node's movement may have some kind of pattern. For instance, the daily activity of the patients in the different locations of the hospital, may have a traceable pattern. People, e.g., going to or returning from work, or public transit vehicles may also exhibit movement patterns that may be explored. This pattern could be calculated, for every pair of nodes, as the probability of being connected. This probability is used to find the most probable existing routing path, when nodes need to send a message to the sink node.

We introduce centralized and decentralized approaches for our probabilistic routing protocol. In the centralized protocol, the routing paths are computed centrally, by "special" nodes which we call *processing* nodes. Processing nodes have the information of the whole network. Nodes request a processing node for the best and most probable routing path, when they need to send a message. The processing node uses well-known Dijkstra's Shortest Path Algorithm [2] to find the best path. Note that the notion of best is determined as a function of the cost of the edges, i.e., communication links between the nodes, of the network. As we discuss later in the paper, we assign those costs based on the likelihood of existence and reliability of these edges, thus the best route is the one with higher probability of a successful message. In the centralized protocol, nodes should inform the processing node(s) about their situation and should communicate with it to ask for the routing path. These message exchanges may be costly in terms of energy, hence rendering this approach not very efficient. In the decentralized protocol, sensor nodes are responsible for finding the routing path. Each node learns which neighbor node has the best message transmission

history. Then, when a node needs to find a routing path for a message, it chooses the neighbor that has the highest chance of transmitting the message successfully.

Protocol evaluation is usually done through simulation-based tools, such as NS-2, OMNeT+, and extensions such as Castalia [3] and SensorSim [4]. Instead, we use simulation tools based on formal techniques, which provide us an abstract way to model and analyze our protocols. Formal techniques can simulate the behavior of the protocol's model and also prove the correctness of it, by inspecting all reachable states of a system. Usual network simulators cannot prove the correctness of the protocol's properties, because it is practically impossible to exhaustively test the protocol in all the possible situations, but can provide quantitative information about the protocol's behavior.

We develop a formal, executable model of our protocol in rewriting logic [5]. This model is analyzed using Maude [6], which is a formal modeling tool. To have a better analysis of the system, we feed our formal model with real data. By feeding a formal model of a protocol with real mobile traces, we could have a more reliable analysis of the model, in addition to taking advantages of the formal analysis. The protocol's analysis includes studying the performance of the protocols and checking the correctness of the protocol's behavior.

This paper is structured as follows. Section 2 review briefly some related research. Section 3 presents our main contribution, the centralized and decentralized versions of the propose routing protocol. Section 4 discusses the modeling of the proposed protocols and Section 5 describes the case study which is used to analyze the protocol and results of the validation of the model. Finally, Section 6 concludes the paper.

2 Related Work

The efficiency of routing protocols are discussed in several recent research studies. The authors of [7] provide an overview of features and mechanisms of energy-aware routing protocols. Studies, such as [8] and [9], aim at reducing the power consumption of the routing process, using cluster head election and multi-hop transmission. Another energy aware localized routing protocol is introduced in [10], and is based on distance-based power metrics. There are a few works which consider the frequent movement of the nodes and its role on the successful message transmission and the energy efficiency of the network. Q-probabilistic routing [11] is a geographical routing algorithm which uses Bayesian algorithm to to find the routing path. This protocol achieves a trade-off between the network's energy usage and the number of message retransmission. Parametric Probabilistic Sensor Network Routing Protocol [12] is another protocol which uses probabilities, based on the nodes' distances, for message retransmission. The PROPHET protocol [13] introduces the delivery predictability factor to choose the messages which are sent to the other nodes, in networks which do not guarantee fully connected networks. In our protocol, we do not need to know the position or the distance of the nodes. We maximize the successful message transmission rate (success rate) of the protocol in an energy efficient way, considering the proba-

bility of the link existence and the link reliability. In the decentralized version of our protocol, we also use the delivery history as a means to predict the success rate of the protocol.

Formal methods are much less explored to analyze the WSNs, but recently start to appear. The TinyOS operating system has been modeled as a hybrid automaton [14] and the UPPAAL tool which has been used to model the LMAC protocol [15] and also to verify the temporal configuration parameters of radio communication [16]. Ölveczky and Thorvaldsen show that Maude is a well-suited modeling tool for WSNs [17]. Maude is used to model and analyze the efficiency of the OGCD protocol [17] and the LMST protocol [18]. A process algebra is introduces specifically for active sensor processes such as sensing [19]. We follow this line of research and use Maude as a tool to develop a routing protocol for WSNs.

3 The Proposed Routing Protocol

This section explains our proposed routing protocols for WSNs, based on the nodes' movement patterns and the probability associated to the communication links. Assume that we have a WSN with nodes which move frequently. In this network, a sensor node has some information, e.g., answers to a query, which need to be sent to the sink node. The problem is how to find the best routing path between the sensor node and the sink node. In this work we consider the best routing path to be the one which is the most reliable and has the higher probability of existence during the message transmission. (We assume that nodes use the same amount of energy to transmit the messages to all their neighbors.) Our contribution is in the design of protocols that allow such best routing path to be obtained at an overall low-energy cost. We proposed two protocols, one centralized and another, more robust and flexible, which is decentralized. In the centralized protocol, the routing decisions are made one or more so-called proceessing nodes. In the decentralized protocol, routing decisions are made autonomously by the nodes themselves. We note that in some situations, e.g., when processing are often not reachable, the decentralized protocol is the only feasible option. In the following, both the centralized and decentralized versions of the proposed protocol are discussed.

3.1 The Centralized Approach

We assume that a WSN is a graph $G = (S, L)$, which $S = \{n_1, n_2, ..., n_N\}$ is the set of nodes and $L = \{l_{i,j}\ \forall i, j \in S$ s.t. n_i is within radio range of $n_j\}$ is the set of the edges (communication links) in G. Our main problem is to find the path that is most likely to deliver a message between two nodes. In order to that we need to keep up-to-date the costs of G's edges updated, and this costs energy. Each link in the graph has a weight that could be based on the cost of the link and other possible properties of the link and the nodes. By having these weights, we use Dijkstra's algorithm to choose the routing path.

Each link $l_{i,j} \in L$ has a probability which we call the *existence probability* or $EP_{l_{i,j}}$. This probability could be calculated using the history of the existence of the network links. We note that if two nodes are not within radio range from each other, a link between them effectively does not exist. Assume that an initial probability is defined for each link $l_{i,j}$, between two nodes i and j. The rate that two nodes meet during a period of time could increase or decrease the initial probability according to a *learning factor* f. Hence we update $EP_{l_{i,j}}$ as $EP_{l_{i,j}} \times (1 - f) + f$ if two nodes i and j meet within a predefined time interval T, or as $EP_{l_{i,j}} \times (1 - f)$ otherwise [20].

The initial probability of the links and the time interval T could be defined specifically for each network which is being studied, based on its properties. Some possibilities could range from a conservative one, e.g., $EP_{l_{i,j}} = 0 \; \forall i, j \in S$, to a more optimistic one $EP_{l_{i,j}} = N \times (\pi\omega^2/A) \; \forall i, j \in S$, where ω and A are the radius of the radio range and the total area under study, respectively, that is, assuming all nodes are uniformly distributed within the area of study.

We use the combination of the existence probability of the link, in addition to its reliability, as the weight of each link. The reliability of the link is important, as well as its existence. If a link exists but it is not reliable, messages could be lost. The reliability of a link could depend on different factors. For example, the physical nature of the environment between two nodes could change the reliability of their link in time. For simplicity we consider the average historical reliability of the link $l_{i,j}$ as a probability $RP_{l_{i,j}}$, which is obtained from the network's history based on the channel quality between every pair of node i and j.

The probability that matters most, and the one that is assigned as cost to each network link $l_{i,j}$, denoted as $P_{l_{i,j}}$ is the combination of these two independent probabilities: $RP_{l_{i,j}}$ and $EP_{l_{i,j}}$, namely $P_{l_{i,j}} = RP_{l_{i,j}} \times EP_{l_{i,j}}$.

We use Dijkstra's algorithm with the network's links cost being the probability $P_{l_{i,j}}$ to find the most probable path between two nodes in the network. This is a centralized process which is performed in the processing node. When sensor nodes move, they inform the processing node about their new situation. Each time that a node needs to find a routing path to another node, it sends a message to the processing node and asks for the routing path. The processing node sends a reply message to the node which includes the result path. These message passings between the processing node and the sensor nodes cost energy, in addition, the processing nodes may not be always reachable. This lead to the need of a decentralized algorithm.

3.2 The Decentralized Approach

In the decentralized version of our proposed routing protocol, each node decides locally which node should be chosen as the next node in the routing path. Nodes just have the information of themselves and their neighbors, not the the total information of the network. There are some methods which can be used in these situations, such as the "ant colony" [21] (the same as what the AODV routing protocol uses). But these methods frequently rebroadcast messages asking for

routing paths, which makes them less energy-inefficient. In the proposed protocol, each node collects information from previous message transmissions in order to learn how to to predict the best route. We note that nodes do not have all the information ideally needed to decide about the total path, but use what is available to them, as per our following discussion, to pass the message to the node which is in the most probable path to the destination. We use acknowledgment (ACK) messages to infer the probability of existence and the reliability of the routing paths to each destination node. The source node s can calculate the acknowledgment probability $AP_{s,e,d}$ for the messages to the destination d through the neighbor node e, as $AP_{s,e,d} = \frac{ackNum^{e}_{s,d}}{msgNum^{e}_{s,d}}$. Where, $msgNum^{e}_{s,d}$ and $ackNum^{e}_{s,d}$ are the number of the sent data messages from node s to d and the number of the ack messages from node d to s, respectively, in both cases through the neighbor node e. After a sufficiently large learning period, this ratio can give us an estimate about the probability of the existence of a routing path is. Consequently, it shows the capability of a successful message transmission through that path. The node s updates the value of $AP_{s,e,d}$, each time it receives an ACK message. The source node s can then calculate the probability os success delivery $P_{s,e,d}$ for each neighbor node e, regarding the destination node d, using $P_{s,e,d} = RP_{l_{s,e}} + AP_{s,e,d}$.

After computing the value of $P_{s,e,d}$ for all the current neighbor nodes e, the source node s chooses the neighbor that yields the highest probability and sends the message to it. This process is repeated in each node which receives the message, until the message is received to the destination, or it is eventually lost, e.g., due to a link failure.

4 Modeling

We utilized Maude [6], as a formal modeling tool, to implement and analyze our proposed routing protocols. Maude provides an executable model, in addition to different tools for testing and validating the model. It is also able to run the model through one path of the state space like a simulator or go through all the reachable states of the model to search for the system failure. Maude could prove if the properties of the system hold and if the model works correctly regarding to these properties. To gather some quantitative data from the protocol functionality and performance, we ran simulations, with each simulation lasting for a predefined number of time units that is captured by rewrite steps, i.e. one time unit corresponds to one rewrite step.

A formal model is an abstract presentation of a system (or a protocol). This abstraction helps us to overview the protocol and find its failures more easily. On the other hand, the abstraction hides some details of the reality from the model. In a network protocol, which has many entries where its behavior is not deterministically predictable or under control, abstractions could affect the result of the model. The goal is to make the result of the modeling more reliable and more real, as much as possible. We combine the formal model with the reality, by feeding the real data as entries to the initial configuration of the model. To the

best of our knowledge, real datasets have not been combined with formal models in other works. In our initial configuration of the model, the movements of the nodes and the time and length of the movements are provided from a public available dataset. In the following we explain our case study and the results of running the model after applying the proposed protocols on this data.

In continuation of this section, we define a formal model of the proposed routing protocols in rewriting logic [5]. A *system configuration* is a multiset of objects and messages inside curly brackets. Following rewriting logic (RL) conventions, whitespace denotes the associative and commutative constructor for configurations. The term $\langle$`O : Node | Attributes`$\rangle$ denotes a `Node` object, where `O` is the object identifier, and `Attributes` a set of attributes of the form `Attr : X` where `Attr` is the attribute name and `X` the associated value. RL extends algebraic specification techniques with transition rules: The dynamic behavior of a system is captured by rewrite rules supplementing the equations which define the term language. From a computational viewpoint, a rewrite rule $t \longrightarrow t'$ may be interpreted as a *local transition rule* allowing an instance of the pattern t to evolve into the corresponding instance of the pattern t'.

In the sequel, we explain the rules and equations modeling wireless sensor networks, node movements, message passing, and the centralized and decentralized routing protocol, using the rewriting logic tool Maude [6].

4.1 Unicast and Broadcast

Unicast messages have the form

```
(M from O P to O')
```

where `M` is the message body (possibly with parameters), `O` the source, `O'` the destination, and `P` the sending power used. *Multicasting* is modeled by allowing a set of destinations and equations which expand the destination set:

```
eq (M from O L P to noneOids) = none .
eq (M from O L P to O'; Os)
 = (M from O L P to O') (M from O L P to Os).
```

Here, `Os` denotes a set of object identities (with ";" as multiset constructor). *Wireless broadcasting* uses messages

```
(M from O L P to all)
```

where `all` is a constructor indicating that the message is sent to *all nodes within range*. We want to model lossy channels, in order to have a more realistic model. In a lossy channel, data messages could be lost. To capture this behavior, we use a conditional rule that removes the messages with a predefined probability.

```
ceq (M from O L P to O') ⟨ O' :Node | A ⟩ = ⟨ O' : Node | A ⟩ if sampleBernoli(P).
```

Here, the messages are lost with the probability of `P` and `sampleBernoli(P)` is a function that returns true in the `P` percent of times. In all the rules and equations, the `CONF` is the systems configuration and the `savedPower` attribute shows the amount of the node's remaining energy. After each activity such as

sending and receiving a message, this value is decreased depending on how much power is consumed for that activity. We use `Pmin(P)` and `Pmax(P)` to refer to the predefined amount of energy that nodes consume when they perform short-range and wide-range communication, based on their power capability `P`. A time variable `T` is added to the global configuration of the system to resolve all nondeterminism. Execution marks are added to the left-hand sides of all rewrite rules, as well as *scheduled* execution marks to their right-hand side, in order to make the new subconfiguration active at a later time, after a random interval of time has passed, following an exponential probability distribution with some fixed rate parameter, in our case 0.1. The random length of this interval is generated using an operation denoted `sampleExpWithRate`, in Maude. Note that, in the current implementation, the rates corresponding to the exponentially distributed waiting times of all scheduled execution marks are equal to 0.1. However, these rates can be given different values for each sensor, to simulate different sensor processor speeds.

4.2 Movement Messages

The rows of the dataset that we used to feed to our model, specify the Identities of two nodes which meet each other, in addition to the time and the length of their meeting, in the form of `[O O′ T′ EndT]`. In our model, each row of dataset is captured by a `movemsgT O′ T′ EndT to O` message which notices node `O` to move towards node `O′` at time `T′` for the length of `EndT`. In the initial configuration, all the `movemsgT` messages are generated, based on the list of all rows in the dataset, utilizing the `multimoveT` equation. In the `Move` rule, when the time `T` reaches the time `T′`, node `O` moves and send a unicast message `SingleHELLO` to the node `O′`, including all the required information. The `infoREP` attribute stores all the information of the node's neighbors to calculate the routing path existence probabilities. The `LinkR` is an equation that returns the reliability which is assigned to each node's message transmission. If the time is not right for the movement when the `movemsgT` is captured, the `Wait to Move` equation reschedules it for another time.

The `SingleHELLO` message is received by the `neighbor send info1` and `neighbor send info2` rules. In `neighbor send info1` rule, node `O′` receives the message from `O` for the first time. It sends an acknowledgment message `HelloACK` to the moving node, including the same information as in the `SingelHELLO` message. It also updates its neighbors' information. Each row in the `neighbor` attribute stores the neighbor id `ID`, its link reliability `F`, the power which is needed to communicate with it `FPOW`, its link existence probability `EP`, and the time until then this neighbor will be around the node `EndT+T`. The `INITPROB` equation returns a predefined value as an initial value. The `inFList(FRT,O)` equation checks for the existence of an element in the list `FRT` which is related to the node `O`. The equation `rmListNat(LLN, O, nil)` removes all the elements related to node`O` from `LLN`.

In the `neighbor send Info2` rule, nodes has previously met and need to update the previous information and probabilities. Here, the `FindRow` equation

find the related rows and changeTable change them. When a node receive a HelloACK message, it updates its information using what is received by this message. In the node get neighbors1 equation, node O does not have previous information of node O′, despite of the node get neighbors2 rules. Node O also sends a INFOtoPN message to the processing node to send the required information to the processing node PN. In the decentralized protocol, these two previous rules are replaced by the dc node get neighbors1 and dc node get neighbors2, which does not send any information to PN. When PN receive this message, it updates its nodesInfo attribute. This attribute contains the same information as the infoREP attribute of the node O, in addition to the identifier O attached to it. Here, we also have two rules PN get info1 and PN get info2 for two cases that PN has or has not previous information about the connection between O and O′. The remove equation deletes the rows related to communication between these two nodes.

```
op multimoveT_ :ListListNat → Msg [ctor] .
eq multimoveT nil = none .
eq multimoveT ( ML LL ) =
(movemsgT second(ML) third(ML) (forth(ML)-third(ML)) to first(ML))(multimoveT LL).

crl [Move] :{ ( movemsgT O' T' EndT to O )
⟨O :Node | id: ID, power: P, infoREP: LLN,savedPower: SPOW, A⟩ CONF execute(O) T }
⟶
 {(SingleHELLO from O P F Pmin(P) EndT LLN to O')
⟨O :Node | id: ID, power: P, infoREP: LLN,savedPower: ( SPOW-Pmin(P)) , A⟩
CONF [T +sampleExpWithRate(0.1), execute(O)] T }
if T > T' ∧ F :=LinkR(O).

crl [Wait_to_Move]{ ( movemsgT O' T' EndT to O )
⟨O :Node | id: ID, A⟩ CONF execute(O) T }
⟶
{ ( movemsgT O' T' EndT to O )
⟨O :Node | id: ID , A⟩ CONF [T +sampleExpWithRate(0.1), execute(O)] T }
if T < T' .


crl [neighbor_send_Info1] :
{( SingleHELLO from O P F FPOW EndT LLN' to O')
 ⟨O' :Node | id: ID',power: P' ,savedPower: SPOW' , infoREP: LLN,
 neighbors: FRT' , A' ⟩ execute(O') T CONF }
⟶
 { ( HelloACK from O' P' F' Pmin(P') EndT LLN to O )
⟨O' :Node | id: ID',power:P',savedPower:( SPOW'-Pmin(P')),infoREP: ( LLN' LLN1 ),
neighbors: ([O#F#FPOW#EP#(EndT+T)]FRT') , A' ⟩
[T +sampleExpWithRate(0.1), execute(O')] T CONF }
if F' :=LinkR(O) ∧ EP :=INITPROB ∧
inFList(FRT',O) = false ∧ LLN1 :=rmListNat(LLN, O, nil) .


crl [neighbor_send_Info2]:
{(SingleHELLO from O P F FPOW EndT LLN' to O')
 ⟨O':Node | id:ID',power:P',savedPower:SPOW',infoREP:LLN,neighbors:FRT,A' ⟩
 execute(O') T CONF}
⟶
{(HelloACK from O' P' F' float(Pmin(P'))EndT LLN to O)
⟨O':Node | id:ID',power:P',savedPower:(SPOW'-Pmin(P')),infoREP:(LLN'LLN1),
```

```
neighbors:FRT2, A' ⟩
[T+sampleExpWithRate(0.1), execute(O')] T CONF }
if inFList(FRT,O) = true ∧ MLF:=FindRow(FRT,O) ∧
F':=LinkR2(ID') ∧ EP:=EProb(T,fifth(MLF),forth(MLF)) ∧
FRT2:=changeTable(FRT,[O#F#FPOW#EP#(EndT+T)],nil)
∧ LLN1:=rmListNat(LLN,O,nil) .


crl [node_get_neighbors1]:
{(HelloACK from O' P' F FPOW EndT LLN' to O )
⟨O:Node | id:ID,power:P,savedPower:
SPOW,neighbors:FRT,infoREP:LLN,A⟩ execute(O) T CONF }
⟶
{(INFOtoPN [O'#F#FPOW#EP#EndT] from O to PN)
 ⟨O:Node | id:ID,power:P,savedPower:(SPOW-Prec(P)),neighbors:
 ([O'#F#FPOW#EP#(EndT+T)] FRT),infoREP:(LLN'LLN1),A⟩
 [T+sampleExpWithRate(0.1),execute(O)] T CONF }
if EP:=INITPROB ∧ inFList(FRT,O') = false ∧
LLN1:=rmListNat(LLN,O,nil).

crl [node_get_neighbors2]:
{(HelloACK from O' X' Y' P' F FPOW EndT LLN' to O )
⟨O:Node | id:ID,xLoc:X,yLoc:Y,power:P,savedPower:SPOW,
neighbors:FRT,infoREP:LLN,A⟩ execute(O) T CONF }
⟶
{(INFOtoPN [O'#F#FPOW#EP#EndT] from O to PN)
⟨O:Node | id:ID,xLoc:X,yLoc:Y,power:P,savedPower:(SPOW-Prec(P)),
neighbors:([ID'#F#FPOW#EP#(EndT+T)]FRT2),infoREP:(LLN'LLN1),A⟩
[T+sampleExpWithRate(0.1),execute(O)] T CONF }
if inFList(FRT,O') = true ∧ FRT2:=remove(FRT,O',nil)∧
MLF:=FindRow(FRT,O')∧EP:=EProb(T,fifth(MLF),forth(MLF))
∧ LLN1:=rmListNat(LLN,ID,nil).

crl [dc-node_get_neighbors1] :
{ (HelloACK from O' P' F FPOW EndT LLN' to O )
⟨O:Node | id:ID,power:P,savedPower:SPOW,neighbors:FRT,infoREP:LLN,A⟩
execute(O) T CONF }
⟶
{⟨O:Node | id:ID,power:P,savedPower:(SPOW-Prec(P)),
neighbors:([O'#F#FPOW#EP#(EndT+T)]FRT),infoREP:(LLN'LLN1),A⟩
 [T+sampleExpWithRate(0.1),execute(O)] T CONF }
if EP:=INITPROB ∧ inFList(FRT,O') = false ∧ LLN1:=rmListNat(LLN,O,nil).

crl [dc-node_get_neighbors2] :
{(HelloACK from O' P' F FPOW EndT LLN' to O )
⟨O:Node | id:ID,power:P,savedPower:SPOW,neighbors:FRT,infoREP:LLN,A⟩
execute(O) T CONF }
⟶
{⟨O:Node | id:ID,power:P,savedPower:(SPOW-Prec(P)),neighbors:
([float(ID')#F#FPOW#EP#(EndT+T)]FRT2),infoREP:(LLN'LLN1),A⟩
[T+sampleExpWithRate(0.1),execute(O)] T CONF }
if inFList(FRT,float(ID')) = true ∧ FRT2:=remove(FRT,float(ID'),nil)∧
MLF:=FindRow(FRT,O')∧ EP:=EProb(T,fifth(MLF),forth(MLF))∧ LLN1:=rmListNat(LLN,O,nil).

crl [PN-get-info1]:
{(INFOtoPN [O'#F#FPOW#EP#LAST] from O to PN )
⟨PN:PNode | id:ID,nodesInfo:FRT,A' ⟩ execute(PN) T CONF }
⟶
{⟨ PN:PNode | id:ID,nodesInfo:([O#O'#F#FPOW#EP#(LAST+T)]FRT),A' ⟩
[T+sampleExpWithRate(0.1),execute(PN)] T CONF }
```

```
if inFList2(FRT,O,O') = false .

crl [PN-get-info2]:
{(INFOtoPN [O'#F#FPOW#EP#LAST] from O to PN)
⟨ PN:PNode | id:ID',nodesInfo:FRT,A' ⟩ execute(PN) T CONF }
⟶
{ ⟨ PN:PNode | id:ID',nodesInfo:([O#O'#F#FPOW#EP#(LAST+T)]FRT2),A' ⟩
⟨ O:Node | id:ID,A ⟩ [T+sampleExpWithRate(0.1),execute(PN)] T CONF }
if inFList2(FRT,O,O') = true ∧ FRT2:=remove(FRT,O,O',nil).
```

During the run of the model, data messages which are related to the sensed data are generated at random times. This process is captured by the `Data-Msg` equation. This equation generates the `DCDATAMSG` messages for the model of the decentralized protocol and the `AskRoute` messages for the centralized protocol. The number of the messages which are generated for each node `O` is equal to the number of the messages in its message queue `MsgQ`.

```
ceq [Data-Msg] :
{⟨ O:Node | id: ID,MsgQ: Y,savedPower: P,power: POW,A ⟩
execute(O) T CONF }
=
{⟨ O:Node | id:ID,MsgQ: (Y-1),power: POW,savedPower: (P-Pmax(POW)),A ⟩
[T +sampleExpWithRate(0.1), execute(O)] T CONF
(DCDATAMSG O "sink") ***OR ***(AskRoute "sink" T from O to PN ) }
if ( Y > 0 ) .
```

4.3 The Centralized Model

In this section, the rules and the equations of the centralized protocol are presented. Node `S` asks for a routing path to node `D` from `PN`, by sending the `AskRoute` message. In `PN-rec-AskRoute` rule, `PN` finds the most probable routing path and send it to `S` by a `RouteReply` message. The processing node uses Dijkstra algorithm to find the routing path. Our Dijkstra model is based on the model in [22]. But we modified this model to be suited to our protocol. Mainly, we changed the `weight` equation which computes the weight of the links between nodes (or graph edges), according to Section 3.1. The Dijkstra algorithm in [22], returns a numerical value which is the cost of the shortest path. Since we need the actual routing path, we defined the `DijkstraP` equation which returns the selected node's links by the Dijkstra algorithm. The `findPath` is an equation which applied on top of the `DijkstraP` equation and returns the total path between the nodes `S` and `D`, based on the links which are selected by the `DijkstraP` equation, and`AllIDs(FRT)` returns all the nodes' identifiers in `FRT`. More information about the Dijkstra model could be found in [22].

```
crl [PN-rec-AskRoute]:
{(AskRoute D LastT from S to PN )
⟨ S:Node | id:X,index:N,power:POW,savedPower:P,Ac ⟩ ⟨ D:Node | id:Y,A ⟩
⟨ PN:PNode | id:ID,nodesInfo:FRT,A' ⟩ execute(PN) T CONF }
      ⟶
{(RouteReply D PATH LastT from PN to S )
⟨ S:Node | id:X,index:(N+1),power:POW,savedPower:(P-Pmax(POW)),Ac ⟩ ⟨ D:Node | id:Y,A ⟩
⟨ PN:PNode | id:ID,nodesInfo:FRT,A' ⟩ [T+sampleExpWithRate(0.1),execute(PN)] T CONF }
if PATH:=findPath(X,X,Y,DijkstraP(FRT,X,AllIDs(FRT),nil),nil).
```

```
op findPath :ListFloat Float Float ListListFloat ListFloat → ListFloat.
ceq findPath(loc#LF,loc1,loc',LLF,LF1) =
findPath(loc,loc1,loc',LLF,LF1) # findPath(LF,loc1,loc',LLF,LF1)
if LF ≠ nil ∧ LF \neq loc'.
eq findPath(loc1#loc',loc1,loc',LLF,LF1) = loc1#loc'.
ceq findPath(loc,loc1,loc',LLF,LF1) = if (LF ≠ nil) then
findPath(LF,loc1,loc',LLF,LF1#loc) else nil fi
if LF :=getConnected(LLF,loc) ∧ loc ≠ loc'∧ inLF(LF,loc') = false.
ceq findPath(loc,loc1,loc',LLF,LF1) = loc#loc'
if LF :=getConnected(LLF,loc)∧ loc ≠ loc' ∧ inLF(LF,loc') = true.
ceq findPath(loc,loc1,loc',LLF,LF1) = LF1#loc' if (loc = loc').

op DijkstraP :ListListFloat Float ListFloat ListListFloat → ListListFloat .
op DijkstraP :ListListFloat Map{Float, Float} ListFloat ListListFloat → ListListFloat .
eq DijkstraP(nil,loc,OidL,LLF') = nil.
ceq DijkstraP(LLF,loc,OidL,LLF') = nil if inLF(OidL,loc) = false.
eq DijkstraP(LLF,loc,OidL#loc#OidL',LLF') = DijkstraP(LLF,loc ↦0.0,loc#OidL#OidL',LLF').
eq DijkstraP(LLF,(loc ↦0.0,MOF),nil,LLF') = LLF'.
ceq DijkstraP(LLF,MOF,loc#OidL,LLF') = DijkstraP(LLF,MOF',sort(OidL,MOF'),LLF1)
if MOF':=update(LLF,MOF,loc,getConnected(LLF,loc)) ∧
LLF1 :=updateNL(LLF,MOF,loc,getConnected(LLF,loc),LLF').

op sort :ListFloat Map{Float, Float} → ListFloat .
ceq sort(OidL#F#OidL'#F'#OidL",MOF) = ␣sort(OidL#F'#OidL'#F#OidL",MOF)
if minor(MOF[F'],MOF[F]).
eq sort(OidL,MOF) = OidL[owise].

op getConnected :ListListFloat Float → ListFloat .
eq getConnected(nil,F) = nil.
ceq getConnected(MLFLLF,F) = if first(MLF) = F then (getConnected(LLF,F)#F1)
else getConnected(LLF,F) fi
if F1:=second(MLF).

op minor :[Float] [Float] → Bool .
eq minor(F, undefined) = true .
eq minor(undefined, F) = false .
eq minor(F, F') = F < F' .
eq minor(FF, FF') = false [owise] .


op update :ListListFloat Map{Float, Float} Float ListFloat → Map{Float, Float} .
eq update(LLF, MOF, loc, nil) = MOF .
ceq update(LLF,MOF,loc,loc'#OidL) = if minor(add(MOF[loc],F),MOF[loc']) then
update(LLF,insert(loc',add(MOF[loc],F),MOF),loc,OidL) else update(LLF,MOF,loc,OidL) fi
if F :=weight(LLF,loc,loc').

op updateNL :ListListFloat Map{Float,Float} Float ListFloat ListListFloat →
ListListFloat .
eq updateNL(LLF,MOF,loc,nil,LLF') = LLF'.
ceq updateNL(LLF,MOF,loc,loc'#OidL,LLF') = if minor(add(MOF[loc],F),MOF[loc'])
then updateNL(LLF,insert(loc',add(MOF[loc],F),MOF),loc,OidL,[loc#loc']LLF')
else updateNL(LLF,MOF,loc,OidL,LLF') fi
if F :=weight(LLF,loc,loc').

op weight :ListListFloat Float Float → Float .
eq weight(nil,F,F') = 0.0 .
ceq weight(MLF LLF,F,F') = (third(MLF)*fifth(MLF)) if first(MLF) = F ∧ second(MLF) = F'.
ceq weight(MLF LLF,F,F') = weight(LLF,F,F') if first(MLF) ≠ F .
ceq weight(MLF LLF,F,F') = weight(LLF,F,F') if second(MLF) ≠ F' .
```

In the node-rec-RouteReply rule, node S which receives the RouteReply message, retrieves the next node F from the selected routing path, to send a data message DATAMSG to it, including the rest of the selected routing path LF. It only happens if at that time, node F is in the neighborhood. Otherwise, no message is generated (the node-rec-RouteReply-timeout rule). The node-rec-RouteReply-no path rule captures the situations that PN could not find any routing path. The node-rec-DATAMSG and node-rec-DATAMSG-timeout rules retransmit the data message, similarly. Finally, in the dest-rec-DATAMSG rule, the destination node receives the data message.

```
crl [node-rec-RouteReply] :
{(RouteReply D (ID#F#LF) LastT from PN to S )
⟨ S:Node | id:ID,reqid:N,NumOfSentMsg:N1,savedPower:P,power:POW,neighbors:LLF,A ⟩
execute(S) T CONF }
⟶
{(DATAMSG (N+1) S D LF LastT from S to object(F))
⟨ S:Node | id:ID,reqid:(N+1),NumOfSentMsg:(N1+1),power:POW,savedPower:
(P-(Prec(POW)+Pmin(POW))),neighbors:LLF,A ⟩
[T+sampleExpWithRate(0.1),execute(S)] T CONF }
if LastT < fifth(FindRow(F,LLF)).

crl [node-rec-RouteReply-timeout] :
{(RouteReply D (ID#F#LF) LastT from PN to S )
⟨ S:Node | id:ID,reqid:N,savedPower:P,power:POW,neighbors:LLF,A ⟩ execute(S )T CONF }
⟶
{ ⟨ S:Node | id:ID,reqid:N,power:POW,savedPower:(P-Prec(POW)),neighbors:LLF,A ⟩
[T+sampleExpWithRate(0.1),execute(S)] T CONF }
if LastT > fifth(FindRow(F,LLF)).

rl[node-rec-RouteReply-no path]:
{(RouteReply D nil LastT from PN to S ) ⟨ S:Node | id:ID,reqid:N,A ⟩ execute(S) T CONF }
⟶
{ ⟨ S:Node | id:ID,reqid:N,A ⟩ [T+sampleExpWithRate(0.1),execute(S)] T CONF } .

crl [node-rec-DATAMSG] :
{(DATAMSG MsgID S D PATH LastT from O1 to O )
⟨ O:Node | id:ID,savedPower:P,power:POW,neighbors:LLF,A ⟩
⟨ O':Node | id:ID',A' ⟩ execute(O) T CONF }
      ⟶
{(DATAMSG MsgID S D LF LastT from O to O' )
⟨ O:Node | id:ID,savedPower:(P-(Prec(POW)+Pmin(POW))),power:POW,neighbors:LLF,A ⟩
⟨ O':Node | id:ID',A' ⟩ [T+sampleExpWithRate(0.1),execute(O)] T CONF }
if (F#LF):=PATH ∧ F = ID' ∧ D ≠ O ∧ LastT < fifth(FindRow(F,LLF)).

crl [node-rec-DATAMSG-timeout] :
{(DATAMSG MsgID S D PATH LastT from O1 to O )
⟨ O:Node | id:ID,savedPower:P,power:POW,neighbors:LLF,A ⟩
⟨ O':Node | id:ID',A' ⟩ execute(O) T CONF }
⟶
{⟨ O:Node | id:ID,savedPower:(P-(Prec(POW)+Pmin(POW))),power:POW,neighbors:LLF,A ⟩
⟨ O':Node | id:ID',A' ⟩ [T+sampleExpWithRate(0.1),execute(O)] T CONF }
if(F#LF):=PATH ∧ F = ID' ∧ D ≠ O ∧ LastT > fifth(FindRow(F,LLF)).

rl [dest-rec-DATAMSG] :
{(DATAMSG MsgID S D PATH LastT from O1 to D )
⟨ D:Node | id:ID,NumOfRecMsg:N,savedPower:P,power:POW,A ⟩ execute(D) T CONF }
⟶
```

```
{⟨D:Node | id:ID,NumOfRecMsg:(N+1),power:POW,savedPower:(P-Prec(POW)),A⟩
[T+sampleExpWithRate(0.1),execute(D)] T CONF } .
```

4.4 The Decentralized Model

This section describes the model of our decentralized routing protocol. Each node O which needs to send a data message, generates a DCDATAMSG message and increase the number of the sent messages in its attributes value. The start-DC1 and start-DC1 rules are used to transmit the DCDATAMSG to the destination. The first rule captures the situations that there is no previous information about the required routing path in the routing table. Otherwise, the second rule is applied. The routingTable attribute is used in the decentralized protocol to decide about the next routing node. It includes the neighbor's ID, the destination node's ID, the number of the sent messages to the destination, the number of the received acknowledgment messages from the destination and the last node which is chosen to retransmit a message to this destination. The BestREP equation chooses the best node with the highest link probability in the neighborhood. The getREP equation retrieves the acknowledgment probability of the neighbor node ID1 and destination node ID'.When the destination node receives the DCDATAMSG message, it sends an acknowledgment message DCACK to the nodes which transmit the DCDATAMSG message.

```
rl [start-DC] :
(DCDATAMSG O D )⟨O:Node|id:ID,reqid:MsgID,NumOfSentMsg:N,A⟩
⟶
(DCDATAMSG (MsgID+1) noneOids D from O to O)
⟨O:Node | id:ID,reqid:(MsgID+1),NumOfSentMsg:(N+1),A⟩ .

crl [start-DC1] :
{(DCDATAMSG MsgID PL D from O1 to O )
⟨O:Node | id:ID,routingTable:LLN,infoREP:LLN',neighbors:LLF,savedPower:P,
power:POW,NumOfSentMsg:N,A⟩ ⟨D:Node|id:ID',A'⟩ execute(O) T CONF }
⟶
if Neighbour(LLF,D) = false then
if Id ≠ 0 then
{(DCDATAMSG MsgID (PL;O) D from O to O')
⟨O:Node | id:ID,routingTable:([ID ID' 1 0 Id]LLN),infoREP:LLN',
neighbors:LLF,savedPower:(P-Pmin(POW)),power:POW,NumOfSentMsg:N,A⟩
⟨D:Node | id:ID',A'⟩ [T+sampleExpWithRate(0.1),execute(O)] T CONF }
else{⟨O:Node | id:ID,routingTable:LLN,infoREP:LLN',neighbors:LLF, savedPower:P,
power:POW,NumOfSentMsg:N,A⟩ ⟨D:Node | id:ID',A'⟩
 [T+sampleExpWithRate(0.1),execute(O)] T CONF } fi
else {(DCDATAMSG MsgID (PL;O) D from O to D)
⟨O:Node | id:ID,routingTable:([ID ID' 1 0 ID']LLN),infoREP:LLN',neighbors:LLF,
savedPower:(P-Pmin(POW)),power:POW,NumOfSentMsg:N,A⟩
⟨D:Node|id:ID',A'⟩ [T+sampleExpWithRate(0.1),execute(O)] T CONF } fi
if inLLN(LLN,ID') = false ∧ IDc:= NodeID(O1) ∧ Id:=BestREP(LLF,LLN',ID',ID,IDc,T,0,0.0)
∧O':=object(Id).

crl [start-DC2] :
{(DCDATAMSG MsgID PL D from O1 to O )
⟨O:Node | id:ID,routingTable:LLN,infoREP:LLN',neighbors:LLF,savedPower:P,
power:POW,NumOfSentMsg:N,A⟩ ⟨D:Node | id:ID',A'⟩ execute(O) T CONF }
⟶
```

```
if Neighbour(LLF,D) = false then
if Id ≠ 0 then
{(DCDATAMSG MsgID (PL;O) D from O to O' )
⟨O:Node | id:ID,routingTable:([ID ID'(X+1) Y Id]LLN1),infoREP:LLN'
,neighbors:LLF,savedPower:(P-Pmin(POW)),power:POW,NumOfSentMsg:N,A⟩
⟨D:Node | id:ID',A'⟩ [T+sampleExpWithRate(0.1),execute(O)] T CONF }
else {⟨O:Node | id:ID,routingTable:(LLN[ID ID' X Y Z]LLN1),infoREP:LLN',
neighbors:LLF,savedPower:P,power:POW,NumOfSentMsg:N,A⟩
⟨D:Node | id:ID',A'⟩ [T+sampleExpWithRate(0.1),execute(O)]TCONF } fi
else {(DCDATAMSG MsgID (PL;O) D from O to D )
⟨O:Node | id:ID,routingTable:(LLN[ID ID'(X+1) Y ID']LLN1),infoREP:LLN',
neighbors:LLF,savedPower:(P-Pmin(POW)),power:POW,NumOfSentMsg:N,A⟩
⟨D:Node | id:ID',A'⟩ [T+sampleExpWithRate(0.1),execute(O)] T CONF } fi
if IDc:=NodeID(O1) ∧ Id:=BestREP(LLF,LLN',ID',ID,IDc,T,0,0.0)∧ O':=object(Id).

op BestREP :ListListFloat ListListNat Nat Nat Nat Float Nat Float → Nat .
eq BestREP(nil,LLN,ID',ID,IDc,T,N,F) = N .
ceq BestREP(MLF LLF,LLN,ID',ID,IDc,T,N,F) = if T < fifth(MLF) then
if F1 > F then BestREP(LLF,LLN,ID',ID,IDc,T,ID1,F1)
else BestREP(LLF,LLN,ID',ID,IDc,T,N,F) fi
else BestREP(LLF,LLN,ID',ID,IDc,T,N,F) fi
if ID1:=first(MLF) ∧ F1:=getREP(LLN,ID1,ID')+(second(MLF)*forth(MLF)).

op getREP :ListListNat Nat Nat → Float .
eq getREP(nil, ID1 ,ID') = 0.0 .
eq getREP([IDc ID X Y Z] LLN1,ID1,ID') = if ID = ID' then if IDc = ID1 then (X/Y)
else getREP(LLN1,ID1,ID') fi else getREP(LLN1,ID1,ID') fi.

rl [dest-rec-DC-msg] :
{(DCDATAMSG MsgID PL D from O1 to D ) ⟨D:Node | id:ID',NumOfRecMsg:N,A'⟩ execute(D) T CONF }
⟶
{(DCACK MsgID from D to PL) ⟨D:Node | id:ID',NumOfRecMsg:(N+1),A'⟩
[T+sampleExpWithRate(0.1),execute(D)] T CONF }.

rl [rec-DCACK] :
{(SingleDCACK MsgID from D to O ) ⟨O:Node | id:ID,routingTable:(LLN'[ID ID' X Y Z] LLN),A⟩
⟨D:Node | id:ID',A'⟩ execute(O) T CONF }
⟶
{⟨O:Node | id:ID,routingTable:(LLN'[ID ID' X (Y+1) Z] LLN),A⟩
⟨D:Node | id:ID',A'⟩ [T+sampleExpWithRate(0.1),execute(O)] T CONF }.
```

5 Experiments

We used the dataset of iMotes devices in Intel Research Cambridge Corporate Laboratory [23] to evaluate the performance of the proposed centralized and decentralized model. An iMote is a small and self-contained device, which is able to communicate and transmit data to the other iMotes through radio links. This dataset represents Bluetooth sightings of the iMotes nodes which are carried by some 16 admin staff, researchers and interns of Intel's lab for six days, in January 2005. One iMote is placed at a kitchen as the stationary node. Only the information of 9 iMotes are available, because the others had frequently reset.

Each row of the dataset includes a pair of these iMote nodes which meet, in addition to the starting time and the length of this meeting. The information about the reliability of links are not available in the dataset. So, we assigned

the equal value of 3/4 to all the links between all the nodes. That is, we assume that the links are fairly reliable, which is a reasonable assumption, otherwise the very attempt to use such WSN should be questioned. We assume that nodes use their a given transmission power to reach their neighbors, and also that this transmission power is ten times larger than the cost of receiving a message; using such power ratio frees our experiment from any particular radio model. In our experiments, we assume there is one processing node which is always reachable for all nodes, and all nodes (but one that is stationary) move according to the dataset's specification. Finally, every nodes has an initial energy budget of 1000 units.

Finally, we run experiments to investigate the protocol's efficiency (energy-cost), effectiveness (delivery rates) and also to investigate some formal properties of the protocol using the Maude model.

5.1 Investigating the Protocol's Efficiency

In the first experiment, each node send one single data message to the sink. The purpose of this experiment is to study the overhead cost of the learning phase in these protocols, as well as the cost incurred by the nodes in the centralized approach in updating the processing node every time they mode. Figure 1 compares the average remaining energy of all the node which send the data message by using centralized and decentralized protocolsafter running the simulation for 3000 time ticks. The figure shows that the decentralized protocol consumes nearly 1/3 less energy than the centralized protocol at the end of simulation run. Since there is one message being delivered we claim that the process of keeping the processing node up-to-node is to be blamed as a non-trivial consumer of energy.

Even though one could argue that, we could have compared our protocols with another protocol, e.g., AODV, we argue that we would not learn anything because AODV does not have a learning phase and since there is only one message being sent, AODV's would be (unrealistic) competitive. Nonetheless, the next experiments includes a comparison to AODV.

We did another experiment to study the power consumption of the network when all the nodes regularly send data messages. In this experiment, each node has 50 data message in its queue. The data messages are sent from different nodes to the sink node, at random times during the simulation. Figure 2 represents the average of the remaining energy of all nodes during running the decentralized, the centralized and the AODV routing protocol models. Again, each simulation run for 3000 time ticks, and we assume there is no message interference in the network. This experiment, which is more faithful to the normal operation of a WSN, allows one to better appreciate the cost of the route requests in the centralized protocol (in addition to the cost of maintaing the processing node up-to-date)

For comparison with our protocols, we now add the AODV protocol to our experiment. In AODV we assume that the links are not probabilistic and they always exist and the messages always reach to the sink. Note that those are rather *optimistic* assumptions. In AODV each time that a node needs to send

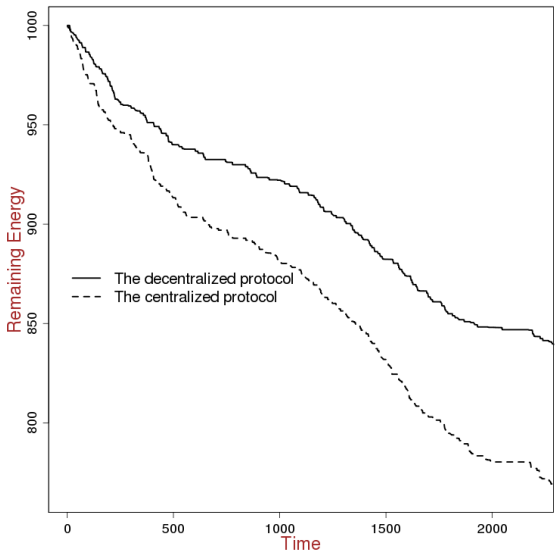

Fig. 1. The comparison of the average energy consumption of the centralized and the decentralized routing protocols for one data message.

a message after it moves, it needs to request a new routing path to the sink. This process is done by broadcasting several route request messages through intermediary nodes. These messages are followed by route reply messages. This routing path discovery process is costly for the network. We assume that nodes also use their minimum power to broadcast the messages in AODV.

As before, both the centralized and the decentralized protocols spend some time in the beginning for learning. We also investigated two scenarios for the centralized protocol. In the first scenario (denotes as "centralized protocol (1)"), we assume, as in the previous experiment, that communicating with the processing node is as costly as communicating with a neighbor node. In the second scenario (denoted as "centralized protocol (2)") is similar to the similar situation with the first scenario except that nodes use twice as much power to communicate with the processing node. While this improves the chances that a processing node is reached, it also consumes more energy, i.e., these scenarios investigates the power consumption of the network when the cost of reaching the processing node increases. The results shows that, in the long term the decentralized protocol consumes 45.9%, 53.5% and 59.7% power less than the first and second scenarios of the centralized protocol and the AODV protocol, respectively. In addition, we note that (*i*) our proposed protocols consume less energy that the AODV protocol, and (*ii*) the decrease in the average amount remaining energy in relatively less pronounced for the decentralized protocol, which is a desirable feature as well.

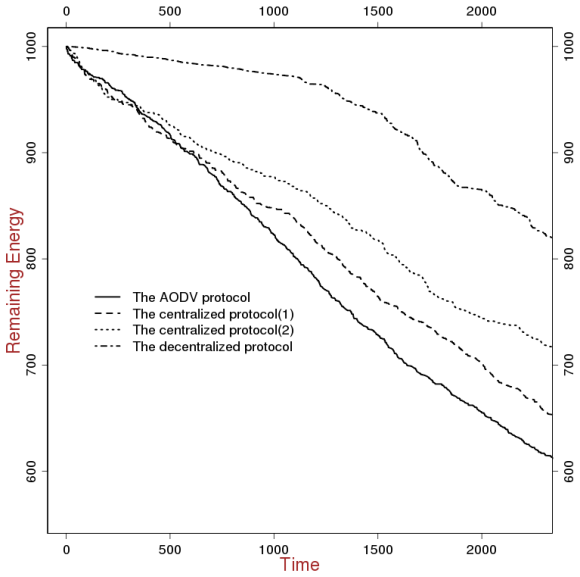

Fig. 2. The comparison of the average remaining energy of nodes in the decentralized protocol, the centralized protocol (1), the centralized protocol (2), which employs twice as much power to reach the processing node, and the AODV routing protocols.

For a better representation of the protocols' energy consumption characteristics, we generated graphs which show the maximum, average and minimum amount of the energy consumed by all the nodes in a simulation. The scenario is the same as considered in Figure 2. These graphs are generated for the decentralized protocol (Figure 3), the centralized protocol (1) (Figure 4) and the centralized protocol (2) (Figure 5). As expected the gap between maximum and minimum is relatively tighter in the centralized version, suggesting that it is more robust.

In our last experiment in this section, all nodes continue moving and sending messages at random intervals until the first node dies, i.e., it runs out of energy. In both the centralized and the decentralized protocols, we show the minimum amount of energy among all nodes. Here we compare only the centralized protocol(1) and the decentralized proposal. Figure 6 shows that the first node dies in the centralized protocol after approximately 5000 time ticks, while the most deplete node in the decentralized protocol still has almost half of its initial energy budget. This experiment suggests that the life time of the network when applying decentralized protocol is at least twice longer than when using centralized protocol.

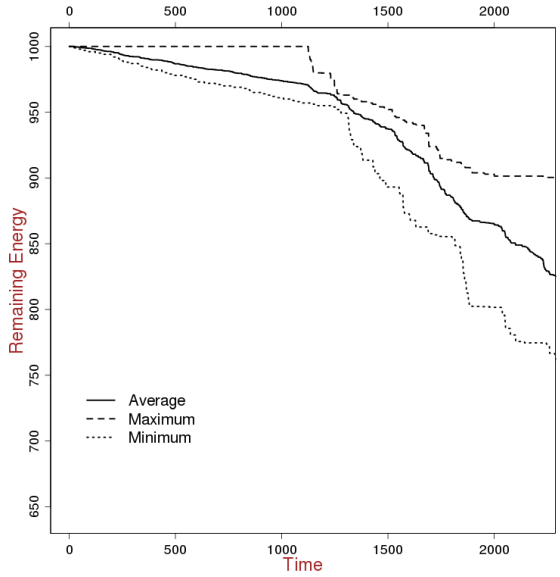

Fig. 3. The comparison of maximum, average and minimum amount of the remaining energy of the nodes in the decentralized protocol.

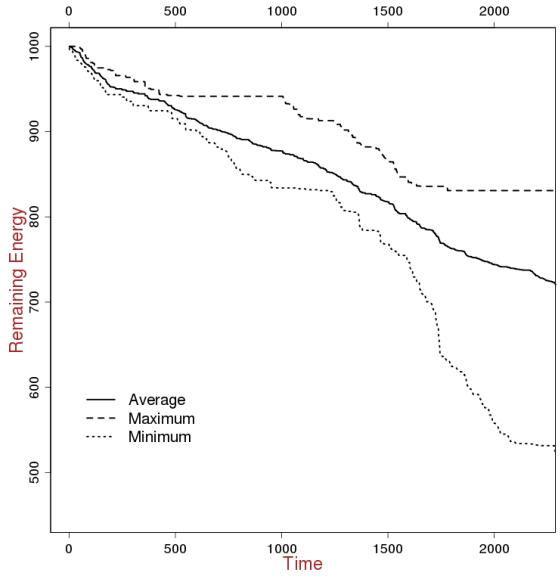

Fig. 4. The comparison of maximum, average and minimum amount of the remaining energy of the nodes in the centralized protocol(1) .

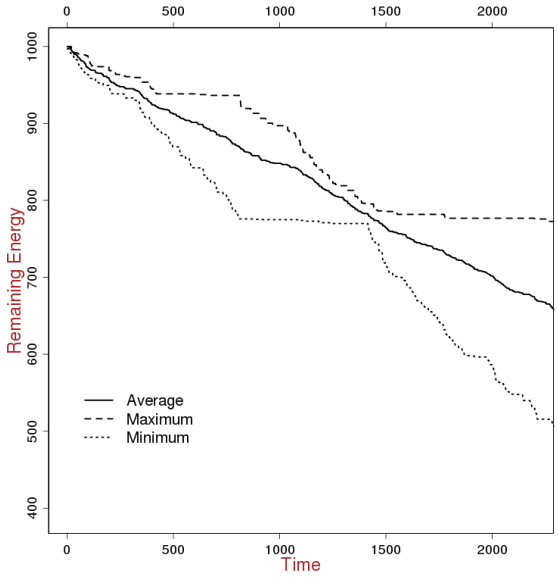

Fig. 5. The comparison of maximum, average and minimum amount of the remaining energy of the nodes in the centralized protocol(2).

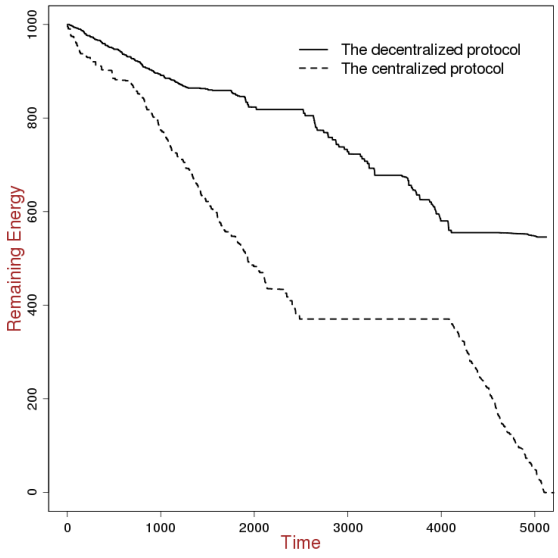

Fig. 6. The comparison of minimum remaining energy of nodes the decentralized and the centralized protocol.

5.2 Investigating the Protocol's Effectiveness

While the above experiments discuss the energy efficiency of the proposed protocols we also need to consider their effectiveness, i.e., what are their actual delivery rates, after all a highly efficient protocol that does not lead to a good delivery rate is of very little practical use.

Considering the successful message transmission rate as the metric of interest, the centralized protocol, as expected, is more effective than the decentralized. On average, the successful message transmission rate of the centralized protocol is 87.1%, while it is 83.6% for the decentralized protocol. According to the average power consumption in Figure 2, the decentralized protocol uses nearly 50% less power more than the centralized protocol at the cost of being merely about 4% less effective. We believe this is quite acceptable trade-off. Furthermore, we note that even though the decentralized protocol made wrong decisions in a few cases, in other cases the centralized approach was not able to deliver its message due, for instance, to unreachability of the processing nodes. That is, the decentralized protocol revealed itself to a very good alternative to the centralized one, as well to the well-known AODV protocol.

5.3 Investigating the Formal Properties of the Protocol

In addition to the quantitative analysis of the performance of the proposed protocols, we took the advantage of the formal modeling methods to prove the validity of the model, regarding its correctness properties. The correctness of our proposed protocols is formalized as a *Linear Temporal Logic (LTL)* formula of $\Box((MSG(m,s,d) \wedge MPP(m,s,d)) \vee (\neg MSG(m,s,d) \wedge \neg MPP(m,s,d)))$. The predicate $MSG(m,s,d)$ is true if a data message m is sent from node s to node d. The predicate $MPP(m,s,d)$ is true if the chosen routing path for m is the most probable path between nodes s and d. This formula means that in all states of the system, the most probable paths are chosen for messages. In other words, there is no state in which this property is violated.

The Maude environment has a *search* tool that searches for failures (the negation of the correctness property) through all possible behaviors of the model. We used this tool to search all the reachable final states of the system to find out if the most probable path is always chosen. For this experiment, nodes only send one message to have a limited search state space. The search tool did not find any violation of the property, meaning that the expected routing path is chosen in all the reachable traces of the model's run, thus asserting the correctness of our proposed protocol.

6 Conclusion

In this paper, we proposed a base routing protocol for WSNs with mobile nodes. This protocol has both centralized and decentralized versions suitable for different network scenarios. We aimed at the energy-efficiency operation of the

network, while improving the rate of successful message transmission. The results show a trade-off between the centralized and the decentralized versions of the protocol. The first one is slightly more effective, while the second one is much more efficient. Both versions of the protocol outperform the well-known AODV protocol in term of energy efficiency and are also more robust, in particular the decentralized version.

In out future work, we are going to refine the protocol by capturing more detailed patterns from the nodes' movement, e.g., temporal patterns. From the modeling view point, we plan to perform probabilistic reasoning of the model via statistical model checking and statistical quantitative analysis.

7 Acknowledgements

This work was partially supported by NSERC DIVA Strategic Network, and it was completed while the first author was visiting the University of Alberta.

References

1. C. E. Perkins and E. M. Belding-Royer, "Ad-hoc on-demand distance vector routing," in *WMCSA*, 1999, pp. 90–100.
2. E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, p. 269âĂŞ271, 1959.
3. H. N. Pham, D. Pediaditakis, and A. Boulis, "From simulation to real deployments in WSN and back," in *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'07)*. IEEE, 2007, pp. 1–6.
4. S. Park, A. Savvides, and M. B. Srivastava, "SensorSim: a simulation framework for sensor networks," in *Proc. 3rd Intl. Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000)*, A. Boukerche, M. Meo, and C. Tropper, Eds. ACM, 2000, pp. 104–111.
5. J. Meseguer, "Conditional rewriting logic as a unified model of concurrency," vol. 96, pp. 73–155, 1992.
6. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada, "Maude: Specification and programming in rewriting logic," vol. 285, pp. 187–243, Aug. 2002.
7. B. S. Olagbegi and N. Meghanathan, "A review of the energy efficient and secure multicast routing protocols for mobile ad hoc networks," 2010.
8. M. Liu, J. Cao, G. Chen, and X. Wang, "An energy-aware routing protocol in wireless sensor networks," *Sensors*, vol. 9, no. 1, pp. 445–462, 2009.
9. J. Wang, J. Cho, S. Lee, K.-C. Chen, and Y.-K. Lee, "Hop-based energy aware routing algorithm for wireless sensor networks," *IEICE Transactions*, vol. 93-B, no. 2, pp. 305–316, 2010.
10. I. Stojmenovic and X. Lin, "Power-aware localized routing in wireless networks," pp. 1–5, 2000.
11. R. Arroyo-Valles, R. Alaiz-Rodriguez, A. Guerrero-Curieses, and J. Cid-Sueiro, "Q-probabilistic routing in wireless sensor networks," in *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, dec. 2007, pp. 1 –6.

12. C. L. Barrett, S. J. Eidenbenz, L. Kroc, M. Marathe, and J. P. Smith, "Parametric probabilistic routing in sensor networks," *Mob. Netw. Appl.*, vol. 10, pp. 529–544, August 2005.
13. A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, pp. 19–20, July 2003.
14. S. C. Ergen, M. Ergen, and T. J. Koo, "Lifetime analysis of a sensor network with hybrid automata modelling," in *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, C. S. Raghavendra and K. M. Sivalingam, Eds. ACM, 2002, pp. 98–104.
15. A. Fehnker, L. van Hoesel, and A. Mader, "Modelling and verification of the LMAC protocol for wireless sensor networks," in *Proc. 6th Intl. Conf. on Integrated Formal Methods (IFM'07)*, J. Davies and J. Gibbons, Eds., vol. 4591, 2007, pp. 253–272.
16. S. Tschirner, L. Xuedong, and W. Yi, "Model-based validation of QoS properties of biomedical sensor networks," in *Proceedings of the 8th ACM & IEEE International conference on Embedded software (EMSOFT'08)*, L. de Alfaro and J. Palsberg, Eds. ACM, 2008, pp. 69–78.
17. P. C. Ölveczky and S. Thorvaldsen, "Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude," vol. 410, no. 2-3, pp. 254–280, 2009.
18. M. Katelman, J. Meseguer, and J. C. Hou, "Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking," in *Proc. 10th Intl. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'08)*, G. Barthe and F. S. de Boer, Eds., vol. 5051, 2008, pp. 150–169.
19. J. S. Dong, J. Sun, J. S. 0001, K. Taguchi, and X. Zhang, "Specifying and verifying sensor networks: An experiment of formal methods," in *10th International Conference on Formal Engineering Methods (ICFEM'08)*, S. Liu, T. S. E. Maibaum, and K. Araki, Eds., vol. 5256, 2008, pp. 318–337.
20. S. Shakya, J. McCall, and D. Brown, "Using a markov network model in a univariate eda: an empirical cost-benefit analysis," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*, ser. GECCO '05. ACM, 2005, pp. 727–734.
21. M. Dorigo and T. Stutzle, *Ant colony optimization.* Cambridge, MA: MIT Press, 2004.
22. A. Riesco and A. Verdejo, "The EIGRP protocol in Maude," Dpto. Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Tech. Rep. SIC-3/07, 2007, http://maude.sip.ucm.es/eigrp.
23. J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau, "CRAWDAD trace cambridge/haggle/imote/intel (v. 2006-01-31)," Downloaded from http://crawdad.cs.dartmouth.edu/cambridge/haggle/imote/intel, Jan. 2006.