ROUTING PROTOCOLS FOR WIRELESS NETWORKED
SENSING AND CONTROL SYSTEMS

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

by

Emmanuel Sifakis

Panos J. Antsaklis, Director

Graduate Program in Electrical Engineering
Notre Dame, Indiana
May 2006
To my parents,

Σηφη και Κατερίνα
CONTENTS

FIGURES ...........................................................................................................iv

TABLES ...........................................................................................................v

ACKNOWLEDGMENTS ...................................................................................vii

CHAPTER 1: INTRODUCTION .........................................................................1

CHAPTER 2: WIRELESS SENSOR NETWORKS .................................................4
  2.1 Previous work .........................................................................................4
  2.2 A new distributed algorithm for “finding multiple backbones in Wireless Sensor Networks” ..........................................................12
    2.2.1 Background ....................................................................................12
    2.2.2 Rationale and description ...............................................................13
    2.2.3 Simulation and results of the algorithm ........................................18
    2.2.4 Performance ..................................................................................25
    2.2.5 Discussion ......................................................................................26
  2.3 A new power-aware routing algorithm ..................................................27
    2.3.1 Summary .......................................................................................27

CHAPTER 3: WIRELESS NETWORKED CONTROL SYSTEMS .........................29
  3.1 Background and previous work .............................................................29
  3.2 Differences with Wireless Sensor Networks ........................................31
  3.3 A novel routing algorithm for Wireless Networked Control Systems ..........35
    3.3.1 Rationale and description ...............................................................36
    3.3.2 Simulation and results of the algorithm ........................................36
    3.3.3 Performance ..................................................................................40
    3.3.4 Discussion ......................................................................................43

CHAPTER 4: CONCLUSION AND FUTURE WORK ........................................51

APPENDIX ........................................................................................................53

REFERENCES ....................................................................................................76
FIGURES

1. Simulation results for 100 nodes in a 100 meters by 100 meters square for the algorithm without extended waiting time for two backbones…………….17

2. Simulation results for 100 nodes in a 100 meters by 100 meters square for the algorithm with waiting time for two backbones…………………………18

3. Simulation results for 100 nodes in a 100 meters by 100 meters square for a random topology……………………………………………………………19

4. Simulation results for 100 nodes in a 100 meters by 100 meters grid…………20

5. Percentage of connected nodes versus the relative density describing the network……………………………………………………………………20

6. Average number of messages exchanged over 100 simulations versus number of nodes in the network in a 120 meters by 120 meters square……….21

7. Percentage of connected nodes using two backbones over 100 simulations for a 100 by 100 field versus the number for a random network…………….22

8. Percentage of connected nodes for two backbones over 100 simulations for a 120 by 120 field versus the number of nodes for a random network……..22

9. Percentage of connected nodes for two backbones over 100 simulations for a 150 by 150 field versus the number of nodes for a random network……..23

10. Percentage of connected nodes for two backbones over 100 simulations for a 100 by 100 grid versus the number of nodes……………………………..23

11. Percentage of connected nodes for two backbones over 100 simulations for a 120 by 120 grid versus the number of nodes………………………………24

12. Percentage of connected nodes for two backbones over 100 simulations for a 150 by 150 grid versus the number of nodes………………………………24

13. One model for a Wireless Networked Control System………………………30
14. Wireless sensor networks and wireless networked control systems........32
15. The cost setup phase.................................................................41
16. The route discovery phase..........................................................41
17. The route maintenance phase.......................................................42
18. The discovered routes...............................................................43
19. Variety of routes..........................................................46
20. Variety of routes and corresponding graph on simulator.................47
21. Importance of the feedback mechanism..................................49
TABLES

1. The MAC-layer parameters..................................................................................44
2. Round-trip times over 50 packets sent.................................................................45
3. Delays in the communication over the 50 first packets.................................47
I would like to acknowledge the help and advice of several people.

The first section of chapter 2 describes a work carried out at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University. This research was completed under the supervision of Dr. Akos Ledeczi and was funded by the DARPA/NEST project. I would like to thank all the people at ISIS who helped and guided me in this work especially Janos Sallai and Peter Volgyesi. Special thanks also to Dr. Janos Sztoipanovits for accepting me at ISIS as an intern in the summer 2005.

Section 2 of chapter 2 was part of a project done for the class on embedded systems taught by Dr. Antsaklis at Notre Dame in the spring 2005 and was done in collaboration with Dr. Haenggi’s student Daniele Puccinelli.

Chapter 3 of my thesis was a work performed under the supervision of Dr. Luis A. Montestruque from EmNet. Without his advice and encouragement this work would not have been possible.

Finally, I would like to thank my advisor Panos J. Antsaklis for his help and encouragement during these 2 years at Notre Dame.
CHAPTER 1
INTRODUCTION

Wireless sensor networks have been the object of many recent studies. They have applications in various domains such as building monitoring, habitat and environmental monitoring. The deployed sensors are able to gather information about sensed phenomena and route them towards a base station for later processing and decision making. The nodes are often deployed in unfriendly environments that are hard to access. Furthermore, sensors are subject to hard energy constraints. The usual goal of a wireless sensor network is to sense in a precise fashion a phenomenon for as long as possible. For all these reasons, the development of power aware routing algorithms i.e. algorithms which would permit an extended lifetime of the network is of capital importance.

More recent research has studied the possibility of using these same nodes to interact as well with the sensed phenomena by controlling some of their parameters. The term of “wireless networked control systems” is used. The use of wireless communication has obvious advantages such as the elimination of costly wiring. The goal of a wireless networked control system is clearly different from the goal of a wireless sensor network, namely to control some parameters of the sensed
phenomenon. In order to do so, it has to respect real time constraints. Therefore, its performance criteria are the ones of a control system i.e. how stable it is to a perturbation, how fast it reaches a desired behavior. For these reasons, the sensed information has to be processed and arrive to the destination node quickly in order, for example, to minimize delays.

In both cases, for wireless sensor networks as well as for wireless networked control systems, the development of adequate routing algorithms is of great importance since those have a direct influence on the performance of the application. More specifically, routing algorithms should be developed to satisfy the specific performance criteria in wireless sensor networks and in wireless networked control systems. The relative lack of adaptive routing algorithms for wireless networked control systems in the literature motivated the development of a novel routing algorithm for wireless networked control system applications described in this thesis.

In Section 1 of Chapter 2, previous work in routing algorithms for wireless sensor networks is described.

In Section 2 of Chapter 2, a simple and distributed algorithm for building multiple backbones in ad-hoc wireless sensor networks is presented. It works in a probabilistic way and reaches good performance for relatively dense wireless sensor networks.

In Section 3 of Chapter 2, a routing algorithm, which selects routes using a joint metric in order to insure good energy balancing, is described.
In Chapter 3, a novel routing algorithm for wireless networked control system applications is presented. It tries to use the best performing route at any time thanks to performance feedback given periodically by maintenance agents.
CHAPTER 2
WIRELESS SENSOR NETWORKS

2.1 Previous Work

In this part, previous work in routing algorithms for wireless sensor networks will be summarized. The list of routing algorithms is of course not exhaustive, but it tries to give a good overview of the different kind of algorithms that are found in the literature.

Routing algorithms can be classified into hierarchical routing, flat routing and location-based routing [AK04]. In flat routing, every node has the same role i.e. each node participates to the routing of the information. On the contrary, in hierarchical routing, not all the nodes play the same role. Some special nodes or cluster-heads are in charge of gathering the information from sensing nodes and sending it towards a base-station. For this reason, due to harder communication constraints, cluster-heads are of higher energy than sensing nodes.

In location-based routing, information about sensor’s position is used in order to route data [ZG04].
Hierarchical routing algorithms

- The LEACH protocol

The LEACH protocol developed by Heinzelman in [HCB00] is a cluster-based protocol. Cluster-heads are elected in a probabilistic fashion and rotate between each other in order to insure good local energy balancing. Cluster-heads collect the information coming from nodes of their cluster and realize data-aggregation and fusion before transmitting to the base-station. Data are collected periodically in a centralized fashion. Thus, LEACH is particularly adapted for applications needing a constant monitoring by the sensor network. Even though LEACH is able to increase the network lifetime, some of its assumptions are restrictive. LEACH supposes that all nodes can transmit with a power that is high enough in order to reach the base station. Thus, LEACH is not adapted to large regions.

- The TEEN and APTEEN protocols

In the Threshold-Sensitive Energy-Efficient Sensor Network protocol (TEEN) and enhanced version APTEEN [MA01], a cluster head sends its cluster members a hard threshold, which is a threshold for the sensed variable and establishes the range of interest, and a soft threshold, which is a small change in the value of the sensed variable. Any of these thresholds triggers the sensor node to communicate the value measured. The hard threshold reduces the number of transmissions by only communicating data when the sensed parameter is on a particular region. The soft threshold further reduces the communication by transmitting only when significant
changes have occurred. The main drawback of TEEN is that if no threshold is triggered the nodes will never communicate.

- Virtual grid architecture routing

The network is divided into rectilinear grids by grouping nodes in regular non-overlapping clusters. Within each cluster, a node is optimally selected to act as a cluster-head. Data aggregation is realized at two levels. First, each cluster-head called also Local Aggregator is in charge of realizing a local data aggregation. Then, a subset of the cluster-heads realizes a global aggregation before sending the data to the base-station. The grid formed can be shifted in order to insure good energy balancing [AK03].

Flat routing algorithms

- Ad-Hoc on demand distance vector routing

The Ad hoc On Demand Distance Vector (AODV) [PR00] is capable of both unicast and multicast routing. It is an on demand algorithm, meaning that it builds routes between nodes only as desired by source nodes. When a node requires a route towards a destination for which it does not have a route yet, it broadcasts a route request packet (RREQ) across the network. As the reply to the route request (RREP) propagates back to the source, nodes set up pointers to the destination. Once the node receives the RREP, it can begin sending data to the destination. Nevertheless, AODV requires the communication to be delayed until a route to the destination is found. This is not suitable for real time applications.
• Dynamic source routing

In dynamic source routing (DSR), a sender of a packet determines the whole sequence of nodes through which the packet will be forwarded in order to reach the destination node [JM96]. Each packet contains a header with the ID of the next node of the route. After receiving a message, if this node is not the final destination, it forwards the message to the next hop of the route and so on until reaching the final destination. The route discovery protocol allows a node to dynamically determine a route to any other node of the network.

AODV and DSR are not suitable for wireless networked control systems. Indeed, they require a large amount of communication as well as computational power to obtain optimal routes between nodes.

• Directed diffusion

Directed diffusion was introduced by Intanagonwiwat et al. as a data-centric, data dissemination paradigm for sensor networks [IGE00]. Nodes requesting information are called sinks and nodes generating information are called sources. Records indicating a need in a certain type of information are called interests. Sinks will generate interests that will propagate throughout the network looking for nodes with matching event records. Events start flowing towards the originators of interests along multiple paths. Interests contain an interval attribute field that indicates the frequency with which data related to this interest should be sent. This longevity of the communication allows the directed diffusion protocol to learn which the good paths
between sources and sinks are. During this communication period, the sensor network reinforces one or a small number of these paths over the period of use of these paths.

This routing algorithm is resilient to node failure since it can reach a destination node through a big variety of routes but leads to an increased amount of traffic by flooding the network with interests.

- Rumor routing

Rumor routing described in [BE02] is a good alternative to directed diffusion. Indeed, in many situations the amount of information to exchange is small. While in the directed diffusion algorithm, requests are sent by sinks in a two dimensional fashion (multiple routes) in order to reach source nodes, in the rumor routing algorithm both sinks and sources spread information in a one dimensional fashion by following a random walk (single route). When the curve emanating from a source meets the one emanating from a matching sink, a path between the two has been established. In general this algorithm reduces the amount of communication by avoiding flooding the network with queries, event notifications, or gradient generating packets. There is a tradeoff between near optimal routes and large bandwidth requirements.

- SPEED

SPEED is a routing algorithm introduced in [SLA03] and is a localized algorithm with minimal control overhead. It tries to enforce a minimum traveling speed for packets to its destination. In order to do so, whenever a packet needs to be
routed, the forwarding node evaluates the distance for the packet to travel using geographical information. It will then forward the data to the node that will most likely be able to forward the packet at a traveling speed above the minimum one. This algorithm tries to enforce the traveling speed of packets. Nevertheless, if the minimum speed cannot be achieved, packets will automatically get dropped in order to reduce network traffic even though alternative paths are available. In networked control systems, a constant flow of information is sometimes more important than a shorter delay.

Location based routing algorithms

• Geographic Adaptive Fidelity (GAF)

GAF is an energy aware routing algorithm described in [EXH01]. The network is divided into a virtual grid. Each node uses its GPS-indicated location to associate itself with a point in the virtual grid. Nodes which correspond to the same point in the grid are considered to have the same cost in packet routing. This equivalence is used in order to send to sleep some nodes of a particular grid and save energy. Thus, GAF increases the network lifetime as the number of nodes increases. There are three possible states for a node within a grid: the discovery state, during which neighbors within a grid are found, the active state if the node participates to the routing and the sleep state, during which the radio of the node is turned off and energy is saved. GAF reaches similar results to classical ad hoc routing protocols in terms of latency and packet loss while reaching an increased lifetime for the network. The GAF scheme is a
location based algorithm but can also be considered as a hierarchical routing algorithm. For each grid at any time, a node acts as a leader and communicates with the base station on behalf of the other nodes. Nevertheless, it does not realize any data aggregation like in most hierarchical routing algorithms.

- Compass Routing Method (DIR) and Geographic Distance Routing (GEDIR)

We consider the simple case in which we wish to route a message to a single destination node d from a node x. DIR (compass routing) described in [KSU99] works in the following way. Among the neighbors y of x that make an angle $\angle dxy < \pi$, we pick the one with the smallest angle $\angle dxy$. GEDIR (greedy distance routing) is described in [BM99]. Among the neighbors y of x closer to d than x, it picks the closest to d. These algorithms are relatively simple and do not require any precomputation for the routing. The main idea is to go in a direction such that the data get closer to the destination. Nevertheless, the delivery of the packet to the destination is not insured in case all neighbors of x are further away from the destination than x itself. In this case, the packet encounters a “hole” and will be stuck at x.

- Geographic and Energy Aware Routing (GEAR)

GEAR was presented in [YEG01]. It operates in two distinct phases. First, it delivers the packet to a node in a desired region and then distributes the packet within the region. It uses geography based and energy aware heuristics in order to route a packet towards the region. The purpose of GEAR is to restrict interests within a region rather than flooding the whole network with interests. Thus, it differs from directed
diffusion. Each node keeps a learned cost for reaching the destination through its neighbors. The estimated cost is a combination of the residual energy of the node and the distance to destination. The GEAR protocol performs the following steps at each visited node x:

- If x has neighbors closer than x to the destination d in both the euclidian distance sense and the learned cost sense, then it picks among these neighbors the one with the smallest learned cost and forwards the packet to that precise neighbor.

- Else, it forwards the packet to the neighbor with the smallest learned cost. This case corresponds to the presence of a hole which occurs when a node does not have any closer neighbor to the target region than itself.

Unlike the algorithms DIR and GEDIR, GEAR manages to solve the problem of holes in a network.

- SPAN

SPAN was developed in [BMCJ01]. SPAN elects coordinators based on their geographic positions. The coordinators form a backbone that is used to route data to the base station. A node should become a coordinator in case two neighbors of a non coordinator node can not reach each other through one or two coordinators. Thus, SPAN requires 2 hops away information in the coordinator election leading to a large amount of exchanged messages.
2.2 A new distributed algorithm for “finding multiple backbones in Wireless Sensor Networks”

2.2.1 Background

Energy efficiency in wireless sensor networks has been the object of many recent studies. It has been accepted that the best way to save energy is to send to sleep some of the nodes in order to operate them at lower duty cycles [EH00]. Nevertheless this should not lead to a deterioration of performance of the wireless sensor network through for example a loss of connectivity. In this section, we focus on the implementation of an algorithm which permits the formation of disjoint and connected dominating sets in a dense Wireless sensor network.

Specifically, consider a shooter localization application [LB04] in which sensing nodes are disseminated over a field to monitor an area for events of interest. When such an event occurs, a sensing node generates some data that need to be routed towards the gateway. In order to do so, a path of active sensors towards the gateway should exist at this time. If no path existed, the node would have to wait until one becomes available which would lead to a delay in the acquisition of the data by the gateway. For this reason, it is necessary to ensure connectivity in the network at all time.

The problem is to find multiple disjoint subsets of the nodes that need to be awake in order to ensure connectivity. By having multiple subsets, we can switch between these awaken subsets in order to increase the lifetime of the network - meaning here the time during which the application would accomplish successfully its
goal-. If we have, for example, two subsets, we multiply by two the lifetime of the network compared to a single subset. After finding these subsets we have to synchronize the nodes within each disjoint subset and these would wake up at the same time in order to ensure the routing. The found subsets would alternate between sleep and active periods, one single subset being awake at a given time. Since these subsets are disjoint, this ensures good energy balancing.

In order to decrease the duty cycle, we should try to find as many subsets as possible. This refers to the problem of finding the minimum connected dominating set which is known as being NP complete [LHF04]. For this reason, we developed with an algorithm that works in a probabilistic way since it does not necessarily ensure connectivity for all the nodes but gives a probability with which a given number of nodes are connected.

The remainder of section 2 is organized as follows. Section 2.2.2 presents the algorithm. Section 2.2.3 describes the performance of the algorithm. Section 2.2.4 compares this algorithm to existing solutions like SPAN or cluster-based approaches. Section 2.2.5 draws some conclusions and describes possible further improvements.

2.2.2 Rationale and description

A graph is a set of objects called vertices connected by links called edges. A dominating set S for a graph G is a set of vertices whose neighbors (meaning here the set of vertices they are linked to), along with themselves, constitute all the vertices in the graph. The algorithm aims at building multiple disjoint virtual backbones or disjoint connected dominating sets in a network. The idea is to have, at a given time,
one backbone that is awake and to switch between the constructed backbones in order to ensure energy balancing.

The algorithm works in a probabilistic way and so doesn’t ensure that each node is connected at the end. After applying the algorithm to the network, each node will have one color and all nodes of the same color will hopefully form one backbone. Further, we will refer to a particular backbone using the respective color.

The algorithm works in the following way:

- The gateway or base-station broadcasts a message. Each of the nodes that receives a message from the base-station picks randomly a color among the list of colors and advertises its choice.

- Each node has a counter for each color counting the representation of each color in its neighborhood (i.e. the number of greens, red and so on) and updates its counters based on what it hears.

- It waits a random amount of time and checks its counters.
  - If after this time all colors are represented then it chooses the least represented color in its neighborhood. This aims at equally representing each color among the network.
  - If one color is not represented in its neighborhood it will delay its decision by a given duration WAITINGTIME (50ms) and check again its counters...and so on until possibly reaching eight time extensions, i.e., a maximum time of 400ms. If the node still has a non represented color after this maximum time, it chooses the color which corresponds to the least represented color (it will not choose a color that is not represented) and broadcasts its choice. In this case the node is said to be "not connected" yet. Indeed, suppose this node goes to sleep and the awaken backbone corresponds to the color that is not represented in its neighborhood. Then if this node wakes up through the sensing board because of an event of interest and needs to send some data towards the base-station then it can not since no node of the awaken backbone is in its neighborhood.

- The algorithm runs until each node chooses a color. So, the algorithm works in such a way that if each node has at least one neighbor of each color then it means that it managed to build k disjoint backbones where k is the number of colors used in order to color the graph. The vertex connectivity of a graph is
the minimum number of nodes whose deletion from the graph disconnects it. If we try, for example, to color the graph with \( k \) colors then the vertex connectivity of the graph should be at least \( k \). Indeed, the number of disjoint dominating sets of a graph is less or equal to the vertex connectivity of the graph. In order for the algorithm to have a chance to build \( k \) disjoint dominating sets the vertex connectivity of the graph should be at least \( k \). The algorithm does not compute the vertex connectivity of the graph. It tries to color the graph leading sometimes to unconnected nodes.

**Importance of the waiting time**

The algorithm was first implemented without the extended waiting time, i.e., a node, after waiting the random amount of time described above, would check its counters and make its decision right away on which color to choose. If all colors were represented, it would choose again the less represented one. If some colors were not represented, it would choose the less represented one among the represented colors. The results of the simulation for this version of the algorithm were worse than those of the newer version. The connectivity results were good for nodes close to the base station but were getting worse far away from the base station leading to the formation of vast regions where only one color was represented. The bigger the network was, the worse the results were for a same node density.

The introduction of the variable `WAITINGTIME` solves this problem. It delays the color decision of a node in case one color is not represented in its neighborhood (meaning here the set of its neighbors) hoping that this missing color will appear during this waiting time through the color decisions of other neighboring nodes. The introduction of the extended waiting time tries to slow down the expansion of an unwanted coloring that would lead to the absence of a color in a region. If after this extended waiting time, the missing color appears in its neighborhood, there is a high
probability that this node chooses this previously missing color since it picks the less represented color. If for example, the node made its decision without waiting, it would choose the less represented color among the represented colors possibly leading to a whole region in which at least one color is not represented. This whole region would be then disconnected if this particular color is used as backbone (cf. figure 1).

In the figures, the square in the center of the figure represents the base station, the squared nodes (filled and unfilled to differentiate the colors) correspond to the connected nodes and the circles (filled and unfilled for the two different colors) are the nodes that the algorithm did not manage to connect. The communication range $r$ of the nodes is constant and of value 30 meters.

Figure 1 shows the formation of regions in which only one color is represented in absence of extended waiting time. In this example, at the end of the algorithm, only 79 percent of the nodes were connected.
Figure 1: Simulation results for 100 nodes in a 100 meters by 100 meters square for the algorithm without extended waiting time for two backbones with a communication range $r$ of 30 meters.

Figure 2 shows that the algorithm with the waiting time has much better results than the one without. Indeed, for the same topology, it manages to connect 94 percent of the nodes.
Figure 2: Simulation results for 100 nodes in a 100 meters by 100 meters square for the algorithm with waiting time for two colors with $r=30$ meters.

2.2.3 Simulation and results of the algorithm

The performance of the algorithm heavily depends on the node density, the communication range, which is supposed to be the same for each node and the number of colors we try to color the graph with. Remind that this number should be less or equal to the vertex connectivity of the graph. The simulations were made using the ISIS developed tool JProwler (a Java version of Prowler) [MLSV03] which is an open source discrete event simulator for verifying and analyzing communication protocols of ad-hoc Wireless sensor networks. The algorithm was implemented in Java. The nodes were randomly placed on a square area and the base station was placed in the center of this area (intersection of the diagonals of the square).
Figure 3 shows the simulation of the algorithm using JProWler for a random topology.

Figure 4 shows the simulation of the algorithm using JProWler for a grid.

Figure 5 shows that for a high relative density, the algorithm managed to connect most of the nodes. These results were obtained by having a fixed range $r$ and by varying the area $A$ of the field between 2500 and 122500 square meters as well as the number of nodes between 50 and 350.
Figure 4: Simulation results for 100 nodes in a 100 meters by 100 meters grid with $r=30$ meters.

Figure 5: Percentage of connected nodes versus the relative density describing the network.
Another important aspect of the algorithm is the number of messages exchanged in order to attribute a color to each node. The number of messages exchanged in the network depends mainly on the number of nodes in the network. Figure 6 shows that the average number of messages exchanged becomes closer to the number of nodes in the network as the number of nodes increases. Thus, we can see that this algorithm is particularly adapted to dense wireless sensor networks. The percentage of connected nodes through the algorithm depends on the size of the field, the number of nodes and on the communication range. Without restriction to generality, the range is supposed to be the same for all the nodes and of value 30 meters.

![Figure 6: Average number of messages exchanged over 100 simulations versus number of nodes in the network in a 120 meters by 120 meters square.](image)
Figure 7, 8 and 9 show that with a relatively dense network more than 95 percent of the nodes are connected when using two colors.

Figure 7: Percentage of connected nodes using two colors over 100 simulations for a 100 by 100 field versus the number for a random network.

Figure 8: Percentage of connected nodes using two colors over 100 simulations for a 120 by 120 field versus the number of nodes for a random network.
Figure 9: Percentage of connected nodes using two colors over 100 simulations for a 150 by 150 field versus the number of nodes for a random network.

Figure 10, 11 and 12 show that the algorithm has much better connectivity results with a grid than with a random topology.

Figure 10: Percentage of connected nodes using two colors over 100 simulations for a 100 by 100 grid versus the number of nodes.
Figure 11: Percentage of connected nodes using two colors over 100 simulations for a 120 by 120 grid versus the number of nodes.

Figure 12: Percentage of connected nodes using two colors over 100 simulations for a 150 by 150 grid versus the number of nodes.
2.2.4 Performance

A lot of methods have been developed in order to find a reasonable number of connected dominating sets. There are two main types; the first type is cluster-based: the network is divided into disjoint clusters [VKCP95] that satisfy the following conditions:

- Each node within a cluster can communicate with nodes in the same cluster.
- Each node within a cluster can communicate with nodes in neighboring clusters.

There are different types of partitions giving different results in terms of average duty cycle and probability of failure [HL05]. The GAF scheme [EXH01] for example uses geographic location information in order to divide the networks into square grids of constant size. At each time, at least one node in each grid is awake in order to ensure the routing.

The LEACH protocol elects rotating cluster-heads which would gather information in order to route it towards the gateway. It includes randomized rotation between elected cluster-heads for better energy balancing. Cluster heads then send the data to the gateway through a single hop communication using high energy transmission. Even though cluster-heads rotate between each other, those that are in clusters that are far away from the gateway would die faster than cluster-heads that are closer since they need a higher energy transmission to reach the gateway.

The second type of method is known as the virtual backbone [BD97]. The SPAN [BMCJ01] algorithm elects coordinators in order to build a connected dominant set that will ensure the routing. The coordinator election is done in an energy
balancing fashion without consequent reduction of the capacity of the network. Nevertheless, it requires for a node to get two hops away information which leads to an increased exchange of messages.

The algorithm introduced in this paper is distributed and particularly adapted for dense wireless sensor networks in the case of two colors. Indeed, in this case, it manages to connect more than 95 percent of the nodes and the number of messages exchanged is close to the number of nodes. Unlike most of the cluster-based approaches it does not require any knowledge about the geographical position of the nodes. The LEACH protocol [BHC00] elects head-clusters that use direct transmission to the base station. This does not ensure good global energy balancing since nodes of a cluster that is far away from the base station need a higher transmission power than nodes of a closer cluster. The algorithm is simple so easily implementable in NesC on the TinyOS platform. It is fast since it requires less than two seconds to attribute a color to all the nodes of the network.

2.2.5 Discussion

In section 2 of this chapter, an algorithm which aims at building multiple disjoint backbones in a wireless sensor network was described. The simulations were made using JProwler on randomly generated networks and grids for 2 backbones. For dense wireless sensor networks (average neighbor count of 24), the algorithm manages to connect more than 95 percent of the nodes for a random topology while exchanging a small number of messages. With a grid, the connectivity success of the algorithm was significantly increased. Furthermore, it requires a low execution time. A trade-off
should be found between the number of connected nodes and the number of backbones. The algorithm did not check for the vertex connectivity of the graph. The algorithm works in a probabilistic way since the problem of finding minimum connected dominating sets is NP complete. It ensures good energy balancing since the built backbones are disjoint. The constructed backbones would switch between each other in order to send to sleep a large part of the network resulting in the reduction of the energy consumption.

Further improvements would be to implement the developed algorithm on real nodes and see how it behaves.

2.3 A new power-aware routing algorithm

2.3.1 Summary

We consider a many-to-one real-time sensor network where sensing nodes are to deliver their measurements to a base station under a time constraint and with the overall target of minimizing the energy consumption at the sensing nodes.

In wireless sensor networks, the unreliability of the links and the limitations of all resources bring considerable complications to routing. Even in the presence of static nodes, the channel conditions vary because of multipath fading effects due to the motion of people or objects in the environment, which modify the patterns of radio wave reflections. Also, sensing nodes are typically battery-powered, and ongoing maintenance may not be possible: the progressive reduction of the available energy needs to be factored in. The quality of the links and the remaining energy in the nodes
are the primary factors that shape the network graph; link quality may be measured
directly by most radios, whereas residual energy is related to the node battery voltage,
which may be measured and fed into the microcontroller. These quantities may be
used to form a cost function for the selection of the most efficient route. Moreover, the
presence of a time constraint requires the network to favor routes over a short number
of hops (a.k.a. the long-hop approach, in the sense that a small number of long hops is
used) in order to minimize delay. Hop number information may be incorporated into
the cost function to bias route selection toward minimum-delay routes. Thus, a
crosslayer cost function is obtained, which includes raw hardware information
(remaining energy), physical layer data (channel quality), and a routing layer metric
(number of hops). A route selection scheme based on these principles intrinsically
performs node energy control for the extension of the lifetime of the individual nodes
and for the achievement of energy balancing in the network; intuitively, the long-hop
approach permits the time-sharing of the critical area among more nodes. A novel,
practical algorithm based on these principles is proposed with the constraints of the
currently available hardware platforms in mind. Its benefits are investigated with the
help of computer simulation and are illustrated with an actual hardware
implementation using Berkeley motes.

A more complete description of this work is available in [PSH05].
CHAPTER 3

WIRELESS NETWORKED CONTROL SYSTEMS

3.1 Background and previous work

*Networked control systems* are control systems which communicate over a network. Wireless networked Control Systems, WNCS, contain sensors, actuators and controllers which are interconnected over a wireless communication channel. They must communicate information with a given time constraint in order to avoid instabilities. They usually require sending more data packets of smaller size than wireless sensor networks.

WNCS offer advantages such as the elimination of wiring as well as the possibility of self-organization of networks of heterogeneous devices. For example, without cables we have the freedom to place the nodes in locations not accessible with cables. Nevertheless, using wireless for control applications brings new challenges. Wireless channels generally have higher bit-error rates than wired. Furthermore, message delays tend to be larger. Delays in the feedback loop due to network
communication latencies lead to decreased phase margins which can destabilize a system.

In order to realize a good control of a plant using wireless networked control systems, routing algorithms must explicitly address these issues.

One Model for a Networked Control System:

![Diagram of a networked control system](image)

Figure 13: One model for a wireless networked control system

Although there are different models of wireless networked control systems, we present this one for simplicity.

- \( u(t) \) and \( y(t) \) are respectively the continuous control input and the continuous output of a given plant.
• $y(k)$ is the sampled output of the plant which will be transmitted over the network to the actuator/controller.

• $h$ is the sampling rate

• $\hat{y}(k) = \begin{cases} y(k) & \text{if sucessfully routed over network (} p \text{)} \\ 0 & \text{otherwise (} 1 - p \text{)} \end{cases}$

The control signal $u(t)$ will not be updated until $\tau_s$ seconds later with probability $p$.

### 3.2 Differences with Wireless Sensor Networks

A sensor network is typically composed of three main entities [THA02]. First, the network contains a set of sensing nodes which take measurements related to a phenomenon and route these measurements to the base station. The second is a base station(s) or data collector(s) which gather(s) information about the sensed phenomenon. The third is the phenomenon which corresponds to the event of interest.

A WNCS is also composed of three main entities. First, it contains a set of sensing nodes which take measurements related to a phenomenon and route these measurements to the actuator/controller; second, actuators and controllers in charge of receiving sensed data and sending a control command to the plant. The third entity is a parameter of interest that is to be controlled.

Unlike wireless sensor networks, wireless networked control systems not only measure and detect events in the environment they are deployed, they also interact with it in order to control some of its parameters.
Wireless sensor networks usually have one base station or data sink while wireless networked control systems have many actuators/controllers which behave like data sinks. (Figure 14)

![Sensor Network (with one Data Collector) and Sensor-Actuator Network](image)

Figure 14: Wireless sensor networks and wireless networked control systems

Wireless sensor networks require an accurate assessment of a sensed phenomenon while control systems may tolerate less accuracy in order to maintain stability. The goal of the networked control systems is to control a plant. In order to meet the performance goals, accurate information is not always the most important criteria. Typical performance goals of a wireless networked control system are to maximize phase margins, minimize overshoot and minimize settling time.

The performance criteria for a wireless sensor network are how accurately the phenomenon of interest is measured. A measurement of the performance of a wireless sensor network would be the difference between reality and the picture taken by the wireless sensor network at a given time. For this reason, it is important for all the data to arrive to the base station. Thus, the data that needs to be routed is usually of much bigger size than that for wireless networked control systems.
For wireless networked control systems, detailed information (in terms of packet size) is not necessary to reach acceptable control performance [EM01], [NE00], [HO02].

Small delays as well as precise plant modeling [MA02] are of capital importance for wireless networked control systems. Indeed, good plant modeling allows to efficiently controlling the plant while sending the sensed data at rather spaced intervals of time and thus leading to reduced traffic. The reduced traffic could possibly lead to reduced delays introduced by the network. Unlike wireless networked control systems, wireless sensor networks are not too sensitive to delays.

In some cases, the use of efficient packet dropping policies [LL02] is relevant. Rather than sending packets that would lead to an increased traffic and possibly to increased delays, some should be dropped. Unlike for wireless networked control systems, dropping packets would deteriorate the performance of the wireless sensor network. It would lead to an erroneous picture of the sensed phenomenon.

Wireless networked control systems typically send small amounts of data periodically but require guaranteed transmission and bounded time delay for messages. Wireless networked control systems require the information to arrive from the plant to the actuator/controller while reducing packet delays. Indeed, from the controller-plant perspective, the network can be modeled by a delay of variable length.

Constant delays are preferable to variable delays i.e. delays with a small jitter are preferable. Indeed, non constant delays could destabilize the system or deteriorate its performance. For wireless sensor networks, variable delays and bursty transmissions are acceptable while they are not acceptable for wireless networked
control systems. Indeed, applications for wireless sensor network do not usually have hard real time constraints.

Sources of delay in wireless networked control systems

The time delays that are important in wireless networked control systems are the sensor to actuator delays [LMT]. These delays can be broken into two parts:

- delays due to device
- delays due to the network

Delays due to the device are introduced at the source and destination nodes. At the source node, it includes the preprocessing time and the waiting time. At the destination node, it corresponds to the post-processing time.

The network time delay is due to the transmission time of a message and to the propagation.

- Pre- and post-processing time

The pre-processing time corresponds to the time required to encode a message into the adequate network data format. The pre-processing is not always negligible.

The post-processing time corresponds to the time required to decode the data network into the physical data format and output it to the external environment. They both heavily depend on the characteristics of the hardware.
• Transmission time

It corresponds to the most deterministic delay in a network system. It depends on the data rate, the message size and the distance between two nodes.

\[ T_{tr} = T_{frame} + T_{prop} \]

\( T_{frame} \) is the time required to send a packet and \( T_{prop} \) the time spent for the propagation between the two devices.

• Waiting time at source nodes

Time spent by the message before being sent by the source. It can spend time waiting in the queue of the sender’s buffer and could be blocked by other messages due to communication in the network (blocking time). It depends on the network protocol and has a huge influence on the performance of the control network. The routing algorithm has a direct influence on the waiting time.

3.3 A novel Routing algorithm for Wireless networked control systems

In the remainder of the thesis, the term “destination node” will be used to denote independently an actuator, a controller or a pair actuator/controller and the term “source nodes” will denote sensors.

Using the characteristics of wireless networked control systems, we can infer that efficient routing algorithms for wireless networked control systems should have the following characteristics:
Continuous flow of information:

Unlike wireless sensor networks, wireless networked control systems do not necessarily require accurate information. For control purposes, it is more important to have a continuous flow of information from the source nodes to the destination nodes. Indeed, delays due to the network could destabilize the system. These delays should also have a small jitter.

Small recovery time:

A single node failure should not destabilize the whole system. In case of a loss of communication due to a node failure, a new communication path should be found from the source nodes to the destination nodes quickly. A node should have many routes leading to a given destination node and should be able to switch from one route to the other very quickly.

3.3.1 Rationale and description

This new algorithm involves the following agents:

Cost setup:

This phase is initiated by the different destination nodes. It permits to each node to establish its cost with respect to each destination node. The cost of a node towards a destination node represents here the minimum number of hops needed to reach this destination node. The setup of the cost will help the discovery agents in their search for routes towards the destination nodes.
**Route discovery agents:**

Agents are sent by each sensor in the network to look for routes towards the destination node. These agents will travel the network in a randomized fashion using the cost. They would usually move in the direction of a decreasing cost but sometimes pick a random direction. Randomness is crucial since it permits to discover new routes and to avoid getting “stuck” in local minima. Imagine discovery agents were only going in a direction of a decreasing cost. Suppose also one node of cost 3 had only one neighbor of cost 2 which unfortunately died, then, in this case, no routes would ever be found through this node. Thus, this method aims at discovering as many routes as possible. These agents are sent periodically in order to discover new routes.

**Maintenance agents:**

Maintenance agents are sent back by the destination nodes to give feedback about the discovered routes in terms of delay, percentage of delivery success.

**Feedback Mechanism:**

This feedback mechanism will select probabilistically a set of the available routes available to transmit data. The decision will be based on the ratio of delivery success of each route. This ratio will be determined by the link with lower packet delivery ratio and will be sent to the source nodes by means of maintenance agents. This feedback mechanism is of great importance since it gives to the source nodes an updated vision of the performance of each route at any time. Thus, those will be able to route data using the best performing route.
This algorithm is inspired from the ant-based route discovery [BDT99]. It aims at establishing quickly a first communication path between a source node and a destination node without worrying about it being optimal. Since packets in wireless networked control systems are of small size, the use of an optimal path might not be necessary right away. The more frequent the communication between a pair sensor-destination node is, the more often discovery agents will be sent leading to an increased number of communication paths between the pair sensor-destination node. A subset of the discovered routes will be selected for the routing based on their performance established through the feedback mechanism.

The algorithm was implemented in the following way:

The first phase is the **Cost setup phase**.

- Each destination node broadcasts a HELLO Message. Each node which receives the HELLO message uses it in order to establish its cost with respect to this destination node.

- Each node then broadcasts its ID, the ID of the destination node as well as its cost $C$ with respect to the corresponding destination node. A node (that hears the message) will replace its previous cost $C_p$ with respect to a given destination node only if the new computed cost $C_{+1}$ is smaller than $C_p$.

- At the end of this phase, each node will have a cost with respect to each of the destination nodes of the network but also the ID and costs of its neighbors.

The second phase involves **the route discovery agents** and corresponds to the route discovery phase.
• When a node needs to send data towards a given destination node, it will send route discovery agents in order to find routes towards the corresponding destination node.

• It will generate a random number between 0 and 1.
  
  - If this number is between a parameter $\beta$ and 1, then it will broadcast a message towards a neighbor which has a higher cost than its own cost which means practically that the message will be broadcasted towards a node which is closer to the destination node.
  
  - If this number is between 0 and another parameter $\alpha$, then it will broadcast the message towards a neighbor which has the same cost as its cost.
  
  - If this number is between the parameters $\alpha$ and $\beta$, then it will broadcast the message towards a neighbor with a smaller cost than its cost.

• The selected node will then save the ID of the node which selected it and repeat the same randomized process.

• The route discovery agents will, in this way, move in a random fashion aided by the costs. The parameters $\alpha$ and $\beta$ are chosen in such a way that, most of the time, the discovery agents will move in a direction such that the cost is decreasing but will sometimes choose a direction other than the one corresponding to the descending cost.

• The route discovery agents will hopefully reach the destination node.

• Each node can save up to five routes towards a given destination node.

The third phase involves the maintenance agents and corresponds to the route maintenance phase.

• The destination node, which received the route discovery agents, will answer by sending periodically maintenance agents towards the node which sent the request for communication for all the routes found.

• The maintenance agents will follow the reversed path by taking care of eliminating cycles in the path.

• They will reach the sender which will update the performance of all the routes.
• The sender can now forward data to the destination node using the best performing route towards this destination node.

3.3.2 Simulation and results of the algorithm

The routing algorithm was simulated on the probabilistic wireless network simulator Prowler [MLSV03]. 100 motes were deployed in 100 meters by 100 meters grid. The range of the sensors is 30 meters. The following values for the parameters $\alpha$ and $\beta$ were chosen for the simulations $\alpha = 0.2$ and $\beta = 0.9$.

Figure 15 shows the cost setup phase. We can observe on the bottom left of screen the presence of the destination node or data sink. Each sensor establishes during the cost setup phase its cost with respect to this destination node, as well as the ID and costs of its neighbors.

Figure 15: The cost setup phase
Figure 16 illustrates the route discovery phase. A sensor randomly chosen among the network has to send data towards the destination node. It sends discovery agents in order to find routes.

In figure 17, the destination node that receives the route discovery agents sends back maintenance agents to the sender through the reversed path in order to give feedback about the discovered route. It takes care of eliminating unwanted cycles in the discovered path.
The sender can store up to 5 routes towards a destination node in a circular buffer. The discovered routes are in red on figure 18.

Figure 17: The route maintenance phase

Figure 18: The discovered routes
3.3.3 Performance

In order to simulate the algorithm, a deterministic model of the radio was used. For the simulation, the following parameters were chosen for the radio as well as for the parameters specific to the algorithm.

MAC-layer Parameters:

The MAC layer parameters are the parameters describing the behavior of the radio of the motes. They involve the following times:

- The waiting time, which is the time spent between channel request and channel idle check.
- The back off time, which is the time waited after an unsuccessful idle check, before another channel check.
- The packet length, which is the time required to send one packet.

All the MAC-layer parameters are measured in bit-times, where one bit-time is the length of one transmitted bit in the radio channel (in the original TinyOS). One bit-time equals to 25µs.

The back-off time should not be picked too small in order to avoid collisions. For the same reason, the waiting time should have a random part. Indeed, if it did not have any random part, it would lead to simultaneous idle channel checks, thus nodes would try to send data at the same time leading to collisions and data loss.

For simulation purposes, we picked the following parameters for the MAC layer (table 1).
TABLE 1
THE MAC-LAYER PARAMETERS

<table>
<thead>
<tr>
<th>MAC layer parameters</th>
<th>Waiting time</th>
<th>Backoff time</th>
<th>Packet length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time(in bit time)</td>
<td>100+800*rand</td>
<td>200+400*rand</td>
<td>800</td>
</tr>
<tr>
<td>Time (in ms)</td>
<td>Between 2.5 and 22.5</td>
<td>Between 20 and 60</td>
<td>20</td>
</tr>
</tbody>
</table>

The packet length is 20 milliseconds and corresponds to a packet size of 800 bits i.e. 100 bytes.

Structure of a packet:

Each packet sent contains:

- The ID of the sender
- The ID of the destination node
- The next hop on the route
- The data

The ID of the sender, the ID of the destination node and the next hop on the route are contained in the header of the message.

Parameters of the algorithm:

Data were sent every 5 seconds (200000 bit time). Maintenance agents were sent every 25 seconds (1000000 bit time) and discovery agents every 50 seconds (20000000 bit time). The time after which a route is considered bad is 50 seconds. If
after this time, no maintenance agents have been received for a given route, the route is considered to be bad and it is not used anymore by the sender.

The cost setup phase takes less than one second in average.

The time required to find a route towards an destination node is very important. It shows how fast a communication path towards an destination node can be found by a source node in order to use it for data routing.

In table 2, the round-trip time was computed i.e. the time composed of the time for the source node to reach the destination node and the time for the discovery agents sent by the destination node to give a corresponding acknowledgement and activate the route.

TABLE 2
ROUND TRIP TIME OVER 50 PACKETS SENT

<table>
<thead>
<tr>
<th>DA #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time(bit time)</td>
<td>7178</td>
<td>7734</td>
<td>9401</td>
<td>10624</td>
<td>11573</td>
<td>10305</td>
<td>15675</td>
<td>13290</td>
<td>10265</td>
<td>11216</td>
</tr>
<tr>
<td>Time(in ms)</td>
<td>179</td>
<td>193</td>
<td>235</td>
<td>267</td>
<td>289</td>
<td>258</td>
<td>392</td>
<td>332</td>
<td>257</td>
<td>280</td>
</tr>
</tbody>
</table>

Variety of routes:

A sender in the middle of the sensor network was considered. We observed then the variety of routes found by the algorithm from this node to the destination node. The algorithm found 24 routes. (Figures 19 and 20)
Figure 19: Variety of routes

Figure 20: Variety of routes and corresponding graph on simulator
Delay:

For a node located in the center of the field (cost of 4), we computed the time delay over the 50 first packets sent. (Table 3).

TABLE 3

DELYAS IN THE COMMUNICATION OVER THE 50 FIRST PACKETS

<table>
<thead>
<tr>
<th>Packet #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time delay(bit time)</td>
<td>X</td>
<td>X</td>
<td>7497</td>
<td>8258</td>
<td>7745</td>
<td>8279</td>
<td>X</td>
<td>7583</td>
<td>7414</td>
<td>8294</td>
</tr>
<tr>
<td>Time delay(ms)</td>
<td>X</td>
<td>X</td>
<td>187</td>
<td>206</td>
<td>194</td>
<td>207</td>
<td>X</td>
<td>190</td>
<td>185</td>
<td>207</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packet #</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time delay(bit time)</td>
<td>8384</td>
<td>7793</td>
<td>X</td>
<td>6832</td>
<td>X</td>
<td>7178</td>
<td>7472</td>
<td>7820</td>
<td>7443</td>
<td>8357</td>
</tr>
<tr>
<td>Time delay(ms)</td>
<td>210</td>
<td>195</td>
<td>X</td>
<td>171</td>
<td>X</td>
<td>179</td>
<td>187</td>
<td>196</td>
<td>186</td>
<td>209</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packet #</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay (bit)</td>
<td>7728</td>
<td>X</td>
<td>7051</td>
<td>8768</td>
<td>7994</td>
<td>X</td>
<td>8009</td>
<td>5043</td>
<td>4799</td>
<td>5562</td>
</tr>
<tr>
<td>Delay (ms)</td>
<td>193</td>
<td>X</td>
<td>176</td>
<td>219</td>
<td>200</td>
<td>X</td>
<td>200</td>
<td>120</td>
<td>120</td>
<td>139</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packet #</th>
<th>31</th>
<th>32</th>
<th>33</th>
<th>34</th>
<th>35</th>
<th>36</th>
<th>37</th>
<th>38</th>
<th>39</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay (bit)</td>
<td>561 7</td>
<td>5036</td>
<td>4449</td>
<td>5056</td>
<td>5701</td>
<td>5196</td>
<td>5390</td>
<td>X</td>
<td>5483</td>
<td>6042</td>
</tr>
<tr>
<td>Delay (ms)</td>
<td>142</td>
<td>126</td>
<td>111</td>
<td>126</td>
<td>143</td>
<td>130</td>
<td>135</td>
<td>X</td>
<td>137</td>
<td>151</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packet #</th>
<th>41</th>
<th>42</th>
<th>43</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
<th>48</th>
<th>49</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay (bit)</td>
<td>534 3</td>
<td>5630</td>
<td>5190</td>
<td>5734</td>
<td>4534</td>
<td>5540</td>
<td>5282</td>
<td>5881</td>
<td>5291</td>
<td>5195</td>
</tr>
<tr>
<td>Delay (ms)</td>
<td>134</td>
<td>141</td>
<td>130</td>
<td>143</td>
<td>113</td>
<td>139</td>
<td>132</td>
<td>147</td>
<td>132</td>
<td>130</td>
</tr>
</tbody>
</table>
A X in the table corresponds to a failure in the delivery of the packet to the destination node. Packet 28 corresponds to a switch to a best performing route in terms of delay ordered by the feedback mechanism.

Figure 21 shows the effect of the feedback mechanism on the route selection scheme. Data are first routed through a given route introducing an approximate average delay of 200ms. Then, data are routed through a new discovered route that performs better than the previous one (approximate average delay of 130ms).

Figure 21: Importance of the feedback mechanism
3.3.4 Discussion

To illustrate, consider an air-conditioning application. The period at which data need to be sent depends on the dynamics of the room mainly here the size of the room. The following values could be chosen:

- A sending period of 5 minutes for the data.
- A sending period of an hour for the maintenance agents.
- A sending period of 2 hours for the discovery agents.
- A route would be discarded if no maintenance agents corresponding to this route are received for a duration of 2 hours.

The period for the sending of data heavily depends on the characteristics of the plant that is to be controlled. The periods for the sending of maintenance agents and discovery agents mainly depend on the control performance one wants to reach using the wireless sensor network. In theory, the more often maintenance agents are sent, the more updated the information about the discovered route is. It would allow to the source node to use the best performing route at any time. In practice, an increase of the frequency at which discovery agents are sent might also lead to an increase of collisions and so to a deterioration of the performance of the wireless networked control system. One has to select these parameters empirically in order to achieve acceptable performance.

Another interesting aspect of the algorithm is the average recovery time i.e. the average time needed for a source node to find a new route towards a given destination node after a previous route becomes useless because of a node failure. An upper bound
for the time needed to discover a route failure by a given sender due to a node failure is a parameter of the algorithm.

Perhaps more importantly, this work gives an insight about what an adequate algorithm for wireless networked control systems could be.
CHAPTER 4

CONCLUSION AND FUTURE WORK

Routing algorithms are of significant importance to wireless sensor networks and wireless networked control systems since they have direct influence on their performance.

In this thesis, three novel routing algorithms for wireless sensor networks and wireless networked control systems were presented.

Energy efficiency in wireless sensor networks is a major field of study. First, a new approach in building multiple backbones in wireless sensor networks was described. Rather than making sure that every node is connected to a backbone, the algorithm works in a probabilistic way. Simulations made using the ISIS developed simulator JProwler showed that the algorithm achieves good connectivity results for dense wireless sensor networks. Future work would be to implement the algorithm on a real test bed and see how it behaves.

The second algorithm is a power aware routing algorithm. Simulations were made using a customized simulator and showed that it reached a gain of an order of magnitude in node lifetime extension.
The emergence of wireless networked control industrial applications, in the industry such as air conditioning applications, shows the actual importance of wireless networked control systems. Wireless networked control is a new and fast evolving area in which significant research still needs to be done on adaptive routing algorithms.

The main contribution of this work was twofold. First, it analyzed the main characteristics of wireless networked control systems in terms of communication requirements and performance criteria. Second, it described a novel routing algorithm that satisfies such communication and performance criteria.

Future work may include the implementation on hardware and, under realistic conditions, to test the validity of the algorithm before it can be deployed.
APPENDIX

Code for the algorithm for finding multiple backbones in ad-hoc wireless sensor networks

```java
package net.tinyos.prowler;
import java.awt.Color;
import java.awt.Graphics;
import java.text.DecimalFormat;
import java.util.Random;
public class ManuApplication extends Application {
    static int nconnected, noveralltime, nmessages;
    static int neighbors;
    static Simulator sim;
    static Random rand = new Random();
    static final int number_colors = 2;
    int[] counter = new int[number_colors];
    int code = 0;
    int msg;
    boolean decided = false;
    boolean wait4event = false;
    int decide_extensions = 8;
    int waiting_time = 150;
    int iterations = 4;
    boolean connected = false;
    private String disptxt = "";
    static int decision_delay = 50; // in milliseconds
    static int METHOD = 3;
    long sendbegintime;
    private static int genVariableWaitTime() {
        int d;
        d = decision_delay - decision_delay / 2 + rand.nextInt(decision_delay);
        return d * Simulator.ONE_SECOND / 1000;
    }
    event_occured myDecideEvent = new event_occured();
    public ManuApplication(Node node) {
        super(node);
    }
    public class event_occured extends Event {

```
public void execute() {
    Mica2Node mNode = (Mica2Node) node;
    // int j=0;
    int minimum = 100;

    if (decided == false) {
        // find minimum != 0
        for (int i = 0; i < number_colors; i++) {
            if (counter[i] == 0) {
                if (decide_extensions > 0) {
                    this.time = sim.getSimulationTime() + genVariableWaitTime();
                    decide_extensions--;
                    sim.addEvent(this);
                    return;
                } else if (counter[i] < minimum)
                    minimum = counter[i];
            } else if (counter[i] != 0)
                if (counter[i] < minimum)
                    minimum = counter[i];
        }

        // count number of colors which have minimum
        int candidates = 0;
        for (int i = 0; i < number_colors; i++)
            if (counter[i] == minimum) candidates++;

        // select randomly one of the colors with minimum count
        int j = -1, r = rand.nextInt(candidates);
        for (int i = 0; i <= r; i++) {
            j++;
            while(counter[j] != minimum) j++;
        }
        code = j + 1;
        for (int i = 0; i < number_colors; i++) {
            neighbors = neighbors + counter[i];
        }
    }

    decided = true;
}

public void receiveMessage(Object message, Node sender) {
    int receivedmsg = Integer.parseInt(message.toString()) % 1000;
    if (receivedmsg == 0)
for (int j = 0; j < number_colors; j++)
    counter[j]++;
else
    counter[receivedmsg - 1]++;

if (connected == false) {
    boolean flag = true;
    for (int i = 0; i < number_colors; i++) {
        if (counter[i] <= 0)
            flag = false;
    }

    if (flag == true) {
        connected = true;
        nconnected++;
    }
}

if (decided == false) {
    if (receivedmsg == 55) {
        code = 1 + rand.nextInt(number_colors);
        msg = code + 1000 * node.id;
        sendMessage(String.valueOf(msg));
        nmessages++;
        decided = true;
    } else if (wait4event == false) {
        myDecideEvent.time = sim.getSimulationTime() + genVariableWaitTime();
        sim.addEvent(myDecideEvent);
        wait4event = true;
    }
}

if (decided == true) {
    if (connected == false) {
        if (iterations > 0) {
            msg = code + 1000 * node.id;
            sendMessage(String.valueOf(msg));
            nmessages++;
            iterations--;
        }
    }
}

public void display(Display disp) {
    Color[] nodecolor = new Color[] { Color.black, Color.magenta, Color.green,
    Color.orange, Color.gray, Color.cyan };
    Graphics g = disp.getGraphics();
    int x = disp.x2ScreenX(node.x);
    int y = disp.y2ScreenY(node.y);
    int displaytype=2;
}
Mica2Node mNode = (Mica2Node) node;

g.setColor(Color.black);

if (mNode.noiseStrength > 0.0125) {
    g.setColor(Color.red);
    DecimalFormat df = new DecimalFormat("#.00");
    g.drawString(df.format(mNode.noiseStrength), x + 5, y + 10);
}

g.setColor(Color.black);
// g.drawString(disptxt, x + 5, y);
// g.drawString(counter1+:"+counter2, x + 5, y);

/*
 * if( sending ){ g.setColor( Color.blue ); }else if( receiving ){ if(
 * corrupted ) g.setColor( Color.red ); else g.setColor( Color.green );
 * }else{ if( sent ) g.setColor( Color.pink ); else g.setColor(
 * Color.black ); }
 */
if (mNode.id == 1) {
    g.setColor(Color.black);
    g.fillRect(x - 6, y - 7, 11, 13);
    g.setColor(Color.white);
    g.drawString("B", x-5, y+4);
    return;
}

if (mNode.sending) {
    g.setColor(Color.red);
    g.fillOval(x - 4, y - 4, 7, 7);
}

switch(displaytype) {
    case 0: //coloring
        if (connected) {
            g.setColor(Color.black);
            g.drawRect(x - 7, y - 7, 13, 13);
            // g.fillOval( x-4, y-4, 7, 7 );
        }
        g.setColor(nodecolor[code]);
        g.fillOval(x - 3, y - 3, 5, 5);
        break;
    case 1: //numbers
        if (connected) {
            g.setColor(Color.black);
            g.drawRect(x - 7, y - 7, 13, 13);
            // g.fillOval( x-4, y-4, 7, 7 );
        }
        g.drawString(String.valueOf(code), x-3, y+4);
        g.setColor(Color.black);
        break;
    case 2: // filled/not filled rectangles
        g.setColor(Color.black);
        break;
}
if (connected) {
    switch(code) {
    case 1:
        g.drawRect(x - 5, y - 5, 9, 9);
        break;
    case 2:
        g.fillRect(x - 5, y - 5, 9, 9);
        break;
    }
} else {
    switch(code) {
    case 0:
        g.fillOval( x, y, 1, 1 );
        break;
    case 1:
        g.drawOval( x-4, y-4, 7, 7 );
        break;
    case 2:
        g.fillOval( x-4, y-4, 7, 7 );
        break;
    }
}

/ *
* if (sent) { g.drawLine(x - 5, y - 5, x + 5, y + 5); g.drawLine(x + 5, * y - 5, x - 5, y + 5); }
*/

/ *
* if ( parent != null ){ g.setColor( Color.black ); int x1 =
* disp.x2ScreenX(parent.getX()); int y1 =
* disp.y2ScreenY(parent.getY()); g.drawLine(x,y,x1,y1); }
*/

/**
 * Starts up a simulator with a ROOT in the middle of a 300 by 300 meters
 * field with 1000 motes and runs it in real time mode.
 *
 * @param args
 * @throws Exception
 */
public static void main(String[] args) throws Exception {
    long time0 = System.currentTimeMillis();
    int nmotes = 80, fieldsize =100, simruns = 100;
    final boolean realTime = false, withDisplay =false;
    boolean grid=false;
    nmessages=0;
    nconnected = 0; // count number of connected nodes
    noveralltime = 0; // number of overall timer ticks
    neighbors=0; //number of neighbors
System.out.println("Simulating (" + nmotes + " motes on a " + fieldsize + "x" + fieldsize + " field, using method " + METHOD + ")");
System.out.print("Thinking");
for (int j = 1; j <= simruns; j++) {
    if (simruns >= 10)
        if (j % (simruns / 10) == 0)
            System.out.print(".");
    sim = new Simulator();
    // creating the desired radio model, uncomment the one you need
    // RayleighRadioModel radioModel = new RayleighRadioModel(sim);
    GaussianRadioModel radioModel = new GaussianRadioModel(sim);

    Mica2Node root = (Mica2Node) sim.createNode(Mica2Node.class, radioModel, 1, fieldsize / 2, fieldsize / 2, 0);
    // root.visited = true;
    ManuApplication baseApp = new ManuApplication(root);
    baseApp.decided = true;
    // base node is always connected
    if(grid){
        int k=2;
        //System.out.println("nline=");
        int nline=(int)Math.sqrt(nmotes);
        //System.out.println("nline=+nline");
        float step=(float)fieldsize/nline;
        for (float x = step/2; x <= fieldsize; x+=step)
            for (float y = step/2; y <= fieldsize; y+=step){
                Mica2Node tempNode = (Mica2Node)
                    sim.createNode(Mica2Node.class, radioModel,k, x, y,0);
                new ManuApplication(tempNode);
                k=k+1;
            }
    }
    else{
        // creating all the other nodes
        Node tempNode = sim.createNodes(Mica2Node.class, radioModel, 2, nmotes, fieldsize, 5);
        while (tempNode != null) {
            new ManuApplication(tempNode);
            tempNode = tempNode.nextNode;
        }
    }
    // This call is a must, please do not forget to call it whenever the
    // mote field is set up
    radioModel.updateNeighborhoods();
    root.sendMessage("0", baseApp);
    nmessages++;

    if (withDisplay) {
        if (realTime)
            sim.runWithDisplayInRealTime();
        else
            sim.runWithDisplay();
    } else {
        //
if (realTime)
    System.out.println("Will not do real-time without display, switching to non-rt...");

    sim.run(20000);
}

while (!sim.endOfSimulation())
    System.out.println("waiting");
    // Thread.sleep(200);

    noveralltime += sim.getSimulationTimeInMillisec();
} // next j

DecimalFormat df = new DecimalFormat("0.0");
System.out.println("(" + df.format((float) (System.currentTimeMillis() - time0) / 1000)
    + " s")");
float success = (float) nconnected / (nmotes * simruns);
int success_pc = (int) (success * 100);

System.out.println("Successfully connected nodes: " + success_pc + ";");
System.out.println("Average algorithm running time: "+ df.format((float) noveralltime / (simruns * 1000)) + " s");
System.out.println("Average Number of messages exchanged: "+(float) nmessages/simruns + "messages");
System.out.println("Average Number of neighbors: " +(float) neighbors/(simruns*nmotes) + "neighbors");
}

Code for the algorithm for wireless networked control systems

SENDER_ID=sim_params('get_app', 'Start_Mote');
if isempty(SENDER_ID), SENDER_ID=1; end

switch event
    case 'Init_Application'
        signal_strength=1;
    %%% Memory should be initialized here %%%
    memory=struct('gradient',1000,'IDBS',0,'ID',0,'updated',0,'counter',4,'number',0,'owngrad1',0,
        'owngrad2',0,'neigrad1',0, 'neigrad2',0, 'neigrad3',0,
        'neigrad4',0,'neigrad5',0,'neigrad6',0,'neigrad7',0,'neigrad8',0,'neigrad9',0,'neigrad10',0,
        'signal_strength',
        signal_strength,'receiveID',0,'discoverID',0,'discoverID1',0,'discoverID2',0,'discoverID3',0,'discoverID4',0,'discoverID5',0,'receiveID1',0,'receiveID2',0,'receiveID3',0,'receiveID4',0,'receiveID5'
if ID==SENDER_ID % this node starts flood
    memory.gradient=0;
    memory.IDBS=memory.ID;
    msg.type=0;
    msg.gradient=0;
    msg.ID=memory.ID;
    msg.IDBS=memory.ID;
    PrintMessage('BS')
    Send_Packet(radiostream(msg, memory.signal_strength));
end

if(ID==56)
    Set_Clock(80000)
end

if ID==SENDER_ID
    Set_Clock(10000)
end

case 'Packet_Sent'
case 'Packet_Received'
msg=data.data; % message
if(msg.type==0)
    if(memory.gradient>msg.gradient+1)
        memory.gradient=msg.gradient+1;
        memory.updated=1;
    end
    if(ID~=1)
        PrintMessage(memory.gradient);
    end
    hood=memtotable(memory)
    updatedhood=neigradtable(hood,msg);
    memory=tabletomem(updatedhood,memory);
    memory.IDBS=msg.IDBS;
    msg.gradient=memory.gradient;
    msg.ID=memory.ID;
    msg.type=0;
    msg.IDBS=memory.IDBS;
    if(memory.counter>0)
        memory.counter=memory.counter-1;
        Send_Packet(radiostream(msg, memory.signal_strength));
    end
end
elseif(msg.type==1)
    if(ID==memory.IDBS&msg.discoverID==ID)
        memory.receiveID=msg.ID;
        memory.number=msg.number;
        memory=returntomem(memory);
        DrawLine('arrow', memory.receiveID, ID, 'color', [0 0 0])
        memory.timeofdeparture=msg.time;
        memory.timeofarrival=t;
        memory.delay=memory.timeofarrival-memory.timeofdeparture;
        PrintMessage(memory.delay);
        msg.type=2;
        memory.colorvector=[rand rand rand];
        msg.colorvector=memory.colorvector;
        msg.receiveID=memory.receiveID;
        PrintMessage(msg.number)
        Send_Packet(radiostream(msg, memory.signal_strength));
    end

if(msg.discoverID==ID&ID~1)
    if(memory.number~=msg.number|memory.receiveID==0)
        memory.receiveID=msg.ID;
    end
    DrawLine('arrow', memory.receiveID, ID, 'color', [0 0 0])
    memory.number=msg.number;
    memory=returntomem(memory);
    tab=memtotable(memory);
    memory=findroute(tab,memory);
    msg.ID=ID;
    msg.discoverID=memory.discoverID;
    msg.type=1;
    Send_Packet(radiostream(msg, memory.signal_strength));
elseif(msg.type==2)
    if(msg.receiveID==ID&ID~56)
        memory.number=msg.number;
        memory.colorvector=msg.colorvector;
        DrawLine('arrow', memory.discoverID, ID, 'color', memory.colorvector);
        memory=maintenanceroutes(memory);
        msg.type=2;
        msg.receiveID=memory.receiveID;
        Send_Packet(radiostream(msg, memory.signal_strength));
    end
    if(ID~56&msg.receiveID==ID)
        DrawLine('arrow', memory.discoverID, ID, 'color', memory.colorvector);
        memory=perf(msg,memory)
memory.timeofarrival=t;
% memory.timeofdeparture=msg.time;
memory.delay=memory.timeofarrival-memory.timeofdeparture;
PrintMessage(memory.delay);
end
elseif(msg.type==3)
if(msg.discoverID==ID&ID~=1)
memory.receiveID=msg.ID;
memory.routenumber=msg.routenumber;
DrawLine('arrow', memory.receiveID, ID, 'color', [0 0 1])
memory=sendingdata(memory)
msg.ID=ID;
msg.routenumber=memory.routenumber;
msg.discoverID=memory.discoverID;
msg.type=3;
Send_Packet(radiostream(msg, memory.signal_strength));
end
if(ID==1&msg.discoverID==ID)
memory.deliverysuccess=memory.deliverysuccess+1;
memory.receiveID=msg.ID;
% memory.timeofarrival=t;
% memory.timeofdeparture=msg.time;
% memory.delay=memory.timeofarrival-memory.timeofdeparture;
% PrintMessage(memory.delay)
DrawLine('arrow', memory.receiveID, ID, 'color', [0 0 1])
end
end

case 'Collided_Packet_Received'
% this is for debug purposes only

case 'Clock_Tick'
if(ID==56)
Set_Clock(t+10000)
if( memory.performance1==0)
memory.performance1=memory.performance1-1;
end
if( memory.performance2==0)
memory.performance2=memory.performance2-1;
end
if( memory.performance3==0)
memory.performance3=memory.performance3-1;
end
if( memory.performance4==0)
memory.performance4=memory.performance4-1;
end
if( memory.performance5==0)
memory.performance5=memory.performance5-1;
end
% PrintMessage(memory.performance1)

if(memory.timerDA==0)
    memory.number=memory.number+1;
    memory=routenumber(memory);
    memory.timerDA=200;
    % PrintMessage('56');
    tab=memtotable(memory);
    memory=findroute(tab,memory);
    msg.ID=ID;
    msg.discoverID=memory.discoverID;
    msg.type=1;
    msg.number=memory.number;
    memory.timeofdeparture=t;
    Send_Packet(radiostream(msg,memory.signal_strength));
    % PrintMessage('Hello')
    % PrintMessage(msg.number)
else
    memory.timerDA=memory.timerDA-1;
end

% if(memory.sendingcounter==0)
%     memory.sendingcounter=20;
%     memory=sending(memory)
% if(memory.routenumber==0)
%     msg.type=3;
%     msg.ID=ID;
%     msg.discoverID=memory.discoverID;
%     % PrintMessage(memory.discoverID)
%     msg.routenumber=memory.routenumber;
%     memory.datanumber=memory.datanumber+1
%     msg.time=t;
%     Send_Packet(radiostream(msg,memory.signal_strength));
% end
% else
%     memory.sendingcounter=memory.sendingcounter-1;
% end
end

% if(ID==SENDER_ID)
% Set_Clock(t+10000)
% memory.timerMA=memory.timerMA-1;
% if(memory.timerMA==0)
%     memory.MA=1;
% end
% if(memory.MA==1)
%     memory.timerMA=100;
%     memory.MAcounter=memory.MAcounter+1;
% if(memory.MAcounter<5)
%     memory=maintenance(memory);
% if(memory.receiveID==0)
% msg.type=2;
% msg.colorvector=memory.colorvector;
% msg=maintenanceroute(memory,msg);
% % PrintMessage(msg.number)
% msg.receiveID=memory.receiveID;
% %
% % PrintMessage(msg.receiveID)
% msg.MA=1;
% Send_Packet(radiostream(msg, memory.signal_strength));
% end
% elseif(memory.MAcounter==5)
% memory=maintenance(memory);
% if(memory.receiveID==0)
% msg.type=2;
% msg.colorvector=memory.colorvector;
% msg.number=memory.number5;
% msg.receiveID=memory.receiveID;
% Send_Packet(radiostream(msg, memory.signal_strength));
% end
% memory.MA=0;
% memory.MAcounter=0;
%
% end
%
% end
%

case 'GuiInfoRequest'
    if ~isempty(memory)
        disp(sprintf('Memory Dump of mote ID# %d:\n',ID)); disp(memory)
    else
        disp(sprintf('No memory dump available for node %d.\n',ID));
    end

case 'Application_Stopped'
    % this event is called when simulation is stopped/suspended

case 'Application_Finished'
    % this event is called when simulation is finished

otherwise
    error(['Bad event name for application: ' event])
end

%%%%%%%%%%%%%%%%%%%%%%APPLICATION ENDS
%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%   HERE      %%%%%%%%%%%%%%%%%%%%%%%%
function b=Send_Packet(data);
global ID t  
radio=prowler('GetRadioName');
b=feval(radio, 'Send_Packet', ID, data, t);

function b=Set_Clock(alarm_time);
global ID 
prowler('InsertEvents2Q', make_event(alarm_time, 'Clock_Tick', ID));

function PrintMessage(msg)
global ID 
prowler('TextMessage', ID, msg)

function DrawLine(command, varargin)
switch lower(command)
    case 'line'
        prowler('DrawLine', varargin{:})
    case 'arrow'
        prowler('DrawArrow', varargin{:})
    case 'delete'
        prowler('DrawDelete', varargin{:})
    otherwise
        error('Bad command for DrawLine.')</nend

function tab=neigradtable(mytab,mymsg)
randompick=ceil(10*rand);
flag=0;
mgradient=mymsg.gradient+1;
for(i=1:10)
    if(flag==0)
        if(mymsg.IDBS==findIDBS(mytab(i))&mymsg.ID==findID(mytab(i)))
            if(mgradient<findgradient(mytab(i)))
                flag=1;
                mytab(i)=mymsg.IDBS*10^5+10^3*mymsg.ID+mymsg.gradient;
            end
        end
    end
if(flag==0)
    for(i=randompick:10)
        if(flag==0)
            if(mytab(i)==0)
                flag=1;
                mytab(i)=mymsg.IDBS*10^5+10^3*mymsg.ID+mymsg.gradient;
                end
            end
        end
    end
if(flag==0)
    for(i=1:randompick-1)
        if(flag==0)
            if(mytab(i)==0)
                flag=1;
                mytab(i)=mymsg.IDBS*10^5+10^3*mymsg.ID+mymsg.gradient;
                end
            end
        end
    end
    tab=mytab;

function mymemory=tabletomem(mytab,mymem)
mymem.neigrad1=mytab(1);
mymem.neigrad2=mytab(2);
mymem.neigrad3=mytab(3);
mymem.neigrad4=mytab(4);
mymem.neigrad5=mytab(5);
mymem.neigrad6=mytab(6);
mymem.neigrad7=mytab(7);
mymem.neigrad8=mytab(8);
mymem.neigrad9=mytab(9);
mymem.neigrad10=mytab(10);
mymemory=mymem;

function mytab=memtotable(mymem)
mytab=zeros(1,10);
mytab(1)=mymem.neigrad1;
mytab(2)=mymem.neigrad2;
mytab(3)=mymem.neigrad3;
mytab(4)=mymem.neigrad4;
mytab(5)=mymem.neigrad5;
mytab(6)=mymem.neigrad6;
mytab(7)=mymem.neigrad7;
mytab(8)=mymem.neigrad8;
mytab(9)=mymem.neigrad9;
mytab(10)=mymem.neigrad10;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function IDBS=findIDBS(owngrad)
IDBS=floor(owngrad/10^3)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function ID=finddowngrad(owngrad)
IDBS=floor(owngrad/100);
ID=floor(owngrad-IDBS*100);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ID=findID(neigrad)
IDBS=floor(neigrad/10^5)
ID=floor((neigrad-IDBS*10^5)/1000)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function gradient=findgradient(neigrad)
IDBS=floor(neigrad/10^5)
ID=floor((neigrad-IDBS*10^5)/100)
gradient=neigrad-IDBS*10^5-100*ID
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function mymemory=findroute(tab,mymem)
random=rand;
% PrintMessage(random)
randompick=ceil(10*rand);
flag=0;
if(random<=0.9&random>0.2)
   for(i=randompick:10)
      if(flag==0)
         if(tab(i)==0)
            if(findgradient(tab(i))-mymem.gradient<0)
               flag=1;
               mymem.discoverID=findID(tab(i));
               %                     PrintMessage(mymem.discoverID);
            end
         end
      end
end
end
if(flag==0)
    for(i=1:randompick-1)
        if(flag==0)
            if(tab(i)==0)
                if(findgradient(tab(i))-mymem.gradient<0)
                    flag=1;
                    mymem.discoverID=findID(tab(i));
                    % PrintMessage(mymem.discoverID);
            end
        end
    end
end

if(flag==0)
    for(i=randompick:10)
        if(flag==0)
            if(tab(i)==0)
                if(findgradient(tab(i))-mymem.gradient==0)
                    flag=1;
                    mymem.discoverID=findID(tab(i));
                    % PrintMessage(mymem.discoverID);
            end
        end
    end
end

if(flag==0)
    for(i=1:randompick-1)
        if(flag==0)
            if(tab(i)==0)
                if(findgradient(tab(i))-mymem.gradient==0)
                    flag=1;
                    mymem.discoverID=findID(tab(i));
                    % PrintMessage(mymem.discoverID);
            end
        end
    end
end

if(flag==0)
    for(i=randompick:10)
        if(flag==0)
            if(tab(i)==0)
                if(findgradient(tab(i))-mymem.gradient>0)
                    flag=1;
                    mymem.discoverID=findID(tab(i));
                    % PrintMessage(mymem.discoverID);
            end
        end
    end
end
if(flag==0)
    for(i=1:randompick-1)
        if(flag==0)
            if(tab(i)~=0)
                if(findgradient(tab(i))-mymem.gradient>0)
                    flag=1;
                    mymem.discoverID=findID(tab(i));
                    %                             PrintMessage(mymem.discoverID);
                end
            end
        end
    end
end
elseif(random>0.9)
    for(i=randompick:10)
        if(flag==0)
            if(tab(i)~=0)
                if(findgradient(tab(i))-mymem.gradient>0)
                    flag=1;
                    mymem.discoverID=findID(tab(i));
                    %                             PrintMessage(mymem.discoverID);
                end
            end
        end
    end
end
if(flag==0)
    for(i=1:randompick-1)
        if(flag==0)
            if(tab(i)~=0)
                if(findgradient(tab(i))-mymem.gradient>0)
                    flag=1;
                    mymem.discoverID=findID(tab(i));
                    %                             PrintMessage(mymem.discoverID);
                end
            end
        end
    end
end
if(mod(mymem.number,5)==1)
    mymem.discoverID1=mymem.discoverID;
elseif(mod(mymem.number,5)==2)
    mymem.discoverID2=mymem.discoverID;
elseif(mod(mymem.number,5)==3)
    mymem.discoverID3=mymem.discoverID;
elseif(mod(mymem.number,5)==4)
    mymem.discoverID4=mymem.discoverID;
elseif(mod(mymem.number,5)==0)
function mymemory=returntomem(mymem)
    % PrintMessage('Hello')
    if(mod(mymem.number,5)==1)
        mymem.receiveID1=mymem.receiveID;
        mymem.number1=mymem.number;
        % PrintMessage(mymem.number1)
    elseif(mod(mymem.number,5)==2)
        mymem.receiveID2=mymem.receiveID;
        mymem.number2=mymem.number;
        % PrintMessage(mymem.number2)
    elseif(mod(mymem.number,5)==3)
        mymem.receiveID3=mymem.receiveID;
        mymem.number3=mymem.number;
        % PrintMessage(mymem.number3)
    elseif(mod(mymem.number,5)==4)
        mymem.receiveID4=mymem.receiveID;
        mymem.number4=mymem.number;
        % PrintMessage(mymem.number4)
    elseif(mod(mymem.number,5)==0)
        mymem.receiveID5=mymem.receiveID;
        mymem.number5=mymem.number;
        % PrintMessage(mymem.number5)
    end
    mymemory=mymem;
end

function mymemory=maintenanceroutes(mymem)
if(mod(mymem.number,5)==1)
    mymem.receiveID=mymem.receiveID1;
elseif(mod(mymem.number,5)==2)
    mymem.receiveID=mymem.receiveID2;
elseif(mod(mymem.number,5)==3)
    mymem.receiveID=mymem.receiveID3;
elseif(mod(mymem.number,5)==4)
    mymem.receiveID=mymem.receiveID4;
elseif(mod(mymem.number,5)==0)
    mymem.receiveID=mymem.receiveID5;
end

mymemory=mymem;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mymemory=routenumber(mymem)
if(mod(mymem.number,5)==1)
    mymem.number1=mymem.number;
    % printf("mymemory.number1 = ",mymem.number1);
elseif(mod(mymem.number,5)==2)
    mymem.number2=mymem.number;
    % printf("mymemory.number2 = ",mymem.number2);
elseif(mod(mymem.number,5)==3)
    mymem.number3=mymem.number;
    % printf("mymemory.number3 = ",mymem.number3);
elseif(mod(mymem.number,5)==4)
    mymem.number4=mymem.number;
    % printf("mymemory.number4 = ",mymem.number4);
elseif(mod(mymem.number,5)==0)
    mymem.number5=mymem.number;
    % printf("mymemory.number5 = ",mymem.number5);
end

mymemory=mymem;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mymemory=perf(mymsg,mymem)
if(mod(mymsg.number,5)==1&mymsg.number==mymem.number1)
    mymem.performance1=200;
    % printf("mymemory.performance1 = ",mymem.performance1);
elseif(mod(mymsg.number,5)==2&mymsg.number==mymem.number2)
    mymem.performance2=200;
elseif(mod(mymsg.number,5)==3&mymsg.number==mymem.number3)
    mymem.performance3=200;
elseif(mod(mymsg.number,5)==4&mymsg.number==mymem.number4)
    mymem.performance4=200;
elseif(mod(mymsg.number,5)==0&mymsg.number==mymem.number5)
    mymem.performance5=200;
end

mymemory=mymem;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mymemory=routenumber(mymem)
if(mod(mymem.number,5)==1)
    mymem.number1=mymem.number;
elseif(mod(mymem.number,5)==2)
    mymem.number2=mymem.number;
elseif(mod(mymem.number,5) == 3)
    mymem.number3 = mymem.number;
elseif(mod(mymem.number,5) == 4)
    mymem.number4 = mymem.number;
elseif(mod(mymem.number,5) == 0)
    mymem.number5 = mymem.number;
end

mymemory = mymem;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mymemory = maintenance(mymem)
    if(mymem.MAcounter == 1)
        mymem.receiveID = mymem.receiveID1;
    elseif(mymem.MAcounter == 2)
        mymem.receiveID = mymem.receiveID2;
    elseif(mymem.MAcounter == 3)
        mymem.receiveID = mymem.receiveID3;
    elseif(mymem.MAcounter == 4)
        mymem.receiveID = mymem.receiveID4;
    elseif(mymem.MAcounter == 5)
        mymem.receiveID = mymem.receiveID5;
    end

    % PrintMessage(mymem.receiveID)
    mymemory = mymem;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mymessage = maintenanceroute(mymem, mymsg)
    if(mymem.MAcounter == 1)
        mymsg.number = mymem.number1;
    elseif(mymem.MAcounter == 2)
        mymsg.number = mymem.number2;
    elseif(mymem.MAcounter == 3)
        mymsg.number = mymem.number3;
    elseif(mymem.MAcounter == 4)
        mymsg.number = mymem.number4;
    elseif(mymem.MAcounter == 5)
        mymsg.number = mymem.number5;
    end

    mymessage = mymsg;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mymemory = colorroute(mymem)
    if(mymem.number == 1)
        mymem.color = [1 0 0];
    elseif(mymem.number == 2)
        mymem.color = [0 1 0];
    elseif(mymem.number == 3)
        mymem.color = [0 0 1];
elseif (mymem.number == 4)
    mymem.color = [1 1 0];
elseif (mymem.number == 5)
    mymem.color = [1 0 1];
end

mymemory = mymem;

function mymemory = sending (mymem)

    randompick = ceil (5 * rand);
    tab = zeros (1, 5);
    tab(1) = mymem.performance1;
    tab(2) = mymem.performance2;
    tab(3) = mymem.performance3;
    tab(4) = mymem.performance4;
    tab(5) = mymem.performance5;
    line = 0;
    flag = 0;
    routeflag = 0;
    for (i = randompick:5)
        if (flag == 0)
            if (tab(i) > 0)
                routeflag = 1;
                flag = 1;
                line = i;
                % PrintMessage(line)
            end
        end
    end
    if (flag == 0)
        for (i = 1:randompick-1)
            if (flag == 0)
                if (tab(i) > 0)
                    routeflag = 1;
                    flag = 1;
                    line = i;
                end
            end
        end
    end
    if (routeflag == 0)
        % PrintMessage('no routes available')
    end
    if (line == 1)
        mymem.discoverID = mymem.discoverID1;
    elseif (line == 2)
        mymem.discoverID = mymem.discoverID2;
    elseif (line == 3)
        mymem.discoverID = mymem.discoverID3;
    elseif (line == 4)
        mymem.discoverID = mymem.discoverID4;
    elseif (line == 5)
  mymem.discoverID=mymem.discoverID5;
  end
  mymem.routenumber=line;
  mymemory=mymem;

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  function mymemory=sendingdata(mymem)
  if(mymem.routenumber==1)
    mymem.discoverID=mymem.discoverID1;
  elseif(mymem.routenumber==2)
    mymem.discoverID=mymem.discoverID2;
  elseif(mymem.routenumber==3)
    mymem.discoverID=mymem.discoverID3;
  elseif(mymem.routenumber==4)
    mymem.discoverID=mymem.discoverID4;
  elseif(mymem.routenumber==5)
    mymem.discoverID=mymem.discoverID5;
  end
  mymemory=mymem;
REFERENCES


