Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group

Multi Hop Send Protocol Tool for TinyNodes

Semesterthesis

Author:	Remo Niedermann
Supervisor:	Prof. Dr. Roger Wattenhofer
Tutor:	Roland Flury

Zurich, February 19, 2009

Acknowledgment

At first I want to thank Prof. Dr. Roger Wattenhofer for the possibility to write my semester thesis at his institute at the Distributed Computing Group of the Swiss Federal Institute of Technology (ETH) Zurich.
Further I am very grateful to Roland Flury because for his continuous help and support as supervisor during the whole semester thesis.

Contents

1. Introduction

1.1. Motivation

Sensor node systems became very popular the last few years. They are used to gather data over a designated area for instance parking space in a town or temperature monitoring in the alps. These systems have in common that the network consists of small computers with small memory and power consumption. The reason for using these small computers is that usually the nodes do not have to compute a lot. The main processing of the measured data is done centrally on powerful machines. As a result, the nodes only need to measure, store the data for a short time, and forward the measurements to a centralized server.
Measuring, storing, processing, and sending over radio needs power. Usually the nodes run on battery because there can be places without power supply installed or it is too expensive to install one (like in the alps). In case the network consists of hundreds of nodes it is inefficient to exchange the battery every two or three days. Therefore the algorithm needs to be power efficient.
The task of this thesis is to find an algorithm to send a command to one or more nodes and getting a response back. The response can be measurements, information about the network, the states of the node, etc. The algorithm should use as little memory and as few messages as possible. Until now the protocol running on the TinyNodes (type of nodes used in this semester thesis) is a kind of static source routing. In this case the routing information is stored in the message and the user must have full knowledge of the network topology because there is no dynamic routing table. This means that the user has to know how the nodes are connected. The problems and limitations of the existing situation are:

- It is very difficult to have the network topology of large networks in mind.
- It is hard to type the whole routing information into the packet.
- There is no mechanism for dynamic links.

- The message has fixed size, which means that the message has a limited range to get to the destination.

These four problems should be eliminated with the Multi Hop Send Protocol (MHSP). The MHSP will consists of a routing table which will be updated from time to time. The algorithm for MHSP should use a small amount of memory and as few messages as possible. In terms of power consumption the algorithm should be divided into time slots. The reason is that there are slots for creating and updating the routing table and slots for sending the data. Outside of these slots the node is idle and can be asleep (in a low power mode) to save power and extend the battery lifetime.

2. Implementation of the Multi Hop Send Protocol

The Multi Hop Send Protocol (MHSP) will be used to send commands from the root to the nodes and send the gathered information to the root. (root-to-„given-node“ and all-to-root communication). There will be no any-to-any communication. Therefore the network can be a tree, on which the messages are forwarded.
Each node should look for a „good“ parent. A good parent can be one which is next to the node or one which has not that many children yet, etc. The MHSP should handle dynamics. Nodes can join and leave the network or links can fail. Therefore the parents should be updated about changes in the tree.

2.1. Requirements

The requirements for the user are:

1. This project bulds on the TinyNode Platform
2. Latest installation of the TinyOS and Mytos

For the MHSP algorithm the requirements are:

1. The implementation should require as less memory as possible because the Tiny-Node has small memory which should be used for other programs also.
2. The creation of the routing table should have a minimal number of messages.
3. The algorithm should work within small time slots. Outside the slots the node is idle and in a low power mode to save power.

2.2. Algorithm

The algorithm will work on a network consisting of a root node which is connected to a computer, and nodes distributed over an area (not all nodes can hear the root). The communication will be from the root node which sends commands to one designated node or to all nodes, and the response from the nodes back to the root node. Therefore the messages need be routed over multiple hops. The simplest way to build the routes, is to create a tree and route the messages along these edges. For the root this means that all nodes it hears and can talk to, are its children. The child stores the parent and how far way, number of hops (hop level), it is from the root node, and also the children it is connected to. The tree is setup when all nodes have a parent and know their hop level. To forward a message to the next node, the node should know all the nodes which are in its subtree. The node has a Routing Table containing all children (Children List) and the routing entries (Routing List). Upon start up, this list is empty and the node asks the child to send its Routing Table. The node sets a relation between the child and the node its hears. With this information the node knows where to forward a message coming from the parent.

If a message arrives at the node, it checks if it is a message for it or for another node. In case the message is for another node the node asks the Routing Table if it has an entry with the destination. The Routing Table returns the ID of the next hop (which is a child). Then the node determines which sequence number it uses for the next hop. The sequence number is needed to detect the loss of a message. If all messages arrive correctly the sequence number always increases by one. After determining the next hop and the sequence number the node forwards the message. The node which gets the message sends back an acknowledgment with the same sequence number, such that the sending node knows that the message arrived at the next hop.

Usually a network is not static. Over time there can be changes in the network. Nodes can enter the network, others leave or crash (empty battery), or they move from time to time. This dynamics must be taken into account. For the algorithm this means that there will be changes in the Routing Table. Before the Routing Table changes, the changes need to be detected. This is done by sending an Information Message (see Chapter 2.3.2.1) which contains the parent ID and hop level. The nodes get this messages and handle them. The node stores a Received Message Counter for each child and parent. The Received Message Counter is one byte long. At the beginning of each cycle of the algorithm, the Received Message Counter is shifted by one bit. If the Information Message from a node or parent arrives, the node sets the LSB bit of the Receive Message Counter to one. This way it can be determined how many times the child or parent is heard during the last eight cycles. (10110111 child hear six times during eight cycles). After each cycle the node checks if the Received Message Counter of a child fell beneath five. If it did the child gets removed. If it is the counter from the parent, the node exchanges its parent. If something is changed in the routing table the variable „number of changes“ is changed. From the Information Message, which contains this „number of changes“ variable, the parent can determine the child which has updates and ask the child to send its updates to the parent. Any modification in the nodes Routing Table updates also the „number of changes“ variable such that the new topology is propagated towards the root.

2.3. Implementation

The main goal of MHSP is to setup and update the routing table of each node. It is important that it does not take to much time to build and update the routing table. Secondly, it should use as little memory and as few messages as possible to set it up. This implementation is based on passing messages over the radio between nodes containing a command and a payload which starts executing some code on the receiving node. The algorithm is synchronous.

At the beginning all nodes should have a unique identification number (ID). The node with the the ID=1 must be attached to the computer over a serial cable and is the root node. The commands are entered on a terminal running the JAVA interface Remote Control.

2.3.1. Architecture of the Protocol

The protocol itself consists of two main modules: The MHSP and the Routing Table. The task of the Routing Table is to store and update the children and the routing entries, give information about the entries and check that the entry does not show up twice. Whereas the goal of MHSP is to send the messages to build and update the Routing Table, getting a parent and update it and handle the messages which come from an other node or module. The last task includes also to check if the message is for this node and forward it to the required module or to forward it to the next node.

2.3.1.1. MHSP Module

The MHSP is responsible to select a parent, create and update the Routing Table and handle the messages coming from the serial or radio. The MHSP has a list of possible parents (Parent List) where the first entry is the real parent of the node. The other par-

ents, called alternative parents, are nodes that this node hears. The alternative parents are used for a fast change of the actual parent. This happens when the node loses the connection to his parent or the parent removes the node as his child. The Parent List consists of the ID of the node, the hop level count and a counter to detect how often the parent or alternative parent was heard during the last eight slots (Received Message Counter) (See figure 2.1). At the startup the node takes the first node it hears as its parent. All the following nodes are put into the Parent List, as long as there is space.
After the selection of a parent the node starts to send an Information Message (See

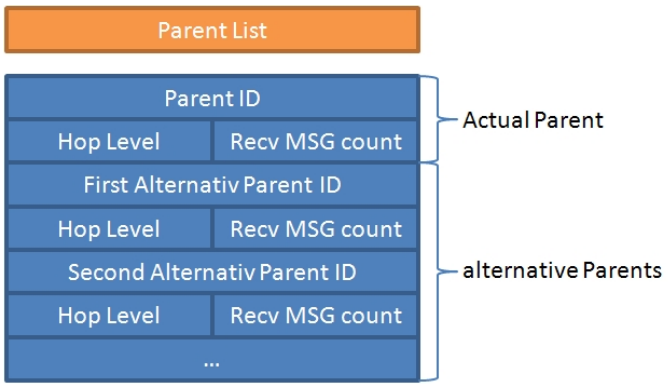


Figure 2.1.: Parent List

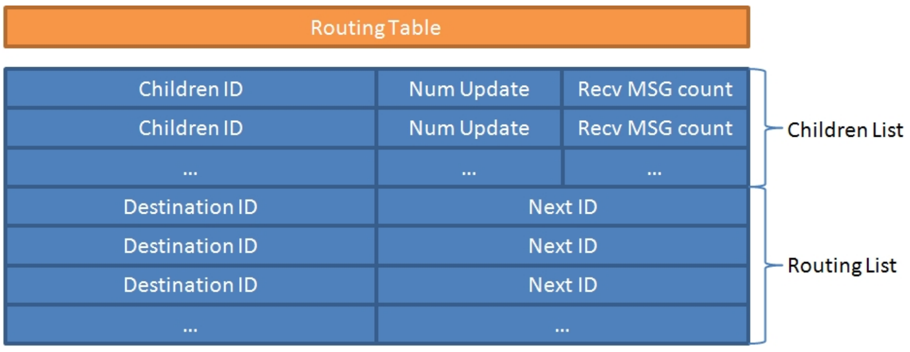
Chapter 2.3.2.1) which it also receives from other nodes in its neighboring. From these messages, a node can determine which neighboring nodes wish to use it as parent node. The IDs of these nodes are stored in the Children List (See Chapter 2.3.1.2).
The parent gets replaced by an alternative parent if a node does not receive the Information Message at least five times during the last eight most recent slots. In this case the parent is replaced by the best alternative parent. The rule for choosing a new parent from the alternative parents is based on the „hop level cound “of the alternative parent and how often the node was heard during the past eight slots. Before changing the parent, the child send a request (see Appendix A.1) to the new parent. If the parent responds (see Appendix A.1) the child changes the parent and the new parent adds the new child to its Children List.
Apart from connecting to a parent node, the MHSP also forwards its routing information towards the root node. For this purpose, the Information Message also contains a byte indicating the number of changes in its routing table.
In the Routing Table Updating Window (See Chapter 2.3.2.2), the parent node searches for children which have changes to report, retrieves the changes from the children and

updates its Routing Table accordingly. (see Chapter 2.3.1.2)

The third task for the MHSP and the reason why MHSP is implemented, is to handle packets and to forward them to the right node or module. If a node gets a message, the MHSP first checks if the message is for this node. If this is the case, it checks if it is a message for the MHSP or for an other module and react as expected. In case the packet is for an other node MHSP asks the Routing Table to get the ID of the next hop and forwards the message.
The problem is that the messages for the MHSP and the messages from other modules do not interfere with each other. Therefore the MHSP has two slots within 32 seconds (See chapter A.2). One slot for the MHSP itself to create the tree and to update the Routing Tables. The other slot is used to forward the messages from other modules. In contrast to the messages from the MHSP itself, the messages from other modules arrive at any time. The MHSP first needs to check if there is enough time to send this message, otherwise this message is buffered and send in the next slot.

2.3.1.2. Routing Table Module

The Routing Table Module consists of a table and some functions to operate on the table. The table is divided into two parts: The Children List and the Routing List. (see Figure 2.2)



Routing Table			
Children ID	Num Update	Recv MSG count	Children List
Children ID	Num Update	Recv MSG count	
...	...	...	
Destination ID	Next ID		Routing List
Destination ID	Next ID		
Destination ID	Next ID		
...	...		

Figure 2.2.: Routing Table

Children List For each child the ID, number of updates, and the Received Message Counter is stored in the list. The number of updates and the Received Message Counter are updated each time the node gets the Information Message.

Routing List The Routing List has two columns: One for the ID of the destination node and one for the ID of the next hop. There are functions to add or remove Routing List entries or just to get the ID of next hop.

2.3.2. Chain of execution of the MHSP Module

The MHSP Protocol can be divided into three windows. The information, Routing Table Updating, and Parent Window (See picture 2.3). The Information Window is to inform the nodes about the states of the node (hop level, ID of the parent, number of children, ...). When the node knows everything about its neighbours, it asks a child to send its updates. Finally the nodes checks the parent to verify the connectivity. If the parent died or the node barely hears its parent, it takes an alternative parent as parent which it hears better.

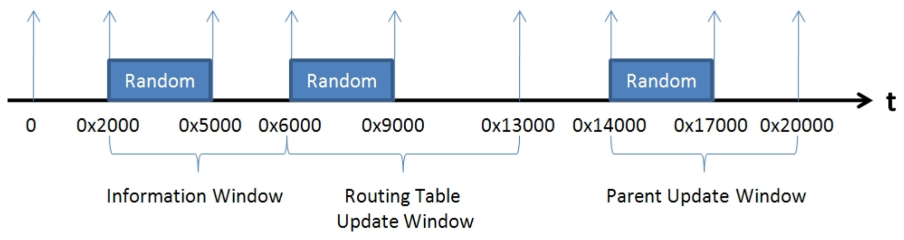


Figure 2.3.: Chain of execution of the MHSP

2.3.2.1. Information Window

The first part is the Information Window. The node takes a random number r_i in the interval [0, 12288] and has to wait for $DELAY + r_i$ jiffies[1] (At the beginning of the slot the radio needs to be turned on which needs some time. After the $DELAY$ all nodes have their radio on). Then the node sends the Information Message which contains the ID of the parent, the nodes hop level, number of children, number of changes in the Routing Table, reset the number of updates of a child and some children IDs. The Information Message is needed to know the nodes in the neighborhood. The parent knows which children it has and if its children have updates for him. The parent can reset the number of updates of the children, to get the whole Routing Table of a children at the next update.

[1] 1 jiffy = 1/32000 seconds

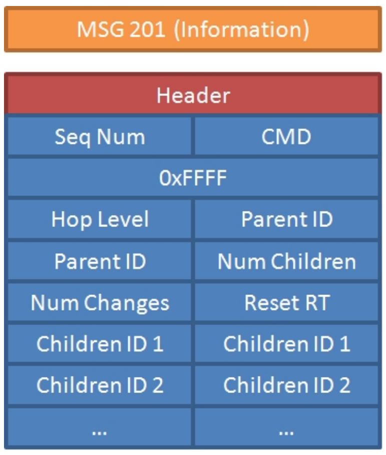


Figure 2.4.: Information Message

2.3.2.2. Routing Table Update Window

The Routing Table Update Window starts after the information part. It is started by a timer which waits $DELAY + r_i + START\_UPDATING$ jiffies. The parent then selects a children which should send its updates. It sends the child the Get Update Message (see Appendix A.1) whereon the children sends its updates. The updates are incrementally, except the parent reseted in the Information Window the variable for the number of changes of the nodes. In this case the children sends the whole Routing Table to the parent.

2.3.2.3. Parent Update Window

The last step is to control, if the node heard its parent enough in the past. As default the node needs to hear its parent at least in 4 of the most recent 8 slots. If this is the case, the protocol terminates for this slot. In the other case the nodes needs to change its parent. Therefore it sends a request to the best alternative parent. If the new parent answers it changes the parent, if the alternative parent does not answer the node does not change the parent for this time but probably in the next slot.

3. Testing

When Testing a program which is running on different nodes, it is difficult to retrace the program states. There is no monitor on which the text can be printed out. The only way to get information about the node is to send messages via the serial connection to the computer. To get the output of all nodes there are a lot of serial cables used or a lot of messages need to be send to a node which has serial connection.
To test a new step of implementation the setup consisted of 4 nodes. The root node with two children and a node, which observes all messages sent over the radio. Each node had a serial connection and printed out debugging text. On the computer side the text or message was printed. This setup just helped to test the added functions on their correctness.
During the semester thesis there were three implementation steps and finally four testing steps:

1. sending and receiving the Information Message and determine the parent
2. creating the Routing Table by adding children and routing entries
3. updating the Routing Table and the Parent List

The testings were done in parallel of the implementation. At the end of each implementation phase there was a test session in small scale, which showed up some implementation mistakes. After the third phase the program worked properly with this setup. The final test was the most important one to see if the implementation also works with more than three nodes.

3.1. Final Test

For the final test the setup consisted of nine nodes where one of them is the root node. The tenth node observes the messages transmitted over the radio. The nodes look for a parent, create the Routing Table and update it. The selection of a parent, accepting the children and building the Routing Table can take some time. At the beginning the nodes will change their parents very often but after some time it should get stable.

3.1.1. Observation

The observation of the output from the serial interface shows that the nodes found very fast their parent and the parent accepted the children. The routing table is created and the updates are sent. The nodes changed their parent very often at the beginning of the testing, as expected. To get the state information of the nodes, to see what happens, there is a lot of output sent over the serial. The messages coming over the serial did not always contain all information. Sometimes some information over the serial got lost but from the information getting at a later message the node always has the correct state and the information loss did not interfere with the node. Another observation and a more problematic one is that the Information Message is not sent all the time of all nodes. In each slot time the Information Message from one or more nodes is not sent. It happened on all nodes and almost never on the same node in consecutive slots. This caused that the nodes change their parent frequently even after a long testing time.

3.1.2. Problem and Approaches

The problem of not sending or receiving the Information Message every time caused that the received message count from the parent drops beneath the threshold of four times heard in the last eight slots and as a result the nodes change their parents frequently which caused an instable tree where the root never had an up to date Routing List. There are several ideas to solve the problem:

1. extend the window size to send the Information Message: A bigger window to send the Information Message reduces the probability that two or more messages interfere each other. In this case the range for the random number is increased to start the routing table update.
2. check the random number generator: The point in time the Information Message is sent, is determined by the random number. If nodes have approximately the same random number, the messages of these nodes can interfere.
3. deleting the debugging messages sent over the serial connection: Printing out information about the state information of the node this could solve the problem.

3.1.3. Solution

The testing of these approaches showed that the first and second ones did not show any significant improvements. The random numbers were different and extending them

did not help either.
Surprisingly the last approach helped. It was found out by implementing MHSP in a simple way again to simulate the message traffic. The simple MHSP just consisted of the slots and the sending of the Information Message in the Information Window. The Information Message was just the header and noise in the payload (keeping the size of the Information Message). This time the print-out over the serial connection just contained the information where the message was coming from and most importantly, the serial message was send at the end of the Information Window. This simple MHSP worked. After commenting all the code which is used for debugging and not necessary to check the functionality of the code, the MHSP worked. The reason is that while sending a message over the serial connection the node could not get the information from the radio and the radio message is lost. For debugging reasons the node sent a message to the computer whenever it got a message from the radio. After receiving the first message over the radio, the corresponding serial message, generated from the debugging code, blocked the reception of other radio messages for several cycles. An extension of the sending window for the information part could solve this problem too but the algorithm would not be efficient anymore.

4. Improvements

In terms of time shortness there are two things which are not implemented jet but which can be useful.

1. There exists a case where the tree can loose a subtree: In case the parent does not hear its child anymore but the child still hears the parent, the parent will send a message to the child to terminate this connection. It can happen that this message gets lost and the child never receives it. For the parent the connection is terminated but for the child the connection is still up. The prospective messages from the child will not get to the parent. In reverse, the child gets the Information Messages from the parent. This causes the problem that the child still believes that the parent is still accepting it as a child, and the entire subtree is not connected anymore to the tree.

   In the MHSP exists a part of the solution. The Information Message contains two or more IDs of children which the parent accepts. These IDs change for each Information Message. The usage of this mechanism is that the children can check if its ID is in the Information Message and see if it is accepted by the parent or if it needs to change the parent. In case the parent has a lot of children there is not enough space for all IDs in the message. This must be taken into account.
   To fix this problem the node needs to check each Information Message from the parent if its ID is in the packet or not. If after several rounds the child is not accepted by the parent the child needs to change the parent. The difficult part is how to react if a children is accepted or not.

2. The MHSP sends messages from a sender to the receiver over multiple hops. The nodes just send an ACK to the previous node. In this case the sender just knows that the message arrived at the next hop but does not know it the message arrived at the destination. The user of MHSP should himself check if the message arrived. The sequence number at the beginning of the message is used for the node to node communication. If a sequence number is used for the source - destination

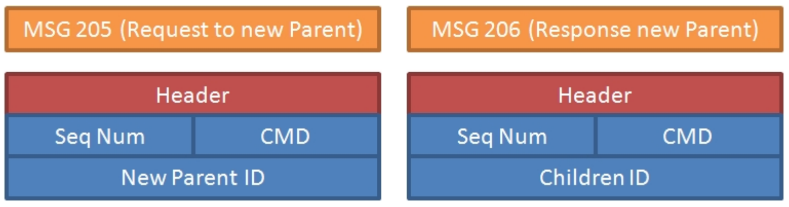
connection over more hops the user of MHSP should use own sequence numbers and also for the acknowledgments which travel back over several hops.

A. Appendix

A.1. Messages

For the messages in MHSP the first four bytes are used for the sequence number (1 byte), command (1 byte) and the ID of the next hop(2 bytes). These three informations are necessary to send a message from one node to another. After these four bytes comes the payload. The payload can be at most 24 bytes long. All the messages used in the MHSP have these four bytes at the beginning.

The Request Message is used to ask a node if it could be the parent of the node. The node answers with the Response MSG to the child which changes the parent..

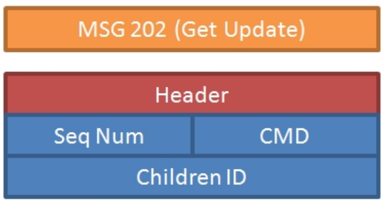

MSG 205 (Request to new Parent)
Header
Seq Num
CMD
New Parent ID
MSG 206 (Response new Parent)
Header
Seq Num
CMD
Children ID

(a) Request Message (b) Response Message

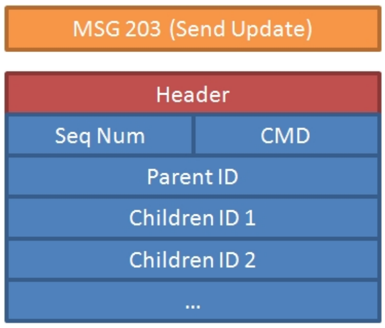
Figure A.1.: Messages used to change the parent

The parent sends the Get Update Message to receive the updates from the children to get its Routing List up to date. The children answers with the Send Update Message

A parent can remove a child by sending the Remove Child Message . It happens when the Information Message (see Figure 2.4) of the child does not arrive at the parent anymore.



(a) Get Update Message



(b) Send Update Message

Figure A.2.: Messages used to get the updates from a child

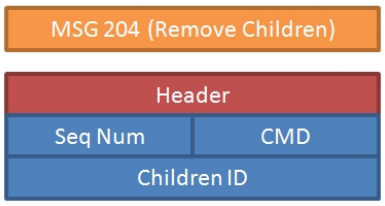


Figure A.3.: Remove Child Message

A.2. Slots

The slots from the Time Schedule Module can't be assigned at random. The Module Time Sync needs some slots in a periodic way namely, four slots within 128 seconds. The MHSP needs four slots to create and generate the Routing Table and four slots to forward the messages from the other modules which use the MHSP. The slots should not intersect with each other otherwise malfunctions can occur. In picture A.4 the distribution of the slots within 128 seconds is showed.

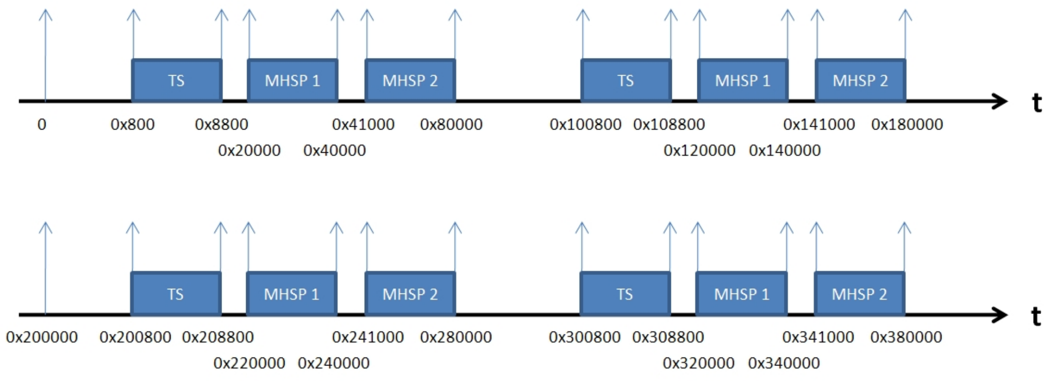


Figure A.4.: Time Scheduling: TS is for the Time Sync module, MHSP1 is used to create and update the Routing Table, MHSP2 are the time slots for the modules, which uses MHSP