UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Design of a low power wireless sensor network for environmental monitoring

by

Gideon Spreeth

Thesis presented in partial fulfillment of the requirements
for the degree of

Master of Science in Electronic Engineering

at Stellenbosch University

Supervisor:
Dr. R. Wolhuter

December 2008

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Abstract

A WSN (wireless sensor network) consists of a collection of small, low power electronic devices that can sense their environment and communicate with each other in order to send data to a base station for logging and monitoring. Research done on WSNs has increased rapidly over the past few years, as the necessary RF hardware has become cheaper and smaller. The wealth of information and hardware available in this field has made it possible to design and deploy networks for a multitude of monitoring purposes, on almost any terrain, without an existing telecommunication infrastructure.

This thesis presents research into some major aspects of WSNs and the implementation of a test system with wireless sensor motes, that can be used for environmental monitoring, conservation purposes, impact studies, early warning systems for floods, fires etc. The system also has a wide range of possible uses in agriculture, as more data and better control over crops can increase yield.

The power constraint of sensor nodes is one of the biggest concerns, as batteries can be depleted quickly and render a system useless. For this reason, work was focused on reducing power consumption of the hardware by means of various methods. Power use was also simulated very successfully, giving a accurate way of predicting node lifetime with a variety of battery types. The system was implemented on the Tmote Sky hardware platform using the open source sensor network operating system, TinyOS.

Opsomming

'n RSN (radio sensor netwerk) bestaan uit klein, lae energie elektroniese toestelle wat hulle omgewing kan monitor en met mekaar kommunikeer om data te stuur na 'n basis stasie vir stoor en monitering. Navorsing in die veld van RSN'e het drasties toegeneem oor die afgelope paar jaar, soos die nodige RF hardeware goedkoper en kleiner raak. Die magdom informasie en hardeware beskikbaar in die veld het dit moontlik begin maak om netwerke daar te stel vir 'n verskeidenheid van moniterings toepassings, op amper enige terrein sonder bestaande telekommunikasie infrastruktuur.

Die tesis bied navorsing aan wat gedoen is op sommige aspekte van RSN'e en die implementasie van 'n toetstelsel met radio sensor motes, wat gebruik kan word vir omgewingsmonitering, bewaringsdoeleindes, impakstudies, vroeë waarskuwingstelsels vir vloede, vure ens. Die stelsel het ook 'n wye reeks moontlike gebruike in landbou, omdat meer data en beter beheer oor oeste dravermoë kan verbeter.

Die beperkte krag beskikbaar tot die nodusse is een van die grootste probleme, omdat batterye baie vinnig pap kan raak en 'n stelsel onbruikbaar maak. Om hierdie rede is die werk gefokus op die vermindering van kragverbruik deur gebruik te maak van verskillende metodes. Kragverbruik is ook suksesvol gesimuleer, wat 'n akkurate manier gee om nodusleeftyd af te skat vir 'n verskeidenheid batterytipes. Die stelsel is gemplementeer op die Tmote Sky hardeware platform, deur gebruik te maak van die die oopbron sensor netwerk bedryfstelsel, TinyOS.

Acknowledgements

I'd like to thank Dr Riaan Wolhuter for all his efforts in helping me produce this piece of work. Another big thank you to all my friends and family who supported me through years of studying.

Contents

List of Figures

List of Tables

Nomenclature

AOA	Angle of arrival
bps	Bits per second
bsl	Boot sector loader
GPIO	General purpose input output
GUI	Graphical user interface
LDR	Light dependant resistor
LED	Light emitting diode
LOS	Line of sight
LQI	Link quality indicator
mAh	milli Ampere hour
MEMS	Micro electrical mechanical systems
NI-Cd	Nickel Cadmium
Ni-MH	Nickel metal hydride
OS	Operating system
QoS	Quality of service
RF	Radio frequency
RH	Relative humidity
RSSI	Received signal strength indicator
SP	Sensornet protocol
SPI	Serial peripheral interface
TDOA	Time difference of arrival
TOA	Time of arrival
USB	Universal serial bus
Ws	Watt seconds
WSN	Wireless sensor networks

Chapter 1

Introduction

Acquiring information on our environment has become increasingly important in many areas over the last couple of years. The increase in natural disasters and weather changes in the recent past are cases in point. Due to population growth, mismanagement and overuse of resources like fresh water supplies, it is clear that we need to look at more efficient ways of producing, using and saving everything we have.

If one could, for example, collect information on ground-moisture level along a riverbank, the effect of alien plant species on the environment could be examined. Forest fires could be detected much earlier if one could monitor temperatures at a number of critical points and be warned immediately of any dangerous levels. Farmers can increase their yield and reduce work at the same time, by having an automated and distributed sensing system in a field. This could monitor information like temperature, humidity and ground moisture and automatically adjust irrigation in different areas according to each one's needs. It is particularly for this type of environmental application that we want to design a robust sensor network that would require minimal maintenance.

To implement such a distributed sensing system, a number of inexpensive sensors is required, as well as a way to get the collected data to a central control point. A wired sensor system is only feasible in a small number of applications because it can only cover a small area. A wired system could be the best choice in control of small greenhouses, home alarm systems or buildings with existing wired network support. In most cases of environmental monitoring, however, the area in need of coverage is much too large and does not have an existing power or communications infrastructure.

Because low power processors and radios have become widely available and inexpensive, WSNs have become a widely researched subject. Figure 1.1 shows a basic WSN, consisting

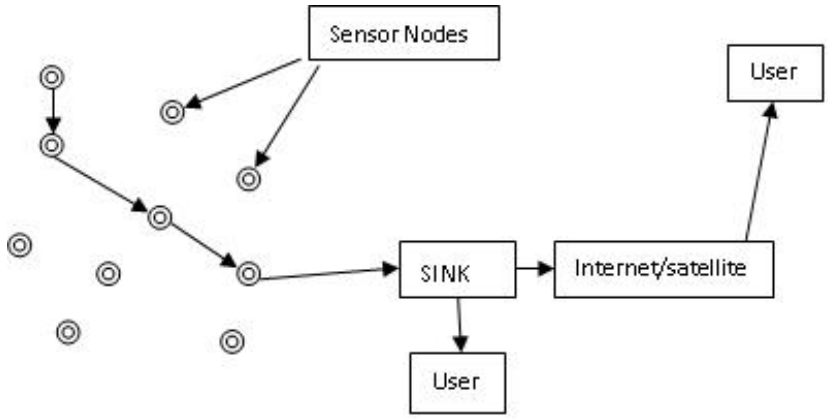


Figure 1.1: *Layout of a typical sensor network*

of wireless nodes with the necessary sensors attached. Collected data can then be sent wirelessly, through a multihop protocol via other nodes in the system, to a central base station or sink. This will be a computer that can be used to monitor the data, or that is connected to a web server for off site monitoring. The wireless nodes[1], need only do basic processing tasks and send sensor data at the required rate. With this functionality, WSNs can be used with great success in the aforementioned applications. The greater resolution and spread of data acquired on a specific area, as compared to what is collected from one weather station, can produce a bigger understanding of weather influences and offer an easy and accurate way to micro manage the environment.

[1]Also called motes, when referred to in the physical sense (as a piece of hardware).

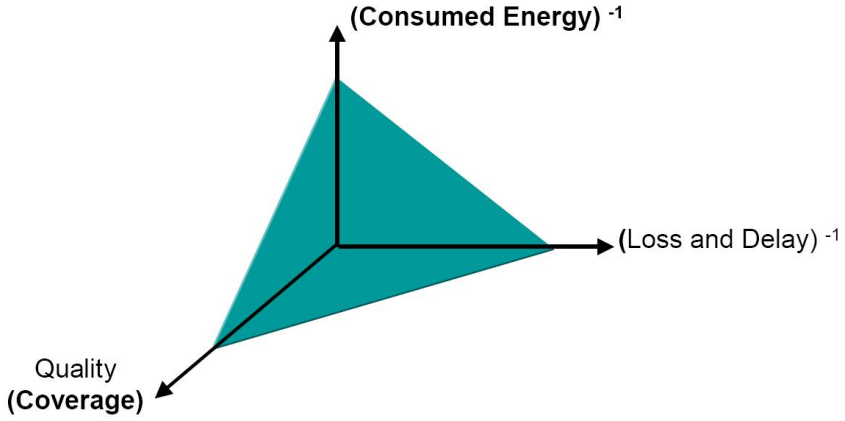


Figure 1.2: *Design criteria for a sensor network*

1.1 Typical network and system requirements

Each application has its own goals, but to design a suitable WSN three main criteria [23] have to be considered. Figure 1.2 shows the relationship between energy use, data loss / network delay and quality of service (QoS). Each of these can vary greatly in relative importance, depending on the application of the network. For our environmental monitoring application, low energy use is by far the most important. The motes will have to run off battery power, as they will be used outside where mains power supply is not available. It is not feasible to replace batteries daily, or even weekly, in most applications, so it is necessary to design the system in such a way that energy use is limited as far as possible. This can be achieved by using low power hardware and also intelligent communication and data gathering schemes.

Because of power constraints radio ranges are usually small. Therefore, next in line of importance will be ensuring good QoS and coverage of the network. The monitored area will be covered by a number of sensors per unit area for an application like plantation moisture control. Big gaps between motes can lead to inadequate data resolution and, therefore, ineffective watering management. The communication range limitations require that a multihop system be implemented. This may result in a problem if certain nodes in the network die, or are too sparsely placed. Network coverage and connectivity of other nodes might then suffer. Therefore, the overall QoS, coverage and connectivity can depend on the functioning of single nodes, if these problems are not addressed accordingly.

It is not always important to have perfect packet transmission. For example, a system that logs the local temperature every thirty seconds will be adequate even if every ten minutes or so a packet is dropped. Especially with weather data that does not change

very quickly, it is also acceptable to have a large network delay of multiple seconds. A good example of where both coverage and reliable packet transmission become much more important is in early warning systems like fire detection. The system would be useless if an area of forest could burn down before a sensor were triggered, or if the chance of an alarm message being dropped were so large that the system could fail totally.

Because of the outdoor placement of the motes it is essential that the electronics be resistant to weather extremes and also the occasional rough handling. These requirements can easily be met by using robust enclosures for the motes.

A big problem with distributed systems like these is the fact that one might need many nodes to effectively cover an area. This might prove too costly, therefore the effectiveness of the system also depends to a large extent on the price of the hardware. If one can use inexpensive nodes, it would not be a problem to have redundant nodes in an area, or even lose a couple every now and then.

If we assume the WSN is up and running and sending data to the sink, the next consideration is being able to use the information received, be it close or far away from the actual system, immediately or even months after actual data gathering. Software is needed for saving data in a usable format and also some kind of interface for visualizing it. Functionality of a WSN greatly improves if data can be made available over a network or the internet. This would enable users to access their monitored environment from anywhere in the world.

The main goal of our system will be to monitor weather and/or ground moisture information. This will be monitored in places where frequent access is not possible. As mentioned, speed and reliability of data delivery is not important with this type of data. With the criteria as stated above, we could now specify that our system will be focussed mainly on ensuring low power consumption and also have robust features, in order to extend lifetime and limit down time. Data will need to be accessible off site, therefore, a system that will reliably sense and relay data to distant users, without requiring frequent maintenance, is a goal to keep in mind while designing the WSN.

1.2 Design approach

Relevant work in the field of wireless communication and sensing systems was studied to determine common standards and the correct approach towards implementing a WSN. The subtle differences between WSNs and other wireless systems had to be kept in mind when considering communication principles and protocols. A basic system specification and communication protocol strategy could now be outlined to address the relevant problems.

With the basic system outline in mind, we now had to find an adequate simulation and hardware platform for system development and testing. There are a number of network simulation platforms available, but not all are suited to WSNs. The power constraint is the biggest issue that must be resolved and most packages do not have support for simulating power usage. Most of the development required on sensor networks is in the software department and not hardware, as a wide range of very capable platforms designed for WSNs is currently available on the market (see section 2.5). It was therefore decided to use off the shelf WSN motes. With a suitable WSN simulator and hardware platform, the system could now be implemented and simulated concurrently. Testing and using a system easily and quickly is essential, therefore an interfacing system had to be constructed. A graphical user interface (GUI) was used throughout the implementation process of the system, to visualize system data for debugging and testing. The simulation and hardware platforms, coupled with the data logging and user interface applications, made it possible to test our improvements and compare them to existing systems and then present the expected system performance.

1.3 Summary of major aspects addressed and contributions

It was decided at the outset that there is merit in carefully investigating few aspects of WSN development and implementation, in an effort to realize corresponding improvements, not just to the individual areas, but also consequently to the overall WSN functionality. These aspects can be summarized as follows:

Hardware platform with open source operating system. A very capable hardware platform was found in the Tmote Sky motes with the TinyOS operating system. TinyOS has many basic components that can be combined to implement a successful system.

This OS was studied to determine its abilities and the shortcomings of the available components.

Smart application. A mote control application was developed with network programming and lockup safeguard feature. This program also automatically detects sensors and reports data through the multihop system.

Power aware multihop transmissions. A basic link quality routing protocol was already incorporated, enabling multihop message transmission. This was studied in depth and extended to make the protocol power-aware, as such a protocol was not yet available in the TinyOS distribution.

Duty cycle power saving incorporated. The routing protocol was used to good effect, in conjunction with a dutycycle system incorporated in TinyOS, to limit power usage as far as possible.

Development of WSN simulator with power simulations. It was necessary to find a suitable simulation platform with the ability to simulate power usage. It was found that most platforms did not have this ability and were not written to take aspects of WSNs into account. This led to the development of a suitable system in Truetime, with very accurate end results.

Range of battery technology tests. Tests were also done to demonstrate the suitability of different battery technologies for sensor networks. This gave a great view of what could be expected from different types of cells.

Mote lifetime tests and estimations. The lifetime of motes with different cell types could be measured and this was very accurately matched by the simulation and mathematical models written. This gave an accurate means of demonstrating power usage of motes and estimations of lifetime extensions without the need for long system test runs.

GUI extensions. Another aspect addressed was the visualization and logging of data. A basic GUI was supplied with TinyOS but this was extended to make development easier and the display of more data channels possible.

Data logging. A basic data logging system was also constructed to store information and make incorporation with systems such as web display possible.

Voronoi diagram as communication analysis tool. Voronoi diagrams were used to calculate the difference in path length and network throughput in multihop routing applications between the best quality and shortest routes.

During these investigations and developments, a complete WSN was implemented as a test platform.

1.4 Thesis layout

The theory behind WSNs will be discussed in Chapter 2, which will provide an overview of the different aspects of sensor networks and the issues to be addressed in order to implement a successful system. From the background study we will go to the hardware and software implementation of the system in Chapters 3 and 4. Performance measurements of the implemented system will be discussed in Chapter 6. Simulations of the system will be shown in Chapter 5 with some comparative results between this and the actual network performance in Chapter 7. Concluding remarks and some proposals on future work required in the field will be given in Chapter 8. The appendices include hardware schematics, system block diagrams and other relevant reference information.

Chapter 2

Theory and previous work

2.1 Communication basics

2.1.1 Radio propagation

Communication is the single most important and difficult aspect of WSNs. The ability of motes to relay data effectively, to maintain connectivity and to provide an acceptable QoS all rely on this. Therefore a short discussion on different natural effects that influence signal propagation will be provided. Our system will consist of stationary nodes and this reduces certain effects, such as Doppler and fading, which are much more apparent in mobile systems.

The quality of a wireless link can vary greatly, even with stationary nodes, because of propagation effects such as:

Reflection from the ground or large objects

Diffraction from edges and corners of terrain or buildings

Scattering by foliage or small objects

Attenuation by rain, the atmosphere or other objects

This list itemizes most of the important effects for frequencies above 500MHz and thus includes the type of wireless system that we are investigating (see section 2.5 on hardware). Generally these effects reduce received signal power and result in path loss calculations being very difficult and imprecise. The Friis formula, 2.1, shows that basic path loss

Material	Frequency	Loss(dB)
Concrete Wall	1.3 GHz	13
Sheetrock	9.6 GHz	2
Plywood	9.6 GHz	4
Chain link fence	1.3 GHz	5-12
Loss between floors	1.3 GHz	20-30
Corner in corridor	1.3 GHz	10-15

Table 2.1: *Attenuation of common Materials*

is $\frac{1}{R^2}$ but this is only nearly accurate if there are no other losses with a perfect single line-of-sight (LOS) path between transmitter and receiver.

$$P_r = \frac{G_t G_r \lambda^2}{(4\pi R)^2} P_t W \tag{2.1}$$

Reflection, diffraction and scattering all have basically the same negative effect of producing signals at the receiver with different amplitudes and time offsets. This causes destructive interference which can greatly reduce received signal strength, even if it seems as if there is a LOS path between sender and receiver. In spite of the negative effects, this multipath effect can also be beneficial in some cases. If one has a wireless communication link without a LOS path but there is a reflective surface in sight of both sender and receiver, the reflected signal might be even greater than the direct signal and therefore produce better communication.

Attenuation is the decrease in signal power due to losses in the propagation path. In table 2.1[1] the attenuation figures of common materials in our everyday environment are shown. A system that gives a theoretical range of hundreds of meters may only be able to relay data indoors over tens of meters. Atmospherical attenuation is negligible at frequencies below 10GHz.

[1]This and more relevant information can be found in [33]

2.1.2 ISM communication bands

Usage of the RF spectrum is controlled differently in each country but a few frequency bands are open for use by the general public, without licensing. These are still subject to the different limitations set by each government. The bands most commonly used in licence free wireless applications are:

433 and 868/915 MHz

The 433 MHz band is open for any wireless communication, but has a +20dBm transmit power limit and a radio duty cycle limit of 10%. The duty cycle states the percentage of the time that the radio may be transmitting. The 868 and 915 MHz bands are split into a couple of groups that are either allocated for alarm systems or open for any use. The duty cycle limits range from less than 0.1% to 100%. Transmit power limits also vary between bands in different countries.

2400 to 2483.5 MHz

This frequency band is the most commonly used communication spectrum, with no limitations on application or transmit duty cycle. The worldwide minimum limit on transmit power in this band is +10dBm. This frequency is used in many applications, such as voice communication radios, cordless phones and most wifi standards.

5150 to 5825 MHz

There are three 100MHz bands in this range that are also used for wifi operation, though to a lesser extent than the 2.4GHz band is. The 802.11a wifi standard, released in 1999, produced a maximum data rate of 54Mbit/s. As this band is used in far fewer applications, it is less prone to interference than other wifi bands. The downside is that the higher frequencies result in a shorter communication range.

2.1.3 802.15.4/ Zigbee communication standard

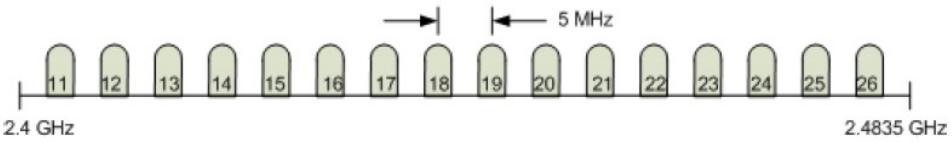


Figure 2.1: *IEEE 802.15.4 operating channels*

This is a IEEE standard that defines the Medium Access control (MAC) and Physical Layer (PHY) specifications for Low-rate wireless personal area networks (LR-WPANs) [17]. This standard is designed to be used with low power and low complexity hardware that communicates wirelessly over short ranges. The standard uses a contention based (CSMA /CA) carrier sense multiple access with a collision avoidance medium access mechanism.

2.2 Connectivity and coverage

One of the most important issues in WSN is coverage [41] of the network. Due to the large variety of network applications, there are multiple interpretations of what coverage entails. In wireless networks it may be seen as the communication coverage of nodes. In environmental monitoring WSNs this issue also relates to how well the system observes a given area and how accurately it can detect and report changes. If, for example, the main goal is to detect fires, an area with poor sensor coverage could already have been burning for some time before detection takes place, by which time the fire has spread further. A system used for irrigation control in a plantation, will also be untrustworthy if an area can dry out and this is not picked up by a sensor node.

The issue of coverage coincides with that of connectivity. The sensor nodes have a limited range of communication and therefore, if an area has too few nodes, the connectivity of the nodes throughout that area might also suffer. The lifespan [21][40][20] of the whole network[2] is much more important in these types of network than the lifespan of individual nodes. Especially in large density networks, certain nodes can die out while the network continues to function with adequate efficiency and accuracy. In securing a good lifetime for the system the connectivity becomes most important. If a certain number of nodes

[2]The network lifetime can be seen as the length of time that enough nodes are still connected to give adequate coverage.

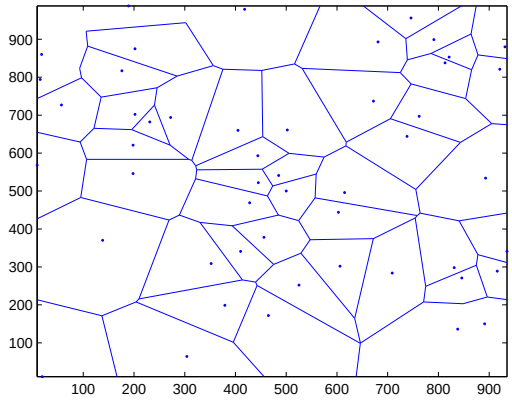


Figure 2.2: *Voronoi tesselation of node locations*

is needed to relay data through a poorly connected zone, the death of one or two critical nodes could also imply the death of a big part of the network.

2.2.1 Voronoi diagrams

Voronoi diagrams [5][6][3] can be basically explained as follows: If one considers a set of coplanar points then, for each point in the set, a boundary can be drawn around the area which includes all the points that are closer to this point than any other. These boundaries are called Voronoi polygons, and the set of all Voronoi polygons for a given area are called a Voronoi diagram. Figure 2.2 shows a Voronoi diagram for a given set of random points.

Voronoi diagrams have been re-invented and studied independently in the fields of applied natural sciences, mathematics and computer science. There are three main reasons why these diagrams receive so much attention. Firstly, Voronoi diagrams arise in nature in various situations. For example, human intuition is often guided by visual perception, so if one sees a structure it can be much easier to understand the underlying problem. Voronoi diagrams also have very interesting mathematical properties and are related to many well known geometrical structures. Finally, Voronoi diagrams have also been a powerful tool in solving seemingly unrelated computational problems and, therefore, have seen increasing use in computer science. It is especially in the field of connectivity and coverage in WSNs that they can be used to good effect.

Structures that are directly related to these diagrams are Delaunay triangulations. These can be found by connecting the points in the Voronoi diagram whose polygons share

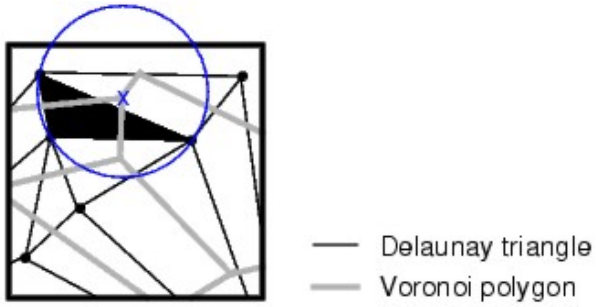


Figure 2.3: *Delaunay triangles with corresponding voronoi polygons*

an edge. As shown in figure 2.3, another interpretation is that the centre of a circle circumscribing a Delaunay triangle is at the vertex of a Voronoi polygon. Neighbour information can be extracted from these triangulations, because points in close proximity are connected. It is with the properties of Delaunay triangulations and Voronoi diagrams that the best and worst case coverage areas can be found.[3]

Worst case coverage- Maximal Breach Path

This is defined as a path through a sensor field with the property that for every point on the path (from arbitrary point A to B), the distance to the closest sensor is maximized. Since the Voronoi polygons maximize the distance to the closest sensor sites, this path will lie on the Voronoi line segments. The algorithm to find this path basically does the following: [24]. Each edge in the graph is given a weight equal to the distance to its closest sensor. A binary search is now made to find a path from A to B through edges with weights more than a predefined breach weight. If a path is found, this weight is increased and another search is made. In the end, the Maximal Breach Path will have been found.

Best case coverage- Maximal Support Path

This is defined as a path between two points through the sensor field where for any point on the path, the distance to the closest sensor is minimized. In this case the Delaunay triangulations produce triangles with minimal edge lengths and, therefore, the path must lie on the Delaunay edges between sensors. The algorithm is very similar to the worst case computation, with the difference that the Delaunay edges are used and they are given

[3]Works equally well for sensor coverage, and communication coverage

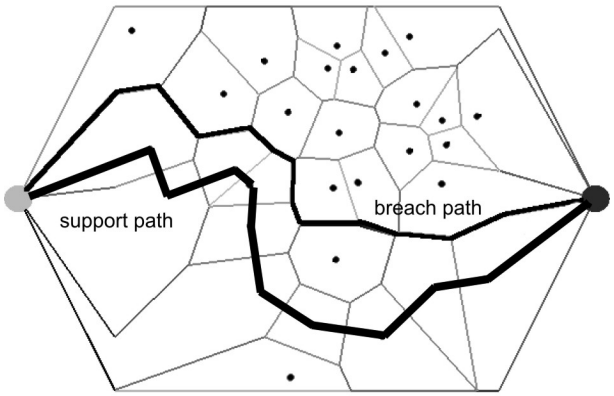


Figure 2.4: *Maximal support and breach paths in a sensor field*

weights equal to their own lengths. Lastly, of course, the support weight parameter is used and must be minimized. Figure 2.4 shows the maximal breach and maximal support path for two points through a sensor field.

2.3 Localization

The problem of localization[4] in WSN's [14][16][28][35][36], or determining where a given node is physically located in a network, can be very challenging but is also important in many applications. Localization, for example, can produce novel ways of reducing power consumption in sensor networks and in context-aware applications it enables the smart selection of devices and supports intelligent coordination among nodes. Because of practical constraints such as the size, cost and available power of nodes, it is not possible to fit motes with GPS, therefore a different approach to localization must be found [7]. A couple of popular schemes will be analyzed to help in deciding which approach would be most beneficial for our application.

2.3.1 Connectivity based

In [11] a localization scheme based on the connectivity of nodes to beacons with known locations is proposed. An idealized radio model is used, because of the simple mathematics behind it. This model compares quite well with an uncluttered outdoor environment. Two

[4]The term localization was borrowed from robotics,where it refers to determining the position of a mobile robot in a certain coordinate system.

assumptions are made in the model:

- Perfect spherical radio propagation
- Identical transmission range for all radios.

A number of nodes in the network with known positions, (X_1, Y_1) to (X_n, Y_n) act as reference points (R_1) to (R_n). These nodes form a mesh and transmits periodic beacon signals containing their positions. It is assumed that in any time interval T, all the reference points have transmitted exactly one beacon signal. The following terms are used in localization calculations:

d Distance between reference points

R Transmission range of reference point

T Time interval between beacon signals transmitted

t Data collection time

Nsent(i,t) Number of beacon signals sent by R_i in time t

Nrecv(i,t) Number of beacon signals sent by R_i that have been received in time t

CM_i Connectivity metric for R_i

CMthresh Threshold for CM

(X_{est}, Y_{est}) Estimated location of the receiver

(X_a, Y_a) Actual location of the receiver

Each node will listen for a fixed period of time t and collect beacon information from all reference points in its area. The information per reference point can now be characterized by the connectivity metric:

$$CM_i = \frac{Nrecv(i,t)}{Nsent(i,t)} \times 100 \tag{2.2}$$

From the information it receives, the node will be seen as connected to a reference point, if its connectivity metric for that reference point exceeds a given *CMthresh*. The receiver can now localize itself to a region that is given by the centroid of the reference points, as in equation 2.3.

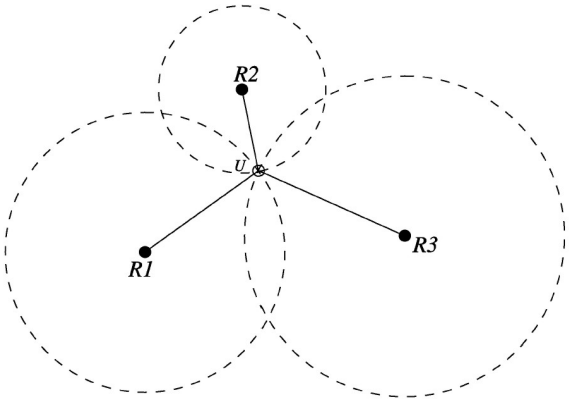


Figure 2.5: *By measuring distances to three reference points, the centre node, U, can determine its position by multilateration*

$$(X_{est}, Y_{est}) = (\frac{X_{i1} + \bullet\bullet\bullet + X_{ik}}{k}, \frac{Y_{i1} + \bullet\bullet\bullet + Y_{ik}}{k}) \tag{2.3}$$

The accuracy of localization can be characterized by the localization error LE [12].

$$LE = \sqrt{(X_{est} - X_a)^2 + (Y_{est} - Y_a)^2} \tag{2.4}$$

By increasing the range overlap, $\frac{R}{d}$ the granularity of the regions can become finer and therefore minimize LE. Tests were done in a 10mx10m grid with reference points at the four corners. Measurements were taken at all of the 121 1mx1m grid intersection points and produced adequate results. The average LE was 1.83m with standard deviation of 1.07m. Minimum error was 0m and maximum 4.12m.

2.3.2 Signal arrival time

Signals that propagate through the air take time to cover a certain distance [9], be they sound waves, light or radio signals. If this time can be measured accurately, one can calculate to a very good degree of accuracy, the distance between the source and destination of such signals. This approach to localization is referred to as TOA (Time of arrival) or TDOA (Time difference of arrival) method. In WSNs radios are already in use, so a RF based system might work, but it is very difficult and often impossible to achieve the time synchronization needed between nodes to make accurate measurements. This is why many schemes use a round trip based system. Other information that can be used is the DOA (direction of arrival). This can be measured if directional antennas or acoustic sensors are used, with this and the TOA information the relative position and orientation of the node can be calculated.

In [25] such a scheme based on acoustic sources in a sensor field is proposed. Each source in turn emits a signal. All sensors attempt to receive source signals and then work out TOA on a network time base. This time base can be established over the network radio links, or by synchronizing times before deployment. Because acoustic signals are used the margin of error is much less, due to the huge difference in speed between sound waves and radio signals. The DOA is estimated based on the mote's local reference direction. Each node now sends its information to a central information processor which, in turn, computes the locations of the nodes.

This process of finding a location by means of given distances is called multilateration and, when angles are used, multiangulation. In a two dimensional plane with n known references *R1* to *Rn*(as shown in figure 2.5, the node U can calculate its position using the distances $d(R1, U)$. A minimum of three equations must be formed as given below. By solving the quadratic equation for U_x and U_y, the position is found.

$$\begin{aligned}
(R1_x - U_x)^2 + (R1_y - U_y)^2 &= d(R1, U)^2 \\
(R2_x - U_x)^2 + (R2_y - U_y)^2 &= d(R2, U)^2 \\
... &= ... \\
(Rn_x - U_x)^2 + (Rn_y - U_y)^2 &= d(Rn, U)^2
\end{aligned}$$

2.3.3 Signal strength

Another way to try and localize a node is by using signal strength information. As shown in section 2.1.1 a signal could be affected by a varying amount of interference and therefore the basic Friis path loss formula is only accurate if there is a LOS connection between nodes.

Calculating distances by signal strength relies on the assumption that the nodes can accurately measure received signal strength. Luckily, radios now usually have some kind of signal strength measurement built in. The two commonly used measurements are (received signal strength indicator) RSSI [38] [37] and (link quality indicator) LQI.

The CC2420 radio from Chipcon gives both these measurements (see [34] for more information on the radio). The RSSI measurement is given as a continuously updated average value over eight symbol periods (128us), while the radio is receiving. Figure 2.6 shows the linearity of the RSSI reading over the whole reception range. The radio also provides an average correlation value for each incoming packet. This value can be seen as the chip

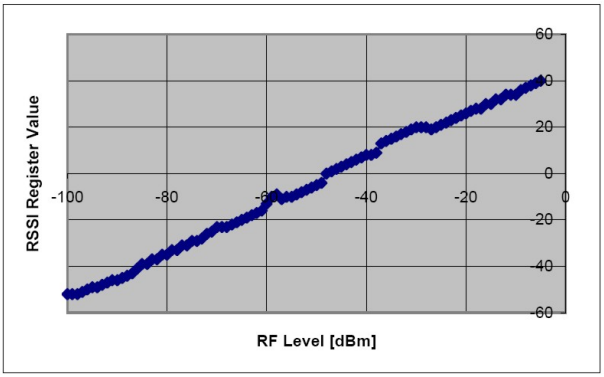


Figure 2.6: *The typical RSSI value measured vs. actual RF input power for Chipcon CC2420 radio*

error rate. The LQI value is therefore more a measurement of packet quality than signal strength.

Let's say a system is deployed and all nodes have a perfect LOS path to one another. This can be achieved in practice if nodes with omnidirectional antennas are placed in an open field above ground. If a constant transmit power is used, the received signal strength can be correlated to a distance and very accurate distance calculations can be made between all nodes within range of one another. Multilateration, as shown in the previous section, can now be used to localize the nodes.

The problem with this is that in most applications it would be impossible to ensure perfect LOS connections, and it is impossible to predict the attenuation resulting from all objects in the area. On the other hand, it might be unnecessary for a node to have its own location to ensure connectivity and efficient routing.

A virtual localization of nodes can be achieved if the measurements between nodes are made not in distance but in transmission cost, or quality. If, for instance, you want to localize nodes just to find the shortest paths for communication between them, it is NOT the distances that matter but, in fact, the cost of transmission over that distance [13]. Therefore it might be advisable rather to use such a cost metric to localize nodes for routing and connectivity calculations.

2.4 Routing

The design of routing protocols in sensor networks can be a difficult issue to tackle. In static and wired networks smart routing can increase throughput and as long as a reliable path [43] is available between two points, communication is always possible. In WSNs the shortest and fastest path might not always be the best and it might not be available at all times [39]. Some of the issues that arise are:

- Variable wireless link quality
- Propagation path loss
- Fading
- Power expended
- Topological changes

As discussed in section 2.1.1, these transmission effects can have a huge effect on signal reception. Coupled with the energy [27] constraints and possible changes in network topology, whether because of nodes dying or moving, a robust and efficient routing protocol must be implemented in WSNs.

2.4.1 Classification of routing protocols

WSN routing protocols can be classified by breaking them up into the three main aspects [19] of the protocol and then classifying these according to the different approaches to the problem. Figure 2.7 gives a graphical view of the different subclassifications of WSN routing protocols.

Path establishment

Path establishment can take place by the three different methods shown. Proactive path establishment means that all nodes work out the necessary paths and store them in routing tables. This means that a path is always available immediately when a message needs to be sent. The problem is that whenever a route changes the new routing tables have to be sent to all the nodes. In very big and power constrained networks this becomes impractical. Reactive protocols only work out a route when it is required, which eliminates

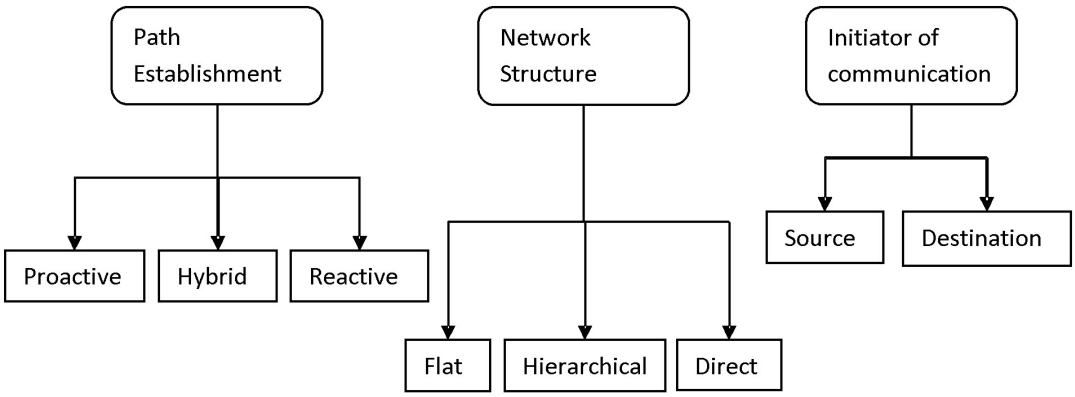


Figure 2.7: *WSN Routing protocol classification*

the problem of unnecessary work being done, but it might be too slow a process if a high throughput is needed. Hybrid protocols are constructed by using a combination of these two approaches.

Initiator of communication

The data sent through the network will either be requested by the destination (sink, base station) or just sent by the source when data is ready. In networks where communication is initiated by the destination, much more communication overhead results. Data requests will have to be flooded through the network, or the sink will need to have reverse paths to all the nodes for individual polling. Source initiated networks are event or time driven. This means that nodes can be set up either to report data at a constant rate or only to send data when a certain event occurs.

Network structure

The network structure constructed by the protocol can be classified in three different ways. The nodes in flat protocols all have the same importance and ability to route messages. Because messages must usually be sent to the same sink, network traffic in nodes closest to the sink will normally be higher. Hierarchical protocols make use of clustering [18] and cluster heads. Nodes are grouped into clusters and they all relay their data to the sink only via their cluster head. In direct protocols, all nodes communicate directly with the sink. No actual routing is therefore necessary, but in WSNs with a limited communication range this is usually not a viable option.

The disadvantage of hierarchical protocols is that the cluster heads will require a higher communication throughput than the other nodes and, therefore, might require more power. Some protocols for WSNs select and rotate cluster heads to even out the power usage, but this just complicates the system and requires more communication overhead. Scalability of the network is also an issue because a larger number of cluster heads will be required, or the overhead at the heads will increase dramatically. A flat network structure is the best suited to an WSN application and in the next section we will be looking at a few of these types of protocol as used in sensor networks.

2.4.2 Relevant protocol summary

Flooding and Gossiping

Flooding is an old and very basic technique for routing messages. In flooding, every node that receives a packet forwards it directly to all of its neighbors by broadcasting. This repeated rebroadcasting only stops when the message reaches its destination or a maximum number of hops for the packet is reached. The benefit of this scheme is that it does not use complicated topology control or route discovery algorithms, but it does present some problems. Implosion occurs when multiple copies of the same message arrive at a node; this will happen if two communicating nodes are connected by more than one neighbor. Each of these will relay the same message to the other node. Overlap occurs if two nodes share an observation region and an event in this region triggers both nodes to send data. Resource blindness is the biggest problem with this approach, as available power is never considered and no attempt is made to limit power usage in transmission. Gossiping is a derivative of flooding and eliminates the implosion problem by not broadcasting messages, but by sending them to only one randomly selected node.

Sensor protocols for information via negotiation

The SPIN family of protocols is based on two simple ideas. Firstly, that sensor nodes can conserve power if they have to send information only on the data sensed and not on all the actual data. Secondly, nodes must monitor their available power resources. SPIN uses a flat network structure and reactive routing. Communication is source initiated.

The protocol uses three different types of packet. First, the node with new data sends an advertisement message (ADV) to all its neighbours. Neighbours that receive the ADV message and are interested in the data now send a request message (REQ). Subsequently

the initial node sends the DATA message to all the interested nodes. All the nodes with the new data now in turn repeat this process. In the end, all nodes in the network have the data of all nodes they are interested in.

Direct Diffusion

This is a popular protocol in WSNs, and many derivatives have been implemented. The protocol is destination initiated and uses data-centric routing [2]. Queries are directed at certain areas of the network and not at specific nodes, or at the whole network. The system comprises three different stages. The interest diffusion stage is when the sink floods an interest in data through the network. This interest includes named data with a certain value, for instance "humidity" with a value of 50 or more. This request is, therefore, made to all nodes making humidity measurements with values above 50. A gradient for the data, or report rate, and *timestamp* is also included in the interest request. In the gradient setup stage, the nodes now set up this report rate for the interest. The *timestamp* gives the duration of the interest (for how long this data must be reported). The nodes now report data along the gradient path, allowing for data aggregation to take place. The nodes receiving the message only report it further if it is new data. Now when the data arrives at the sink it, in turn, reinforces the paths to specific nodes that reported the relevant data. This reinforcement can ask for a higher report rate and gives a different *timestamp* to give an extended report time to the node. In the data report stage, the node now sends data at the requested report rate along the reinforced path.

Minimum energy communication network

This protocol was proposed to work with motes that have variable transmit power. Energy can be saved in the transmission of packets by minimizing the energy used by each node in routing of the packet. An energy efficient sub-network of a given communication network is created. Each node is given a circular relay region which includes nodes close enough to minimize transmission power for each transmission hop. The protocol uses the *minimum-energy* property which states that in the network a minimum energy path exists between all connected nodes.

Manufacturer	Device	RAM (kB)	Flash (kB)	Active (mA)	Sleep (μA)	Release
Atmel	AT90LS8535	0.5	8	5	15	1998
	Mega128	4	128	8	20	2001
	Mega165/325/645	4	64	2.5	2	2004
General Instruments	PIC	0.025	0.5	19	1	1975
Microchip	PIC Modern	4	128	2.2	1	2002
Intel	4004 4-bit	0.625	4	30	N/A	1971
	8051 8-bit Classic	0.5	32	30	5	1995
	8051 16-bit	1	16	45	10	1996
Philips	80C51 16-bit	2	60	15	3	2000
Motorola	HC05	0.5	32	6.6	90	1988
	HC08	2	32	8	100	1993
	HCS08	4	60	6.5	1	2003
Texas Instruments	TSS400 4-bit	0.03	1	15	12	1974
	MSP430F14x 16-bit	2	60	1.5	1	2000
	MSP430F16x 16-bit	10	48	2	1	2004

Table 2.2: *A basic microcontroller history*

2.5 Hardware

Moore's Law states that about every 18 months the size of computer logic hardware will be halved. This means that the number of transistors you are able to fit in the same area doubles. It has been shown over the last couple of decades that this law is followed rather accurately. Another such law, Bell's Law, states that every decade we can expect to see a whole new class of computers. This is also an accurate statement if one looks at how we progressed from the monster vacuum tube computers of yesteryear, to the personal computer, PDAs and cellphones and are now moving into the field of micro computers that stream data to and from the physical world. These steady advances in the electronics industry have led to us now being able to realistically look at deploying large networks of small devices. This section describes some hardware relevant to sensor networks.

Operation	Telos	Mica2	MicaZ
Minimum Voltage	1.8V	2.7V	2.7V
Mote Standby (RTC on)	5.1 μA	19.0 μA	27.0 μA
MCU Idle (DCO on)	54.5 μA	3.2 mA	3.2 mA
MCU Active	1.8 mA	8.0 mA	8.0 mA
MCU + Radio RX	21.8 mA	15.1 mA	23.3 mA
MCU + Radio TX (0dBm)	19.5 mA	25.4 mA	21.0 mA
MCU + Flash Read	4.1 mA	9.4 mA	9.4 mA
MCU + Flash Write	15.1 mA	21.6 mA	21.6 mA
MCU Wakeup	6 μs	180 μs	180 μs
Radio Wakeup	580 μs	1800 μs	860 μs

Table 2.3: *Measured current consumption of Telos compared to Mica2 and MicaZ motes*

2.5.1 Microprocessors

A microprocessor sits at the core of most electronic devices these days. In sensor network motes you need a processor to read sensors, store and read data from memory and to control the radio communication. To control these actions and to implement smart protocols, the motes need a processor with enough computational power, coupled to high energy efficiency.

Table 2.2 gives a comparison of a number of microprocessors that have become available over the years. As indicated by the typical individual power usage, only some of them are suited to low power devices like sensor motes. Luckily, chip manufacturers have made great advances in supplying devices suited to ultra low power applications.

Some of the popular choices for WSN motes are the MSP430 processors from Texas Instruments (TI) and the Mega range from Atmel. These have a very low power consumption in active mode and they provide different levels of standby and sleep modes to further reduce consumption.

2.5.2 WSN motes

During the last couple of years more and more mass produced motes have become available. These have made WSN research and development much easier, as an effective

testbed can now be deployed very quickly and cost effectively. It is especially the MICA and Telos motes that have been most widely used in WSN applications. The first MICA mote was introduced in 2002 as an open experimental platform. With a low power 40kbps radio and ATmega128 processor, this was the first of a couple of generations of MICA motes to come. The MICA2 and MICAz use the same processor but the MICA2 uses the TI CC1000 radio that operates in the 868-870 and 902-928 MHz frequency ranges. The MICAz motes uses the TI CC2420 2.4GHz radio which is IEEE 802.15.4 and Zigbee compliant. The Telos motes[5] use the same CC2420 radio, but its processor is the MSP430, also from TI. These motes all have expansion connectors for external devices such as sensor boards with a multitude of sensing capabilities. Table 2.3 gives a current consumption comparison between the Telos and MICA motes.

The Telos mote has a clear advantage because of its much lower power consumption. It can also run at the low voltage of 1.8V which means that basically all of the capacity of two series 1.5V cells will be used, whereas the MICA motes will not do this. These three are all, incidently, the size of a two AA battery pack and usually ship with this included. Another big advantage is the USB capability of the Telos motes. The WSN can be accessed through this interface and the motes are also programmed through it. The MICA motes need an external serial interface for programming and PC connectivity.

Crossbow offers a wide range of sensor board add-ons for their mote which range from extra analog inputs to accelerometers, actuator relays, sensors and GPS modules. They also have a Cricket version of the MICA2 mote with a ultrasound transmitter and receiver for distance measurements. Moteiv now offers the Tmote Mini and Tmote Mini Plus, mote cores. The Mini is a 25x20mm version of the Tmote Sky in the industry-standard miniSDIO form factor. The Tmote Mini Plus is the same but with a 100mW power amplifier for a range of up to 500m. These cores make it possible to embed motes in other hardware for a compact and easily deployable WSN or pervasive computing[6] solution.

---

[5]This is the original name but Moteiv corporation markets their Telos motes as Tmote Sky

[6]Pervasive computing is the term given to embedded computers in everyday objects

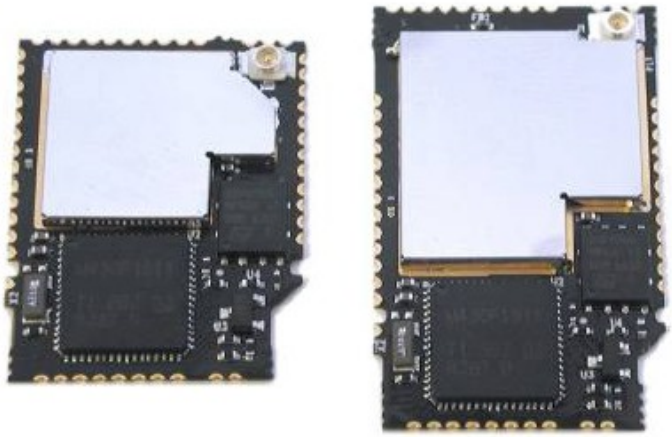


Figure 2.8: *Tmote Mini and Tmote Mini Plus mote cores*

2.6 Powering WSN hardware

2.6.1 Battery technology

The most commonly used cell is definitely the AA, and for further discussions this will be the cells that are referred to. This is usually the cheapest and most readily available power source. This is also why most available motes are designed to use around 3V (2 cells in series). A very good lifetime can be achieved with these if correctly used in low power electronics. The huge number of batteries available, with different capacity and current ratings, might make decisions on which ones are best for an application difficult. Most people can also tell from personal experience that these ratings are not always believable. For this reason the following discussion on popular battery technology and what could be expected of the different types, could be useful.

In the industry, batteries are usually rated in milliampere hour. This basically means that a 1mAh cell will power a 1 mA load for 1 hour. This can be a very inaccurate measurement, because different discharge rates have a huge effect on the efficiency of power delivery. This is also why non-rechargeable batteries usually do not carry such a rating.

Battery type	Typical household use	Capacity (mAh)	Typical drain (mA) designed for
D	Radio , Flashlight	12000	200
C	Radio , Flashlight	6000	100
AA	Clocks, cameras	2000	50
AAA	remotes	1000	10
9 Volt	Cordless mics, etc.	500	15

Table 2.4: *Approx. power ratings for common alkaline-manganese dioxide batteries*

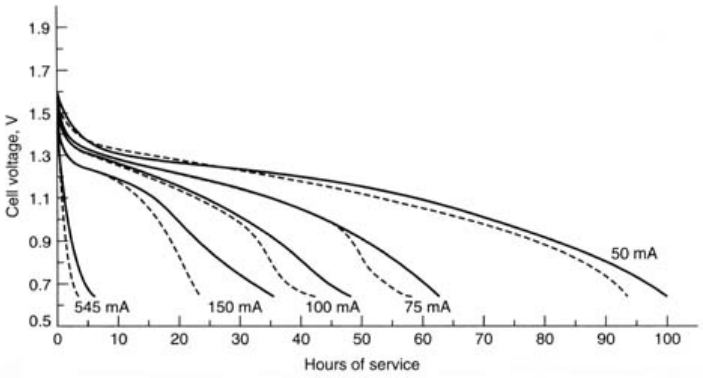


Figure 2.9: *Typical alkaline cell lifetime at different discharge rates*

Non-rechargeable cells

The most commonly available cells are non-rechargeable alkaline types. Table 2.4 provides the capacities and applications of typical cells. They are all designed for a typical drain current and if used far outside this range, results can be far worse. The voltage of alkaline cells steadily drops with usage. Maximum voltage is usually 1.54 V and it can drop down to a minimum of 0.9 V. At the 50 percent discharge point, the voltage is around 1.25 V. These cells exhibit an increase in capacity when warmed and a rapid decrease when temperatures drop. Figure 2.9 shows the different discharge curves at varying current.

Rechargeable cells

Shops stock mostly Ni-Cd and Ni-MH rechargeable cells. They can range from low capacity of a couple of hundred mAh to the biggest cells these days, at more than 2500mAh. Rechargeable cells can deliver much more of their rated capacity than alkaline cells, due to their more complicated design. Ni-Cd and Ni-MH cells have a lower voltage level

than alkalines, at 1.2V. They can still be used instead of alkalines, because they stay at this voltage level until they are almost depleted, where alkalines do not. The problem with these rechargeables is that they normally lose a few percent of their charge per day without a load. Freezing them can greatly reduce this effect.

Ni-Cd and Ni-MH cells do not differ very much in power delivery or lifetime. Ni-MH cells of the same size normally have a larger capacity and suffer less from the voltage depletion effect after multiple discharge cycles, than Ni-Cds. The biggest problem with Ni-Cd cells and why Ni-MH technology is better, is their use of the highly toxic heavy metal Cadmium. This is why they need to be recycled and not discarded. Ni-MH batteries require a somewhat more sophisticated charger than those used for Ni-Cd, and care should be taken not to use the wrong charger as this could lead to overcharging and damage to Ni-MH cells

2.6.2 Solar energy

Alternative energy has been a heavily debated topic in recent times. Solar energy is clean, renewable and is available everywhere most of the time (weather permitting). The earth receives about 174 PW of power at the level of the upper atmosphere. This energy is reflected, absorbed and affected by other atmospheric effects. Received solar energy at ground level has been measured at an annual average of about 6 kWh $/m^2$ per day in South Africa.

Photovoltaic cells convert solar energy into electric power and these have been around for many years. The efficiency of these cells has been increased many times over in recent times and at the moment most commercially available modules are about 15% efficient. This means that a square meter array of cells can produce an average of 900 Wh of energy per day in South Africa.

The benefits of powering devices with solar power is seen especially in remote areas where a normal electricity supply is not available and also where the replacement of batteries is an issue. For these reasons it is also a very good option for powering sensor networks. A Telos mote uses about 66mW of power at 3V with its processor and radio switched on. This equates to about 1.59Wh of energy use each day. It can now be calculated that a 5cmx5cm solar panel with 15% efficiency can produce 1.8Wh per day and therefore produces more than enough power to run one of these motes indefinitely. The problem is that about 90% of this is delivered in a 7 hour period, so one would need to store this energy efficiently for the rest of the day. Therefore, rechargeable batteries and charger

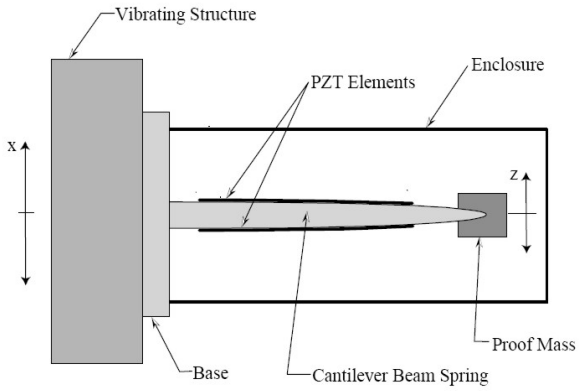


Figure 2.10: *Piezoelectric energy harvester model*

electronics are required with each mote. This, and the cost of solar panels, make it a much more expensive approach than only using batteries.

2.6.3 Piezoelectrical energy harvester

Piezoelectric materials (PZT) can be used to great affect to power low consumption devices. When exposed to vibrations straining the PZT element, it converts these vibrations to electric energy. In [4], Microstrain Inc. reports a test system used to power a wireless network of strain gauges. Figure 2.10 shows a model of the energy harvester. The PZT elements are mounted on a tapered cantilever beam (50 mm long) with a 250g proof mass (resonant frequency of 60 Hz) at the end. This creates a nearly uniform strain on the elements.

This system was shown to supply up to about 3mW of power with 57 Hz vibrations between 100 and 130 mGs. As shown in section 2.5.2, modern motes use much less power than this in standby mode. With duty cycling techniques it is therefore possible to run a mote almost indefinitely with such a harvester subjected to constant vibrations. A system that runs off this type of harvester has been implemented in airplanes for hull integrity checking.

2.7 Summary

This chapter contained sections on various relevant areas studied to enable good system implementation. Radio communication will be used between motes, so the different effects to be expected when working with RF based motes were studied. A number of communication spectral bands are available, but because of standards and hardware availability (more in next chapter), the 2.4GHz band was selected.

Coverage and connectivity are important aspects to investigate in sensor networks. Coverage in the sense of sensor coverage and communication coverage is relevant in designing a WSN. In this field the Voronoi diagram was shown to be a very handy geometric tool for calculating best and worst case coverage and connectivity regions.

Nodes might require their location information, so localization is another issue. Some ideas were presented that could be used to localize nodes very accurately. In our system, however, this is not a critical aspect. This is only required in a "communication localized" sense, so a signal strength system is used to aid the routing protocol.

Routing of packets in WSNs can be a tricky subject, as range and power is limited. This can produce novel ideas on how to route data efficiently and reliably. We decided to use a routing scheme that is power aware, to try and prolong the battery life of motes. Another type of scheme that produce better throughput or route data more reliably is not necessary, as these are not important features in low data rate WSNs.

The hardware required to implement these ideas must meet the energy and communication criteria in order to be a good choice for sensor networks. A very good WSN platform was found in the Tmote Sky modules, and the community of users that work on the associated open source operating system, TinyOS, made this a good choice. Powering a sensor network can be achieved in a couple of ways. The cheapest and most commonly used way is with AA batteries. However, the decrease in the price of solar cells and the improvements to vibration energy harvesters, would make these viable options in the near future.

Chapter 3

Hardware implementation

In section 2.5 the range of hardware platforms available for the WSN was considered. As discussed, the Telos motes from Berkeley are a very good choice for a low power system. They were also the cheapest and most easily acquired motes available in South Africa. In this chapter relevant information will be given on the motes and also on other hardware that was added on for more functionality.

3.1 Tmote Sky

The Telos Revision B module [26] [32] was an upgrade of the original model designed by Berkeley. The Tmote Sky module is made by the Mote*iv* corporation to be functionally equivalent to the Telos B motes. They provide a very easy way of deploying and testing low power sensor networks. Some of the key features that makes these motes a better choice than others currently available are provided below. [1]

- IEEE 802.15.4 interoperable Chipcon Wireless Transceiver
- Largest on chip RAM size (10kB)
- Integrated onboard antenna with 125m LOS range
- Optional integrated humidity, temperature and light sensors
- Ultra low current consumption

[1] Circuit diagrams of the motes are given in appendix A for reference.

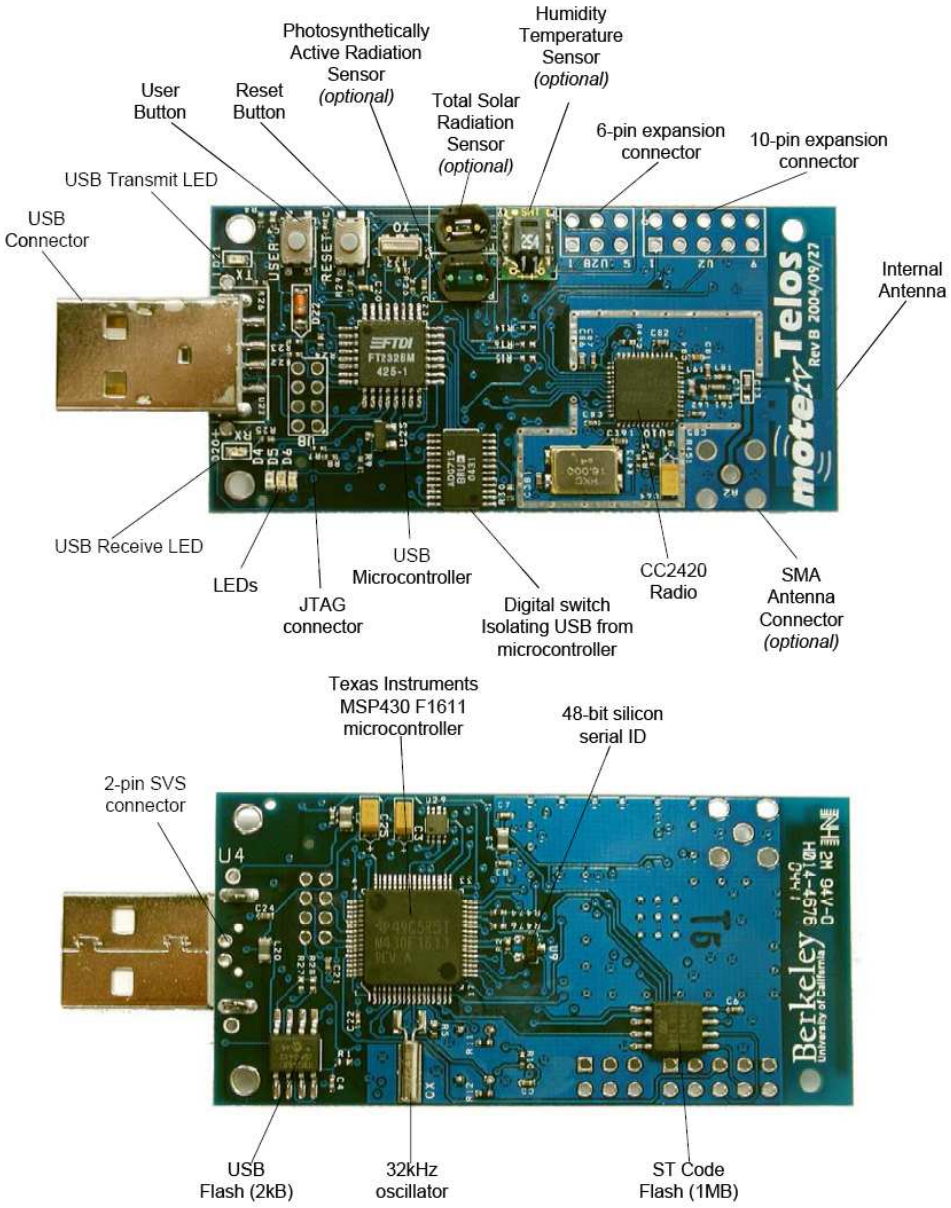


Figure 3.1: *Tmote Sky module*

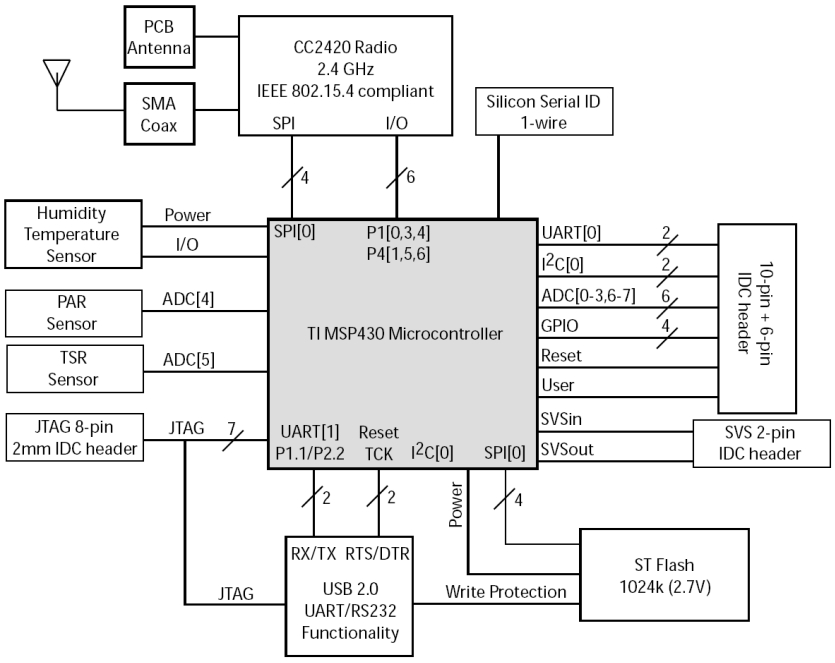


Figure 3.2: *Tmote Sky Block diagram*

- Fast wakeup from sleep (less than 6us)
- Hardware protected 1Mb external flash memory
- Programming and data collection via USB
- Hardware link-layer encryption and authentication
- TinyOS support: mesh networking and communication implementation
- Network programming support

Figure 3.1 shows the Tmote with all the major parts marked. It is easy to see the small size of the motes compared to the USB connector. The block diagram of the mote in figure 3.2 shows the peripheral devices connected to the microprocessor. The processor is connected to a USB controller via the RS232 serial interface. This makes it possible to connect the node to the USB port of a PC and to use the connection as a normal serial COM port for programming and data interfacing. A 1024k flash chip, also interfaced via SPI, can be used for data storage or program images.

PA LEVEL	Output Power (dBm)	Current Consumption (mA)	RSSI
31	0	17.4	42
27	-1	16.5	40
23	-3	15.2	39
19	-5	13.9	36
15	-7	12.5	34
11	-10	11.2	30
7	-15	9.9	25
3	-25	8.5	20

Table 3.1: *Radio power levels*

3.1.1 Radio

The CC2420 radio is controlled via the SPI interface and can be connected to either the onboard antenna or the optional SMA connector that can be used for a 2.4 GHz external antenna. The radio can be switched off and on by the processor as needed, making it possible to use a low power duty cycle system. The radio also has support for different transmit power levels. Table 3.1 gives the eight different settings, with their corresponding output power levels, current consumption and measured RSSI value at antenna output.

3.2 Sensors

The processor also has a number of ADC and GPIO ports connected to two expansion connector headers for external transducers. Figure 3.3 shows the 6 and 10 pin headers. These supply 6 ADC ports and 2 GIO lines as well as a UART connection for serial devices and the I2C bus. There are also lines connected to the reset and user buttons for external triggers, if these should be necessary for an application.

3.2.1 On chip sensors

The MSP430 microprocessor has internal voltage and temperature sensors, usable through the ADC interface. Because of the possible outdoor use of the motes it is convenient to have a temperature gauge, for system safety. This sensor is internal and therefore shows

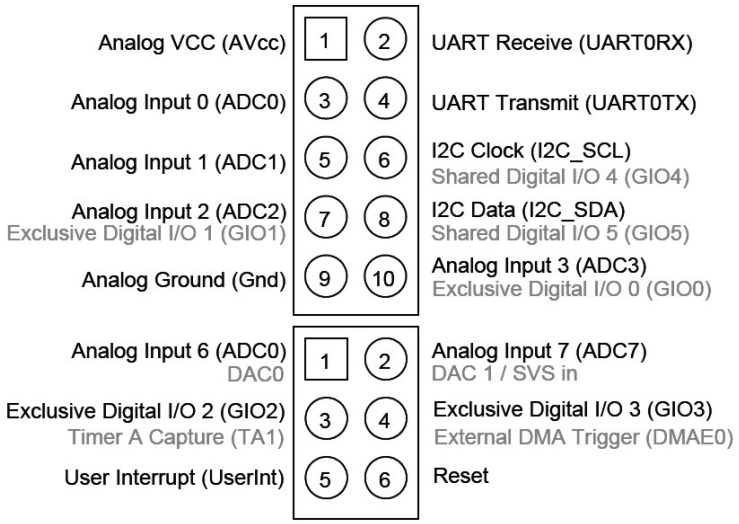


Figure 3.3: *Functionality of the 6 and 10 pin expansion connectors(Alternative pin usage in gray)*

the actual hardware temperature, which varies with processor use. The voltage sensor will be used for battery monitoring to support the low power protocol stack.

Temperature

The temperature sensor consists of an uncalibrated diode. This means there is an offset in the reading, and this should be corrected before the sensor is suitable for accurate use. This offset can easily be calculated using an accurate sensor to get a temperature measurement at the same location as the sensor that has to be calibrated. The linear equation 3.1 can now be used to calculate the actual temperature from the output voltage of the sensor.

$$V_{TEMP} = 0.00355(TEMP_C) + 0.986 + (offset) \tag{3.1}$$

Figure 3.4: *The SHT11 Temperature and relative humidity sensor*

Voltage

The 12-bit ADC monitors the output of a voltage divider circuit. To get the voltage reading this raw ADC output can be converted with equation 3.2.

$$DV_{cc} = \frac{ADCCounts}{4096} \times V_{ref} \times \frac{2R}{R} \tag{3.2}$$

3.2.2 Mote integrated sensors

The integrated sensors are an optional extra when buying the Tmote Sky modules, but can easily be added later if needed. Only the humidity sensor has been implemented on some nodes in our system.

Humidity and temperature sensor

The available port can be used for the SHT11 or SHT15 model sensors from Sensirion. The SHT11 shown in Figure 3.4 was used. The difference between the two is only that the SHT15 produces readings of higher accuracy. These sensors are calibrated with their calibration coefficients saved on the sensor's own EEPROM. The sensors are coupled with a 14-bit A/D converter which has a digital output. Relative humidity from 0% to 100%

can be measured in steps of 0.03%. The temperature sensor has a range of -40 to 123.8 degrees with a 0.01 degree resolution.

The relative humidity (RH) sensor is non-linear and to calculate the relative humidity from the sensor reading equation 3.3 must be used. The humidity readings obtained are very accurate, but in temperatures very much above or below 25 degrees Celsius, the temperature coefficient of the RH sensor must be taken into consideration by using formula 3.4.

$$RH_{linear} = -4 + 0.0405SO_{RH} - 2.8 \times 10^{-6} SO_{RH}^2 \tag{3.3}$$

$$RH_{true} = (T_{\deg C} - 25) \times (0.01 + 0.00008SO_{RH}) + RH_{linear} \tag{3.4}$$

The SHT11 temperature sensor is a very linear bandgap PTAT (proportional to absolute temperature) sensor. The temperature can be calculated from the sensor reading by using formula 3.5[2].

$$Temperature = -39.55 + 0.01SO_T \tag{3.5}$$

[2]Note that the constants in this formula are dependant on supply voltage. See data sheet for more information

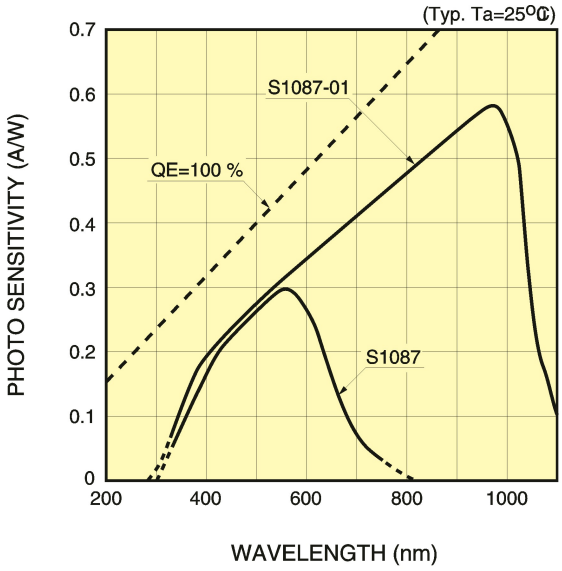


Figure 3.5: *Photo sensitivity of the light sensors*

Total solar radiation sensor

The default photo sensor is the S1087-01. Light in the visible spectrum is measured. This is an analogue sensor connected to ADC port 5 of the processor.

Photosynthetically active radiation sensor

The default sensor is the S1087 and it measures only solar radiation used in photosynthesis. This sensor is also analogue and connected to ADC port 4 of the processor. Figure 3.5 gives the sensitivity ranges of the two different types of light sensor.

3.2.3 Analogue sensors

To illustrate the ease of use of the external ADC ports, some basic sensors were added for extra functionality in testing. The possibilities of what can be measured are almost endless, as multitudes of sensors measuring light, pressure, speed, etc. are obtainable in basic analogue format.

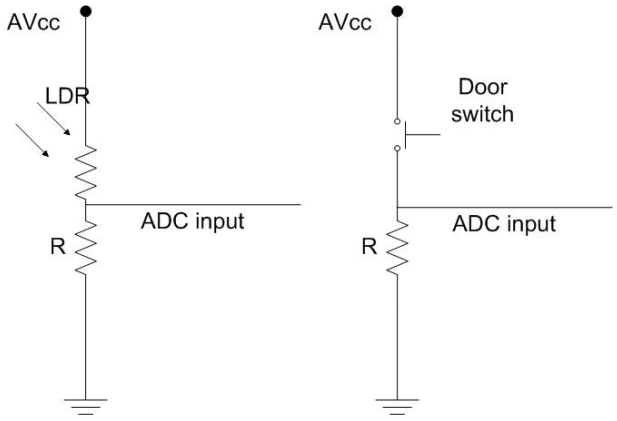


Figure 3.6: *Analogue sensor add-on circuits*

Light intensity

A light dependant resistor connected to the ADC port 0 was used to measure light intensity as the mote's optional light sensors were not available. Figure 3.6 shows the basic circuit that connects this to the ADC port, as well as the open door detector. The value read at the ADC port will depend on light intensity, the properties of the LDR and also the resistor R. The parts can be chosen to give readings in a required range.

Door open detector

A detector to determine whether a door is open or not was added to one of the ADC ports. It is a basic switch connected to the door. If not connected (door open) a 0 V measurement is made and if the switch is activated (door closed), it will produce a higher reading and the door will then be recognized as closed.

3.3 Summary

This chapter has presented the Tmote Sky, used as the platform for implementation of our system. The basic mote layout has been shown, with the most important parts explained. The hardware was mostly used as is, but some additional sensors were included, enabling system deployment for a test network. Also included in the chapter are the methods required to be used to accurately interpret sensor information for real world data acquisition.

Chapter 4

Software

4.1 TinyOS

Figure 4.1: *The TinyOS Logo*

TinyOS is an open-source event based operating system (OS) designed for wireless sensor networks. The OS has a component based architecture which enables fast implementation and also minimizes memory usage. The TinyOS core requires as little as 400 bytes of code and memory combined. Because all TinyOS code are open source, the growing community of users is actively contributing code and numerous groups are working together to establish standards and interoperable networking services.

The TinyOS component library comprises network protocols, distributed services, data acquisition tools and sensor drivers. These components can be combined as standard to implement a network, or they may be refined and extra code added for more specific applications.

4.1.1 Boomerang

Boomerang is the Mote*iv* certified distribution of TinyOS. This consists of components specially made to be compatible with all Mote*iv* hardware. This includes a ready to use multihop communication system and complete example programs with server-side tools. An extensive sensing library for compatibility with common Tmote Sky add-on hardware is incorporated. This system is fully compatible with existing TinyOS 1.x applications and also incorporates key features of TinyOS 2.x. It is not intrusive on other systems as it can perfectly coexist with an existing TinyOS installation.

4.1.2 NesC - Network embedded system C

NesC is a component based C dialect for embedded systems such as motes, and has been used to implement TinyOS. NesC separates the construction and the composition of programs. Components are assembled or "wired" together to form whole programs. Components are split into two scopes. One defines their specification, containing the names of their interfaces, while the other consists of the component implementation. Threads of control pass to a component through its interfaces and these threads are rooted in the form of tasks or hardware interrupts.

Interfaces in the specification may be used or they may be provided by the component. The interfaces specified in the component represent the functionality provided to the user by its implementation. The interfaces utilized, on the other hand, represent other components that are required to perform a particular job. These interfaces are bidirectional, because they specify a set of functions (commands) implemented by the provider and also a set to be implemented by the user (events). This is necessary because the completion of lengthy commands in TinyOS (like send packet) is signalled through an event. A component can therefore only use the commands provided if it, in turn, provides an implementation of the "command done" event. Commands will typically call downwards from application components towards the hardware level, while events call upwards from the hardware.

4.2 Sense application

After hardware initialization, the sense application is the main program that starts up. This links in all the TinyOS components describing the other system layers, such as

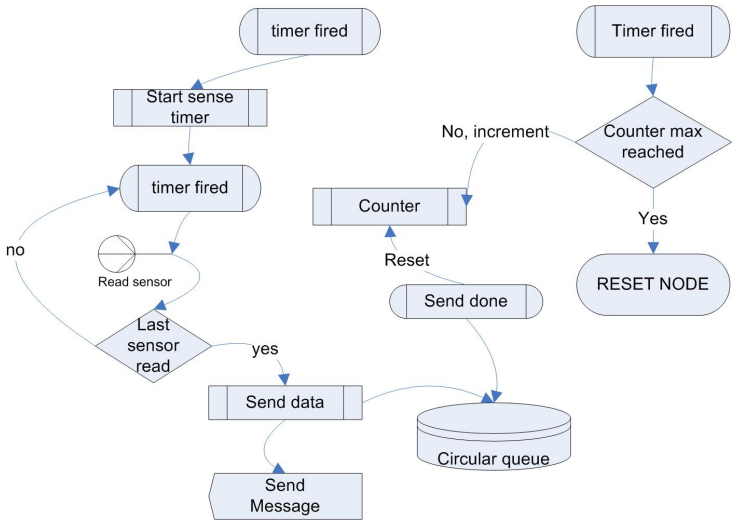


Figure 4.2: *Sense application block diagram*

medium access and routing. The program is driven by periodically firing timers and on execution of these events sensing and packet sending takes place. Figure 4.2 shows the main flow of the program. A timer fires at a set interval and this initializes another short timer (in the range of 50ms) which lets each sensor switch on in turn, to sense the surroundings. After all sensors are done, the program uses the multihop system to send data to the base station. All packets are stored in a circular queue until they have been sent successfully, which is reported by the *senddone* function. Also shown is the watchdog timer implemented to reset the system.

4.2.1 Sensor interfaces

TinyOS components are available for the SHT11 humidity sensor and the ADC ports. As the internal temperature and voltage sensors are connected to the ADC, these can be accessed by binding an ADC interface number to the actual input ports. In this way separate interfaces are created, with reference voltages to each port that needs to be read. The extra ports available in the expansion header can be used by creating similar interfaces. The I2C and SPI interfaces provided can also be used to construct components for any kind of required sensor that runs on these standards. The interfaces were incorporated in such a way that if a mote had a specific sensor connected, it would be initialized and used correctly. If the sensor was not connected, the mote would automatically not try to collect data off it and send a zero result. This enables the coding of all nodes in a network with the same software, regardless of which sensors are attached.

4.2.2 Watchdog timer

Even with a very stable system, something can always go wrong while running software on a microprocessor. In a sensor network application it is not acceptable to lose motes due to unforseen software errors, because resetting a mote physically is not always possible. The processor has support for setting a Watchdog timer that runs in the background, and when its timer is run down, the system is automatically reset. A counter is incremented each time a logging period passes and a packet is not successfully sent. If a set maximum is reached, the watchdog is started and the mote reset. Even with the relatively higher level language of TinyOS, support is still given to set basic hardware level registers in the MPS430 platform drivers. The code segment below sets the watchdog timer to run for 250ms and then reset the mote.

```
WDTCTL = WDT_ARST_250;
```

4.3 Multihop routing

4.3.1 Data routing

The MultihopLQI routing system from Mote*iv*'s Boomerang, was studied in depth and used as a base to develop the power aware add-ons. The system is split into two parts. The data message section handles only data message forwarding. Messages sent by the Sense application are received in this layer and then queued for transmission to its multihop parent using SP. Messages sent by other nodes are also received at this layer and a check is made to see if this node should do anything with them. If the packets require forwarding, they are also queued for transmission to the current node's multihop parent. The information on the parent and power level which is required for routing to take place is acquired by using the interfaces provided by the route setup section.

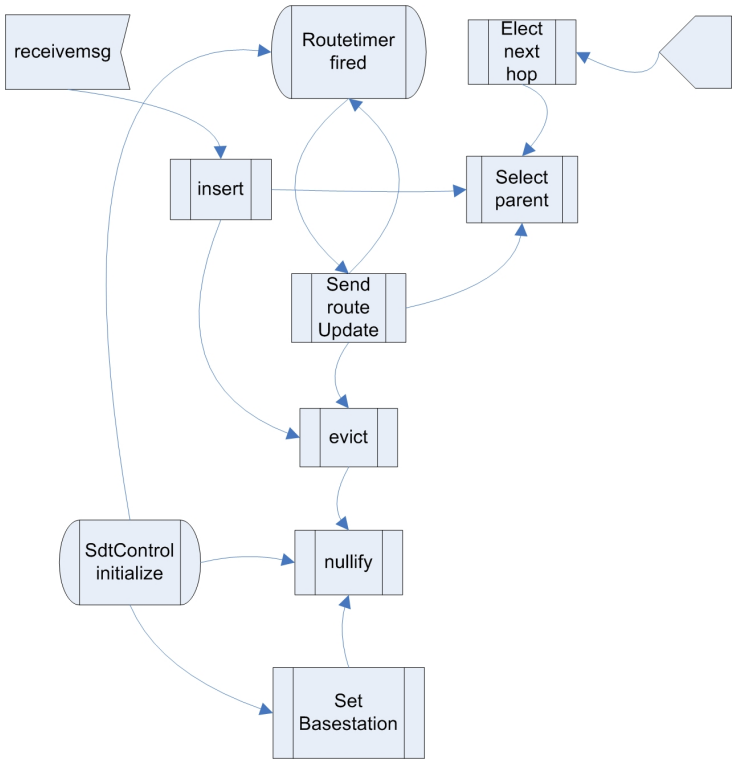


Figure 4.3: *Routing system functional diagram*

4.3.2 RSSI and Battery level route setup system

This part of the routing layer sets up the node's parents and other information required to route packets. Figure 4.3 gives the basic TinyOS function flow of the routing subsystem. At initialization of the routing system, the node first checks to see if it is the base station and then nullifies all parent information the memory might contain. A timer is then started, which fires at a fixed interval. Each time the timer is fired, all parents that have timed out are evicted and the best parent is selected. A route update message is now broadcasted to all neighbours, using SP [1]. Each time a route message is received from another node the information is examined to see if it might be a better parent. The new best parent is selected for routing to the base station. This subsystem provides the interface through which the next hop in the route is sent to the data sending subsystem. The following sections explain the functionality of the routing system by looking at the critical functions used in route discovery and how they control the route setup.

[1]Sensornet Protocol, which bridged the link and network layers. (Discussed in section 4.4)

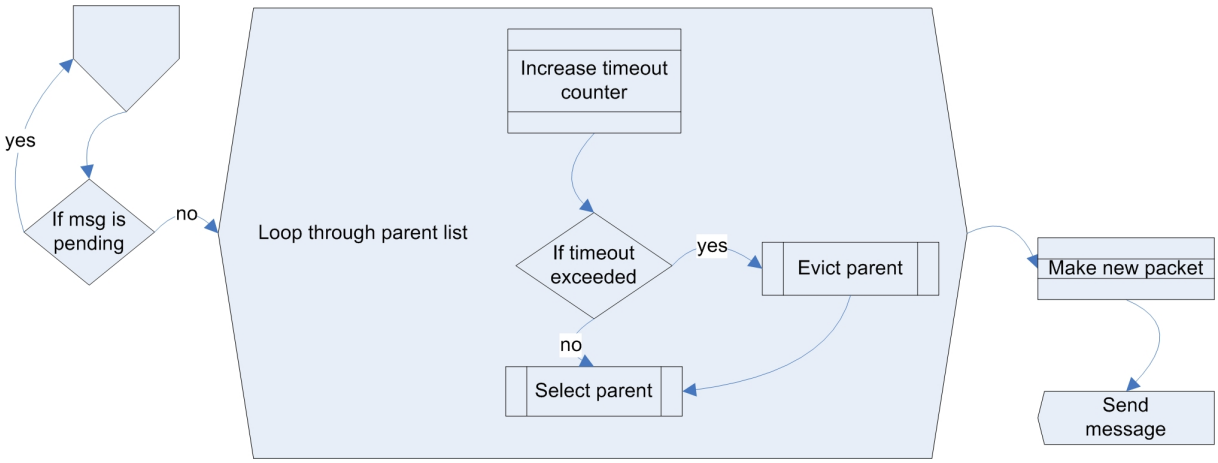


Figure 4.4: *Sendrouteupdate functional block diagram*

4.3.3 Sendrouteupdate

This function (Figure 4.4) is called when the route timer is fired to send the next beacon message. A check is made through the parent list to see if any parents have timed out. This will happen if they have not been heard from in the last couple of beacon message intervals. This ensures that no outdated information is sent, which could negatively affect route setup of neighbour nodes. The information is then inserted into the packet structure and sent, using SP. The message contains the following information:

```
16bit parent;
16bit cost;
16bit hopcount;
16bit batterylevel;
32bit timestamp;
```

The node's current multihop parent address is included, together with the hopcount and cost of transmission to the base station through this parent. The cost includes the cost of all hops to the base station from the current node. A timestamp is included, plus the node's current battery level, which is received with the *nodestats* interface provided by the sense application. The setup of the cost metric is described in the next section.

4.3.4 Route cost setup

The cost of transmission over a link is based on the power necessary to transmit and the available battery power of the node. The radio has different transmit power levels (see Table 3.1) which require anything form 17.4mA to transmit at 0dBm, down to 8.5mA for transmission at -25dBm. This means that power usage for the radio can be halved if two nodes are close together, and the radio range is adjusted accordingly. Setting the radio output power each time before a packet is transmitted is facilitated by using the radio control interface. The number (31) in the code below is the power amplifier level corresponding to the output power, as shown in table 3.1.

```
call CC2420Control.SetRFPower(RESOURCE_NONE,31);
```

The battery level of the node is also used to a degree to manually deteriorate links as battery levels go down. This ensures that nodes will not rely on a route through a node with low power. The power usage of the node is thereby limited somewhat and dropped packets due to node failure will be minimized. The LQI measurements (examined together with RSSI in chapter 6) can be used as an additional indicator, to ensure that very low quality links are not used. A lower cost path might be made through bad links, but depending on the importance of data, this can be limited.

4.3.5 Insert

The insert function is used each time a beacon message is received from another node. The function runs through the current parent list, which contains the following information for each of the possible parents:

```
addr;           Parent's network address
cost;           Parent's cost to base
batterylevel;   Parent's battery level
powerlevel;     Power level needed to transmit to parent
lqi;            Link quality
rssi;           Link RSSI value
hopcount;       Parent hopcount to base
lastheard;      Update intervals since last received message
```

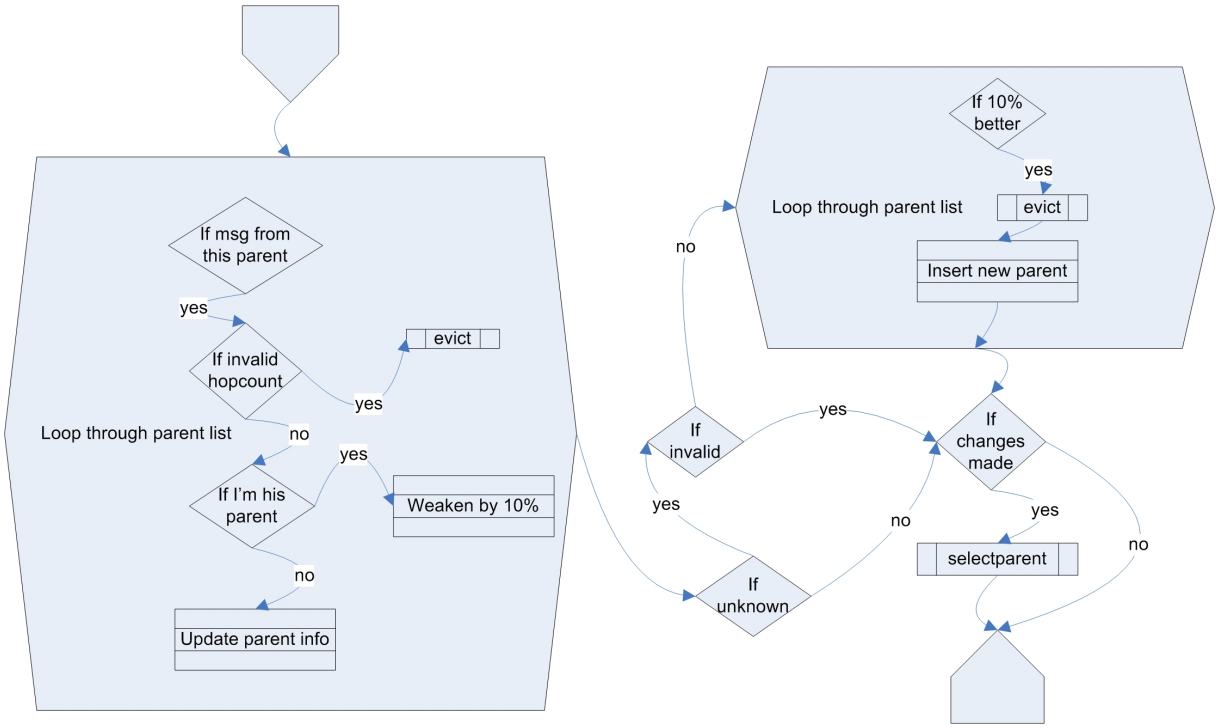


Figure 4.5: *Insert function block diagram*

If the function finds that the node in the list that has just sent the message now has an invalid hopcount, it will evict it from the list. This will ensure that nodes will not try to route messages through others that do not have valid parents. If, for some reason, the other node has set this one as its own parent, the link cost will be increased by 10 %. If, therefore, a loop is created between two nodes, the link will start to appear very bad after a couple of messages and another parent will be used. If there is no problem with the node the newly received information is inserted into the parent entry.

If the node is not found in the parent list, a check is made to see if it has a valid route to the base and that it has not set the current node as its parent. If this test is passed, a loop is again made through the current parent list and the new node will then replace some other node in the list, provided that its link is 10 % better. In the end, if changes have been made to the parent list, the new best parent is selected. This might be different from the previous one, as link costs between nodes change constantly. Even if no new nodes appear, the best parent today might not be the best one tomorrow. This can be due to a multitude of effects. See section 2.1.1 for radio propagation details.

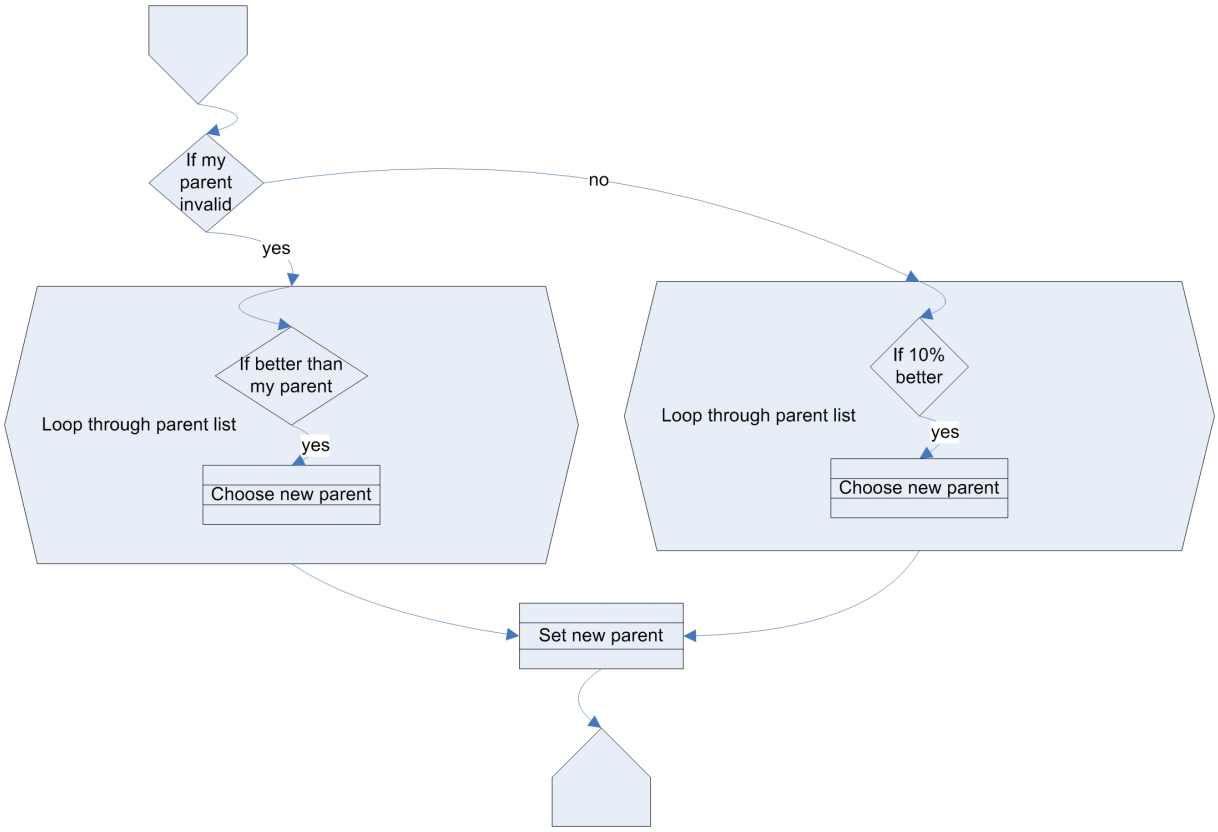


Figure 4.6: *Selectparent function block diagram*

4.3.6 Selectparent

This function is called a couple of times in the route setup process to ensure that the best possible parent is always used for packet routing. A check is always made to see if the node has a valid parent. If it does not, the first one in the parent list is selected and its cost is compared to the others. The best one is then selected. This will ensure that there is always a selected parent.

If the node has a valid parent, it is compared to all the others in the list and a switch is only made if the other one's link is 10% better. This hysteresis limit is imposed to combat the unnecessary switch between parents with similar link costs at every update round.

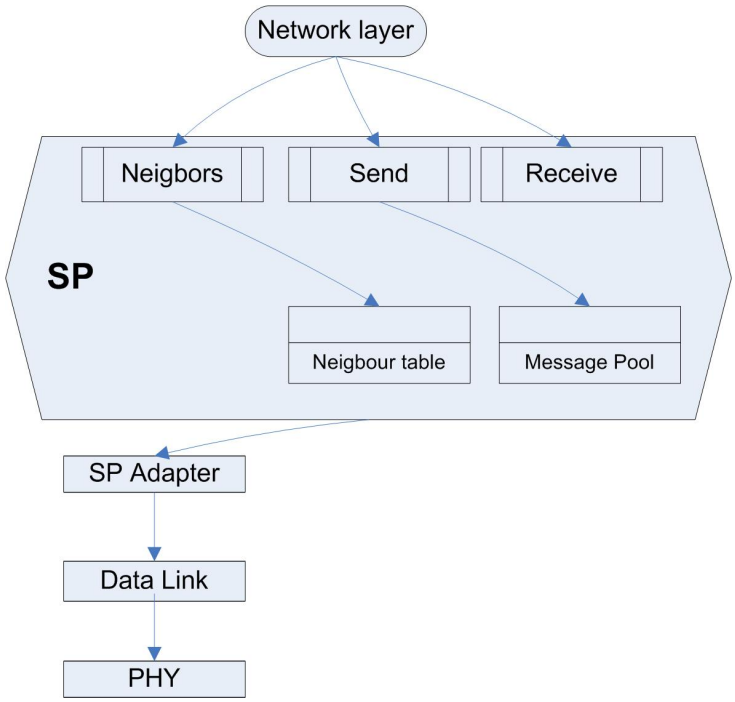


Figure 4.7: *SP connection between network and link layer*

4.4 SP : Sensornet Protocol A Unifying link abstraction

SP [30] [31] bridges the link and network layers. Figure 4.7 shows the interfaces provided by SP to link the network layer to the lower levels, down to the radio control component. This decoupling of the layers does not affect performance negatively but, in fact, provides the necessary primitives to implement efficient network protocols. SP supports many network protocols implementing a variety of functions like collection for data delivery and aggregation. This is, in fact, the main function of SP to let different network protocols coexist effectively. The SP abstraction can be implemented on a number of link technologies using different physical technologies, media access mechanisms and power management. SP performs three main tasks which are data reception, data transmission and neighbour management.

Data reception

When a packet arrives on the link interface it is sent to the associated network layer. TinyOS messages work on an Active Message (AM) type system and in this way a decision can be made in SP on which layer it must be sent to. In our system, Beacon messages will

be sent to the routing system for route setup and data messages will go to the data message part of the multihop system to be examined and sent to upper layers, or forwarded to other nodes.

Data transmission

This is implemented using a shared message pool data structure in SP. The different network layers submit messages to be sent to the pool when a link is available. After a message is sent, SP provides feedback to the network protocol. Messages pending in the pool can be inspected by the link layer, or other network protocols, to change their behaviour based on the message pool contents.

Neighbor management

SP allows the link and network layers to cooperatively maintain a list of usable neighbours. This information is stored in the neighbor table data structure, which contains the relevant information such as link cost, power information and scheduling of each neighbour.

4.5 Duty cycling system

The Netsync [2] system sets up radio duty cycling by synchronizing the nodes to switch their radios on and off at the same time. The base station sets up a schedule with a two second period. A 1% duty cycle, therefore, means that every 2 seconds the radio is only switched on for 20ms. This can equate to a large amount of power saving. The schedule information is sent to other nodes with a 24 byte packet containing the following information:

```
uint16_t addr;         // 2
uint32_t on;           // 4
uint32_t off;          // 4
uint32_t period;       // 4
uint32_t local_time;   // 4
uint32_t global_time;  // 4
uint8_t hopcount;      // 1
uint8_t seqno;         // 1 -- 24 bytes total
```

This enables the other nodes to synchronize their time to the base station's schedule and also to set the correct duty cycle period and ON and OFF times. At the start of setup the other nodes will keep their radios on until they have successfully joined the schedule. With a system running at a very low duty cycle it might be necessary to repeat this listen and setup phase more frequently, to ensure stability.

4.6 Deluge network programming

Deluge [3] provides an effective method to send large data objects like program images through the wireless network to many nodes. This, combined with a bootloader and command dissemination, provides an effective network programming mechanism. Deluge offers multihop support for reprogramming large networks without having to work with individual nodes. A continuous epidemic propagation method is used to ensure that program images reach all nodes. Overlapping CRC calculations are made to ensure the integrity of program images at all nodes. The system also lets nodes store multiple

[2]The name given to the TinyOS module that sets up the radio dutycycle.

[3]The name of the TinyOS module that enables network programming.

program images to make it possible to switch between programs without continuous code uploading. An isolated bootloader is used, that will execute at each reset regardless of what program is used. This bootloader programs the microcontroller with a program image and will recover from errors by installing a golden image that cannot be overwritten over the network, but only by means of a USB link to a PC. The reset button on the node can also be pushed three times, after which the golden image is installed. This makes it possible to always recover nodes without individual reprogramming, no matter what software errors might occur. This whole system uses only 150 bytes of RAM.

4.6.1 Using Deluge

The Deluge system uses the onboard memory of the mote to store program images. Before it can be used, the memory must first be formatted. The *flashformat* application, available with the TinyOS distribution is used to prepare the memory. When it is installed and the mote reboots, it starts reformatting the memory and will indicate via one of the LEDs when formatting is complete. The *Delugebasic* application can now be installed. This will install a basic Deluge program as well as TOSBoot, which is the bootloader for Deluge. With the initial system setup now done, there are a couple of commands in the Deluge java tool chain that can be used to control the Deluge network programming system. These are:

```
java net.tinyos.tools.Deluge --ping
java net.tinyos.tools.Deluge --inject --tosimage=<file> --imgnum=<imgnum>
java net.tinyos.tools.Deluge --reboot --imgnum=<imgnum>
java net.tinyos.tools.Deluge --erase --imgnum=<imgnum>
java net.tinyos.tools.Deluge --reset --imgnum=<imgnum>
java net.tinyos.tools.Deluge --dump --imgnum=<imgnum> --outfile=<xml>
```

You can *ping* a node to receive a command line printout with information on all currently installed images. This includes the image compile time and size. *Inject* is used to send a compiled program image to all the nodes and save it in a specified image spot in memory. The *reboot* function with a chosen image number reprograms all the motes in the network with the program image stored in that slot. *Erase* is used to erase a certain image from all the nodes. *Reset* erases the versioning information of a program image to combat the occurrence of cross-pollination of images between different networks that might exist in the same area. *Dump* extracts an image file from the network to the PC. This can be run on any node in the network to retrieve images if required.

This system can put some strain on communication while programs are rewritten, and accessing the memory frequently can drain a lot of power.[4] The system is, therefore, very well suited to simplify development, but overuse with deployed systems is not recommended.

4.6.2 Led debugging

The Tmote has an array of three LEDs, red green and blue, to give a method of visual output to the user. It also has LEDs to show sending and receiving of data to and from the USB interface. At power up, the LEDs have a certain flashing sequence to show when it has successfully booted. It will then flash the red LED a couple of times if the battery level has dropped too low to reprogram the mote. The mote also has another flash sequence while it is reprogramming.

The LEDs were used for easy debugging in different layers, to give an indication of program flow.

Sense Application

In the Sense application, the red LED was switched on when a data packet was sent. It was only switched off after sending of the packet had been successfully completed. When a data packet was received, the blue LED was flashed. In the lab situation, while doing development, the nodes are used close together and this flashing sequence gave a good indication of when and why packets were not delivered successfully. Also, if a certain route was tested and packets were received by the wrong nodes, the problem could be addressed accordingly.

Route setup

Similar to the Sense application two LEDs were used in the routing layer to show sending and receiving of beacon messages. This would show if messages were being sent at the interval given, and also if messages were successfully received by all the sending node's neighbours.

---

[4]It must be noted that the memory can only be reprogrammed while the mote voltage is above 2.6V

Packet routing

The LEDs could be used here to show the queueing of packets and their successful transmission. The reception of messages from SP was also shown, as well as the subsequent transmission of these packets to the upper or lower layers. Errors created by packets wrongfully forwarded to upper layers could be detected and addressed.

dutycycling

A LED was used here to indicate whether a node had a valid schedule. Another LED would show the radio state. An indication could also be given if schedule messages were sent and received and a LED could give an indication of nodes in the setup phase.

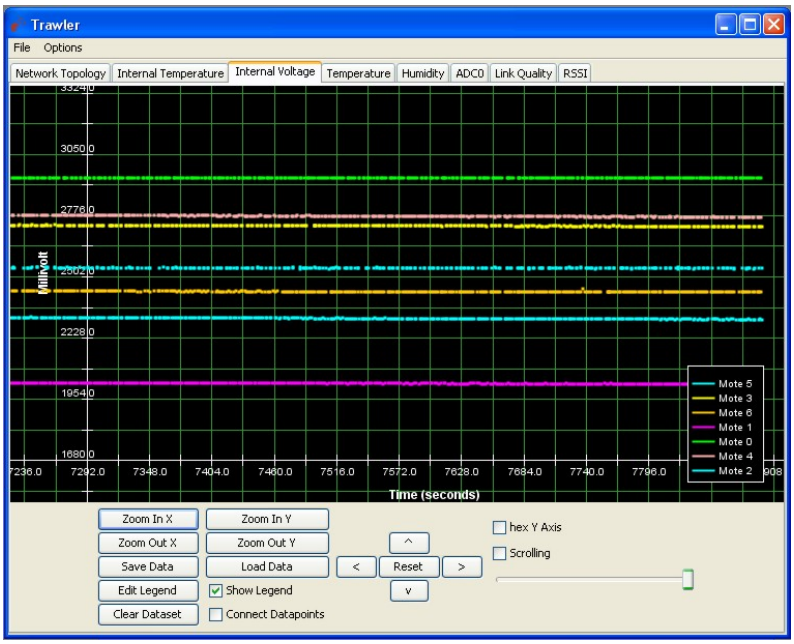


Figure 4.8: *Trawler user interface*

4.7 Java interfacing

Included with the TinyOS system are a range of Java applications, to provide an interface to the motes running TinyOS. Some programs can be used very easily to display data received from motes and they can also be adapted to cater for more specific needs.

4.7.1 Trawler user interface

This basic graphical interface comes with the Boomerang distribution of TinyOS. It was extended to display all information received from the motes. This includes panels to display LQI, RSSI or link cost between motes, which greatly helped development, as a continuous stream of data from the motes could be seen and analyzed in real time. Display panels for the various incorporated sensors are also included. Another good feature, provided by Trawler, is a graphical view of the network topology which is used to show all links between nodes, with their respective costs. See Figure 4.8 for the node voltage display in Trawler and note the tabs showing other information panels.

Figure 4.9: *The Serialforwarder java application which connects client applications to the motes*

4.7.2 Serialforwarder

The serial forwarder is used to connect to the base station mote. It reads all the messages coming from the motes and forwards them to other connected applications. This makes it possible to connect multiple programs to the same sensor network, and also to connect programs through a computer network. With this interface the WSN can be accessed through a program like Trawler from any computer on the same network as the one which has the serial forwarder connected to the mote base station.

4.7.3 CSV data logging

A basic logging program was written that connects to SerialForwarder and then interprets the packet data. The relevant readings were then converted with the right formula (see section 3.2) and the data stored in comma separated files. These files could now be accessed by an ASP web page, actively displaying the newest available data from nodes. This exercise gave good insight into the development of web based logging. The CSV files can also easily be imported into Matlab or any spreadsheet program for data plotting or processing.

The files are written as shown in Table 4.1. For each packet received a new row of information is added. This includes the packet time of arrival and sensor readings. In this example the voltage, internal and external temperatures and the humidity were logged every four seconds. A separate file is created for each network address, with the specific data columns as desired for that specific node.

Time	Voltage	Internal Temp	External Temp	Humidity
16:25:03	2509	19.14	18	53.57
16:25:07	2509	19.14	18.01	53.63
16:25:11	2507	19.24	18.02	53.6
16:25:15	2510	19.24	18.04	53.54

Table 4.1: *Segment of CSV file written by data logging program*

4.8 Summary

This chapter explains the TinyOS operating system as used in the network under discussion. The existing components were studied to find their strong and weak points. This led to to the effective use of:

```
Sense application
Power aware multihop routing
SP link abstraction
Netsync duty cycling system
Deluge network programming
LED Debugging
```

Delta is a basic application that logs temperature data, through the LQI multihop routing system(with the option of duty cycling) and SP to the base station with Trawler. This was used as a base to develop Sense. This application incorporated the other components as listed above. The LQI multihop router was studied in depth and then adapted to make it power aware. This included utilization of the battery level and RSSI information of neighbour nodes to make intelligent power saving routing decisions. Other incorporated features like the Deluge network programming system and SP link abstraction is explained, together with the use of LEDs for debugging purposes in the separate layers. The duty cycling system was used as a big energy saver, with some adjustments made to synchronization timing for stable dutycycling at different settings. A discussion on the Java interface programs used on the basestation PC is provided at the end. The extensions to Trawler and the CSV logging program facilitated development and test system deployment.

Chapter 5

Simulations

In this chapter the construction of a simulation model for the network is discussed. A few feasible simulation platforms are examined to find one best suited to our needs. The construction of the model is then explained, giving its simulation abilities and the different outputs provided.

5.1 Simulating sensor networks

A number of simulation packages are available for network simulations. NS2, Erlang and others have been developed and used for a wide range of simulations. The problem is that they usually do not have support for the more specific needs of wireless sensor networks. Simulation packages have better support for wireless communication protocols, but components for newer standards, eg. Zigbee, have not yet been fully developed and this creates problems when attempting to simulating them. Another problem is the design criteria of sensor networks, as opposed to those of other wireless communication systems. Very important performance measurements like throughput and packet delivery success rate take a back seat to power usage when dealing with sensor networks. For this reason a simulation package must be used that can cater specifically for the needs of sensor networks, as opposed to more standard wireless systems.

5.1.1 TOSSIM

TOSSIM [22] is a discrete event simulator for sensor networks running TinyOS. Code written to run on a mote can be compiled into the TOSSIM framework, which runs

on a PC. This allows for debugging, testing and analysis of algorithms in a controlled environment. Thousands of nodes running the same TinyOS program can be simulated simultaneously.

TinyViz is a java-based GUI that allows one to visualize and control the simulations while running. This makes it easy to inspect debug messages and the simulated radio and UART packets. Packets can also be statically or dynamically injected into the network.

TOSSIM focuses on simulating TinyOS and its execution and not the real world. It cannot, therefore, be used for absolute evaluations, because of all the assumptions that have to be made (for instance in radio propagation). Energy usage is also not modeled, which is one of the main areas in which we would like to have accurate simulation output.

Newer versions of TOSSIM are always in development and Power TOSSIM for TinyOS 2 has built in support for power usage simulations. This is a big step forward, but there are compatibility issues with TOSSIM and the Boomerang distribution of TinyOS. As Boomerang is mostly TinyOS 1 based and specifically designed for the Tmote Sky motes with some version 2 functionality, TOSSIM is not a conveniently usable platform for our system.

5.1.2 OMNeT++

OMNeT++ is a public-source discrete event simulation environment [1] and is free for academic and non-profit use. It is component based, modular and has an open-architecture simulation environment. The primary application is the simulation of communication networks but, because of it's flexible architecture, it has been used successfully in other areas like queueing networks, hardware architectures and even business processes. Several open source simulation models have been published, like TCP/IP and IPv6 for internet simulations, 802.11 wireless standard, mobility and ad-hoc simulations as well as models covering other areas.

OMNeT++ models are built from modules which communicate by exchanging messages. These modules are linked together into the required hierarchy of communication and the structure is defined using the NED language. NED can be edited in a text editor or in GNED, the graphical editor of OMNeT++. Figure 5.1 shows a proposed model for a sensor network node, where simple modules are combined to construct the *nic* which includes the MAC and PHY models. This and and other modules, such as a battery and network layer, are used to construct the node model. The active components(simple modules) are linked together with NED to build the model, which is programmed in C++.

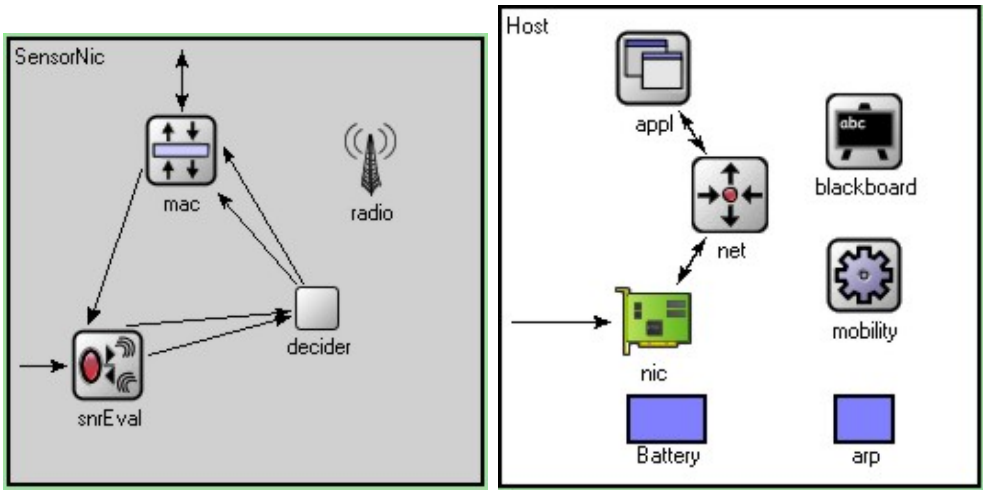


Figure 5.1: *GNED graphical simulation model buildup*

When the simulation program is built, it will run in a user interface showing command line outputs and interactive user interfaces. Simulation results are written into output vector and scalar files, which can be visualized by using the Plove and Scalars programs. Analysis of this data with packages like MATLAB and Octave is also possible.

While the community of users and available models is growing rapidly, the simulation of sensor networks is still problematic as models for standards like Zigbee, on which mote radios are based, are still not available. Writing accurate models for them can become a very time consuming job. In recent months more functionality, such as a battery simulation model, has become available. This growing number of models is helping to make OMNeT++ very easily usable and a great prospect for any network simulations in the near.

5.1.3 Truetime

Truetime [15] [29] is an event based simulation add-on package for Matlab/Simulink written in C++ MEX. It facilitates the co-simulation of controller task execution in real-time kernels, network transmission and continuous plant dynamics. Truetime supports both wired and wireless networks, with models for the IEEE 802.11b/g WiFi and IEEE 802.15.4 Zigbee wireless standards. This is one of the first simulation tools that offers support for 802.15.4 as standard. This, and the fact that a model for battery usage is available, made this the best choice for simulating our WSN.

Truetime adds an extra block library to the existing Simulink environment. The library (shown in figure D.1) has a couple of useful blocks that can be used in conjunction with

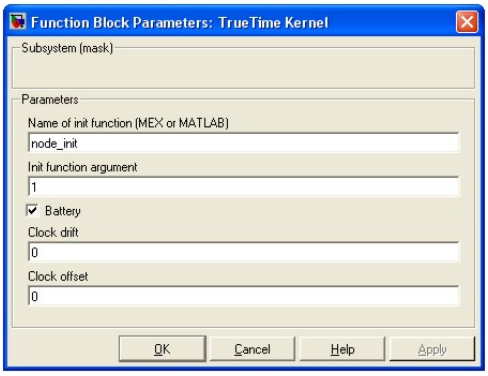


Figure 5.2: *Truetime kernel block mask dialog setup box*

standard simulink blocks to build simulation models. The usage of the blocks will be explained in section 5.2. The functions describing the system running on the model and simulation tasks can be written in Matlab m-files or C++ code.

5.2 Simulation model buildup

5.2.1 Node model

The wireless node model is built around the Truetime kernel block. The block has some parameters that can be set in the block mask dialog shown in figure 5.2. The Init function is the name of the Matlab script run to initialize the kernel block. This can be seen as the application run on the motes to set data delivery and so on. The Init function argument is an extra parameter that can be sent to the initialization script. In this instance, it is used to set the network address of the node in the simulation. The kernel can also be set to use a battery. This enables it to be connected to the truetime battery block. This was used to simulate the battery usage of the nodes. A clock drift and offset can be implemented for the node, but were not used as precise timing and synchronization were not used in our system.

Figure 5.3 shows how the kernel and battery blocks are connected with other simulink blocks to build the simulation model. The kernel has analogue input and output channels, of which the input was used to measure the battery level and forward it to the node function. Support is given for external interrupts and monitors, but these were not used. For power usage simulations, the kernel has an energy input port available which is connected to the battery so that the system has an indication that there is still sufficient power remaining for operation, as the node will stop functioning as soon as the power runs out.

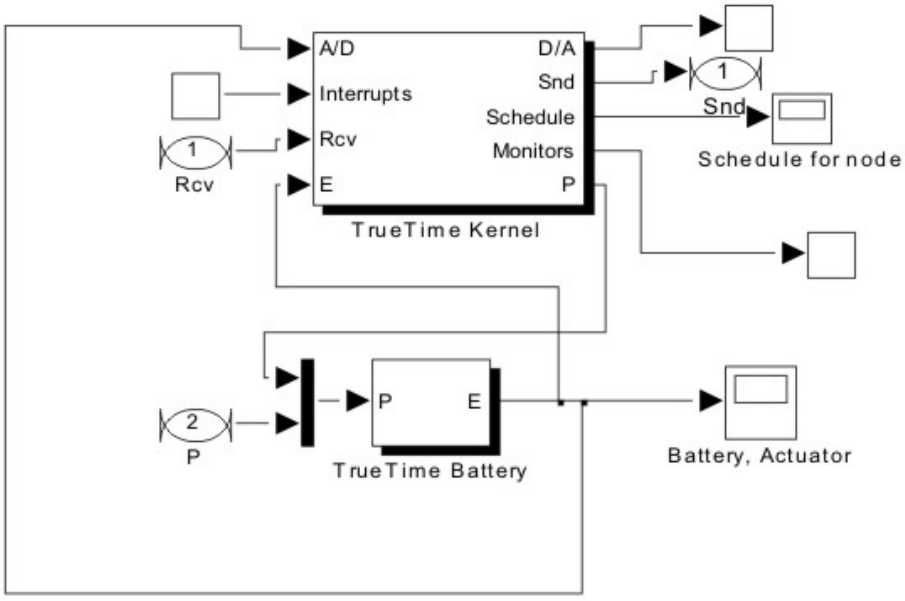


Figure 5.3: *Wireless node truetime simulation model*

The power usage output port is also connected to the battery and drains the battery as the system runs. Three other ports are used for connection to the rest of the simulation model. The send and receive ports are for sending and receiving packets through the wireless system. The power port connects to the network model to change power usage dynamically, as packets are transmitted and received. The schedule (example of schedule output in figure 5.8) gives a graphical output of packet transmission and reception.

5.2.2 Network model

The Truetime wireless network block simulates medium access and packet transmission in a wireless network. This block has a mask dialog box (Figure 5.4), where the type of network required can be set up and also network specific parameters. At the moment models are given for the 802.11b and 802.15.4 wireless standards. As we are using a 802.15.4 radio, this is the model that will be specified. Some parameters must also be set up to ensure precise simulations of the radio hardware and wireless transmissions, these are: **Network number** The number of the network block. This is to ensure that all nodes are connected to the correct network. This is only critical if multiple networks are simulated.

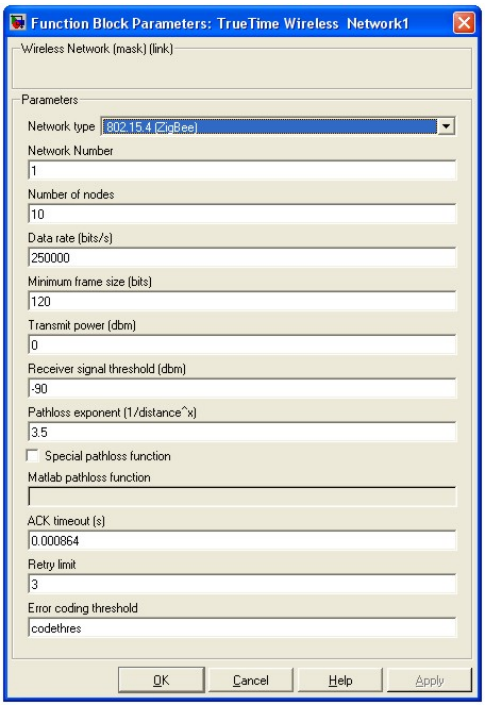


Figure 5.4: *Wireless network mask dialog setup box*

Number of nodes The number of nodes that are connected to the network is set, so that the network block knows how many send, receive and schedule channels are required.

Data rate (bits/s) This sets the transmission speed of the network. The CC2420 radio transmits at 250kb/s.

Minimum frame size (bits) Most network protocols have a fixed number of header and tail bits, so the frame must be at least the length of the tail plus the header. If it is not, the message is padded to reach the set minimum length.

Transmit power This determines the radio signal level and, thereby, how far the signal will reach. This is set to the radio maximum which is 0dBm.

Receiver signal threshold This sets the minimum receive level at which the radio will detect messages. this is set to -90dBm for the CC2420 radio.

Path-loss exponent The path loss of the radio signal is modeled as $1/d^a$ where d is the distance in meters and a is a suitably chosen parameter to model the environment, which is typically chosen in the interval 2-4 for standard over the air transmission.

ACK timeout This is the time a sending node will wait for an acknowledgement message before retransmission.

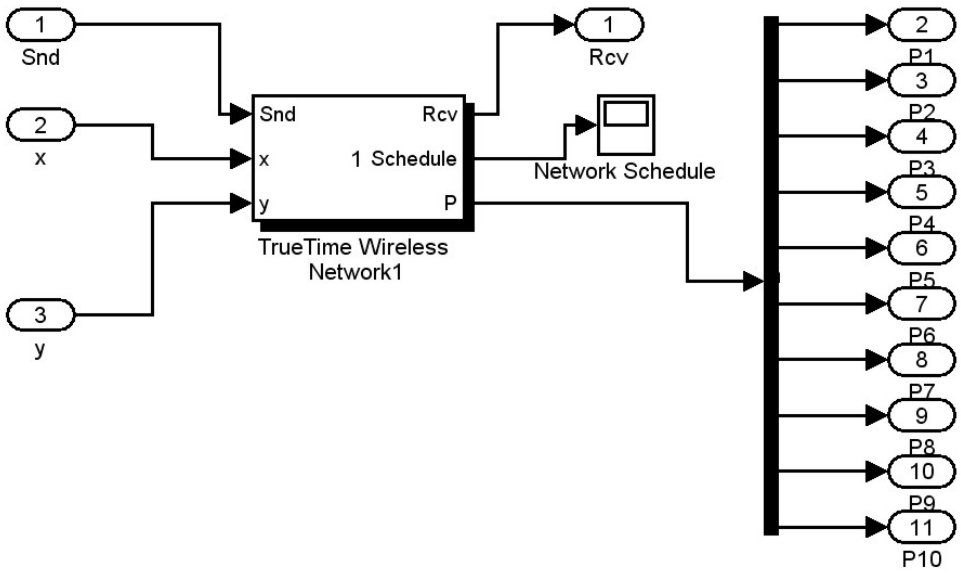


Figure 5.5: *Wireless network Truetime simulation model*

Retry limit The maximum number of times a node will try to retransmit a message before giving up.

Error coding threshold A number in the interval [0, 1] which defines the percentage of block errors in a message that the coding can handle. For example, certain coding schemes can fully reconstruct a message if it has less than 3% block errors. The number of block errors is calculated using the signal-to-noise ratio, where the noise constitutes all other ongoing transmissions.

The network model built with the truetime and simulink blocks is shown in Figure 5.5. When a node sends a message, a signal is sent to the network block on the send input channel. When the transmission simulation is done, the network block sends a message on the receive output channel to the corresponding node's receive port. The network block also has an input for the x and y coordinates of each node. A schedule output is given to provide information on the whole network's packet transmission. The power port is connected to all the nodes in the network to simulate the changes in power usage as packets are transmitted and received.

5.2.3 Overall model structure

The node and network models can now be used to construct the whole network as required for simulation. The node model was used as a single block with the three ports connected

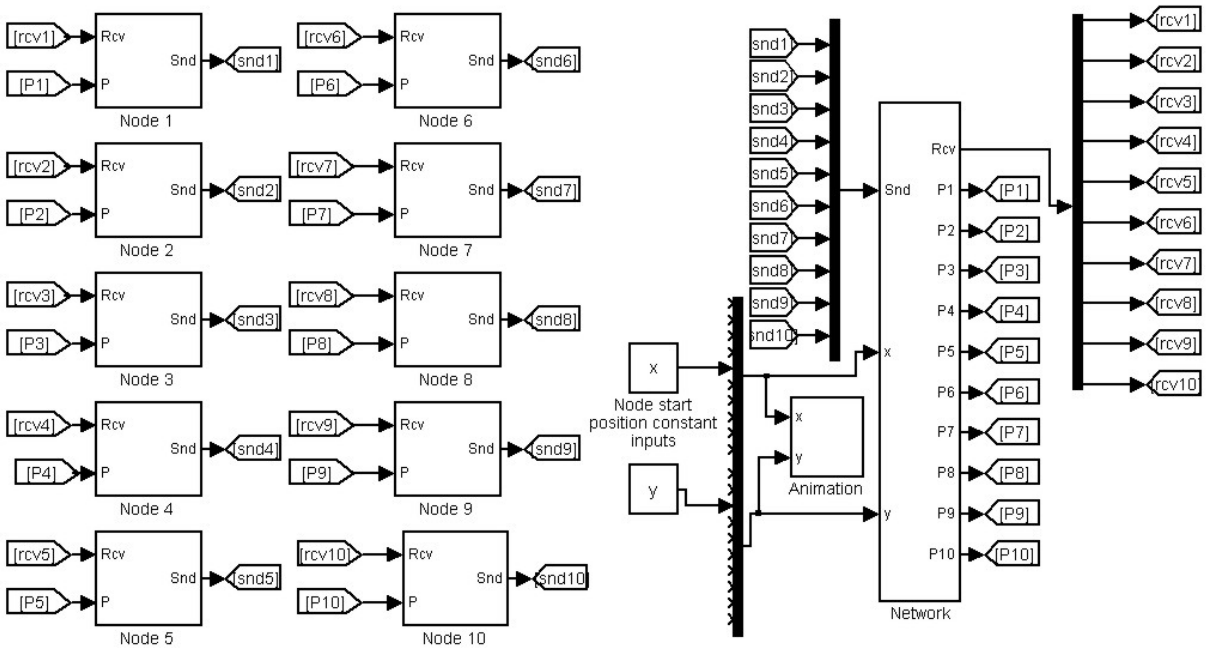


Figure 5.6: *A 10 node network Truetime simulation model*

to the outside. This block can now easily be duplicated to simulate multiple nodes. The network model was also built into a single block with the send receive and power ports to all nodes in the network on the outside. Figure 5.6 shows the overall system as implemented in Matlab. Also provided is the mote animation block which has as input the x and y positions of each node and then gives a graphical output of the node positions with their transmission range radii. This simulation can now easily be manipulated to include a larger number of network nodes by just pasting in more node blocks and adjusting the necessary parameters in the network setup.

5.3 Functions

The functions controlling the behaviour of the nodes built with the truetime blocks are written as Matlab m-files. It is also possible to use more simulink blocks to build the system, and less code. The idea was to keep the model of the network as close as possible to the real system running on the motes. For this reason the code for the main functions used in the TinyOS system was ported directly to Matlab code, for implementation in the simulation. The similarity between C++, NESC and Matlab code made it possible to adapt the code with changes only in syntax and minor functions where needed. This ensured that changes made in simulation code could be incorporated precisely in TinyOS.

5.3.1 Initialization functions

A main simulation initialization function is run first, when the simulation starts. This sets up initial power usage, global variables and structures needed in the simulation and also the node start positions. The node initialization function is now run for each node built into the network. Interrupt handlers used in the simulation are set up here, as well as periodic tasks like the sending of data and routing packets.

5.3.2 Node functions

The routing system was built on the same functions as described in Chapter 4, by porting the TinyOS code as described earlier. Each time a new routing message has to be sent, as set in the node initialization script, the *sendrouteupdate* function is called. From here on all other functions, such as *Insert* (to create a parent structure), are used precisely as on the motes.

Interrupt handlers were created to simulate battery usage. Each time change occurs in power usage, eg. when messages are sent or received, or a node goes from the OFF to the ON state (duty cycling system), the appropriate interrupt is called and the power consumption is set to the correct level until the system goes into another power state.

5.4 Simulation outputs

5.4.1 Text output

The simulation was written to create command line outputs in whatever function was required. This gave real time information on packet transmission, reception and in-function routing system reports. Every event was reported with a precise time stamp, node number and other relevant information. The precise moment when a certain node receives a routing packet and, subsequently, when the parent *insert* function executed and what checks are made to see if it should be inserted into the parent list can, for instance, be seen.

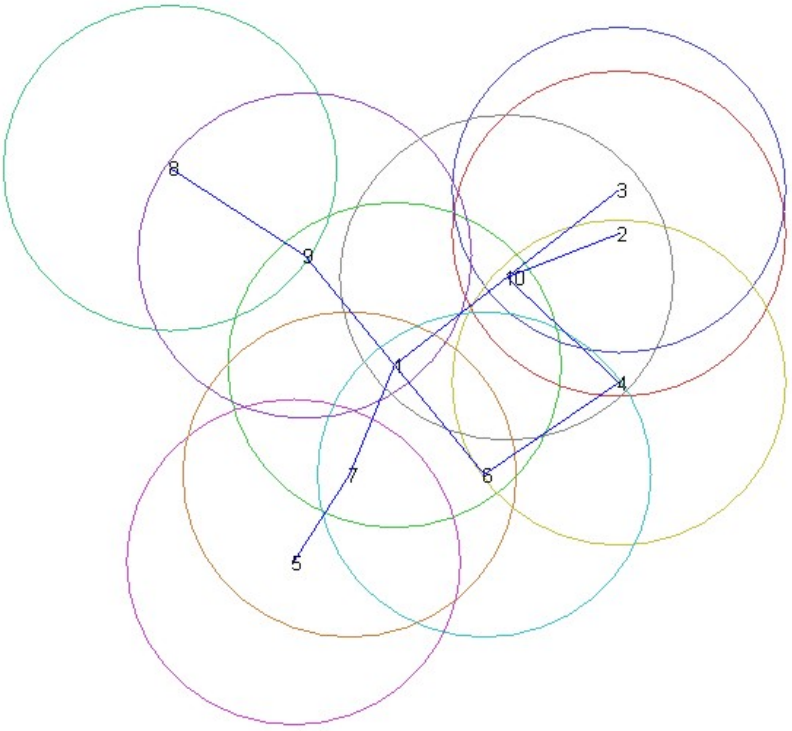


Figure 5.7: *Network simulation node visualization*

5.4.2 Node animation

Figure 5.7 shows the output figure made by the *moteanimation* Matlab function. The motes are represented by their address at their given locations in a 2D field. The circles represent their radio ranges, which are computed by using the radio parameters supplied to the network block. The lines connecting the nodes show the best links to the base station (node 1) that they have set up. It is also possible to change the node locations while the simulation is running, to simulate mobile nodes This movement will also be shown on the animation output figure.

Note the two paths of node four through node six and node ten. Node four first found node six to be its best path to the base station. After more rounds of route discovery messages, node ten was chosen, as this is the better option. This shows the effect of the paths filtering through from the base station to the outer nodes as usable links are set up.

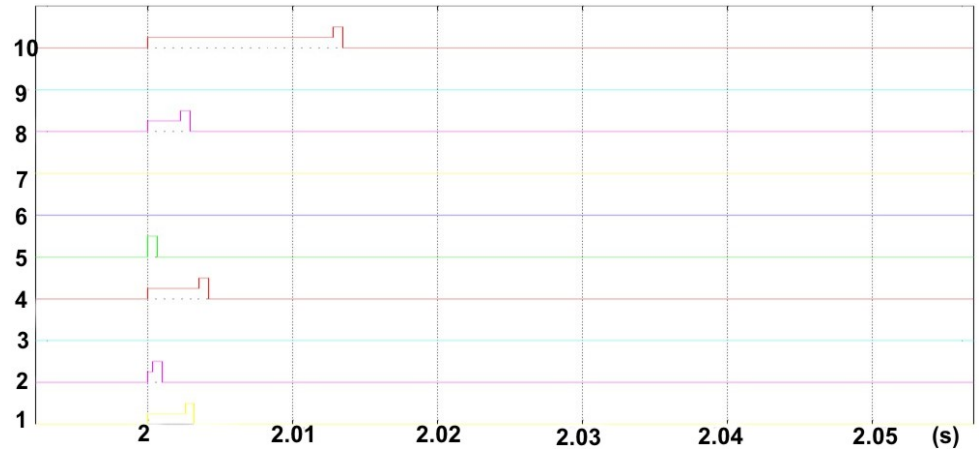


Figure 5.8: *Simulation network packet schedule*

5.4.3 Network schedule

The network schedule, given as output by the wireless network block, is a graphical presentation of packets sent over the transmission channel. Figure 5.8 is an example of a network schedule output over about 50ms showing the nodes' transmission of route setup packets. The channel activity for each node is represented on a line. This line advances to one level to show that the channel is activated and to a higher level when packet transmission takes place. The different transmission times for each node demonstrate the random backoff periods incorporated in the transmission system. With this information problems like channel congestion and any fluke errors, such as multiple unnecessary retransmissions, can be seen. Packet timing and transmission delays can easily be seen and used in calculations. Another use for the schedule output is to assist with debugging of the system. The route through which a packet is sent can, for instance, be monitored as can undesired reactions.

5.4.4 Battery levels

The battery level is the most important output obtained from the simulations. With this, the effects of the different battery saving techniques can be seen and compared to real life batteries. Figure 5.9 shows a battery simulation output over 150s. Note that the power usage graph will always be close to linear, as the network runs on periodic cycles. Therefore, only short simulation runs are required to get a very good approximation of power usage over a longer time. The battery capacity is measured in $Joule$ or Ws. This is the easiest unit to work with in simulations, as power requirement is measured in $Watt$ and this usage information, combined with the length of time at different power levels,

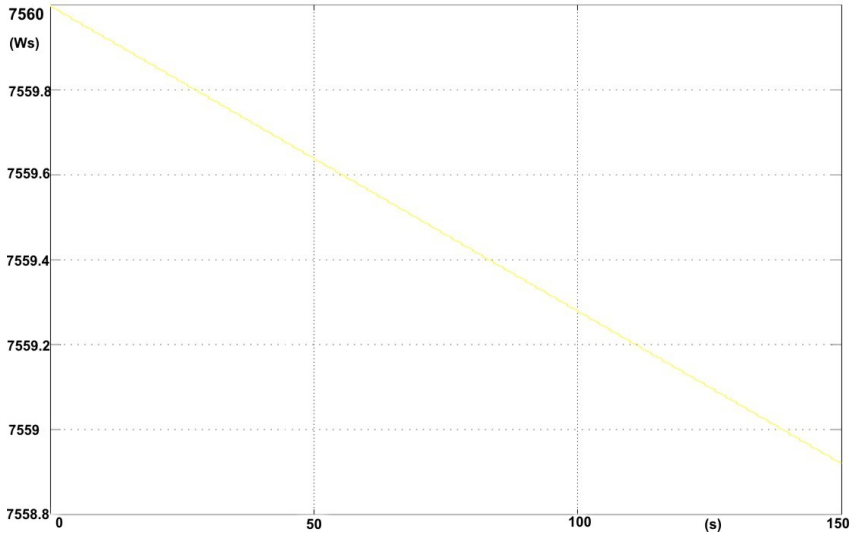


Figure 5.9: *Simulation battery usage output*

provides output measured in Ws. This unit can easily be converted to the standard battery capacity measurement unit, which is mAh. In the example below, 2500mAh cells at 3V are used, as this is the supply voltage and capacity of two AA cells as used in our motes.

$$\begin{aligned} 9000000mAs &= 2500mAh \times 3600 \\ 27000000mWs &= 9000000mAs \times 3V \\ 27000000mWs/1000 &= 27000Ws \end{aligned}$$

An 2500mAh cell can deliver a current of 2500mA for one hour, therefore, it can deliver 9000000mA of current for one second,(not physically possible, but just for the sake of explanation). At a supply voltage of 3V this means that 27000000mW of power can be delivered for one second; converting this to Ws gives a final value of 27000Ws, which is equivalent to 1W power delivered for 27000 seconds, or 20mW for 125 hours.

5.5 Summary

The simulation platform used to develop a model for our system is Truetime. It incorporates an accurate model of the routing system and uses the Truetime blocks to closely simulate wireless transmissions done by our transceiver. The most important development in the model was the extensions made to simulate the power usage of motes, as this is a feature that was not available with standard network simulation tools. This gave an easily scalable platform for network simulations, that helped in developing the actual system and can be used to great effect in power use modeling (as demonstrated in following chapters).

Chapter 6

Performance measurements

This chapter reports the results from measurements made while developing and testing the system. This includes measurements on the link quality estimators that were necessary to improve the routing system. Also included are the results of tests run to show the energy usage improvements made. A basic system was deployed and weather data monitored to observe environmental effects on the mote operation over an extended period of time.

6.1 Battery life

Our main focus was placed on measuring the lifetime of motes, and then improving on this, using the various methods described in Chapter 4. Different types of AA cells, as described in Chapter 2, were used to run the motes. By continuously logging the battery level over its whole lifetime, discharge curves for the cells and the total lifetime of the motes could be obtained.

The motes were set up for full power operation with no duty cycle. A beacon message rate of one every two minutes and a data message rate of one every thirty seconds were set for data logging. New sets of cells were used and the rechargeables put through a couple of charge and discharge cycles, to reach top capacity. All test motes were used in the same lab environment, with similar environmental conditions.

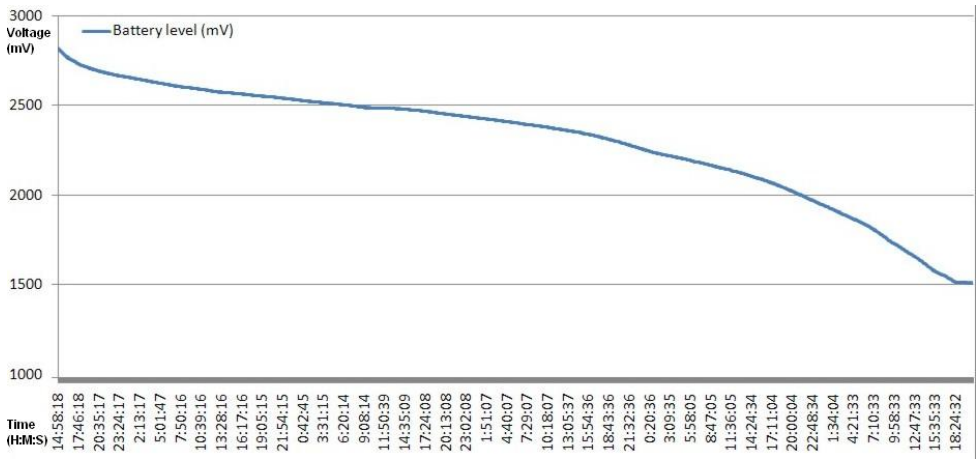


Figure 6.1: *Discharge curve of alkaline cells when used to power motes*

6.1.1 Alkaline

The discharge curve of alkaline cells (figure 6.1) shows the expected behaviour of a steady voltage drop over their lifetime. The mote logged data with no packet loss over the whole voltage range, down to the 1.5V limit. Around the measured level of 1.513V the motes started to die and recover intermittently for another 2 hours. This can be explained by the "recovering" effect of alkaline cells while current is not drawn. The motes had a lifetime of about 125 hours on a set of cells. This steady alkaline cell discharge curve provides a convenient way to predict battery level. As the motes will normally use a maximum of 21.8mA, the power requirements are low enough to make normal AA alkaline cells a good choice.

6.1.2 Ni-Cd

For the next test a set of 700 mAh Ni-Cd cells was used. The discharge curve in figure 6.2 shows that these cells stay at a combined steady voltage level of around 2.5V, for most of their discharge cycle. The voltage then drops down steeply to the 1.5V limit in the last couple of hours after about 2 days. This can be a good feature, because the higher level for longer periods of time will ensure that more devices on the node could continue to operate properly. For example, reprogramming of the mote with the Deluge system will be available for longer, because the external flash memory can still be used. The problem is that battery lifetime predictions cannot be made accurately, and a mote might seem viable, but then die within a couple of hours. This is not an issue for system communication, as there is enough time for different routes to be set up. A problem

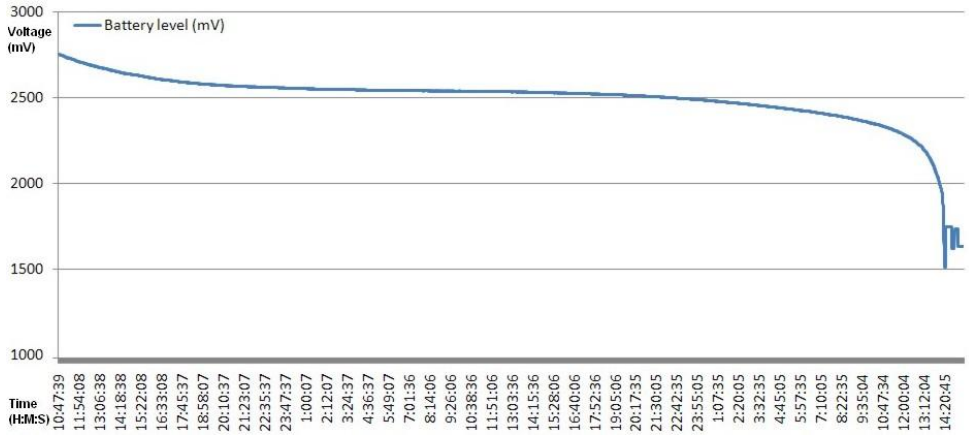


Figure 6.2: *Discharge curve of 700mAh NI-CD cells when used to power motes*

arises, however, if there are critical nodes that might need a battery change, but this fact is detected too late.

6.1.3 Ni-MH

A set of 2500mAh Ni-MH cells was used next. The discharge curve in figure (Figure 6.3) show that their discharge characteristics are very similar to that of the Ni-Cd cells, except that an even sharper voltage drop at the end of a cycle was noticed. The cells provided a mote lifetime of about 123 hours. This proves the fact that alkaline cells have around 2500mAh of capacity. As these cells give very much the same results as Ni-Cd cells they have the same pros and cons[1]. The problem with all rechargeable cells in long life applications is that they always lose charge, even under no load. The sets of cells were charged and their voltage tested daily without being used. They had a similar discharge curve as when they were used, and reached the 2.5V mark after eight days. Cells in use reached this level at about the halfway mark of their life. These cells will therefore, produce a poor lifetime in motes that might survive years (See next chapter) with alkaline cells.

[1]They are, however, safer to the environment and have other features as described in Chapter 2

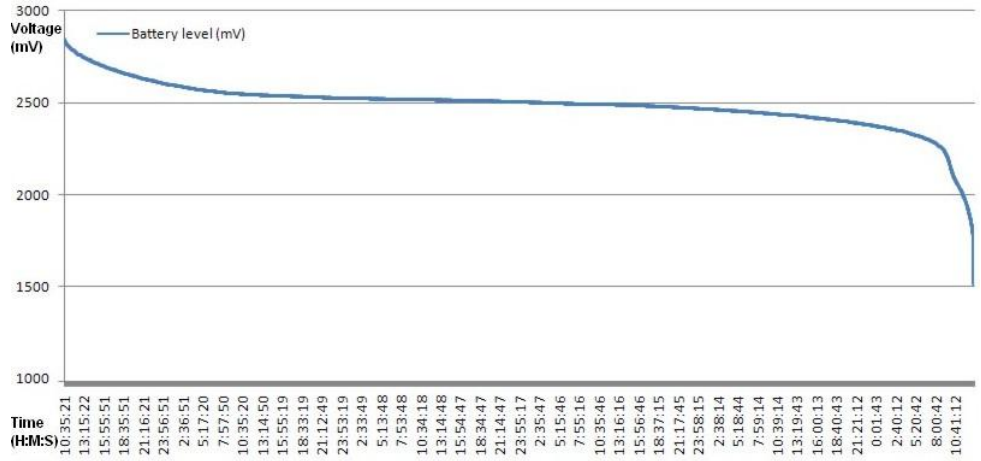


Figure 6.3: *Discharge curve of 2500mAh Ni-MH cells when used to power motes*

6.2 Duty cycle tests

6.2.1 Synchronization test

The performance of the duty cycle system could be monitored while developing the system, by measuring the voltage of motes with an oscilloscope. The voltage drops to a lower level, faster, at times of larger current draw. Figure 6.4 shows a section of time including two ON periods. The mote in the test was running a 4% duty cycle with a 2 second period. This percentage equates to the 80ms ON time, shown by the lower voltage levels in the graph. The synchronization of nodes could now be studied easily and adapted by using this visualization method. The base station (running off USB power) voltage dropped about 133mV when the system switched on, while the voltage of the battery powered nodes only dropped by about 11mV. This large drop could be because of resistance in the supply line over USB.

6.2.2 Discharge rates

Motes were now set up with different duty cycles to test the difference in drain on cells. The same Ni-MH cells were used for all the tests and care was taken to fully charge them in between runs. While quick charging, the cells can get very hot and they were trickle charged until cooled down, to ensure full capacity. The discharge rate improved as expected and can be seen in figure 6.5, when reducing the duty cycle percentage. See the next chapter on life expectancy of a system related to duty cycle. Due to the significant

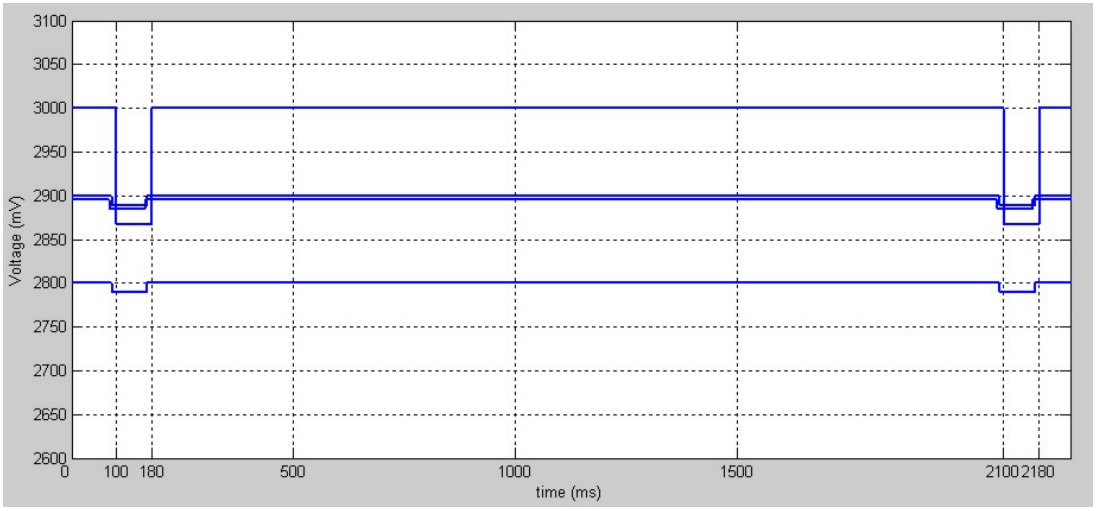


Figure 6.4: *Oscilloscope view of voltage difference while running duty cycle system*

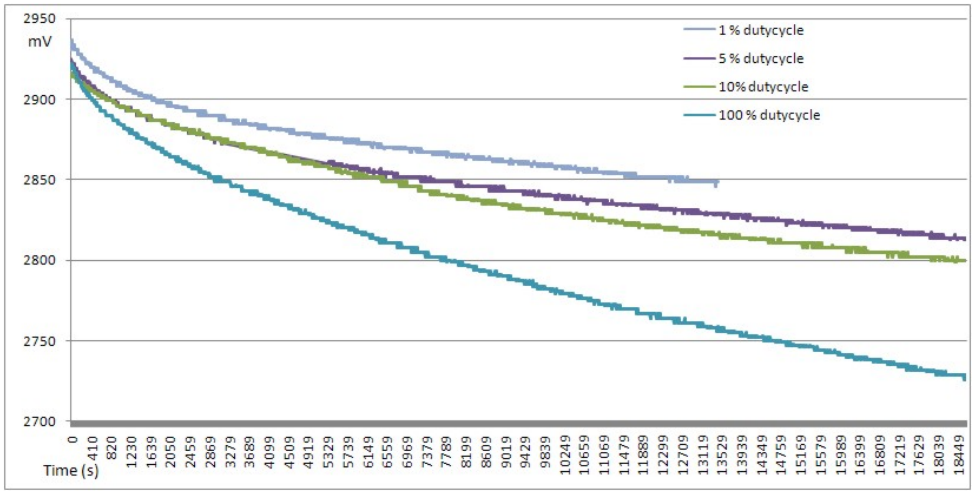


Figure 6.5: *Ni-MH discharge curves for different duty cycle settings*

lifetime extension it was not feasible to do lifetime tests with the system in this mode. For this reason only the basic trends were visualized and used as a lifetime estimate. The nodes reported data with a 100% success rate down to a duty cycle of 1 %. When setting this to 0.5% the nodes still synchronized, but on average about 50% of packets were dropped and nodes disappeared intermittently.

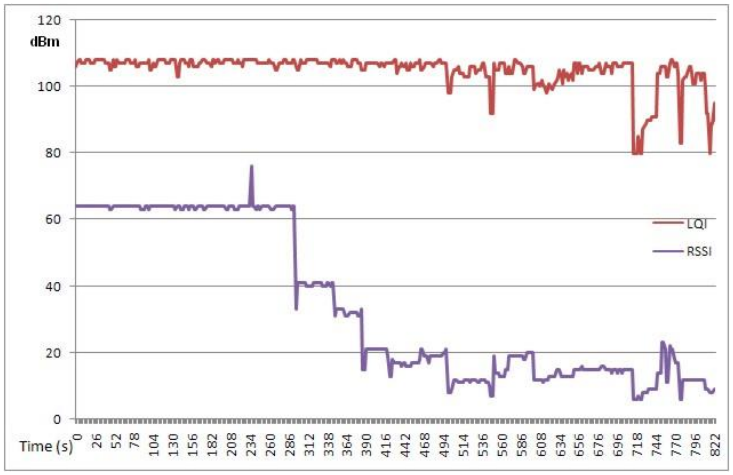


Figure 6.6: *RSSI vs LQI measurements for the same packets received*

6.3 RSSI and LQI radio link measurements

In this section, results are presented for tests done on the link estimators. These are the RSSI and LQI values calculated by the radio for each packet received (explanation given in Chapter 2). The basic multihop component, supplied with Boomerang, uses only LQI to estimate link qualities. To make range adjustments, and therefore power savings possible, RSSI values also had to be studied to determine its feasibility as a range estimator.

6.3.1 RSSI and LQI comparison

Through all the tests run, the RSSI and LQI measurements could be monitored and logged to compare them with external variables. To expedite variations in the radio link, the packet interval of the nodes was set to two seconds. The first test was done to compare LQI and RSSI over the range of reception. It can be seen in figure 6.6, that with RSSI levels[2] down to about 10, the LQI remains steady over the 100 mark. From here it dropped down quickly, to around 60 at an RSSI level of 4, but big variations were seen from packet to packet. The absolute limit of reception is at a value of -94dBm, which is 0 on our scale. Very few packets were ever received with a RSSI value lower than 4, and the minimum level measured was 2. This means that the LQI measurement can only really be relatively accurate in the final range of reception. This drastic fall in the link quality means that LQI can be compared very well to packet reception rate[32].

[2]This level minus 50 is the RSSI register value that can be converted to dBm using Figure 2.6, giving -84dBm for a value of 10

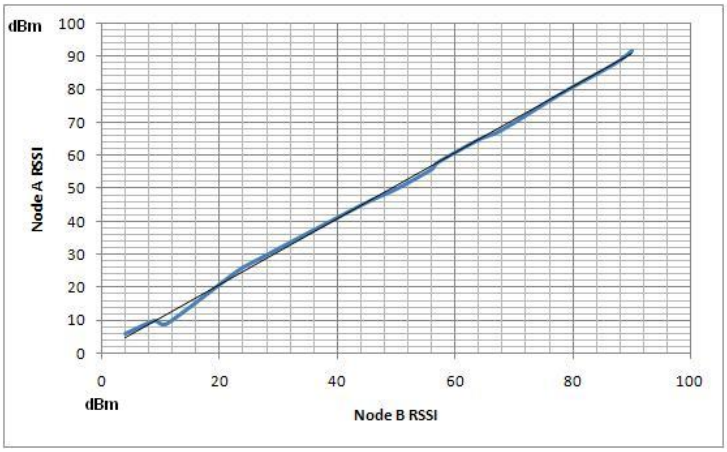


Figure 6.7: *RSSI correlation between 2 nodes*

6.3.2 RSSI accuracy

The accuracy of RSSI was measured to check stability and correlation between measurements of both links between two nodes. As seen in figure 6.7, the variation between measurements are small and always within about 4dBm of each other, throughout the whole reception range. This makes it possible to predict the signal strength of packets sent to other nodes without receiving this information from the particular targeted node, which assists with the accuracy of radio range adjustments.

6.3.3 Transmission limits

To safely adjust radio range, the limits of packet transmission have to be found at all power levels. This experiment was run with route packets sent at full power to measure the RSSI value. Data packets were sent at the seven different power levels available, to ascertain the RSSI level where perfect transmission is no longer possible. Down to this level RSSI on its own is a good measurement of link quality as LQI stays relatively the same, as shown in the previous section. Figure 6.8 shows that there is a good linear relationship between lowering power levels and received power, as is to be expected.

6.3.4 RSSI vs Distance

The stability and accuracy of RSSI values at different ranges enable the prediction of distance between nodes. In these tests, nodes were placed at measured distances from each other at a constant level above ground. A LOS path was always ensured as anything

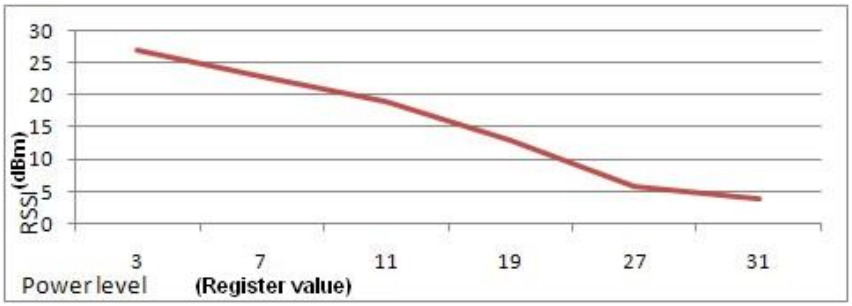


Figure 6.8: *RSSI limits of transmission at different power levels*

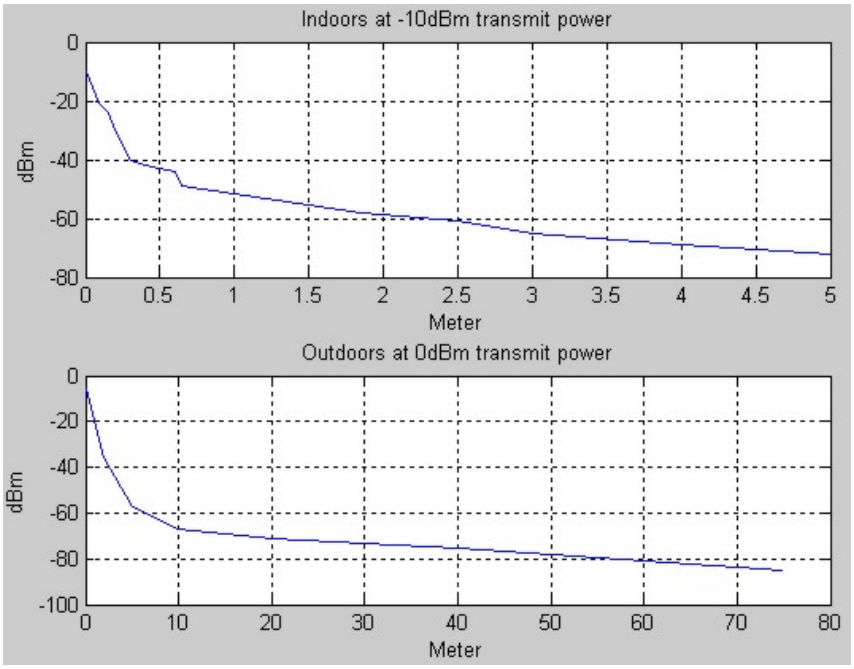


Figure 6.9: *RSSI correlation with distance*

in the way produces variable attenuation. The values of a number of equal length links were now averaged to get the RSSI correlation to distance. Figure 6.9 gives the RSSI values over distance for measurements made indoors. Nodes were transmitting at -10 dBm to limit range for easier indoor usage. Due to multipath issues indoors, very big variations were seen. Outdoors, the measurements were made at 0dBm. Readings are a lot more stable in an environment without too many buildings and too much other clutter, therefore, in outdoor applications where stable links are available, range can be much more accurately estimated from RSSI. The important point here is to recognize that the propagation range can be adjusted conveniently by observing the RSSI.

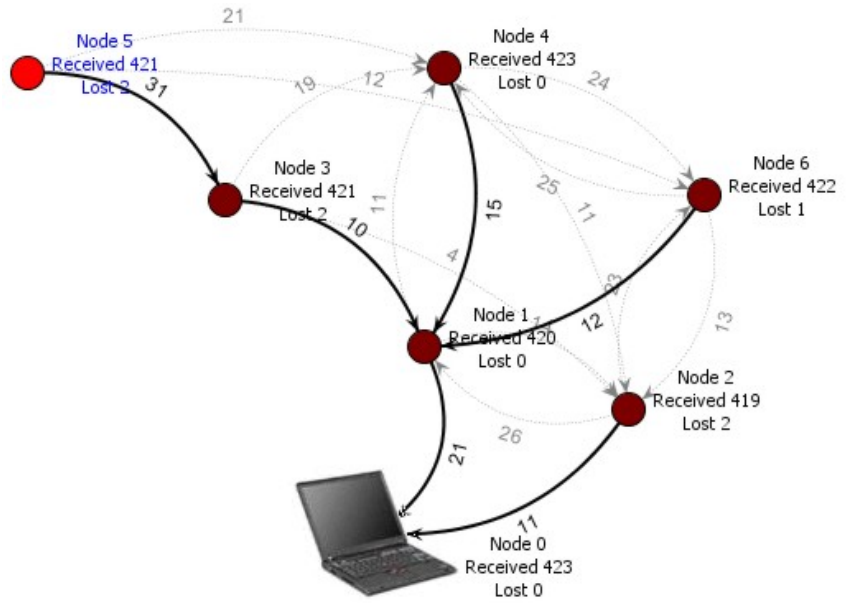


Figure 6.10: *Trawler display of constructed multihop network*

6.4 Network setup and communication

6.4.1 Route discovery

System setup time can be very long, depending on the interval between beacon messages. Figure 6.10 shows a network of 7 nodes with their current route paths in dark lines and connections to other neighbours in the lighter lines. In this example beacon message interval was set to 10 seconds. Nodes one and two will be the first to have valid routes, as soon as they receive a message from the base station. After 10 seconds, when they send their beacon messages, nodes three, four and six, will each receive valid routes from both these nodes. All three of these might set node 2 as their parent, if they receive its message first, but when node one sends a message they will make a decision as to which one is best. Only after another ten seconds will node five receive beacon messages from nodes three four and six with valid routes and be able to set a path. Each one of these could be set as its parent, depending on the first message received, but in the end the best path is through Node three and then one. The maximum time taken for nodes to go online, therefore, is the minimum number of hops to the base, multiplied by the beacon message interval. The message queue size can be set, to ensure packets are not dropped due to setup time in large networks. If a node is 5 hops away from base and beacon and data message interval is at 10 seconds, setting the data message queue to 5 will ensure that all messages are sent, even if route discovery takes the maximum amount of time.

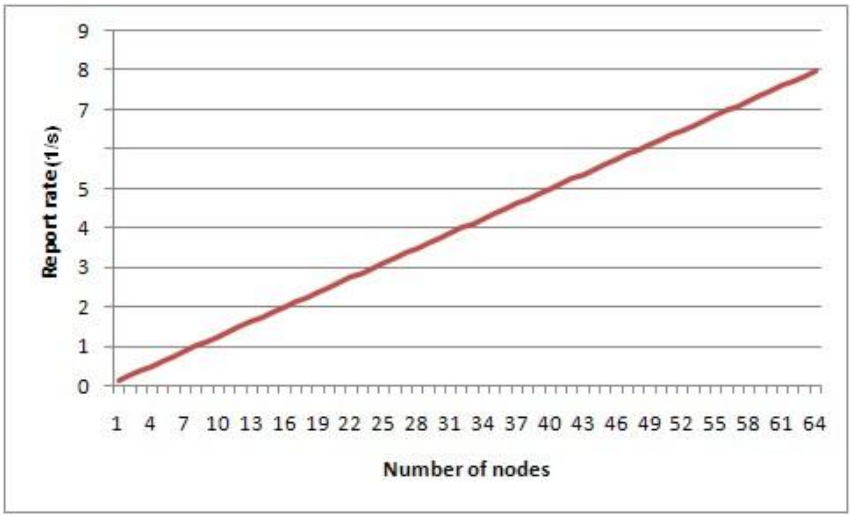


Figure 6.11: *Maximum report rate against number of nodes in the network*

6.4.2 Network throughput

No dutycycle

High throughput in a sensor network such as the present is not very critical, as environmental data does not change dramatically in short periods of time. Measurements were made to find the maximum report rate that the system can produce, as limits can be reached in large networks. The bottleneck is seen at the base station which has to receive data from all other nodes before reporting it over USB, to a PC.

Tests were run with nodes set up with a report rate of eight packets per second. This means that every second 448 bytes (See appendix C on packet data) of data packets is sent to the base station. This was the limit at which the base station could receive and send messages from one reporting node without loss. The problem is that with multiple nodes this rate will go down accordingly. Packet loss with 2 nodes was on average 50% and with three nodes 64%. The maximum report rate, therefore, decreases with the number of reporting nodes. This is shown in figure 6.11.

1% Dutycycle

When setting the system for 1% dutycycle operation, the rate at which data can be sent is dramatically reduced, as the motes are only operating for 20ms every 2 seconds. This means that the maximum report rate of nodes is only once every 2 seconds. With one node, packet transmission success at this rate is still 100% but with two nodes, on

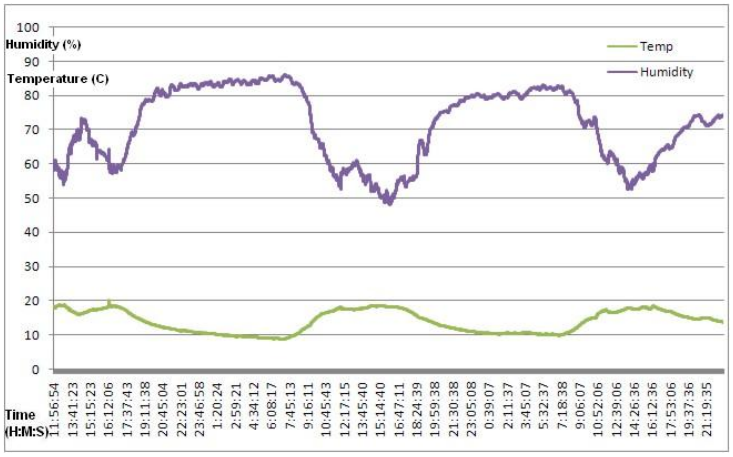


Figure 6.12: *Outdoor humidity and temperature measurements*

average about 50% of packets are dropped. When halving the rate to once per 4 seconds, packets can again be received perfectly. This means that the maximum rate of perfect transmission with this setup is the number of nodes multiplied by the 2 second period. If all nodes send at the same time, then packets will still be dropped, but this evens out until they send correctly.

6.5 Sensor readings

A couple of motes were deployed with sensors attached to obtain performance data over a couple of days. It was found that packet delivery was near perfect over the whole deployment time, with the only packet loss measured while moving the motes, or with a lot of activity around the base station.

6.5.1 Outdoor measurements

The temperature and humidity[3] in figure 6.12, were measured outdoors with a mote setup for 3% duty cycle operation. Note that the voltage level at the start of measurement in figure 6.13 increases for a while. This is because of the recovery effect of cells, after the higher load due to programming and reset. The mote, now running in low power mode, uses much less power. Another interesting thing to note is the very limited voltage drop during the day and then the much higher loss at night. Over the two and a half day test

[3]It was noted that when the humidity measurements reached over 80 at night it started raining

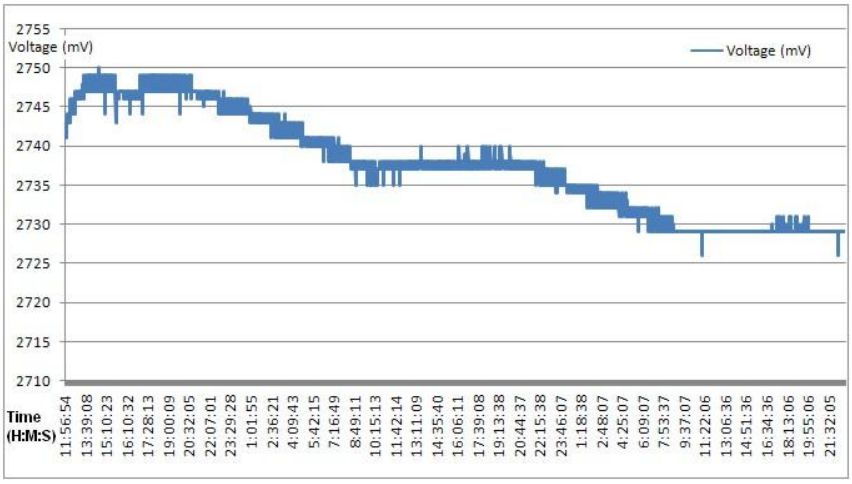


Figure 6.13: *Outdoor mote voltage*

period, the battery voltage dropped by about 17mV. This difference correlates well with the lower temperatures at night, during which the cells lose charge more quickly.

6.5.2 Indoor measurements

Indoor temperature and humidity (figure 6.14) varied much less than that outdoors, as can be expected. From Figure 6.15 it is clear that the indoor mote's battery voltage dropped by much less over the same period of time than that of the outdoor motes. Due to higher indoor temperatures a drop of only 7 mV was measured. The batteries of the two motes were matched to have about the same charge level at the start of test. The light intensity measurements shown in figure 6.16 are not measured according to a specific scale. This was calibrated only to show the intensity difference between night and day. A value of over 4000 is darkness and in the day, under lights, the value was around 200. Note the steady climb and fall at evening and morning, and the sharp step increase/decrease when a light was switched on or off.

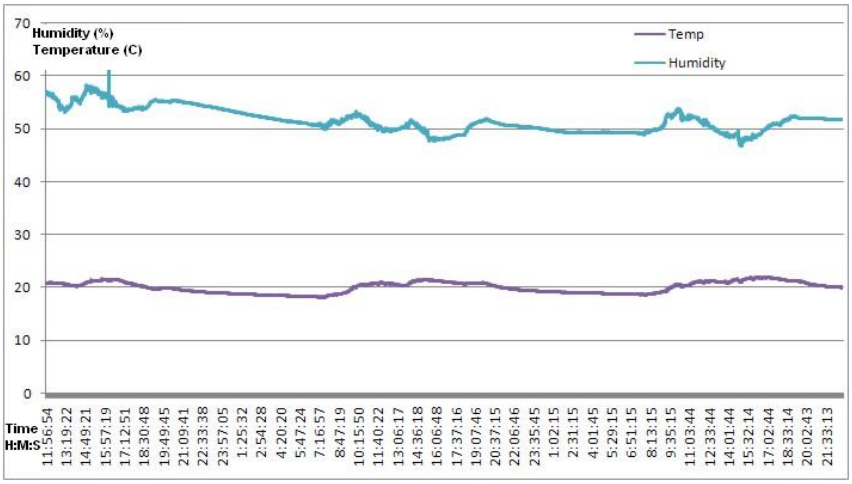


Figure 6.14: *In lab humidity and temperature measurements*

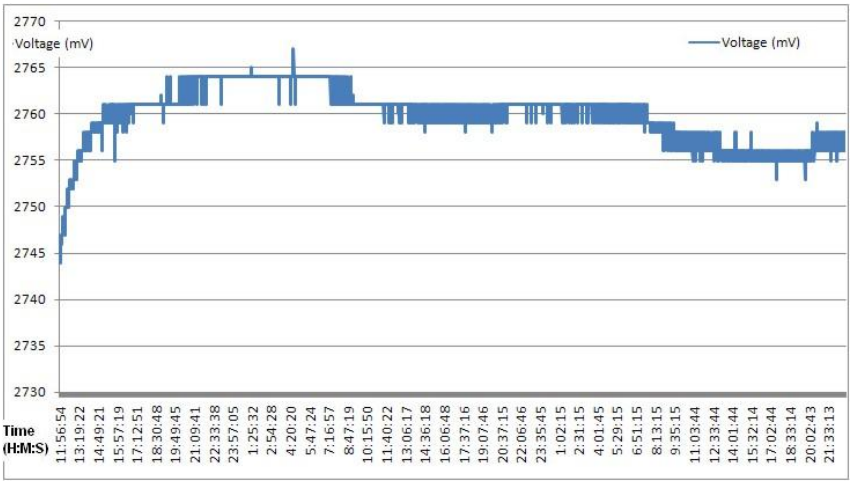


Figure 6.15: *Indoor mote voltage*

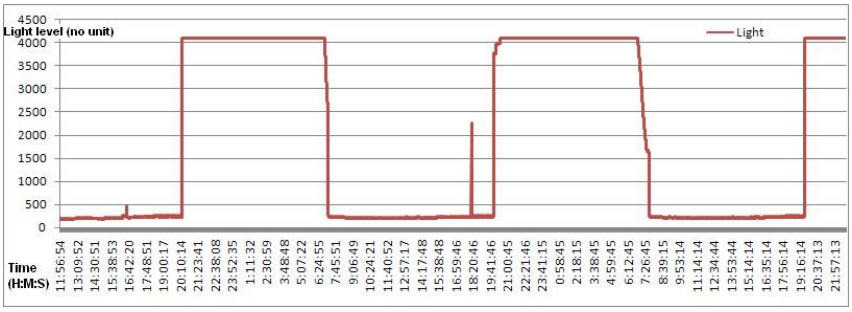


Figure 6.16: *Light intensity inside lab*

6.6 Summary

Three types of AA cells were used to run the motes in lifetime tests. This provided discharge curves for the cells and node lifetime for comparison with models in the next chapter. The suitability of the cell types for mote operation was examined by using their respective discharge curves. These can also be used to good effect, to visualize the difference in energy use with different duty cycle settings.

The RSSI and LQI link estimators, provided by the radio, were tested and compared. The RSSI data was found to be a very accurate measurement of received signal strength. This could be used to facilitate range adjustments by utilizing it as a signal strength, neighbour localization tool. The LQI measurements was found to be a good link quality estimator at the limits of packet reception, and can be used to ensure good routing links on top of the power saving features. The limits of successful data throughput were measured to ascertain the speed at which data can be logged. It was found that a much larger amount of data can be handled than is normally necessary for environmental monitoring applications. A deployed system gave steady results and good insight into environmental effects on the system.

Chapter 7

Comparative results

This chapter presents a comparison between real system test runs and system models. The models in question are the simulations described in Chapter 5 plus a mathematical model.

7.1 Battery life

The two formulas below can be used to model [10] the basic power usage of motes. The amount of energy used is E and the supply voltage is V. This $\frac{E}{V}$ is measured in mAh and, therefore, can be related to the rating of the batteries used. I_m and t_m are the μprocessor's current draw and the time it is switched on respectively. I_r and t_r are the current draw and time the radio is in receive mode. I_t and t_t are the radio current draw and time the radio is in full power transmit mode respectively.

Table 7.1 provides the power consumption for the various parts of the motes to be used in calculations. The summated power used by other components, eg LEDs and sensors, must also be added when power usage is calculated for a deployed system. I_{ci} and t_{ci} represent the current draw and ON time of these components. This will mostly be ignored, as power calculations are utilized to provide information on the effects of communication on energy usage.

$$\frac{E}{V} = I_m t_m + I_r t_r + I_t t_t \tag{7.1}$$

$$\frac{E}{V} = I_m t_m + I_r t_r + I_t t_t + \Sigma_i I_{ci} t_{ci} \tag{7.2}$$

Operation	Current draw
Mote Standby	5.1 μA
MCU active	1.8 mA
MCU + Radio RX	21.8 mA
MCU + Radio TX (0dBm)	19.5 mA
(-1dBm)	18.6 mA
(-3dBm)	17.3 mA
(-5dBm)	16 mA
(-7dBm)	14.6 mA
(-10dBm)	13.3 mA
(-15dBm)	12 mA
(-25dBm)	10.6 mA

Table 7.1: *Current draw of various operations by the Tmote*

7.1.1 Always ON system

The similarity between the lifetimes of rechargeable 2500mAh cells and alkalines was shown in the previous chapter. With these, a lifetime of around 125 hours was achieved with a system not running on a duty cycle. For our basic model, given in equation 7.1, we combined the processor current draw with that of the radio component as it will always be turned on. The usage calculations can now be made by the following derived equations:

$$mAs = I_{tr}t_{tr} \tag{7.3}$$

$$mAs = I_{tr}t_{tr} + I_{tt}(1 - t_{tr}) \tag{7.4}$$

$$mAs = I_{tt}t_{tt} \tag{7.5}$$

With equation 7.3, the power usage is calculated for a system always in receive mode and no transmissions made. Equation 7.4 takes receiving and occasional transmissions into account. Equation 7.5 expresses power usage for a continuously transmitting theoretical system. A lifetime of 114h 40m 55s is predicted for the first case with this model and a lifetime of 128h 12m 20s for a continuously transmitting node. The expected life of a node will, therefore, be somewhere between these values.

During the tests (Chapter 6), a maximum transmission rate of 3584bps was achieved by sending the standard 56 byte packets. This equates to a 10 minute battery lifetime extension when compared with our model as presented in figure 7.1. Our simulation model also compared well with these figures, as a 125 hour total lifetime was achieved, in

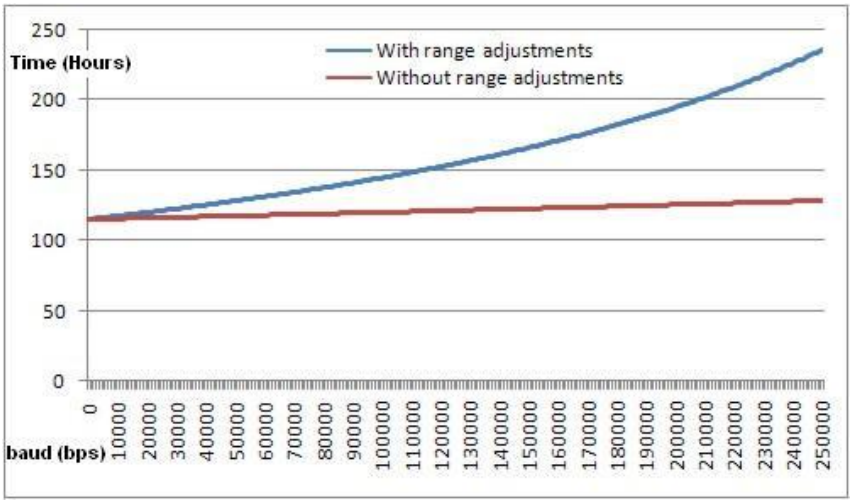


Figure 7.1: *Math model battery life comparison*

keeping with the tests[1]. The fact that our mathematical model provides a slightly lower lifetime when compared with the other methods of testing can be explained by the fact that the current draw stays the same while the voltage drops. This may cause a mismatch between the *Ws* and *mAs* rating of batteries.

Including radio range adjustments

To obtain the effects of radio range adjustments on energy consumption, the model can be extended. Equation 7.6 below, shows how consumption at the seven different transmission power levels is brought into the calculations.

$$\frac{E}{V} = I_m t_m + I_r t_r + \Sigma_7 I_{txi} t_{txi} \tag{7.6}$$

Lifetime can be extended to 235h 50m 56s if a system constantly transmits at the lowest power level (-25dBm). With the 3584bps throughput attained in the test system, these adjustments can equate to a one hour battery life extension, when running on 2500mAh cells.

[1]This can be seen in Figure 7.2 at the 100% dutycycle point. Simulation outputs are not always shown as this would have taken an impractical amount of time.

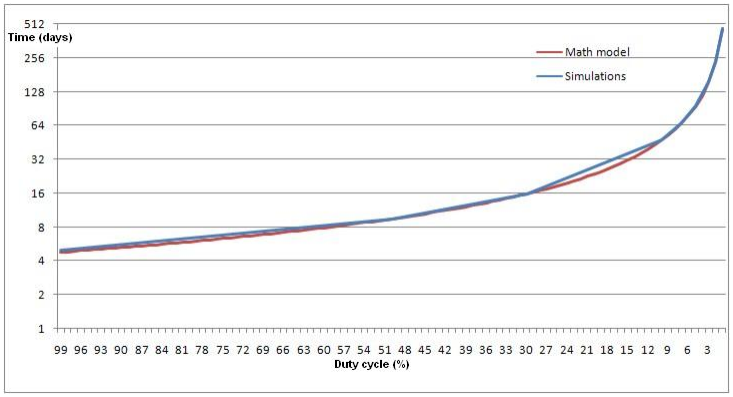


Figure 7.2: *Lifetime of nodes at different duty cycle settings.*

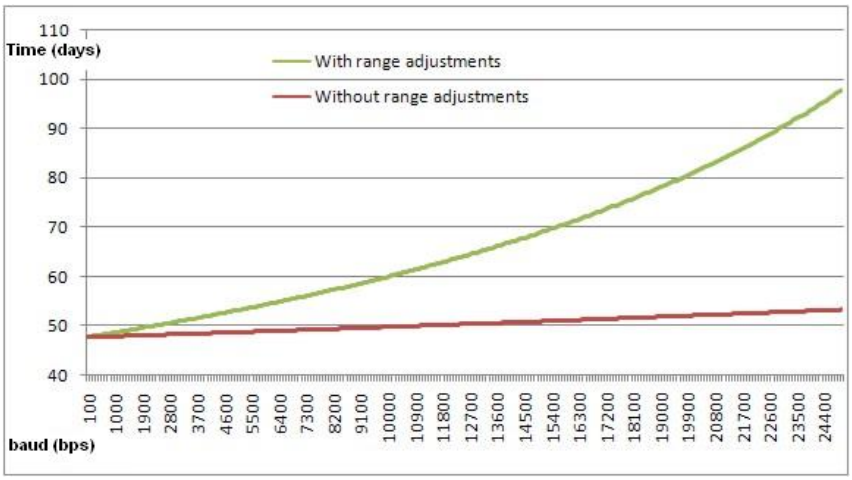


Figure 7.3: *Math model node lifetime predictions at 10% duty cycle*

7.1.2 Duty cycling system

The duty cycle system is the main power saving feature, as a mote in standby only draws about 5.1 μA of current. The math model below is extended by including the current draw while the mote is in standby mode, $I_l t_l$. Figure 7.2 shows the dramatic improvements in lifetime when the system duty cycle is reduced. From just over 5 days lifetime is increased to 467 days at a 1% duty cycle.

$$\frac{E}{V} = I_m t_m + I_l t_l + I_t t_t + I_r t_r \tag{7.7}$$

The lifetime of a mote transmitting 448 bits every 2 seconds, can be increased to about 471 days. The range in lifetime due to changes in bit rate is shown in figures 7.3 to 7.5, for duty cycles of 10 %, 5% and 1% respectively. For each one, again, this lifetime was calculated over the whole theoretically possible range of bitrates.

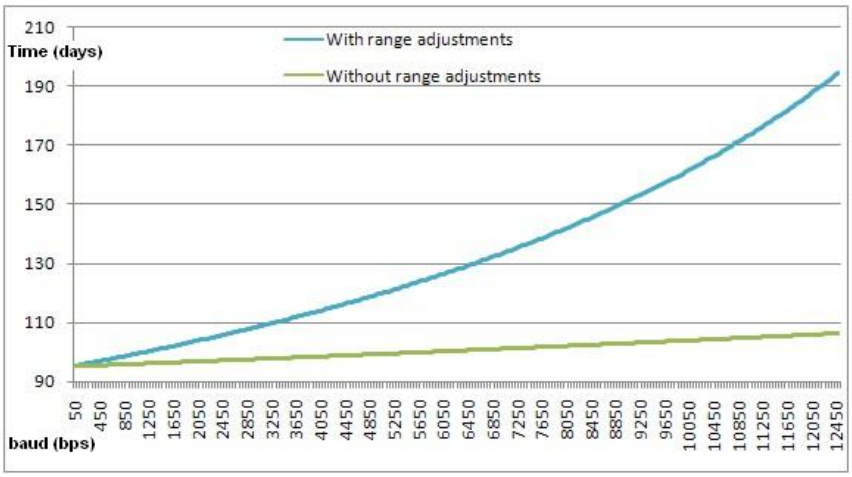


Figure 7.4: *Math model node lifetime predictions at 5% duty cycle*

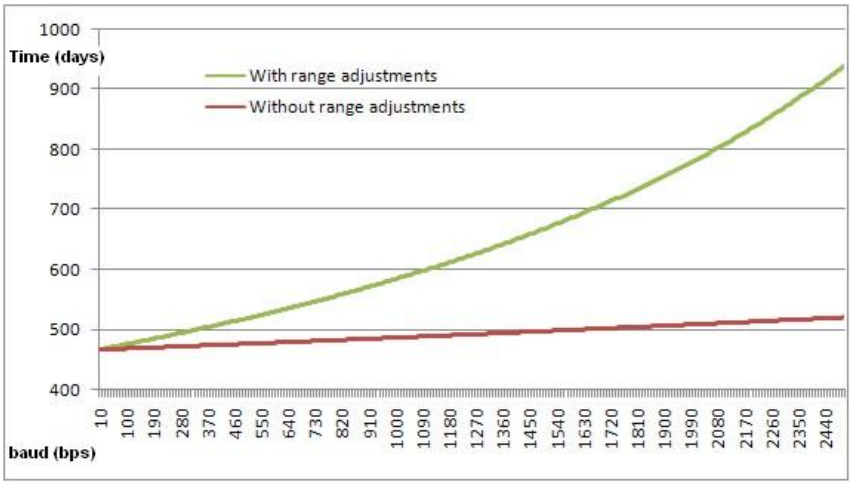


Figure 7.5: *Math model node lifetime predictions at 1% duty cycle*

Including radio range adjustments

To include the effect of radio range adjustments, the model was once again extended to include the term for the different power levels. With the test bit rate previously used, equating to a throughput of 224bps, battery life is extended from 467 to 488 days, when minimum transmission power is used. Tests on packet sizes of up to 80 bytes (320 bps) still produced perfect packet transmission. This equates to a lifetime of 498 days. So a lifetime extension of a full month is available over a system without range adjustments.

$$\frac{E}{V} = I_m t_m + I_l t_l + \Sigma_7 I_{txi} t_{txi} + I_r t_r \tag{7.8}$$

Figures 7.3 to 7.5 clearly indicate the possibility of lifetime extension by radio range adjustments if the system bit rate can be further improved.

7.2 Voronoi routing analysis

Routing data through neighbours with the best link quality provides better connectivity. As described before, the Voronoi diagram can be used to find the best connected path between two points (maximum support path). This means that a path between nodes and the base station can be found that has the best possible link quality between hops. Using the Voronoi diagram it will now be proven that the best quality path (shortest transmission links) in a sensor network is not always the best one.

In figure 7.6, one hundred nodes are placed randomly in a 400x400 grid. Let us assume the distance between nodes is the "radio" distance (90 corresponding to -90dBm which is about the limit of reception). The routes of three nodes to the base station are shown with the red lines indicating the best connected path (best support path) and the green lines the shortest distance. The best connected path uses links which are all shorter than the radio range at minimum transmit power.

It was found that on average the routes utilizing the best support path are 25% longer. Another point to note is that nodes will have to route an average of 109% more packets (correlating to 109% more hops) when using the best support path. In a 100 node network the average maximum of nodes which will route data through a certain node is 50, but by using the shortest path this drops to below 20. This may become a big problem with the dutycycle system. As tested earlier, with low dutycycle settings, nodes do not have

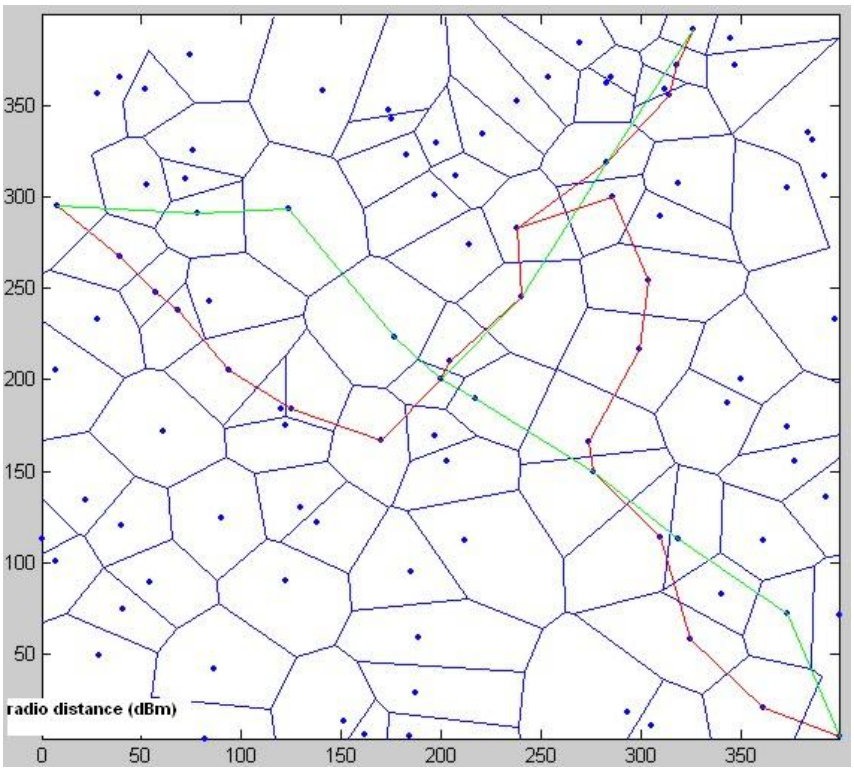


Figure 7.6: *Difference between shortest path and closest neighbour routing*

a large communication throughput. Routing more data packets, therefore, means larger data loss and ineffective routing, especially in larger networks.

The maximum support path computed by using the Voronoi diagram can, therefore, be used to get the best quality links with the shortest hops possible, but will have adverse effects on throughput. What also might happen is that transmissions continue beyond the dutycycle ON time, which will cause higher energy consumption; synchronization might need to be redone because of packet loss. It can be argued that transmitting at lower power levels saves power, but to get packets through effectively, a higher duty cycle will be required in large networks, meaning that this won't make a positive difference.

7.3 Summary

A mathematical model was constructed for different types of system operation. These include a system running with and without a duty cycle, and with the radio range adjustments either on or off. It was also demonstrated that battery lifetime was very accurately modeled by the simulation and mathematical model. The lifetime of nodes can be very long, therefore, it was not feasible to do full lifetime runs. For this reason, the simulation and mathematical models can be very good tools in estimating system lifetime. The life extension achieved by using low duty cycle operation and radio range adjustments can be very large, but it depends on the amount of data transmitted by a node. Voronoi diagrams were used to research the best ways to route data in a sensor network and it was found that just using the shortest (the best quality) connections are not the best option.

Chapter 8

Conclusions and future developments

WSNs could be used in a very large range variety of applications. These range from military threat detection to early warning systems. This led to the idea of designing a WSN for environmental monitoring applications, which could be used in crop micro management and forestry research, among others.

The differences between WSNs and conventional wireless networks are mostly based on hardware limitations of WSNs. These include the limited computational power and storage space of processors, as well as the limited radio ranges. This is due mostly to power constraints and the need for hardware cost reduction. Other differences are in the WSN control software, where issues like coverage, connectivity, localization and routing need to be addressed in novel ways to ensure energy efficient and stable system performance over long periods of time.

The following sections provide a summary of the major work that has been addressed and contributed to the field in this thesis and also some future developments in the WSN field.

8.1 Summary of contributions

This thesis set out to improve and extend aspects of existing WSN development, simulation and implementation software tools, as well as to create some additional ones. This serves to increase the overall predictability and functionality of the WSN application. In this, it is felt that some success has been achieved. The main areas of focus, can be summarized as follows:

Low power hardware platform. A comparison of available hardware platforms led to the decision to use using the Tmote Sky platform. These motes were found to have very low energy consumption characteristics as well as good support for sensor add-on and the ability to be controlled by TinyOS. One of the main research areas [8] still open, is in the reduction of mote power consumption, while ensuring good communication and overall system performance. This was achieved by writing and adapting, as well as using currently available, TinyOS components.

Smart sensing application. The Sense application was developed with network programming ability provided by Deluge, and a lockup safeguard feature in the form of a watchdog timer. This program also automatically detects sensors and reports data through the multihop system. All these features ensure a stable and robust logging system.

Power usage reduction by use of smart communication strategy The routing system was made power aware. By using the multihop routing layer that was developed, in conjunction with the duty cycling system, a big reduction in power consumption could be achieved. It utilizes battery level information to estimate lifetime and signal strength information for virtual localization, to make range adjustments possible. The aforementioned information was used in conjunction with a link quality indicator as a route cost setup metric. This provided very stable routes, with nodes limiting power use where possible. The in depth study of the radio link indicators made it possible to predict what could be expected in different scenarios and they were, therefore, incorporated with great success. With all the incorporated features, a stable and deployable system could be developed with a lifetime extension from 5 to about 490 days. It provided more than adequate performance for environmental monitoring systems.

Voronoi routing analysis. The Voronoi diagram is a useful geometric shape and was used to show the problems that might arise in large networks when using the paths with the best connectivity, in contrast to the shortest paths. Network throughput will be adversely affected. Especially with a power saving, duty cycling system, using the wrong

routing strategy will produce and ineffective or faulty system.

WSN energy use simulations. Comprehensive tests to obtain the lifetimes at different settings are not feasible on a real system, as it would be too time consuming. For this reason, a simulation model was developed to simulate power usage while accurately modeling network communication. Modeling of node power consumption in sensor networks was not an easily achievable task, but the developed model makes it possible to successfully simulate this for a variety of scenarios. The simulations, together with a basic math model for comparison, produced results that were very close to actual hardware measurements and which can, therefore, be used for accurate lifetime estimations.

AA cell comparison. Tests were only carried out over a short period, but with a range of batteries to compare performance and capacity. It was found that alkaline cells would be the best to use in deployed systems. They have a steady discharge curve and do not lose capacity after long periods of time, while still producing similar power delivery to 2500mAh Ni-MH cells designed for high current applications.

User and developer tool creation. Software tools are needed to enable easy implementation and design of future WSNs. Display of all data throughout system development was made possible by creating an extended version of the Boomerang Trawler application. It can visualize all sensor channels, as well as link cost information and network topology. The CSV data logger can be deployed as a system for data logging and display through a web page. This paved the way for future developments in online data logging.

Open source program use. The system was designed using open source software where possible. TinyOS and the Boomerang distribution, including all parts thereof is open source software. The Truetime simulation package is also available as a freeware MATLAB add-on.

An Ubuntu Linux distribution with built in TinyOS, called UbunTOS is available. This distribution also has the ability to be installed on a 1GB flash drive, providing a fully open source and compact WSN development system. This enables the development of a purpose built and compact computer that can function as the base station.

Figure 8.1: *Golem dust mote with American penny for size comparison*

8.2 Hardware of the future

8.2.1 Motes

Hardware advances in the recent past have led to the development of cheap and readily available WSN motes. Reduction in manufacturing cost, size and power usage, are the key features addressed in hardware design, and great strides have been taken recently. Motes cores with a size of a couple of square centimeters(eg. Tmote mini, figure 2.8)have become available. The advances in the area of MEMS hardware are very advantageous to sensor networks, as transducers and communication hardware can be successfully manufactured in miniature sizes.

An example of such miniaturized motes, is the Golem Dust mote (Figure 8.1) from Berkeley. These motes are solar powered with bi-directional communications and sensing (acceleration and ambient light) within a total displaced volume of 4.8 mm^3

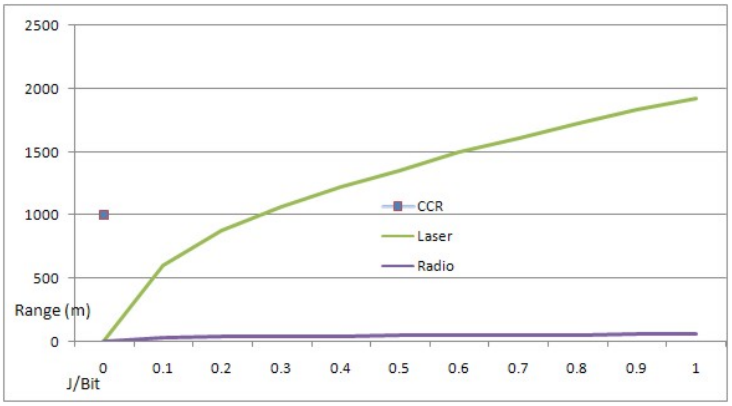


Figure 8.2: *Energy used per bit transmission for different methods*

8.2.2 Communication

A lot of work has been done on communications [42] to limit power consumption, per bit sent. Figure 8.2 presents a comparison of the energy required to transmit data via RF, laser and a corner cube retroreflector (CCR). Transmission by light uses far less power than radio signals and messages can therefore, be sent over much larger distances without using massive amounts of power. The passive CCR reflective system (used in miniature hardware such as Berkeley Dust motes), bounces light back to a transmitter in modulated format by actuating the mirrors. This uses about 16pJ/b for up to a range of 1km, as no radiation generation is necessary. The problem with light transmission schemes is, that a LOS path between nodes is essential.

8.2.3 Power supply

Batteries are still most commonly used as a power supply for sensor networks and are the cheapest power solution. Solar panels and energy harvesters are becoming a more popular choice, especially in miniature systems like dust motes etc. where the required energy is very low and the size of normal batteries becomes a problem.

Other alternative methods of supply are wind power and even fuel cells. These cells have been developed in the recent past and used successfully in small devices like cell phones etc. Small wind power generators can also be used, in conjunction with solar panels if required. With smart managing, motes can now be almost indefinitely powered with these types of renewable energy.

8.2.4 Pervasive computing

Pervasive or ubiquitous computing is the next step in human-computer interaction in which computing power are available in everyday devices. This differs from classic sensor networks in the sense that pervasive computers are more intelligent and not only perform sensing tasks, but can interact with and control one's surroundings. Think, for example, of a coaster that detects when your coffee cup is empty and then tells the coffee machine to start up, which in turn lets you know when it is done by communicating with a pervasive computer in your glasses. This might seem like a very unnecessary application, but the whole idea behind this, is that anything is possible. A step to this type of system has been taken by Mote*iv*, which has changed its name to Sentilla and is now developing its hardware as pervasive computers, using a new Java platform, promising even easier system development.

8.3 System Deployment

The system described in this thesis is ready for deployment. Some things that might have to be addressed are the following:

- Weatherproof enclosures
- Safe and sturdy placement
- Base station placement
- Sensor connections

Sensors need to be connected in such a way that they can take accurate measurements, but without hindering placement or enclosure of the motes. The base station will need additional hardware to facilitate logging and monitoring. This should be kept in mind in the supply power and safety of the hardware. Additions like an external antenna, which would produce a better radio range can be made, if this is necessary.

8.3.1 Off site monitoring

Using the serial forwarder application and Trawler, it is already possible to monitor data over a computer network as if one is sitting at the base station. The issue is that in most applications a standard computer network is not available at the base station. A connection like a satellite, a cell phone network or WiMAX can be used to connect the base station to the internet. A web server with any desired functionality can now be implemented to enable accessing of the sensor network data from anywhere in the world.

Bibliography

[1] "Omnet++ discreet event simulation system." http://www.omnetpp.org, 2007.

[2] AKYILDIZ, I. F., SU, W., SANKARASUBRAMANIAM, Y., and CAYIRCI, E., "A Survey on Sensor Networks." *IEEE Communications Magazine*, August 2002.

[3] ARAUJO, F. and RODRIGUES, L., "Survey on Position-Based Routing." 2006.

[4] ARMS, S., TOWNSEND, C., CHURCHILL, D., GALBREATH, J., and MUNDELL, S., "Power Management for Energy Harvesting Wireless Sensors." MicroStrain, Inc., SPIE Intl Symposium on Smart Structures & Smart Materials, March 2005.

[5] AURENHAMMER, F., "Voronoi Diagrams A Survey of a Fundamental Geometric Data Structure." *ACM Computing Surveys*, September 1991, Vol. 23, No. 3.

[6] BOSE, P. and MORIN, P., "ONLINE ROUTING IN TRIANGULATIONS." tech. rep., School of Computer Science, Carleton University.

[7] CARUSO, A., CHESSA, S., DE, S., and URPI, A., "GPS Free Coordinate Assignment and Routing in Wireless Sensor Networks." tech. rep.

[8] CHOU, P. H., "Challenges on Low-Power Platform Design for Real-World Wireless Sensing Applications." 2006.

[9] CONG, L. and ZHUANG, W., "Non-Line-of-Sight Error Mitigation in Mobile Location." 2004.

[10] DUNKELS, A., OSTERLIND, F., TSIFTES, N., and HE, Z., "Software-based On-line Energy Estimation for Sensor Nodes." Swedish Institute of Computer Science, EmNets 07, June 25-26.

[11] ESTRIN, D., BULUSU, N., and HEIDEMANN, J., "Gps-less low cost outdoor localization for very small devices."

[12] ESTRIN, D., BULUSU, N., and HEIDEMANN, J., "Adaptive Beacon Placement." in *21st International Conference on Distributed Computing Systems*, April 2001.

[13] GAO, Q., BLOW, K., HOLDING, D., MARSHALL, I., and PENG, X., "Radio range adjustment for energy efficient wireless sensor networks." *Ad Hoc Networks*, 2004, Vol. 4.

[14] GOBER, P., ZIVIANI, A., TODOROVA, P., DE AMORIM, M. D., HUNERBERG, P., and FDIDA, S., "Topology Control and Localization in Wireless Ad Hoc and Sensor Networks." *Ad Hoc and Sensor Wireless Networks*, 2005.

[15] HANSEN, M. S. and STA, S., "Practical Evaluation of IEEE 802.15.4/ZigBee Medical Sensor Networks." Master's thesis, Norwegian University of Science and Technology, 2006.

[16] HU, L. and EVANS, D., "Localization for Mobile Sensor Networks." Department of Computer Science University of Virginia, MobiCom'04, September 2004.

[17] IEEE. *802.15.4 Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, 2003.

[18] IYENGAR, R. and SIKDAR, B., "Scalable and Distributed GPS free Positioning for Sensor Networks." 2003.

[19] JACO LEUSCHNER, C., "The design of a simple energy efficient routing protocol to improve sensor network lifetime." Master's thesis, University of Pretoria, 2005.

[20] KINNUNEN, K., "Summary of efficient algorithms for maximum lifetime data gathering and aggregation in wireless sensor networks." tech. rep., 2005.

[21] KIVILUOTO, L., "Summary on Improving Wireless Sensor Network Lifetime through Power Aware Organization by Mihaela Cardei and Ding-Zhu Du." tech. rep., 2005.

[22] LEVIS, P. and LEE, N., *TOSSIM: A Simulator for TinyOS Networks*, September 2003.

[23] LOUREIRO, A. A. F., RUIZ, L. B., and NOGUEIRA, J. M. S., "Management of wireless sensor networks." Tutorial at IEEE/IFIP IM, May 2005.

[24] MEGUERDICHIAN, S., KOUSHANFAR, F., POTKONJAK, M., and SRIVASTAVA, M. B., "Coverage Problems in Wireless Ad-hoc Sensor Networks." tech. rep., Computer Science Department and Electrical Engineering Department, University of California, Los Angeles.

[25] MOSES, R. L., KRISHNAMURTHY, D., and PATTERSON, R., "A Self-Localization Method for Wireless Sensor Networks." tech. rep., Department of Electrical Engineering, The Ohio State University, 2001.

[26] MOTEIV CORPORATION. *Ultra low power IEEE 802.15.4 compliant wireless sensor module*.

[27] MOUSTAFA, H. and LABIOD, H., *Adaptive Path Energy Conserving Routing in MANETs*. Old City Publishing, Inc., 2005.

[28] NAGPAL, R., SHROBE, H., , and BACHRACH, J., "Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network." tech. rep., Artificial Intelligence LaboratoryMassachusetts Institute of Technology, 2003.

[29] OHLIN, M., HENRIKSSON, D., and CERVIN, A., *TRUETIME 1.5Reference Manual*. Department of Automatic Control Lund University, January 2007.

[30] POLASTRE, J., *A Unifying Link Abstraction for Wireless Sensor Networks*. PhD thesis, UNIVERSITY OF CALIFORNIA, BERKELEY, 2005.

[31] POLASTRE, J., HUI, J., LEVIS, P., ZHAO, J., CULLER, D., SHENKER, S., and STOICA, I., *A Unifying Link Abstraction for Wireless Sensor Networks*. SenSys05, 2005.

[32] POLASTRE, J., SZEWCZYK, R., and CULLER, D., "Telos: Enabling Ultra-Low Power Wireless Research." tech. rep., University of California, Berkeley, 2005.

[33] POZAR, D. M., *MIcrowave and RF designof wireless systems*. Wiley, 2001.

[34] PRODUCTS FROM TEXAS INSTRUMENTS, C., *2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*.

[35] SAVARESE, C., RABAEY, J., and LANGENDOEN, K., "Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks."

[36] SHANG, Y., RUML, W., and ZHANG, Y., "Localization from Mere Connectivity."

[37] SRINIVASAN, K., DUTTA, P., TAVAKOLI, A., and LEVIS, P., "Understanding the Causes of Packet Delivery Success and Failure in Dense Wireless Sensor Networks." tech. rep., Department of Electrical Engineering, Stanford University Computer Science Division, UC Berkeley, 2006.

[38] SRINIVASAN, K. and LEVIS, P., "RSSI is Under Appreciated."

[39] STOJMENOVIC, I., "Localized Network Layer Protocols in Wireless Sensor Networks Based on Optimizing Cost over Progress Ratio." in *IEEE Networks*, IEEE, January 2006.

[40] TAN, H., "Maximizing Network Lifetime in Energy-constrained Wireless Sensor Network." Australian National University, Faculty of Engineering and Information Technology, IWCMC'06, July 2006.

[41] VARIOUS, *Handbook of sensor networks, algorithms and architecture*. Wiley series on Distributed Computing. Wiley, 2005.

[42] VARIOUS, *Smart dust: sensor network applications, architecture, and design*. Taylor and Francis, 2005.

[43] ZOU, L., LU, M., and XIONG, Z., "Pager-m: A novel location-based routing protocol for mobile sensor networks.".

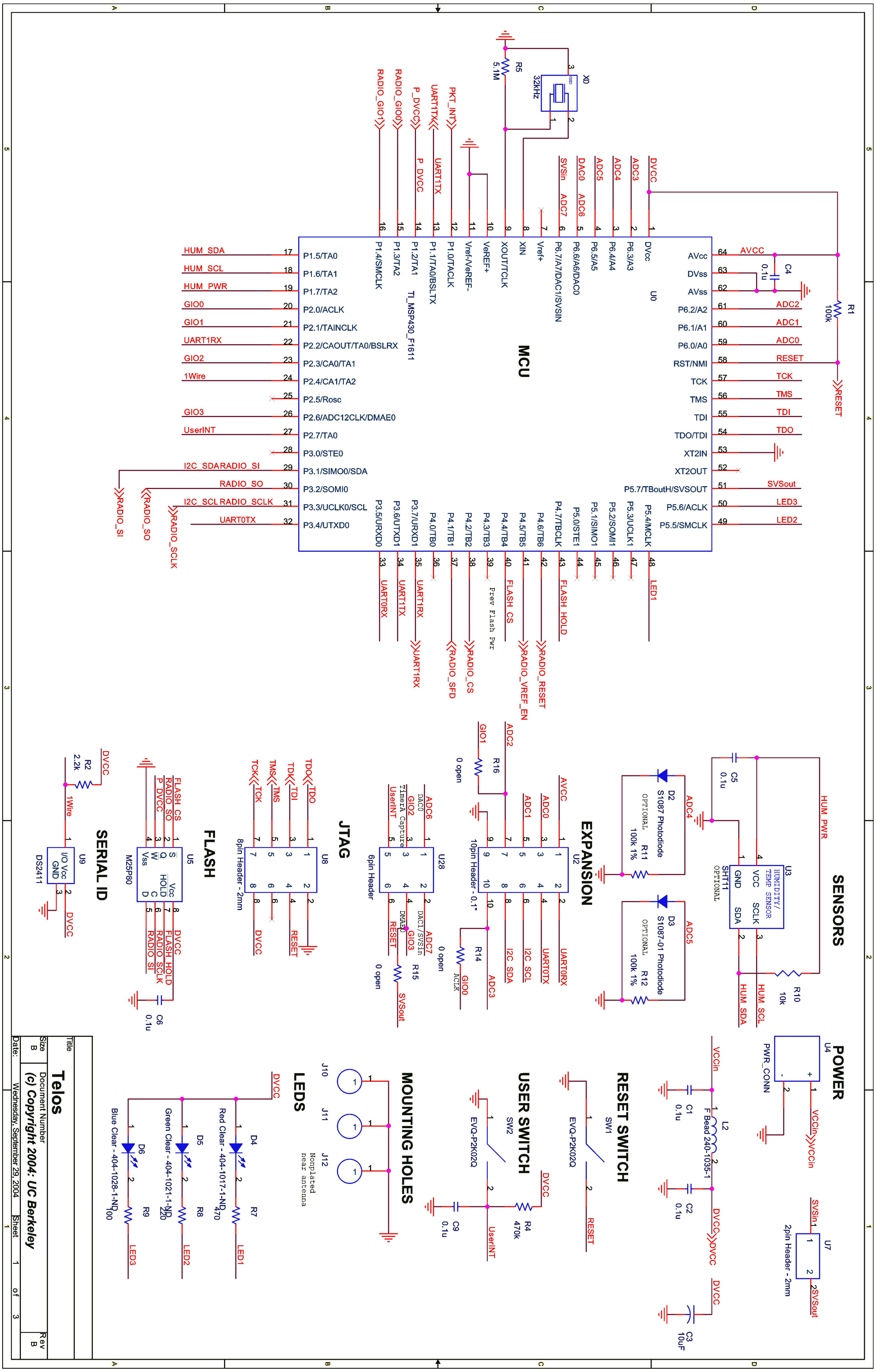
Appendix A

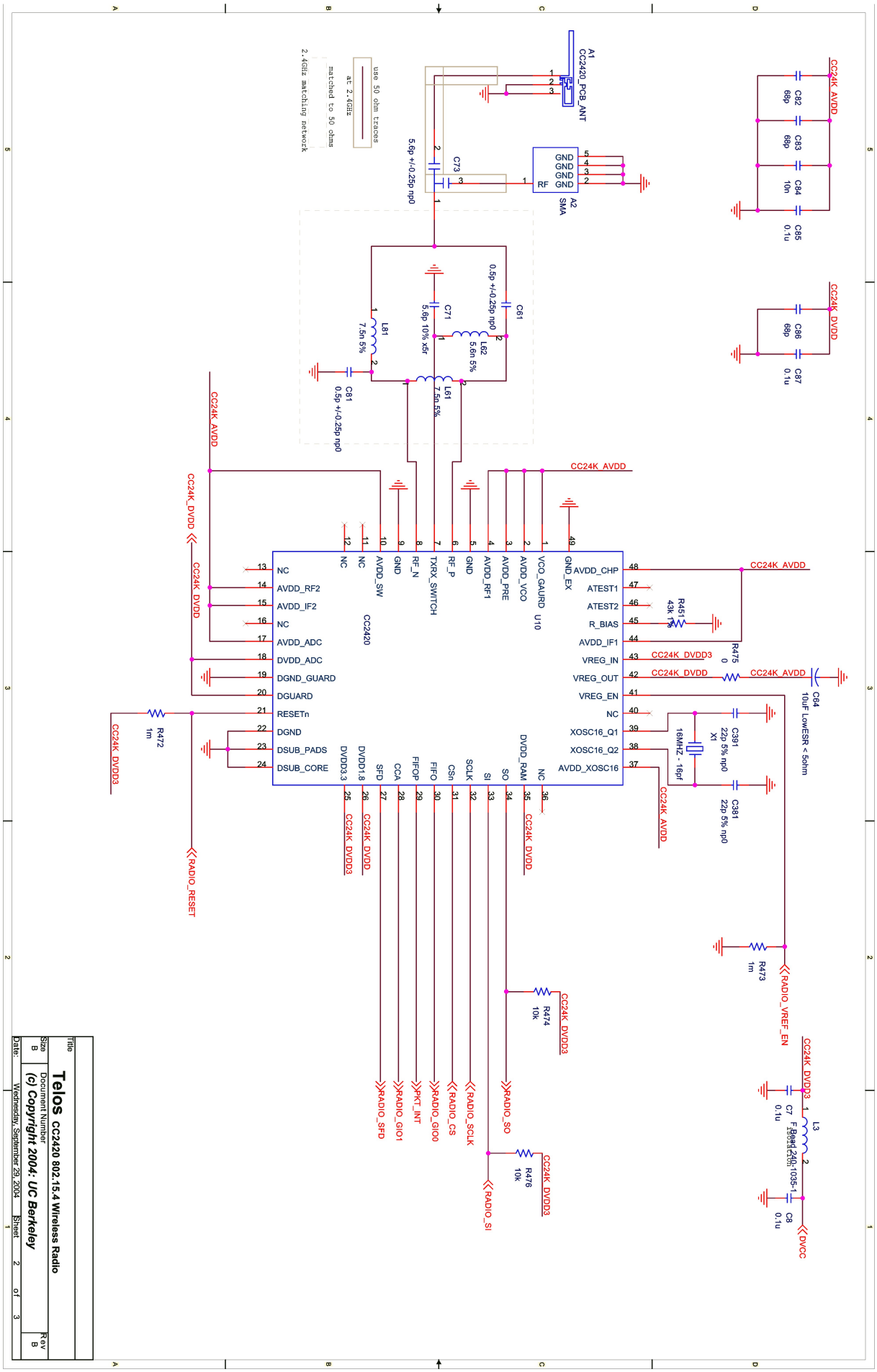
Tmote Schematics

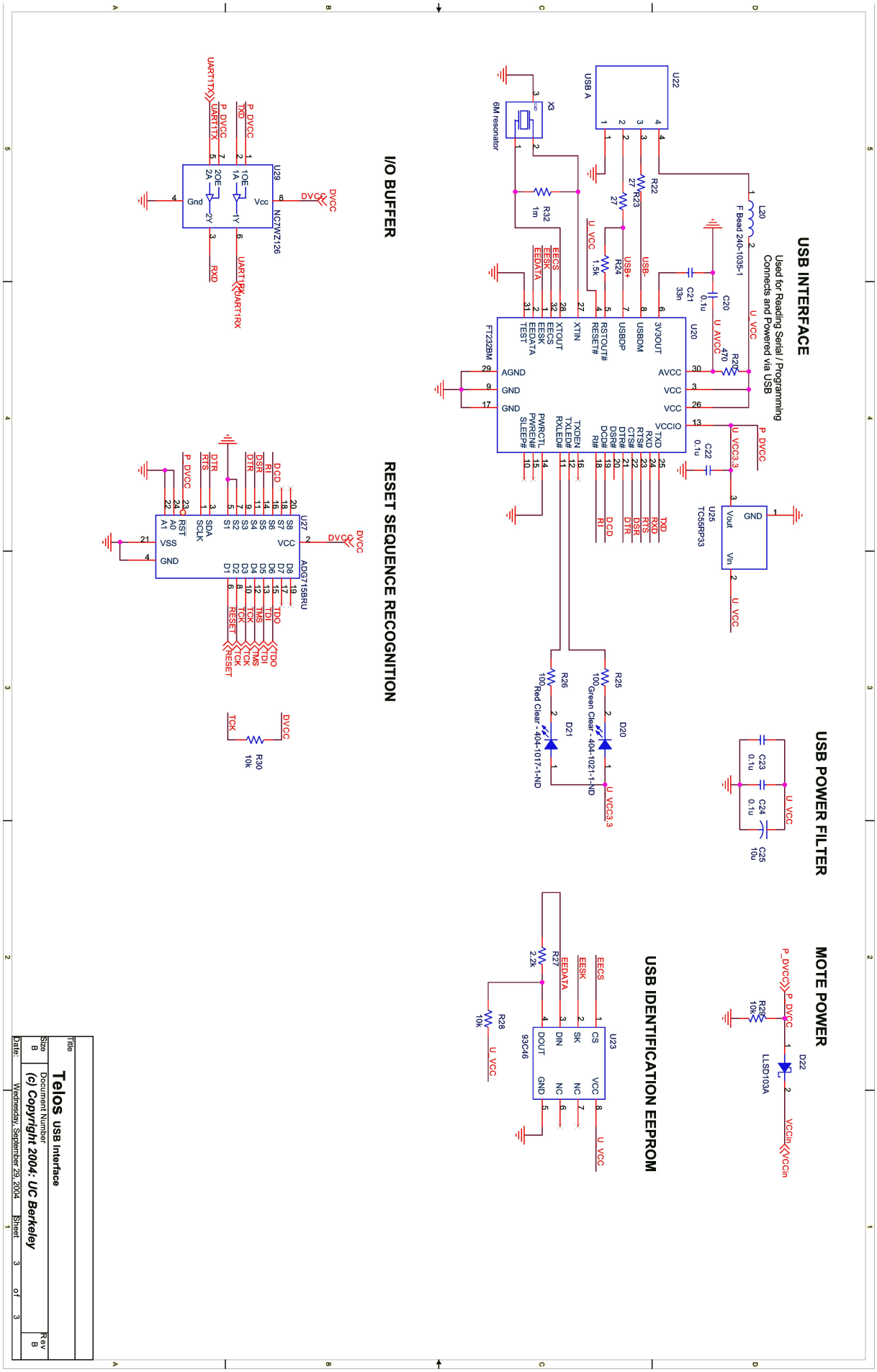
This section includes the schematics of all the hardware on the Telos module. In the first schematic, the MSP430 processor is described, with connections to other peripheral devices, like the sensors, flash memory, expansion connectors and LEDs.

The second schematic shows the CC2420 radio with necessary components and connections. Note the 2.4 GHz antenna matching network, with matched traces to the onboard antenna and to the SMA connector for external antenna. The choice can be made between these two by connecting C73 (B5-C5 on schematic) to one or the other.

The third schematic shows mainly the USB interface hardware. Also included is the power connections, either from the battery pack or through a filter from USB. Both of these are active at the same time.


MCU
U0
TI_MSP430_F1611
SENSORS
EXPANSION
JTAG
FLASH
SERIAL ID
POWER
RESET SWITCH
USER SWITCH
MOUNTING HOLES
LEDS
Title
Telos
Size B
Document Number
(c) Copyright 2004: UC Berkeley
Rev B
Date: Wednesday, September 29, 2004
Sheet 1 of 3


Telos cc2420 802.15.4 Wireless Radio
(c) Copyright 2004: UC Berkeley
Wednesday, September 29, 2004
CC2420
U10
CC24K_AVDD
CC24K_DVDD
CC24K_DVDD3
RADIO_RESET
RADIO_VREF_EN
RADIO_SCLK
RADIO_CS
RADIO_GIO0
RADIO_GIO1
RADIO_SFD
RADIO_SO
RADIO_SI
PKT_INT
DVCC
16MHZ - 16pf
2.4GHz matching network
use 50 ohm traces
at 2.4GHz
matched to 50 ohms


USB INTERFACE
Used for Reading Serial / Programming
Connects and Powered via USB
USB POWER FILTER
MOTE POWER
USB IDENTIFICATION EEPROM
I/O BUFFER
RESET SEQUENCE RECOGNITION
Telos USB Interface
(c) Copyright 2004: UC Berkeley
Wednesday, September 29, 2004

Appendix B

Flow Diagrams

This section includes connection diagrams of the major TinyOS components used in our system. This is to give a better understanding of how the components and interfaces connect to form the operating system. These can be generated automatically with the TinyOS system, simply by compiling with:

```
make tmote doc
```

Documentation in web page format is then created, including diagrams and descriptions of all the components used in the compiled OS. In the diagrams, a component is represented by a block, and the grey ovals pointing to it are the interfaces it provides. A component can also point to another component whose interface it uses.

Figure B.1 describes the Sense application. Sense uses a number of interfaces provided by the sensor, multihop routing and timer components. Figure B.2 next shows the multihop routing system used. It consists of the Data component, providing interfaces used by the Sense application. It also includes the Route setup component (MultiHopRSSIBATM) which both use interfaces provided by the SP, timer and radio components, among others. The SP, link abstraction layer component and the relevant interfaces used and provided by it are presented in figure B.3. The component written for the SHT-11 humidity sensor is shown in figure B.4. Figure B.5 incudes the component for the internal voltage sensor and the ADC component, that will be the same for all other ADC ports.

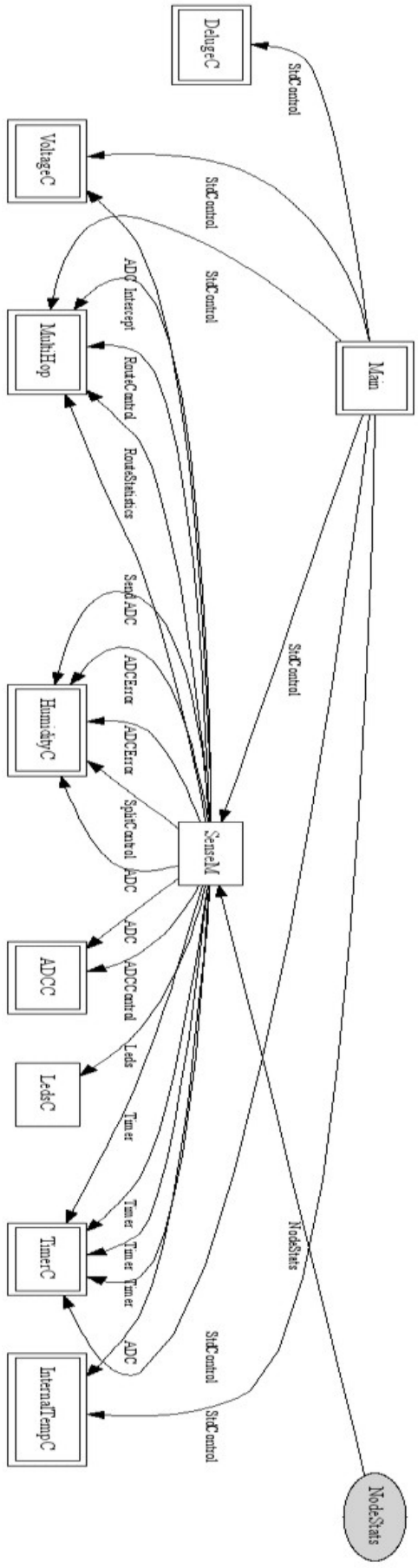


Figure B.1: *Sense application components and interfaces*

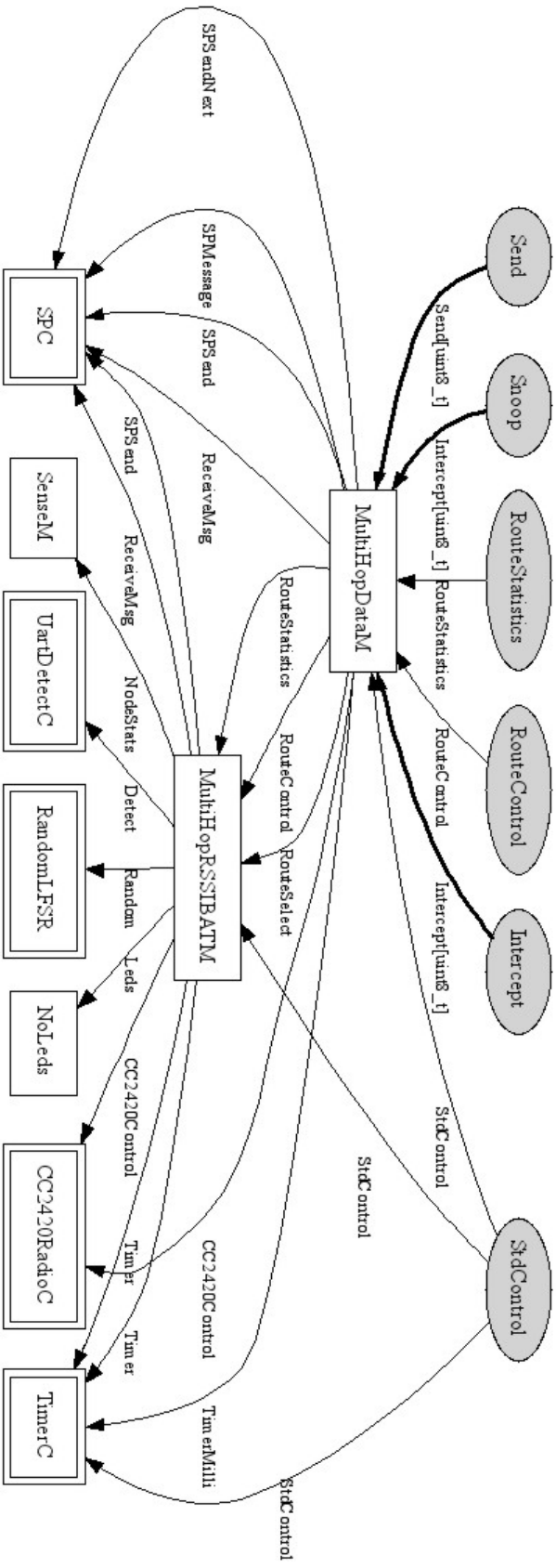


Figure B.2: *Multihop routing components and interfaces*

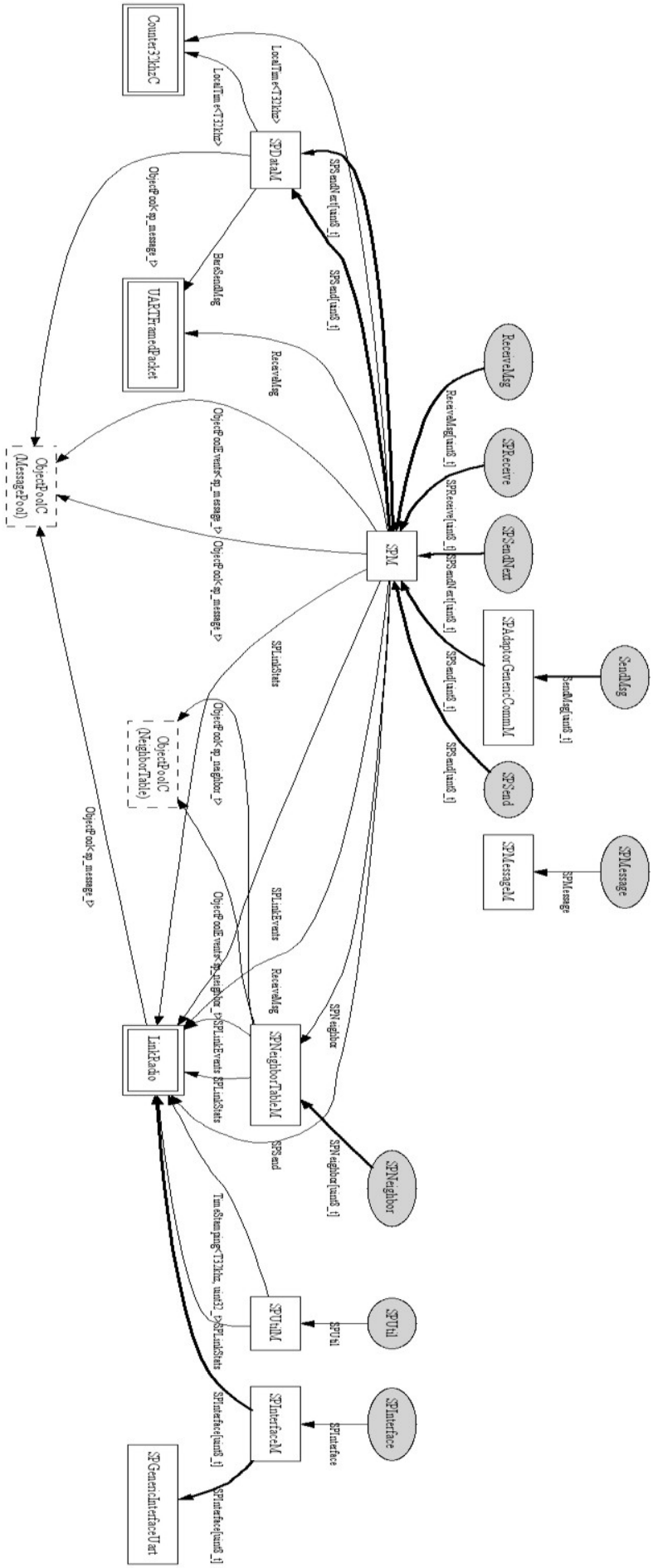


Figure B.3: *SPC link abstraction components and interfaces*

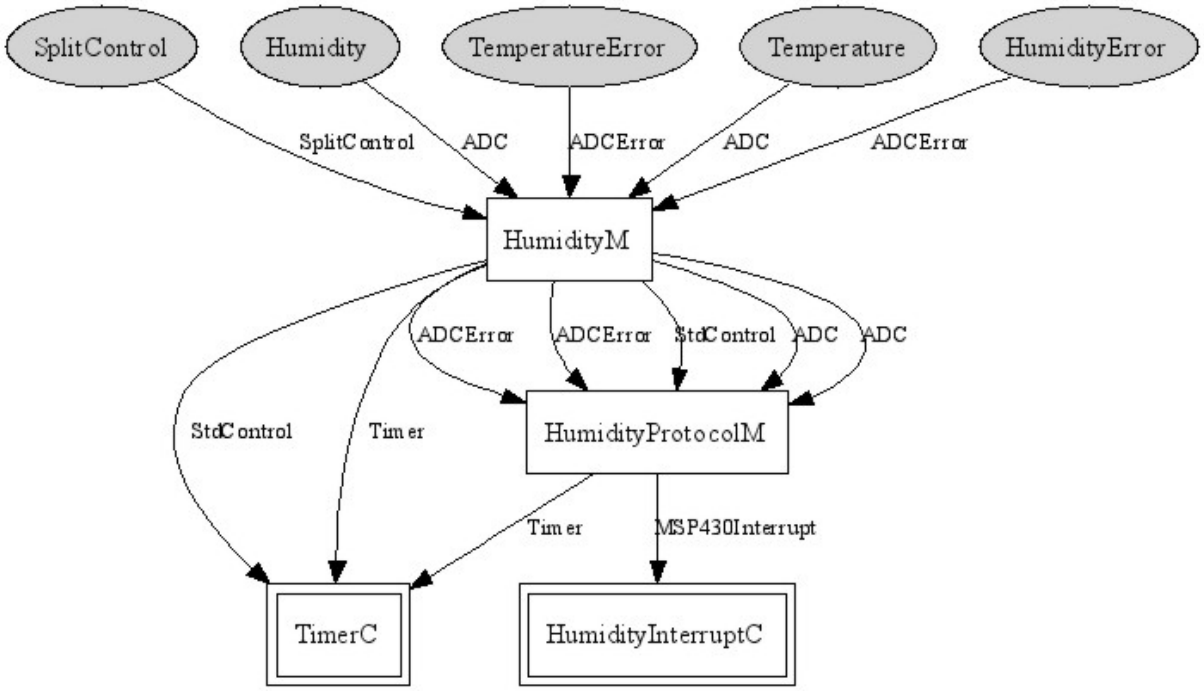


Figure B.4: *Humidity sensor components*

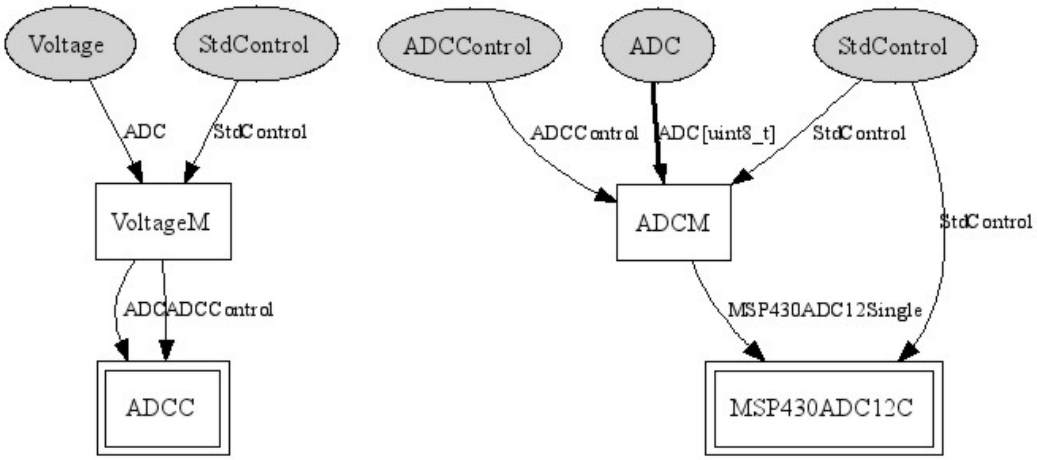
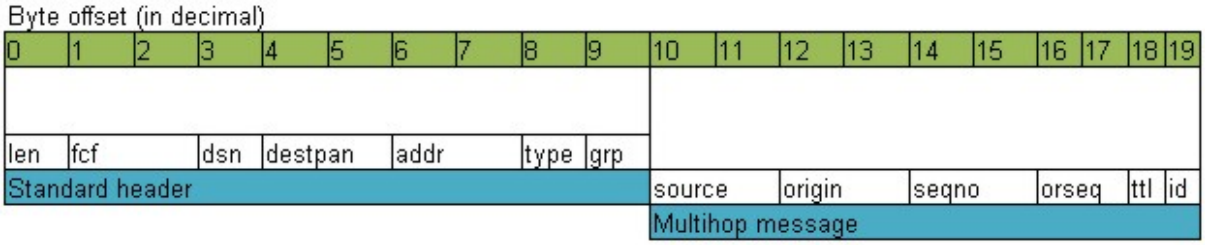


Figure B.5: *Internal voltage sensor components*

Appendix C

Message Format

The two main messages sent are the data messages from the Sense application, and the beacon messages from the multihop routing layer. Messages consist of a standard header (Figure C.1) that consists of data required for compatibility with the 802.15.4 radio. An extra multihop header is added for data messages sent through the multihop routing system. This includes information on the source and destination nodes, with sequence numbers. Figure C.2 includes the beacon and data message information. The Beacon message is added to the standard header when sent, producing a 20 byte message. The Sense message is sent through the multihop system and is, therefore, added after the standard header and multihop header, producing a 56 byte message.



Header description
TOSMsg.length; message length in bytes not including header
TOSMsg.fcf; IEEE 802.15.4 frame control field [reserved]
TOSMsg.dsn; IEEE 802.15.4 data sequence number [reserved]
TOSMsg.destpan; IEEE 802.15.4 Destination personal area network identifier [reserved]
TOSMsg.addr; TinyOS destination address (0x007E is the special UART address)
TOSMsg.type; TinyOS active message number
TOSMsg.group; TinyOS group id

Multihop message description
MultihopMsg.sourceaddr, address of the previous hop
MultihopMsg.originaddr; address of the origin of the message
MultihopMsg.seqno, sequence number of the previous hop of multihop messages
MultihopMsg.originseqno; sequence number from the origin of multihop messages
MultihopMsg.ttl; message time to live
MultihopMsg.id; Message ID

Figure C.1: *Message format*

Sense message description

SenseMsg.seqno; sequence number of message
SenseMsg.voltage; ADC reading from the voltage sensor
SenseMsg.itemperature; ADC reading from voltage sensor
SenseMsg.temperature; external temperature sensor reading
SenseMsg.humidity; external humidity sensor reading
SenseMsg.adc0; ADC reading from port 0
SenseMsg.parent; My current parent
SenseMsg.neighborsize;
SenseMsg.retransmissions;
SenseMsg.neighbors[]; neigbour adresses
SenseMsg.quality[]; neighbours link quality
SenseMsg.strength[]; neihbours RSSI value

Beacon message description

BeaconMsg.Parent;
BeaconMsg.cost;
BeaconMsg.hopcount;
BeaconMsg.batterylevel;
BeaconMsg.timestamp;

Figure C.2: *Message format*

Appendix D

Program installation

D.1 TinyOS

D.1.1 Requirements

To run the TinyOS system one needs a PC running Windows XP or Linux, with USB support. Some long USB cables might come in handy to make node placement easier.One will need the Mote*iv* Tmote tools installation; the last version available was 2.0.0.4. This includes the bash shell, cygwin and the Java platform. Also included are the standard TinyOS components and the Boomerang distribution, that runs on top of the standard TinyOS system.

D.1.2 Installation

Run the Tmote tools install package, which will automatically install, first cygwin, then the Java tools, TinyOS system and then Boomerang on top of that. After setup is done the first Tmote can be connected to a USB port in the PC. Windows will ask if you want to install new hardware. Select the USB driver directory included with the tools CD. A new USB serial port will be installed, and the Tmote will be connected as a COM port.

D.1.3 TinyOS usage

To use the system one now need to run the Cygwin shell which takes you to a command prompt. The system is configured at /opt/moteiv/ . This directory includes the following directories:

```
apps
doc
tinyos-1.x
tools
tos
```

The apps directory holds the applications that can be compiled and installed on the Tmotes. It includes sample programs like *Delta*, *RadioDemo* and others. This is also where our *Sense* application is located. The *doc* directory holds compiled system documentation. *tinyos-1.x* incudes the standard TinyOS subsystem, with interfaces and components, and other contributed code. The *tools* include all the Java interfacing programs like the serial forwarder, Trawler data display application and others. Tools also holds extra *make* rules that are included when applications are compiled. *tos* holds all the Boomerang interfaces and components (in tos/lib/ you can find the multihop system, SP and others) you can use in your applications with platform specific code for the Tmote.

To compile our sense application, for example, you have to enter the right directory and compile the system for the Tmote platform (see code below). Another useful command is the "motelist" utility that displays the COM ports, that represent connected Tmote modules.

```
cd opt/moteiv/apps/sense
make tmote
motelist
```

After the application has compiled successfully it can now be installed on the Tmote by using the following commands.

```
make tmote reinstall,0
make tmote reinstall,0 bsl,4
```

The first command can be used if only one mote is connected on a USB port. The Tmote will be programmed with its network address set as 0. As is standard, address 0 is normally used for the base station node connected to the PC. When more Tmotes are connected to the PC the bsl,x command is added on. This means that the program must be sent to the boot strap loader on COM(x+1). (bsl programmer starts port numbering from 0).

The currently installed program on the mote is erased and the new code programmed. This process will give output in the Cygwin shell and the motes connection lights will also be flashing while programming is in progress. The motes are now ready to deploy.

D.1.4 Java apps

The Java tools used in our system are situated in the following directories:

```
opt/tinyos-1.x/tools/java/net/tinyos/tools/
opt/tinyos-1.x/tools/java/net/tinyos/sf/
opt/moteiv/tools/java/com/moteiv/trawler/
```

The Listen program in tools/ was edited to make the CSV logger. Also shown is the links to the Serial forwarder in sf/ and Trawler code. These directories all include a prewritten makefile and when editing the java files you need to run the make command to recompile the applications.

The applications can be started using the following commands:

```
MOTECOM=serial@COM4:tmote java com.moteiv.trawler.Trawler
java net.tinyos.tools.Listen
java net.tinyos.sf.SerialForwarder
```

The MOTECOM variable tells the java tools to communicate with the Tmote, in this example over the serial connection on COM4. The variable can also be set for connection to the serial forwarder.

This process can be made easier by writing the command as shown above in a text file and then creating a shortcut. Trawler, for example, is automatically installed with a desktop shortcut with the command below. The file /opt/moteiv/tools/bin/trawler sets the MOTECOM variable and starts the Trawler application.

```
C:\cygwin\bin\bash.exe --login /opt/moteiv/tools/bin/trawler
```

D.2 Truetime

D.2.1 Requirements

TRUETIME currently supports Matlab 7.x (R14 and later) with Simulink 6.x and Matlab 6.5.1 (R13.1) with Simulink 5.1. A C++ compiler is also required to run TRUETIME in the C++ version. For the Matlab version, pre-compiled files are provided in the archive downloadable from the TRUETIME web site.

The following compilers are currently supported (it may also work using other compilers):

- Visual Studio C++ 7.0 (for all supported Matlab versions) for Windows
- gcc, g++ - GNU project C and C++ Compiler for LINUX and UNIX

D.2.2 Installation

Extract the compressed archive (truetime-1.5.zip). Extracting the file creates a truetime-1.5 directory (which will be referred to as DIR in descriptions below). Before starting Matlab, you must set the environment variable TTKERNEL to point to the directory with the TRUETIME kernel files, DIR/kernel. This is typically done in the following manner:

- Unix/Linux: export TTKERNEL=DIR/kernel
- Windows: use Control Panel / System / Advanced / Environment Variables

Now add the following lines to your Matlab startup script, this will set up necessary path to the TRUETIME kernel files.

```
addpath([getenv(TTKERNEL)])
init_truetime
truetime
```

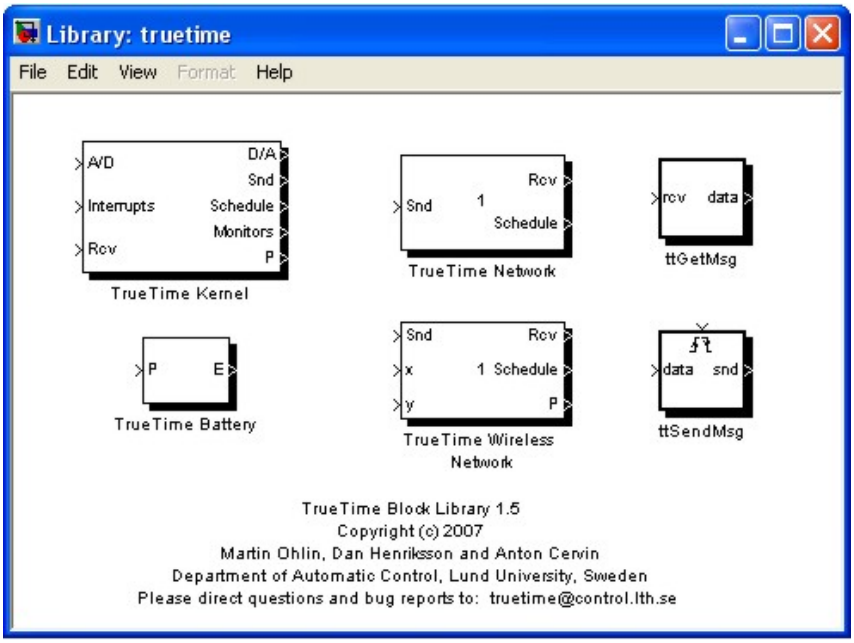


Figure D.1: *Truetime Block library*

Running *truetime* in the Matlab prompt, will now open the TRUETIME block library, see Figure D.1.

You can now start to build your model out of standard Simulink blocks combined with the Truetime blocks.

D.2.3 Compilation

Since the TRUETIME archive contains pre-compiled files, no compilation is required to run TRUETIME with the M-file API. TRUETIME, however, also supports simulations written in C++ code, which must be compiled. In this case, you first need to configure your C++ compiler in Matlab. This can be done by issuing the command $>> mex - setup$. In the setup, make sure that you change from the Matlab default compiler to a proper C++ compiler. see [29] for more information.

Appendix E

CD-ROM guide

Included with the thesis is a CD that contains all the software used in, and developed for this project. The following sections give short descriptions of the directories and software it contains.

E.1 Simulations

The simulations directory includes the software for the different simulations and models developed.

Matlab code Voronoi diagrams

Omnet++ Proposed Omnet++ simulation model

Truetime Simulation model

E.2 System software

This includes the Java interface applications as well as the mote operating system, with documentation.

Interface applications PC software

Operating system Mote software

Documentation Nesdoc software documentation

E.3 Thesis Files

The thesis latex code as well as a PDF compiled document.