Distributed Data Delivery
in Partially Connected Networks

Group 720
Anders Grauballe, Mikkel Gade Jensen, Achuthan Paramanathan, Janne Dahl Rasmussen

Supervisors: Tatiana Kozlova Madsen, Frank H.P. Fitzek

Worksheets

7th Semester
Autumn 2007
Distributed Systems and Network Planning
Aalborg University

Preface

This is the worksheets from the project "Distributed Data Delivery in Partially Connected Networks" carried out on the 7th semester, autumn 2007 at Distributed Systems and Network Planning, Aalborg University.
Citations are shown as e.g. [3] and figures and tables are numbered as e.g. 4.2 for the second figure/table in Chapter 4. We would like to thank our supervisor Tatiana Kozlova Madsen and co-supervisor Frank Fitzek for support and guidance throughout the semester.

The enclosed CD contains:

- Worksheets and paper in PDF format
- Project poster from SEMCON 2007 in PDF format
- Source code for the simulation and prototype implementation
- Results and log files from tests
- MATLAB files used to generate graphs

Anders Grauballe

Mikkel Gade Jensen

Achuthan Paramanathan

Janne Dahl Rasmussen

Contents

Chapter 1

Introduction

The basic scenario in this project is a cluster of sensors which are partially connected. A sensor is a small device which can measure e.g. temperature, pressure etc. in the surrounding environment. They are also able to communicate with other sensors in the area and relay messages to a control unit or gateway.

The sensors are partially connected because they are not all active at the same time, they are e.g. out of range or powered off. This cluster should be used to get a message from a sender to a receiver. The sender and receiver could, independently of each other, be either a gateway which is connected to one, more or all sensors, or a sensor which is part of the cluster. If a message is send to an inactive sensor/gateway, a method must be used for storing this message in the cluster until the receiver is active again. The goal of this project is to develop such a method. The sensors in this project can thus be used as storage devices and will in this case be referred to as motes.

There are two very simple methods for solving the problem. The first method is to store the message on one mote, which is very unreliable because it could be inactive when the receiver mote is active again. The other method is to store the entire message on all other active motes. This is very reliable but requires a large storage capacity.

The left graph in Figure 1.1 shows the relationship between reliability and the number of motes a specific messsage is stored on. The right graph in Figure 1.1 shows the relationship between the capacity needed and the number of motes one message is divided among.

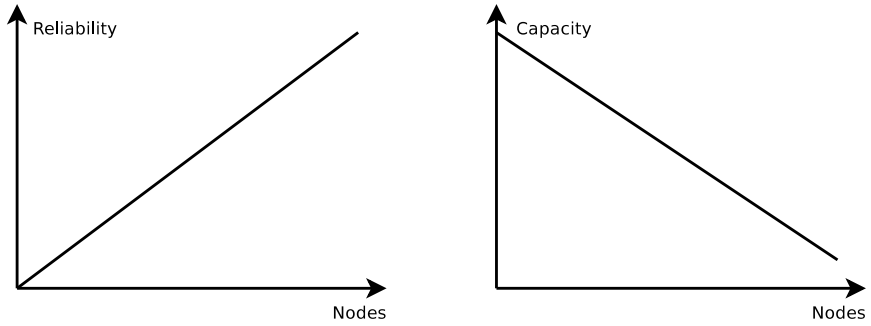

Figure 1.1: *The left graph shows how the reliability of getting a messages increases when the message is copied to more nodes. The right graph shows how the needed capacity on each node decreases when a message is split onto more nodes.*

The method which should be developed during this project needs to be a trade-off between the two simple methods. This means that the need for capacity should be as small as possible and the reliability should be as high as possible. This leads to the following initial problem:

- Develop a message which maximize reliability and minimize resource consumption when a message needs to be send through a partially connected cluster of sensors.

Parameters like throughput and bandwidth is not intended to be optimized in this project and will not be investigated further.

1.1 Scenarios

There are two possible scenarios in which the messages could be distributed: One-hop, where every node can communicate directly with all other nodes (similar to a star topology), and multi-hop where messages is sent through the network in an ad-hoc mannor.

1.1.1 One-hop scenario

This scenario is shown in Figure 1.2 where a gateway has a direct link to every node except for those which are currently offline. The scenario applies to e.g. a cluster of sensors spread in the ocean to measure changes in current. The gateway could be a ship which passes by on a regular basis to collect the measurements.

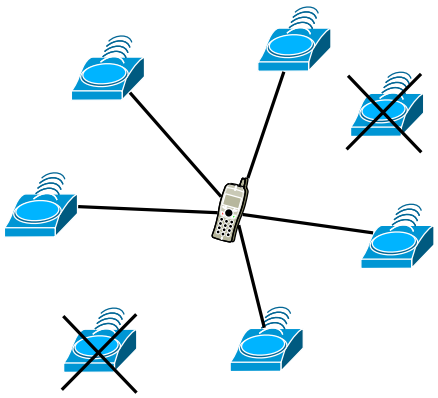

Figure 1.2: *The figure shows a gateway connected to motes in the network through one hop. Some are in a disconnected state (sleeping, off-state etc.)*

The problem specified in Section 1 involve the necessity to store a whole message or part of such in different sensors, so if this should be relevant it is only interesting to look at the aspects where neither the gateway or the sensor is available all the time. With regard to the gateway it would be interesting to look at the following scenarios:

- Gateway receives measurements from sensors.
  - Measurements from all sensors must be collected.
  - Only some of the sensors are available at the same time as the gateway.
  - Measurements from the offline sensors must be stored on the on-line sensors.
- One gateway sends a message through the sensors to the other gateway.
  - The whole message must be received.
  - The sensors must store the message until the receive gateway is available.
  - The sensors can't store the entire message.

	Options	
Gateways	One	More
Gateway availability	Always	Some periods
Sensor availability	Always	Some periods
Sensor stationarity	Static	Dynamic
Sensor offline	Random	Specified interval

Table 1.1: *Options for one-hop scenario*

So for both scenarios the sensors can only store some partial information, and therefore they must decide how to distribute the information so the entire information is available even if some of the sensors are not.

1.1.2 Multi-hop scenario

Multi-hop in networks is occurring when there is more than one hop in the route to the destination. The state and appearance of the node beyond the next known node, may be unknown because this node can not be seen by the initial node.

Different multi-hop scenarios:

- Gateway - network - gateway
- Gateway - network - sensor
- Sensor - network- gateway

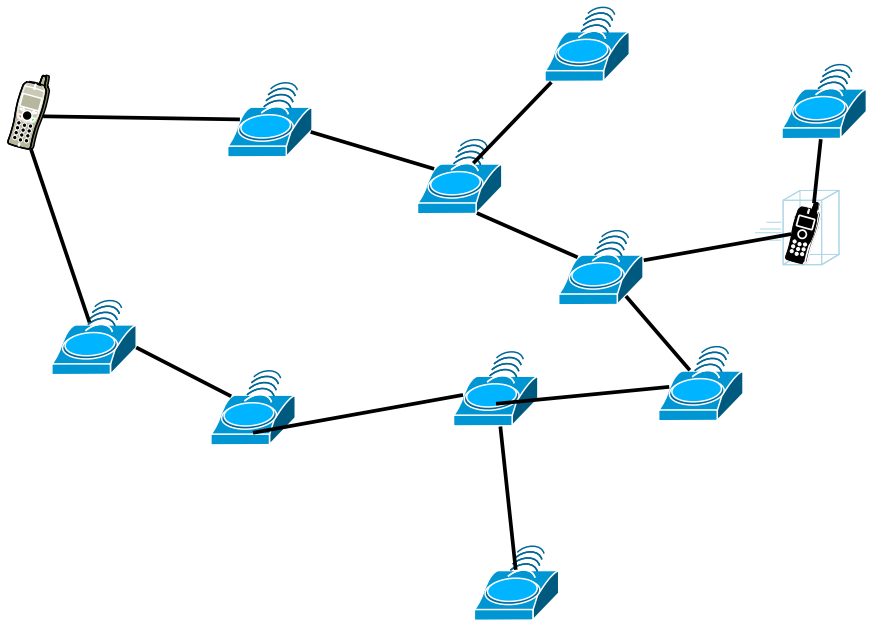

Figure 1.3: *A network where multihop is needed to send a packet*

A scenario can be seen in Figure 1.3. One gateway e.g. a mobile phone wants to transmit a mesage to another gateway. This can be done by selecting the shortest path in the network. If one node in this path fails then the message could be corrupted or lost therefore another approach could be used where the packets are broad/multicasted into the network so the risk of failure is lowered. The packets will then be broadcasted by the nodes in the network until

they reach the destination. The disadvantage in the approach is however that the network can become overloaded, therefore it may be better to multicast to a limited number of nodes instead of broadcast to everyone. In case of 30 nodes in the network, the packet could be sent to 1/10 $\approx$ 3 nodes in the network, then the performance might be better but the risk of failure is still limited.

Further explanation of the communication flow

The gateway will broadcast a packet to its neighbors that are in vicinity or the gateway. Then the nodes that have received this packet will broadcast it to their neighbors. If a node receives a packet which is already received it will discard the packet. In this way the packet will at some time reach the destination.

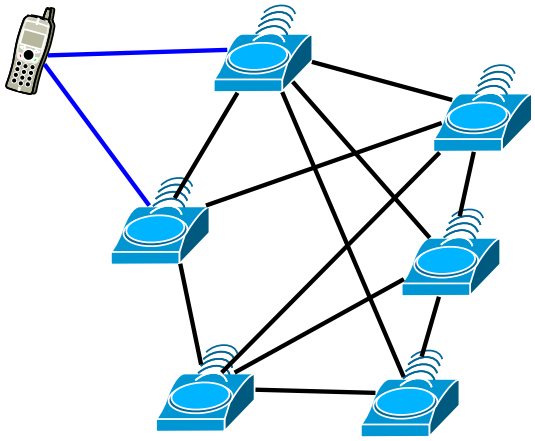

Figure 1.4: *Packets are distributed and stored in the network*

The scenario in Figure 1.4 shows a gateway partially connected to a cluster. By sending the packet to more than one node in the network the probability of failure is lowered.

1.2 Epidemic Algorithm

Epidemic algorithm can be used to distribute information in a dynamic ad hoc network where nodes have limited knowledge of other active nodes or the property of that network. This algorithm is defined to follow nature regarding spreading information in an environment by just having limited knowledge of that particular environment. An example of such behavior could be: A virus infected person A gets in contact with an uninfected person B. Then person B will be infected with A's virus with some probability, later on B infects 5 more persons etc. [3].

1.2.1 State classification

In epidemic algorithm a node is defined to be in one of the following state at a specific time:

1. Susceptible: A node is said to be in susceptible state, when it does not have any message or information which is relevant to the network and therefore it may be open to receive any information.
2. Infective: In this state a node has a piece of information that it wants to spread to other nodes in the network. The time-to-live concept c_{max} can be defined in this state, as the maximum limit of times a message is transmitted e.g. each time a node transmit c_{max} will be decreased by one.
3. Removed: A node in this state has the information, but does not spread it.

By combining those states, classification of epidemic algorithm can be made. Those classifications are defined as follows:

Susceptible - Infective (SI)

In this classification all the nodes in the network is initially susceptible. An event may trigger a node to move to the state infective. When a node is in infective state it remains there until the whole population is infected or c_{max} is set down to zero, this scenario is depicted in Figure 1.5.

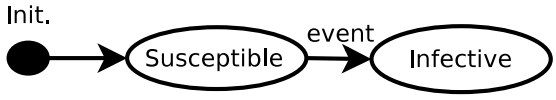

Figure 1.5: *This Figure shows a nodes behavior in a SI class. A node moves from susceptible to infective when it receive an information.*

Susceptible - Infective - Susceptible (SIS)

This classification reminds of the SI class, the difference however is; after being infected, a node is able to go back to susceptible state as shown in Figure 1.6. The event that triggers this could for example be, the information that is to be spread is outdated or $c_{max} = 0$.

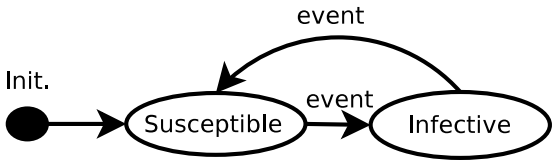

Figure 1.6: *This Figure shows the SIS classification.*

Susceptible - Infective - Removed (SIR)

In this classification after being infected a node begins to infect others in a network, this node keeps infecting others until a specific event occurs after that it stops spreading and it will never be infected again this scenario is shown in Figure 1.7. For instance, if a node for some reason did not spread the information, e.g. do to hibernation, it may conclude that the information is already distributed to the whole population.

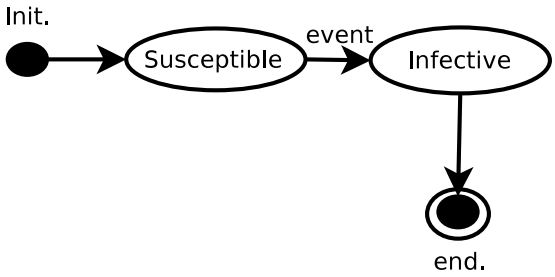

Figure 1.7: *This Figure shows the SIR classification.*

1.2.2 Communication methods

Based on the sates mentioned above, the communication paradigm between the nodes fall into three major categories:

Push

An infective node x initiate by choosing a communication partner y, for an instance the closest node in its reach, after there is a connection x sends its information to y, this is shown in Figure 1.8.

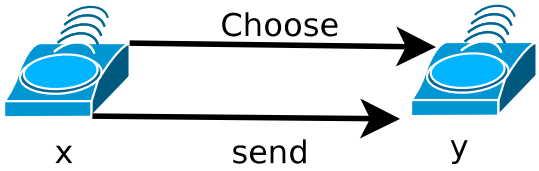

Figure 1.8: *An infective node chooses its partner and then infects it.*

Pull

In this method a susceptible node x initiate the communication and request any new information from the connected node y. If y is infective x will receive the requested information, se Figure 1.9.

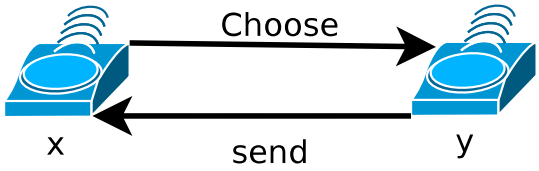

Figure 1.9: *A susceptible node chooses its partner and then gets any information from the infective node.*

Push and Pull

This method combines both the pull and push methods that is, each node may choose its partner and then either request a push or undertake the pull operation, this is shown in Figure 1.10.

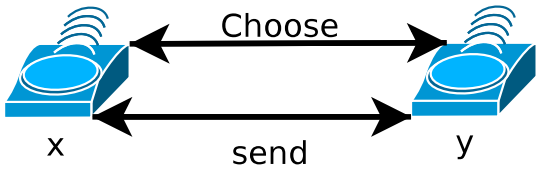

Figure 1.10: *Each node in this method may choose a partner and thenpush or pull the information.*

Chapter 2

Error/Erasure correction

In the following, some commonly used coding schemes for error/erasure correction is discussed to determine which one is best suited for this project. The schemes are describing how redundant data can be added to the original data to make it fault tolerant. The aim of this project is to distribute data once it has been encoded. The distribution of a message can be seen in Figure 2.1 where it is split into parts, redundant data is added and the parts are distributed among a number of motes.

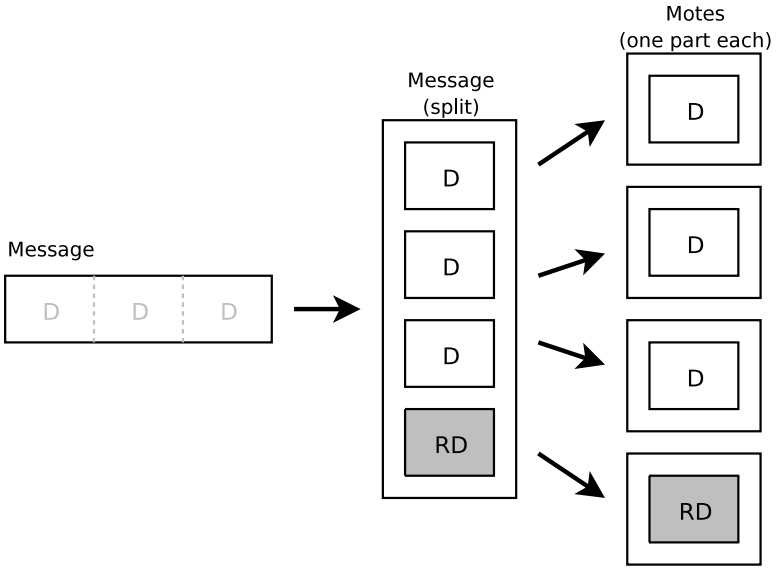

Figure 2.1: *In this case a message is split into 3 data parts (D) and one redundant data part (RD) is added. The parts are distributed among 4 motes*

2.1 Basic error detection/correction algorithms

(Main source: http://en.wikipedia.org/wiki/Hamming_code)
There are two main categories of error correction code. Convolutional codes and block codes. Convolutional codes works on a bit stream / channel with arbitrary length, while block codes works on blocks of bits with predefined fixed length. Block codes is suited for this project as messages should be split into blocks (of the same length) to be distributed among the motes.

In the following, some of the block codes will be described.

2.1.1 Parity

Before Hamming codes, some simple and inefficient codes were used. E.g. a single parity bit which was added to a block of bits representing an even or odd number of "1" bits in the block. 1 indicating an odd number and 0 indicating an even number i.e. an error free bit block with parity always contain an even number of bits (even parity). Parity bits is only usefull for detecting single bit errors as two bit "flips" in one block results in the same even/odd number of ones. Further more, a single parity bit cannot correct the bit error if any, as the position of the faulty bit is impossible to know.

2.1.2 Repetition

Repetition schemes on the other hand is a simple but inefficient way to correct single bit errors. The simplest is the (3,1) scheme (triple modular redundancy) where each bit is repeated 3 times before sending the the next one. Thus a 1 is transmitted as 111 and 0 as 000. A single bit error would result in e.g. 101 with 1 as the majority of the three bits. This block will be corrected into 111 and seen as a 1. The block might also be the result of a double bit error with 0 as the original value. In that case the block would wrongfully be interpreted as a 1. The efficiency of this repetition scheme is only 1/3 as three times the actual information must be sent.

2.1.3 Hamming code

Hamming developed a more efficient code which can correct single bit errors, but also detect double bit errors. There are various versions of the Hamming code, but the simplest one is the (7,4) code. This means that 4 data bits are encoded by adding 3 parity bits, 7 bits in total. The 3 parity bits makes it possible to determine whether one of the 7 bits has flipped during transmission, and exactly which one it was. Even is the flipped bits is one of the parity bits it can be corrected. Furthermore, the Hamming code can detect double bit errors i.e. if a bit flip occurs in 2 of the 7 bits, but is not able to correct any of the 2 bit errors. The actual encoding and decoding algorithm is not described here, but the following is an example of the encoding:

The bit sequence 1011 is encoded into **01**1**0**011 where the bold bits are the added parity bits.

2.2 RAID

Redundant Array of Inexpensive/Indepentent Discs (RAID) is a way of overcomming two problems in data storage on hard disc drives. The first problem is the fact that computer processing power is increasing more rapidly than the speed of reading or writing to/from a disc drive or another data storing device. This is a problem when processing a large amount of recorded data and storing the result, because of the bottleneck in the disc I/O and the processor can not be fully used. By combining more discs in an array, it is possible to perform parallel operations and still consider the array as a single disc. This results in a theoretical multiplier of the read/write speed depending on how many discs are included in the array.

The second problem in storage devices is hardware failiure. No matter how expensive or high quality a disc drive is, it will most likely fail at some point in time. Such a failiure can result in loss of data which can be highly critical. RAID can provide a fault tolerant setup where data is being replicated between two discs, or make use of parity bits or entire parity discs. In case of disc failiure on one disc, no data is lost either because a copy exists on another disc, or because the lost data can be rebuilt by the knowledge of the parity bits. Typically RAID systems will use either Hamming or Reed-Solomon code for parity calculation[8].

RAID comes in many different setups depending on the desired properties. The setups can also be combined to obtain both increased I/O speed and data redundancy. The 6 generel setups are described below and a schematic setup is shown in Figure 2.2.

RAID level 0 - Striping: In RAID level 0 each disc in the RAID array is split into strips which is alligned as shown in Figure 2.2. Data is written to the RAID array in consecutive strips across the discs. This way, a read operation on data devided among severel discs, can be executed in parallel. This level offers no fault tolerance, but only enhanced performance.

RAID level 1 - Mirroring: RAID level 1 works like level 0 by dividing data among the discs, but it also introduces data redundancy by creating a full backup of each disc in the array. This is shown in Figure 2.2 with an array of four discs duplicated on four other discs. Write speed is the same as level 0, but read speed is potentially doubled for large amount of data as the backup discs can be used in parallel. In case of failiure of one disc, no data is lost as there exists a copy which can be used in stead.

RAID level 2 - Hamming words: Level 2 operates on word basis. E.g. in the setup shown in Figure 2.2, each four bits are coded with Hamming (7,4) code to create seven bit Hamming words. Each word is written across the discs, one bit at each disc. In case of

failiure the faulty disc can be replaced by a new one and the data rebuilt from the parity bits on other discs. This setup results in a lower overhead i.e. less space used for backup.

RAID level 3 - Dedicated parity: Level 3 uses simple single bit parity (even number of 1-bits in a word) on a dedicated disc as shown i Figure 2.2. In most communication protocols single bit parity can only detect errors, not correct them, but since it is known which disc is broken in case of failiure, the faulty bit in each word can be corrected.

RAID level 4 - Striping parity: Like level 0 and 1, level 4 operates with strips, and uses dedicated parity like level 3. As shown in Figure 2.2 the parity of e.g. strips 0-3 is placed on a the first strip of the fifth disc. This level performs well in case of failiure, but is a bad choice for a system with frequent small updates.

RAID level 5 - Distributed parity: Level 5 works like level 4 except that the parity strips are distributed uniformly through the discs as shown in Figure 2.2. This is done to compensate for the possible heavy load on a dedicated parity drive. In case of failiure however, the reconstruction of the lost data is a more complex process.

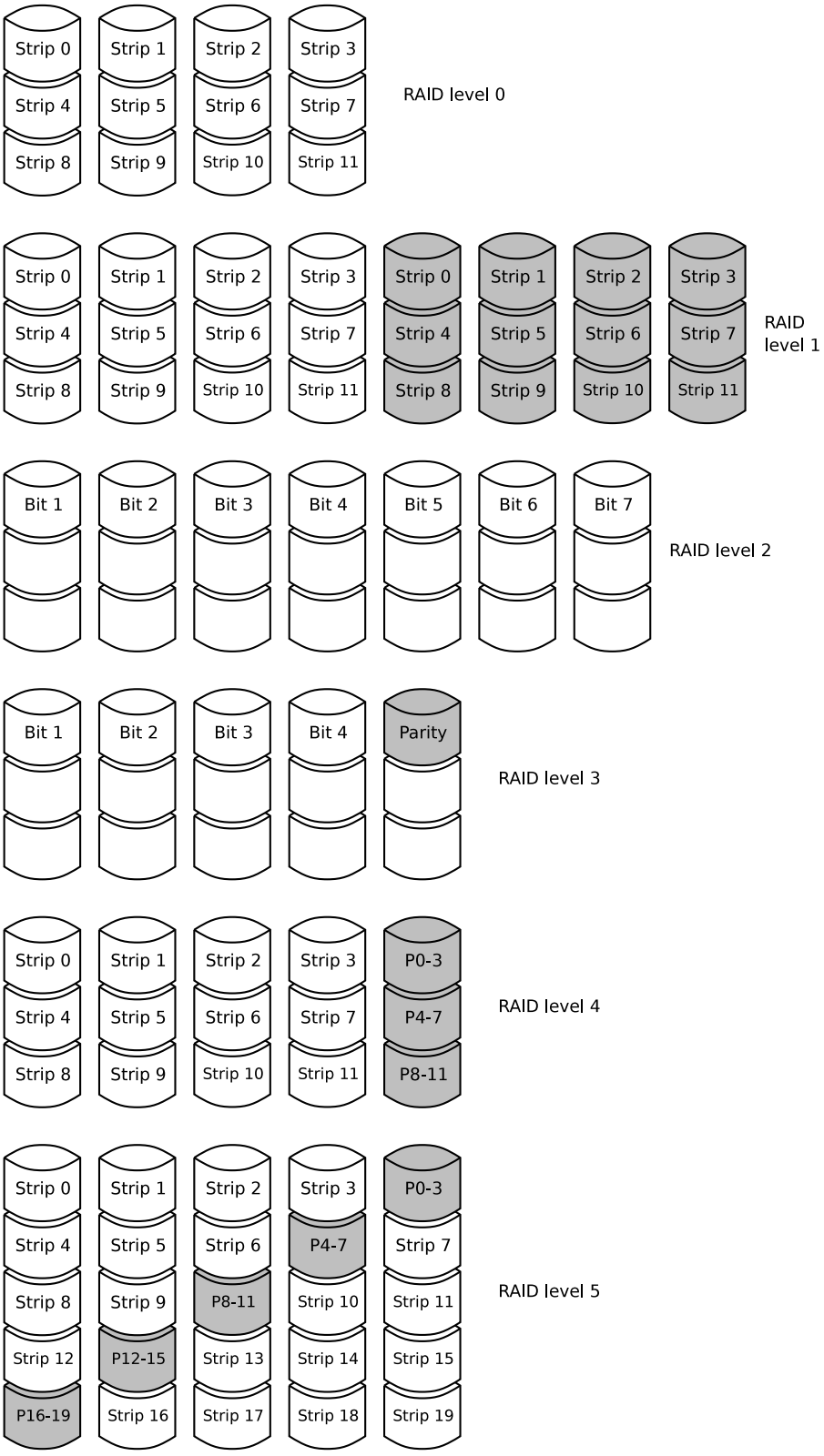

Figure 2.2: *RAID levels 0 to 5. The grey devices are backup and parity.[8]*

2.3 XOR parity

This algorithm takes advances of the fact that it is possible to determine which part of a message is missing and not just that some is missing. This case is called an erasure. Every two bit, one bit is added so only two of the tree bits is needed to retrieve the last.

Detailed explanation: Bit sequence: 0110011110. Split up in groups of two: 01-10-01-11-10 If the two bits are identical a 0 is added else 1 is added. This make the sequence look like this: 011-101-011-110-101. If sensor A gets the first bit in every group, sensor B gets number two and sensor C gets the last one.

A: 01011
B: 10110
C: 11101

If one sensor is missing it is possible to determine which data was stored on it. If sensor C is missing no need for recovery is needed because the C-bits are just the extra bits. If A is missing we have: x11-x01-x11-x10-x01. And we have to find all x's. First group the C-bit is 1 meaning A-bit and B-bit is not identical which makes the A-bit 0. This is done for all other groups and the original sequence without the C-bits is found: 01-10-01-11-10.

This is a very simple algorithm, but it also takes up a lot of space. If it is assumed that all sensors have a message of equal length and they share it with the others, they all store the message length times 1.5. If this number should be lower it is possible to only make an extra bit for every 3, 4, 5 bit and so on, but on the same time only one sensor out of 3, 4, 5 and so on can be missing if the entire message should be readable. The extra bit is then determined from the sum of the other bits, 0 if even and 1 if odd.

2.3.1 Extended XOR parity

The XOR parity algorithm can be extended be general for $n + 1$ nodes where n is the number of data nodes and the extra node is a parity node. XOR operations are performed on the data nodes to calculate the data of the parity node which makes the system recoverable if one random node is lost. Eg. if one data node is lost, then the information of this node can be calculated by performing XOR operations on the remaining nodes plus the parity node.

The following shows a scenario where one node is lost:

In a network with 4 nodes:

A: 0101
B: 1100
C: 0011
D: 1111

The data of parity node E is calculated by performing XOR:

$$0101 \oplus 1100 \oplus 0011 \oplus 1111 = 0101 = E$$

If D is lost it can be recalculated by performing XOR on the remaining nodes plus the parity node:

$$0101 \oplus 1100 \oplus 0011 \oplus 0101 = 1111 = D$$

2.4 Reed-Solomon

Reed-Solomon (RS) is a coding technique used to ensure reliable data and to make systems fault-tolerent. Examples of the use of RS are CD's, DVD's, Raid storage and DVB systems [4].

In the following section a description of how RS can be used in a Raid like system is given. [5]

In RS l defined to be the number of data devices and m is the number of checksum devices. The total number of devices in the system is $n = m + l$. The data on each devices is divided into words w that is the word size in bits. The RS coding is performed as a linearly combination of these words and checksum words on the checksum devices. In RS it is possible to regenerate the data if up to any m devices in the system are missing.

The following shows and example on how to recover after failure. Some theory is also explained in this example:

In a system there are 4 devices each holding 4 bits of information, so $l = 4$ and $w = 4$. The goal of this example is to recover the devices in the case where any 3 devices fail, so the number of checksum devices must be $m = 3$. The system then consist of 7 devices and a controller (gateway) which detects how many and which devices will fail.

The mathematical operations multiplication and division is over a Galois Field that is defined to be $GF(2^w)$, addition and subtraction are performed by XOR operations and are denoted by $\oplus$. The operations over Galois Field is in this example performed by a table lookup [5].

The information on the devices are the following:

$$d_1 = 0101 = 5$$

$$d_2 = 1000 = 8$$

$$d_3 = 1101 = 13$$

$$d_4 = 1011 = 11$$

To compute the checksum for the checksum devices a function F is applied to the data devices:

$$c_i = F_i\left(d_1, d_2, ..., d_n\right) = \sum_{j=1}^{n} d_j f_{i,j}$$

F is a $m \times l$ Vandermonde matrix which in this case is calculated like this:

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & \cdots & l \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \cdots & l^{m-1} \end{bmatrix} \begin{bmatrix} 1^0 & 2^0 & 3^0 & 4^0 \\ 1^1 & 2^1 & 3^1 & 4^1 \\ 1^2 & 2^2 & 3^2 & 4^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 5 & 3 \end{bmatrix}$$

When the Vandermonde matrix is found the checksum can be calculated Multiplications are performed over $GF(2^4)$ and the additions are done by XOR operations.

$$\begin{aligned} c_1 &= (1)(5) \oplus (1)(8) \oplus (1)(13) \oplus (1)(11) \\ &= 5 \oplus 8 \oplus 13 \oplus 11 \\ &= 0101 \oplus 1000 \oplus 1101 \oplus 1011 = 1011 = 11 \end{aligned}$$

$$\begin{aligned} c_2 &= (1)(5) \oplus (2)(8) \oplus (3)(13) \oplus (4)(11) \\ &= 5 \oplus 3 \oplus 4 \oplus 10 \\ &= 0101 \oplus 0011 \oplus 0100 \oplus 1010 = 1000 = 8 \end{aligned}$$

$$\begin{aligned} c_3 &= (1)(5) \oplus (4)(8) \oplus (5)(13) \oplus (3)(11) \\ &= 5 \oplus 6 \oplus 12 \oplus 14 \\ &= 0101 \oplus 0110 \oplus 1100 \oplus 1110 = 0001 = 1 \end{aligned}$$

To recover from failures a matrix A and vector E must be defined. A is a matrix with the identity I matrix in top and the Vandermonde matrix F in the bottom. E is the information on the data and checksum devices.

$$A = \begin{bmatrix} I \\ F \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 5 & 3 \end{bmatrix} \quad E = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \\ 13 \\ 11 \\ 11 \\ 8 \\ 1 \end{bmatrix}$$

In A' and E' the rows and elements of the failing devices are removed (in this case d_2, d_3 and d_4 are lost).

$$A'D = E' \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 5 & 3 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \\ 8 \\ 1 \end{bmatrix}$$

The system must be solved for D by performing Gaussian elimination or an equivalent operation to get the inverse of A', eg. by performing the matlab operation $inv(gf(A_{prime}, 4))$.

$$D = (A')^{-1}E' \Rightarrow D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 2 & 6 & 7 \\ 4 & 5 & 7 & 6 \\ 6 & 6 & 1 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 11 \\ 8 \\ 1 \end{bmatrix}$$

To regain the information the following operations are performed:

$$\begin{aligned} d_2 &= (3)\,(5) \oplus (2)\,(11) \oplus (6)\,(8) \oplus (7)\,(1) \\ &= 15 \oplus 5 \oplus 5 \oplus 7 = 8 \end{aligned}$$

$$\begin{aligned} d_3 &= (4)\,(5) \oplus (5)\,(11) \oplus (7)\,(8) \oplus (6)\,(1) \\ &= 7 \oplus 1 \oplus 13 \oplus 6 = 13 \end{aligned}$$

$$d_4 = (6)\,(5) \oplus (6)\,(11) \oplus (1)\,(8) \oplus (1)\,(1)$$
$$= 13 \oplus 15 \oplus 8 \oplus 1 = 11$$

The information in the system is then recovered.

2.5 Evenodd coding

Evenodd coding can tolerate a number of 2 erasures and is only based on parity (XOR). The number of data devices in the system must be a prime > 2 which is a disadvantage for this coding scheme. If n in a system is not a prime $p - n$ virtual data devices must be added e.g. if $n = 8$ then the next prime is $p = 11$ and 3 extra virtual data devices must be added to the calculation. The calculation of the checksum devices will not be explained but is basically just consisting of some XOR operations.

The evenodd scheme can be extended to be able to tolerate 3 erasures by using the Star approach where an extra checksum device is calculated.

Chapter 3

Comparison of coding methods

Several different coding methods has been described and in this worksheets a comparison will be made so it is possible to decide which method should be used in the rest of this project. The methods which will be compared are listed below:

- Full copy
- XOR
- RS
- No Redundancy
- Even-odd (with or without star extension)

There are several ways to compare these methods so the ones used in this worksheets will be described in the following.

Storage requirements for one message in the system:
The first comparison point is how much memory is needed in the entire system when checksum or redundancy is added. This is important to determine because one part of the aim for this project is to find a method where memory consumption on each mote is minimized.

How many motes are needed to reconstruct the message:
The second point is to determine how many motes need to be available for the message to be reconstructed. This is the second part of the aim for this project, namely to find a method where the reliability is maximized.

Scalability:
Scalability is important to consider because the cost of implementing the system with a different number of motes can be the decisive factor when chosing a method. In this project high scalability is defined to be when the ratio between online and offline motes stays the same or

when the ratio decrease i.e. the number of offline motes increases more than the number of online motes. Lov scalability is when the scalability increases i.e. the number of online motes increases more than the number of offline motes when more motes are used.

Rate of code:
Rate of code is a number that shows the trade-off between memory consumption and reliability. Higher rates are more space efficient but less fault tolerent [6]. The formula is : $R = \frac{n}{n+m}$, where n is number of data devices and m is the number of coding devices.

In the following it will be assumed that the cluster size is 12 if nothing else is mentioned.

3.1 Full Copy

In the method "Full Copy" the full message is distributed to each of the other motes. This means that in a cluster of 12 the message size is 12 times the size of one message.

When the entire message is distributed it also means that only one mote has to be available for the gateway to get the message. Therefore 11 out of 12 can be offline if the message should be read by the gateway. Any number of the 12 motes can be missing except one, as all messages can be found on every mote.

This method has very high scalability because the number of posible offline motes are always the number og total motes minus one.

In this method, the rate of code is calculated with $n = 1$ and $m = 11$. This gives a rate of code of $R = \frac{1}{1+11} = \frac{1}{12}$

3.2 XOR

We are considering two different kind of XOR-methods. The first one where a parity bit is added in the end of each method and one where a parity bit for each two bit in the message. In both methods each bit is distributed to different motes. The first method will be called XOR and the second XOR-group. In a cluster of 12 motes a message using the XOR-method has the size of 1.09 times the original message. When using XOR-group one message has the size of 1.5 times an original message.

Using XOR one mote out out of the 12 can be offline and using XOR-group one out of each three motes can be offline when the gateway should read the message.

With XOR it does not matter wich mote is offline, but with XOR-group it does matter in this method witch motes are offline. Considering a scenario where each mote in a 12-mote cluster has a message they need to distribute to the others. Each mote adds the parity-bit and sends a part to each of two neighbors, see Figure 3.1. Then 4 motes should be dispensable but if two

motes in the same group are offline the message they share can not be received by the gateway.

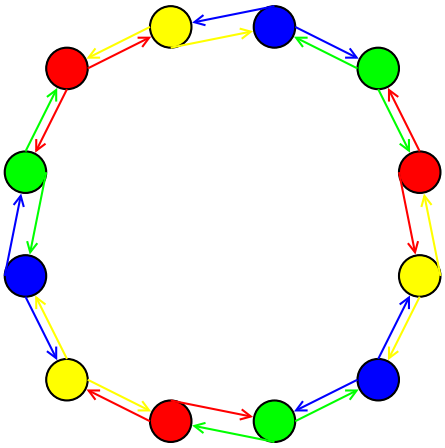

Figure 3.1: *This figure shows how a cluster of 12 motes distribute their messages. Only the same color can be dispensable at the same time. E.g. if a red mote is offline, only more red motes can be offline at the same time, not any other (blue, yellow or green)*

Scalability for XOR is low because only one mote can be missing no matter how many is added. For XOR-group it is low to medium. This is choosen because as long as the number of motes can be devided by three the ratio between offline and online motes stays the same, but when this is not the case less motes can be offline (See Figure 3.2).

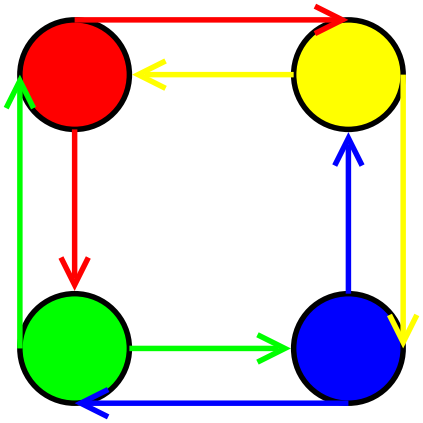

Figure 3.2: *This figure shows four motes where each distribute a message to two others. A group consist of a mote in a given color, e.g. blue, and the two motes which have blue arrows pointing to them. In this case only one mote can be offline at the time, because two motes from each group need to be online*

In XOR, the rate of code is calculated with $n = 11$ and $m = 1$. This gives a rate of code of $R = \frac{11}{11+1} = \frac{11}{12}$

In XOR-group, the rate of code is calculated with $n = 2$ and $m = 1$. This gives a rate of code

m \ n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	-	2	1,5	1,33	1,25	1,2	1,17	1,14	1,13	1,11	1,1	1,09	1,08	1,08	1,07
2	-	-	3	2	1,67	1,5	1,4	1,33	1,29	1,25	1,22	1,2	1,18	1,17	1,15
3	-	-	-	4	2,5	2	1,75	1,6	1,5	1,43	1,38	1,33	1,3	1,27	1,25
4	-	-	-	-	5	3	2,33	2	1,8	1,67	1,57	1,5	1,44	1,4	1,36
5	-	-	-	-	-	6	3,5	2,67	2,25	2	1,83	1,71	1,63	1,56	1,5
6	-	-	-	-	-	-	7	4	3	2,5	2,2	2	1,86	1,75	1,67
7	-	-	-	-	-	-	-	8	4,5	3,33	2,75	2,4	2,17	2	1,88
8	-	-	-	-	-	-	-	-	9	5	3,67	3	2,6	2,33	2,14
9	-	-	-	-	-	-	-	-	-	10	5,5	4	3,25	2,8	2,5
10	-	-	-	-	-	-	-	-	-	-	11	6	4,33	3,5	3
11	-	-	-	-	-	-	-	-	-	-	-	12	6,5	4,67	3,75
12	-	-	-	-	-	-	-	-	-	-	-	-	13	7	5
13	-	-	-	-	-	-	-	-	-	-	-	-	-	14	7,5
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	15

Table 3.1: *RS table showing memory consumption of one message in the entire system as a multiplier of one message. n is the total number of motes with one packet each and m is the tolerated mote failures.*

of $R = \frac{2}{2+1} = \frac{2}{3}$

3.3 Reed-Solomon

When using this method it is possible to decide how many motes should be dispensable and therefore Table 3.1 has been made.

It shows how man motes are dispensable in a given cluster size and what the message size is.

In this method any motes can be dispensable as long as the total number of offline motes does not surpass the number of checksum motes. For more information see Section 2.4.

The scalability for Reed-Solomon is hard to determine because the ratio can vary. We have chosen medium to high scalability, because it is possible to keep the ratio or to decrease it. It is also possible to descrease it, but this will most likely not be used when the other options are available.

3.4 No Redundancy

In this method each message is divided and distributed among all the nodes. This means that the message always have the same size as the original message i.e. 1, because no parity bit or redundancy is added. No motes can be dispensable as all of them have a different part of the message.

The scalability for this method is not possible to calculate because no motes can be offline.

In this method, the rate of code is calculated with $n = 12$ and $m = 0$. This gives a rate of code of $R = \frac{12}{12+0} = 1$

3.5 Even-Odd

In the Even-Odd method 2 devices are requirered as coding devices, which means that the system can tolerate 2 erasures. The system must consist of a prime number of data devices, if this is not the case a number of virtual devices must be added to be able to perform en/decoding. In this case with a cluster size of 12 it means one virtual device must be added to make the number of data devices equal a prime. The method can be extended to Even-Odd Star which tolerates 3 erasures by adding an extra code device.

Even-odd has low scalability because the number of offline motes does not change when more motes are used.

In Even-Odd without star, the rate of code is calculated with $n = 10$ and $m = 2$. This gives a rate of code of $R = \frac{10}{10+2} = \frac{5}{6}$

In Even-Odd with star, the rate of code is calculated with $n = 9$ and $m = 3$. This gives a rate of code of $R = \frac{9}{9+3} = \frac{3}{4}$

3.6 Comparison conclusion

To get an overview of the different methods the Table 3.2 has been made. Reed-Solomon has been calculted with 8 data motes and 4 checksum motes.

Neither full-copy or No Redundancy will be used in this project because the method needed should be a mix of these two methods.

Reed-Solomon and XOR has the same size of one message and same number of offline modes, but Reed-Solomon has the advantage of random offline motes (any 4 motes can be offline) and higher scalability. The Even-odd-method has the disadvantage that only 2 (or 3) motes can be offline at the time, and they have low scalability, as an advantege the message has a smaller size in the entire system than XOR and Reed-Solomon.

	Full Copy	XOR	XOR-group	RS	No redundancy	Even Odd without star	Even Odd with star
One message	12	1,09	1,5	1,5	1	1,2	1,33
Offline motes	11	1	4	4	0	2	3
Random offline	Yes	Yes	No	Yes	-	Yes	Yes
Scalability	Highest	Low	Low medium	Medium/ high	-	Low	Low
Rate of code	1/12	2/3	2/3	-	5/6	3/4	

Table 3.2: *This table shows the different parameters which has been use to compare the different methods. Reed-Solomon has been compared with 8 data-devices and 4 checksum-devices.*

From this comparison it has been chosen to use Reed-Solomon in the rest of this project.

Chapter 4

Modelling

4.1 Scenario

This scenario is given to provide an example of the system and to create assumptions about system parameters prior to the modelling.

The scenario will describe these different parameters in the system:

- Time of measurements
- Number of measurements
- Online/offline pattern
- Arrival time of the gateway

Common assumptions of parameters for the scenarios are:

- Propagation delay = 0
- The online/offline period is x seconds
- The probability for a mote being online is p_{on} and offline is $p_{off} = 1 - p_{on}$ (Bernoulli random variable)
- The system consists of n motes
- The system can tolerate m failures ($m < n$)
- No motes will fail (uncontrolled breakdown) at any time
- The gateway should only arrrive after the measurement/distribution phase has ended

The motes are all turned on and each will decide to enter online mode with probability p_{on} or offline mode with probability $1 - p_{on}$. If a mode is online it will perform a measurement and afterwards try to encode and distribute this measurement as n packets to the other motes. If some motes are offline the distributing mote will extend its online time until the offline motes change to online and the packets can be distributed.

After this procedure the mote will again decide whether the next period should be online or offline. This will happen on all motes, so after some time which can be denoted as measurement and distribution time $M + D$, all motes will contain n packets. When distribution is done, the motes will enter the idle phase I, shifting between online and offline mode with probability p_{on} to enter online mode.

The gateway can then arrive at a random time within a time interval G after the $M+D+I$ phase has ended, triggering the online motes to send their data so the gateway is able to regenerate the measurements from the received packets. When the gateway has left, the $M + D$ phase will start again and the motes which were offline when the gateway was present will delete their packets and start measuring again. This scenario can be seen on a time line in Figure 4.1.

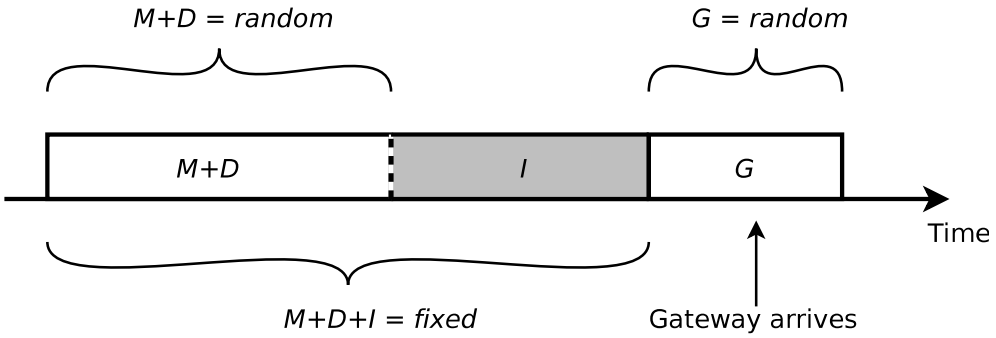

Figure 4.1: *The time line shows the different phases in one cycle in the system. $M+D$ is a random period and I is the remaining time until the gateway phase G*

From this scenario it is seen that the gateway is able to recover measurements if they are correctly distributed within $M + D$, and if enough motes are online at a certain time after the $M + D$ phase. The aim of the modelling is then to minimize the probability p_{on} which will save battery time and still be able to meet the reliability requirement p_R for reconstruction of a message.

4.2 Probabilistic system model

In this section a mathematical model of the sensor network is derived. The aim of this model is to describe the probability p_r that a gateway successfully will reconstruct the distributed message from the network. In order to do so, some motes have to be online at the same time

when the gateway request for data. By using the Reed-Solomon coding scheme, the system can tolerate losing as many data parts as there are checksum parts, that is, the probability of a successfull reconstruction of the message given a correct distribution.

$$p_r = \Pr(\text{Number of offline motes} \leq m) = \Pr(\text{Number of online motes} \geq n - m) \tag{4.1}$$

As the number of online motes is a series of n independent trials with a probability of success p_{on} and a probability of failure $1 - p_{on}$, it is a binomial random variable.[7]

4.2.1 Reconstruction probability

Calculation of p_r given a correct distribution, by binomial distribution

$$\begin{aligned} p_r &= \Pr(\text{Message successfully reconstructed}) \\ &= \Pr(\text{Number of offline motes} \leq m) \\ &= \Pr(\text{Number of online motes} \geq n - m) \\ &= \Pr(n - m \text{ motes online}) + Pr(n - m + 1 \text{ motes online}) + \cdots + Pr(n \text{ motes online}) \\ &= \sum_{k=n-m}^{n} \binom{n}{k} p_{on}^k (1 - p_{on})^{n-k} \end{aligned} \tag{4.2}$$

The probability of the message being successfully passed to gateway p_r is based on all k online motes with individual offline probability of p_{on},

The following shows an example of a calculation of the probability p_{on} using the binomial distribution. To do this it is assumed that:

- The ReedSolomon RS(12,4) coding scheme is used (total number of motes is set to $n = 12$ and $m = 4$)
- The message must be reconstructed with a probability $p_r = 80\%$.

In the example p_{on} is unknown and X denotes the number of online motes

$$P\{X \geq 8\} = \sum_{k=8}^{12} \binom{12}{k} p_{on}^k (1 - p_{on})^{12-k} = 0.8 \tag{4.3}$$

$$\Leftrightarrow \quad p_{on} = \begin{cases} -0.38 \\ 0.73 \end{cases} \tag{4.4}$$

According to equation 4.4, motes must be online with a probability of at least 0.73 as a probability is a non-negative number between 0 and 1.

Figure 4.2 shows the value of p_{on} as a function of p_r. It can be seen that each mote has to be online with a probability of 0.73 at 80%. The figure also shows the non-cooperative case with 12 motes. In this case motes must be online with probability 0.98 to make the system 80% reliable.

From Equation 4.2 using $m = 0$ the non-cooperative formula is:

$$p_r = p_{on}^n \tag{4.5}$$

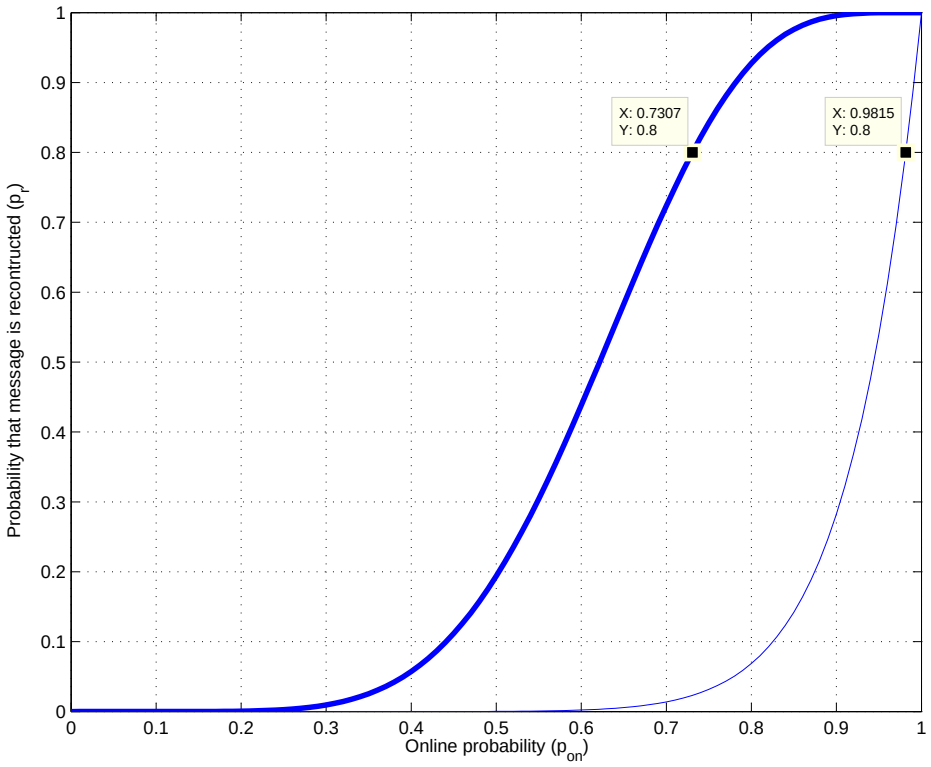

Figure 4.2: *The graph shows the relationship between the system reliability p_r and the online probability of the individual mote p_{on}. The bold line is RS(12,4) and the thin line is RS(12,0) i.e. the non-cooperative case.*

4.2.2 Distribution probability

Probability that one message is distributed to all motes within the $M + D$ phase under the assumption that only one message is present in the system. I.e. a sensor is distributing a message to n motes which are in idle state.

Figure 4.3 shows the decision periods of a mote in idle state. This case shows the worst case of a continuously offline mote until $(N-1)T$ which means that the message can still be recovered in the last period where the mote is online.

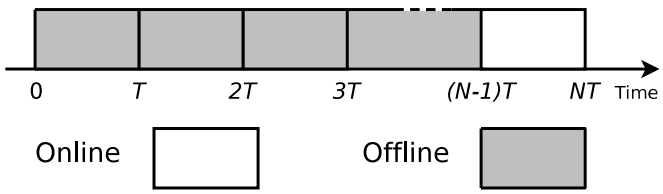

Figure 4.3: *Worst case of a mote being continuously offline until time $(k-1)T$ and still receive a message before time kT*

The probability of successful distribution is also calculated based on the binomial distribution where each trial is a mote being offline all the time within $N \cdot T$ or not. At each period T in time, the mote will make an independent decision with probability p. Thus the probability of the same state (on/off) in N periods is p^N

$$
\begin{aligned}
p_d &= \Pr(\text{Message successfully distributed in time } N \cdot T) \\
&= \Pr(\text{All motes online somewhere within time } N \cdot T) \\
&= 1 - \Pr(\text{One or more motes still offline at time } N \cdot T) \\
&= 1 - \Pr(Y \geq 1\text{mote still offline at } N \cdot T) \\
&= 1 - \sum_{i=1}^{n} \Pr(Y = i) \\
&= 1 - \sum_{i=1}^{n} \binom{n}{i} (p_{off}^N)^i (1 - p_{off}^N)^{n-i} \\
&= 1 - \sum_{i=1}^{n} \binom{n}{i} ((1 - p_{on})^N)^i (1 - (1 - p_{on})^N)^{n-i} \qquad (4.6)
\end{aligned}
$$

In Figure 4.4 the distribution probability p_d is shown as a function of the online probability p_{on} with 10 different N.

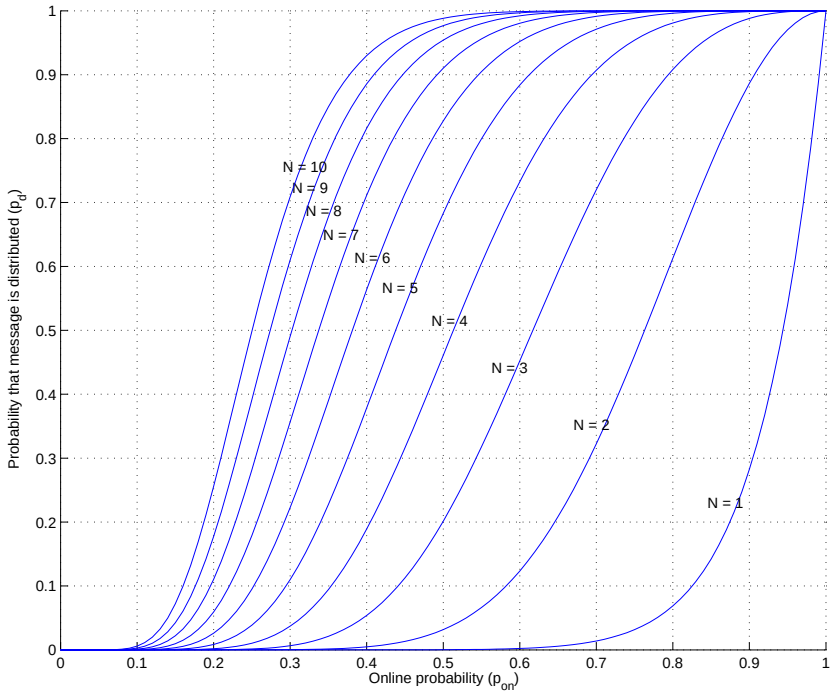

Figure 4.4: *The probability that the message is distributed within $N \cdot T$ as a function of the online probability p_{on} . The graphs are showing the relationship for different N. Still RS(12,4) is used*

4.2.3 Total system reliability

From Equation 4.6 and 4.2 the total system reliability is:

$$p_R = p_d \cdot p_r = \{1 - \sum_{i=1}^{n} \binom{n}{i} ((1-p_{on})^N)^i (1-(1-p_{on})^N)^{n-i}\} \sum_{k=n-m}^{n} \binom{n}{k} p_{on}^k (1-p_{on})^{n-k} \quad (4.7)$$

Assuming independency of the two events, and that the message can not be reconstructed before distribution is complete. Figure 4.5 shows the graphs of Equation 4.7. From the graph it can be seen that the relationship is approximately the same as p_r for large N.

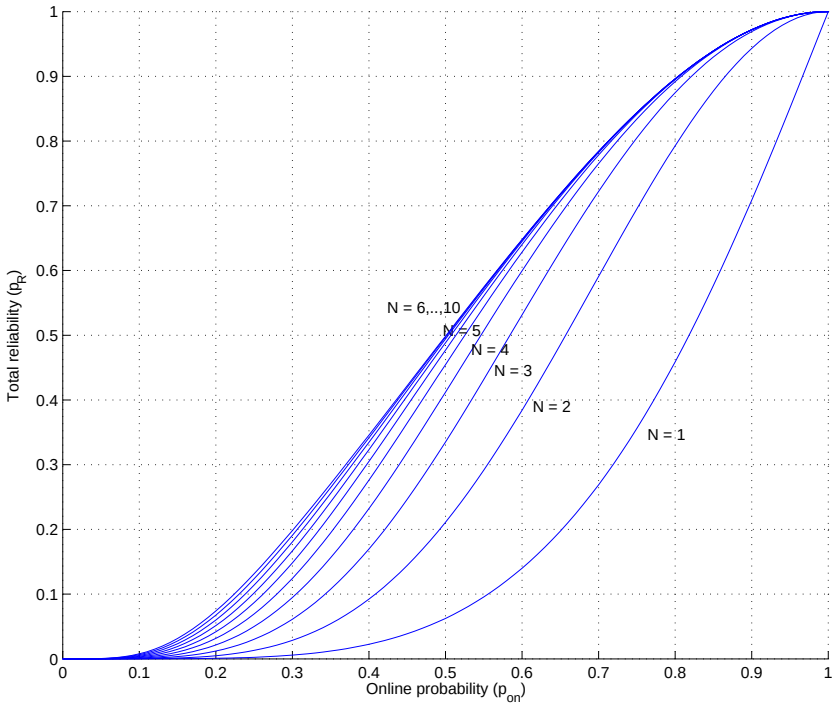

Figure 4.5: *The total reliability p_R as a function of the online probability p_{on}. The graphs are showing the relationship for different N*

The system reliability can be increased by making a small change of the scenario e.i. if the distribution is not complete when gateway arrives, the distributing motes can just send the whole message to the gateway directly. This yields the following reliability:

$$\begin{aligned} p_R &= \Pr(\text{Message successfully reconstructed}) \\ &= \Pr(\text{Distribution successfull}) \cdot \Pr(n-m \text{ motes online}) \\ &\quad + \Pr(\text{Distribution unsuccessful}) \cdot 1 \\ &= p_d p_r + (1 - p_d) \end{aligned} \tag{4.8}$$

Figure 4.8 shows the graph of the extended scenario of Equation 4.8 compared with the basic scenario of Equation 4.7. It can be seen that be increased for smaller p_{on}, but not for higher p_{on}. It seems that the system reliability goes to 1 as p_{on} goes to 0, but this also means that few motes will be online when the gateway arrives and thus few measurements will be available. Of course if $p_{on} = 0$ no measurements will be available.

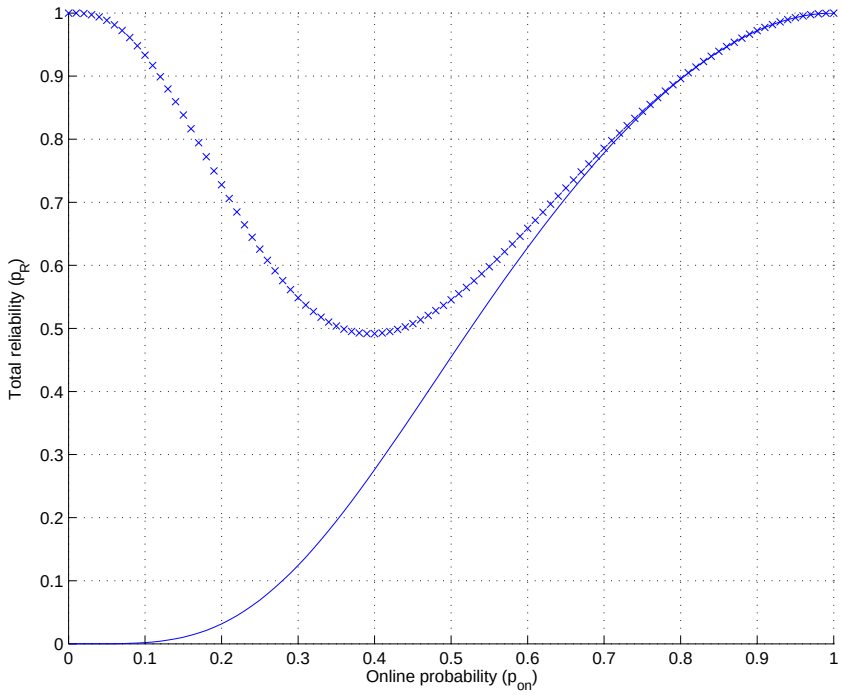

Figure 4.6: *The total reliability p_R as a function of the online probability p_{on}. The graphs are showing the relationship for $N = 5$. The solid line is Equation 4.7 and the x marks is the extended scenario of equation 4.8. RS(12,4)*

4.3 Resource consumption

4.3.1 Memory

The memory consumption in the entire system is shown in Table 3.1 for different n and m. The general formula is derived in the following.

The system consists of n motes in total, the number of checksum motes is m which leaves $n-m$ data motes. Each data mote and checksum mote holds $\frac{1}{n-m}$ of the message. Thus the total memory consumption in the system c_{total} is:

$$\begin{aligned} c_{total} &= \frac{n-m}{n-m} + \frac{m}{n-m} \\ &= \frac{n}{n-m} \end{aligned} \tag{4.9}$$

This ratio is also the total memory consumption in each device when holding messages for all

other devices (including own message).

4.3.2 Energy

The energy consumption can be expressed as the online probability times idle energy consumption plus the energy used for distribution: $E_{total} = p_{on} \cdot E_{idle} + E_{dist}$

Chapter 5

System Description

5.1 Use Case analysis

The use case diagram of the system, which is shown on figure 5.1, describes the general interaction between motes, environment and the Gateway. These interactions are grouped in two areas of use.

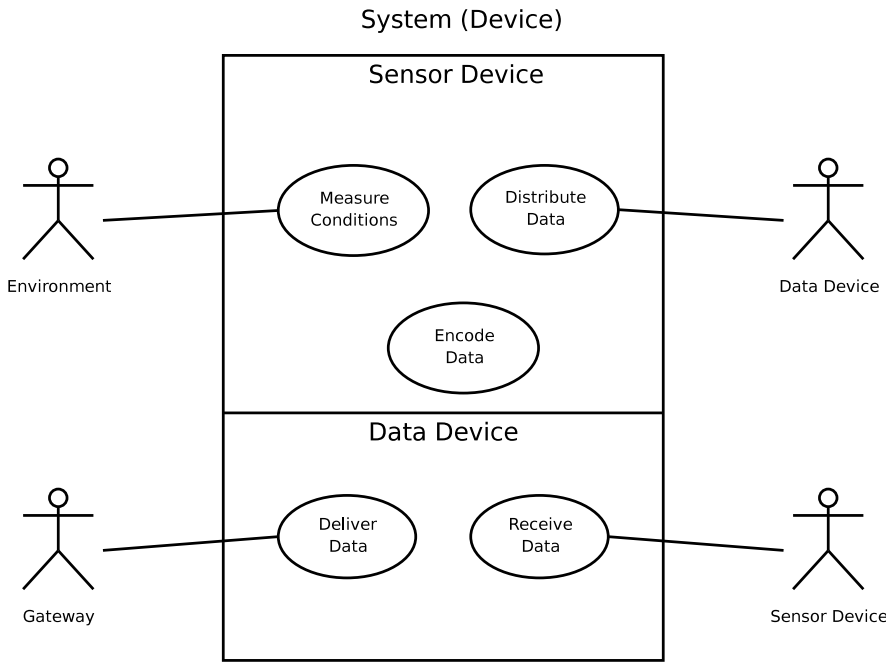

Figure 5.1: *Use Case diagram of the system*

5.1.1 Sensor

Measure Conditions
The sensor must be able to measure various conditions in the environment e.g. temperature, light intensity etc. The following actions is executed:

1. New measurements are needed (determined by e.g. periodic scheduling or significant changes in the environment)
2. The sensor retrieves measurements from the surrounding environment.
3. The measurements are stored in the memory of the sensor

Encode Data
The measured data must be encoded by adding redundant data and split into n parts for distribution to other motes. This is an internal use case which does not involve any external actors. The following steps are executed:

1. The data (measurement) is devided into n parts
2. The encoding is done to obtain m pieces of redundant data. Same size as the data parts.
3. A sequence number is atached to each part (data and redundant data) to link them together for later decoding.

Distribute Data
The encoded data must be distributed to $n+m$ devices to ensure reliability. The following steps are executed:

1. The data and redundant data parts are sent one by one to different available motes.
2. One part is kept by the device i.e. using itself as one of the $n+m$ motes.
3. If less than $n+m$ devices is online, the sensor device must wait for the remaining devices to become online.

5.1.2 Mote

Receive Data
Receive data, which involves the interaction between two motes, where a sensor sends packet to a mote in the network. The procedure for this case includes:

1. Listen for incomming data.

2. Receive available data from the network.

3. Stored the received data in memory.

Deliver Data
This use case shows the interaction between a mote and the gateway. A mote provides access to its data e.g. to be collected by a Gateway. The following states the procedure of this case.

1. The Gateway broadcasts requests for data to the network.

2. When a data mote detects the presence of a Gateway, it will transmit its data.

5.2 Activity Diagrams

In this section the UML activity diagrams of the system is analyzed. Each diagram shows which activity is executed and in what order.

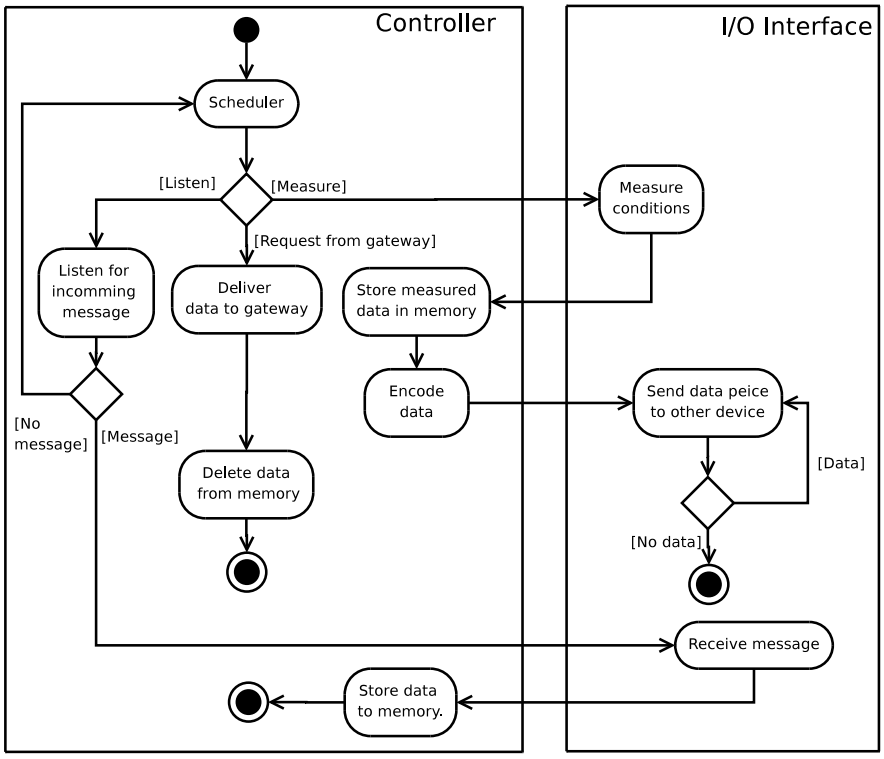

Figure 5.2: *Activity diagram for device*

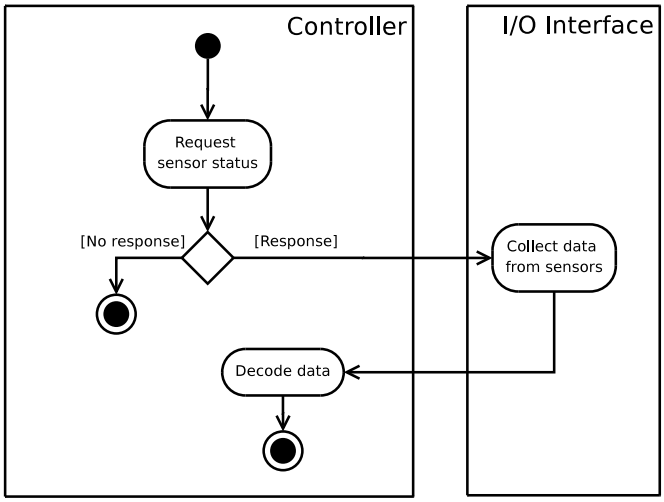

Figure 5.3: *Activity diagram for gateway*

5.2.1 Activity Scheduler

In this section the scheduling activity of the embedded device is described. The scheduling is driven by a kernel which consists of a number of processes; Measure, State decision and Receive. Each process is able to apply for processor time, when it receives a message e.g. measurement data from a neighboring sensor, se figure 5.4. The received message is then stored in a mail-queue according to the priority of the message. The scheduler then gives processor time to the highest priority process to execute its received message. Two equally prioritized messages will be handled by the scheduler regarding to the FIFO (first in first out) principle.

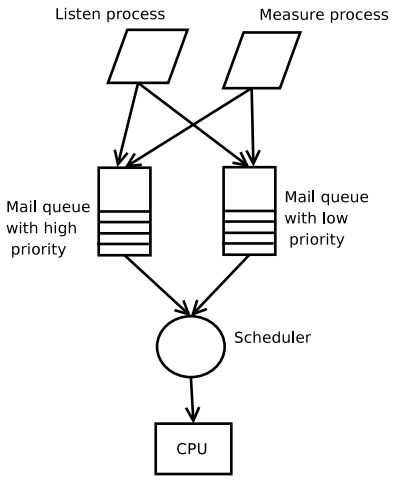

Figure 5.4: *Design diagram of the data driven kernel.*

As it described in figure 5.4, the data driven kernel has three processes where each has access

to a mail-queue where the received messages are stored, meaning each processes can apply for processor time for their messages according to the priority of the received message. Following explanation gives an overview of a scenario that may accrue:

1. The receive process initiates by listing to the RF to see if there is any message e.g. from the gateway or from other devices in the network.

2. If receive process identifies a pending message from the gateway it applies for processor time by sending a high priority message to the mail-queue. After that it also identifies a pending message from a sensor e.g. a measurement data, and aging it applies for processor time, but this time by sending a low priority message to the mail-queue. In case of no pending messages, the Listen process will release the processor, by sending a message to the measure process, but in this scenario this case will not accrue.

3. The scheduler will begin to execute the highest priority message from the mail-queue, which in this case is the message from the gateway, because of its priority. After this task is done, the scheduler will execute the next highest priority message, which is the message from a sensor.

5.3 Requirements Specification

This chapter contains the requirement specification of the system. The requirements are aimed for the prototype of the system.

5.3.1 System Requirement

1. The system must consist of at least 4 motes and a gateway.

2. The system must be setup, in as a one-hop network.

3. The system must tolerate at least 1 erasures

5.3.2 Mote Requirement

Requirements for mote and sensor devices

1. After the on/offline period of 5sec. each mote must decide whether to enter online or offline state.

2. The decision parameter must be uniformly distributed with a probability p for online state and 1-p for offline state.

3. Each received data payload must be maximum 27 Bytes.

4. Each mote, according to a request from the gateway, must transmit its data.

5.3.3 Sensor Requirements

Additional requirements for a sensor device

1. The sensor must collect measurements after each gateway presence.

2. Measured data must encode data using RS.

3. The encoded data must be distributed to other motes

5.3.4 Gateway

1. The gateway must announce its presence to all devices

2. The gateway must collect data from the motes

3. The gateway must decode the collected data parts into the original measurement data

5.3.5 Preconditions

1. Messages are never lost in the system. (All send packets are also received correctly)

2. The system contains a static number of devices.

3. Underlying MAC protocol is working correctly

5.4 State diagram

From the scenario different requirement or rules has been found. The list can be seen here below:

1. When a mote is distribution data it has to stay online till the distribution is done

2. The Gateway locks all online motes when it is present

3. The sensors only measure one time between each time the Gateway is present

4. When a mote has delivered the data to the Gateway it starts new measurements

5. After a mote has done a measurement, the data is encoded and distributed

6. The session number is increased by 1 every time the Gateway is present
7. If a mote receive data from another mote with a higher session number than the data already stored on the mote, the old data is removed and the mote starts to do new measurements
8. A mote decides to be online or offline independent of previous decision.

These requirements has been chosen from the idea that the first system should be as simple as possible. Then it can always be moderated and expanded later in the process.

From the requirements above a state diagram has been made. This can be seen in Figure 5.5.

The initial state is the measurement/distribution state. All sensors needs to be in this state exactly one time between each time the Gateway is present. From this state the more can choose to be either offline or online, resulting in two new states. If the mote is in offline state is stays there one time slot, after this time slot the same decision has to be made, either the mote stays offline or it becomes online. If the mote becomes online it is listening for motes who wants to distribute data to it or for the Gateway who wants the data which is stored on the mote. Id the mote receive a request from another mote it is in the receive data state. If the session number of the new data is higher than the data already sent it moves to increase session number and after this back to the measurement/distribution state. If the session number is the same as the saved data the mote changes to either offline or online state. If the mote in online state receives a request from the Gateway it moves to the Gateway present state where it delivers the data to the Gateway and afterwards starts measurering and distributing again. The statediagram can be seen in Figure 5.5

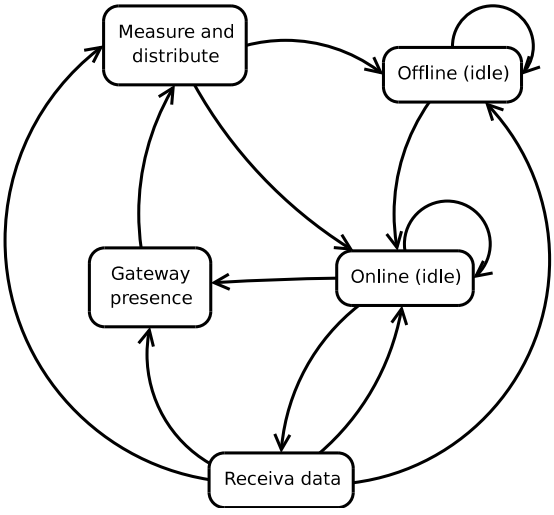

Figure 5.5: *Statediagram for one mote*

It is important to determine which states can occur at the same time. E.g. can a mote be offline while another is offline. Or can one mote measure and distribute while another is offline.

	GW	Measure	Offline	Online	Receive	Session no.++
GW	+	-	+	+	-	-
Measure	-	+	+	+	+	+
Offline	+	+	+	+	+	+
Online	+	+	+	+	+	+
Receive	-	+	+	+	-	+
Session no.++	-	+	+	+	+	-

Table 5.1:

The result can be seen in table 5.4

5.5 Time line

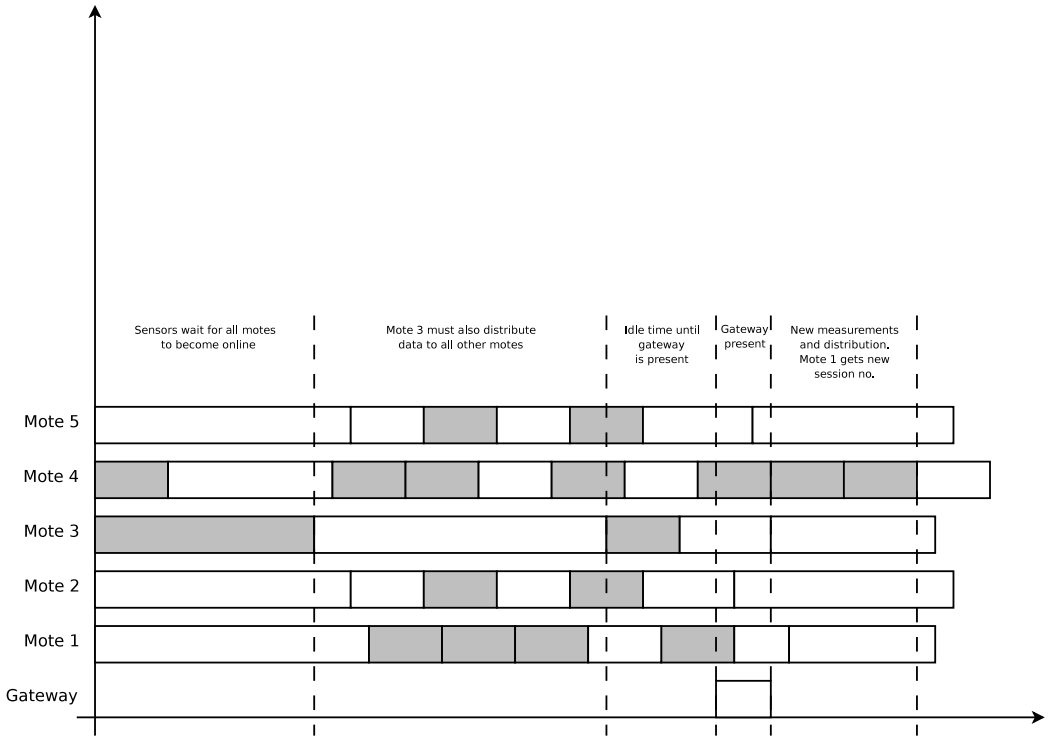

Figure 5.6: *An example of events in a cluster of 5 motes an a gateway*

Chapter 6

Design

6.1 Mote - Mote communication

The communication between the motes are nessesary when they need to distribute the messages so the Gateway can get them even when some of the senors are offline. There are two different flowcharts, one for sending data from a mote and one for receiving the data. First the send flow will be described.

6.1.1 Send flow

Figure 6.1 shows the flowchart when a mote is sending data to another mote.

The data needs to be encoded with the Reed-Solomon method and then split into packets, one for each other mote in the system.

- The sensor measures environmental conditions: This mote start by measuring som enviromental conditions which is the data that should be ditributes between the other motes.
- The measurement is encoded and split into packets: The data needs to be encoded with the Reed-Solomon method and then split into packets, one for each other mote in the system.
- The protocol header is added to the first packet: The Gateway must be able to decode the message again when it has recieved all the packets. Therefore each packets needs to have a header which describe which message the packet is a part of and which part it is.
- The device searches for the next online device in the cluster: The mote send the data-packets buy trying one of the other motes at the time. If the other mote is online, the packet is send, if the mote is offline the mote searches for the next mote which should receive a packet.

- The packet is sent to an online device which replies with a packet telling the sensor if the gateway session number is OK or obsolete: Because not all motes are online when the Gateway comes, it is important to tell the ones that was offline the det data they have saved is no longer important.
- If the session number is obsolete it will be increamented to the current session number and the packet retransmitted. Otherwise the next packet will be transmitted: If the mote was not online when the Gateway was present last, its session number will be obsoltete. Them it will delete all old packets it has stored. It must also encrease the session number at add a new header to the packet. If it has allready sent some packets to other motes, it must retrancmit there packets with the new header.

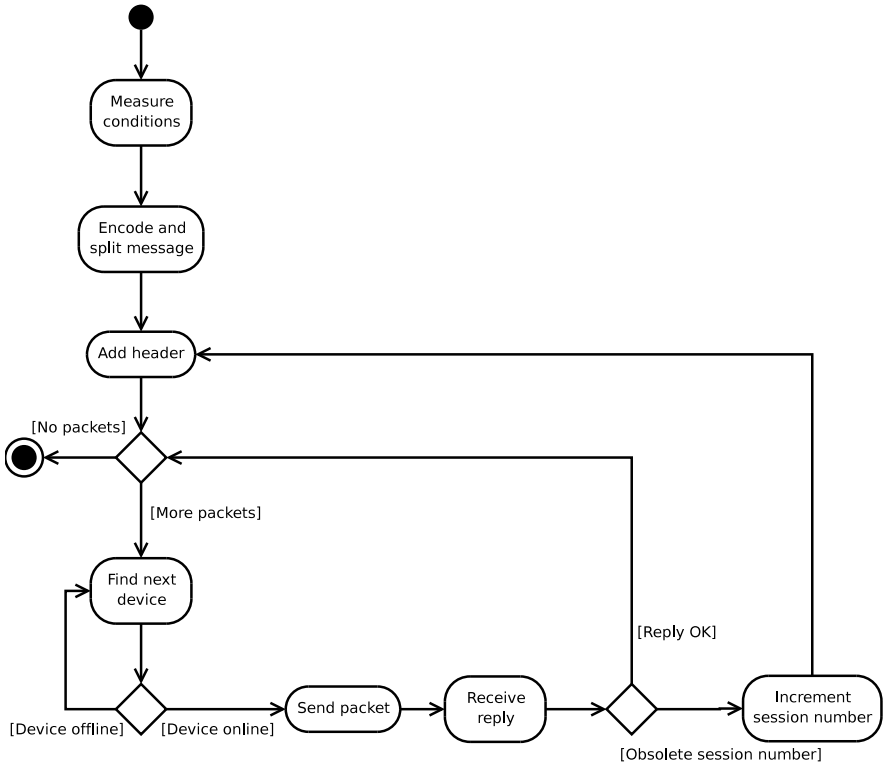

Figure 6.1: *Send procedure of two sensors*

6.1.2 Receive flow

Figure 6.2 shows the flowchart when a mote receives data from another.

1. The device is idle, listening for messages: All motes has some time where they are listning

for incoming request. This can be request from the Gateway or other motes which want to distribute there messages.

2. Upon reception of a packet, the session number is checked: When the mote receives a packet from another mote the session number must be checked. because the mote might have data which are obsolete.

3. The session number is obsolete: If the session number is smaller than the number the motes has stored then the send-mote is made aware of this so if can delete old data and retransmit the new packet.

4. The session number is new: The receiver-mote must delete all data, update the session number and store the receved packet.

5. Same session number: If the session number match the session number on the receiver-mote, the packet is just stored.

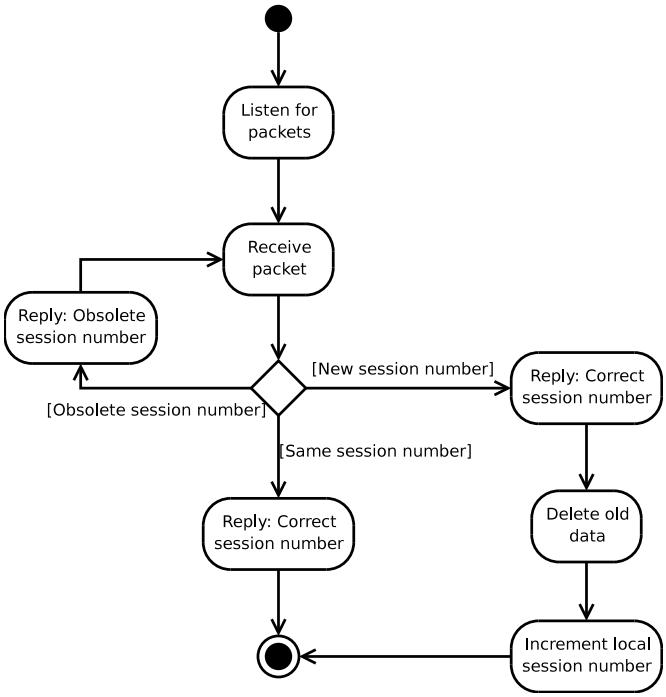

Figure 6.2: *Receive procedure between two sensors*

6.2 Gateway to mote communication

The communication between the Gateway and the motes is nessesary when the messages need to be send to the Gateway. This is done with the Gateway requesting the different packets from the motes and then decode the message.

6.2.1 Gateway to mote flow

Figure 6.3 shows the flowchart of the communication from the Gateway to a mote.

1. The gateway is entering a cluster and announces this to the cluster: All online motes needs to be told that the Gateway wants there packets. When the Gateway has broadcastet its present the online motes are locked and will therefor not be online til the Gateway had all the packets.

2. A request for data is made for each mote in the cluster: When the Gateway have broadcasted its presence it request the packets from one mote at a time.

3. When all requests are done the gateway decode the messages and leave the cluster: The Gateway needs to request packets from all online motes so the probability that the messages can be decoded is maximized.

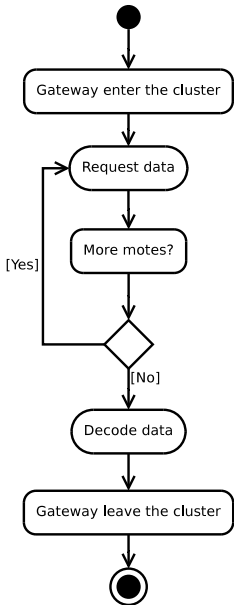

Figure 6.3: *This Figure shows the communication flow of the gateway when it enters a cluster*

6.2.2 Mote to gateway flow

Figure 6.4 shows the flowchart of the communication from the mote to the Gateway.

1. When the mote has received the announcement from the Gateway, it waits until it receive a request for data: When the mote has received the broadcast about the Gateways present it is locked until the Gateway has received its packets. This means that no other mote can dirstribute to it and it can not go offline.
2. When the request from the gateway is received, the mote will transmit its data.
3. After transmission the data on the mote is deleted and the sequence number is incremented: It is important that any old data is removed from the motes because the memory is limited.

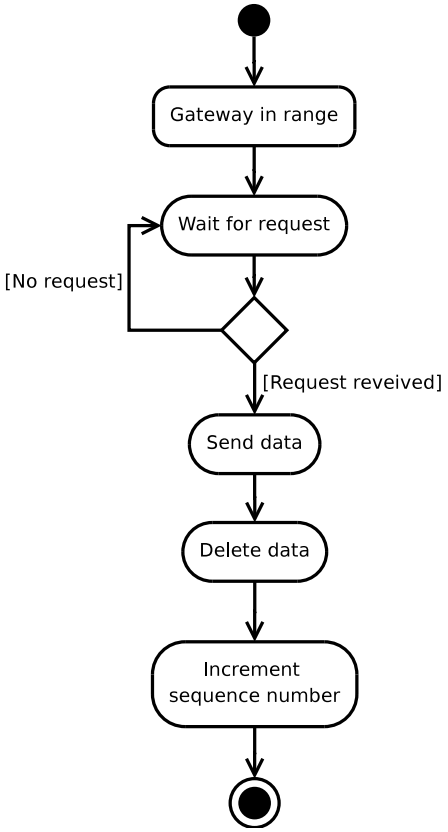

Figure 6.4: *The Figure shows the communication flow of a mote when a gateway enters the cluster*

6.3 Definition of internal tasks

From those activity diagram described earlier, following task are stated:

- Request from gateway
- Data from mote
- Collect data
- Encode data
- Distribute data
- Online/offline

The gateway will perform the following tasks:

- **Broadcast gateway presence:** When the gateway is present it will broadcast a message to the motes announcing its presence.
- **Decode data:**

Those tasks are categorized so those who share the same interest are sat together in a sub task, this is done as follows:

- Receiver
  - Request from gateway
  - Data from mote
- Measure
  - Collect data
  - Encode data
  - Distribute data
- State decision
  - Online
  - Offline
- Gateway
  - Broadcast presence
  - Decode data

6.3.1 Functions

Meassure

Collect data

The purpose of this function is to measures environmental conditions. After the measurement, the collect data function calls a sub function in order to store the measured text string in to buffer.

- Input:
- Output: Text string
- Subfunctions: Write to buffer
- Parent fuctions:

Encode data

This function reads the measured data from the buffer and encodes it regarding to Reed-Solomon scheme.

- Input: Measurement data, which is a text string
- Output: The data is split into n number of text string
- Subfunctions: GF-lib
- Parent fuctions:

GF-Lib

This function is a part of the Reed-Solomon coding scheme.

- Input: Text string
- Output: n packet
- Subfunctions:
- Parent fuctions: Encode data.

Distribute data

The encoded data is distributed into n packets.

- Input: Encoded data, which is a text string.
- Output:
- Subfunctions: Add header, send packets and write to buffer.

- Parent fuctions:

Add header
This function adds a header to on each n text string and creates a packet.

- Input: n text string
- Output: n packets
- Subfunctions:
- Parent fuctions: Distribute data

Send packets
This function handles the packets send functionality, that is, packets are send to online motes on the network. This is done until there are no packets left.

- Input: n packets and n mote
- Output: Boolean, where 0 for a success transmission and 1 for error
- Subfunctions:
- Parent fuctions: Distribute data

Write to buffer
Each incoming data e.g. from: measurement, packets from motes and message from gateway is stored in memory.

- Input: text string from measurement, packets from motes and message from gateway
- Output:
- Subfunctions:
- Parent fuctions: Distribute data

State decision

Calculate probability
This function will calculate the online probability based on the actual number of devices.

- Input: Number of motes n and number of checksums m
- Output: Probability p
- Subfunctions:

- Parent functions:

Make decision
This fuction will make a decision whether to be on/offline by generating a random number and comparing this with the probability.

- Input: Probability p
- Output: Boolean value of 0 and 1 to indicate on/offline decision
- Subfunctions: Go online and Go offline
- Parent functions:

Go onlone
This function will switch the mote's status to online for a specified amount of time

- Input: Time for a period t
- Output:
- Subfunctions:
- Parent functions: Make decision

Go offline
This function will switch the mote's status to offline for a specified amount of time

- Input: Time for a period t
- Output:
- Subfunctions:
- Parent functions: Make decision

Receive

Read input
This function will read the input and write it to an incoming buffer

- Input:
- Output:
- Subfunctions: Process input
- Parent fuctions:

Process input
This function will process the input based on the sender address in the MAC header of the packet

- Input: Received input header
- Output: Boolean to indicate whether the input was from a mote or the gateway
- Subfunctions: Write to buffer
- Parent fuctions: Read input

Write to buffer
This function will store the packet in the buffer if it was received from a mote

- Input: Received packet
- Output:
- Subfunctions:
- Parent fuctions: Process input

Gateway

Broadcast request
This function will broadcast a specified number of requests to the motes in the network. The online motes will respond on this request.

- Input: Number of times the request should be made
- Output: Addresses of online motes
- Subfunctions: Request data
- Parent fuctions:

Request data
This function will request a mode bu it's address to send the packet it contains.

- Input: Adress of mote
- Output: Packet
- Subfunctions: Write to buffer
- Parent fuctions: Broadcast request

Write to buffer
This function will store incoming packets in a buffer.

- Input: Received packet
- Output:
- Subfunctions:
- Parent fuctions: Request data

Decode data
This function will decode the message if enough packets exist in the buffer

- Input: Packets from buffer
- Output: Text string
- Subfunctions: GFLib
- Parent fuctions:

6.4 Network interfaces

When a mote have packets to distribute it will try to contact the other motes one by one. This is done by sending a request to a mote asking for the status, if the receiving mote is online it will respond by sending an alive packet otherwise the sending mote will get a timeout and try the next mote. When the sending mote receives the alive packet it will send a payload packet, which the receiver will respond by sending an acknowledgement. An illustration of this can be seen in Figure 6.5.

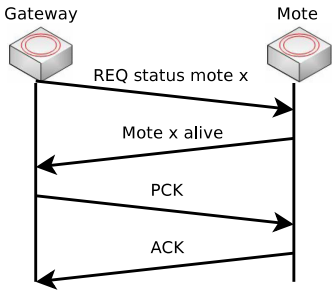

Figure 6.5: *The Figure shows the communication flow between two motes*

When the gateway appears it will request status of the motes and those that are online will respond by sending an alive packet. Then the gateway will request for packets from the first

online mote and the mote will send a packet which must be acknowledged by the gateway. When the last packet is sent the gateway will send a done packet and start requesting packets from the next online mote. This can be seen in Figure 6.6.

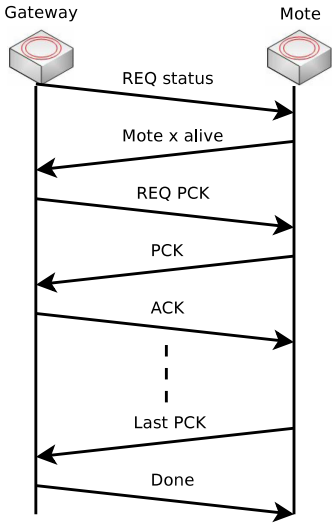

Figure 6.6: *This Figure shows the communicationflow between the gateway and a mote*

6.5 Header design

The MAC header from [1] will be used to provide control on the MAC layer. The two reserved bits in the MAC header will be used for session number indicating the meassage number from 0-4.

The payload header will be 2 bytes where:

5 bits are used to indicate whether a packet is data or checksum and its number. The first bit will be 0 if data and 1 if checksum. The rest 4 bits will provide a umber between 1 and 16. An example of this is data packet 2 as 00010.

The next 5 bits will be used to provide information about the number of data packets which can be between 1-32. The same amount of bits will be used to show the number of checksum packets also between 1-32.

The last bit is reserved for future purposes.

27 bytes is left for payload.

The header can be seen in Figure 6.7.

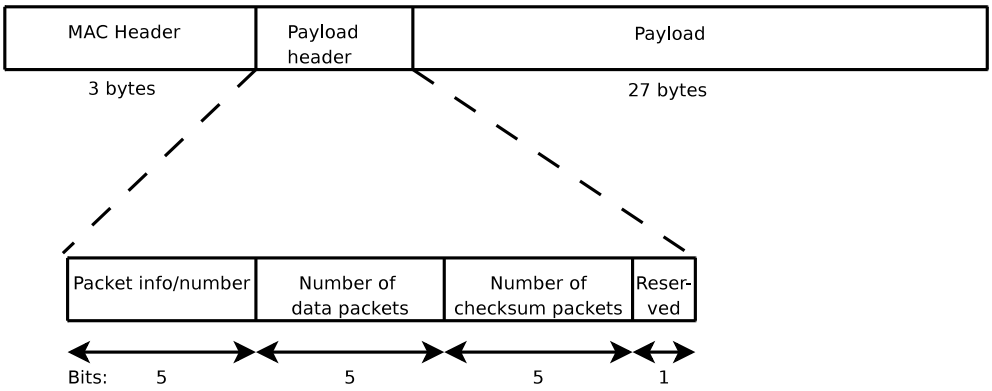

Figure 6.7: *The Figure shows the payload header*

Chapter 7

Test

7.1 Simulation test

To varify the probabilistic system model in Section 4.2, a simulation of the network has been implemented in Java. This Java application is design to simulate the entire system with both motes and gateway. Each mote and the gateway is implemented as a seperate thread which allows each each device to operate independently. It is possible to change the following parameters:

- Total number of motes
- Number of checksum motes
- Online probability
- Period time for the mote

The simulation is designed to be scalable and support an arbitrary number of motes. To ensure a correct and realistic simulation it is necessary that the motes are independent regarding state shift. This means that each mote runs in a separate thread, which means the simulation can run with up to approximately 50 motes depending on computation power. Also the gateway must be independent of the motes thus it is also a thread. Each mote must be able to distribute to all other motes and stay online until the distribution is complete. This means that when a mote is asked to receive it must also be able to inform the distributing mote about the present online/offline state. Furthermore each mote must be able to decide upon the online/offline state independently from the previous state according to the online probability. The gateway must be able to request data from each mote and calculate if enough motes are online for the message to be reconstructed. A screenshot of this visualization is shown in Figure 7.1

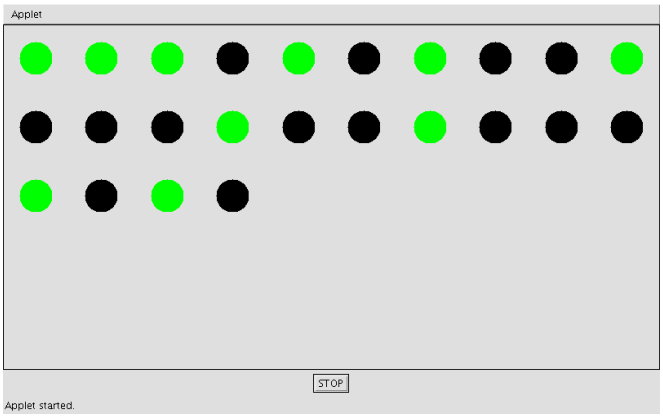

Figure 7.1: *Screen shot of the simulation application in case of 48 motes. The black circles are offline motes and the grey are online motes*

For data processing the gateway thread creates an output for each appearance saying how many motes is offline at this time instance, and if the reconstruction of messages was successful or not. The output can be seen in Listing 7.1

Listing 7.1: *Output from the gateway thread in case of RS(3,1) and online probability 0.5*

```
0 2 % Session 1 Success! 2 was online
1 1 % Session 2 Failure! 1 was online
1 0 % Session 3 Failure! 0 was online
0 2 % Session 4 Success! 2 was online
1 0 % Session 5 Failure! 0 was online
1 0 % Session 6 Failure! 0 was online
0 3 % Session 7 Success! 3 was online
1 1 % Session 8 Failure! 1 was online
0 2 % Session 9 Success! 2 was online
```

7.1.1 Test cases

The simulation is testing the part of the system model regarding probability for reconstruction of the messages given a successful distribution:

$$p_r = \sum_{k=n-m}^{n} \binom{n}{k} p_{on}^k (1-p_{on})^{n-k}$$

Each of the following cases has been tested with $p_{on} = \{0.5, 0.7, 0.8, 0.9\}$ and 1000 samples:

- RS(3,1)
- RS(3,2)
- RS(21,7)
- RS(21,14)
- RS(48,16)
- RS(48,32)

Figure 7.2 to 7.7 show the results of the tests. The line is the system model and the circles are the test results. The figures shows that the probabilistic model is correct becasue the test results are very close to the teoretical model.

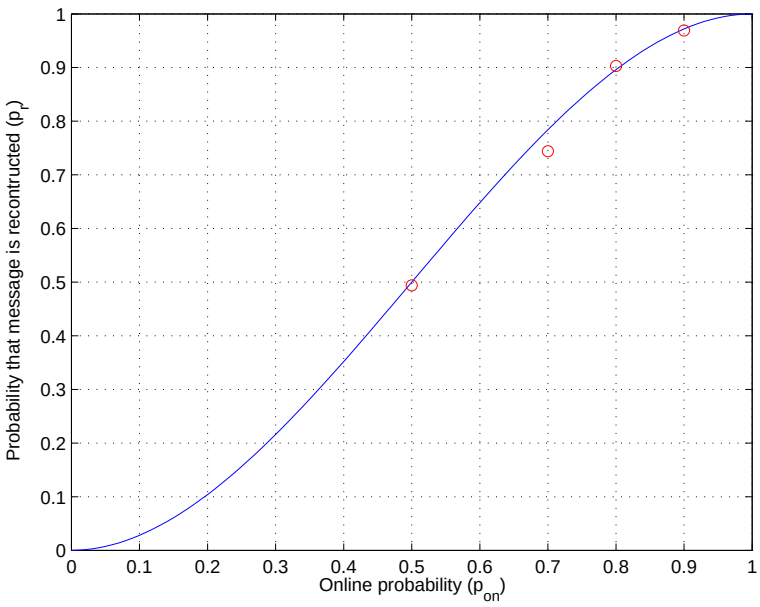

Figure 7.2: *Graph showing the model (solid line) and test results (circles) for RS(3,1)*

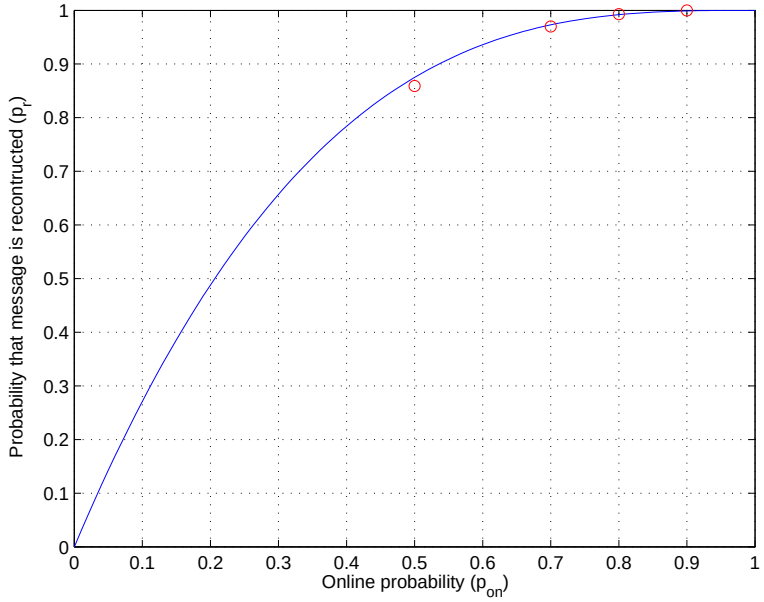

Figure 7.3: *Graph showing the model (solid line) and test results (circles) for RS(3,2)*

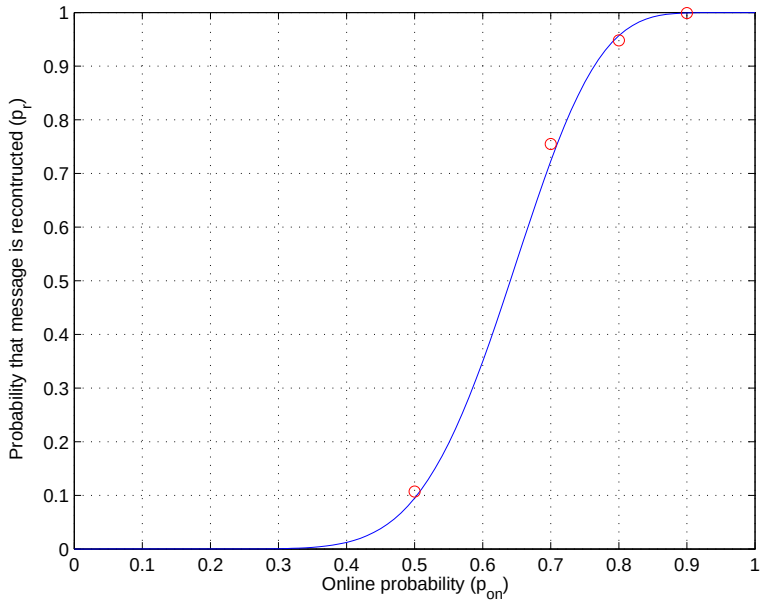

Figure 7.4: *Graph showing the model (solid line) and test results (circles) for RS(21,7)*

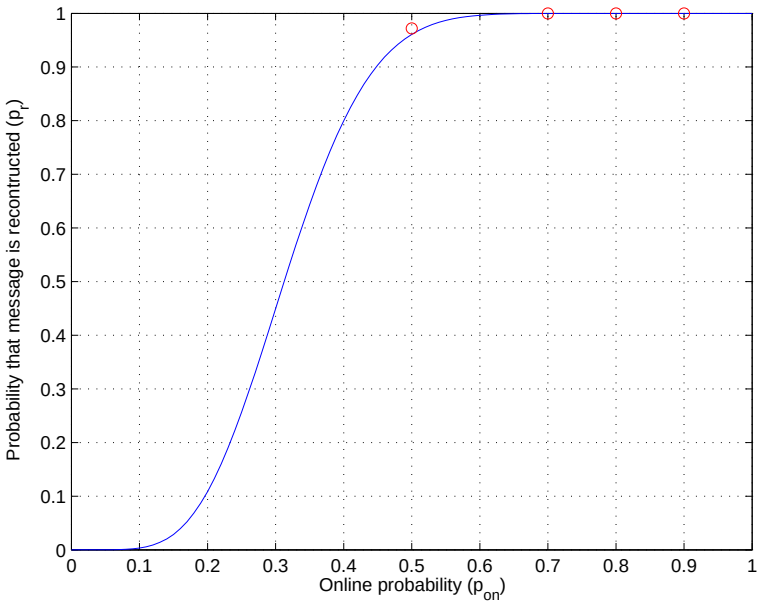

Figure 7.5: *Graph showing the model (solid line) and test results (circles) for RS(21,14)*

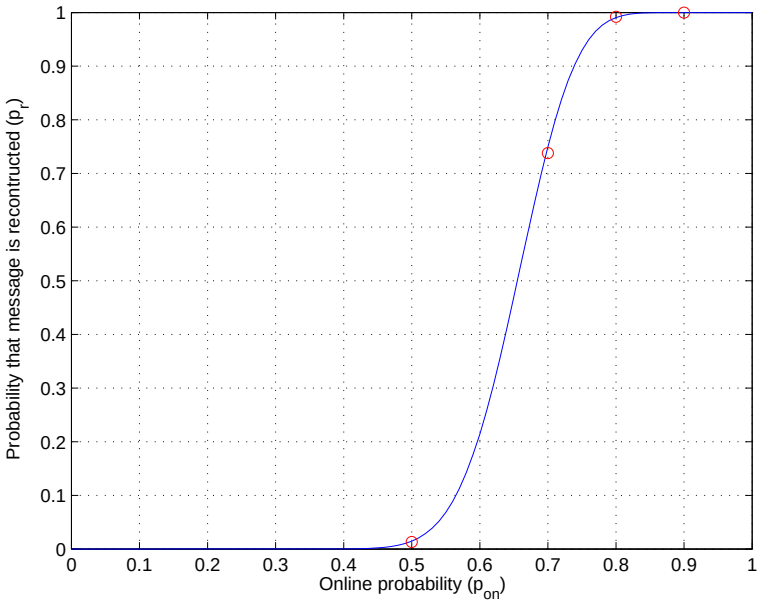

Figure 7.6: *Graph showing the model (solid line) and test results (circles) for RS(48,16)*

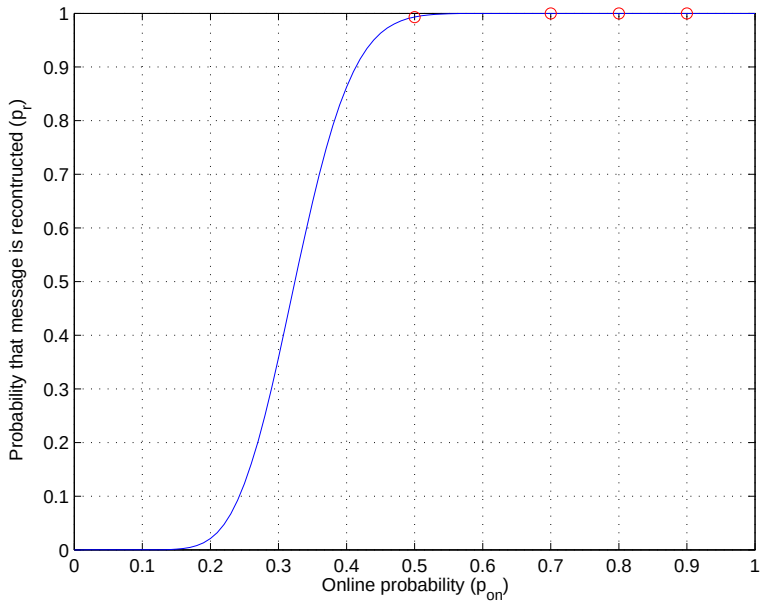

Figure 7.7: *Graph showing the model (solid line) and test results (circles) for RS(48,32)*

7.2 Test of prototype

In order to test the designed distributed model, a prototype has been implemented using sensor boards. The sensor board prototype consist of 3 motes and a gateway, where each mote is implemented such they follows the routine described in Section 4.1.

Test purpose

The Purpose of the test is to show the probability of recovering the messages, using the Xor scheme with one checksum bit, on the gateway.

The prototype will be tested in three cases (50%, 80% and 90% online probability) where:

- Period time of the motes is 5 seconds
- Gateway will be present each 15 seconds and will be present 200 times
- The test time is 50 minutes (15 seconds * 200)
- The online probability of each mote is 50% in case 1, 80% in case 2 and 90% in case 3

Each mote and the gateway is during the test generating a log file, the motes log each time they enters a new state and the gateway logs whether the recovery attempt is a success or failure. The generated log files are then analyzed in MATLAB by generating a stair diagram to show which states the motes enters during the test. The following shows those states:

Online probability [%]	Reliability [%]
50	42,5
80	60,5
90	79,5

Table 7.1: *The results from the prototype tests*

- State 1: Measure
- State 2: Gateway presence
- State 3: Not implemented (Message from motes)
- State 4: Decision (online/offline)
- State 5: Distribute data
- State 6: Online states
- State 7: Offline states

The results generated from the log file of the gateway can be seen on table 7.6.

Expectations
It is expected that the probability of recovering the messages is lower than the theoretically calculated values due to external factors as the MAC protocol, the physical medium due to noise and disturbance etc.

7.2.1 Results

Stair case diagram

Table 7.2 shows how frequent a single mote is on a state during 50 min test duration, for each test case.

Test 50%

- 1xPeriod: 5sec.
- Gateway presence: Once per 15 sec.

The following diagram is plotted by taking some samples from the log file, in order to see diagram over 200 samples see Appendix A. The log files for each mote in those 3 cases can be found on the included CD.

Online probability [50%]			
	Mote 1	Mote 2	Mote 3
Measure	103	114	102
Message from GW	102	113	101
Decision	708	721	756
Distribution	445	455	481
Online	334	356	384
Offline	374	356	372
Samples (50min)	4133	4213	4393

Table 7.2: *Results from 50% online probability test. From this it can be seen that each mote is online approximately 50% as it meant to be.*

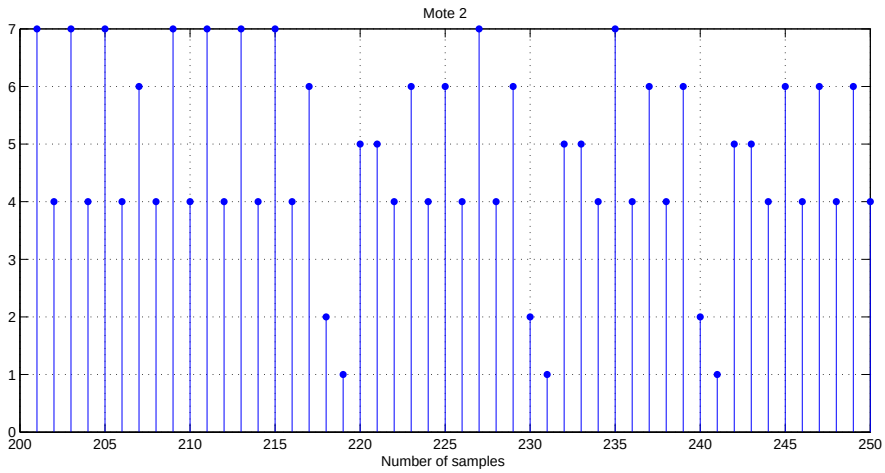

Figure 7.8: *The figure shows state routine of mote 2. Each sample on the x-axes indicates a state chance and y-axes shows which state it is on. From this figure it can be seen that the gateway have appeared 3 times which is approximately 45 sec of the 50 min test.*

Test 80%

- 1xPeriod: 5sec.
- Gateway presence: Once per 15 sec.

Online probability [80%]			
	Mote 1	Mote 2	Mote 3
Measure	148	166	160
Message from GW	148	166	160
Decision	889	1098	870
Distribution	915	1474	897
Online	702	904	697
Offline	187	194	173
Samples (50min)	5979	8005	5915

Table 7.3: *Results from 80% online probability test.*

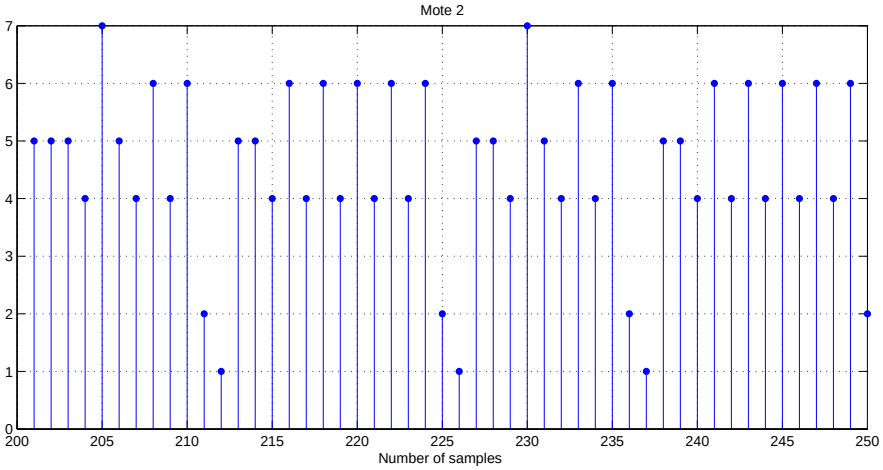

Figure 7.9: *The figure shows state routine of mote 2.*

Test 90%

- 1xPeriod: 5sec.
- Gateway presence: Once per 15 sec.

Online probability [90%]			
	Mote 1	Mote 2	Mote 3
Measure	177	187	181
Message from GW	176	183	181
Decision	1370	1607	859
Distribution	2246	2975	707
Online	1229	1439	758
Offline	141	168	101
Samples (50min)	10679	13119	5575

Table 7.4: *Results from 90% online probability test.*

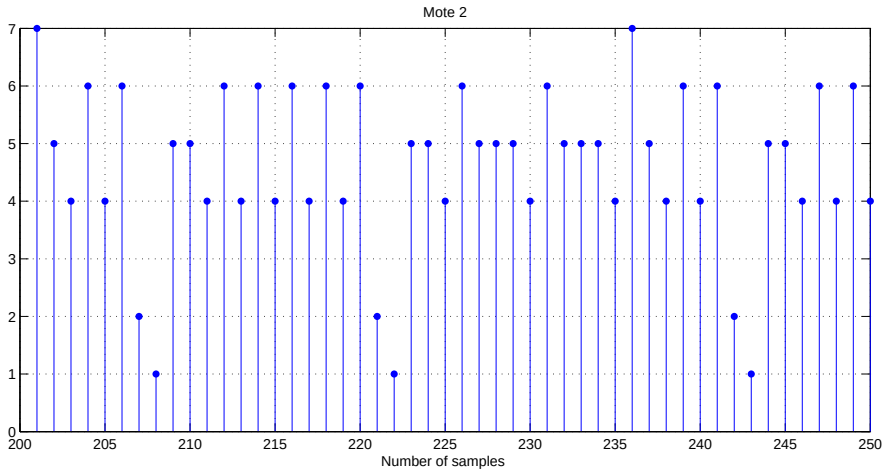

Figure 7.10: *The figure shows state routine of mote 2.*

The Figures above shows that the listed states are entered as well as the measurement and distribution phase which is described in Section 4.1. The stair case diagrams for each test case can be seen in Appendix A.

7.3 Test conclusion

The simulation and prototype have been tested. The simulation is used as an optimal implementation of the model where it is possible to have many samples where the MAC protocol is considered to be error proof. These tests can therefore be used to verify the probabilistic model. The tests from the prototype are used to decide how well the model reflects a real life scenario.

RS / p_{on}	**(3,1)**	**(3,2)**	**(21,7)**	**(21,14)**	**(48,16)**	**(48,32)**
0.5	0.494	0.859	0.107	0.972	0.013	0.993
0.7	0.774	0.97	0.775	1	0.738	1
0.8	0.903	0.993	0.948	1	0.992	1
0.9	0.969	1	0.999	1	1	1

Table 7.5: *The results from the simulation tests showing the reliability at four different online probabilities for six RS schemes.*

One sample in these tests is defined as the result from the gateway in one cycle. In Fig. 4.1 one cycle is shown. The result from the gateway is an answer of whether the reconstruction of the message was possible or not.

It is assumed that each mote in the network successfully have distributed their messages before the gateway arrives.

7.3.1 Simulation

Each of the following cases has been tested with online probability = {0.5, 0.7, 0.8, 0.9} :

- RS(3,1)
- RS(3,2)
- RS(21,7)
- RS(21,14)
- RS(48,16)
- RS(48,32)

In each case 1000 samples are collected.

The conducted results can be seen in Table 7.5.

7.3.2 Prototype

Three different test cases with distinct online probabilities: 0.5, 0.8 and 0.9 are tested. In each test case 200 samples are collected. Because only 4 devices are available the test case will be the RS(3,1).

The conducted results from these cases showed that for an online probability of 0.5, the gateway was able to reconstruct 85 times out of 200 runs that is a 42.5% reliability, and for 0.8 online

Online probability	Reliability [%]
0.50	42.5
0.80	60.5
0.90	79.5

Table 7.6: *The results from the prototype tests showing the reliability at three different online probabilities*

Online probability	Probabilistic model [%]	Simulation [%]	Prototype [%]
0.50	50	49.4	42.5
0.80	90	90.3	60.5
0.90	97.5	96.9	79.5

Table 7.7: *The results from the tests, where the reliability for probabilistic model, simulation and prototype is given for three distinct online probabilities.*

probability gave 60.5% reliability and finally a reliability of 79.5% for a online probability of 0.9. The result from this test can be seen in Table 7.6.

7.3.3 Result conclusion

The results from the simulation and the prototype has been compared with the probabilistic model. Only the test case with RS(3,1) was done for both tests, therefore these are the test results which will be compared. The results can be seen in Table 7.7.

The results from Table 7.7 is shown in the graph on Fig. 7.11. From this graph it is clear that the simulation test verifies the probabilistic model and the results from the prototype test is lower. The dashed line on the graph is the model where there is non-cooperative and it can be seen that the probabilistic model performs better than the non-cooperative method.

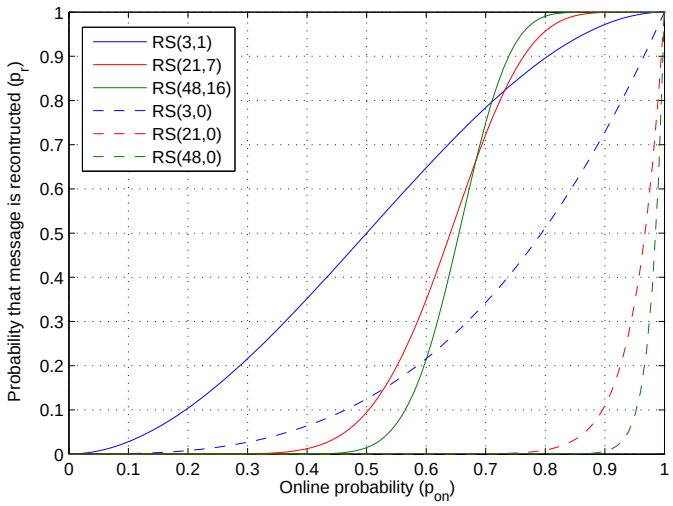

Figure 7.11: *The system reliability as a function of online probability for Reed-Solomon coding with two data devices and one redundancy device in comparison to no distribution*

Chapter 8

Conclusion

The objective of this project was to propose a reliable solution that enables distribution of data to a gateway in a partially wireless connected sensor network, in comparison to a non-cooperative system where no distribution is performed. To fulfill this, the Reed-Solomon coding scheme has been selected to enable the gateway to recover data with a high reliability.

A probabilistic system model has been derived to find a relation between the online probability of each mote and the reliability of the system. This system model has been validated through a simulation and a prototype has been developed to test the model in a real life scenario. The result shows that the proposed solution has an enhanced performance compared to the non-cooperative case.

8.1 Discussion

The results of the tests have shown that the methods have proven to be suitable for the chosen scenario. The results can be seen in Figure 7.11. The results from the simulation verifies the probabilistic system model and matches the results, there is only a small deviation ($\pm 0.6\%$). This deviation is most likely due to the random number generator in a PC which is not truly random, but only pseudo random [2]. The results from the prototype test shows a deviation from the system model and simulation, which must be considered acceptable since the prototype is just a real life proof of the model. The deviation is mainly due to a MAC protocol not suited for the scenario and a shared wireless medium. Even though collision avoidance mechanisms are implemented, it still needs fine tuning.

Figure 7.11 and 8.1 shows that using the cooperative method using Reed-Solomon is better than the non-cooperative case. For example in Reed-Solomon a reliability at 50 % can be guaranteed with only 0.50 online probability where no distribution needs an online probability at approximately 0.80 to ensure the same reliability. A online probability in both methods at

0.70 gives a reliability in Reed-Solomon at 78.5 % and in no distribution at 35 %.

From Fig. 8.1 it can, as well, be seen that the system reliability is higher than the online probability of each mote, for large online probability. Furthermore it is seen that the Reed-Solomon coding scheme is able to maintain the same level of reliability, when the online probability is decreased and the number of motes in the system is increased.

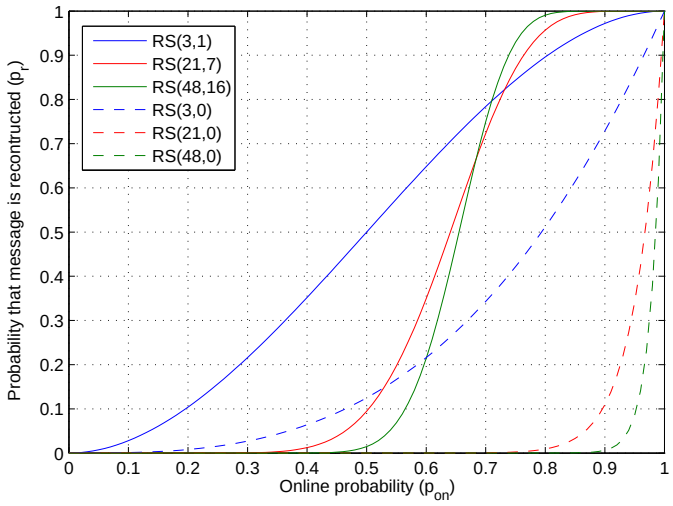

Figure 8.1: *Each solid line represents Reed-Solomon coding with a ratio of 3:1. RS(3,1) as an example shows Reed-Solomon coding with three devices, where one acts as redundancy device. The corresponding dashed lines show no distribution (ND) models.*

8.2 Future perspectives

The models proposed in this paper are static in the sense of the distribution phase, because all motes need to be online before the distribution can finish. In this section ideas to improve the model are proposed to make the model act dynamical in the distribution phase.

Distribution model

As stated earlier in this paper, the distribution is performed to all motes in the cluster. This results in extended online periods for each mote because it must wait for all motes to become online. For a small and predefined system this approach is suitable, but for a large and uncontrollable system it might take a long time to distribute if it is possible at all because a failure in one mote can result in a single point of failure. Instead of having such a static system where each mote is dependent on the other motes to be able to distribute its data a more dynamical approach could be introduced.

Search for available motes before distribution

Instead of waiting for motes to become online the distributing mote can perform a neighbor

discovery to search for available motes and then distribute to those who are available. This will ensure that the distribution is performed much faster than by waiting for others to be online, but on the other hand the number of motes who will have a part of the message will not be as high as in the static approach. For a large scale network it can be assumed that a high percentage related to the online probability will have a part of the message. The implementation will also be better for a scalable network because the motes do not need to know how many neighbors it has in the cluster.

Multi hop distribution

The proposed model in this project is distributing the packets by making one-hop connections in the network. Another approach of distributing the packets could be to introduce multi-hop connections to let other motes in the system route the packets to their destination e.g. by using the epidemic algorithm [3]. By doing this, motes not in range of each other might still be able to communicate because some motes in between them can be used to convey the packets.

Poisson distribution

The binomial distribution is used to calculate the probabilistic model but it is limited to small n and m due to computation precision. If the model should show how the reliability is for high m and n the Poisson distribution must be used.

Bibliography

[1] Anders Grauballe and Mikkel Gade Jensen. Mac protocol for wireless sensor networks. University project report, 2007.

[2] Mads Haahr. *Introduction to Randomness and Random Numbers*. `http://www.random.org/randomness/`, 2007.

[3] Tom Daniel Hollerung. *Epidemic Algorithm*. `http://wwwcs.uni-paderborn.de/cs/ag-madh/WWW/Teaching/2004SS/AlgInternet`, 2004.

[4] Robert H. MarelosZaragoza. *Art of Error Correction Coding*. John Wiley and Sons Inc., 2th edition, 2006. ISBN: 0-470-01558-2.

[5] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[6] J. S. Plank. Erasure codes for storage applications. Tutorial Slides, presented at *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, `http://www.cs.utk.edu/~plank/plank/papers/FAST-2005.html`, 2005.

[7] Sheldon M. Ross. *Introduction to Probability and Statistics for Engineers and Scientists*. Elsevier Academic Press, third edition, 2004. ISBN: 0-12-598057-4.

[8] Andrew S. Tanenbaum. *Structured Computer Organization*. Pearson Prentice Hall, 5th edition, 2006. ISBN: 0-13-148521-0.

Appendix A

Results

A.0.1 Prototype test 50% online

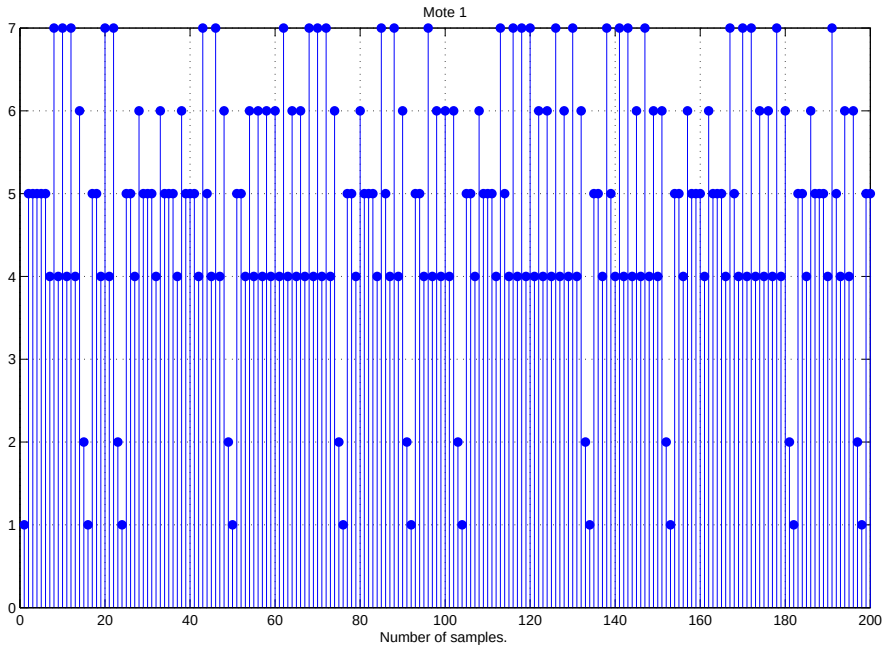

Figure A.1: *The figure shows state routine of mote 1.*

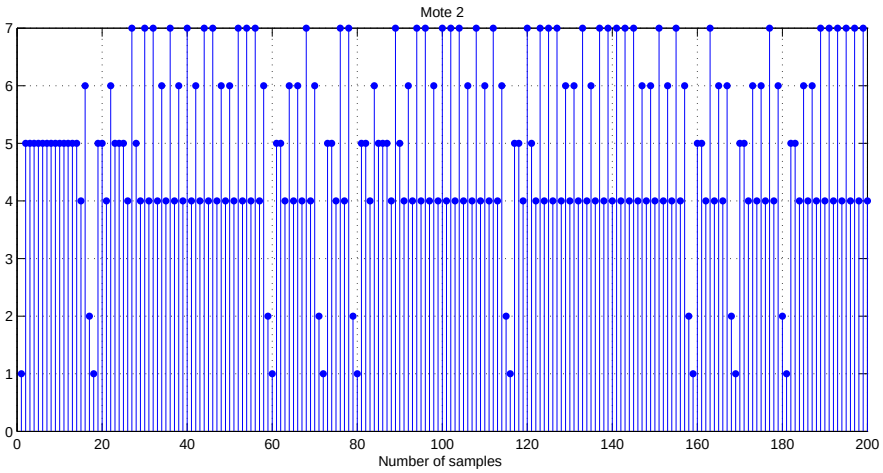

Figure A.2: *The figure shows state routine of mote 2.*

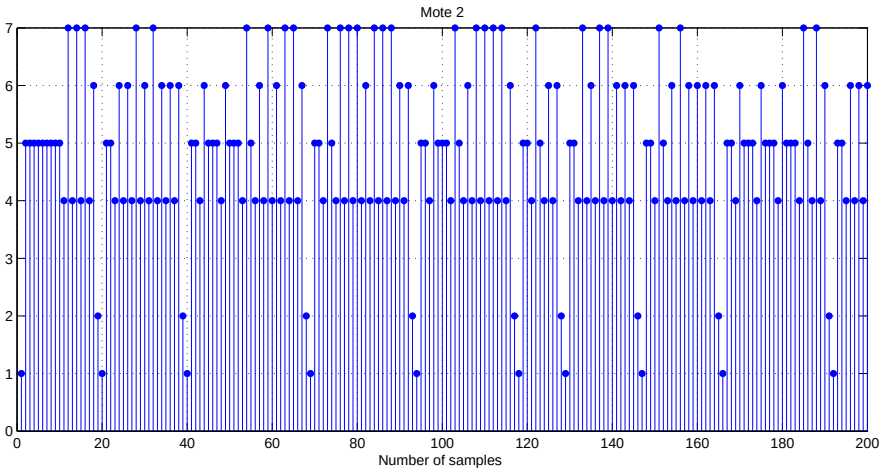

Figure A.3: *The figure shows state routine of mote 3.*

A.0.2 Prototype test 80% online

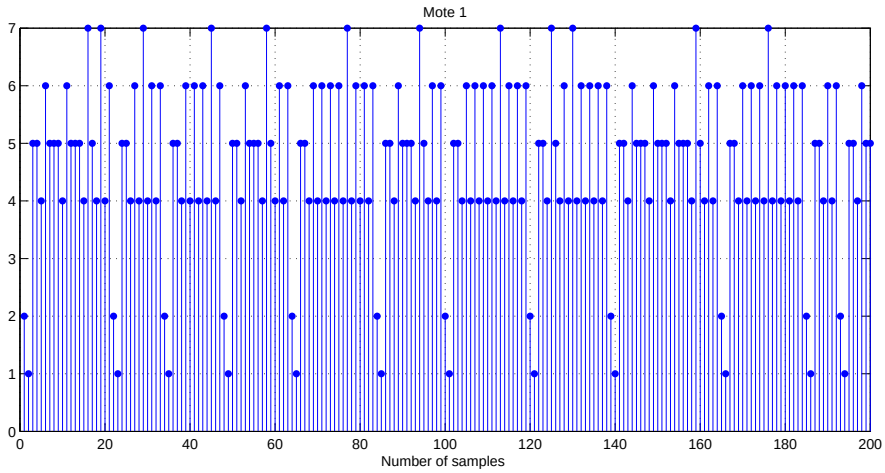

Figure A.4: *The figure shows state routine of mote 1.*

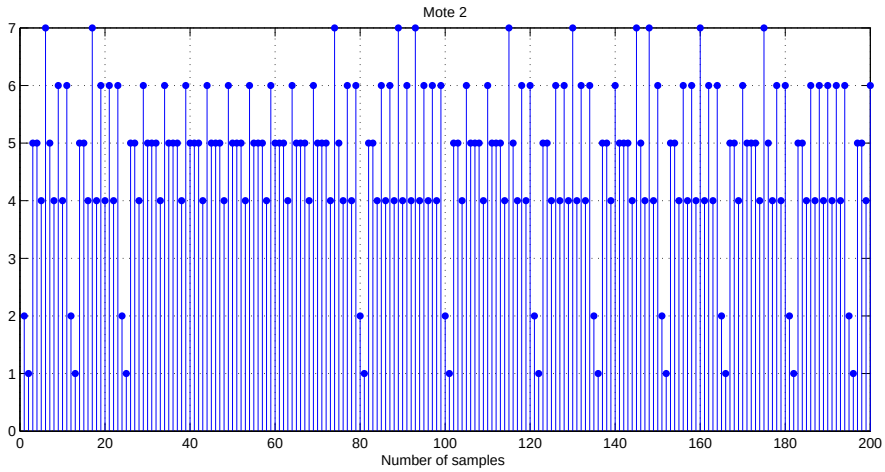

Figure A.5: *The figure shows state routine of mote 2.*

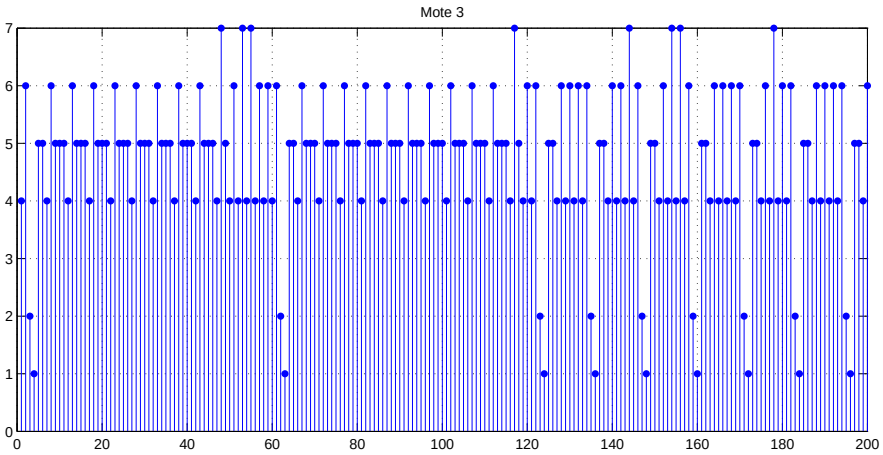

Figure A.6: *The figure shows state routine of mote 3.*

A.0.3 Prototype test 90% online

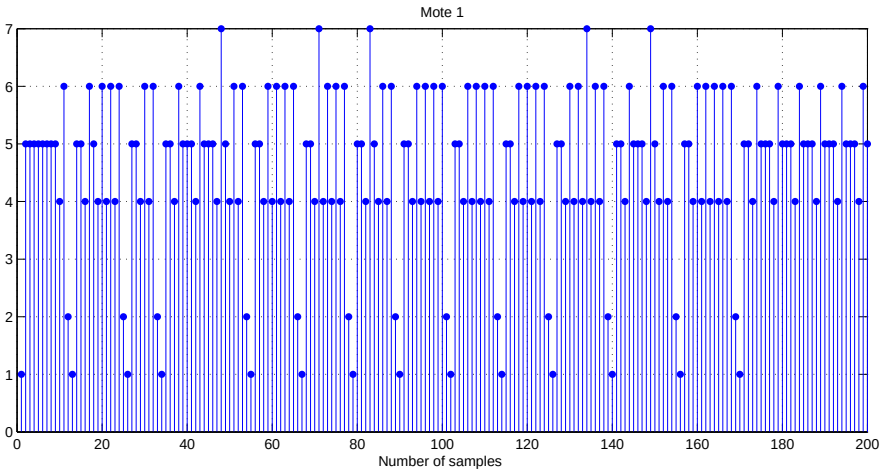

Figure A.7: *The figure shows state routine of mote 1.*

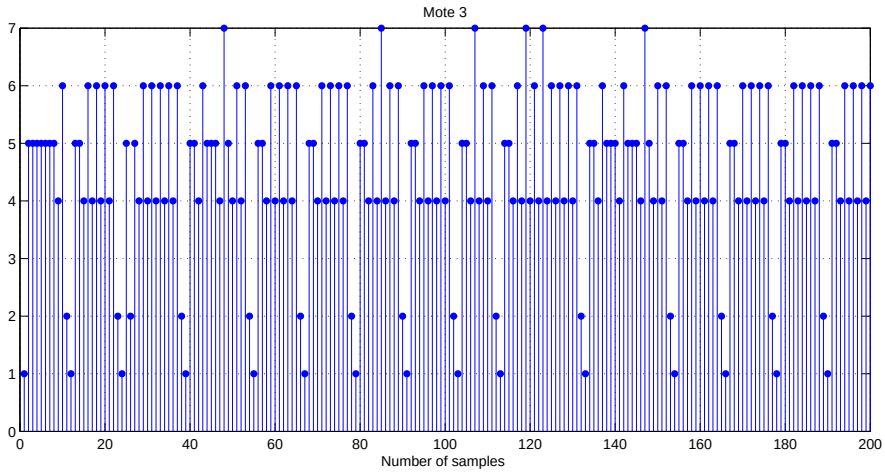

Figure A.8: *The figure shows state routine of mote 2.*

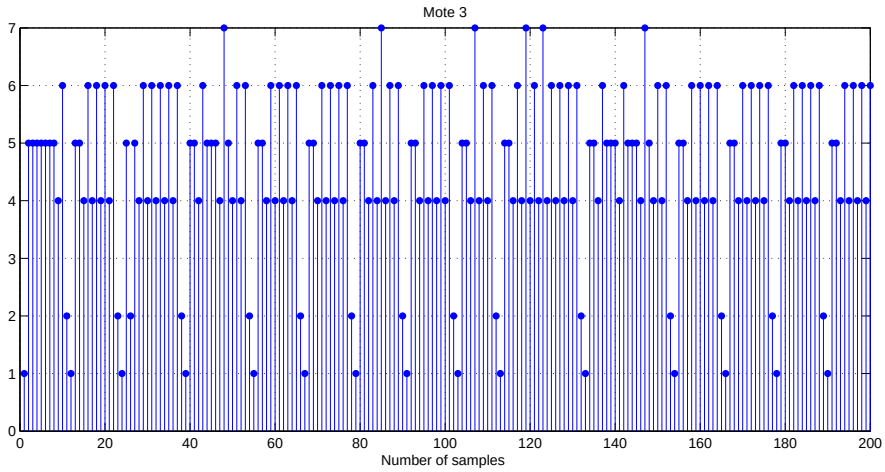

Figure A.9: *The figure shows state routine of mote 3.*

Appendix B

MAC Protocol

When working with, in particular wireless networks it is important to have a working MAC protocol. This project has its focus on the application layer, and must rely on an existing MAC protocol. Implementing the project prototype design in a simulated environment does not raise any critical requirements for a MAC protocol as the environment is controlled and no channel error can occur. The physical sensor platform of this project does already have an existing MAC protocol implemented as the result of a 6th semester project at AAU[1].

This MAC protocol has been developed to support a multihop connection in a line topology network. It features carrier sensing (CSMA), a dynamic medium reservation scheme (RTS and CTS) known from IEEE 802.11, acknowledgment (ACK) and a one bit sequence number to prevent packet duplicates. The total packet size of the platform is 32 bytes, 3 bytes is used for MAC header which leaves 29 bytes for payload and higher layer protocols. The packet including MAC header bits can be seen in Figure B.1. Notice the two reserved bits for adding extra features like broadcast (i.e. no ACK reply from receivers) which can be usefull in this project. The field Packet Type of 4 bits is also not fully used as only 4 types is currently supported: RTS, CTS, Payload and ACK. [1]

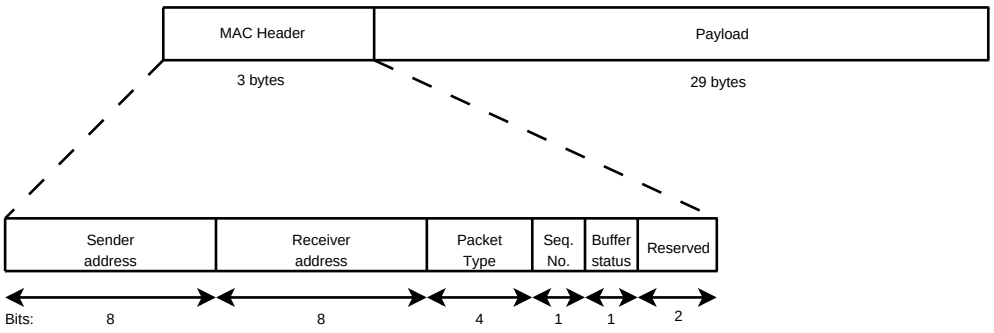

Figure B.1: *The figure shows a packet with the MAC header added. 2 bits are reserved for modification and add-ons.[1]*

A communication diagram of the reservation scheme can be seen in Figure B.2. The packets RTS, CTS and ACK are all 3 bytes long and a payload packet (Pck) is 32 bytes in total.

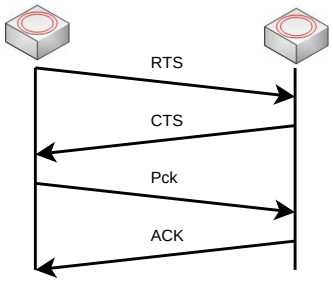

Figure B.2: *Communication flow between two motes in the network [1]*