TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

**Faculty for Electrical and Computer Engineering** Deutsche Telekom Chair of Communication Networks

# Implementation of Network Coding in XDK Sensors

**Borja Macho Pisano**
Born on: 2nd October 1996 in Tanos, Spain
Matriculation number: 4754140

## Student Thesis

Supervisor
**M.Sc. Maroua Taghouti**

Submitted on: 24th July 2018

## Statement of authorship

I hereby certify that I have authored this Student Thesis entitled *Implementation of Network Coding in XDK Sensors* independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

Dresden, 24th July 2018

Borja Macho Pisano

## Abstract

The data collection has become more important for the industrial world in the last years. The data collected can be used for a large variety of purposes, as detecting fires and intruders, controlling machines' behavior or simply controlling the environmental regulation of the heating or humidifiers. This work aims to implement and test the first Wireless Sensor Network (WSN) using XDK© small wireless sensors created by Bosch GmbH, that have been used for gathering environmental data and exploiting the benefits of network coding. We introduce a simple routing protocol that allows the nodes to determine the useful packets, enabling them to collaborate with each other to route their measurements to the sink. Thanks to the network coding implementation, we will get some of the advantages, such as high reliability and ordered delivery, using the UDP protocol over wireless links even when the channel conditions are adverse. The proposed network coding scheme will implemented the finite field GF(2), which does not require high computational capabilities, as it only needs to use the XOR operation for encoding and decoding the packets. Moreover, the use of network coding takes advantage of the broadcast nature of the wireless medium, reduces the packet transmissions, has the potential of increasing the traffic balancing, therefore the nodes' lifetime, which makes it perfect for the WSN world. Our implementation results will prove that the XDK sensors are capable of using network coding with small fields and small generation sizes. Finally we will present the future lines of work to improve the performance of the test bed, its utility and make its use and future updates more convenient for the users and the developers.

# Contents

# List of Figures

# List of Tables

# Nomenclature

| | |
|---|---|
| $K$ | Total number of packets sent each generation. |
| $N$ | Generation Size. Actual number of information packets each generation. |
| $n$ | Number of redundancy packets sent after the systematic packets. |
| $r$ | Number of randomly coded packets sent each iteration. |
| $R = K - N$ | Redundant coded packets sent each generation. |

# 1. Introduction

The use of wireless sensor networks for data gathering has been increasing significantly in the last years, due to the advantages regarding the easy deployment of the network and the simple maintenance that it requires as they don't need any wires to communicate nor for being powered as they count on a built-in battery. The economical factor is also another reason to use wireless sensors, because the installation and wiring expenses can rise highly the cost of the devices [1]. The increment on the use of these devices has encouraged the publishing of researches regarding protocols to optimize this kind of networks [2] some of them even implementing network coding solutions [3].

Wireless sensors are small devices powered by battery, thus allowing to place them almost everywhere. When talking about wireless sensors they use to be separated in four main components [4], the sensing sub-system, the processing sub-system, the communication sub-system and the power sub-system. The sensing sub-system is the one that provides the node with sensing capabilities, which may differ from an application to another. Additionally they have the capability of communicating wirelessly, thanks to the communication unit, from one sensor to another and with a special node called sink or gateway whose mission is collect all the data measured by the sensors and process it. The processing capabilities and memory capacity are part of the processing unit, they use to be really small in order to make the nodes cheaper as normally they are bought wholesale. The assignment of the power sub-system is to supply energy to the rest of the units and consist mainly in the battery. In this particular case the XDKs are not as cheap as the wireless sensors use to be because they are meant for testing and develop over this kind of networks. They also have wide range of applications and sensors, whereas the wireless sensors use to be application designed, only counting with the resources needed for doing its task. We could say that the increase in the price of these particular sensor is due to the oversize of its features.

Network coding is a promising alternative to the classical store-and-forward technique where the intermediate nodes take an active role in the communication not simply retransmitting the packets as they were received but processing and modifying its content. There are two different groups when talking about network coding, the inter-flow network coding and the intra-flow network coding, depending on the way that they encode the packets, whether if the intermediate nodes recode packets belonging to the same session or not. Network coding can be used as a FEC (Forward Error Correction) code as it sends linear combinations of the packets, thus if one of the sent packets is lost it can be recovered from other of the received combinations of it, the problem with this method is that for decoding it needs at least the same number of packets that created originally the generation, having an impact on latency. The use of network coding can improve the behavior of communication networks in multiple crucial points such as throughput, reliability, energy efficiency and security [5]. Network coding shows its best in wireless environments where it can take advantage of the broadcast nature, allowing more nodes than the destination to take part in the communication procedure.

In order to guide all the measured data to the sink a routing protocol should be used. The most known and widely used would be the couple TCP/IP but as it is known TCP does not work its best in wireless environments [6] as it was designed for wired technologies and many approaches have been presented to improve its performance, some of them using network coding [7]. Moreover, wireless sensor networks use to have lightweight routing protocols. There are also some researches trying to get a reliable transport over wireless links combining random linear network coding with UDP [8]. The network coding technique used during this thesis is more like this last one, as it provides a reliable and ordered deliver of the packets over the UDP transport protocol.

The thesis is structured in five chapters. Chapter I describes the wireless sensor which will be implementing the network, the environmental variables that are going to be measured and the problems found while working with these devices. Chapter II will consist on the explanation of how works the routing protocol that has been implemented in the sensors and the topologies that were used for the results. Chapter III is about the usage of network coding in the network and how the encoders and decoders implemented work. Chapter IV covers the aspects of the practical implementation of the network and the results obtained during the tests. Chapter V sums up the future improvements of this implementation and a brief conclusion about the thesis.

# Part I.

# XDK

# 2. XDK Node

## 2.1. Hardware Platform

The nodes used during this thesis are the XDK sensors [9] (Cross Domain Development Kit) developed by Bosch and meant for the IoT world. Each sensor includes the following components:

- MCU: 32-bit microcontroller ARM Cortex M3 EFM32GG390F1024 (Silicon Labs)
- Communication interface
  - Bluetooth Low Energy
  - Low power IEEE 802.11b/g/n WLAN
- Inertial sensors (9DOF)
  - Accelerometer: BMA280 (Bosch Sensortec)
  - Gyroscope: BMG160 (Bosch Sensortec)
  - Magnetometer: BMM150 (Bosch Sensortec)
  - Inertial measurement unit BMI160 (Bosch Sensortec)
- Environmental sensors
  - Combined humidity / temperature / air pressure sensor: BME280 (Bosch Sensortec)
  - Ambient light: MAX44009 (Maxim Integrated)
  - Microphone for noise detection: AKU340 (Akustica)
- Internal Li-Ion rechargeable battery, 560 mAh capacity
- Integrated antennas
- User interface:
  - 3 programmable status LEDs
  - 2 programmable push-buttons
  - Micro SD card1
  - J-Link debug interface2
  - Interface for extension board

Table copied from [10] page 5. Further information about the sensors can be found in the technical overview PDF [11].

In this thesis we used some of the environmental sensors to measure the data, the WLAN interface to transmit it, the programmable LEDs to show the status of the node and the buttons to interact with them.

## 2.2. User Interface

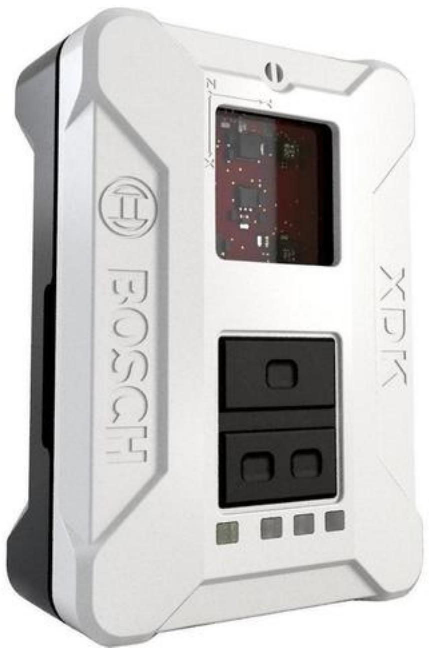


Figure 2.1.: XDK© Sensor by Bosch GmbH

The three LEDs included in the XDK sensor are used to show the status of the sensor while the program is running.

- Yellow LED: Turned on if the sensor is connected to the WiFi network.
- Orange LED: Turned on if the sensor is sending or receiving data.
- Red LED: Turned on if the sensor is active and measuring data.

Table 2.1.: LEDs' meaning during the application run

| LED | ON | OFF |
|---|---|---|
| Yellow | Connected to WiFi network | Disconnected from WiFi network |
| Orange | Sending/Receiving packets | Idle/Measuring |
| Red | Sensor active | Sensor inactive |

By default after starting, the sensor node only tries to connect to the WiFi network set in the configuration, then the yellow LED is the only one that will be turned on if no button is pressed.

Once the first programmable button is activated the red LED will be turned on and the sensor will start measuring the environmental status and sending that data to the sink.

The second programmable button is used to stop the measures in the sensor and thus the send and receive activity. The red LED will be turned off.

## 2.3. Sensors used

Two sensors were employed to measure the environment, the BME280 and the MAX44009. The BME280 is the combined environmental measuring sensor which determines temperature, pressure and humidity. Each of those measures is saved in a 32 bit variable (4 bytes), thus the combined size of them is 12 bytes. Here is a table indicating the accuracy of the sensor [10]:

Table 2.2.: Sensors used and accuracy

| Sensor | Accuracy | Unit |
|---|---|---|
| Pressure | $\pm 10$ | hPa |
| Temperature | $\pm 2$ | K |
| Humidity | $\pm 10$ | %RH |

It's current consumption is really low $3.6\mu A$ if it's measuring every second [10], which can offer long lifetime of the network. The MAX44009 is an illuminance sensor, its output is the millilux measured by the sensor in a 32 bit variable (4 bytes). Its measuring range is from 45 millilux to 188,000,000 millilux. This sensor is also low consuming, with less than $1\mu A$ operating current [10].

Those four environmental measures will make the total data size 16 bytes, which will be the payload of each packet.

## 2.4. Problems

1. Most of the problems found were related to the WLAN interface, such as the incapability of enabling promiscuous mode to allow overhearing, that is the reason why all the packets are sent to the broadcast address of the network, enabling thus all sensors in range to receive the packets even if they were not meant to be delivered to them.

2. The WLAN device only works in "infrastructure mode", not allowing it to work in "ad-hoc mode" which would be the perfect mode for this kind of network as the nodes would be able to communicate directly with each other without the need of an access point.

3. As the sensors need to be connected to the same WiFi network, all of them will be in communication range with the rest, hence the topologies had to be artificially implemented.

4. Related to the debugging of the program, the XDK needs special hardware to be debugged and it can't run instruction per instruction nor gives information about segmentation faults without the mentioned hardware.

5. Incapability of using the KODO libraries as the architecture of the processor is currently not supported by KODO [12]. As a matter of fact, KODO is library available for multiple programming languages for implementing Network Codes and the current fastest Network Coding solution which would have simplified the work during this thesis as the encoder nor decoder shouldn't have been manually programmed being implemented in stead with that library.

# Part II.

# Routing

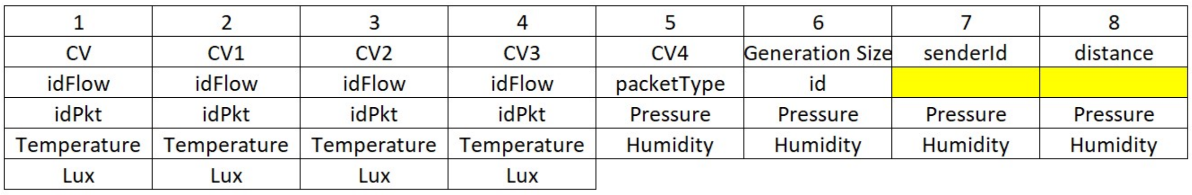
# 3. Routing Protocol

## 3.1. Routing Protocol

For solving the information guiding issue a routing protocol had to be introduced. As no complex metrics were needed, it was designed a simple distance-based routing protocol, whose only metric is the hop count to the sink. Each node knows its own distance to the sink based on the distance that the nodes in its reception range are sharing. The transport protocol used is UDP [13] and all the packets are sent to the IP broadcast address of the network, allowing all the nodes in range to receive all the packets. The nodes will only process the packets that interest them, which will be further explained in below section 3.2.

The packet's components can be seen in the figure 3.1 and they are as follows:

- Header (20 bytes):
  - Coding Vectors: Each coding vector is 1 byte long, as we need 1 bit per packet component (GF2) it enables to XOR up to 8 packets each. 5 coding vectors are used, then the maximum generation size is 40.
  - Generation Size: 1 byte allows up to 255 generation size, but it's limited by the number of coding vectors.
  - Sender Id: The ID of the node sending the packet. 1 byte allows up to 255 nodes.
  - Distance: Distance of the node sending the packet. 1 byte allows up to 255 hops.
  - Id Flow: 4 bytes that identify the block which the packet belong.
  - Packet Type: 1 byte that identify the type of packet: 0- Info, 1- Urgent Info, 2- ACK.
  - Id: The ID of the original node where the packet was created. 1 byte allows up to 255 nodes.
  - 2 bytes unused (Align): Can be used for future upgrades, e.g. adding 2 coding vectors more which will rise the generation size to 56.
  - Id Pkt: 4 bytes that identify the packet into the flow.
- Data (16 bytes): 4 bytes of data each measure.
  - Pressure
  - Temperature
  - Humidity
  - Lux



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| CV | CV1 | CV2 | CV3 | CV4 | Generation Size | senderId | distance |
| idFlow | idFlow | idFlow | idFlow | packetType | id | | |
| idPkt | idPkt | idPkt | idPkt | Pressure | Pressure | Pressure | Pressure |
| Temperature | Temperature | Temperature | Temperature | Humidity | Humidity | Humidity | Humidity |
| Lux | Lux | Lux | Lux | | | | |

Figure 3.1.: Packet components

The header of the packet is really big compared with the data size, which leads to a big overhead $20/16 = 1.25 \Rightarrow 125\% \ Overhead$ more than half of each packet is the header, but this header does not change with the information size so it could code up to the maximum MTU of the network with the same header. For example in Ethernet + UDP/IP the MTU is 1500 bytes, the overhead due to the IP header is 20 bytes, the UDP header is 8 bytes and the Network Coding header is 20 bytes, making a total overhead of 48 bytes. Then the maximum payload that can be sent in each packet is $1500 - 20 - 8 - 20 = 1452$. In this example the total overhead of the packet would be $(20 + 8 + 20)/1452 \approx 0.0331 \Rightarrow 3.31\% \ Overhead$.

One problem appears using that UDP + Broadcast combination, the MAC retransmission is disabled as the destination is not a specific node so the packets are sent only once. That's why a triple ACK is sent, to increase the reception rate and reduce retransmissions.

## 3.2. Routing Algorithm

Each node has its own ID that can be any integer between 2 and 255, the 1 is reserved for differentiating the sink from the sensing nodes. The ID is specified in the configuration parameters when loading the application into the node and also determines the IP address. In the first stages of the network when the nodes have just been turned on, they do not know their distance to the sink, thus they start sending the messages with maximum distance, 255 as the distance is saved in a unsigned char variable (1 byte). The nodes listen to the received messages and keep an updated node list with the nodes listened, their distances and the last time they were seen. The own node's distance is computed as the lowest of the distances of the seen nodes + 1, or 0 if it has a direct link with the sink. If one node is not detected for longer than the 'timeout' threshold it is deleted from the list and the distance of the node keeping the list is updated if needed, creating the possibility of changing the route to the sink through a node that was at a further distance before. As the data is not deleted from the node until other other node at closer distance has decoded it these topology changes can hardly lead to data loses.

If the node deleted from the list was the one with lowest distance to the sink, in other words the route to the sink, the distance will be reduced to the next lowest distance + 1, as the next lowest distance will probably be relying on the current node's distance it will be increased to the updated distance in the next iteration, creating thus a loop (like could happen in RIP, count-to-infinity [14]) that ends once a lower distance to the sink is found or the maximum distance is reached.

Each time a packet is received it is added to a linked list called *RecvBuffer* after adding all the packets to the list they are processed in the following way:

- Each node is only interested in information coming from nodes with greater distance to the sink, if information coming from lower or equal distances is received, it's discarded.
  - If it is coded information the appropriate decoder is selected and the coded packet is added to its packet list buffer *decoder->RecvPkts*.
  - If a decoder is changed by any incoming packets it is marked as "check". After processing all the packets in the *RecvBuffer* all the decoders marked as "check" try to decode their information.

- In the opposite way each node is only interested in ACKs coming from lower or equal distances:
  - ACKs from a lower distance aiming to the node itself mean that the block has been properly decoded in a node closer to the sink so the encoder can start encoding the next block.
  - ACKs from the same distance will clear the decoders of the node, as it means that another node has ended decoding the block before than the node itself so there is no reason to continue decoding that block.

```
if ACK to node i received then
    if This node == i then
        Clear Send Queue;
        Send retries = 0;
    else
        Clear decoder i;
    end
end
if Send Queue is empty then
    while Send Queue < Max Generation Size do
        Add data measured to Send Queue;
    end
    while Send Queue < Max Generation Size do
        Add decoded packets to Send Queue;
    end
    if Send Redundancy != 0 then
        n = Number of redundancy packets to send after systematic;
    end
    r = Number of randomly coded packets to send each iteration;
else
    Send retries ++;
end
if Send retries == 0 then
    Send systematic;
    if Send Redundancy != 0 then
        Send n extra randomly encoded packets;
    end
else
    Send r randomly coded packets;
end
while Recv Buffer not empty do
    Process received packet;
end
if Decoder i has been modified then
    Try to decode i;
    if Decoded i successfully then
        Send block ACK;
        Clear decoder i;
    end
end
```

**Algorithmus 1 :** Routing Protocol

# 4. Network Topologies

For getting the practical results three different topologies were used at different distances: all together: longest distance $\approx 15cm$, in the same room: longest distance $\approx 6m$ and in different rooms: longest distance $\approx 15m$. As all the sensors were in range of each other during the runs the topologies were artificially implemented ignoring selectively some of the nodes as if they were not in the receiving distance.

## 4.1. Star Topology

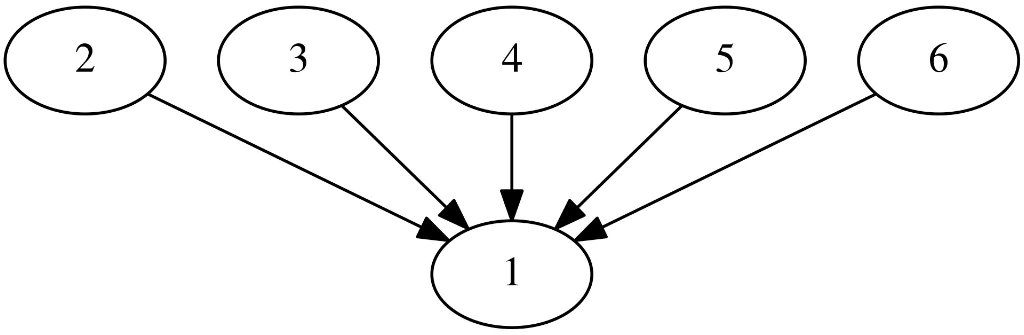


Figure 4.1.: Star Topology, the circle 1 represents the sink, the rest of the circles from 2 to 6 are the wireless nodes

The first and easiest topology is a one-hop 'star' topology, represented in figure 4.1 where all the sensor nodes communicate directly with the sink, which decodes the information sent by all of them. All the nodes in this topology play the role of leaf node, they just encode and send the data measured to the sink.

## 4.2. Single Cluster Head Topology

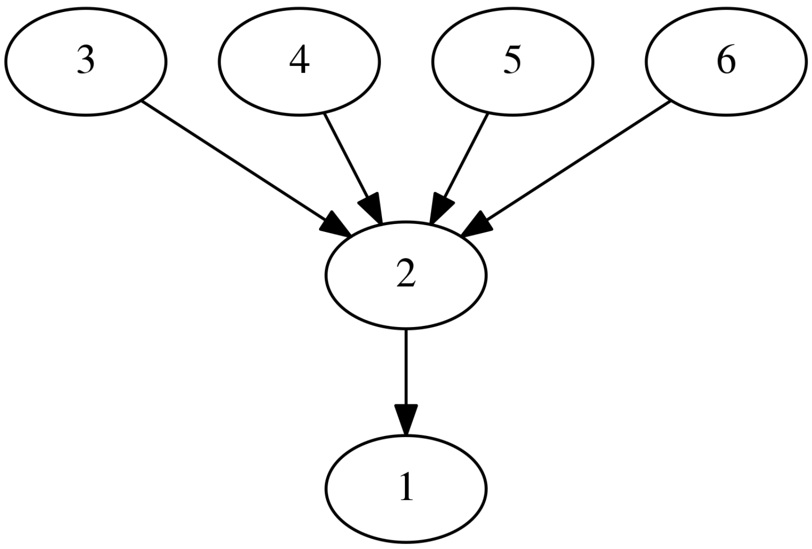


Figure 4.2.: Single Cluster Head Topology, the circle 1 represents the sink, the rest of the circles from 2 to 6 are the wireless nodes

The second topology used is similar to the one before but all the decoding of the nodes 3, 4, 5 and 6 is done in the wireless node 2 which is in charge of encoding again all the data received and send it to the sink, as it is visible in the figure 4.2. Node 2 plays the role of the cluster head in a hierarchical topology.

## 4.3. Dual Cluster Head Topology

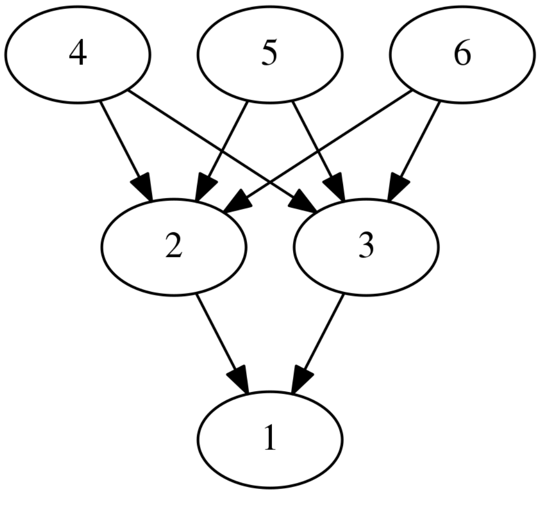


Figure 4.3.: Dual Cluster Head Topology, the circle 1 represents the sink, the rest of the circles from 2 to 6 are the wireless nodes

The last topology could be described as a hierarchical topology too, depicted in figure 4.3, the wireless nodes 2 and 3, higher level, decode the packets sent by 4, 5 and 6, leaf nodes, and then they encode the data again to send it to the sink.

# Part III. Network Coding

# 5. Network Coding Usage

Network coding is one of the most promising trends in the current researches for improving the network efficiency, it is based on the idea of making out of the intermediate nodes active entities that can manipulate the packets that they receive in stead of just retransmitting them. When talking about network coding we find two different approaches intra-flow network coding as MORE [15] and inter-flow network coding as COPE [16] or even combinations of both schemes as CORE [17]. The method used along this thesis will be part of the intra-flow network coding systems. The behavior is really simple a node encodes the data measured and another receiving node closer to the sink decodes it. Once it has decoded the whole block, the data is aggregated with the own node's measurements and other possible decoded blocks into a new generation and it's sent to the next node, repeating the process until the data reach the sink.

The two key parameters that have to be chosen regarding network coding are the field size and the generation size, both having impact on overhead, latency and complexity. The bigger the field size is the more complex is the decoding of the data and also the coding vectors become bigger but the advantage is that the linear dependencies are reduced with bigger fields. The finite field used during the tests is GF(2) because it is the easiest to implement and compute, as the sensors are not really powerful in terms of processing power. The problem using a small field, as said before, are the linear dependencies of the packets as each packet has a 0.5 chance (only two coefficients 0 or 1) to be added to the coded packet, the last packets have a lot of chances of being redundant. Those linear dependencies lead to 1.6 average extra packets [18] in case of error-free communication and is checked in section 7.2. The generation size is variable and it can change each generation. It depends on the number of packets that the sensor has in the moment of creating the new generation, that number also relies on the amount of data measured and the packets decoded by the node, because they will be added together in the next block. As the generation size varies with time the decoders also change according to the current size. The bigger than the generation size is, the bigger the latency becomes, as it's necessary to transmit at least as many packets as the generation size is to be able to decode, but greater generation sizes lead to better linear dependence overhead [19].

The use of network coding combined with UDP in wireless environments can outperform TCP when the channel presents losses [8], with greater losses the coding gain increases. Using network coding the number of overhead packets can be reduced, as there is no need of feedback from the sink until the decoding has been completed.

The field used is GF(2), the process of encoding in this field consists on XORing different packets belonging to the same generation into a new packet and keeping a record of the operations done to that packet in the coding vectors to make possible the decoding in the receiver's side. During this thesis the packets are sent first in the systematic manner and then coded. One of the advantages of using network coding is that redundancy packets can be sent without receiving feedback from the receiver, therefore it has been implemented an option to enable the source to send extra coded packets in the first group of packets to recover from losses that might have been produced by the channel.

# 6. Encoder

## 6.1. Send Queue Creation

The encoder that is being used during the simulations has been briefly introduced during the routing protocol algorithm (Algorithmus 1). Here it will be thoroughly analyzed.

The first stage of the encoding algorithm is to create a *SendQueue*, which is just a linked list with all the packets ready to be encoded and sent. First it takes the data stored in the node and waiting to be sent from the *ListData* list. It creates a packet with each of them, adding the corresponding header to the data:

- Coding vectors: They depend on the position on the "SendQueue" list.
- senderId: Always the sending sensor ID.
- idFlow: It is randomly generated each time a new "SendQueue" is created.
- packetType: 0 - Information, 1 - Urgent Information, 2 - ACKs
- id: In this first part, where the own sensor's information is being added to the SendQueue the id parameter is the sensor's ID.
- idPkt: In this first part, where the own sensor's information is being added to the SendQueue the idPkt parameter is randomly generated.

If the sendQueueLong is smaller than the maximum generation size more packets can be added to the queue. If the sensor has decoded some packets they are added to the SendQueue in the same way as the data was added but changing two of the header parameters.

- id: In this second part, the id parameter is the original sensor's ID.
- idPkt: In this second part, the idPkt parameter is the original packet's ID.

After creating the whole SendQueue the size of it is added to each packet, this sendQueueLong is the generation size (N) of the current block.

## 6.2. Send Redundancy

Once the generation size (N) is added to the packets the encoder is ready to start working. The first send attempt is a systematical one, sending the packets without encoding them. If the send

Table 6.1.: Number of redundant packets sent after systematic in different Send Redundancy versions

| Send Queue Long | $n$ in Send Redundancy V1 | $n$ in Send Redundancy V2 |
|---|---|---|
| 0 - 4 | 0 | 0 |
| 5 - 9 | 0 | 1 |
| 10 - 24 | 1 | 2 |
| 25 - 40 | 2 | 4 |

redundancy feature is enabled a number of random redundant packets (n) is sent after the systematical encoding to correct possible packet losses.

First, the Send Redundancy V1 was used but after some time running the number of packets was not high enough to really increase the successful decoding probability, then the second version was introduced, adding more packets to the first send. This number of packets can be easily modified to be according to the channel's lose rate.

Once the sending protocol is running it is in charge of checking if an ACK packet has been received before sending. If the ACK is found the encoder is cleared (empty SendQueue) and a new SendQueue is created as it was explained above.

If no ACK is found the "sendretries" parameter is increased and n randomly encoded packets are sent. The first random encoded send attempt with SendRedundancy 2 sends r, the number of redundant packets, less than the following attempts to reduce the overhead caused by this feature as shown in the table below.

Table 6.2.: Comparison between packets sent for Send Redundancy 1 and 2, with N = 10.

| Send Retries | 0 | 1 | 2 | Total packets |
|---|---|---|---|---|
| Send Redundancy V1 sent packets | 11 | 5 | 5 | 21 |
| Send Redundancy V2 sent packets | 12 | 3 | 5 | 20 |

## 6.3. Random Encoding

The random encoder works in the following way. It first creates a random coding vector with maximum length equal to the generation size and one all 0s packet which will be XORed to the packets marked by the random coding vector. After creating that random coding vector the encoder checks packet by packet if it is a component of the final random packet, this means that the packet's coding vector has its '1' in the same position than the random coding vector has a '1'. If it is a component it is XORed to the "sum packet" if not the next packet is checked. After checking all the packets in the SendQueue the randomly generated CV and the sum packet's CV should be the same.

With this method each original packet has a 0.5 probability of being included in the randomly generated packet. The encoder also checks if the coding vector is the "zero" vector, if it is the case the coding vector is randomly generated again.

Table 6.3.: Relation between generation size (N) and the number of randomly encoded packets sent per iteration (r).

| Generation Size, N | 1 - 5 | 6 - 20 | 21 - 40 |
|---|---|---|---|
| Packets per random iteration, r | $N-1$ | $\lceil N/2 \rceil$ | $\lceil N/3 \rceil$ |

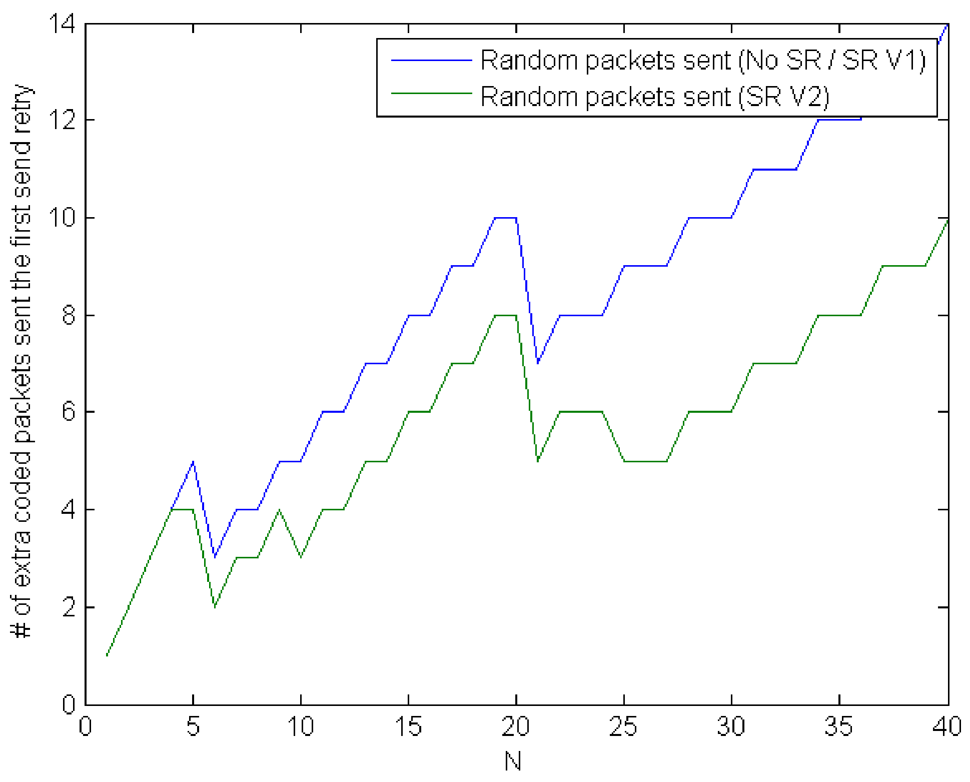


Figure 6.1.: Number of random packets sent the first send retry vs generation size for the two send redundancy versions

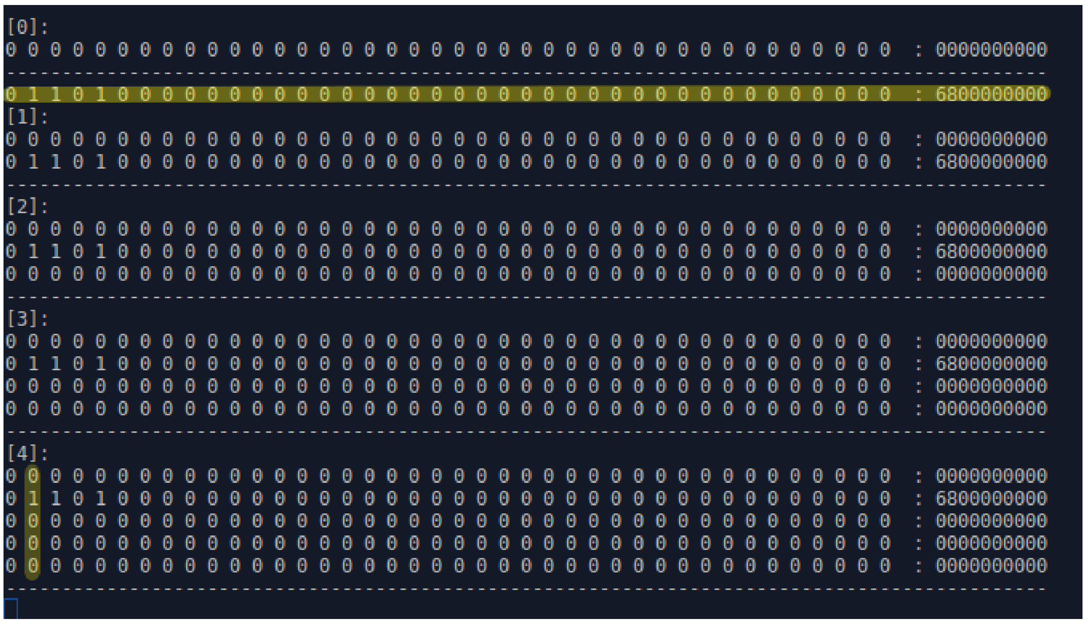
The number of packets sent in each randomly encoded iteration, r, also depends on the generation size, N. Figure 6.1 depicts r depending on N for the first send retry for both versions of Send Redundancy. The green line represents the random packets sent in the first send retry using Send Redundancy V2 and the blue line is the value of r using Send Redundancy V1 or no Send Redundancy. The rest of the send retries send the same number, r, of packets for both schemes.

# 7. Decoder

## 7.1. Decoder Operation

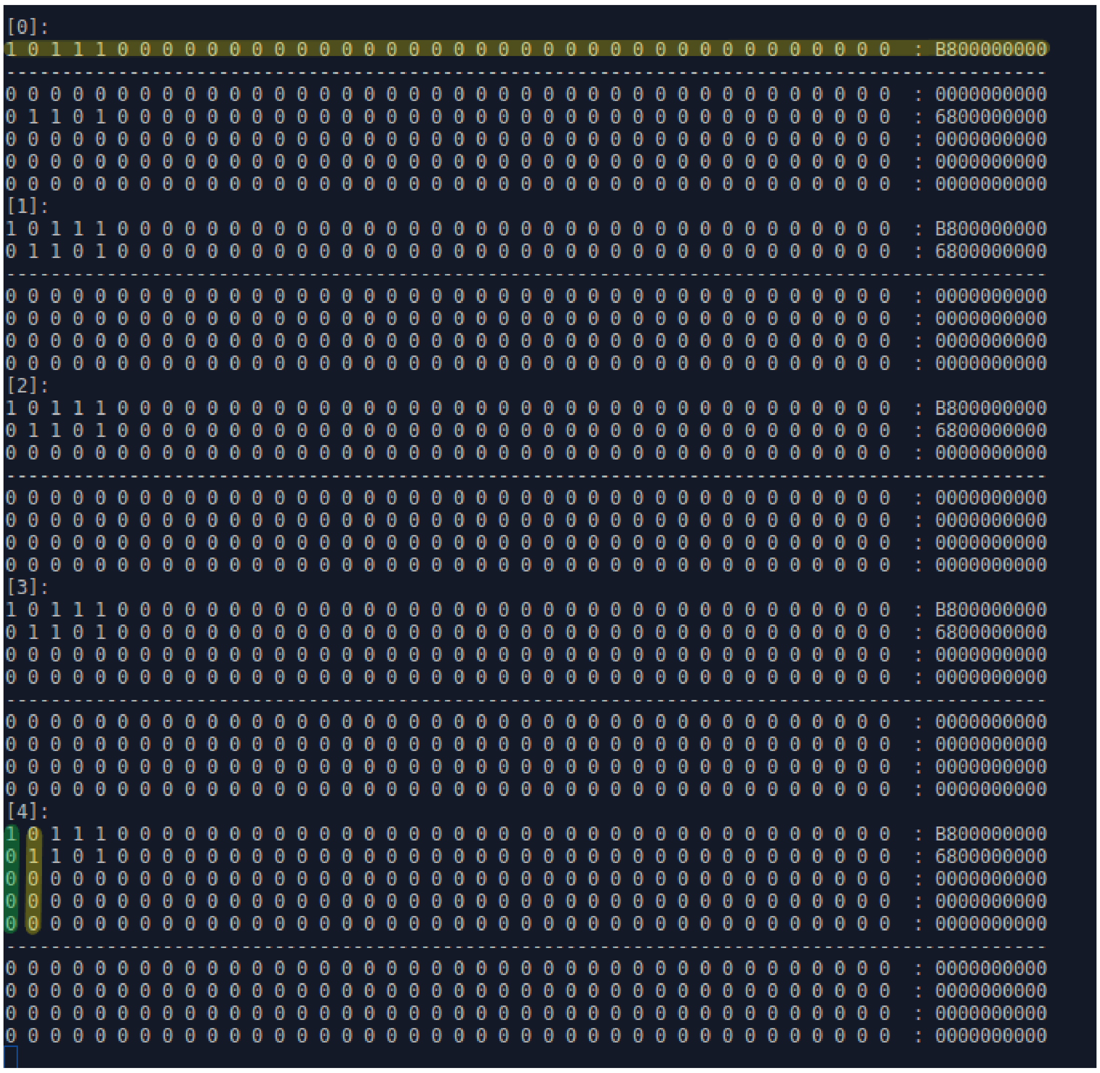
We implemented simple decoders in the nodes, as they don't have a lot of processing power and the field GF(2) does not require a complex decoder because only XOR operations are performed. Each time a packet from a certain node 'i' farther from the sink is received it is added to the decoder's *RecvPkts* list. Then for all the positions in the coding vector (Generation size) the decoder tries to find a packet whose "pivot", understanding pivot as the first non-zero element in the coding vector, is in the desired position in the coding vector. It checks the whole *RecvPkts* list, if a packet is found it's added to the *newList* list and deleted from *RecvPkts*. That *newList* is actually a new decoding matrix which is changing each time a not linearly dependent packet is received. If no packet is found a "dummy packet" (all 0s packet) is added in that position in *newList*. If the packet was found all linear dependencies with that position are deleted. In other words if other packets have a '1' in that position the original packet is XORed with them in order to have all 0s in that column. After that process is done with all the positions we get the identity matrix $rank = N$, which means that the data has been successfully decoded. Then the packets are copied to the *EncoderQueue* and a block ACK is sent.

Figure 7.1 to 7.5 are screenshots, of a decoding process for a generation size $N = 5$ and random packets. The yellow row is the new added packet, we can see how the decoder selects the packet with a '1' in the desired position and moves it to its place and the clears the column. In the last step all the "clear" columns are highlighted with colors, each of them increase the rank by 1.



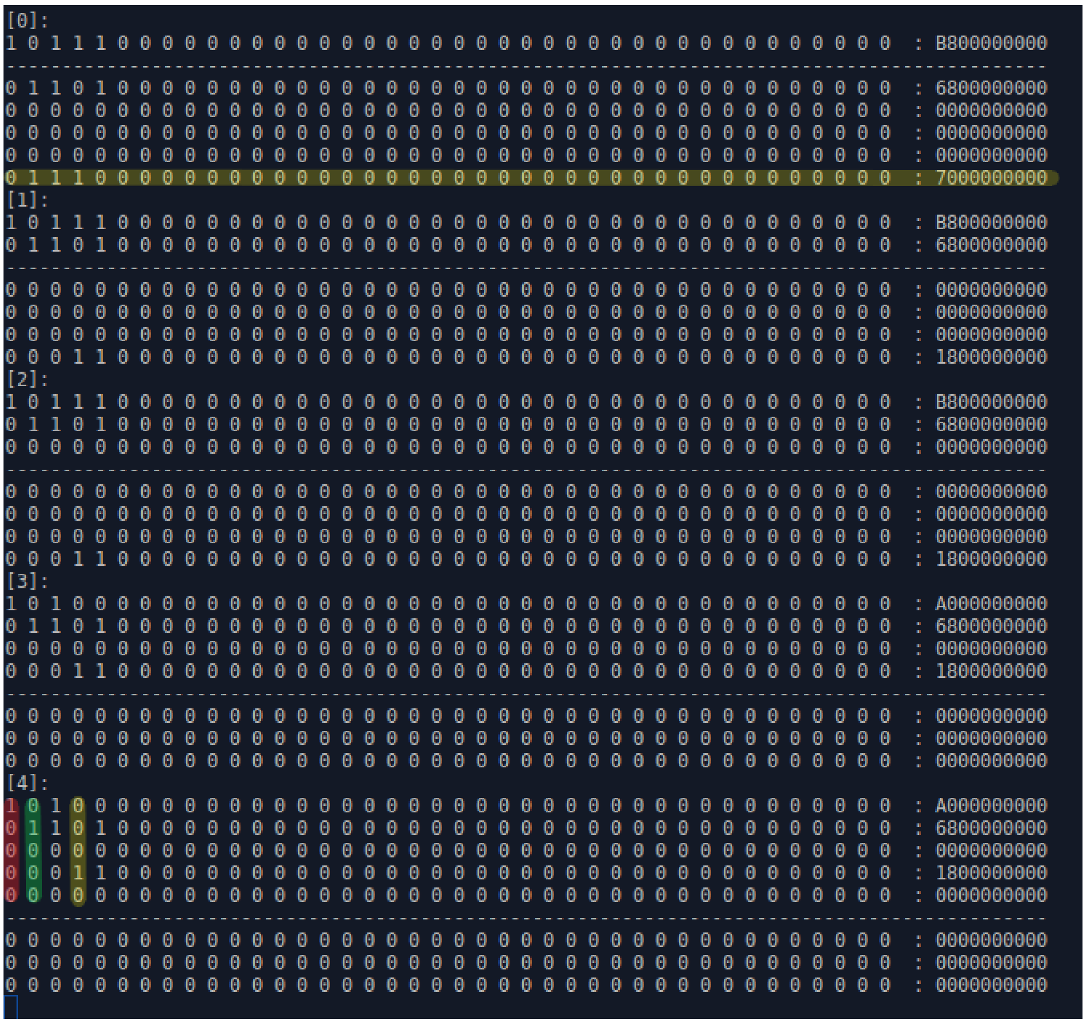
```
[0]:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
-------------------------------------------------------------------------------------------
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
[1]:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
-------------------------------------------------------------------------------------------
[2]:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
-------------------------------------------------------------------------------------------
[3]:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
-------------------------------------------------------------------------------------------
[4]:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
-------------------------------------------------------------------------------------------
```

Figure 7.1.: Decoding procedure: Rank [1/5]



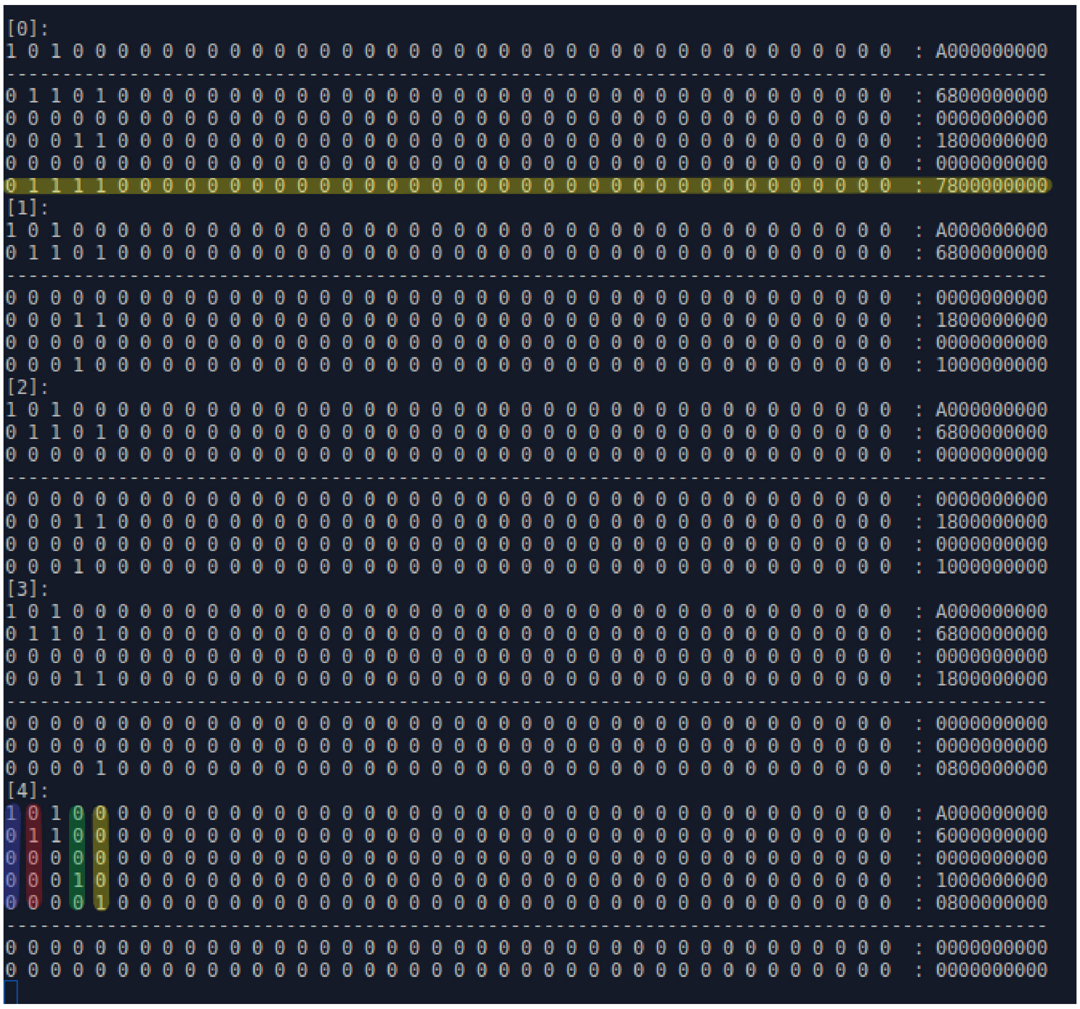
```
[0]:
1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : B800000000
-------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
[1]:
1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : B800000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
-------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
[2]:
1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : B800000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
-------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
[3]:
1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : B800000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
-------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
[4]:
1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : B800000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
-------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
```

Figure 7.2.: Decoding procedure: Rank [2/5]



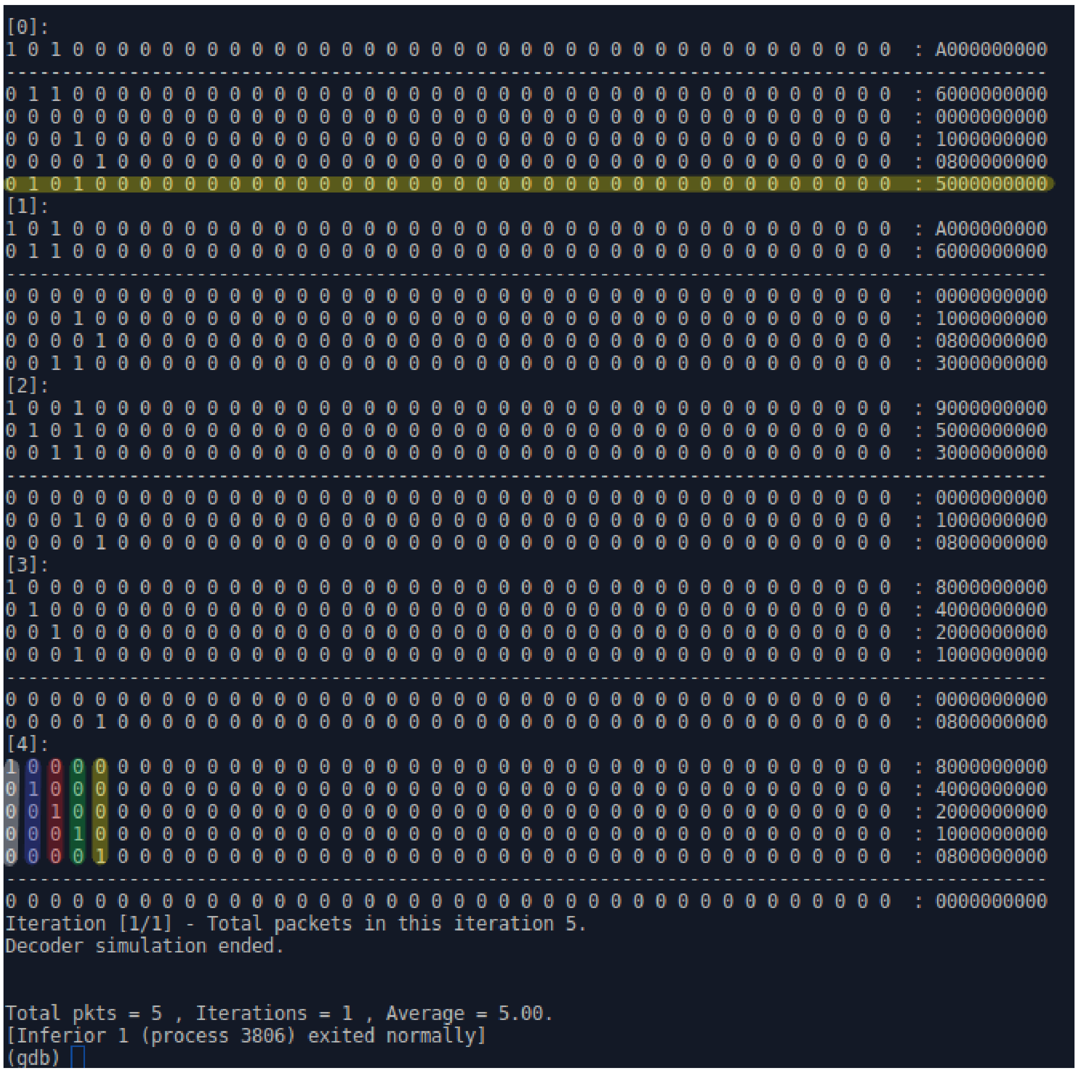
```
[0]:
1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : B800000000
---------------------------------------------------------------------------------------------
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 7000000000
[1]:
1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : B800000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1800000000
[2]:
1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : B800000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1800000000
[3]:
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : A000000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1800000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
[4]:
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : A000000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
```

Figure 7.3.: Decoding procedure: Rank [3/5]



```
[0]:
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : A000000000
---------------------------------------------------------------------------------------------
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 7800000000
[1]:
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : A000000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1000000000
[2]:
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : A000000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1000000000
[3]:
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : A000000000
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6800000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1800000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0800000000
[4]:
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : A000000000
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1000000000
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0800000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
```

Figure 7.4.: Decoding procedure: Rank [4/5]



```
[0]:
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : A000000000
---------------------------------------------------------------------------------------------
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1000000000
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0800000000
0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 5000000000
[1]:
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : A000000000
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 6000000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1000000000
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0800000000
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 3000000000
[2]:
1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 9000000000
0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 5000000000
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 3000000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1000000000
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0800000000
[3]:
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 8000000000
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 4000000000
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 2000000000
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1000000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0800000000
[4]:
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 8000000000
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 4000000000
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 2000000000
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 1000000000
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0800000000
---------------------------------------------------------------------------------------------
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  : 0000000000
Iteration [1/1] - Total packets in this iteration 5.
Decoder simulation ended.


Total pkts = 5 , Iterations = 1 , Average = 5.00.
[Inferior 1 (process 3806) exited normally]
(gdb)
```

Figure 7.5.: Decoding procedure: Rank [5/5]

## 7.2. Decoder Tests

To measure the decoder's behavior, a simple simulation was performed. A clear decoder, this means whose buffers are completely empty, was given packets one by one and tried to decode the information, then the total number of packets was saved in a file. The minimum number of packets to decode is 40 as the simulation was executed with the maximum generation size, thus every number of packets above 40 has overhead.

After 5000 iterations the output file was analyzed with a Matlab script and these were the results:

- Average overhead = 1.5866, really close to the theoretical 1.6 [18]
- No overhead probability = 0.2930
- Probability Distribution Function of overhead packets

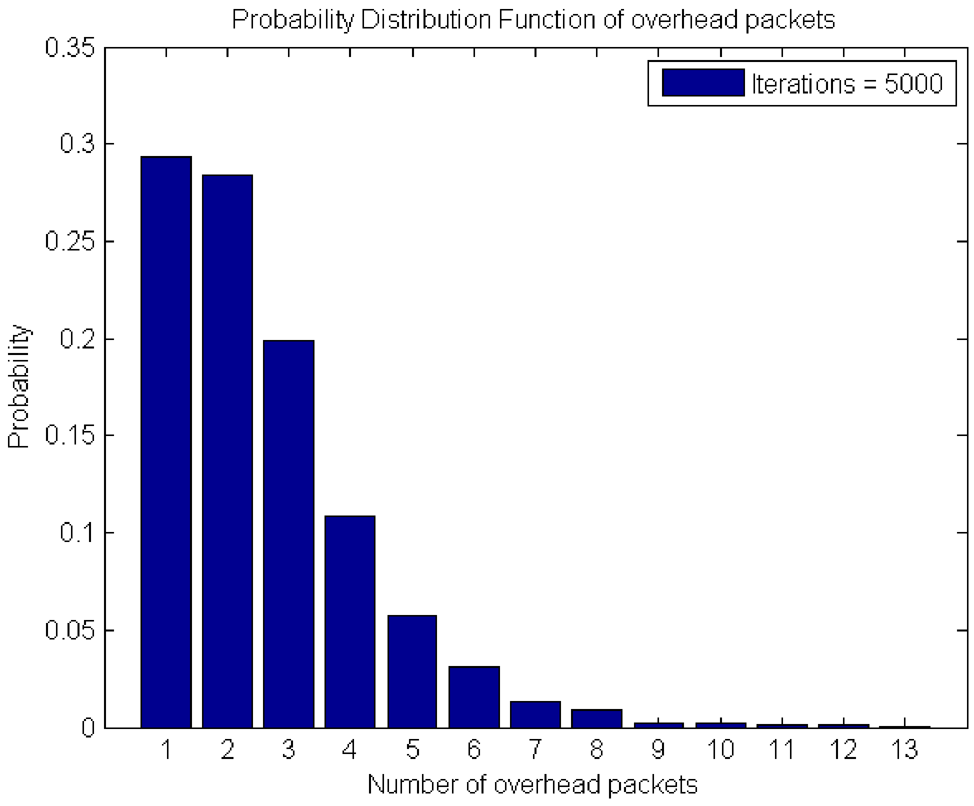


Figure 7.6.: Probability Distribution Function of overhead packets

# Part IV.

# Practical Implementation

# 8. Practical Test Bed

The network consists of 5 XDK sensors and a sink, which is a computer running a program to collect the data measured by the sensors. The WiFi access point is a TP-Link AC1900 router. The computer's hardware is listed below:

- Processor: Intel® Core$^{TM}$ i7-6700T, 64-bits, 4 cores, 8 threads, 2.80-3.60 GHz, 8 MB Smart-Cache
- RAM memory: 1 bank, 16 GiB, 2133MHz
- Graphics adapter: Sky Lake Integrated Graphics
- Ethernet interface: Intel® Ethernet Connection I219-LM, 1000Base-T
- Wireless interface: TP-Link TL-WN722N
- Main disk drive: Samsung MZNLN512, ATA Disk, 512 GB, M.2 OEM SATA solid state drive
- Secondary disk drive: Toshiba MQ01ABD1, ATA Disk, 1 TB, Serial ATA-300 hard drive
- OS: Ubuntu 16.04.2 LTS

All the tests were executed during 1 hour with people moving and working around the sensors, with around 10 WiFi networks overlapping in the same channel.

Each node sends its status to the sink after every iteration. They send its ID, the total amount of information packets and the total number of packets (information and coded retransmissions) sent since starting the node, the amount of information and total packets sent in each generation, the encoder queue length (number of packets decoded and waiting to be encoded again), the generation size, the send retries and the nodes in sight. With all that data the results could be calculated. The sensors measure the environment every 2 seconds and send the information each 6 seconds, creating 3 packets each time they want to send information to the sink, therefore the minimum generation size will be 3.

The actual location of the sensors during the simulation are shown in the maps below, for the together simulation all sensors were placed really close to each other, around 2 centimeters away and close to the sink, less than half a meter. In the same room simulation all the sensors were in the same places (Figure 8.1) independent of the topology used. Two different placements were used for the different rooms simulation (Figures 8.2 and 8.3).

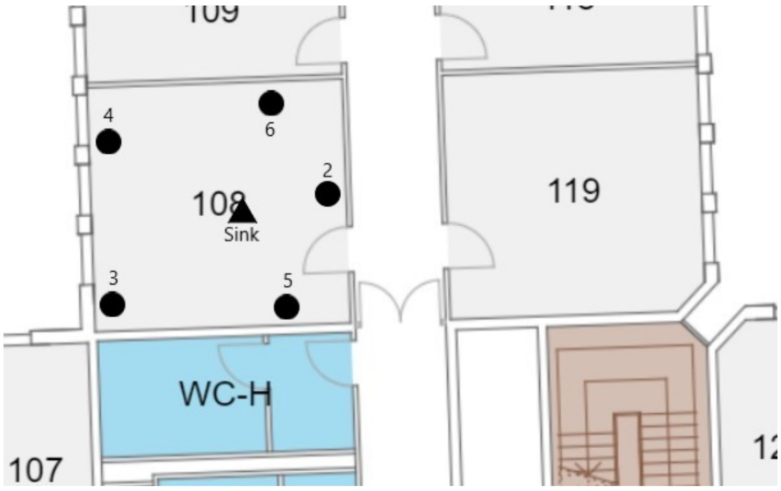


Figure 8.1.: Test bed for same room tests, the black triangle represents the sink and each of the black circles is a node, the number close to the black dot is the node's ID

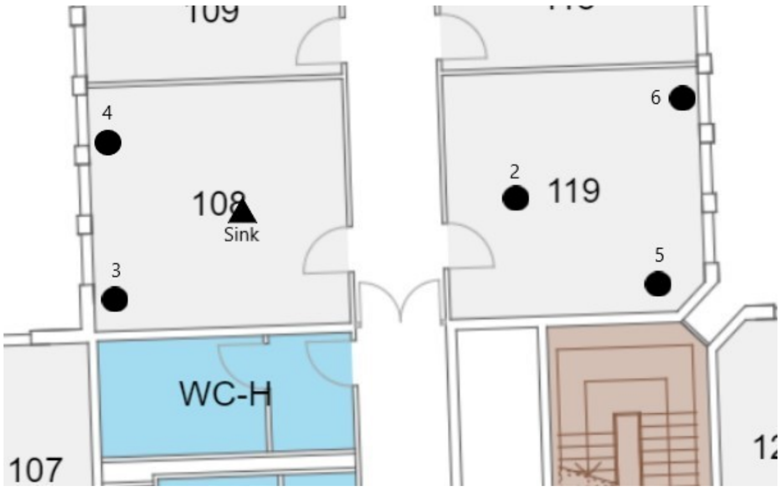


Figure 8.2.: Test bed for different room tests with the star and single cluster head topology, the black triangle represents the sink and each of the black circles is a node, the number close to the black dot is the node's ID

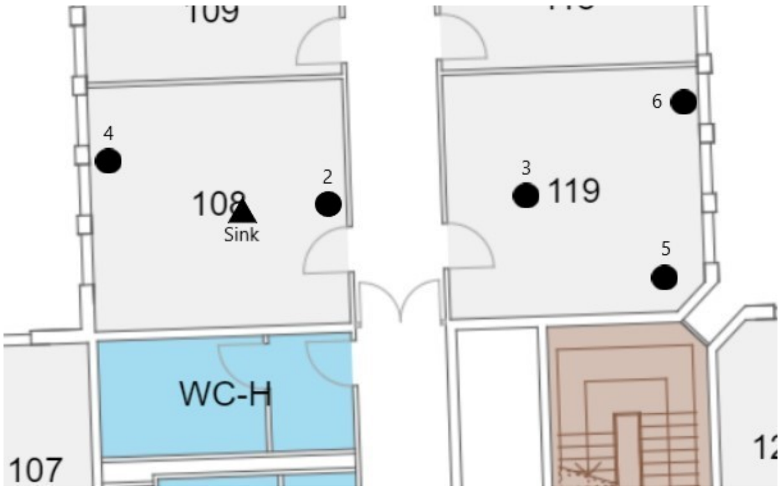


Figure 8.3.: Test bed for different room tests with the dual cluster head topology, the black triangle represents the sink and each of the black circles is a node, the number close to the black dot is the node's ID

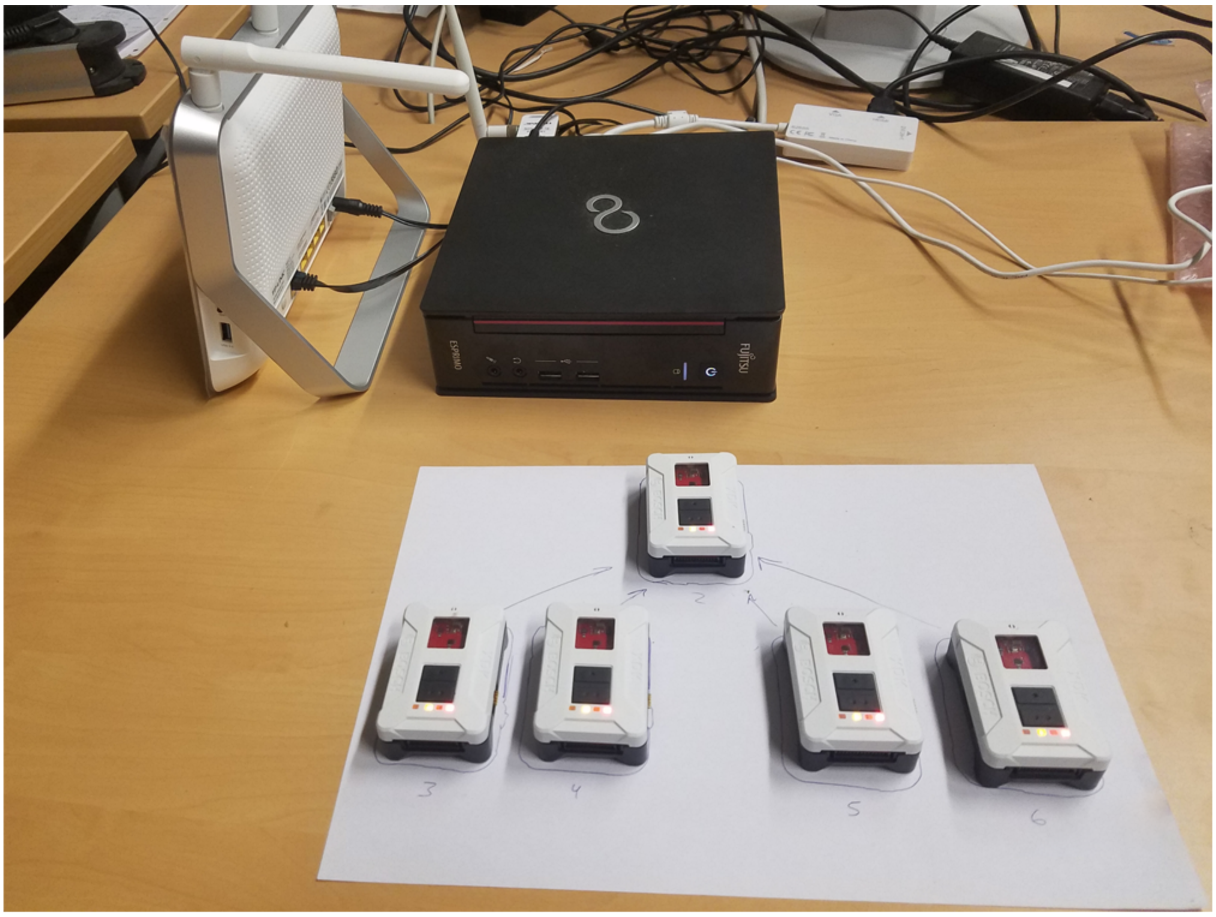


Figure 8.4.: Photo of the test bed during the Single Cluster Head Topology, Together Test. In the photo the WLAN access point (WiFi Router, Left), the sink (Computer, Center) and the five wireless sensors (XDKs, Front) can be clearly seen.

# 9. Results

The different topologies and distances have been compared looking at different parameters, such as the average packets sent per information packet, the decoding probability, the average generation sizes and the number of overhead packets. The plots are usually expressed in terms of iterations, keep in mind that each iteration is 6 seconds after the one before, the time between sends. Most of the tests have been one hour long:

$3600\ seconds/6\ seconds\ per\ iteration = 600\ iterations$

## 9.1. Star Topology

This is the simplest topology that is going to be used. As all the nodes are connected in the same way to the sink, just one hop, and they are being directly decoded by it and they are expected to behave similarly.

### 9.1.1. Together

In this first test the nodes will be really close to each other and the sink, low overhead and really high successful decoding probability are expected as the sink decodes better than the nodes. All the nodes are connected with the sink, thus they will be directly decoded. Low generation sizes are expected, around 3 packets which is the minimum; the send retries are also expected to be low. Both Send Redundancy versions will be analyzed next.

#### Send Redundancy 0

Figure 9.1 shows the average number of packets that are sent for each information packet every 100 iterations. This is the ratio resulted of dividing $K/N$ each generation. The optimal number is 1 packet per information packet, which means that no retransmissions nor linear dependencies happened in any sensor in any of the generations, anything above that ratio will mean that overhead appeared. These optimal points are represented by the blue circles in the charts. As the plot shows, the number of extra packets is really low, not any of the asterisks exceeding the 1.07 ratio, which mean less than 7 extra packet each 100 information packets or, in other words, less than 7% overhead. An increase of one point in this plot mean error bursts during those iterations. As all the points are around the same values, the loss rate remains invariable during the test.

These data have been calculated dividing the generation size, N, which is the actual number of information packets, by the total number of packets sent in that generation, K, each time that the generation changed for every sensor. Then the average of those numbers was done in the iterations multiple of 100. Finally the resulting numbers were added to each other and divided by the number of nodes.

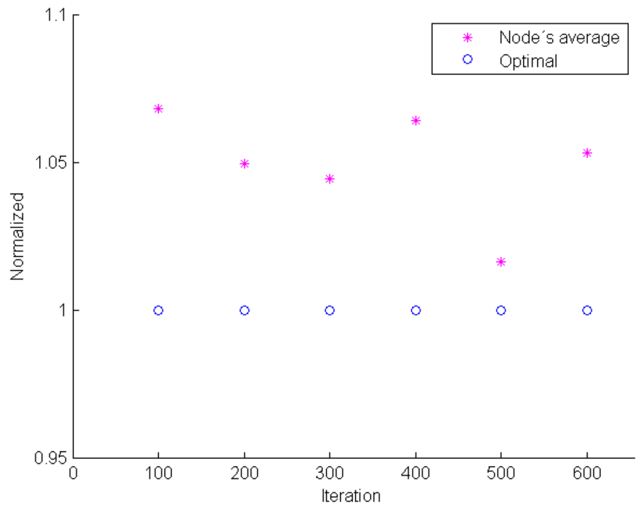


Figure 9.1.: Average K/N ratio each 100 iterations, Star Topology, SR0, Together

The figure 9.2 is the CDF of successful decoding depending on the "group of packets send times" meaning the first position that the generation was decoded at first send, the second at first send + 1 send retry, the third at first send + 2 send retries and so on. The number of packets sent each iteration is variable, firstly sending $N + n$ packets and the rest of the send retries $r$, the values of $n$ and $r$ are obtained according to tables 6.1 and 6.3 respectively. The first point is the probability of the data being decoded after the first send round, is really high a bit over 95%. Looking thoroughly at the chart we can say that after 4 group send times the data will be decoded. The greatest send retries during this implementation were 4 (5 send times), which could have been an unlikely event.

The plot data have been calculated with an array as long as the greatest send retry + 1, then each time a block was decoded the position in the probability array with the number of send retries of that block + 1 was added one. After checking all the data each position in the array was divided by the sum of all of them. The CDF was calculated adding to each element in the array the one before starting from the second. The really unlikely cases with a probability under 1e-5% have been ignored.

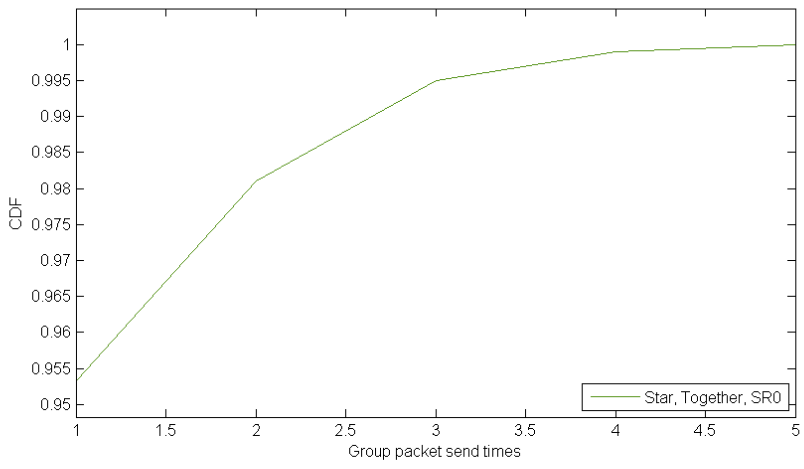


Figure 9.2.: CDF of successful generation decoding, Star Topology, SR0, Together

The figure 9.3 is a comparison between the nodes, as they are all connected directly to the sink and at the same distance they are supposed to behave in a similar way, having the same overhead, decoding probability, send retries and generation size, which are the four parameters shown in the plots. The most different behaving node seems to be the 3, having a bit bigger packet overhead than the others, despite none of them exceeding 7% of overhead, this results are in accordance with the ones from the figure 9.1 which indicated an average overhead under 7%. According to the theory the nodes should behave similar, and actually they do as the differences are really small from one node to another in the plots. In other topologies there will be greater differences as some nodes will have different roles. In these plots the relation between the overhead, the generation size and the send retries is clearly visible in the node 3 as all those parameters are closely related. If the decoding probability decreases retransmissions are needed, which will increase the send retries, the overhead increases due to those retransmissions and as the node has to retransmit the current generation some data will be enqueued for the next generation, increasing the generation size.

The measures have been made in the same way for all the sensors. The packet overhead was calculated with the last round of information received from each node with formula 9.1.

$$Packet\ Overhead = \frac{sum(K) - sum(N)}{sum(N)} * 100 \tag{9.1}$$

The decoding probability was calculated dividing the number of times that the data was decoded at the first send by the total number of times that the data was decoded. The average send retries and generation size were calculated keeping the sum of them each time a block was decoded and then dividing by the times that a decoding was done.

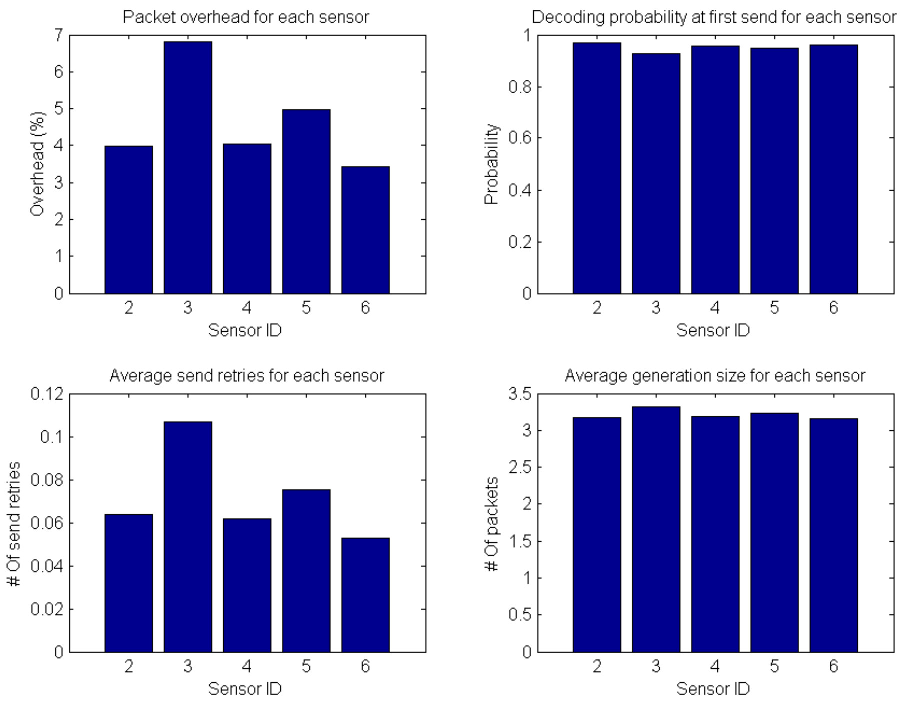


Figure 9.3.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Star Topology, SR0, Together

Figure 9.4 show the information regarding the node 3 status, which was the one with bigger overhead. As we can see in the third plot (Average packets sent per information packet) in the first 400 iterations the retransmissions are greater than in the last 200, lowering from $\approx 1.1$ *to* $\approx 1.02$. That decrease can also be seen in the overhead packet count (2nd) and the generation size (4th) charts, where a reduction in the number of overhead packets that were sent is remarkable in the last stages and a reduction in the generation size is clearly visible too. The first plot shows the last received total number of packets and total information packets as it is the last updated. The second plot was obtanined substracting the information sent packets, N, to the total sent packets, K, each iteration and substracting the accumulative overhead of the iterations before if they were from the same generation, because K is the total amount of packets sent in that generation and we are interested in the iteration's overhead:

```
if generation of iteration i == generation of iteration i-1 then
    $ovpkts(i) = totalSentArrayThisIt(i) - infoSentArrayThisIt(i) - additiveOverhead;$
    $additiveOverhead = additiveOverhead + ovpkts(i);$
else
    $ovpkts(i) = totalSentArrayThisIt(i) - infoSentArrayThisIt(i);$
    $additiveOverhead = 0;$
end
```

**Algorithmus 2 :** How the number of overhead packets is calculated each iteration.

The third plot has been calculated in the same manner than the overall packets sent per information packet. The fourth graph is just the representation of the data sent by the node, as it was one of the parameters that were sent in the status packets.

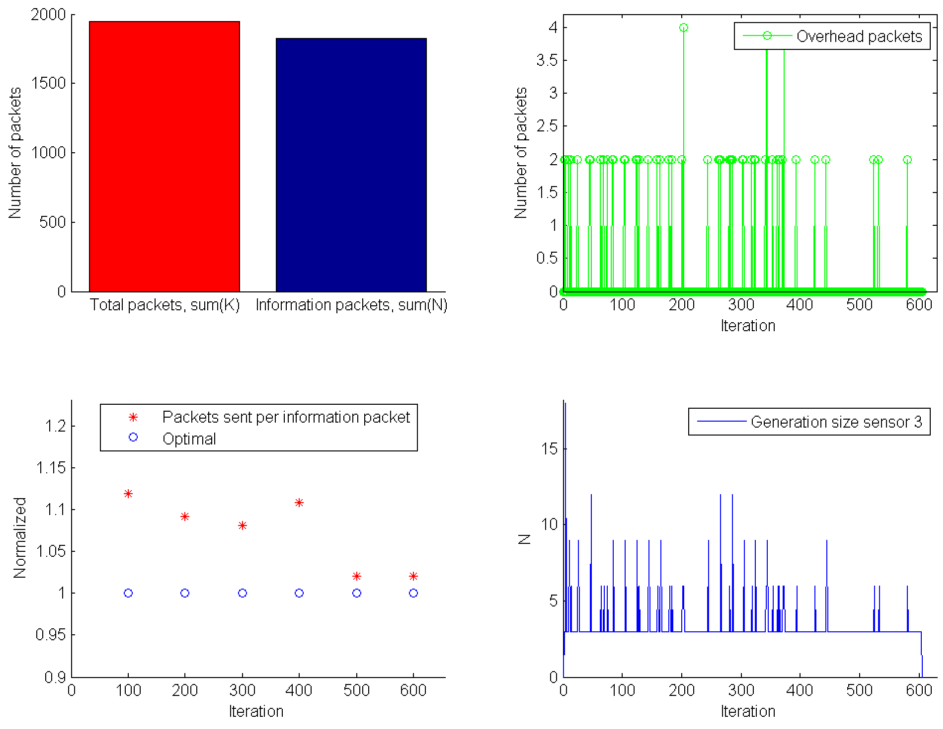


Figure 9.4.: Node 3's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Star Topology, SR0, Together

### Send Redundancy 2

For this second part no important differences are expected compared with the Send Redundancy 0 as the generation sizes are really small and the send redundancy algorithm starts working above 5 packets in the generation size. Although after one first failed send the next generation will have at least 6 packets, 3 for each iteration, as in the iteration before the sensor was retransmitting the failed generation. In this case at least one redundancy packet will be added after the systematic encoded packets (Table 6.1) and the overhead will be higher. All the plots were calculated in the same way as in the first part, this will be a common point for each of the results shown.

The figure 9.5 shows the average packets sent for each information packet. We see the first differences, the first 400 iterations were around the same ratio ($< 1.1$) as with Send Redundancy 0 but after, in the last 200 iterations, that ratio increases up to 1.3 which will be a big difference in the overhead caused by the retransmissions which also will lead to a lower decoding probability at the first send.

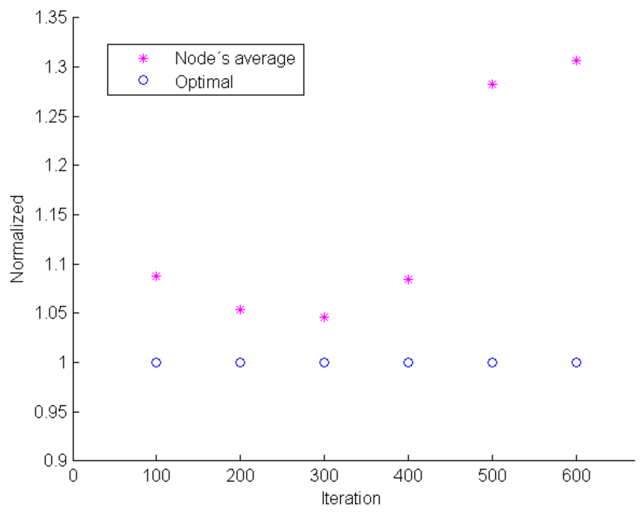


Figure 9.5.: Average K/N ratio each 100 iterations, Star Topology, SR2, Together

As expected once we saw the K/N ratio, during this test there were greater losses, that could be caused by interferences, that is the reason why when we look at the figure 9.6 we can appreciate that it is worse than in the case before even when the theory says that it's supposed to be higher, dropping the decoding at first send probability to almost 0.88. If the last 200 iterations (higher losses) are ignored the chart looks more like the one without Send Redundancy as is shown in figure 9.7, where the decoding at first send probability is close to 0.94. For both charts we could say that the block would be successfully decoded after 4 group of packets send times.

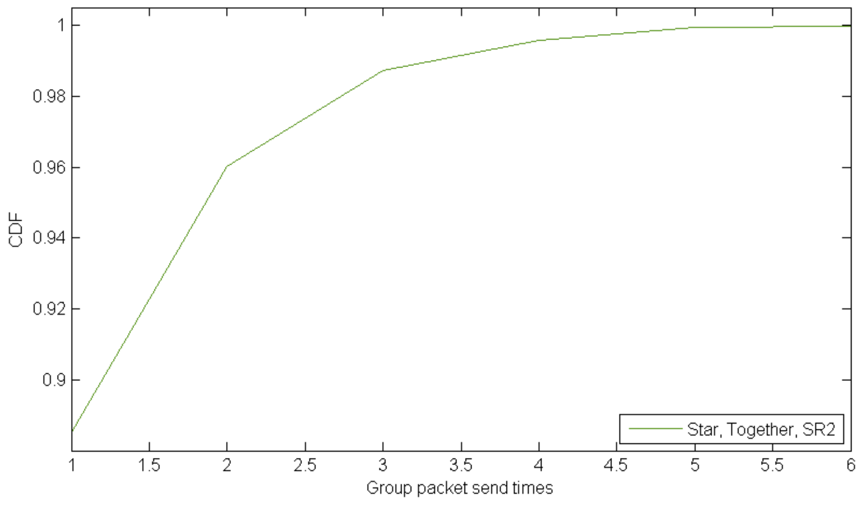


Figure 9.6.: CDF of successful generation decoding, Star Topology, SR2, Together

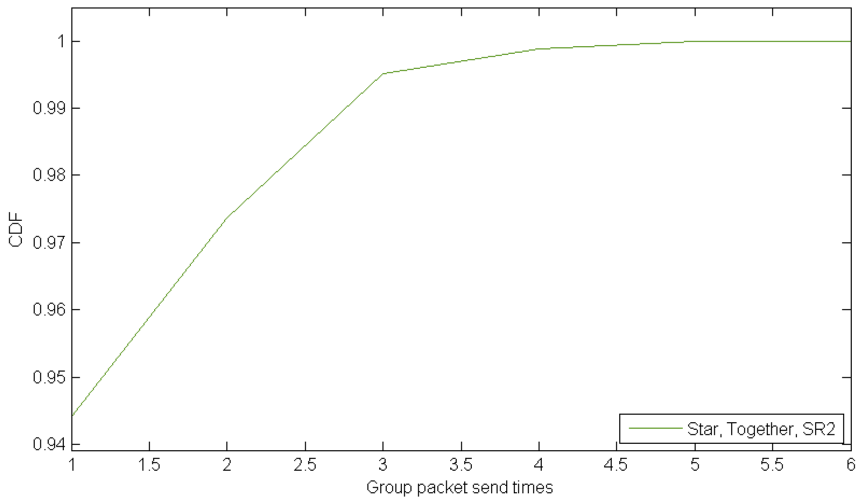


Figure 9.7.: Shrunken CDF of successful generation decoding, Star Topology, SR2, Together

Figure 9.8 shows four charts with the nodes' status according to the same parameters that were analyzed in the results before. As expected the overhead is greater than in the test without Send Redundancy, due to the redundancy packets and the burst of errors. But contrary to the theory the decoding probability is worse, as has been explained before. Keeping in mind that the last part of the test had higher errors the average send retries are not so high, rounding 0.17 for each sensor, which means that in average 17 out of 100 generations had to be retransmitted once. Those retransmissions also increase the average generation sizes, which are around 3.5 whereas in the Send Redundancy 0 test were always beneath it.

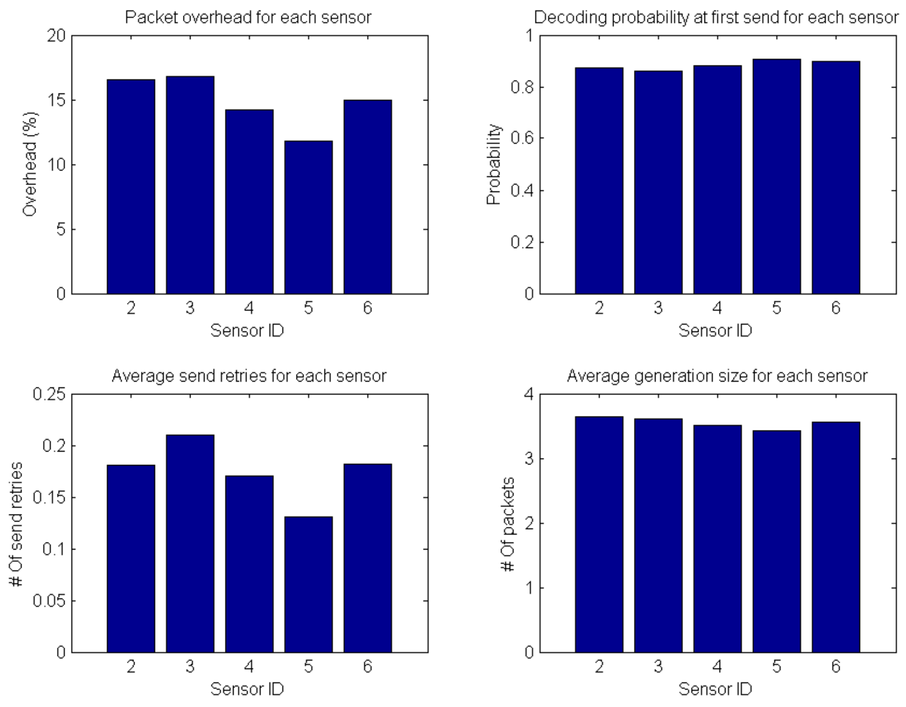


Figure 9.8.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Star Topology, SR2, Together

The figure 9.9 show the status of the node 3 during the test, the second, third and fourth plots of that figure reflect the same phenomenon that was visible in the average packets sent per information packet, the last 200 iterations have greater losses. In the sensor 5 this difference was even more clear as is shown in the figure 9.10 where during the first 400 iterations almost no overhead packets appeared and after they were in almost each generation.

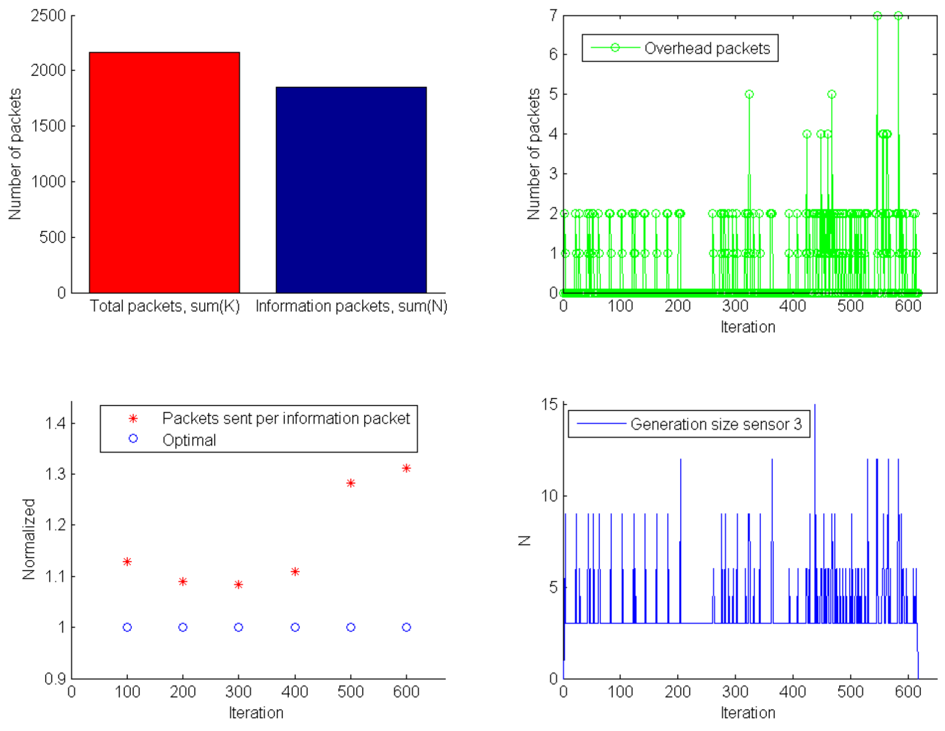


Figure 9.9.: Node 3's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Star Topology, SR2, Together

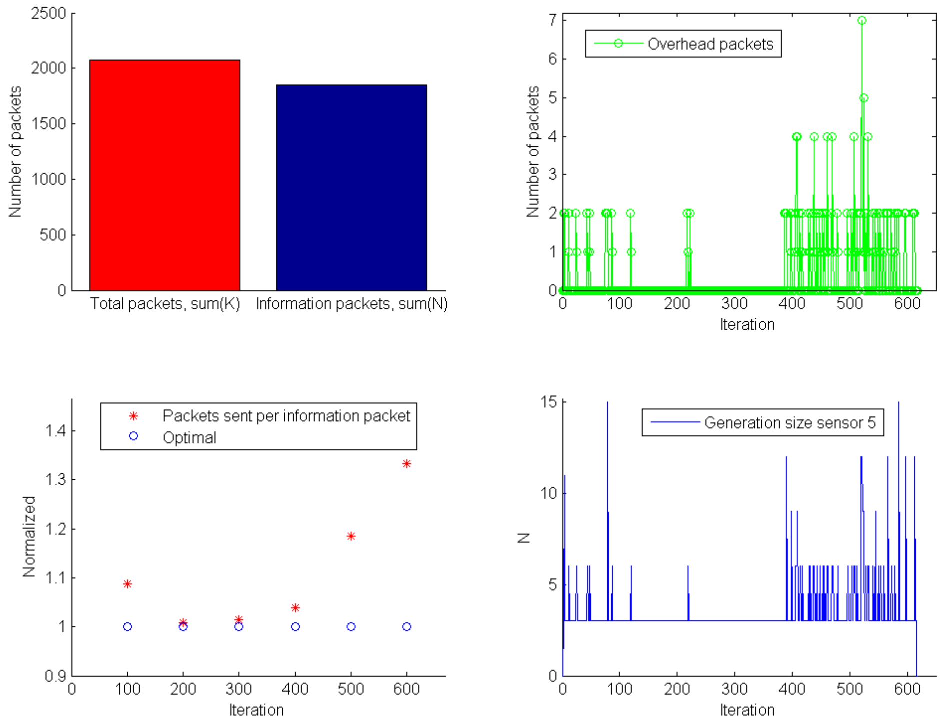


Figure 9.10.: Node 5's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Star Topology, SR2, Together

### 9.1.2. Same Room

As the distance has been increased the error rate is expected to increase, thus decreasing probability of decoding and growing the overhead compared with the previous test.

#### Send Redundancy 0

Following the previous structure, the figure 9.11 shows the average K/N ratio each 100 iterations. Looking at the graph no big differences over time are seen but for iterations 400-500 where the ratio was almost optimal. We can suppose that no bursts of losses appeared during the test. The low ratio, almost reaching the optimal one, is also an indicator of the low overhead that we can expect in the next charts.

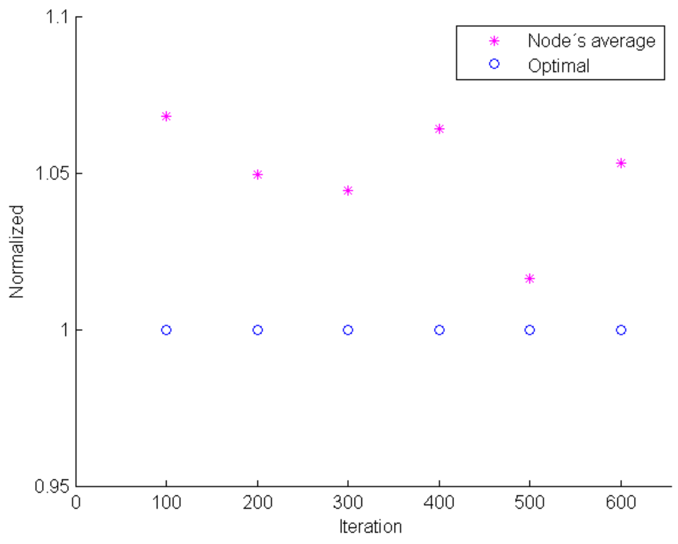


Figure 9.11.: Average K/N ratio each 100 iterations, Star Topology, SR0, Same Room

Figure 9.12 shows that the decoding probability is a bit lower than at closer distances, according with the theory and the fact that no burst of losses occurred, event that would have disturbed the results. After 3 group send times the probability of decoding the generation is almost 1.

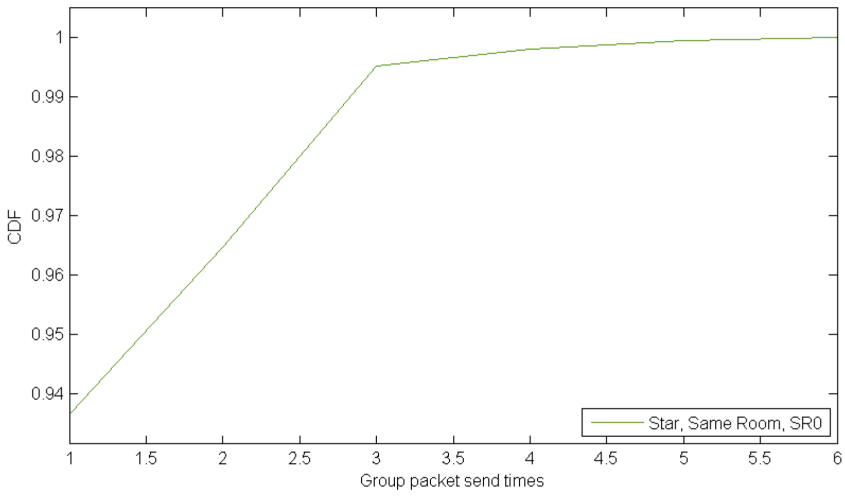


Figure 9.12.: CDF of successful generation decoding, Star Topology, SR0, Same Room

Figure 9.13 has four graphs which show the same parameters as in previous sections for each node. The overhead is really low for all nodes, having the node 3 the highest with 10%, the send retries remain low too, as well as the generation size which is between 3 and 3.5 for all nodes.

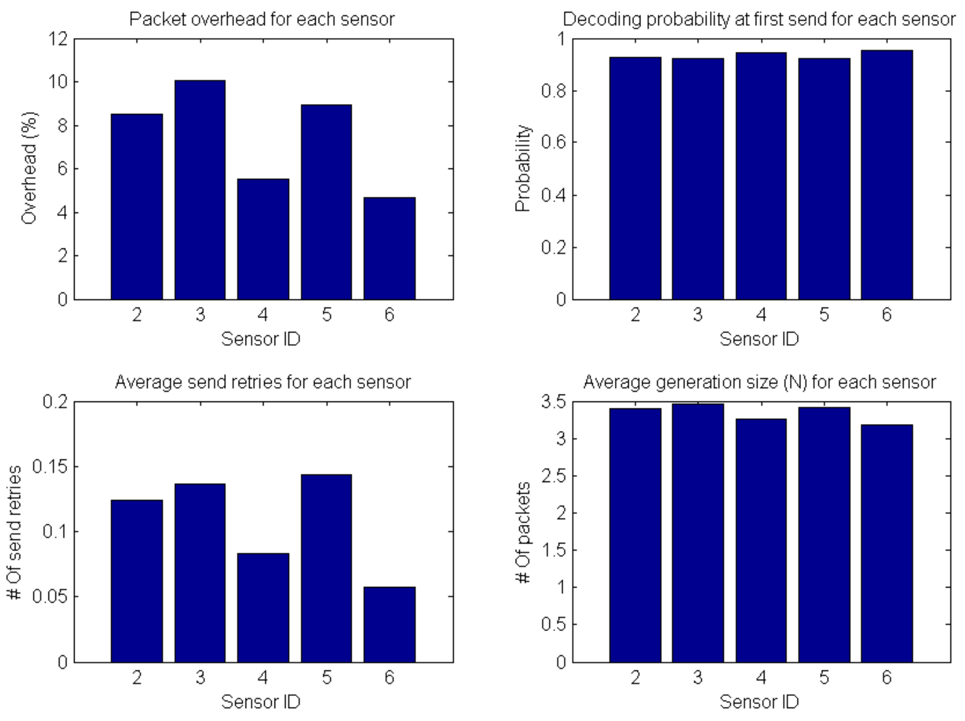


Figure 9.13.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Star Topology, SR0, Same Room

Figure 9.14 shows the status of the node 3 during the test. As it was the sensor with highest overhead its status should be the most interesting. As it is visible in the third chart the losses were approximately the same during the course of the test but in the first 100 iterations where they were a bit lower. The second and fourth chart show the overhead in each iteration as well as the generation size, those high spikes, reaching up to 18 packets in the generation, are the reason why the node 3 has a small difference in overhead compared with the other nodes.

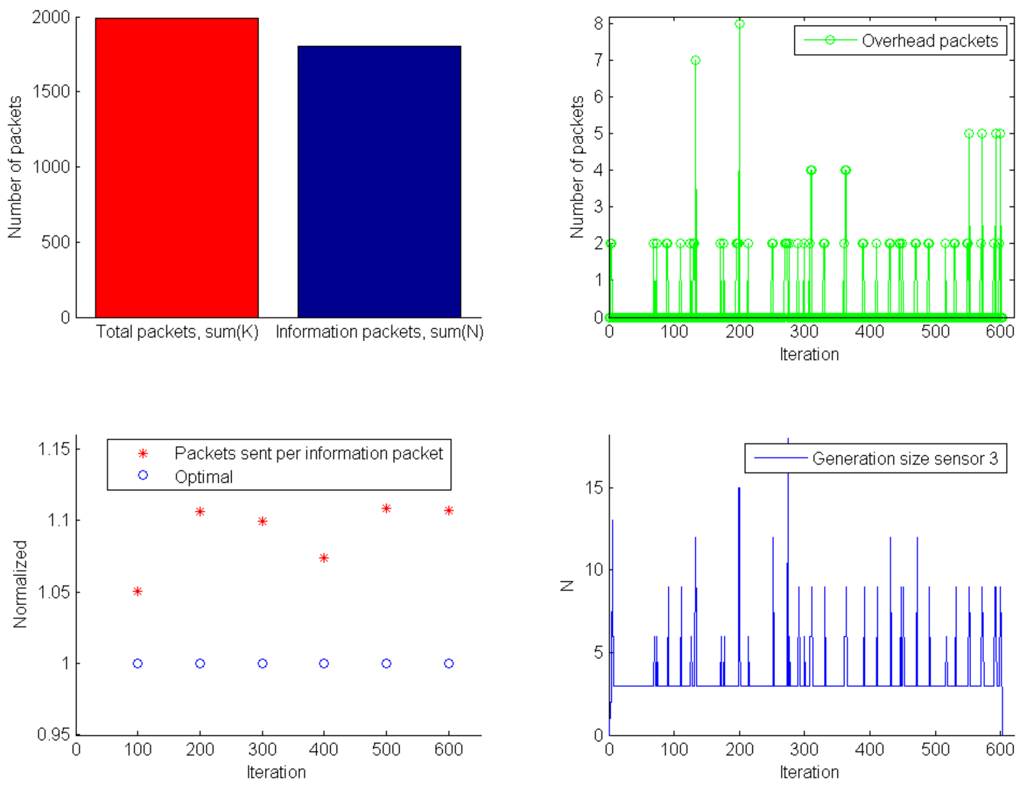


Figure 9.14.: Node 3's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Star Topology, SR0, Same Room

### Send Redundancy 2

As said in the previous analysis of the Send Redundancy 2 implementation, this is in theory a trade-off between the overhead and decoding probability at the first send, trying to increase this last parameter at the expense of increasing the first one too.

In the figure 9.15 average K/N ratio is shown, it looks like the one belonging to Send Redundancy 0, as mentioned in the star topology, the generation sizes are expected to be low which makes the Send Redundancy mechanism not to be make an important difference. Most of the measures are around the 1.06 ratio, which would make a 6% overhead in average.

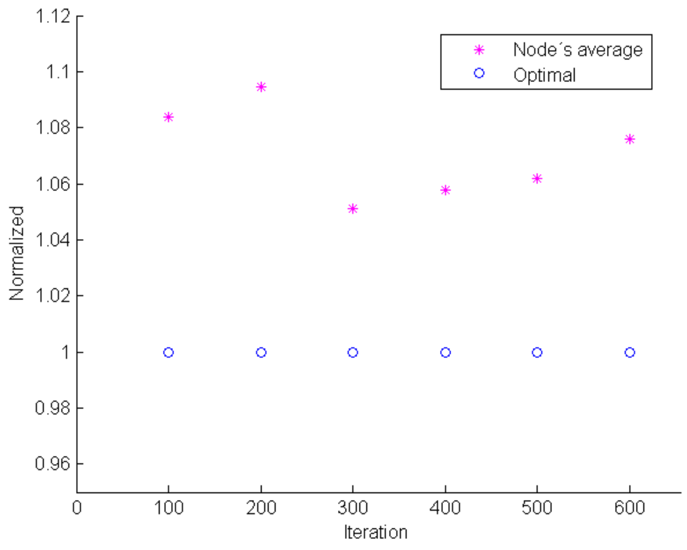


Figure 9.15.: Average K/N ratio each 100 iterations, Star Topology, SR2, Same Room

Figure 9.16 shows the CDF of decoding depending on the group sends. The probability of decoding after one packet group send is a bit over 95%, as said by the theory has increased slightly compared with the one in Send Redundancy 0. According to the chart we can say that after 4 group send times the data block will be successfully decoded.

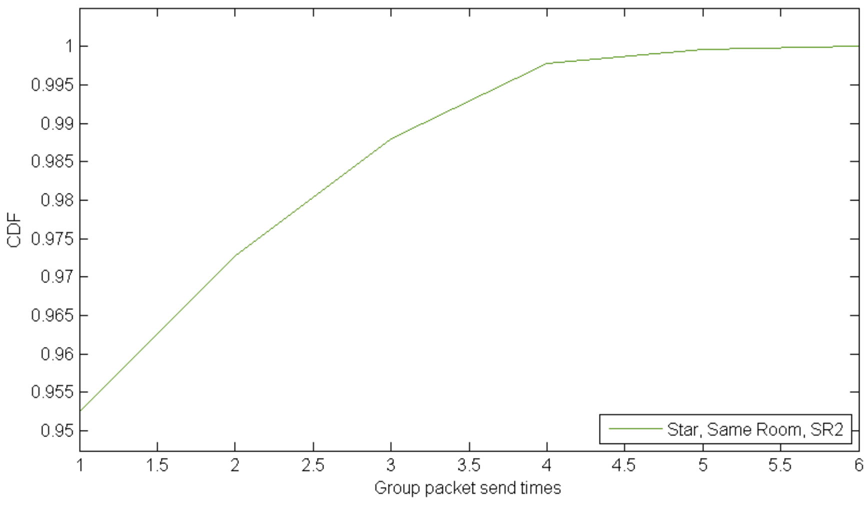


Figure 9.16.: CDF of successful generation decoding, Star Topology, SR2, Same Room

The figure 9.17 shows the status of the different nodes during this test. They were expected to be the same as they are all leaf nodes connected directly to the sink but the packet overhead is much higher in the nodes 3, 4 and 5, specially in the node 3, whose status will be shown in figure 9.19. The plots show again the relation between the overhead, the decoding probability at first send, the number of send retries and the generation sizes.

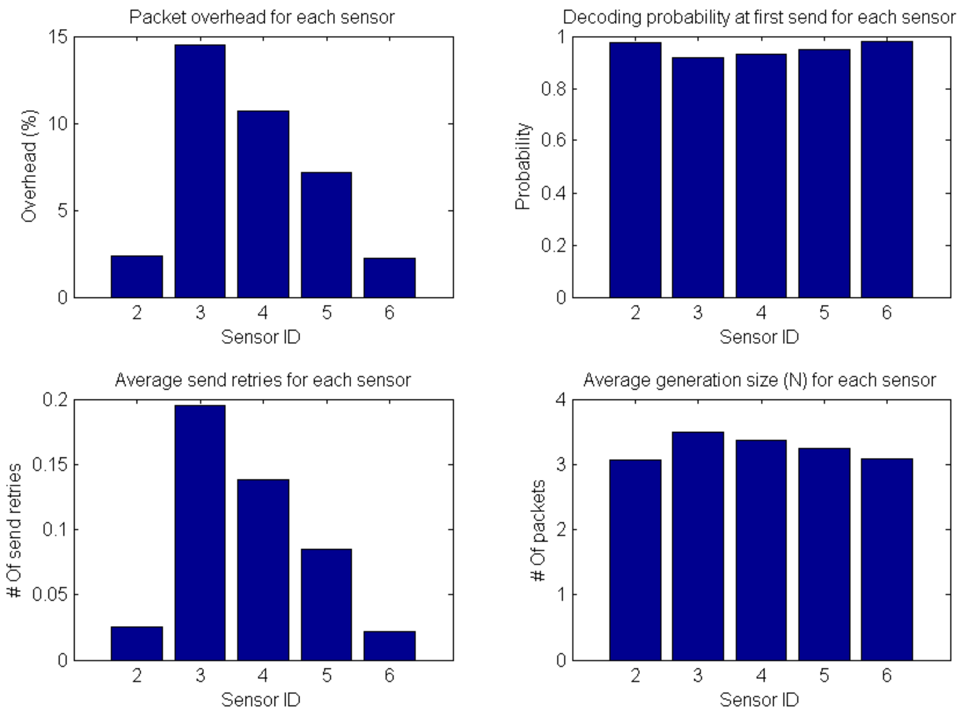


Figure 9.17.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Star Topology, SR2, Same Room

As an example of one of the nodes with best overhead figure 9.18 shows the status of the node 2 during the test. As we can see almost no overhead seem to appear during the test and the generation size almost doesn't move from the minimum.

On the other hand, figure 9.19 are the four charts regarding the status of the node 3, which was one with greatest overhead in this test. The overhead graphic, the second one, exposes why it has such a big overhead compared with the other nodes, it had to retransmit packets in a lot of iterations, which is also seen in the generation size plot, where a lot of spikes moving from the minimum indicate that there were retransmissions.

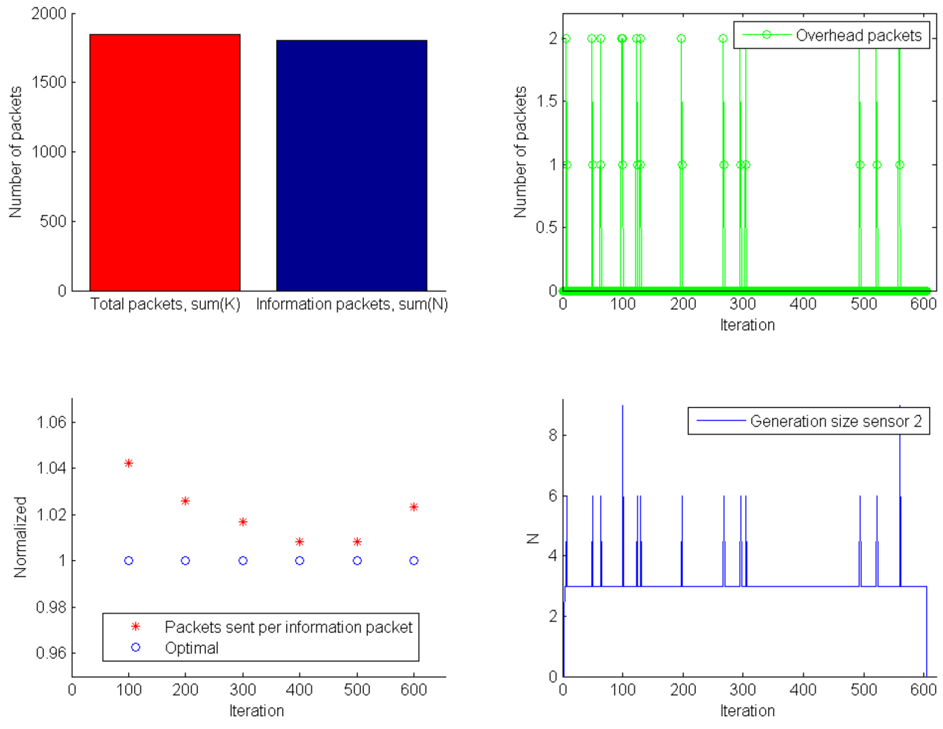


Figure 9.18.: Node 2's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Star Topology, SR2, Same Room

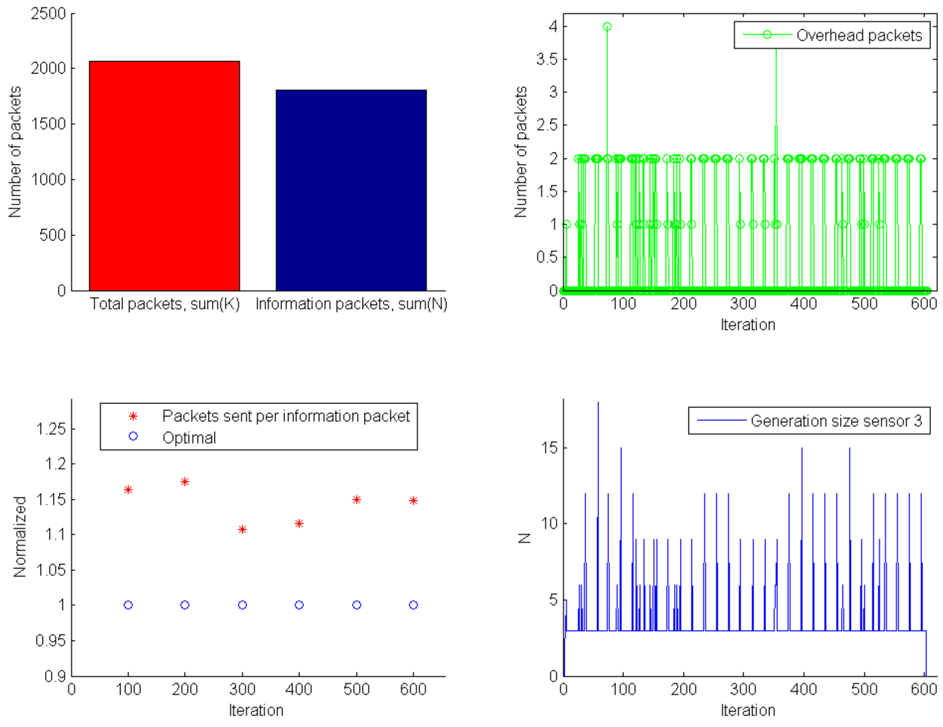


Figure 9.19.: Node 3's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Star Topology, SR2, Same Room

### 9.1.3. Different Rooms

The distance has been increased again in this test, thus the same theoretical changes are expected, lower decoding probability and greater overhead. This time only the results for the test with Send Redundancy 2 will be shown as the differences are minimal.

#### Send Redundancy 2

Figure 9.20 shows the average K/N ratio. As is common in this topology it is close to the optimal point, but for the first 100 iterations, as we will see later the nodes 2 and 6 had really big bursts of errors during that time.

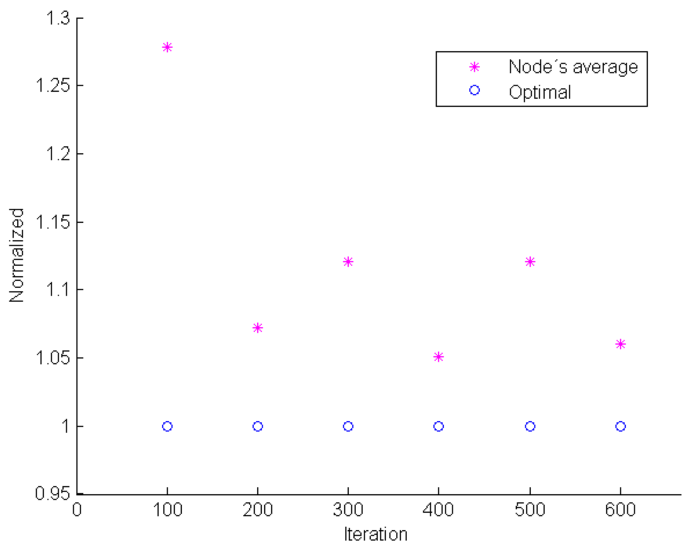


Figure 9.20.: Average K/N ratio each 100 iterations, Star Topology, SR2, Different Rooms

The figure 9.21 represents the CDF of successful decoding after the number of send group times. As we can see the decoding probability after the first send is almost 92% and after 3 send times more than 99% of the generations were decoded successfully.

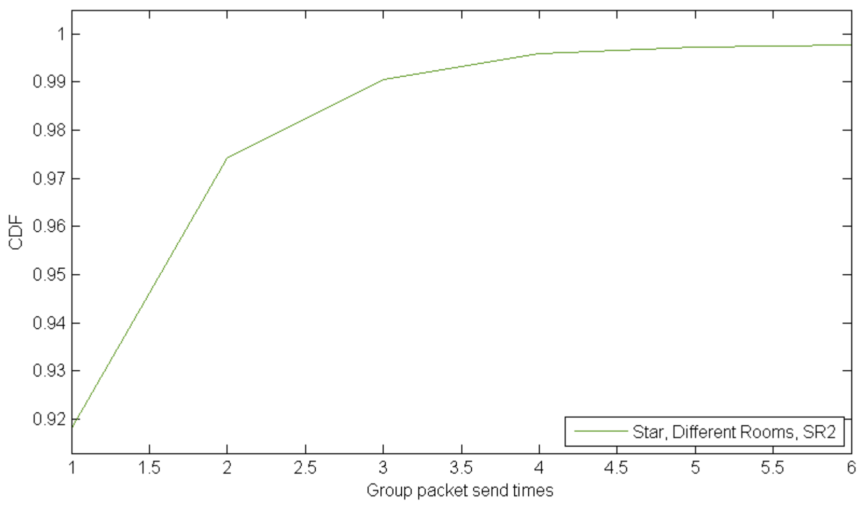


Figure 9.21.: CDF of successful generation decoding, Star Topology, SR2, Different Rooms

In the node comparison graph depicted in figure 9.22 we can appreciate that the nodes have approximately the same behavior but for node 6 which has much greater overhead.

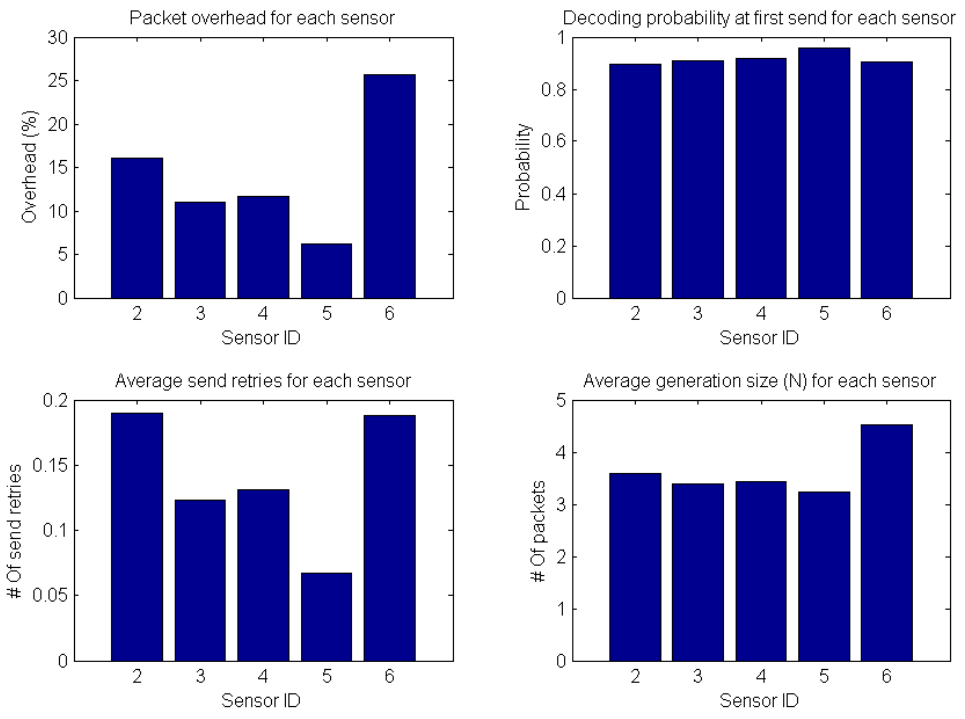


Figure 9.22.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Star Topology, SR2, Different Rooms

As advanced when analyzing the first figure the node 2 shows big losses in the first 100 iterations which disappear in the following ones, as can be seen in figure 9.23. Leading to an increase in the first point in the average packets sent per information packet chart. The same lossy pattern in the first iterations is visible in the node 6's overhead plots in figure 9.24, being in this case even bigger, up the 1.6 ratio.

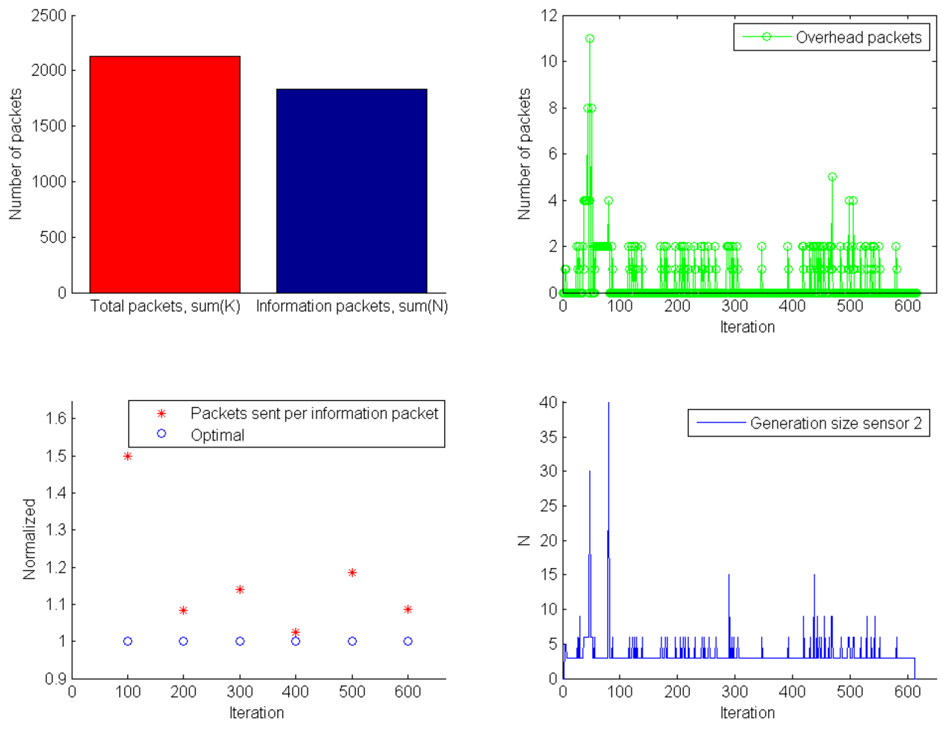


Figure 9.23.: Node 2's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Star Topology, SR2, Different Rooms

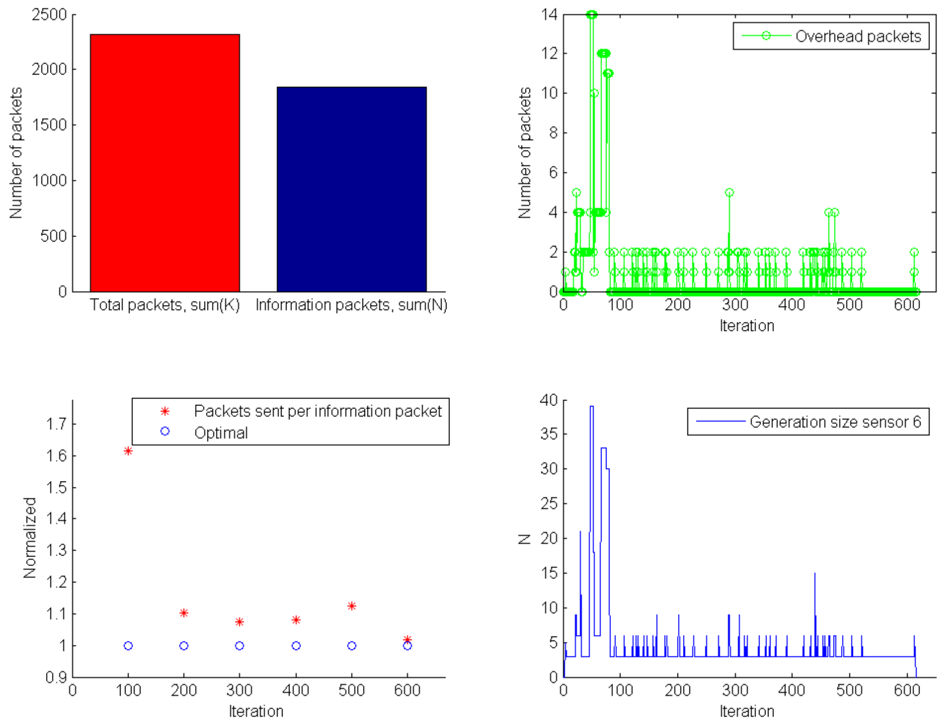


Figure 9.24.: Node 6's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Star Topology, SR2, Different Rooms

## 9.2. Single Cluster Head Topology

This topology introduces several changes compared with the star topology. First of all now the nodes' data will be decoded by one of the nodes, which will encode the data again and will send it to the sink, therefore we are passing from a single hop topology to a multi-hop one. The node will have smaller computing resources than the sink, such as computing power and memory, which will end in problems as will be explained in section 9.4.2 the node's WLAN interface will probably be worse than the sink's. Now that the role of one node has changed the difference with different generation sizes will be more visible.

### 9.2.1. Together

The first test results shown will be at the minimum distance.

#### Send Redundancy 0

Figure 9.25 shows the average packets sent per information packet. We can clearly see that the overhead for the recoding node 2 is much smaller than the overhead for the rest of the nodes, this is because bigger generation sizes are more efficient than smaller. It is also visible a huge increment in the iterations from 300 to 600 because the nodes 3 and 4 had huge losses in the that last part.

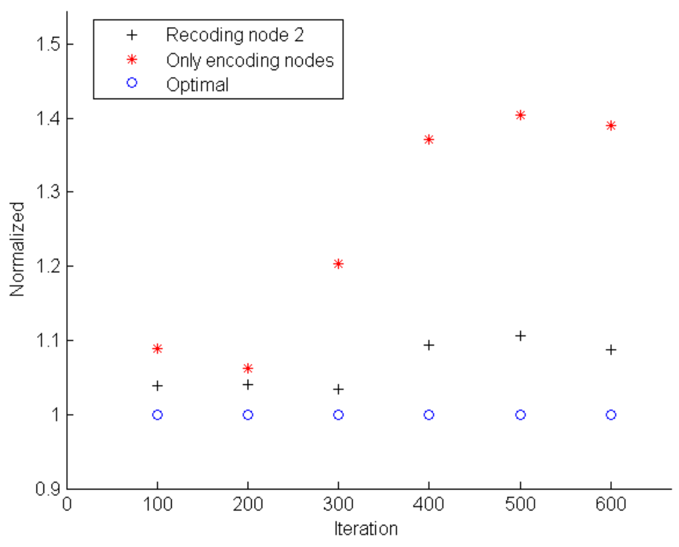


Figure 9.25.: Average K/N ratio each 100 iterations, Single Cluster Head Topology, SR0, Together

In the figure 9.26 it's clear that the decoding probability at first send has decreased a bit but there is a huge jump in the second group send time and at the third we could affirm that the block will be successfully decoded.

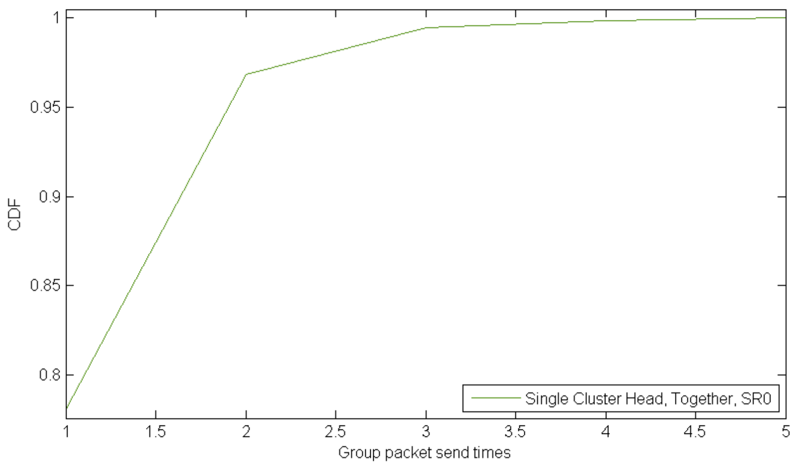


Figure 9.26.: CDF of successful generation decoding, Single Cluster Head Topology, SR0, Together

In the figure 9.27 we can appreciate the effects of those losses over the overhead, the decoding probability, the send retries and the generation sizes. The overhead for the nodes with losses has highly increased up to 45% and the overhead for the node 2 which was aggregating all the data is a bit smaller than the overhead for the nodes 5 and 6. When looking at the generation sizes graphic we can see that the average for node 2 is a bit higher than the total traffic of all the nodes, mainly because those failures in the nodes 3 and 4, which have a bigger generation size than three, the amount of packets that they were supposed to send each iteration.

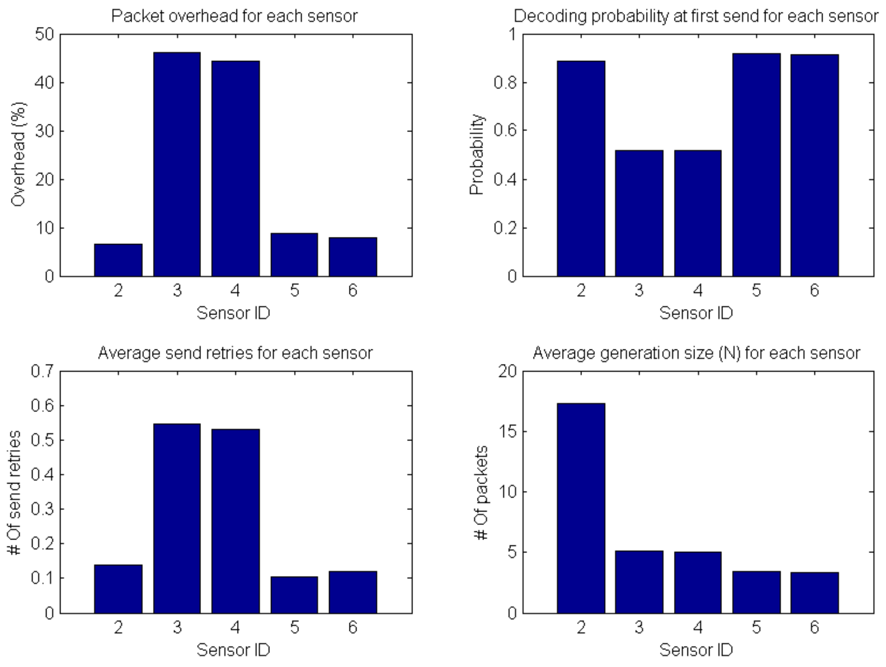


Figure 9.27.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Single Cluster Head Topology, SR0, Together

Figure 9.28 shows how that loss rate also affects the node 2, increasing the overhead and making the generation sizes fluctuate as sometimes it won't receive packets from nodes 3 and 4 and then it will receive all the missing packets in one generation.

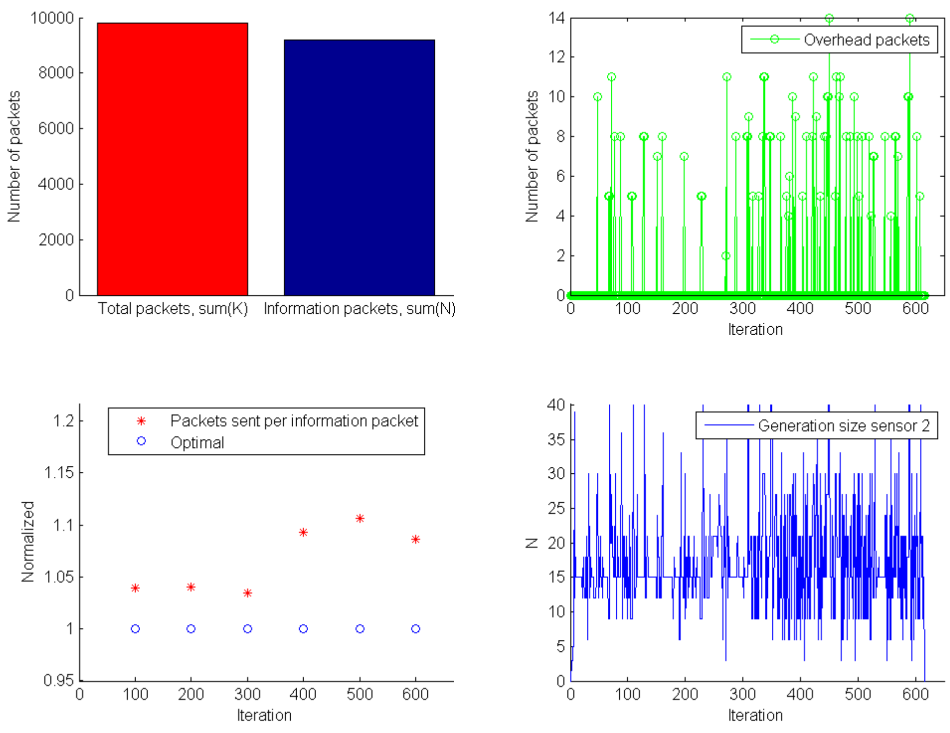


Figure 9.28.: Node 2's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Single Cluster Head Topology, SR0, Together

Figure 9.29 shows clearly the losses produced in one of those nodes, where the overhead increase is clearly visible. When looking at the generation size plot it's remarkable that most of the generations are decoded with the next group of packets, so probably only a couple of packets are lost and the problem would be solved with some redundancy packets.

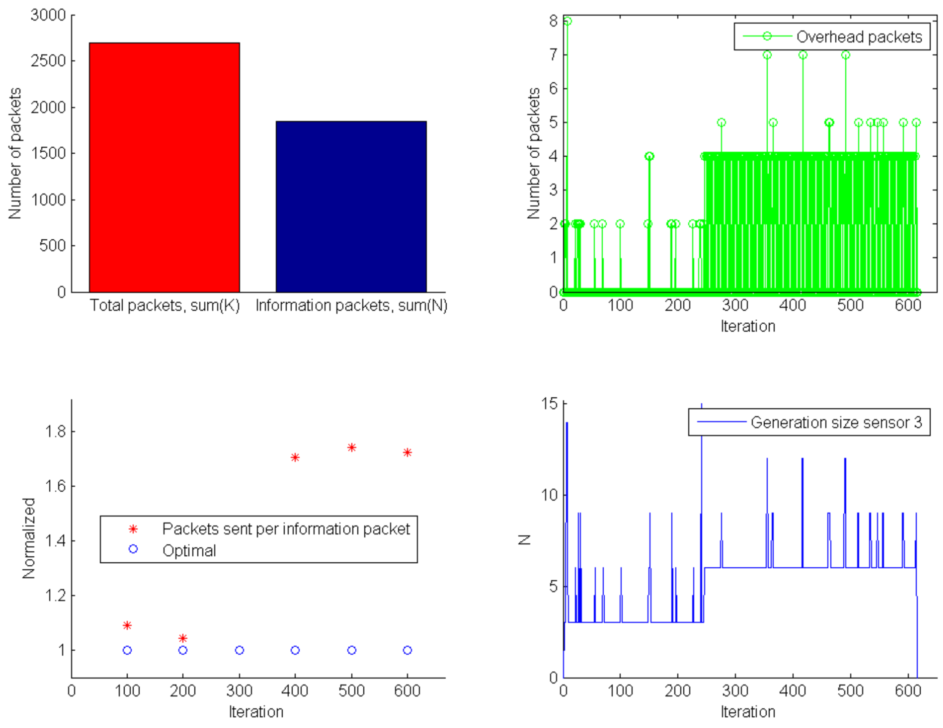


Figure 9.29.: Node 3's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Single Cluster Head Topology, SR0, Together

## Send Redundancy 2

In this topology and distance the Send Redundancy 2 won't be discussed as the differences with this last part were minimal and not worth commenting, just the expected improvement in the decoding probability at first send and the increase in the overhead due to the redundant packets.

### 9.2.2. Same Room

This subsection will show the results of the same topology at a greater distance and only for Send Redundancy 2 which were the most interesting.

#### Send Redundancy 2

Figure 9.30 depicts the average packets sent per information packet, it's around the expected values but for the last 100 iterations' average that the encoding nodes seem to be closer to the optimal point than the recoding node, contradicting the theory.

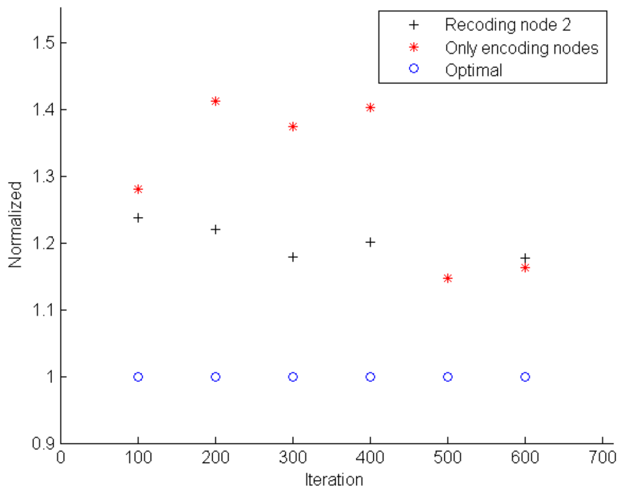


Figure 9.30.: Average K/N ratio each 100 iterations, Single Cluster Head Topology, SR2, Same Room

In the figure 9.31 the CDF of decoding the data depending on the group of packets sent is shown. It has a good decoding probability at first send but, despite that we can't say that all the generation will be successfully decoded after at least sending 5 group of packets.

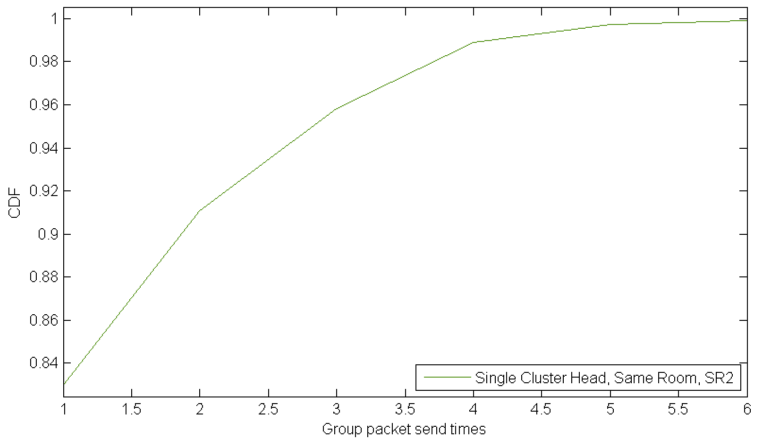


Figure 9.31.: CDF of successful generation decoding, Single Cluster Head Topology, SR2, Same Room

Here the comparison between nodes in figure 9.32 seems to be closer to what theory says, having the lowest overhead the node 2, with greater generation sizes as it's aggregating all the traffic. The other nodes behave the same, but it's noteworthy that the node 5 has much higher overhead.

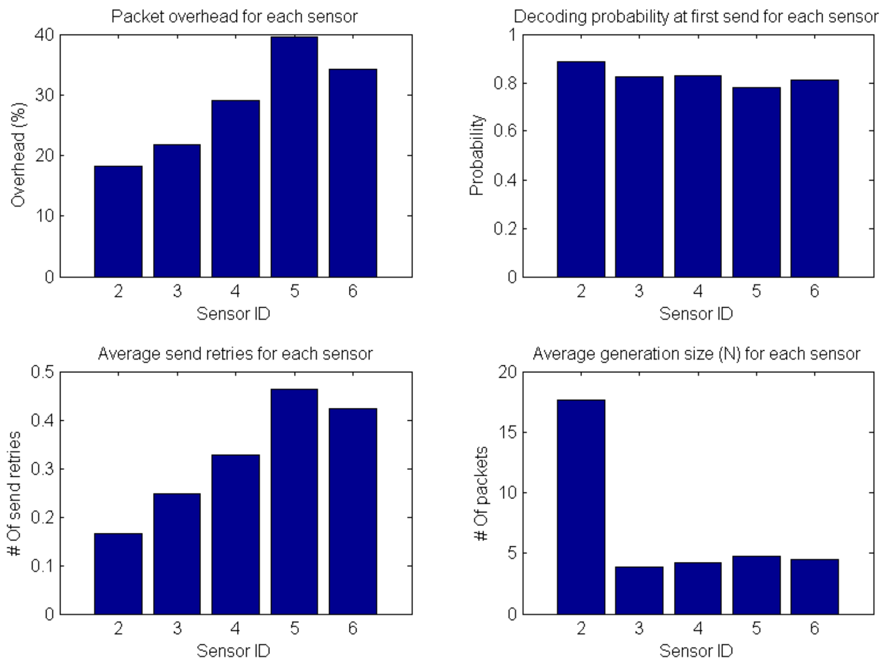


Figure 9.32.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Single Cluster Head Topology, SR2, Same Room

Having a fast look at the node 2's status plots in figure 9.33 the most remarkable fact is that the generation size is fluctuating during the first 450 iterations and seems a bit more stable after. The overhead decreases a bit during that last part of the test too. In figure 9.34 is shown the status of node 5, which was the one with the highest overhead is shown. It can be seen how the overhead decreases in the last part of the test as was predicted by the previous results.

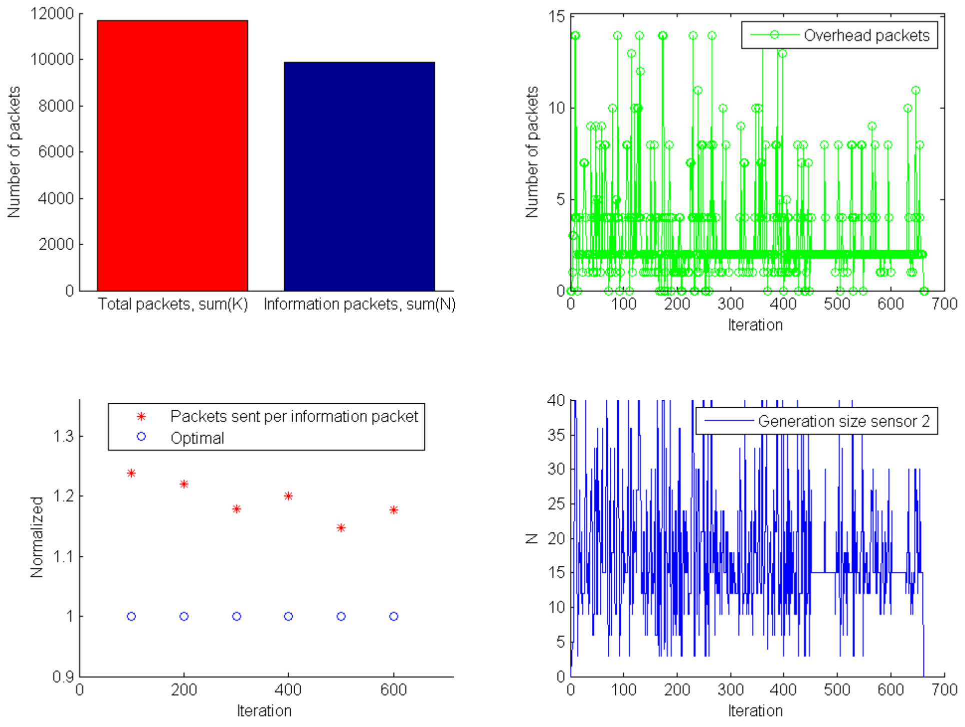


Figure 9.33.: Node 2's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Single Cluster Head Topology, SR2, Same Room

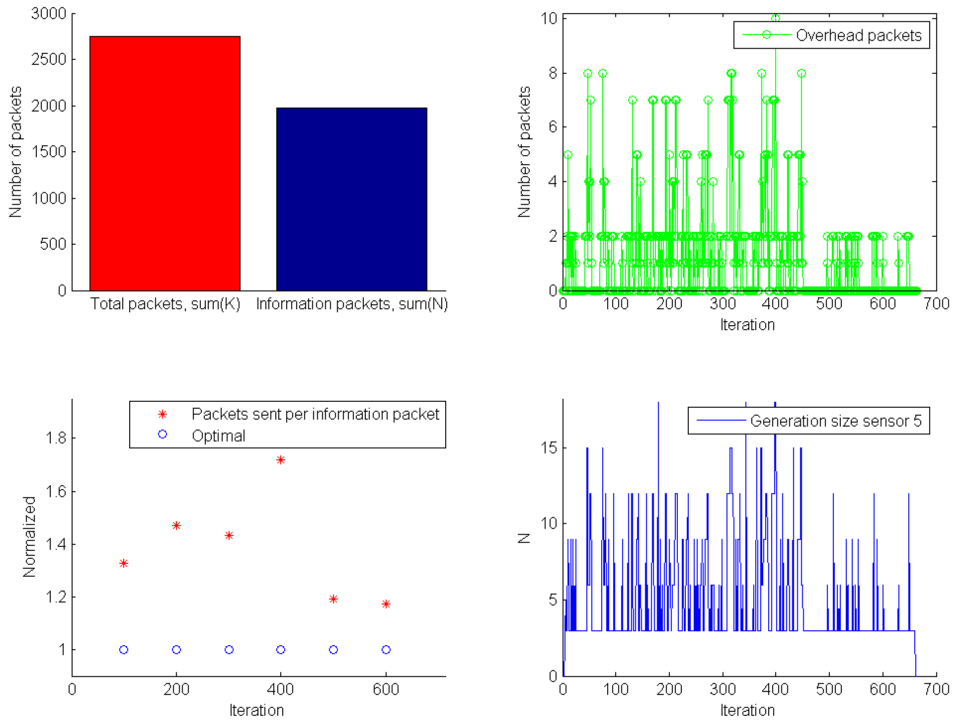


Figure 9.34.: Node 5's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Single Cluster Head Topology, SR2, Same Room

### 9.2.3. Different Rooms

In this last subsection the test will be performed with some of the sensors in different rooms and Send Redundancy 2.

#### Send Redundancy 2

Figure 9.35 shows that the networks shows the same lossy behavior in the first iterations again and seems to stop in the end too. In general during the lossy period the overhead of the recoding node is much smaller than the leaf nodes, but after the losses decrease the ratio looks much similar.

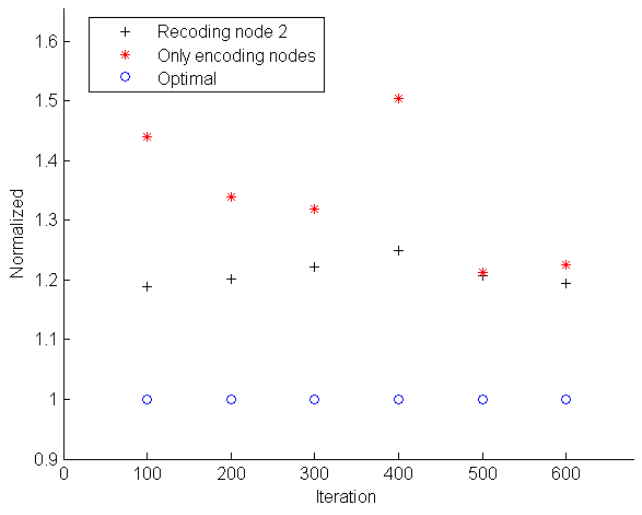


Figure 9.35.: Average K/N ratio each 100 iterations, Single Cluster Head Topology, SR2, Different Rooms

Looking at the CDF in figure 9.36 we can see that the decoding at first send is 78%, a bit smaller than in the previous tests as expected it is decreasing with the distance. After 4 groups of packets sent 98.5% of the generations were decoded, thus we could affirm that they will be decoded before sending those 4 groups.

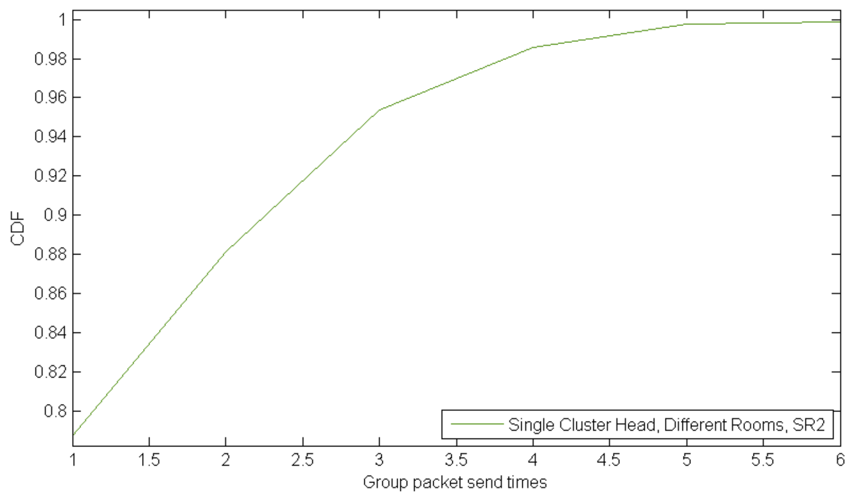


Figure 9.36.: CDF of successful generation decoding, Single Cluster Head Topology, SR2, Different Rooms

In figure 9.37 the theoretical forecast is fulfilled again, showing the smallest overhead for the node 2, and bigger overhead with the smaller probabilities of decoding at first send. Those probabilities are smaller than in tests at closer distances also.

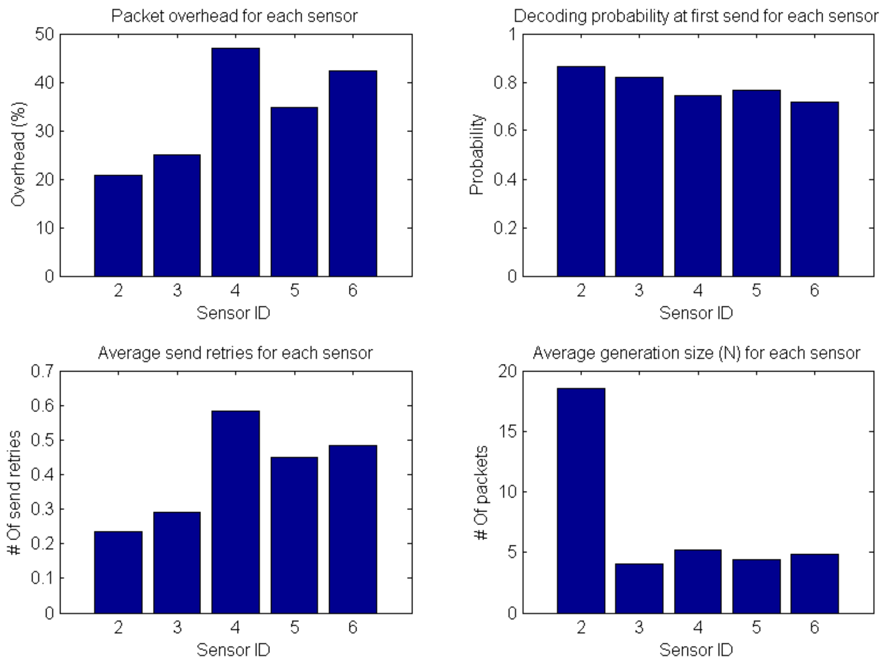


Figure 9.37.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Single Cluster Head Topology, SR2, Different Rooms

The generation sizes of the node 2 in figure 9.38 don't seem to stabilize in this test as happened in the one in the same room. The overhead seems to remain constant for the node 2 during the test. The node 4 had the greatest overhead, its status is shown in figure 9.39 where high spikes in the overhead packets and generation sizes are visible, those spikes are the visualization of the losses affecting this node during the test.

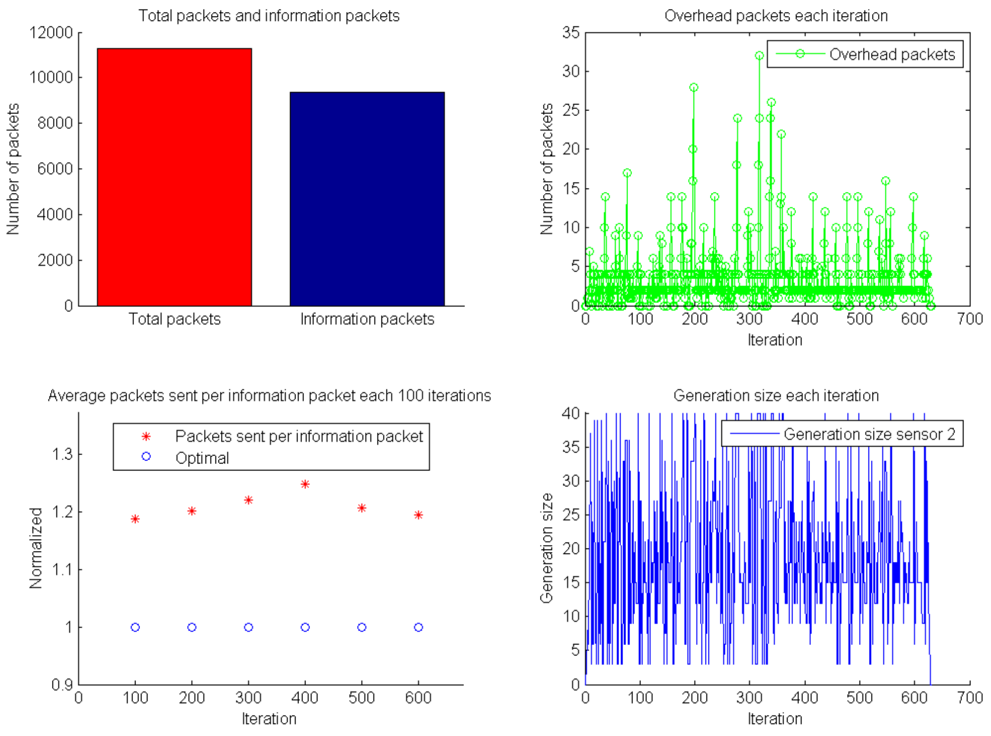


Figure 9.38.: Node 2's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Single Cluster Head Topology, SR2, Different Rooms

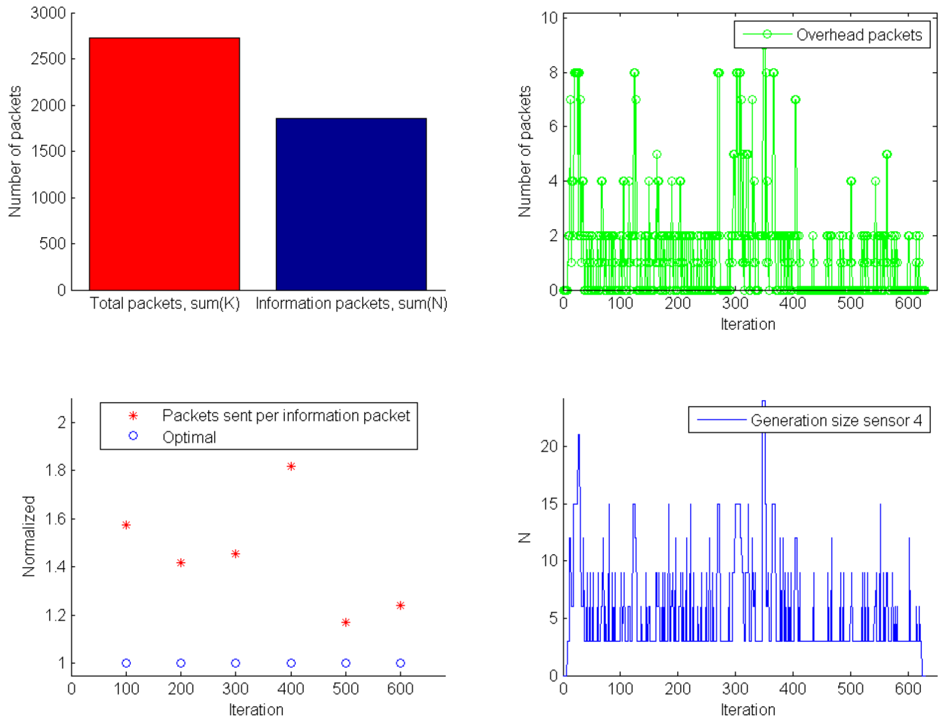


Figure 9.39.: Node 4's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Single Cluster Head Topology, SR2, Different Rooms

## 9.3. Dual Cluster Head Topology

This could be the most interesting of the topologies that have been tested during this thesis. Now two of the nodes are decoding the remaining three. This should in theory divide the data traffic between those two nodes, but as we will see, due to the high success rate the node that was started before, and thus process the data received before, will decode most of the data, in case that this node fails decoding the other decoding node will have a chance.

### 9.3.1. Together

#### Send Redundancy 0

Figure 9.40 represent the average packets sent per information packet each 100 iterations differentiating between the recoding nodes and the only encoding nodes, again having this last group the worst ratio, which gets worse in the last part of the test,increasing the gap.

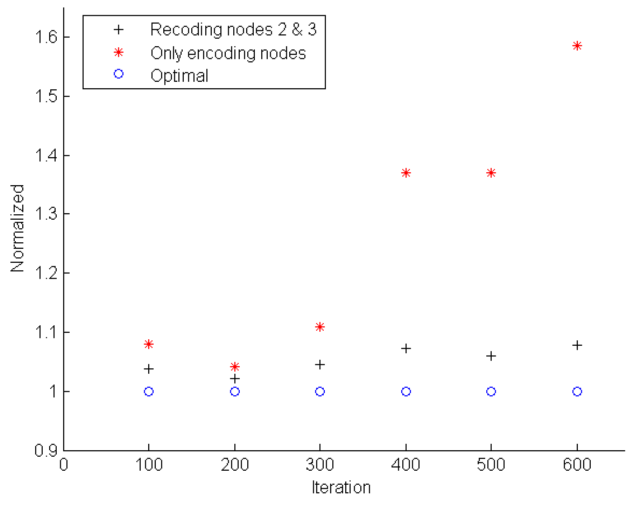


Figure 9.40.: Average K/N ratio each 100 iterations, Dual Cluster Head Topology, SR0, Together

The decoding probability for this topology is expected to be better than in the single cluster head topology, because now there are two nodes decoding the other three in stead of one decoding four. Figure 9.41 depicts the CDF of successful decoding, we can appreciate that the probability of decoding at first send has risen up to 84%, after the second group is sent most of the generations were decoded, around 95% and after 4 group send times all but for unlikely cases were successfully decoded.

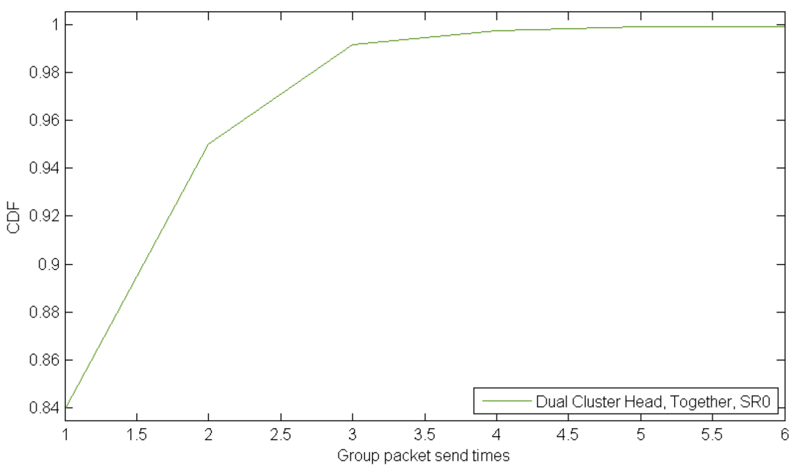


Figure 9.41.: CDF of successful generation decoding, Dual Cluster Head Topology, SR0, Together

In figure 9.42 the differences between the two groups of nodes are clearly visible. The nodes decoding and with direct link to the sink have much smaller overhead and better decoding probability. Looking into those two nodes a small difference can be seen, the node 3, which is decoding most of the packets from the leaf nodes, has a greater average generation size and despite having the same decoding probability at first send than the node 2 has smaller overhead. For the other group of nodes the 4 seems to have the worst link with the recoding nodes as it has much bigger overhead than the other two nodes and smaller decoding at first send probability.

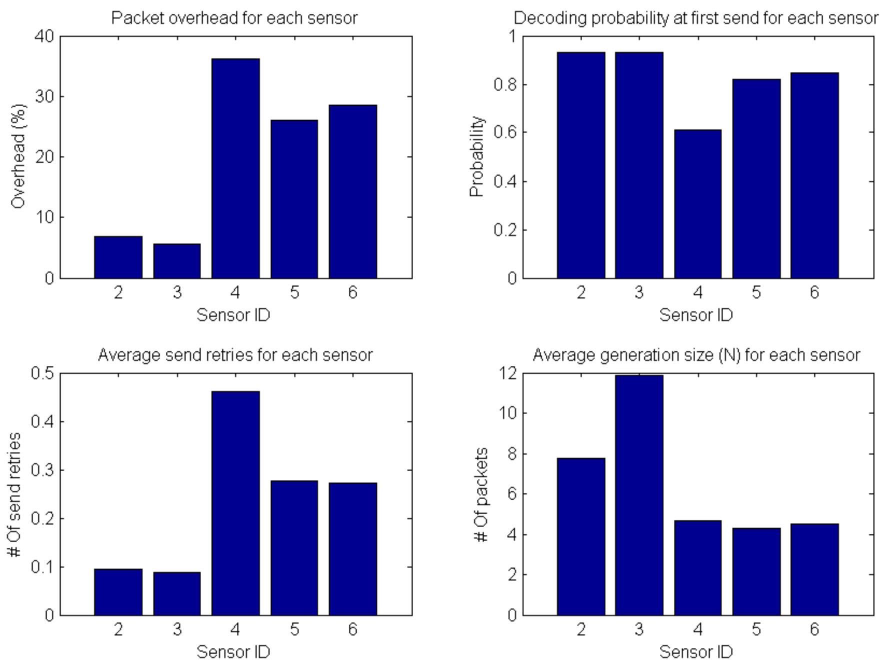


Figure 9.42.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Dual Cluster Head Topology, SR0, Together

Now we will compare both cluster heads, first figure 9.43 shows the status of node 2, the one that is decoding less data. As we can observe during the first part of the test where no losses appeared the generation size was almost always 3, the minimum, this means that the node 2 was not decoding none of the leaf nodes' data and it was only sending its own measures. But after the loss rate increased the generation size increased too, and as we will see in figure 9.44 it decreased for node 3 as now the traffic is more balanced between both nodes.

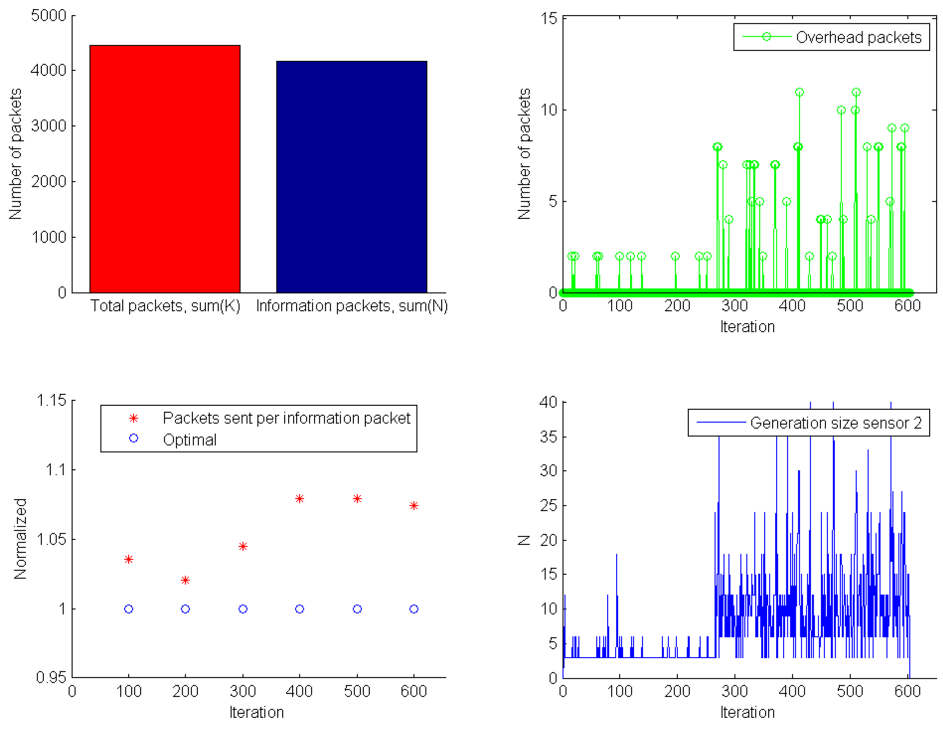


Figure 9.43.: Node 2's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR0, Together

As it was advanced before the generation sizes for node 3 during the first part of the test with low losses are rounding 12, which mean that the node 3 is sending its own data and the data belonging to the three leaf nodes as can be seen in figure 9.44. But after the losses increase, the generation sizes start to oscillate as the traffic is now divided between the two cluster heads.

In figure 9.45 is shown the status of the node 5, one of the leaf nodes, during the test. In the first 300 iterations the overhead is really low as the node was almost always decoded by node 3 and it that node failed decoding it node 2 had a chance to do it, thus the decoding probability at first send was really high during that first part. But after the iteration 300 burst of losses appeared, balancing the traffic between both recoders but increasing the overhead for all the nodes. Where in the first part of the test the generation sizes were almost always the minimum in the second part they grow up to 12, which means that at a time at least 3 send retries were needed to decode some of the generations.

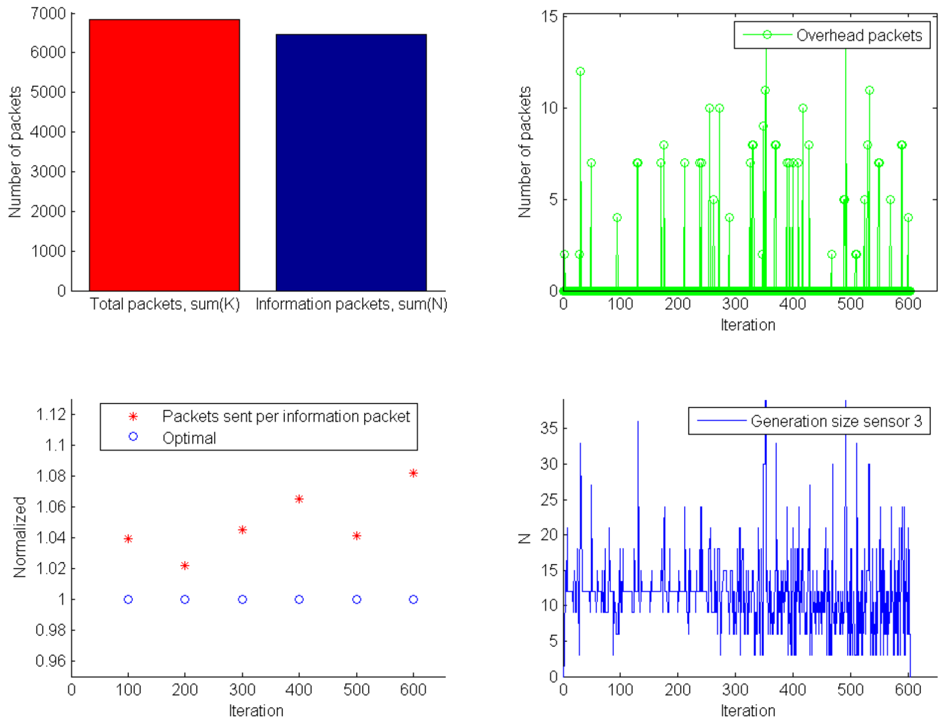


Figure 9.44.: Node 3's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR0, Together

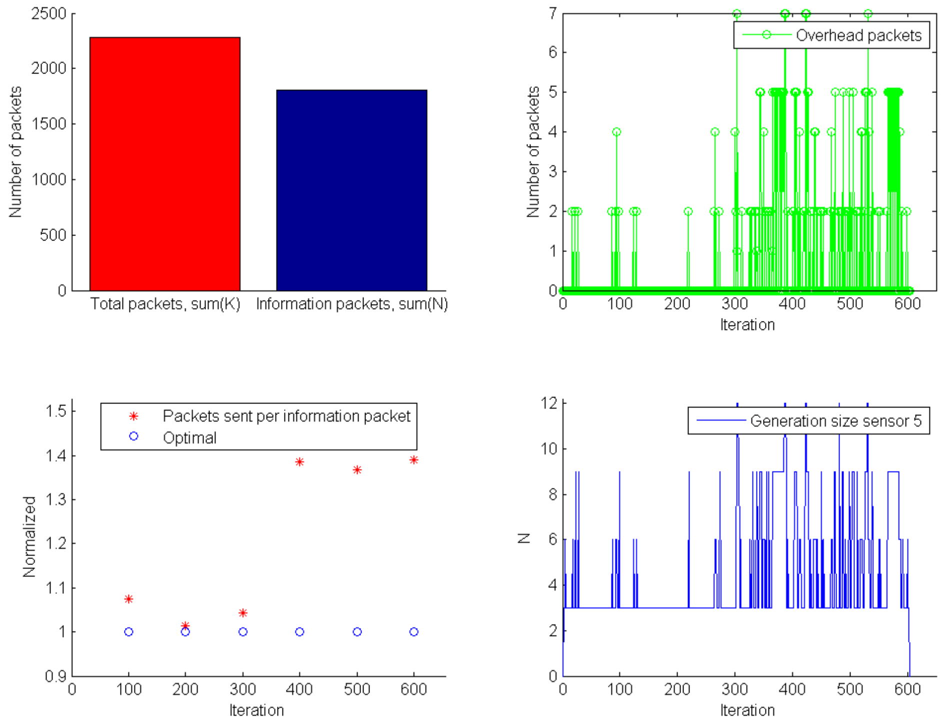


Figure 9.45.: Node 5's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR0, Together

### Send Redundancy 2

In this test happens a strange phenomenon, in figure 9.46 we can observe that the effect normally seen of the recoding nodes having better ratio than the only encoding ones is the opposite now. This can be due to the really high decoding at first send that we will see in figure 9.47 as the generation sizes for the leaf nodes are small no redundancy packets will be added. For the recoding nodes the generation sizes are greater then they will have that redundancy overhead.

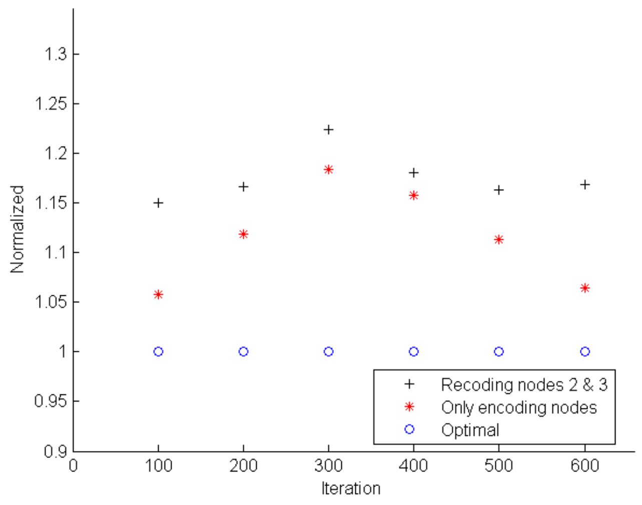


Figure 9.46.: Average K/N ratio each 100 iterations, Dual Cluster Head Topology, SR2, Together

Figure 9.47 shows the CDF of decoding successfully. With the send redundancy the probability of decoding after sending one group of packets has increased until almost 90%, but after has more or less the same probability as the one without it. We could consider all the generations decoded after 4 group sends.

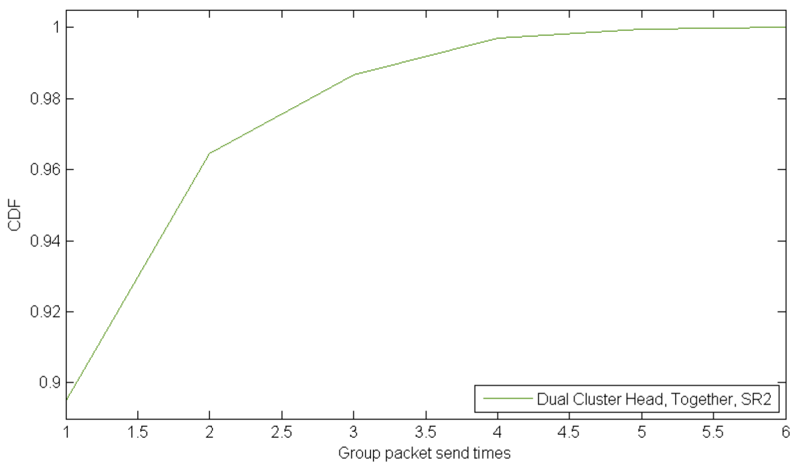


Figure 9.47.: CDF of successful generation decoding, Dual Cluster Head Topology, SR2, Together

Looking at the comparison between the nodes in figure 9.48 it is visible that for almost the same decoding probability the overhead seems greater for the recoding nodes, growing with the generation size. This can be due to the redundant packets sent in the first group. In this test the node 3 decoded almost all of the blocks coming from the leaf nodes due to the high success rate, as it is visible in the generation sizes chart.

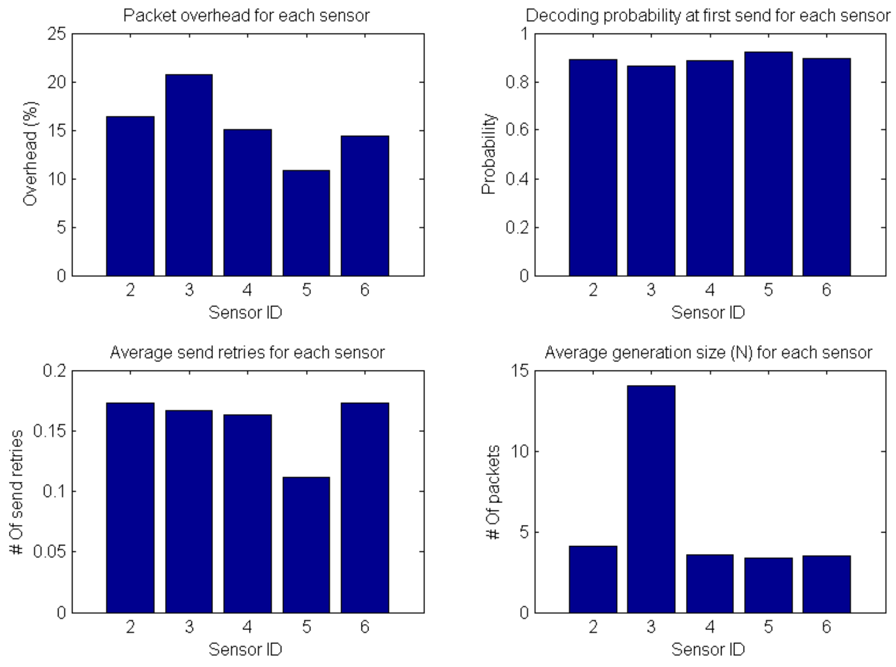


Figure 9.48.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Dual Cluster Head Topology, SR2, Together

Figure 9.49 shows the status of the node 3 during the test. It is remarkable that the floor of the overhead packets is almost always 2, the number of redundant packets sent for generation sizes between 10 and 24 (table 6.1), which can increase the overhead up to a 20% if the generation size is 10. That redundancy overhead and the one caused by the losses lead to the high ratios shown in the third plot. The status of the node 5, one of the leaf nodes, is represented in the figure 9.50. As we can see the generation sizes are in the minimum value for most of the iterations, which result in a small ratio between the packets sent per information packet.

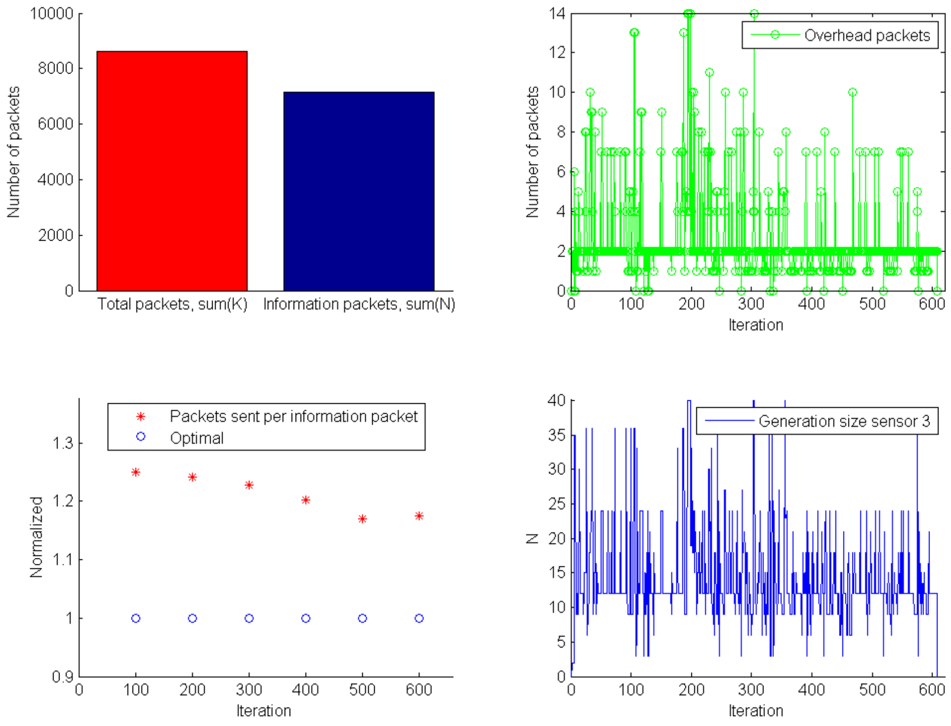


Figure 9.49.: Node 3's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR2, Together

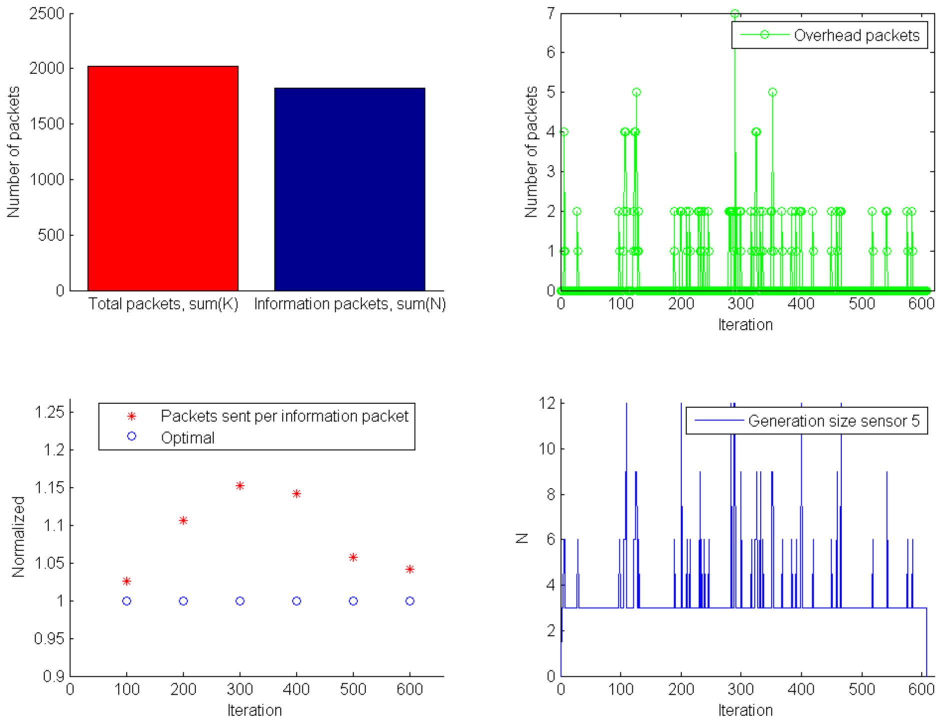


Figure 9.50.: Node 5's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR2, Together

### 9.3.2. Same Room

#### Send Redundancy 0

For this test the results were almost the optimal ones, as the decoding probability after one group send was really high, which led to a tiny overhead. In figure 9.51 we can observe that the ratios are for both kind of nodes almost the same not appreciating any remarkable difference.

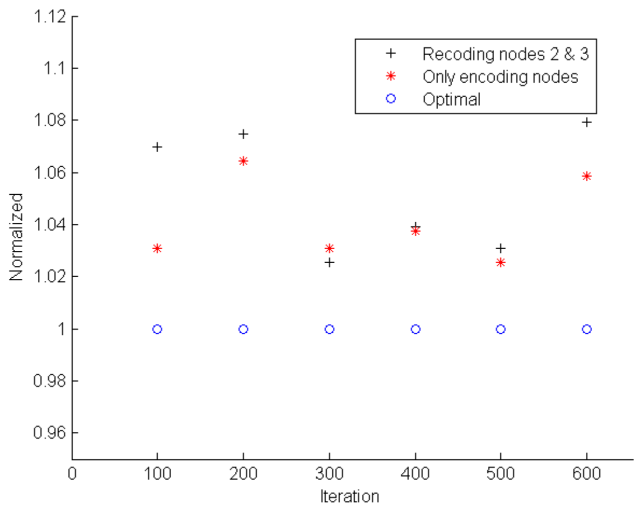


Figure 9.51.: Average K/N ratio each 100 iterations, Dual Cluster Head Topology, SR0, Same Room

Figure 9.52 is depicting the CDF of decoding successfully. It is visible that the probability of decoding at first group send is 94% and at the second group send 99% of the generations were properly decoded.

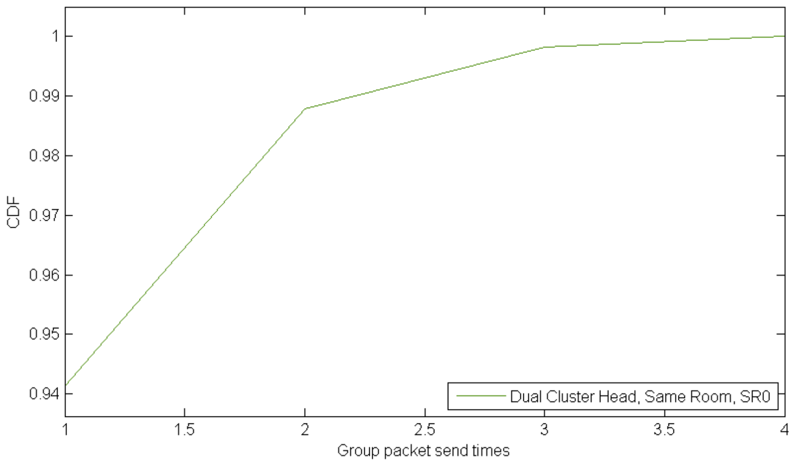


Figure 9.52.: CDF of successful generation decoding, Dual Cluster Head Topology, SR0, Same Room

In the figure 9.53, which shows the comparison between the nodes, we can see that the node 3 has the greater overhead and the lowest decoding probability at first send, this is because if only one of the packets sent, in average 14 (Generation Size), is lost another group send will be needed, and it's easier to miss one packet out of 14 than out of 3, which is the generation size for the rest of the nodes.

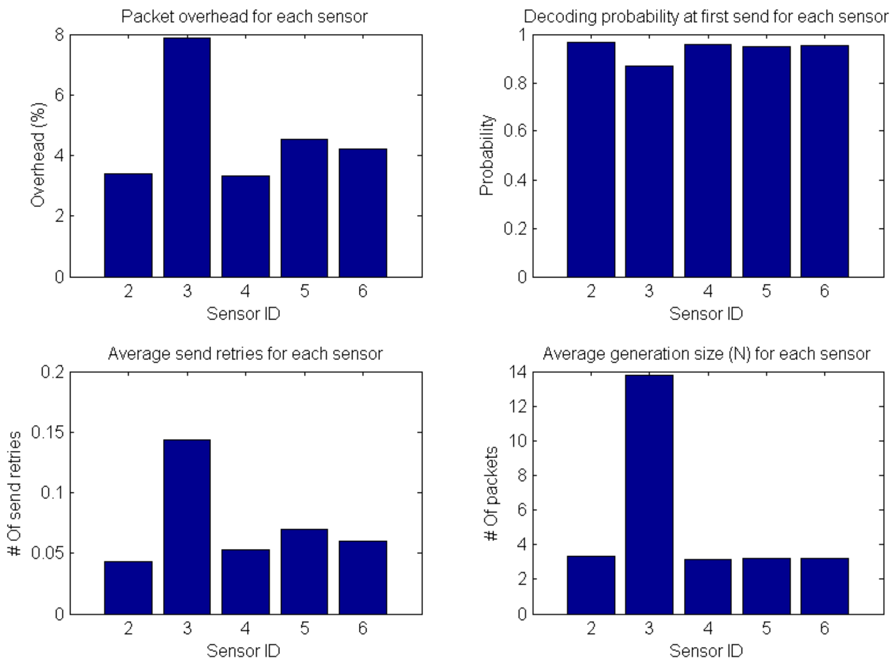


Figure 9.53.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Dual Cluster Head Topology, SR0, Same Room

When looking at the status of the node 3 during the test, which is shown in figure 9.54 we can see that most of the generations are sized 12 packets, which means that the three of leaf nodes were properly decoded by the node 3 in one send time and that node 3 was successfully decoded by the sink the generation before. If one of those conditions is not accomplished the generation size will vary. It is also visible that sometimes when one generation is not successfully decoded by the sink and the next one is bigger, it's more likely to fail when being decoded at first send. The figure 9.55 shows the status of the node 4 during the test. It is decoded almost always at the first send time, having the minimum generation size in most of the iterations. The overhead is really low because of that high successful decoding probability at first send.

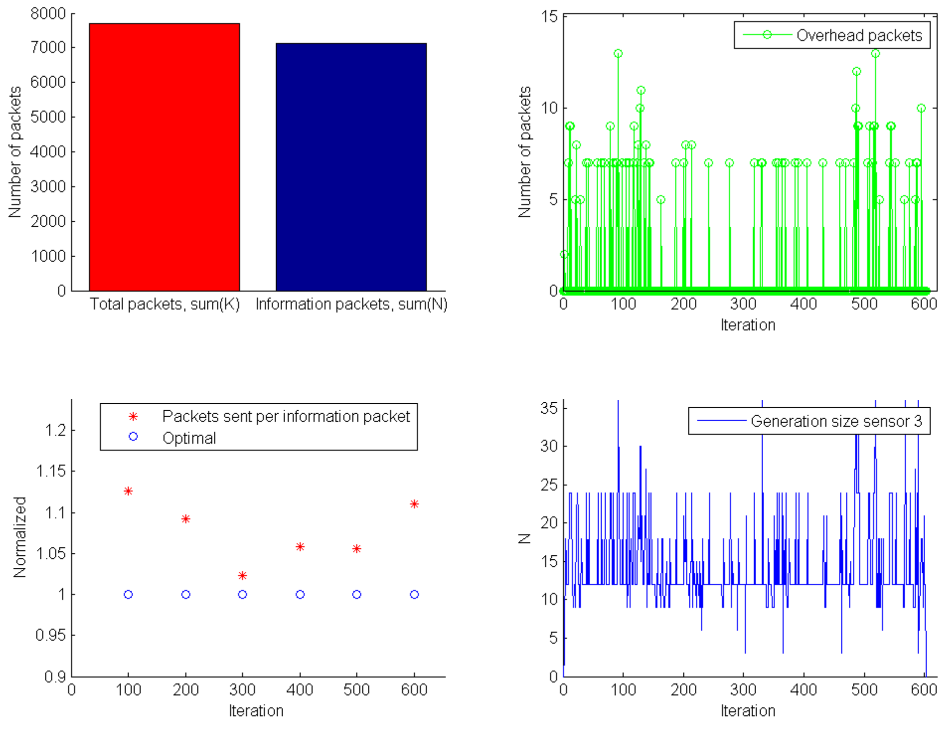


Figure 9.54.: Node 3's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR0, Same Room

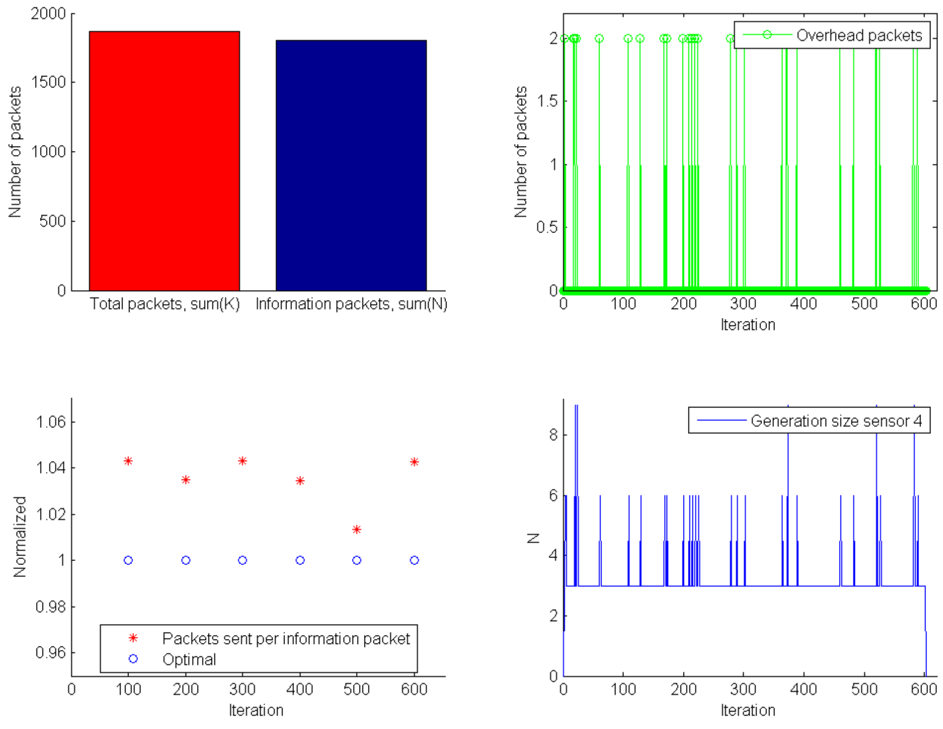


Figure 9.55.: Node 4's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR0, Same Room

### 9.3.3. Different Rooms

For the different rooms test only the results with Send Redundancy 2 will be shown.

#### Send Redundancy 2

Figure 9.56 shows the average packets sent per information packet each 100 iterations differing from the recoding nodes and the leaf nodes. In this test we have the same event where the recoding nodes have worse packet ratio than the encoding ones. This is due to the losses, as the leaf nodes have two nodes decoding they seem to have more chances to be decoded. This can be seen in the figure 9.58 where the average send retries for the recoders is greater than for the leaf nodes, as well as the decoding probability at first send is a bit bigger for that group.

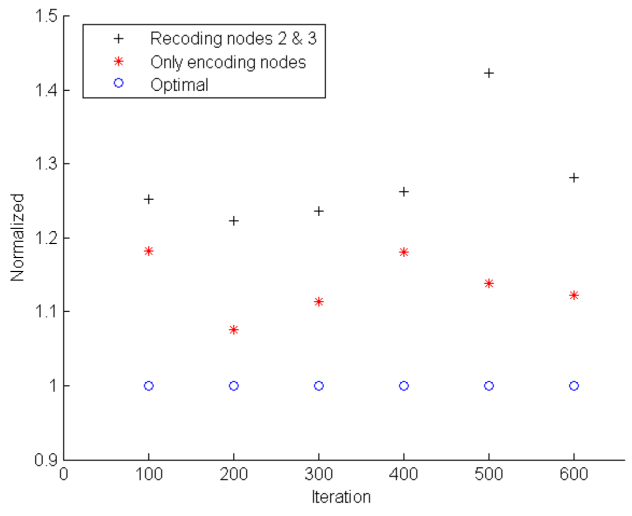


Figure 9.56.: Average K/N ratio each 100 iterations, Dual Cluster Head Topology, SR2, Different Rooms

In this test the decoding probability at first send is a bit smaller than in the previous one as can be seen in figure 9.57 which shows the CDF of successful decoding. After 4 group send times we could consider all the block successfully decoded.

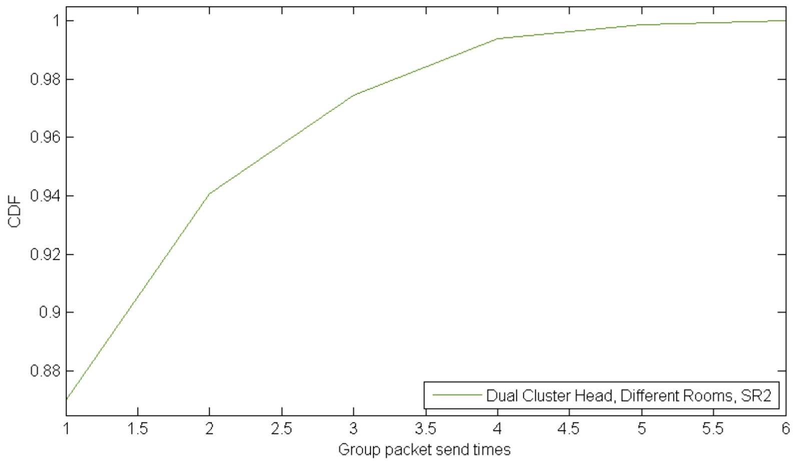


Figure 9.57.: CDF of successful generation decoding, Dual Cluster Head Topology, SR2, Different Rooms

Figure 9.58 depicts the comparison between all nodes. As said before the two decoding nodes have greater overhead than the other three. Looking at the send retries graph we can see that the nodes 2 and 3 have greater send retries than the leaf nodes which could be the cause of the overhead. Again the decoding nodes' traffic is unbalanced, this time having the node 2 most of the traffic.

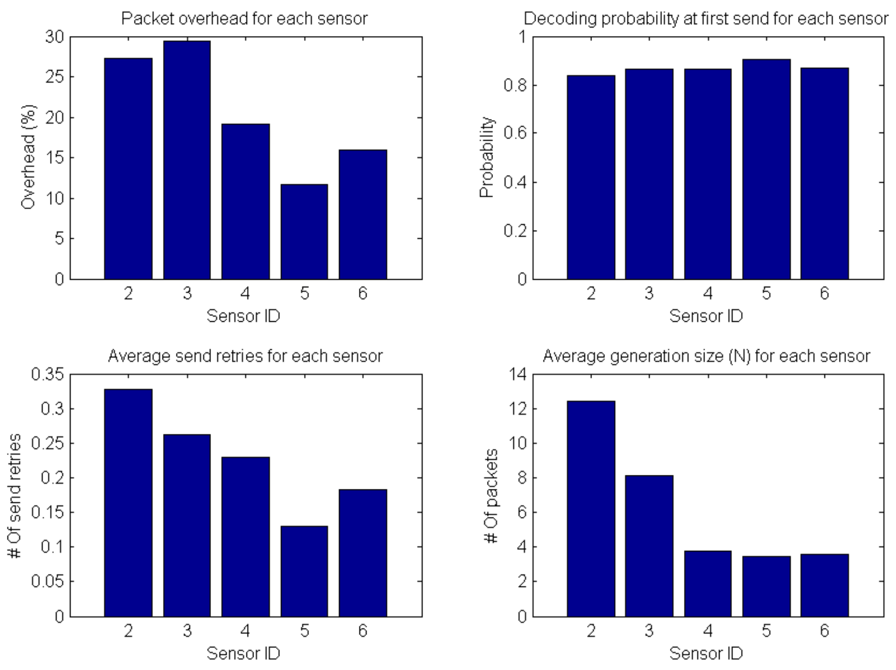


Figure 9.58.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Dual Cluster Head Topology, SR2, Different Rooms

The figure 9.59 represents the status of the node 2 during the test. The generation sizes are changing during the whole test due to the losses, which also affects the overhead packets increasing the amount of them as a lot of the generations need to be retransmitted.

As an example of the leaf nodes the status of the node 4 is represented in figure 9.60 we can appreciate that though the generation sizes are most time in the minimum value, they can rise up to 15, which means that the previous generation was decoded after 5 group send times. In fact if the look at the generation sizes they are shaped like high spikes, because one block fails to be decoded for two or three iterations and then a bigger generation is created and decoded successfully after a couple send times.

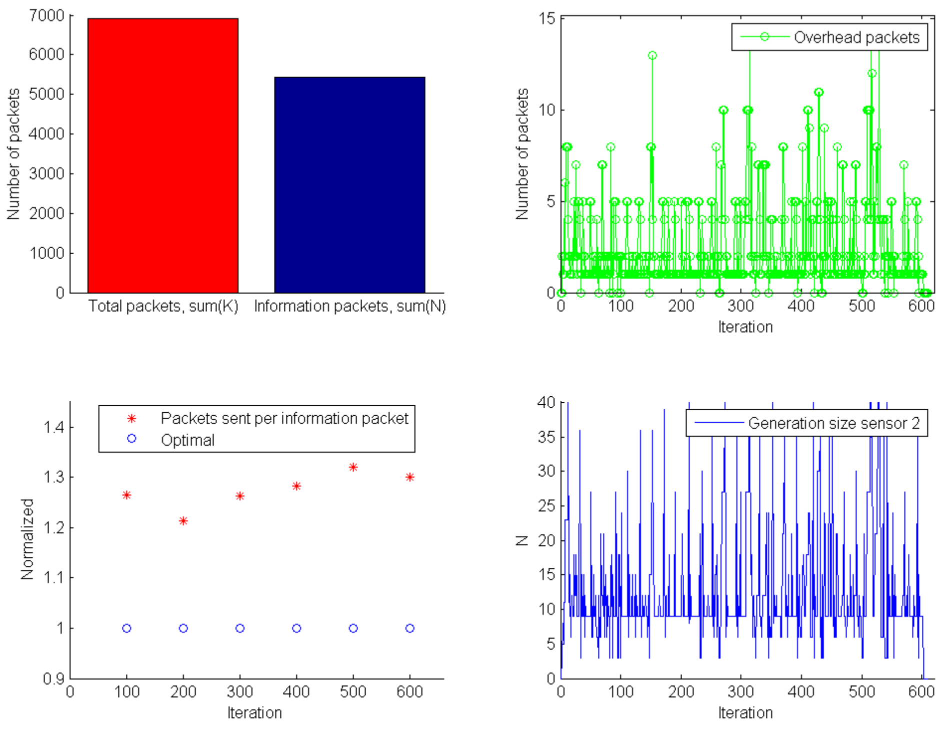


Figure 9.59.: Node 2's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR2, Different Rooms

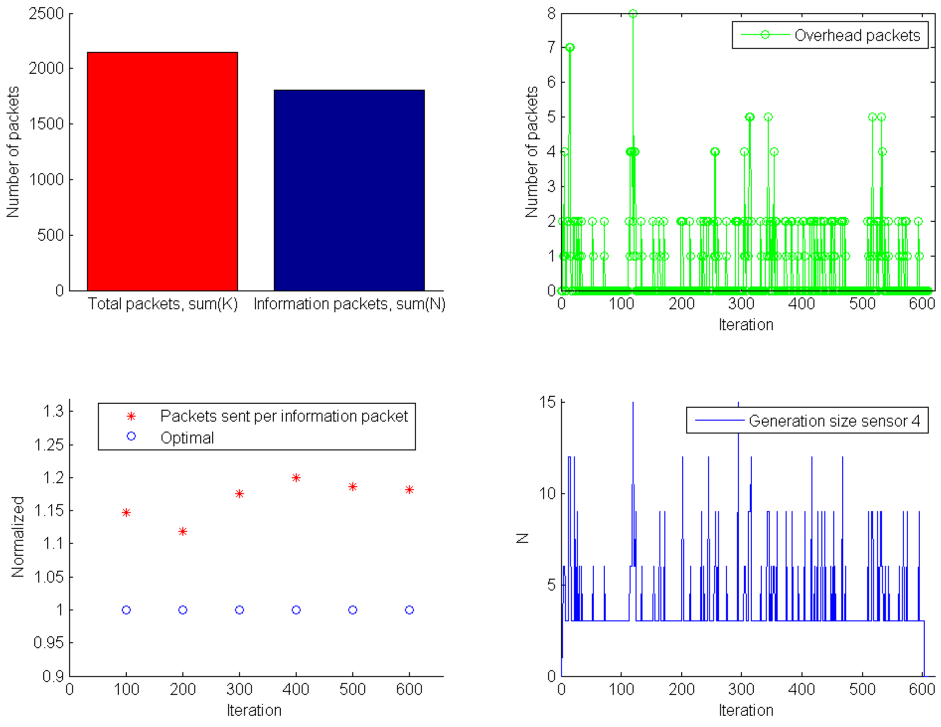


Figure 9.60.: Node 4's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR2, Different Rooms

## 9.4. Artificial Loss Rate

If we look at the previous results we can see that the loss rate of the channel should really low and the differences from one test to another mainly depended in the topology and the bursts of errors. That was an incentive to do some channel loss rate test, where 10000 packets were sent from one sensor to the sink and then the ratio between the received packets and the sent one was calculated. After 4 tests the results at the different distances were:

- Together: Received 9893/10000 : Success rate = 98.93%
- Room: Received 9610/10000 : Success rate = 96.10%
- Different rooms: Received 9480/10000 : Success rate = 94.80%
- Max sensor range: Received 9822/10000 : Success rate = 98.22%

With such high success rates, network coding couldn't show it's potential, so an artificial channel loss rate was introduced. For the following tests the nodes and the sink have a 40% of chances of discarding each received packet, even the ACK packets. This artificial loss rate is computed after the real one. The same topologies will be analyzed with this new loss rate.

The tests in this section were suppose to be one hour and half long, 5400 seconds which will make 900 iterations.

### 9.4.1. Star Topology

As it has been explained in the sections before in this topology all the nodes have the same role and are directly connected with the sink and have the same error rate, thus the results are expected to be the same for each sensor.

Figure 9.61 shows the average packets that had to be sent for each information packet they are all rounding 2.3, around two times the ratio that was measured in the sections before, and it seems to be plain during the test as the link losses are always the same.

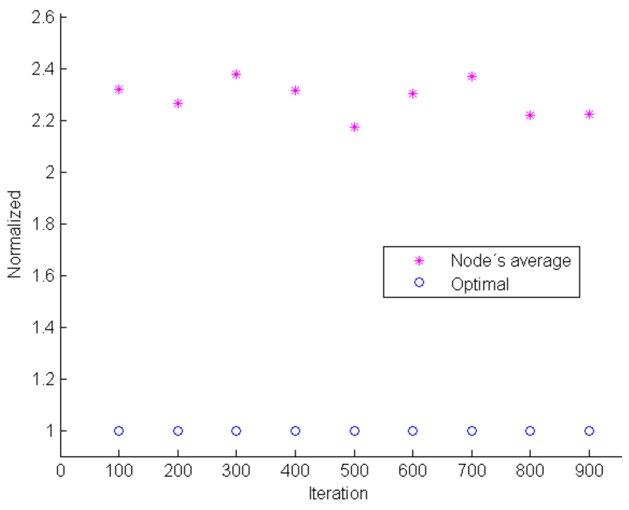


Figure 9.61.: Average K/N ratio each 100 iterations, Star Topology, SR0, 40% Loss

The figure 9.62 shows the CDF of decoding successfully according to the group of packets send times, as it was expected the probability of decoding at first send is really low with 40% channel losses, all the packets sent should be properly received in the sink, as the first send is systematic no linear dependencies will appear. Receiving all the packets will have a probability of $0.4^N$, the greater the generation size, the lower the probability will be. We can suppose that the first point at 1.5% probability should have been a small generation size. We can appreciate a huge jump from the first to the fourth send time reaching up to 91% of successful decoding chances. At the fifth send time the decoding probability is 97.5% allowing us to almost ensure that the block of data will be decoded after those 5 block send times.

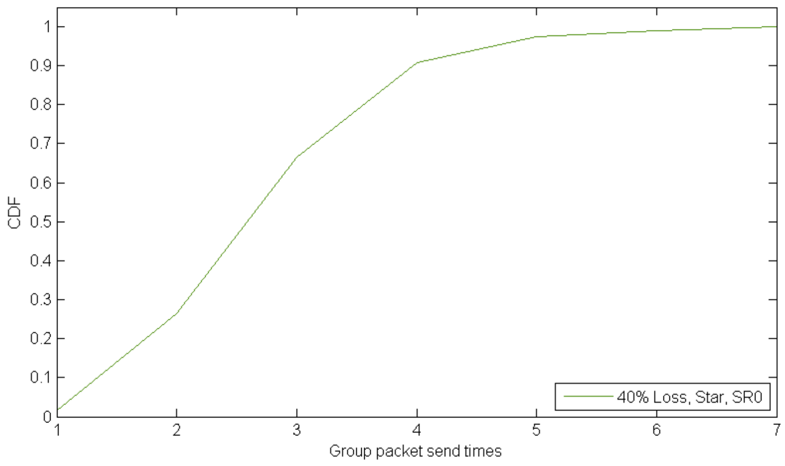


Figure 9.62.: CDF of successful generation decoding, Star Topology, SR0, 40% Loss

Figure 9.63 shows some parameters of all the nodes, as said they look all the same due to their role in the network. The first plot shows an average 120% packet overhead for all the sensors. As we could see in the figure 9.62 the chances of decoding at first send are really low which is also visible in the second chart. The third graphic shows the average send retries of each sensor, they are all around 2.3, which would explain the average generation size too, as they need around 3 or 4 iterations to send the block, 9-12 measurements should be added to the data queue rising the average generation size to almost 10.

The figure 9.64 shows the status of the node 3 during the test as it was the one with a slightly higher overhead. If we compare this plots to the ones in previous sections with much higher success rate we can appreciate a huge increase in the overhead packets in both first and second graphs. Whereas in the figure 9.4 the generation sizes were almost always 3, the minimum, in this plot is really unlikely to see it reaching that value.

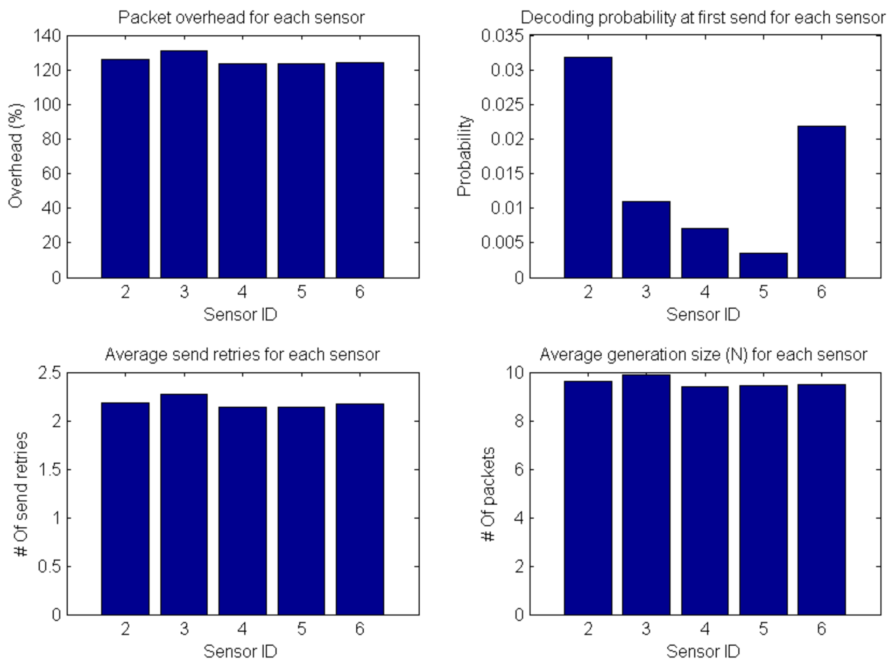


Figure 9.63.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Star Topology, SR0, 40% Loss

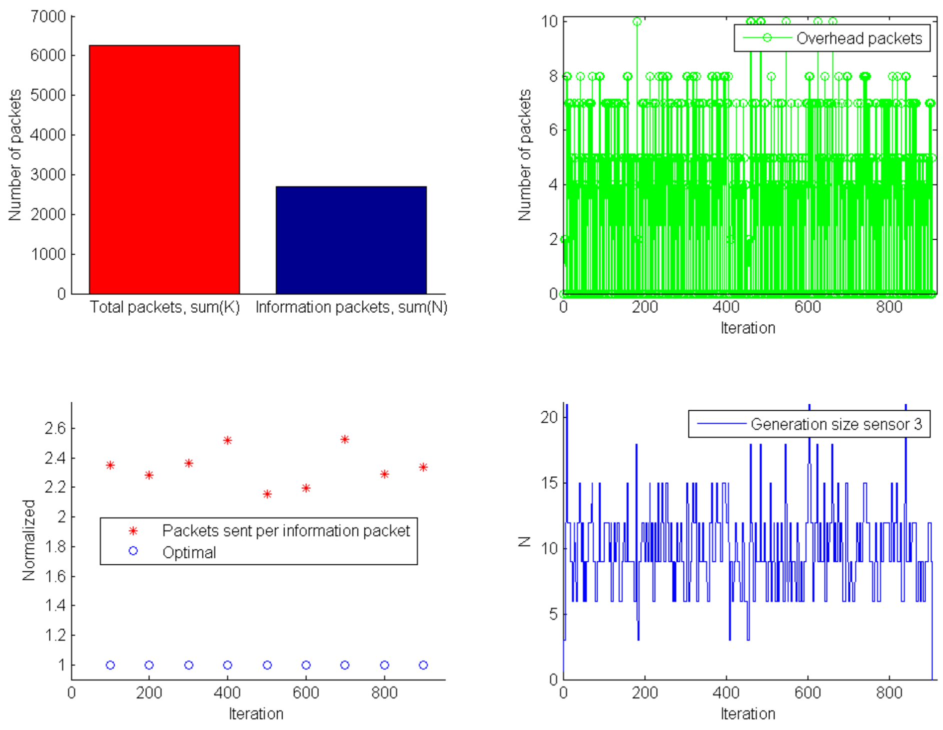


Figure 9.64.: Node 3's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Star Topology, SR0, 40% Loss

### 9.4.2. Single Cluster Head Topology

This second test was supposed to last one hour and half like the other ones, but it ended much before because the node 2 which was supposed to decode and recode the data coming from the rest of the nodes ran out of memory and crashed. That was caused because the outgoing link to the sink had not enough capacity to get rid of all the packets created by the sensors, creating a bottle neck that made the sensor 2 store all the packets until it filled all its memory provoking a heap and stack collision. That amount of data waiting to be encoded will be reflected in figure 9.68 where the generation size won't be smaller than 40, meaning that there will always be more than 40 packets / data waiting to be encoded. The figure 9.65 only shows the first 100 iterations because the node 2 crashed before reaching the 200. As we can see the node 2 has a bit better ratio between the packets sent per information packet than the rest of the nodes. That is because network coding works better with bigger generation sizes and the node 2 was aggregating the traffic from the rest of the nodes.

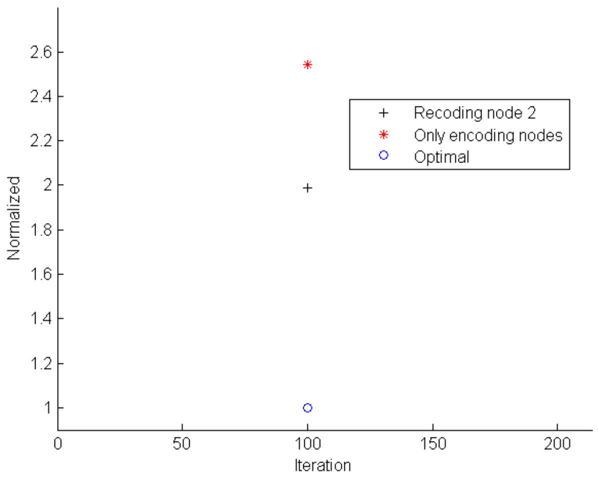


Figure 9.65.: Average K/N ratio each 100 iterations, Single Cluster Head Topology, SR0, 40% Loss

Figure 9.66 looks a lot like the one in the star topology. The decoding at first send in this case is a bit higher, 2%, the reason is probably the small number of iterations of this test as the probability of decoding at first send are more likely to happen in the beginning when the generation sizes are small. After 5 group of packets send times the probability of decoding successfully is almost 95%, we could ensure that the block will be decoded at 5 send times or before.

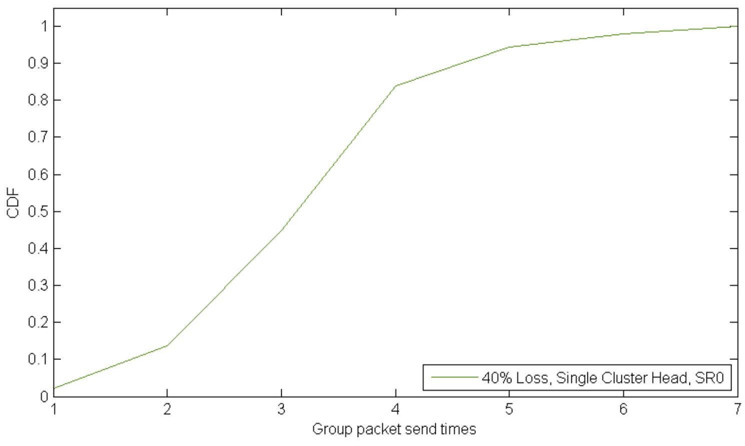


Figure 9.66.: CDF of successful generation decoding, Single Cluster Head Topology, SR0, 40% Loss

The figure 9.67 shows the differences between the sensors. There is a big difference between the node 2 and the rest as they play distinct roles in this topology. The overhead for the node 2 is 100%, fact that was anticipated by figure 9.65 as the ratio of 2 means that in average you have to send 2 packets per information packet, in other words 100% overhead. The rest of the nodes are rounding 150% packet overhead. The probability of decoding at first send is almost 0 for all the nodes as it's really hard to receive all the packets with 40% loss rate. Looking at the third graph we can see that the number of send retries doesn't seem to be related with the generation size but with the channel losses. The fourth plot shows the average generation size, the second sensor has almost the maximum generation size, proving that it was overloaded. The rest of the nodes have approximately the same generation size than in the star topology.

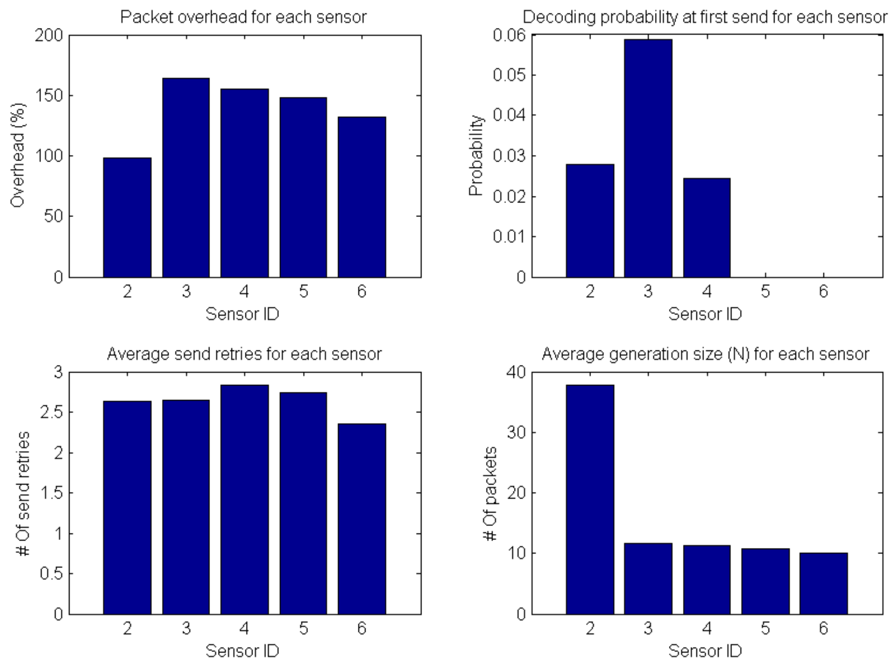


Figure 9.67.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Single Cluster Head Topology, SR0, 40% Loss

Figure 9.68 depicts the status of the node 2 during the test, in these charts is clearly visible that the node was overloaded as the generation size does not go under 40 after reaching it.

The figure 9.69 shows the status of one of the leaf nodes during the same test, the node 6 in this case. Compared to the previous figure it's clearly visible that the generation sizes are much smaller as it only has to send it's own information.

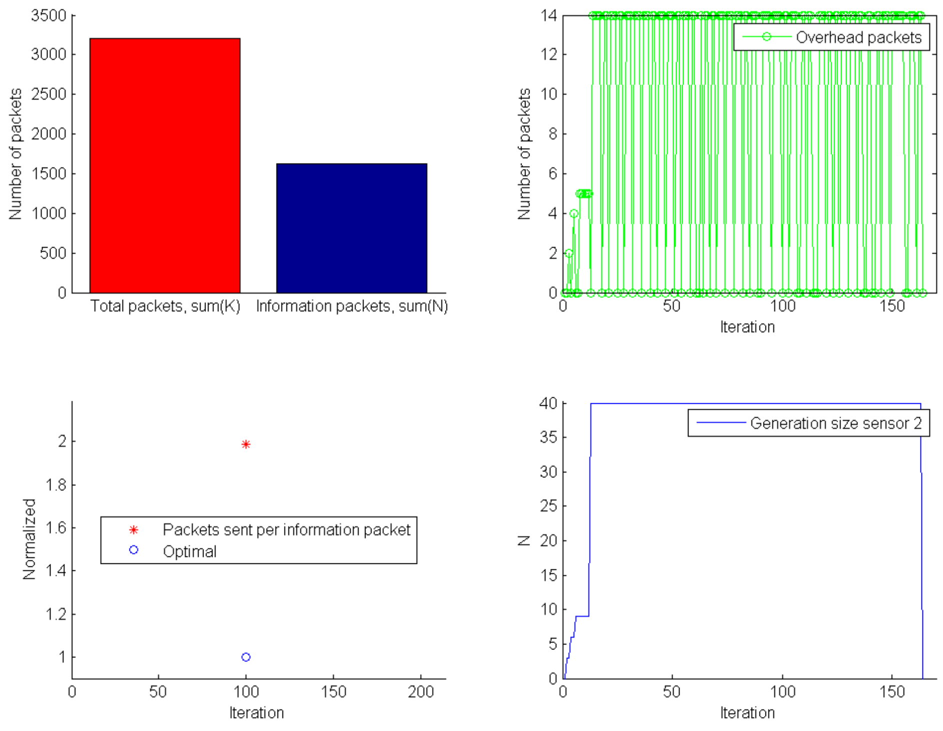


Figure 9.68.: Node 2's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Single Cluster Head Topology, SR0, 40% Loss

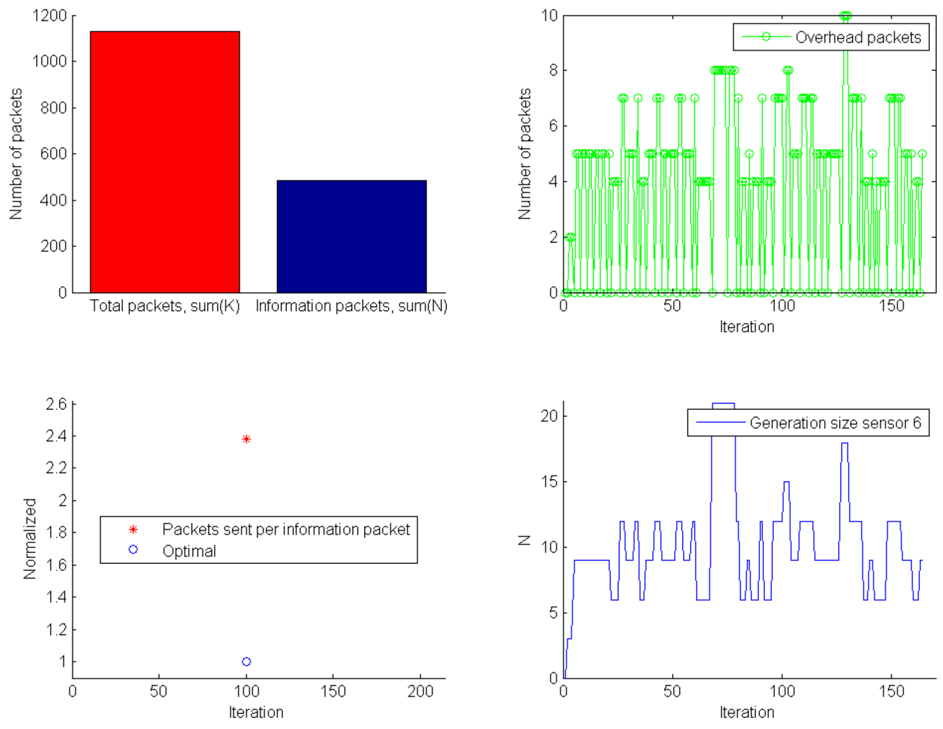


Figure 9.69.: Node 6's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Single Cluster Head Topology, SR0, 40% Loss

### 9.4.3. Dual Cluster Head Topology

In this last section the topology with two cluster heads will be analyzed. As we have seen one node does not have enough capacity to send all the information coming from four nodes, in this last topology we will show that two nodes can send properly the information coming from three of the nodes and that, compared with the dual cluster head test before in this case with higher loss rate, the number of decoding packets is better distributed.

Figure 9.70 shows the difference in terms of packets sent per information packet between the nodes aggregating the traffic and the ones that only send their information, as explained before network coding shows it's best with greater generation sizes, that's the reason why the ratio for the decoding nodes is better than for the rest. That ratio is a bit worse than in the case before (Figure 9.65) because the average generation size for those nodes is smaller in this test as they are not overloaded.

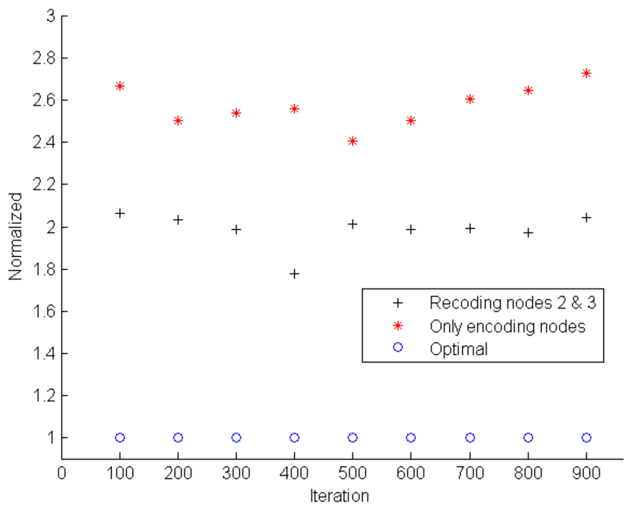


Figure 9.70.: Average K/N ratio each 100 iterations, Dual Cluster Head Topology, SR0, 40% Loss

The figure 9.71 shows the CDF of decoding probability depending on the group send times. As is common in this artificial loss rate section the probability of decoding at first send is really low 0.7%. We can appreciate a huge jump in the probability rising up to 82.52% for four send times and we can almost ensure the block decoding at 6 send times with 98.68%.

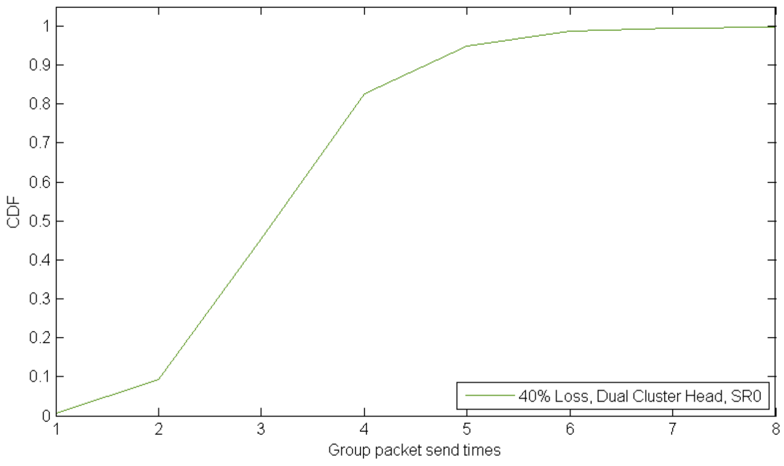


Figure 9.71.: CDF of successful generation decoding, Dual Cluster Head Topology, SR0, 40% Loss

In figure 9.72 the differences between the decoding nodes and the only encoding is clearly visible. The packet overhead for the nodes in the first group is around 100% whereas for the second group is around 150%, proving again the efficiency of network coding with bigger generation sizes. Again the probability of decoding at first send is close to 0% for all nodes due to the high losses of the channel. The average send retries are again at the same values than in the test before, in the range between 2.5 and 3, seeming to be a bit smaller for the decoding nodes than for the only encoding ones. As we can see in this test the generation sizes of the node 2 and 3 are not so different as in the previous tests of 2 and 3 decoding where mainly one node decoded all the packets. The generation sizes are close to the maximum, even more in the case of the node 3, which could be a problem causing one of the nodes to start overloading and crashing as happened in the test before. The other group of nodes have an average generation size of 10, which was the same as in the other tests with the artificial loss rate.

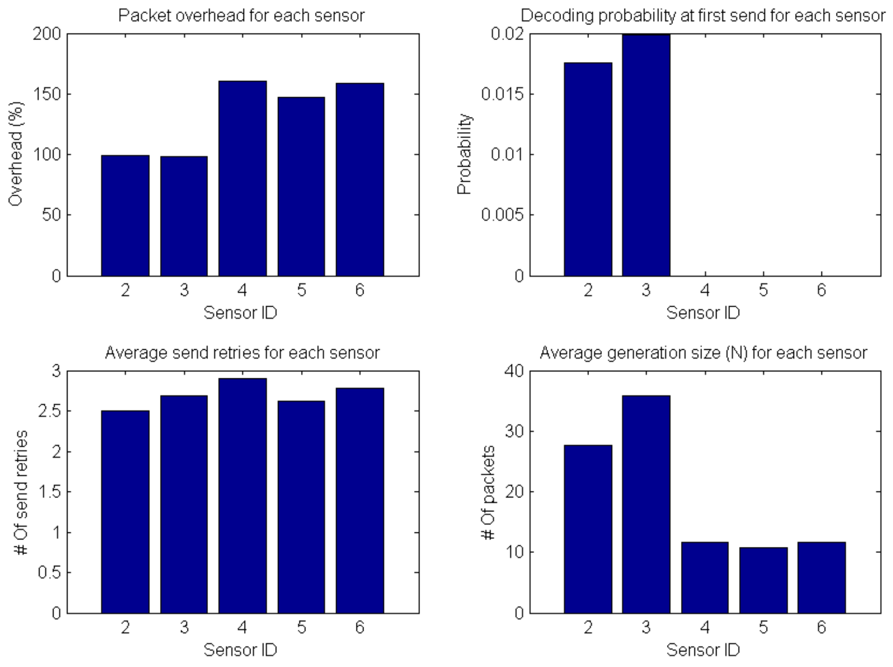


Figure 9.72.: Comparison between nodes' overhead, decoding at first send probability, average send retries and average N, Dual Cluster Head Topology, SR0, 40% Loss

Figure 9.73 shows the information gathered about the status of the node 3, the decoder that seemed to decode most of the packets. The ratio between the sent packets and information packets is always around the expected value 2. If we have a look at the generation sizes it's visible that the node can reach the load limit as it has the maximum generation size an important part of time. But the main difference with the previous test with only the node 2 decoding is that in this chart we can see that sometimes the generation size goes under 40, proving that the node was capable of emptying its encoding queues, event that didn't happen in figure 9.68.

To compare one of the only encoding nodes with the ones decoding we have the figure 9.74 which shows the status of the node 4, which seemed to have the greatest overhead of the three. The most remarkable thing in these figures is that the generation size is rounding one quarter of the maximum one, thus we can say that the nodes are not under heavy load and are being properly decoded.

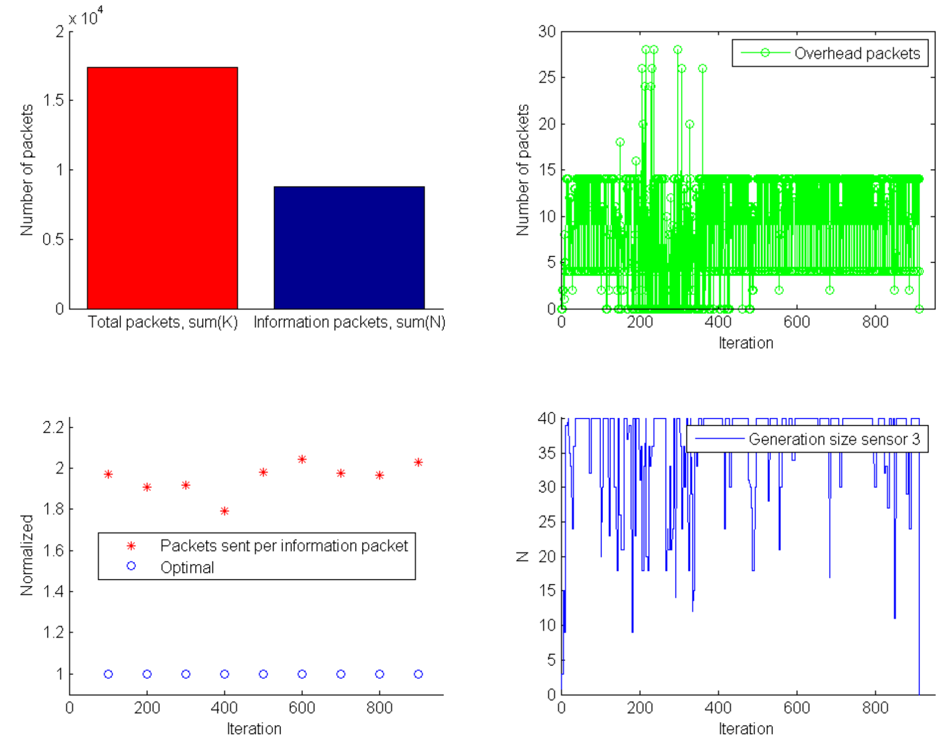


Figure 9.73.: Node 3's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR0, 40% Loss

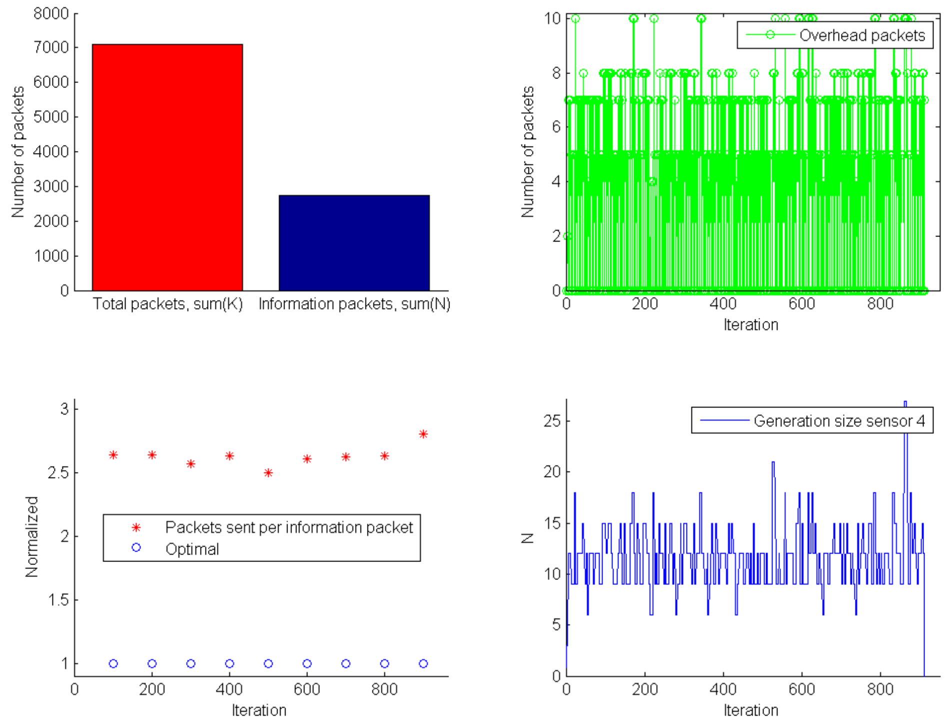


Figure 9.74.: Node 4's packets sent, overhead packets per iteration, average K/N ratio, generation size per iteration, Dual Cluster Head Topology, SR0, 40% Loss

## 9.4.4. Comparison Between Topologies

In this last section of the results chapter the differences between the topologies will be discussed. For this comparison we used the results from the previous section but for the case of the single cluster head topology, as said the node 2 crashed, thus the test was repeated with only three nodes being encoded by node 2, this change didn't turn the network stable but it allowed the node 2 to work for at least the hour that was used for taking measures. As those measures for single cluster head topology only lasted one hour whereas the measurements for the other two topologies were an hour and half long the data for all to 2 topology will end in the 600 iterations in stead of the 900 of the other two.

Figure 9.75 shows the comparison between the average packets sent per information packet in each topology. As expected no big differences over time are seen as the losses are introduced artificially and don't change over time. The overhead for all the topologies looks quite similar, between the 2.2 and 2.5 ratio.

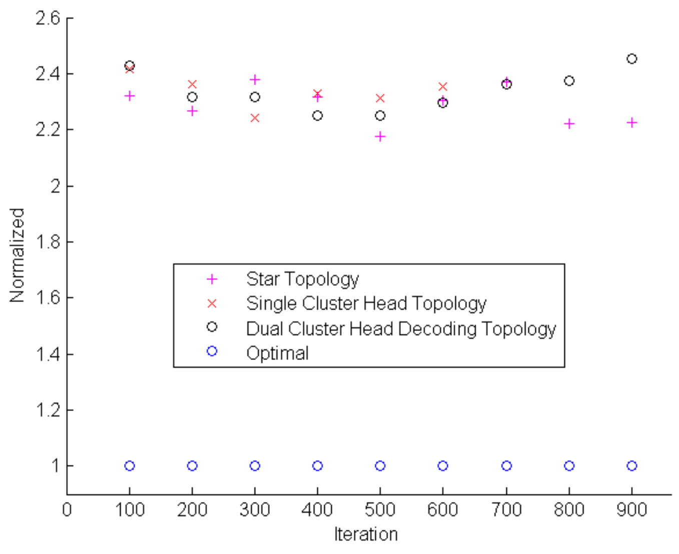


Figure 9.75.: Average K/N ratio each 100 iterations, Comparison, 40% Loss

The figure 9.76 shows the different CDFs for each topology. None of the topologies have a decent probability of decoding at first send, but all of them decode almost all of the blocks after sending 5 groups of packets. We could have expected the dual cluster head topology to have the best decoding capabilities as it has two decoders in stead of one as the single cluster head topology has. The reasons for this event to happen are the same for explaining why the star topology has a better CDF than the other two topologies. The number of packets sent after each non-confirmed group send were meant for small loss rates, then the greater the generation size is, the smaller is the relation between the generation size and the coded packets sent (Table 6.3) thus with greater average generation sizes and high loss rates the groups of packets sent needed for decoding are bigger. The second reason is that the sink seems to decode better than the wireless nodes.

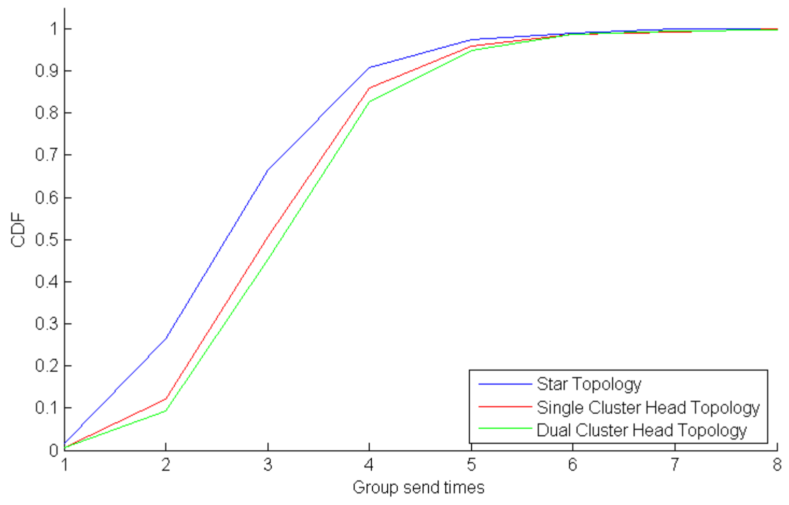


Figure 9.76.: CDF of successful generation decoding, Comparison, 40% Loss

To check the affirmation said before to explain the differences between the decoding probabilities we should look at the figure 9.77 which shows the generation sizes for each sensor and each topology. Keep in mind that for the single cluster head topology one of the leaf nodes was unused, that's why the node 4 has no bar in that test. According to what was said before the generation sizes are bigger in the dual cluster head topology than in the single cluster head topology, and bigger in that one than in the star topology. In this charts is also visible how the node 2 and 3 divide equitably the traffic from the leaf nodes in the dual cluster head topology.

The figure 9.78 shows information regarding the node 2 in the different topologies. The most remarkable information that can be extracted from these charts is the relation between the overhead and the generation sizes shown before, where it's apparent that the bigger the generation size is the smaller the overhead becomes.

In figure 9.79 a comparison of the node 3's behavior along different topologies is shown. In the first two topologies the node 3 acts as a leaf node, while in the last one it becomes one decoder for the other 3 nodes. The apparent bigger overhead in the leaf nodes is remarkable in the second and fourth charts, where the maximum overhead appears for the single cluster head topology with up to 164%.

Figure 9.80 represents the status of the node 5 in the different topologies. It was always playing the same role of leaf node, thus only small differences are visible from one topology to another, showing a slightly greater overhead in the topologies where another node did the decoding of the data, as well as worse decoding probability in those topologies.

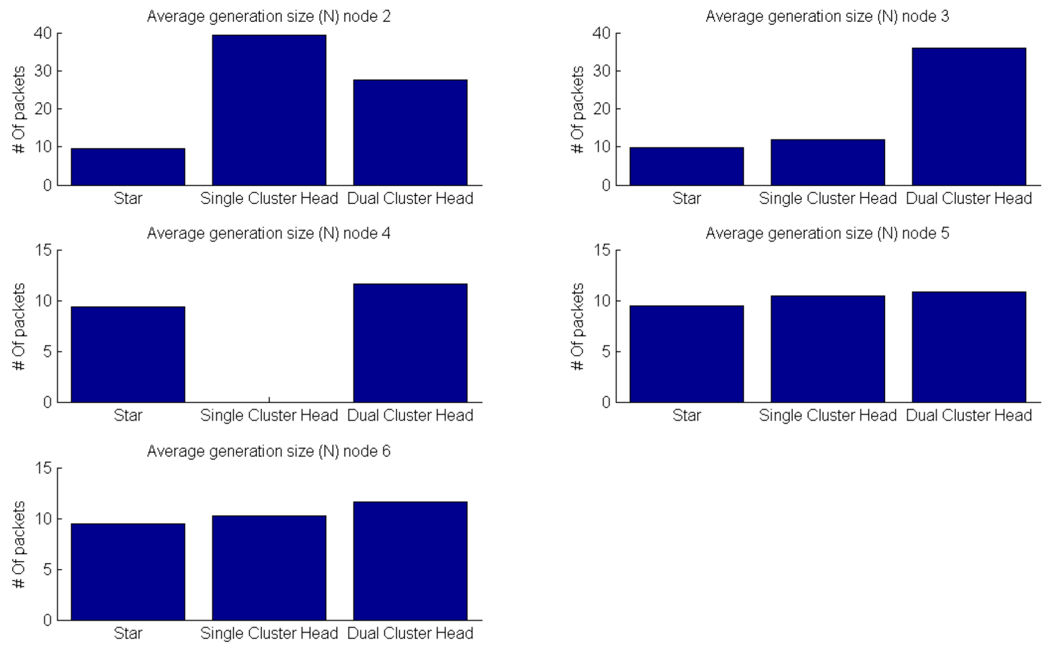


Figure 9.77.: Different nodes' generation sizes, Comparison, 40% Loss

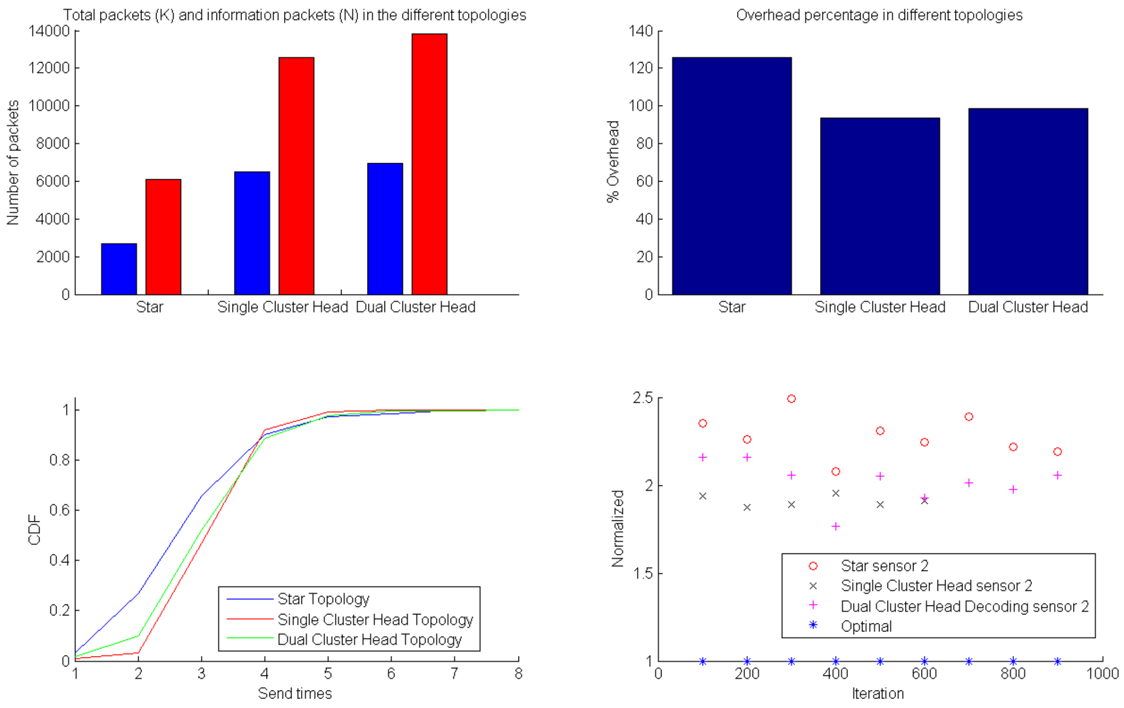


Figure 9.78.: Node 2's packets sent, overhead packets %, decoding CDF,average K/N ratio, Comparison, 40% Loss

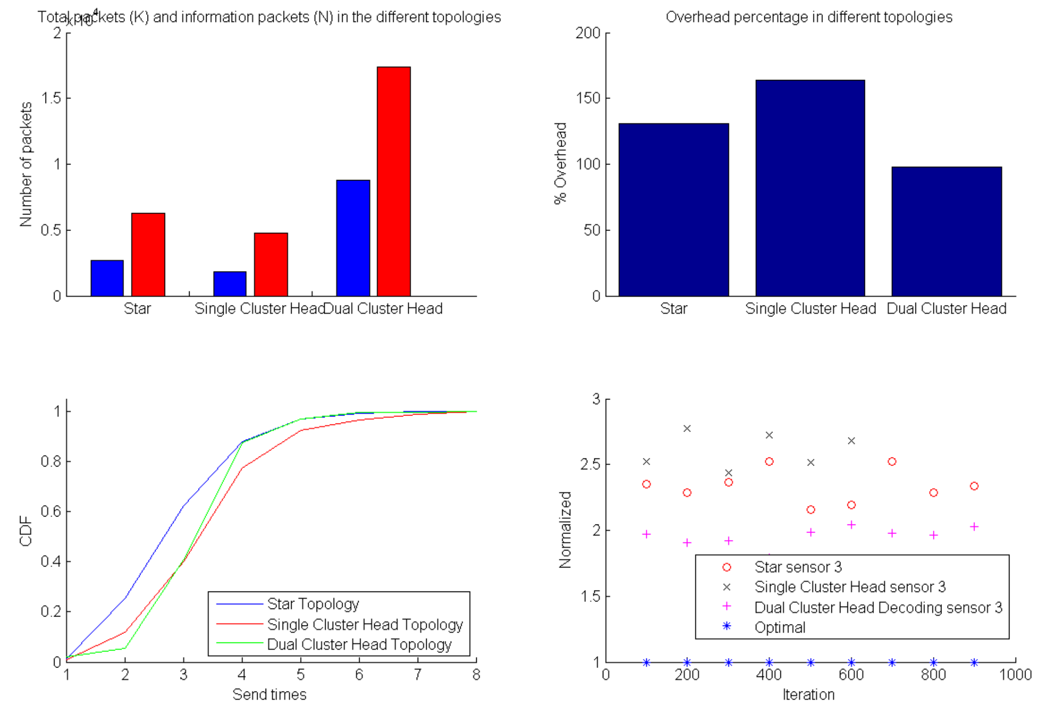


Figure 9.79.: Node 3's packets sent, overhead packets %, decoding CDF,average K/N ratio, Comparison, 40% Loss

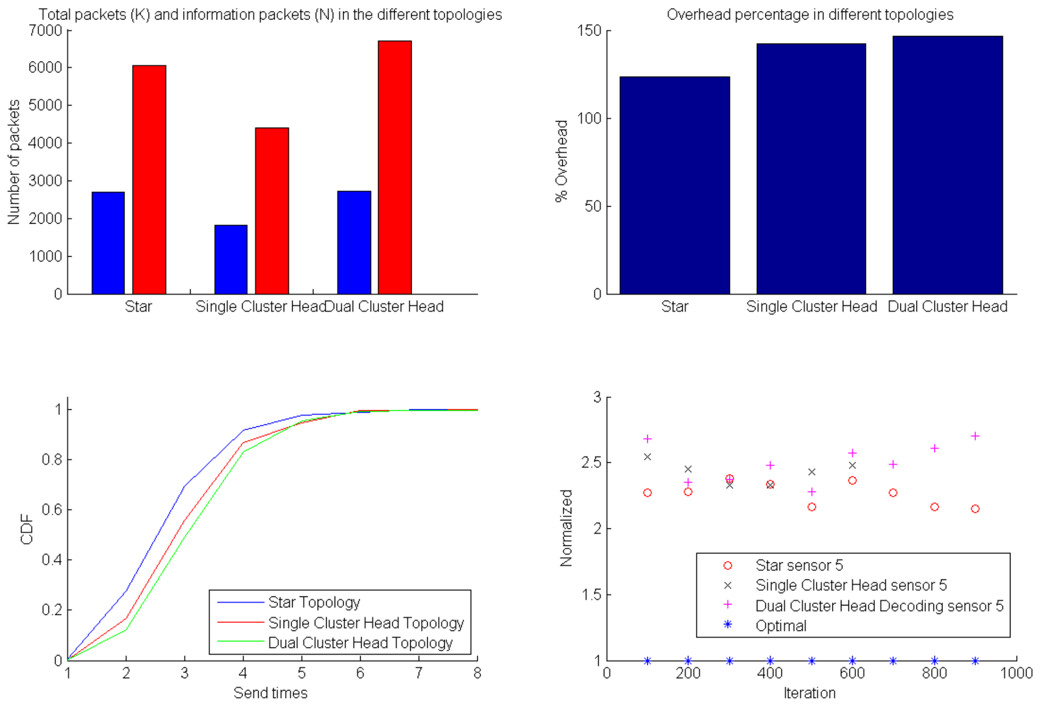


Figure 9.80.: Node 5's packets sent, overhead packets %, decoding CDF,average K/N ratio, Comparison, 40% Loss

## 9.5. Network Lifetime

Some network lifetime tests have been performed changing the send and measurement times but all of them lead to the same results not lasting for less than 19 hours even leaving half an hour between sends, which is really short lifetime for a wireless sensor network. The reason is that even if the sensor is not sending nor receiving data the WiFi hardware keeps being connected to the network and wasting energy. One way to solve this could be the use of duty cycles [20] which will require time synchronization between the nodes or using a beacon system. For this scheme a synchronized system would be better because it would let continue exploiting the advantages of overhearing as the nodes will be awake and sleep at the same times. Keep in mind that during this tests some of the nodes might have not been completely charged, thus we should look at the maximum value of lifetime value.

Figure 9.81 shows the lifetime for the network when the measures are taken every 10 minutes and the packets are sent every 30 minutes, as we can clearly see there is not a big difference in terms of lifetime compared with the tests at 2 seconds between data collection and 6 seconds between sends, figures 9.82, 9.83 and 9.84. All the test were done with the dual cluster head topology, as we can see the nodes that process and send most packets, 2 and 3, don't seem to have lower lifetime than the rest of the nodes, this could be due to the overhearing as all of them are receiving the packets sent by the rest.

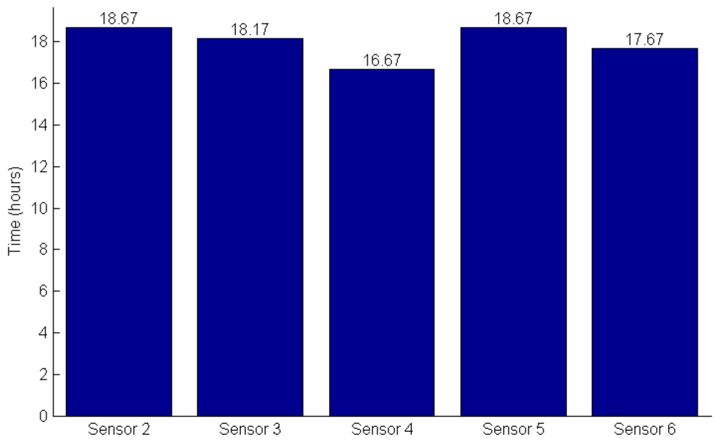


Figure 9.81.: Node Lifetime, Measures each 10 minutes, Sends each 30 minutes, Dual Cluster Head, Same Room

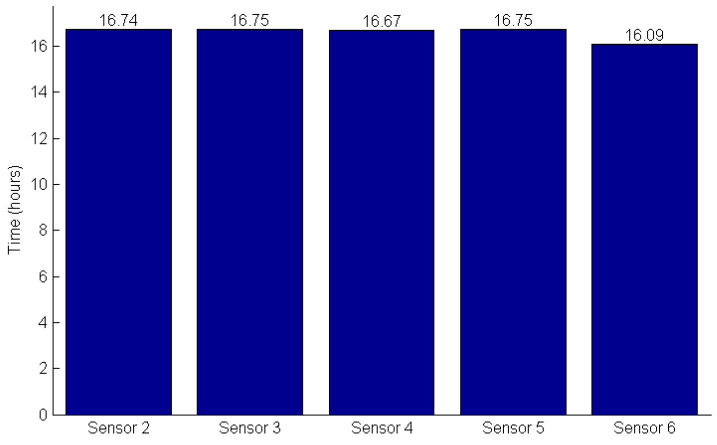


Figure 9.82.: Node Lifetime, Measures each 2 seconds, Sends each 6 seconds, Dual Cluster Head, SR0, Same Room

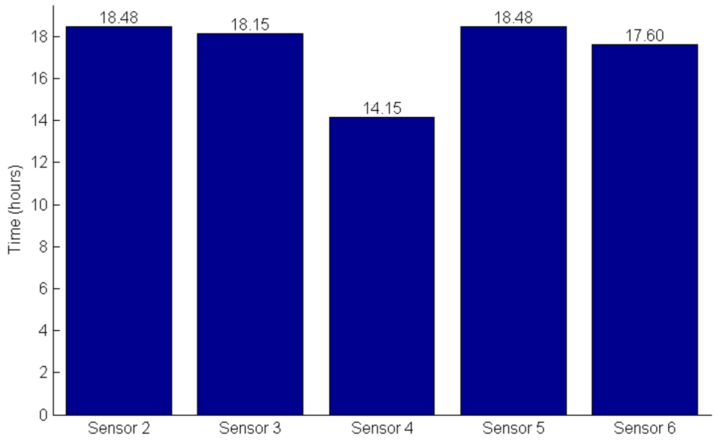


Figure 9.83.: Node Lifetime, Measures each 2 seconds, Sends each 6 seconds, Dual Cluster Head, SR1, Same Room

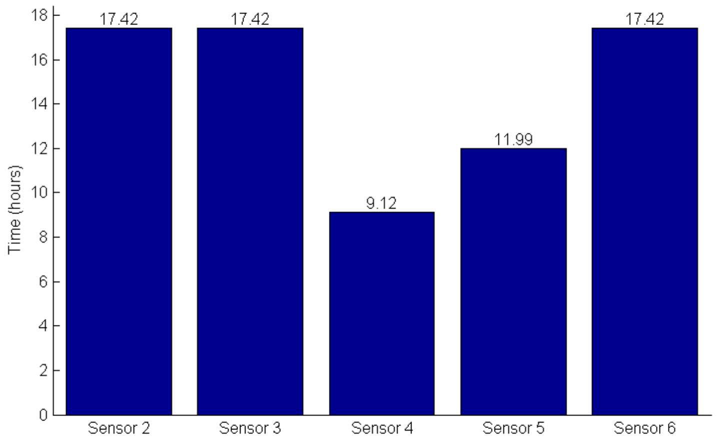


Figure 9.84.: Node Lifetime, Measures each 2 seconds, Sends each 6 seconds, Dual Cluster Head, SR2, Same Room

# Part V.

# Future Work and Conclusion

# 10. Future Work

## 10.1. Routing Improvements

- Incorporation of a set-up phase: Where the nodes only listen until they know their distance and start transmitting after discovering it. This will decrease the number of transmissions in the first stages of the network.
- Changing the IDPkt for a time stamp to know the time when the data was generated, and still identifying the data as the time can't be repeated.
- Over the air commands: Command packets to control the nodes from the sink, this will need to implement a routing protocol from the sink to the nodes, working in opposite direction than the current one.
- Network duty cycle: Synchronization between the nodes to sleep and wake up in the same time windows in order to save energy.

## 10.2. Sensor Improvements

- Discarding data or stopping to decode data if the buffers are too full, to prevent the node from crashing.
- Over-the-air updating: Allows to make changes in the code without having to gather up the nodes and place them again.
- Disable the hardware when it's not needed, this will be related with the duty cycle mentioned in the section before, as they together would save a lot of energy.

## 10.3. Network Coding Improvements

- Use the nodes in the way to the sink as recoders in stead of waiting until the whole generation has been decoded to start sending to the next node. With this method the final decoding of each sensor would be done in the sink.
- Using variable SendRedundancy packets according to an estimated link loss rate. The numbers used for the redundancy packets sent in this test were meant for really small loss rates as in the first tests, when the loss rates increased the redundancy packets didn't make any difference. A link quality estimator could select the best number of redundancy packets to send increasing the decoding probability at low send times and thus the latency.

# 11. Conclusion

The benefits of wireless sensor networks and network coding are widely known, as well as their synergy. Along this work we did a practical implementation a of network coding scheme into the XDK sensors to test its behavior in a hazardous environment for the communications such as the wireless medium. Allowing the nodes to sense the environment and collect all the data in a common point. Following that purpose, a wireless meshed network constituted by those nodes was established. But to make the nodes capable of communicating properly a routing protocol was needed as they need to know how to guide the information. We designed a simple routing protocol which was meant for taking advantage of the broadcast nature of the wireless medium. It exploits the benefits of the spatial diversification of possible interested receivers that will be overhearing all the packets that are sent in their reception range. Built out of those components, a reliable data gathering protocol with the advantages of the wireless sensor networks and network coding was be implemented using UDP. The final scheme was tested under different conditions as increasing the distance between the nodes, changing the topology, the number of redundant packets sent and artificially changing the link losses showing the relation between the packet losses and the extra packets needed for recovering the data.

# Bibliography

[1] Philipp Nenninger and Marco Ulrich. Harvest time. *ABB review*, 2012.

[2] Markus Leinonen, Juha Karjalainen, and Markku Juntt. Distributed power and routing optimization in single-sink data gathering wireless sensor networks. *2011 19th European Signal Processing Conference*, 2011.

[3] Chong Tan and Junni Zou. A multicast algorithm based on network coding for wireless sensor network. *2007 IET Conference on Wireless, Mobile and Sensor Networks (CCWMSN07)*, 2007.

[4] Michael Healy, Thomas Newe, and Elfed Lewis. Wireless Sensor Node hardware: A review. *SENSORS, 2008 IEEE*, 2008.

[5] David Gómez Fernández. *Uso de técnicas de Network Coding y Multipath para la mejora de las comunicaciones sobre redes malladas inalámbricas*. PhD thesis, 2015.

[6] Ye Tian, Kai Xu, and Nirwan Ansari. TCP in wireless environments: problems and solutions. *IEEE Communications Magazine*, 43, 2005.

[7] DJay Kumar Sundararajan, Devavrat Shah, Muriel Médard, Szymon Jakubczak, Michael Mitzenmacher, and João Barros. Network Coding Meets TCP: Theory and Implementation. *Proceedings of the IEEE*, 99:490 – 512, 2011.

[8] David Gómez, Eduardo Rodríguez, Ramón Agüero, and Luis Muñoz. Reliable Communications over Lossy Wireless Channels by means of the Combination of UDP and Random Linear Coding. *2014 IEEE Symposium on Computers and Communications (ISCC)*, 2014.

[9] Bosch GmbH. Start page bosch xdk.

[10] Bosch GmbH. Xdk general information.

[11] Bosch GmbH. Xdk technical overview.

[12] Steinwurf ApS. Including kodo-c in your application.

[13] George Xylomenos and George C. Polyzos. TCP and UDP performance over a wireless LAN. *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, 1999.

[14] Parth Mannan and Jayavignesh T. Alternate simplistic approach to solve count-to-infinity problem by introducing a new flag in the routing table. *2016 Second International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, 2016.

[15] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. MORE: A Network Coding Approach to Opportunistic Routing. *Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory*, 2006.

[16] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, Senior Member, and Jon Crowcroft. XORs in the Air: Practical Wireless Network Coding. *IEEE/ACM Transactions on Networking*, 16, 2008.

[17] Jeppe Krigslund, Jonas Hansen, Martin Hundebøll, Daniel E. Lucani, and Frank H. P. Fitzek. CORE: COPE with MORE in Wireless Meshed Networks. *2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*, 2013.

[18] Oscar Trullols-Cruces, Jose M. Barcelo-Ordinas, and Marco Fiore. Exact Decoding Probability Under Random Linear Network Coding. *IEEE Communications Letters*, 15, 2011.

[19] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Muriel Médard. On Code Parameters and Coding Vector Representation for Practical RLNC. *2011 IEEE International Conference on Communications (ICC)*, 2011.

[20] Rashmi Ranjan Rout and Soumya K. Ghosh. Enhancement of Lifetime using Duty Cycle and Network Coding in Wireless Sensor Networks. *IEEE Transactions on Wireless Communications*, 12, 2013.

# CD Content

- Two sided PDF version of this work
- XDK application project
- Sink code
- Status receiver code
- Decoder code
- Plots of the results