University of Stavanger

Faculty of Science and Technology

BACHELOR'S THESIS

Study program/ Specialization:	Spring semester, 20..... Open / Restricted access
Writer:	…… (Writer's signature)
Faculty supervisor: External supervisor(s):	
Thesis title:	
Credits (ECTS):	
Key words:	Pages: …… + enclosure: …… Stavanger, …….. Date/year

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Wireless Error Control

Bachelor's Thesis in Computer Science
by

Ahmad Saleh Kazzaz

Internal Supervisors

Gianfranco Nencioni

May 15, 2023

"A model is a lie that helps you get at the truth."

Howard Skipper

Abstract

Wireless communication has become an integral part of our daily lives. This thesis explores various wireless error control techniques that can detect and correct errors during wireless communication. The thesis provides background knowledge on wireless error control techniques. The thesis presents different studies about which tools to use to simulate wireless error control techniques. I have presented several tools that can be used to simulate some of these techniques and compared these tools with each other. I found that Matlab is a suitable tool for the simulation of wireless error control techniques. The thesis includes a simulation of three techniques hamming code, convolutional codes, and Go-Back-N ARQ. The impact of noise and SNR on these three techniques are evaluated in the thesis. The main contribution of the thesis is the evaluation of different error control techniques under varying conditions and provides a comprehensive overview of different wireless error control techniques and their effectiveness under different conditions. The main finding is that the hamming code and Go-Back-N ARQ are more stable than the convolutional codes. Also, a higher SNR leads to a better performance of the techniques and the lower SNR leads to a poorer performance of the techniques.

Acknowledgements

I would like to thank my supervisors for their fantastic enthusiasm and help with writing this thesis.

Contents

Chapter 1

Introduction

In recent years, wireless communication has become an essential part of our daily lives. Wireless technology has revolutionized the way we communicate and connect together. One of the major challenges in wireless communications is the presence of errors in the transmission of data. Errors can occur due to various reasons such as noise. These reasons can lead to different errors like information loss, data corruption, or reduced network performance. Wireless error control is a process that helps to detect and correct errors that occur during data transmission in wireless communication. The goal of error control is to ensure that the received data is free from errors, even in the presence of noise and other disturbances. Several techniques are available for wireless error control, such as convolutional codes, hamming code, and Go-Back-N ARQ.
This thesis will explore the various techniques used for wireless error control. We will learn the different types of wireless error control that can detect and correct errors in wireless communication. We will also see how these techniques work under different conditions.

The thesis will be organized as follows. Chapter 2 presents background knowledge on wireless error control techniques, allowing us to understand the various techniques theoretically.
Chapter 3 is named "State of Art". This chapter is presenting the tools we can use to simulate the wireless error control techniques. In this chapter, we will also provide a comparison of the tools. The chapter also presents various studies about the simulator tools we can use to simulate wireless error control techniques.
Chapter 4 will provide the simulations of some of the wireless error control techniques. We will see simulations of how they work practically. Finally, we will evaluate the results of the simulations and conclude with the key findings.

Chapter 2

Background

Wireless error control refers to the techniques used to detect and correct errors in blocks of data during transmission. Communication channels can often be unreliable, causing data to be corrupted during transmission. To detect errors, two methods can be used: CRC (Cyclic Redundancy Check) and parity check. There are two types of errors that can occur in wireless communications: single-bit errors and burst errors.
For error correction, two methods can be used: Forward Error Correction (FEC) and Backward Error Correction (BEC).

2.1 Forward error correction

Forward error correction(FEC) is a type of error correction method. The technique is used to control errors in data transmission over noisy channels. The central idea for the FEC is that the sender redundantly encodes the message which allows the receiver to detect and correct errors. There are several techniques that use forward error correction like block codes and convolutional codes.

2.1.1 Block Codes

Block Codes are a large and important family of error correction codes that encode data in blocks. They are used to minimize errors that may occur during the transmission or storage of data. The idea behind block codes is to provide the sender or receiver with a set of tools that they can use to detect and correct errors in the code without needing to contact the source of the code. There are many types of block codes, including Hamming code, Reed-Solomon codes, and Hadamard codes.

2.1.2 Hamming code

Hamming code is a type of forward error correction (FEC) technique used to detect and correct errors that may occur during data transmission or storage. It is particularly effective in correcting single-bit errors. The hamming code can detect one-bit and two-bit errors. The technique has been invented by Richard W. Hamming in 1950. In Richard's original paper, he focused specifically on the hamming(7,4) code which adds three parity bits to four bits of data. The hamming code includes two methods: **Hamming check bits**: Hamming check bits are additional bits. These check bits are inserted at positions that are powers of two, starting from the right side at the data block. The Hamming check bits are calculated using the XOR (exclusive OR) operation on the position of the data block bits that are equal to 1.

Syndrome word: This is calculated by performing the XOR operation on the position numbers of the received data-block bits that are equal to 1 and the hamming check bits. The syndrome word has several characteristics: if it has one of the 1s, it means that the error is in the check bits. If it has more than 1, then it indicates the position of the error. Here's an example to help you understand how to calculate the hamming check bits and syndrome word.

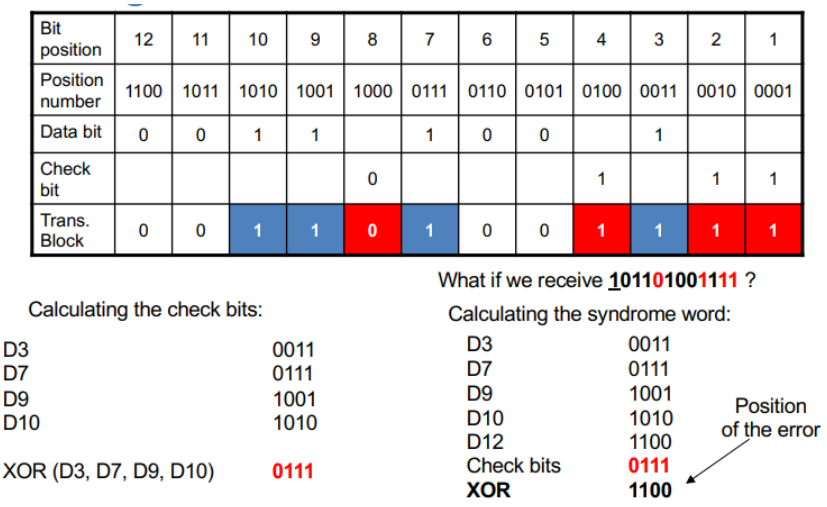

Figure 2.1: Calculation example

2.1.3 Convolutional codes

Convolutional codes are a type of forward error correction code that generates redundant bits. The parameters of the Convolutional codes are (n,k, K). Where k is a data bit processed at a time, K is a constraint factor and n is the output of bits from each k bit and n output bits depending on K x k input bits.
A trellis diagram is a tool used in Convolutional Codes, which is a type of error-correcting code used in communication systems. The diagram consists of nodes arranged in vertical slices of time, with each node connected to at least one node at an earlier and later time. The earliest and latest times in the diagram have only one node. Trellis diagrams are used in encoders and decoders processes for communication theory encryption and in the Viterbi algorithm.

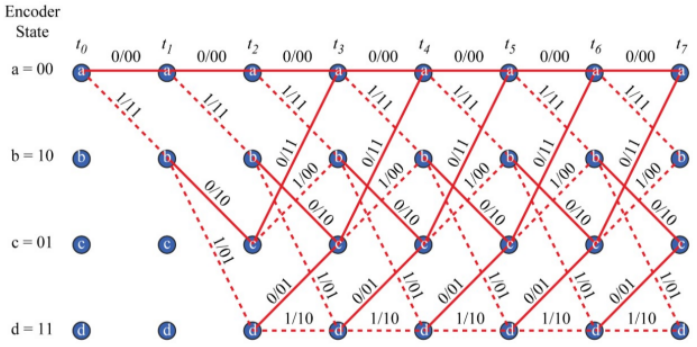

Figure 2.2: Trellis diagram

In this example above, we follow the path a-b-c-b-d-c-a-a in the Trellis diagram, which gives us the output 11 10 00 01 01 11 00. The input for this example is 1011000, and the invalid path is a-c. The Viterbi algorithm is a recursive algorithm used to find the most likely input that produced the invalid output. We use hamming distance metrics in this algorithm.

2.1.4 Turbo codes

Turbo codes are a type of forward error correction(FEC) that offers high performance. The technique has been used in 3G and 4G wireless communication systems. Turbo codes have been created based on principles of convolutional codes, which is also a form of forward error correction.

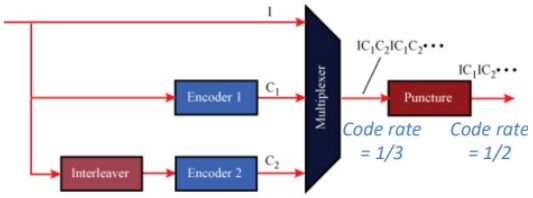

Figure 2.3: Turbo codes - encoding

The figure above shows an example of the encoding process for turbo codes. Turbo codes are considered recursive systematic convolutional codes that reduce the number of transmitted bits while maintaining a high level of error correction. Puncturing is used to further reduce the number of the transmitted bits by selecting only one check bit from the encoding output.

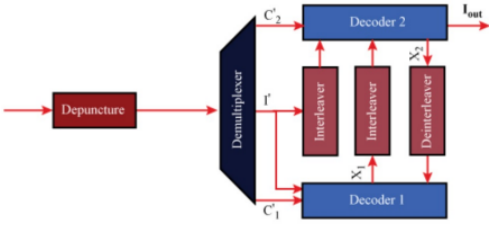

Figure 2.4: Turbo codes - decoding

The figure above shows an example of the decoding process of turbo codes. The decoders perform the process of de-puncturing, which involves estimating the missing check bits through soft decision decoding. The Decoding process is iterative, meaning that it is repeated multiple times to improve the level of confidence in the decoded output. However, this iterative process also introduces a high level of confidence and delay.

2.2 Backward error correction

Backward error correction is a type of error control correction used in communication systems where the receiver detects errors in the data block and requests the source device to resend the data block. This technique is particularly useful when data is lost or corrupted during transmission.
Following to the backward error correction the return channel of the communication is used to indicate errors in the transmission. If there are no errors, the acknowledgment will be positive. However, if an error is detected, the acknowledgment will be negative, and the receiver will request the sender to resend the data block.

2.2.1 Automatic repeat request

Automatic Repeat Request (ARQ) is a process used for data transmission where a Protocol Data Unit (PDU) is either lost or damaged during transmission. In such cases, the sender is requested to resend the PDU. ARQ is used in both the data link and transport layer of a communication protocol. In the ARQ process, the data block is sent as a PDU.
The latest version of the ARQ process is the Go-Back-N ARQ. The properties of Go-Back-N ARQ include the use of positive acknowledgment if there are no errors detected and negative acknowledgment if there are errors detected. Additionally, the transmitter timer may expire. The priority of ARQ is to ensure flow control and error control in the transmission of data blocks. Error control involves detecting and correcting any errors that occurred during transmission. Flow control, on the other hand, ensures that the receiver entity is not overwhelmed with too much data.

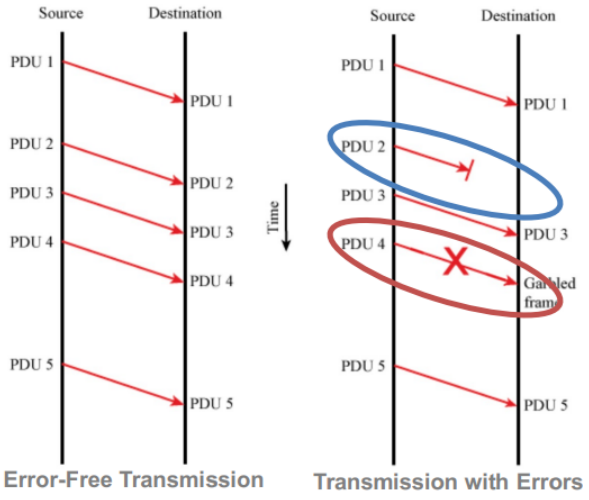

Figure 2.5: ARQ

In the figure above we can see how the Automatic Repeat Request (ARQ) process works to detect errors during data transmission. The figure shows both an error-free transmission and a transmission with errors that were detected and corrected through the ARQ process. By using ARQ, the sender can automatically resend any data packets that are lost or corrupted during transmission, improving the reliability of the overall data transfer process.

There are limitations to using ARQ in wireless applications. For instance, in wireless links, the bit error rate (BER) can be high, leading to a large number of re-transmissions. Additionally, in some cases such as with satellites, the propagation delay can be very long due to the transmission time of the frame. These factors can negatively impact the efficiency and effectiveness of the ARQ process.

2.3 Parity check

A parity check is a technique of error detection used in wireless communication systems. There are two types of parity check: **Even parity** where the parity bit is set to 1 or 0 to make the total number of 1's or 0's in the transmitted data even. The other type is **Odd parity** where the parity bit is set to make the total number of 1's or 0's odd. The technique works by adding a single extra bit to the data that is being transmitted. The extra bit is called the parity bit, and it is set to either 1 or 0 to make the total number of 1's in the transmitted data even or odd depending on the parity scheme used. When the data is received, the receiver checks the parity bit to ensure that the total number of 1s in the received data is still even or odd, as expected. If the total number of 1s in the parity bit is not even or odd as expected, then the receiver can request the sender to re-transmit data.

2.4 Cyclic redundancy check (CRC)

A cyclic Redundancy check is a technique to detect errors in transmission data. The technique is working by adding a small amount of redundant data (usually extra bits) to the end of the transmitted message. Redundant data is a value check and when the data is transmitted the receiver will take a new check and see if the new check of data is matching the redundant data. If they matched then the message is correctly transmitted. If checks are not matched, the receiver assumed that the transmitted data is corrupted and will request a re-transmission. CRC is widely used in network protocols, such as Ethernet, to ensure the accuracy of transmitted data.

Chapter 3

State of the Art

The Internet is very important in our daily life to have the Internet we can use wireless which can share the Internet to our devices like Mobil, TV, or laptop. Sometimes errors can occur in wireless communications. To study these errors in practice we can use simulator tools to simulate the communication and the techniques we can use to ensure error-free communication. For my thesis, I have searched for tools that can be used to simulate wireless error control techniques. I will provide tools that I think are useful for my thesis. I will provide these tools in the "tools" section along with a comparison of tools. In the next section, I will review several studies that have used these tools to simulate, evaluate and analyze wireless error control techniques.

3.1 Related works

In this section, we will present a review of studies that evaluate the effectiveness of tools used to simulate and evaluate wireless error control techniques in wireless communication systems.
The study "A Case Study of Various Wireless NetworkSimulation Tools". The study has compared the tools we can use to simulate wireless error control techniques. Authors have compared tools like Matlab, NS-2, and NS-3. The study showed that OMNET++ and NS-3 are more flexible and accurate tools to simulate wireless error control techniques. The study also found that Matlab is one of the most used tools for wireless communication because that Matlab is an easy tool to use, and the tool flexibility. Another reason that Matlab is one of the most used tools for wireless communication is that Matlab is able to handle large amounts of data. The study found also that Matlab is a suitable tool to analyze wireless error control techniques.

Another study is named ”A review of simulation techniques for some wireless communication system”. This study has compared simulation tools like Matlab, and Opnet. The study showed that Matlab is a popular simulation tool in wireless communication because of its flexibility and ability to handle large amounts of data. The tool can be used to simulate and analyze different wireless communication systems such as wireless error control. The study showed that OPNET is commonly used in wireless communication. OPNET is known for its ability to simulate large-scale wireless networks and is also known for the tool’s flexibility in modeling various wireless communication system components. One another study “A comparative study of different network simulation tools and experimentation platforms for underwater communication”. The underwater communication includes also wireless error control techniques such as convolutional codes, hamming codes, and Go-Back-N ARQ. The study has compared the simulation tools such as Matlab, Opnet, OMNET++, NS-2, and NS-3. The study showed the weaknesses and strengths of each of the simulation tools. The study showed that Matlab is a popular tool because of its built-in features and the study mentioned also that Matlab is a good tool to analyze the performance of wireless error control techniques. The study found that NS-3 is a good tool to evaluate different protocols and NS-3 provides a simulation with different scenarios.

3.2 Tools

3.2.1 Matlab

Matlab is a powerful tool that we can use to study wireless error control. Matlab provides a range of functions and tools for simulating and analyzing wireless communication systems, including error control techniques. One of the main advantages of using Matlab for studying wireless error control is for flexibility the tool provides. Matlab allows users to model wireless channels, and implement various error control schemes such as error correction codes. Matlab also gives users a range of toolboxes for wireless communications like the Communication toolbox, Signal Processing Toolbox, and RF toolbox. Matlab has its coding language which is named MATLAB.
Matlab is a popular tool there are, millions of engineers and scientists worldwide using Matlab in range in academia and industry. Many students around the world use Matlab as part of their studies.
There is a help center for Matlab. Students of Stavanger University can download Matlab for free.

3.2.2 NS-3

NS-3 is a tool to simulate networks. The tool support wireless network simulations, including technologies such as 5G, Wifi, and Ethernet. NS-3 allows the users to simulate wireless error control techniques. NS-3 is considered one of the most powerful tools available.
NS-3 is open-source and free to download and use. NS-3 is widely used for simulating and analyzing various network protocols and applications.
NS-3 uses C++ programming language which provides a high-performance and low-overhead programming language that is well-suited for developing complex simulation models.
NS-3 includes a variety of simulations and several tools for analyzing the simulation results. The program is widely used in academia and industry for education and research of network technologies and applications.

3.2.3 OPNET modeler

OPNET Modeler is a widely-used simulation tool for creating network models including wireless error control techniques in both academic and industry settings. The program offers its own Modeler Internal programming language, which is a high-level language similar to C++ and C that enables users to create and define the behavior of network components. Additionally, OPNET Modeler allows the use of C++ or C for programming. However, it is important to note that OPNET Modeler is not a free tool and may require a higher budget compared to other simulation tools.

3.2.4 OMNET++

OMNET++ is a simulator tool used to simulate wireless communication including wireless error control techniques. The tool can simulate networks without models for network protocols for example HTTP. The tool uses C++ as a programming language. OMNET++ can be used for free for non-commercial simulations like at academic, and institutions and for teaching. OMNET++ is widely used in the academic and research communities for simulating and evaluating different aspects of wireless communication networks such as wireless error control techniques.

3.2.5 Comparison of tools

When we select a simulation tool for studying wireless error control techniques it's important to compare their strengths and weaknesses based on their specific research needs. Matlab offers a powerful and versatile platform for simulating wireless communication and implementing wireless error control techniques. Its built-in functions and syntax for signal processing and modulation make it a convenient choice for researchers who are familiar with Matlab programming. OMNET++ is another powerful tool used for simulating wireless communication including wireless error control techniques. Unlike Matlab, it uses C++ as a programming language, which can be a disadvantage for researchers who are not familiar with C++. However, OMNET++ provides more flexibility and control over the simulation compared to Matlab, allowing researchers to customize the simulation at a low level. OMNET++ is also free for non-commercial simulations like at academic and research institutions. NS-3 is more suitable for researchers who require a high level of detail in network simulation. OPNET may be more appropriate for researchers who have a higher budget and require a commercial-grade simulation tool. OPNET and NS-3 provide C and C++ programming languages. While Matlab tool uses Matlab programming language which I as a student of Stavanger University have a good knowledge of Matlab programming language from my course work. Therefore Matlab is a very good simulation tool for my case to study wireless error control and do some exercises which can help to understand error control techniques better.

Chapter 4

Design and solution

This section will show the design of wireless error control types and will show also the solutions to the problem. In the previous sections, we have explained how Wireless control error techniques work theoretically, in this section, we will simulate some of the wireless error control techniques. The simulations will include both forward error correction and backward error correction. We will evaluate the performance of these techniques under different conditions like noise and also under different SNR levels. We will evaluate the techniques based on parameters such as packet error rate(PER), and bit error rate(BER). The packet error rate presents the number of packets that have an error at percent and BER presents the number of errors per unit of time. The goal of simulation of error control techniques is to help us to understand how these techniques work in reality after we understood them theoretically. Also, it will help us to evaluate and analyze these techniques.

4.1 Simulating hamming code

During wireless communications, many transmissions will happen through those transmissions an error or several errors can occur. Those errors will disrupt communication in the network. For solving those errors we can use wireless error control techniques. I have chosen to simulate hamming code which is using forward error correction. Hamming code technique helps us to detect errors and correct these errors. I will use Matlab to simulate how the technique works in reality. The simulation will allow us to evaluate the performance of hamming code under noise.

In the coming example, I will create a code for a simulation of hamming code(4,7). This simulation will not include the syndrome word or hamming bits check because it may be more complex if we will use them in our simulation example. In this simulation example, I will create a message and then encode it by adding parity bits to the message. The message in this simulation example consists of 4 bits. To encode it we add 3 parity bits to the message to form a code word of 7 parity bits. Then we include the code words which is mean the set of valid code words defined. These code words are used to check for errors in the received data(decoded message) by comparing the received bits with each of the valid code words. If the received bits are closer to one of the valid code words than any other, the received bits are decoded to that code word. This simulation example will include noise. This will help us to evaluate the hamming code performance under noise. After that, we can move to the decoding part where we decode the transmitted message by using either hard decision decoding or soft decision decoding. I have decided to provide both hard decision decoding and soft decision decoding which can help us to understand how they both work.

Now let us see the code of the simulation. The figure below will show how I have created a message and encoded the message.

```
% Initialize the error counter
Nerrs =0; Nblocks=1000;
for i=1 : Nblocks
    msg = randi([0 1],1,k);
    %encoding
    codeword = [msg mod(msg(1)+msg(2)+msg(3),2)...
                    mod(msg(2)+msg(3)+msg(4),2)...
                    mod(msg(1)+msg(2)+msg(4),2)];
s=1-2*codeword; %symbol converstion
r=s+sigma *randn(1,n); %add noise
```

Figure 4.1: Create a message and encode it

First, I created a message by using the Matlab function "randn()". The message is a binary value consisting of 0s and 1s. Parameter "k" represents the length of the message which is 4 in our example. Then I created the noise and added it to the message by using also the "randn()" function. The S variable will give us a symbol vector for the code and r will give us the received vector for the code. We now move on to the decoding part. The decoding part includes both hard decision decoding and soft decision decoding.

```
%hard decision decoding
b=(r<0);
distance = mod(repmat(b,16,1)+codewords,2)*ones(7,1);
[mind1,pos]=min(distance);
msg_capl = codewords(pos,1:4);

%soft decision decoding
corr = (1-2*codewords)*r';
[mind2,pos]= max(corr);
msg_cap2 = codewords(pos,1:4);
```

Figure 4.2: Decoding part

The code in the figure above shows the parts of hard decision decoding and soft decision decoding. In hard decision decoding, the first step involves creating a binary vector "b". The second step computes the hamming distance between the received code word and each of the 16 possible code words by creating a matrix of size 16 x 7. The third step is to find the minimum value in the distance vector. Which corresponds to the code word that is closest to the received code word. The code then stores the minimum value in "pos" and the minimum distance in "mind1". In the final step, the code extracts the first four bits of the closest code word which represents the original message.
In soft decision decoding. the first step involves calculating the correlation between the received signal "r" and each of the 16 possible code words. The second step is to find the maximum correlation value of the "corr" vector. This maximum value is the most likely to be the transmitted code word, and it is assigned to the variable "pos". The third step extracts a 4-bit message corresponding to the most likely code word from the code words matrix. in the last step, the code updates the total number of decoding errors by adding the number of errors found for the current message.

```
>> Hammingcode
Bit Error Rate: 0
>> msg

msg =

     0     1     0     1

>> msg_capl

msg_capl =

     0     1     0     1

>> msg_cap2

msg_cap2 =

     0     1     0     1

fx >>
```

Figure 4.3: Results

The figure above displays the results of the simulation example, which indicates that the decoding process was successful in recovering the original message without any errors. We can see also the bit error rate(BER) is 0. Therefore, the code has effectively corrected any errors that occurred during the transmission of the message.

4.1.1 Evaluating with SNR

Noise-to-signal parameters help us to evaluate the performance of the hamming codes and also it allow us to determine the ability of the hamming codes to correct errors in the transmitted data. In our example, the SNR results are shown in the following figure.

```
>> Hammingcode
Bit Error Rate: 0
Bit Error Rate: 4.5696
```

Figure 4.4: SNR

The figure above shows the SNR value of the hamming codes, which is 4,5696 and the number of errors is equal to 0. This means that the received signal is 4.5696 times stronger than the noise in the communication channel. We see also that the number of errors is 0 and the BER is 0.0 percent.

To evaluate the performance of the hamming codes, we can change the SNR by adjusting the power of the noise. In this example, we can change the power of the noise by changing the value of the EbNodB parameter in the simulation.
First, we set the EbNodB parameter to equal 10 and simulate with the new EbNodeB. The figure below shows the results.

```
>> Hammingcode
Bit Error Rate: 0
Bit Error Rate: 7.5696
```

Figure 4.5: Higher SNR

We can see, The SNR value has been increased to 7.5696 and BER is 0.0 percent. The BER value means that the number of errors is 0. The SNR number means that the received signal now is 7.5696 times bigger than the noise in the communication channel. That increase in SNR has led to a better performance of the hamming codes in correcting errors in the transmitted data. The next step is to set the EbNodB to 5 and run the code one more time with the new EbNodB. The result is shown in the figure below.

```
>> Hammingcode
Bit Error Rate: 0.0065
Bit Error Rate: 2.5696
>>
```

Figure 4.6: Lower SNR

The figure shows that the SNR value has decreased to 2.5696 and the BER value is 0.0065. The result means that the received signal is now only 2.5696 times stronger than the noise in the communication channel. The SNR decrease will lead to poorer performance of the hamming codes in correcting errors in the transmitted data.

These three results which are displayed above show that when the SNR value decreases the number of errors increases. That means the hamming codes work poorer in low SNR values and the hamming codes perform better in high SNR values.

4.2 Simulating Convolutional codes

Convolutional codes are a type of forward error correction code that uses a trellis diagram to encode and decode messages. I will use Matlab to simulate how the technique works in reality and the simulation allow us to evaluate the performance of convolutional codes under noise.

In the coming example, I will create a code for convolutional codes. First, we need to define the generator polynomial and the constraint length.

The generator polynomial is a polynomial with a binary coefficient that is used to determine the output bits of the encoder for each input bit. It defines the encoding process of the code.

The constraint length of the code is the number of previous input bits that are used to determine the current output bit. The constraint length is considered in the encoding process.

The trellis structure is a graph that shows all the possible paths that the encoder can take for a given input sequence. It's determined by the generator polynomial and the constraint length of the code.

Once the generator polynomial and the trellis structure are defined, the message can be encoded by passing it through the trellis diagram. The corresponding output sequence is generated by the encoder.

In the decoding process, the trellis diagram is used to find the most likely transmitted message. The received message is passed through the trellis diagram and the most likely path is determined by comparing the received sequence to all possible sequences in the trellis diagram. The Viterbi algorithm is often used to perform the decoding process.

The figure below displays how I have created the generator polynomial, constraint length, and trellis diagram for the message and how I have created the message. The figure displays also the encoding process.

```
% Define the generator polynomial in octal notation
genPoly = [7 5];

% Define the trellis structure
trellis = poly2trellis(3, genPoly);

% Generate a random message sequence
msg = randi([0 1], 1, 10);

% Encode the message sequence using the convolutional code
encoded = convenc(msg, trellis);
```

Figure 4.7: Encoding process

To create convolutional codes, you first need to choose a generator polynomial and determine the constraint length of the code.
After we have chosen the generator polynomial and the constraint length, we can use the "poly2trellis" function in Matlab to define the trellis structure for the convolutional codes.
Next, we create a message by using the "Randi" function in Matlab, which generates a vector of random values of either 0 or 1.
Then the message has been encoded by using the "convenc" function in Matlab. The function takes two arguments, the message and the trellis structure of the convolutional codes, and outputs the encoded sequence generated by the convolutional encoder.

```
% Add noise to the encoded message sequence
EbNo = 4;
SNR = 10*log10(EbNo) + 10*log10(2/3);
noisy = awgn(encoded, SNR, 'measured');
% Quantize the noisy sequence to a binary sequence
thershold = 0;
quantized = (noisy > thershold);

% Decode the quantized message sequence using the
% Viterbi algorithm with hard decision
decoded = vitdec(quantized, trellis, 5, 'trunc', 'hard');
```

Figure 4.8: Decoding part

The figure above shows how we have added the noise in the communication and it's shows also the decoded part. We add the noise by using the "awgn" function in Matlab. We calculate the SNR which we will use to evaluate the performance of the technique.
After adding noise, we quantize the noisy signal to a binary sequence by setting a threshold value and comparing each sample of the noisy sequence to the threshold. If the sample is greater than the threshold, it is assigned a value of 1, otherwise, it is assigned a value of 0. The resulting binary sequence is stored in the "quantized" sequence.
To decode the message, we use the "videc" function which provides a Viterbi decoding algorithm. We use "trellis", "trunk", and the "hard" as parameters for the function. The trellis presents the trellis diagram we have created before. We use a trellis to find the most likely message to be the original message. The hard parameter means that we want to decode the message using a hard decision decoding process.
Before running the simulation code, we need to find the number of bit errors that occurred during the transmission. We can do this by comparing the original message "msg" to the decoded message "decoded" using the "sum" and " = " functions in Matlab, and storing the number of bit errors in the "err" variable. Then we compute also the bit error rate (BER) which can help us to evaluate the performance of the technique.

```
% Compute the number of bit errors
err = sum(msg ~= decoded);
disp(['Number of bit errors: ' num2str(err)]);
% Compute the Bit Error Rate
[~, ber] = biterr(msg, decoded);
disp(['Bit Error Rate: ' num2str(ber)]);
```

Figure 4.9: Compute the Bit error rate and number of errors

After running the simulation code multiple times, we will check the number of bit errors that occurred and compare the decoded message to the original message. This will help us determine the effectiveness of the convolutional codes in correcting errors, and its help us to evaluate the convolutional codes performance.

```
>> Convolutionalcode
Number of bit errors: 0
Bit Error Rate: 0
>> msg

msg =

     1     0     1     1     0     1     0     0     1     0

>> decoded

decoded =

  1×10 logical array

   1   0   1   1   0   1   0   0   1   0

fx >>
```

Figure 4.10: Result of first attempt

The figure above displays the result of the first try of running the code, we did not encounter any errors in the message after adding noise and decoding it. This means that the code worked correctly and managed to encode the message and then decode it accurately. We also confirmed that the original message and the decoded message were equal.

To ensure the code's reliability, we will run it again and check if we get the same result in terms of the number of bit errors as the first run.

```
>> Convolutionalcode
Number of bit errors: 1
Bit Error Rate: 0.1
>> msg

msg =

     0     0     1     1     1     1     0     0     0     0

>> decoded

decoded =

  1×10 logical array

   0   0   1   1   1   1   0   0   0   1

fx >>
```

Figure 4.11: Result of the second tray

In the second attempt of running the code, we got a different result from the first attempt. The number of bit errors in the decoded message was equal to 1 and BER is 0.1, which means that one bit was transmitted incorrectly due to the noise added to the encoded message. This caused the decoded message to be different from the original message. The noise in the transmission introduced errors and affected the decoding process.

4.2.1 Evaluating with SNR

The noise-to-signal parameter is very important to evaluate the performance of convolutional codes. First, we run the code to see the result of SNR in our example.

```
>> Convolutionalcode
Number of bit errors: 2
Bit Error Rate: 0.2
Signal to  NOISE Ratio: 4.2597
```

Figure 4.12: SNR

The figure above shows that SNR for our example of convolutional codes is 4.2597 and we see also the number of errors is 1 and the Bit error rate is 0.1 in this example.
To evaluate the performance of convolutional codes in different conditions we can change the SNR by adjusting the power of the noise. In our example, we can change SNR by changing the value of the EbNo parameter. First, we set the EbNo parameter to 9 and run our simulation example. The figure below displays the results.

```
>> Convolutionalcode
Number of bit errors: 0
Bit Error Rate: 0
Signal to  NOISE Ratio: 7.7815
```

Figure 4.13: Higher SNR

As we can see in the figure, the SNR value has increased to 7.7815 and we can see also that the number of errors is 0. The increase in SNR has led to a better performance of the convolutional codes related to correcting errors in the transmitted data.
Next step we set the EbNo 3 and run the code again. The results are shown in the figure below.

```
>> Convolutionalcode
Number of bit errors: 5
Bit Error Rate: 0.5
Signal to  NOISE Ratio: 3.0103
```

Figure 4.14: Lowe SNR

We can see in the figure that the SNR value has decreased to 3.0103 and the number of bit error rate (SER) is 0.5 which has increased. This means that the received signal is now 3.0103 times stronger than the noise in the communication channel. The SNR decrease has led to poorer performance of the convolutional codes in correcting errors in transmitted data.
Following to results we have got in changing the SNR values we can say that the decrease in SNR value will increase the number of errors and the number of bit error rate (BER). That means the convolutional codes perform poorer in low SNR values and it performs better in high SNR values.

4.3 Simulating ARQ Go-Back-N

ARQ is a type of backward error correction. The N in ARQ Go-Back-N presents the sender's window size. In ARQ Go-Back-N the sender can send multiple frames at the same time before receiving the acknowledgment for the first frame from the receiver. The following example is a simulation of ARQ Go-Back-N.
The frames in Go-Back-n ARQ are numbered sequentially. ARQ sends frames multiple times at a time slice. The technique uses the number of frames for distinguishing the frame. These numbers are named sequential numbers. The number of frames for every send of frames is depending on the sender's window size.
To simulate Go-Back-N ARQ, I will use Matlab. This will help me to archive my goal so that the readers of this thesis can understand how the technique works practically after we have explained how it works theoretically. We will add random noise to the simulation of the technique. Adding the noise will help us to evaluate the performance of the technique under noise.
First of all, we will set the simulation parameters, and then we can initialize the counters which we need in our simulation. The figure below shows the code for parameters and the initialization of parameters.

```
% Set up simulation parameters
window_size = 4; % window size
pkt_error_prob = 0.3; % probability of packet error
ack_error_prob = 0.6; % probability of ACK error
pkt_noise_power = 0.2; % power of additive noise on packets
ack_noise_power = 0.3; % power of additive noise on ACKs
timeout = 2; % timeout in seconds

% Initialize counters
sent_packets = 0;
recv_packets = 0;
next_pkt_to_send = 1;
expected_pkt_to_receive = 1;

```

Figure 4.15: ARQ-Go-Back-N settings parameter

As the figure shows, we set the window size equal to 4 and the timeout to 2 seconds. The window size parameter determines the number of packets that can be sent before waiting for the acknowledgment before re-sending the packets. The "timeout" parameter determines the time limit for waiting for an acknowledgment. The "pkt error prob" and "acknowledgment error prob" parameters specify the probabilities of a packet or an acknowledgment being corrupted due to noise in the channel. The "pkt noise power" and "acknowledgment noise power" parameters represent the power of the additive noise on packets and acknowledgments.

```
% Generate packets
packets = randi([0 1], 10, 10); % generate 10 packets, each with 10 bits of data

% Loop through packets
while recv_packets < size(packets, 1)
    % Send packets in window
    for i = next_pkt_to_send:min(size(packets, 1), next_pkt_to_send+window_size-1)
        packet = packets(i,:);
        if rand < pkt_error_prob % simulate packet error
            packet = ~packet;
        end
        packet_with_noise(i,:) = packet + pkt_noise_power*randn(size(packet)); % add noise to packet
        sent_packets = sent_packets + 1;
        fprintf('Sending packet %d\n', i);
    end
```

Figure 4.16: Create a packet and add noise

As we see in the figure above, first we generate packets with 10 bits of data and store them in packets. It uses the MATLAB function "randi()" to generate random integers of either 0 or 1 and create 10 packets with 10 bits of data each. We create a while loop that loops through the packets we have created. Thereafter we create a loop that sends the packets in the window where in our example the window size is 4. The loop continues to send packets until all packets have been received, which is determined by the variable recv-packets" being less than the total number of packets in the "packets" matrix. Within the loop, it selects a subset of packets to send in the current window. It selects the packets starting from "next pkt to send" up to the minimum of either the total number of packets or the next packet to send plus the window size minus one. It then simulates packet errors by flipping the bits of the packet with a probability of "pkt error prob", and adds noise. It updates the "sent packets" counter and prints a message indicating that the packet has been sent.

```
% Wait for ACKs
t_start = tic; % start timer
while true
    if rand < ack_error_prob % simulate ACK error
        continue; % ignore ACK
    end

    % Check for ACK
    ack = packet_with_noise(expected_pkt_to_receive:next_pkt_to_send+window_size-1,:);
    ack_with_noise = ack + ack_noise_power*randn(size(ack)); % add noise to ACK
    if isequal(round(ack_with_noise), round(ack)) % check if ACK matches packets
        fprintf('Received ACK for packets %d-%d\n', expected_pkt_to_receive, next_pkt_to_send+window_size-1);
        recv_packets = recv_packets + (next_pkt_to_send+window_size-1-expected_pkt_to_receive+1);
        expected_pkt_to_receive = next_pkt_to_send;
        break; % exit ACK wait loop
    end
```

Figure 4.17: Acknowledgment check

As we see in the figure above we start a timer for acknowledgment. Thereafter we simulate an error in the acknowledgment number. Then We add the noise to the acknowledgment and then we check if the acknowledgment which been sent is equal to the acknowledgment which been received. If they are equal so we know they have been transmitted successfully. Thereafter we update the received packet and which packets we expect to receive.

```
        % Check for timeout
        if toc(t_start) > timeout
            fprintf('Timeout: resending packets %d-%d\n', next_pkt_to_send, next_pkt_to_send+window_size-1);
            break; % exit ACK wait loop
        end
    end

    % Resend packets if necessary
    if expected_pkt_to_receive > next_pkt_to_send
        next_pkt_to_send = expected_pkt_to_receive;
    end
end

% Display results
SNR_ack = 10*log10(1/ack_noise_power^2);
fprintf('Sent: %d\n', sent_packets);
fprintf('Received: %d\n', recv_packets);
fprintf('Packet loss rate: %.2f%%\n', 100 * (1 - recv_packets / sent_packets));
disp(SNR ack)
```

Figure 4.18: Check for timeout and resend if necessary

After the sender has sent a set of packets we now wait for the acknowledgment for the oldest packet in the window which hasn't yet received its own acknowledgment. The receiver starts a timer and enters into an infinite loop until the acknowledgment is received or a timeout occurs. Within the loop, The receiver checks if the received acknowledgment matches the sequence number of the oldest packet in the window which hasn't yet received its own acknowledgment. If the received acknowledgment matches, it means that packets have been successfully received. Then sender sends new packets in the window. We have done this check in our simulation example by using the "isequal()" function in Matlab. If they are equal, then the "recv-packets" updates with the number of packets received with the correct acknowledgment. If the timer exceeds the timeout value, the sender re-sends all the packets in the current window starting with the oldest packet which hasn't received its own acknowledgment yet. Thereafter we can check if there is a need to resend any of the packets. We check if there is a need to resend the packet by checking if the "expected pkt to receive" variable is greater than the "next pkt to send" variable.

We run the simulation code now and see the result. The result is shown in the figure below.

```
>> ARQ_GO_BACK_N
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sent: 12
Received: 12
Packet loss rate: 0.00%
   10.4576
```

Figure 4.19: ARQ results

The results show that the technique has successfully corrected the errors that occurred in both the packets and acknowledgment during wireless communication. Although the pressure from the noise the technique ensured error-free communication with a packet loss rate of 0, deteriorating that the Go-Back-N ARQ technique performs well in noisy channels.

4.3.1 Evaluating with SNR

The noise-to-signal (SNR) parameter is important to evaluate the performance of the Go-Back-N ARQ. We will run the simulation code with different SNR values and see the results of changing the SNR values. In the Go-Back-N ARQ simulation, there are two different SNR ratios. The first ratio is the SNR packet ratio and the second is the ratio for acknowledgment. The packet ratio will affect the packet loss rate and also the time of running the code. The acknowledgment ratio will affect the time of running the code. First, we run the code with acknowledgment of 0.3 for both the "pkt-noise-power" and "ack-noise-power" parameters. Also, the same powers are in our example.

```
>> ARQ_GO_BACK_N
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sent: 12
Received: 12
Packet loss rate: 0.00%
Signal to  noise for ack: 10.4576
Run time in secounds: 0.37321
Signal to  noise for packets: 10.4576
fx >>
```

Figure 4.20: Acknowledgment SNR and packet SNR

The figure shows the result for Acknowledgment SNR and SNR packet in our example is 10.4576 for both. We see that the time of running is not high which is 0.37921. The packet loss rate is also 0 which means there were no packets lost in transmission. We change now the value of the "ack-noise-power" parameter to 0.5 which leads to the SNR ratio for acknowledgment will lesser. When we run the code the code continues to run and it doesn't end. This means the sender waits for the receiver to send the acknowledgment before continuing the transmission for the rest of the packets. I waited for ca. 10 min and the code doesn't stop running which means that the code may be run to an infinite time. I have noticed that the packets have been sent many times due to the timer expiration. We run now the code with a change of the "pkt-noise-power" parameter to 0.5 which will lead to a lesser SNR packet ratio.

```
>> ARQ_GO_BACK_N
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Timeout: resending packets 1-4
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sent: 16
Received: 12
Packet loss rate: 25.00%
Signal to  noise for ack: 10.4576
Run time in secounds: 2.9423
Signal to  noise for packets: 6.0206
```

Figure 4.21: Lower SNR packet

The figure above shows the results of packets SNR equal to 6.0206 which gave us a packet loss rate of 25 percent and a higher running time which is equal to 20.9423 seconds. We can see also that the sender had to send 4 packets again.
We change the value of both "pkt-noise-power" and "ack-noise-power" to 0.2 which will lead to a higher value of both SNR ratios.

```
>> ARQ_GO_BACK_N
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sending packet 1
Sending packet 2
Sending packet 3
Sending packet 4
Received ACK for packets 1-4
Sent: 12
Received: 12
Packet loss rate: 0.00%
Signal to  noise for ack: 13.9794
Run time in secounds: 0.026572
Signal to  noise for packets: 13.9794
```

Figure 4.22: Hiegher acknowledgment SNR and packet SNR

The figure above shows the results of packets SNR and acknowledgment ratios equal to 13.9794. Higher SNR ratios have affected both the packet loss rate and run time of the code. We see that the run time is lesser than the previous change examples. The run time is low which is equal to 0.026573, and the packet loss rate is equal to 0.0 percent and all the sent packets have been successfully transmitted.

We can evaluate that the higher SNR of the acknowledgment will lead to a lesser time of run time and the higher SNR of the packets will lead to a better packet loss ratio and a lesser time for the running code. When the SNR for the acknowledgment was low. The run time was higher and in some situations, the code didn't stop running for over than ca. 10 min. The receiver just kept resending the packet after the timer limit expired. In these situations, I have assumed that the Go-Back-N ARQ simulation code may run to infinity. The low value of the SNR packet ratio leads to a higher run time because of the packets which have been lost under transmission and have been sent again to the receiver. That has also led to a higher loss packet ratio. We can also assume that the lower SNR leads to worse performance of the technique. That means the ARQ Go-Back-N has better performance in a higher SNR value.

Chapter 5

Conclusions

In this thesis, I have presented the problem that can happen in wireless communication. To correct and detect the errors that occur in the data that have been transmitted we need to use wireless error control techniques. In the chapter background, I have presented several techniques of wireless error control which have helped us to understand how these techniques work theoretically.

I have also presented several studies about which tool can we use to simulate those techniques. We have also presented the different tools we can use to create a simulation of some wireless error control techniques. The tools we have presented are Matlab, NS-3, OPNET modeler, and OMNET ++. In the comparison of tools part, we have gone through which tool is better for us to use to simulate some of these techniques. I have chosen Matlab because it's a suitable tool to simulate wireless error control techniques. Also, Matlab is an open-source and free program. I also have mentioned that one of the reasons I have to choose MATLAB over the other tools is because of the Matlab programming language I have been familiar with before.

In the Design and solution part, I simulated three different techniques. Two techniques of these three use the forward error correction and one of these techniques uses the backward error correction. The two forward error correction was the hamming code and convolutional codes. The technique we simulated also which is a type of backward error correction was Go-Back-N ARQ. I have used Matlab to simulate all of these three techniques. We have first gone through how we gonna simulate each of these three techniques and the goals of creating the simulation. Then we presented the code we used to simulate each of these three techniques. We have also added noise to the techniques of simulated communication which has helped us to study these technique's performance under a noisy channel. We have also changed the SNR to see how the changing of the SNR wanna affect the performance of the techniques. We have found that the hamming code and ARQ Go-Back-N were more stable than the convolutional codes technique.

We have also found out that a higher SNR will lead to better performance for these techniques for correcting errors. Also, we found when the SNR is low the performance gets worse in all of the three techniques we have simulated.
This study has some limitations, in this study, we have simulated only three techniques. There are maybe other techniques that are worth exploring. Also, the real word wireless channels may have more complex and dynamic characteristics which could affect the performance of the wireless error control techniques and wireless communication. Therefore, further research could focus on exploring the scope of the wireless error control techniques and also studying the techniques in more realistic conditions. The work we have presented in this thesis contributes to advancing our understanding of wireless error control techniques and provides a base for further research on wireless error control techniques.

Appendix A

Instructions to Compile and Run System

In this thesis, we have used Matlab as the simulation tool. We will provide in this appendix installation instructions of Matlab.

1. Get the license file and installation key.

If you are a student you may get a license through your university contact the IT department in your university to get the license information.

2. Start the installer.

Start the installation by running the "setup" file.

3. Accept the license agreement.

Read the license agreement which is presented by the installer and accept it.

4. Enter the file installation key.

Enter the installation key when it is prompted by the installer.

5. Select the license file.

Select the license you have chosen in step one.

6. Select the destination folder.

Select the folder you want to install Matlab in.

7. Select products.

Choose the products you need to install. In our simulations example we needed only Matlab.

8. Select options.

Select additional options such as the toolboxes.

9. Confirm selections and install. Check your selections and confirm to start installing.

When the Matlab is installed you can open the program. You have to log in with your Mathworks email and password if you don't have you can create a new Mathworks user and you have to create a password. If you are a student you can log in with your student number as a user email.

Bibliography

• A Case Study of Various Wireless NetworkSimulation Tools By V. Venkataramanan1, S. Lakshmi. 2018.

• A Review of Simulation Techniques for Some Wireless Communication System BY Bodunrin Isa Bakare, Joseph Diema Enoch, 2019.

• A comparative study of different network simulation tools and experimentation platforms for underwater communication By Tejaswini R. Murgod, S. Meenakshi Sundaram, 2021.

• https://en.wikipedia.org/wiki/Error$_c$*orrection*$_c$*ode*

• https://en.wikipedia.org/wiki/Hamming$_c$*ode*

• https://www.javatpoint.com/go-back-n-arq

• https://www.javatpoint.com/computer-network-error-correction

• DAT610-Wireless Communication lecuture4

• https://se.mathworks.com/products/matlab.html

• https://en.wikipedia.org/wiki/MATLAB

• https://www.nsnam.org/about/what-is-ns-3/

• https://opnetprojects.com/opnet-network-simulator/

• https://en.wikipedia.org/wiki/OMNeT

• https://en.wikipedia.org/wiki/Hamming$_c$*ode*.

• https://se.mathworks.com/products/new$_p$*roducts*/*latest*$_f$*eatures.html*