# Variable Rate Transmission Over Noisy Channels

*Christoph Burger-Scheidlin*


Doctor of Philosophy

Institute for Digital Communications
School of Engineering and Electronics
The University of Edinburgh

May 2009

# Abstract

Hybrid automatic repeat request transmission (hybrid ARQ) schemes aim to provide system reliability for transmissions over noisy channels while still maintaining a reasonably high throughput efficiency by combining retransmissions of automatic repeat requests with forward error correction (FEC) coding methods. In type-II hybrid ARQ schemes, the additional parity information required by channel codes to achieve forward error correction is provided only when errors have been detected. Hence, the available bits are partitioned into segments, some of which are sent to the receiver immediately, others are held back and only transmitted upon the detection of errors. This scheme raises two questions. Firstly, how should the available bits be ordered for optimal partitioning into consecutive segments? Secondly, how large should the individual segments be?

This thesis aims to provide an answer to both of these questions for the transmission of convolutional and Turbo Codes over additive white Gaussian noise (AWGN), inter-symbol interference (ISI) and Rayleigh channels. Firstly, the ordering of bits is investigated by simulating the transmission of packets split into segments with a size of 1 bit and finding the critical number of bits, i.e. the number of bits where the output of the decoder is error-free. This approach provides a maximum, practical performance limit over a range of signal-to-noise levels. With these practical performance limits, the attention is turned to the size of the individual segments, since packets of 1 bit cause an intolerable overhead and delay. An adaptive, hybrid ARQ system is investigated, in which the transmitter uses the number of bits sent to the receiver and the receiver decoding results to adjust the size of the first, initial, packet and subsequent segments to the conditions of a stationary channel.

# Acknowledgements

This thesis marks the end of four years of work at the Institute for Digital Communications at the University of Edinburgh. Looking back, I still cannot quite believe that I have indeed managed to pull through. I can say with certainty that I would not have managed without the advise and support of my supervisors. **Norbert Görtz**, who always had an open door and took the time out of his busy schedule to help me move along, develop my ideas and patiently listening to the many strange ones that almost never worked out. **Steve McLaughlin**, who always kept an eye out for my progress and most of all kept me on my toes using nothing more than his reputation, his to-the-point questions and, occasionally, just a simple "prove it!". Finally **Tim Farnham**, who provided insights from a different perspective and allowed me to gain experience at Toshiba's Telecommunications Research Lab. Most of all, I am immensely thankful to my supervisors for not losing trust in me. In a time when all I could do was stumble, they kept me on my way and focused towards finishing, which now, better late than never, I have finally finished. I would like to use this opportunity to also thank Toshiba Research Europe for their support of this work. I am most grateful for having had the possibility to exchange ideas present my work in an industrial setting, as well as their financial support of me during my PhD.

Working in an office full of PhD students has its own dynamics with the "old hands" helping the newcomers find their way around. During my stay in Edinburgh, I was privileged to meet and become friends with Peter Hillman and fondly remember the lunch time rants and the excessively technology-driven solutions that we came up with for even such mundane things as labelling a parcel. I would also like to thank Mostafa Afgani and Zubin Bharucha, who both, despite being loaded with work, took the time to help me get this thesis printed and submitted.

Finally, there is nothing in this world more important to me than family. My warmest thanks and deepest gratitude belongs to all those that have lent support when it was most desperately needed. I consider myself fortunate to have such a family and group of friends that never ceased to support and encourage me. I love you all.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text. This work has not been submitted for any other degree or professional qualification except as specified.

(*Christoph Burger-Scheidlin*)

In loving memory of my father

# Table of Contents

# List of Figures

# List of Tables

# List of Algorithms

# List of Acronyms

ACK . . . . . ACKnowledgement Feedback message in ARQ systems indicating error-free reception of a data packet.

ARQ . . . . . Automatic Repeat reQuest Protocol to automatically request retransmission for missing or erroneous packets.

AWGN . . . . . Additive White Gaussian Noise Commonly used channel model consisting of only a Gaussian distributed noise sample which is added to the transmission.

BCJR. . . . . BCJR Algorithm to optimally decode a linear code. Named after the initials of the authors, Bahl, Cocke, Jelinek and Raviv, who proposed it.

BER. . . . . Bit Error Ratio The ratio of errors per *input bit* observed at the decoder.

BIAWGN. . . . . Binary Input AWGN Additive white Gaussian noise channel with the limitation that the input can only take one of two values.

BPSK . . . . . Binary Phase Shift Keying Modulation method with the mapping $0 \rightarrow +1$ and $1 \rightarrow -1$.

BRCS . . . . . Block-Randomised Code Structure Ordering strategy that takes into account the code structure and randomises blocks of bits made up of systematic and parity bits separately.

CRC. . . . . Cyclic Redundancy Check An error-detecting code that uses the remainder of a polynomial division to ensure integrity of the input data.

FEC . . . . . Forward Error Correction  Error control method that uses a channel code to correct errors at the receiver.

FSM . . . . . Finite State Machine  A machine which can be described entirely by the transition among a limited number of states.

GSM . . . . . Global System for Mobile Communications  Telecommunication standard for cellular, mobile communication.

HARQ . . . . . Hybrid ARQ  Combination of ARQ with FEC.

ISI . . . . . Inter-Symbol Interference  Interference where the channel output is also dependent on the adjacent symbols put in the channel.

LLR . . . . . Log-Likelihood Ratio  Soft measure of reliability calculated by the logarithm of the ratio of the probability of two possible outcomes.

LOG-MAP . . . . . Log-MAP  Implementation of the BCJR algorithm using LLR values.

MAP . . . . . Maximum A-Posteriori

NAK . . . . . Negative AcKnowledgement  Feedback message in ARQ systems indicating reception of a data packet with errors.

RSC . . . . . Recursive Systematic Convolutional  Class of convolutional codes, where the first generator polynomial is used not as an output bit but added to the input bit.

SNR . . . . . Signal-to-Noise Ratio  Ratio of the signal power to the transmission noise power.

# Nomenclature

| | |
|---|---|
| $n_{\text{bo}}$ | Number of packets to back-off after successful transmission |
| $c \dashleftarrow a$ | A backward transition potentially involving multiple steps from $a$ to $c$ |
| $b \leftarrow a$ | A backward transition from $a$ and $b$ |
| $\overleftarrow{a}$ | Indicates that $a$ belongs to a backward iteration |
| $r_{\text{base}}$ | Rate of the unpunctured base code |
| $p_\varepsilon$ | Bit error probability |
| $L$ | Block length |
| $\mu(b)$ | Metric of a transition (branch) in a trellis |
| $a$ | Channel coefficient |
| $m$ | Channel memory |
| $\mathbf{U}$ | Output of the encoder as a matrix |
| $c_{t,i}$ | $i^{\text{th}}$ code bit at time $t$ |
| $D$ | Delay element, indicating that a bit from one time index $t$ earlier is used. |
| $\mathbf{P}_\Delta$ | A differential puncturing table, i.e. the difference between two puncturing tables |
| $\hat{A}$ | Indicates that A has been distorted |
| $A'$ | Indicates that A has been encoded |
| $\mathbf{u}$ | Sequence of encoded bits |

$u$ Encoded bit

$\lambda_{\text{ext}}(\cdot)$ The extrinsic L-value of $\cdot$, used in Turbo decoding

$\overrightarrow{a}$ Indicates that $a$ belongs to a forward iteration

$\times$ A puncture

$\mathbf{I}$ For matrices the identity matrix, for permutations the identity permutation $1,2,3,..n$

$n_{\text{lim}}$ Maximum number of increments in a delay-limited transmission system

$\mathbf{x}$ Sequence of input bits

$x$ Input bit

$\pi$ An interleaver, used e.g. in a Turbo Code.

$\lambda(\cdot)$ The L-value of $\cdot$

$\boxplus$ Sum over L-values, see equation 2.10.

$\varepsilon_{\text{max}}$ Maximum error in rate of a puncturing scheme

$r_{\text{max}}$ Maximum rate that a transmission system is able to achieve at high SNR

$M$ The modulation alphabet of a transmission

$\sigma_z^2$ Variance of Gaussian noise

$\mathcal{N}(a,b)$ A normal distribution with mean $\mu = a$ and variance $Variance = b$

$\mathbf{0}$ Matrix or Vector containing only 0.

$n_{\text{bits}}$ Number of bits

$n_{\text{inc}}$ Number of transmitted increment packets

$n_{\text{packets}}$ Number of transmitted packets

$\tilde{n}_{\text{states}}$ Number of possible states

$n_{\text{req}}$ Number of transmitted increment packets

$n_{\text{sent}}$ Number of sent bits

$n_{\text{states}}$ Number of states

$\mathbf{1}$ Matrix or Vector containing only 1.

$\check{r}$ Optimal rate of a code

$n$ Constraint length of an RSC code

$\lambda_{\text{pri}}(\cdot)$ The prior L-value of $\cdot$, used in Turbo decoding

$p(\cdot)$ The probability of $\cdot$

$\mathbf{w}$ Sequence of bits where some bits have potentially been punctured

$w$ Bit that is part of a sequence that has been punctured

$\mathring{r}$ The rate of a punctured code

$\mathring{\mathbf{U}}$ Indicates that the output of the encoder, expressed as a matrix, has been punctured

$\mathring{A}$ Indicates that A has been punctured

$P$ The puncturing period of a rate-compatible puncturing table

$\mathbf{P}$ A puncturing table

$r$ Rate of a code

$N$ Denominator of a rate $r = 1/N$ code

$a$ Rayleigh Fading factor

$\mathbf{v}$ Sequence of reordered bits

$v$ Bit after being reordered

$\prod$ A permutation that defines a transmission order

$s_{\mathbf{u}}$ Size of a sequence of bits produced by the encoder

$s_{\text{inc}}$ Size of an increment

$s_{\text{initial}}$ Size of an initial packet

$s_{\mathbf{x}}$ Size of a sequence of input data

$s_{\check{\mathbf{w}}}$ Size of a punctured sequence that is optimally punctured

$s_{\text{packet}}$ Size of a packet

$s_{\mathbf{w}}$ Size of a punctured sequence

$s_{\mathbf{v}}$ Size of a reordered sequence

$c_s$ Certainty of a state

$q_t^{(i)}$ Output of encoder tap $i$ at time index $t$

$\mu(s)$ Metric of a state in a trellis

$s_t$ Systematic bit at time $t$

$t$ Time index for discrete cases, continuous time otherwise

$c_b$ Certainty of a transmission

$a \dashrightarrow c$ A transition potentially involving multiple steps from $a$ to $c$

$a \rightarrow b$ A transition between $a$ and $b$

$\sigma^2$ Variance

$a \boxplus b$ Addition of the L-values $a$ and $b$, see equation 2.9

$A \circledcirc B$ Operator to indicate that $A$ is punctured with pattern $B$

# Chapter 1

# Introduction

Digital information has become part of our daily lives. Its use has become second nature, the desire to transmit and access this information is constantly increasing. Soon, unhindered, wireless access will be commonly expected. Any wireless transmission, however, bears the possibility of corruption through errors, which can be catastrophic due to the efficiency with which analogue signals are turned into digital data. Consider the effect of a single bit-error on a frame of the "Foreman" sequence, encoded with the JM reference implementation, version 10.1, of the H.264/AVC standard [61], the latest development in video coding (see e.g. [138] for an overview). A darkening of the background (1.1.2) or a slightly distorted frame (1.1.3) might sometimes still be acceptable. The image in Figure 1.1.4, however, bears little resemblance to the original frame (1.1.1). As mobile devices, such as digital camcorders, become smaller, storing data on the device itself becomes increasingly difficult. A wireless link can be used to offload the recorded data to a non-mobile, therefore cheap, storage location such as a home server or even a remote hosting provider. Data corruption during transmission, possibly leading to an image such as 1.1.4 as the *only* copy, presents a serious obstacle.

(1) Original Frame (2) Darkened area (3) Moved Blocks (4) Unrecoverable

Figure 1.1: Effect of single bit error on an H.264/AVC encoded frame.

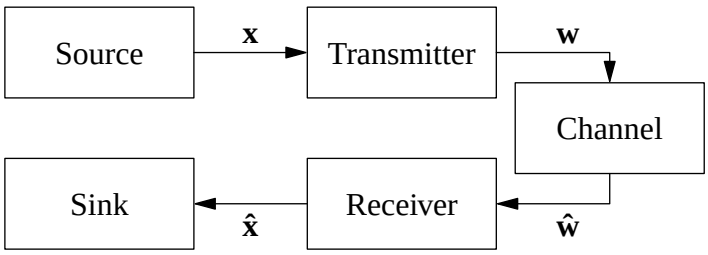



Figure 1.2: Generalised communication system.

Figure 1.2 shows a simple block diagram of a generalised transmission system. A transmitter takes the information sequence $\mathbf{x}$ and processes it to form $\mathbf{w}$, suitable for transmission. The sequence $\mathbf{w}$ is affected by the transmission channel and observed at the receiver as $\hat{\mathbf{w}}$, which is processed to yield $\hat{\mathbf{x}}$. A reliable transmission system needs to cope with the errors that occur during transmission and ensure that $\mathbf{x} = \hat{\mathbf{x}}$. Preventing transmission errors can be achieved through two distinct approaches. Detection of errors at the receiver, triggering a retransmission of the data, and error correction by the receiver requiring extra, redundant, information to be sent by the transmitter.

In error detection, the basis of Automatic Repeat reQuest (ARQ), a detected error triggers a retransmission of the data, $\mathbf{w}$. Error detection requires only a small amount of extra information, such as a Cyclic Redundancy Check (CRC) sum. However, ARQ also requires a (reliable) communication link from the receiver back to the transmitter in order to trigger retransmission. Error detection with retransmissions through ARQ works well for low error probabilities or block errors that corrupt entire packets and has been used successfully in the Transmission Control Protocol (TCP), one of the underlying protocols of the internet.[1] Error correction provides a method to recover the original data, $\mathbf{x}$, from the potentially corrupted sequence $\hat{\mathbf{w}}$. Contrary to error detection, error correction does not require a link from the receiver to the transmitter, however, in order to successfully correct errors, the transmitter needs to add a significant amount of information to the transmitted sequence $\mathbf{w}$ making it much larger than the information sequence $\mathbf{x}$. This information needs to be added regardless of the occurrence of errors because the transmitter does not have any knowledge about what happens on the channel. This is the reason why this type of error correction is referred to as Forward Error Correction (FEC), used e.g. in the Global System for Mobile Communications (GSM).

[1] ARQ in TCP also protects against packet losses on the channel, however this thesis deals with bit-errors only so discussion of that aspect is omitted

Error detection and retransmission via ARQ and FEC via channel codes both have one respective shortcoming. ARQ relies on a low probability of errors on the transmission channel. If an error occurs the transmission is discarded and the receiver waits for the arrival of the retransmission. Yet, no guarantee can be given that the *retransmitted* packet is going to be error-free. If it is not, the receiver requests another retransmission, waiting again for the, hopefully error-free, retransmitted packet. The time spent on waiting for packets to arrive and signalling the transmitter to start a retransmission significantly increases the latency of the *reliable* transmission of the input data $\mathbf{x}$. Moreover, the retransmission of packets occupies time on the channel and once the error rate on the channel is high enough so that each transmitted packet experiences at least one error, reliable transmission with ARQ is not possible any longer. FEC on the other hand is a suitable approach for channels with errors. The redundant information added by the encoder is used to correct errors that occurred during transmission, as long as the used code is strong enough. Channel codes cannot correct all error patterns and there is always a chance that residual errors affect the transmission. Furthermore, the rate of the channel code is fixed. Thus, for reasonably good channels redundant bits are transmitted without benefiting the receiver.

Hybrid-ARQ combines the strengths of both FEC and ARQ. In Hybrid ARQ (HARQ) transmission systems, transmitted data is protected by a channel code but retransmission is possible should the channel code be insufficient to correct a particular error pattern. Thus a (weaker) channel code, needing fewer redundant bits, can be used that corrects only the average case of errors. The ARQ mechanism, on the other hand, is used to handle the case of uncorrectable errors. Such a type-I Hybrid ARQ system works very well for a fairly constant noise level, since the channel code corrects these errors and the ARQ mechanism is only needed occasionally in order to correct those error patterns that the channel code fails to correct [76]. Alternatively, the transmission system can be modified to only deliver the redundant information used by the channel code once errors are detected. Retransmission of the entire packet does not take place. Instead, the receiver keeps the corrupted first packet which contains one part of the total transmission sequence. Additional information to correct transmission errors is supplied in the second packet (the former retransmission). These transmission systems with *incremental redundancy* are called type-II Hybrid ARQ schemes.

Transmission systems employing FEC and type-I HARQ are typically static in the sense that their parameters are fixed for the transmission. However, it is often desirable to adapt the information rate of the transmission to the current situation. Generally, two distinct reasons can be given for situational changes. Firstly, the demands of the user have changed and more or less data needs to be transmitted. In transmission systems that use some sort of multi user multiplexing, such as time, frequency or code division multiple access schemes, it is simple to react to such a demand by allocating more time slots, frequency bands or spreading codes, respectively, to the user requiring transmission. The second reason for wanting to adapt the information rate is that of changing channel quality. Typically, transmission systems will exhibit times when a lot of data can be transmitted over them with relative ease and some times when transmission is difficult, for example when moving indoors with a wireless device. A transmission system with its parameters fixed at design time will necessarily cater to some minimum service level, wasting resources when the channel exhibits better quality. Several approaches are possible to exploit this increased channel quality. Firstly, the rate of the FEC code used by the transmission system can be changed by e.g. replacing the channel code. Secondly, the FEC method can be adjusted. A common such adjustment is the change of puncturing pattern, if a punctured code is used. This is the approach taken in this thesis. Thirdly, the constellation size of the modulation scheme can be switched to one that transmits more information bits per transmitted symbol. This typically requires some explicit signalling to the receiver, however, in order to notify the receiver of the changed modulation scheme. Adapting the puncturing, on the contrary, can be implemented simply by transmitting fewer or more bits. Lastly, in a transmission system with multiple antennas, the space time codes employed in such systems can be switched to favour throughput instead of diversity. Clearly, the four methods listed are somewhat exclusive, since all of them “use” some portion of the channel quality in order to effect more throughput. Or put differently, using these methods leads to an increase in error probability of the transmitted data, so employing all methods at the same time is often not practical. In this thesis, the adaption of the transmission rate via adaption of the puncturing pattern is studied in an incremental redundancy, type-II, ARQ system. Since the redundancy, in the form of parity bits, is transmitted incrementally, two questions arise naturally. Firstly, how should the available bits be ordered for optimal partitioning into consecutive segments which are then transmitted incrementally? Secondly, how large should the individual segments, i.e. the increments, be? In answering these two questions, this thesis contributes

- an illustration of the shortcomings of plain (unchanged) and random strategies that do not take the code structure into account,
- a novel ordering strategy which is independent of the used code polynomial for turbo codes and with minimal adjustments for Recursive Systematic Convolutional (RSC) codes that works for AWGN, ISI and Rayleigh channels independent of the blocksize of the code,
- an extension of the developed ordering strategy that is deterministic and that shows no significant performance loss for turbo codes,
- an explanation of the different nature of systematic and parity bits for RSC and Turbo codes how this difference affects the performance of punctured cods,
- a practical strategy to determine the optimal assignment of generator polynomial pairs to $g^{(0)}$ and $g^{(1)}$, eliminating 50% of possible generator pair assignments when searching for optimal Turbo Codes,
- an algorithm to bound the performance of a particular transmission order by way of considering the recoverability of erasures,
- empiric evidence of the importance of systematic bits for Turbo Codes, refuting a statement by Rowitch and Milstein[109] that claims otherwise,
- an empiric display of a waterfall behaviour of Turbo Codes that occurs not with increasing Signal-to-Noise Ratio (SNR) but with decreasing *rate* at fixed SNR
- a packetisation strategy that achieves good performance and low delay for low SNR channel conditions with fixed packet sizes by simply aggregating smaller packets into one initial transmission,
- an adaptive packet size selection algorithm that allows good performance at low delay to be achieved even at high SNR.

The rest of this thesis presents these contributions in more detail. The presentation of the topics is as follows:

**Chapter 2** introduces concepts of information theory, and coding theory in particular, that assist in understanding this thesis as well as present the notations used throughout this thesis. Topics covered are transmission channels, forward error correction and the

use of the BCJR algorithm, followed by automatic repeat requests and the two hybrid ARQ schemes. Finally puncturing is discussed since it forms the foundation on which incremental redundancy transmission systems are built.

**Chapter 3** is dedicated to the investigation of RSC codes. Strategies are discussed that can be used to define an order of transmission on the bits produced by the channel encoder. The straightforward approaches of not reordering the encoder output at all or randomising the output do not lead to the desired performance characteristics. The reason for this lies in the ability of the RSC code to recover from bit-erasures and chapter 3 provides an upper bound for the performance of the random ordering strategy. Using the rate-compatibility requirement proposed by Hagenauer[46] and taking into account the structure of an RSC code and the insights from the poor performance of the random ordering strategy, a new ordering strategy is developed based on the layout of a binary tree in order to obtain a maximum rate of $r_{\max} = 1$ for high quality channel. It is then shown that the performance of this ordering strategy is not affected by constraint length, input blocksize and channel type. However, transmitting systematic and parity bits first can increase the performance of the ordering scheme. It is shown, that the choice of systematic or parity bits depends on the choice of generator polynomials and that the priority of systematic or parity bits is dependent on the assignment of the generator polynomials to the recursive ($g^{(0)}$) or parity-producing ($g^{(1)}$) polynomial.

Following the investigation of RSC codes, ordering strategies for the more powerful Turbo Codes are evaluated in **chapter 4**. Even though Turbo Codes use RSC codes as their constituent encoders, due to the use of sequential decoding, the systematic bits play a more important role for Turbo Codes. Like for RSC codes, blocksize, choice of generator polynomials and channel type are investigated in this chapter, showing that the developed ordering strategy can also be used successfully for Turbo Codes. However, the additional constituent encoder of the Turbo Code allows for a larger selection of bit combinations. It is shown that it is essential for Turbo Codes that the systematic bits are transmitted first and without omission, refuting the claim made by Rowitch and Milstein[109] that it is beneficial to omit systematic bits in favour of parity bits. In a comparison with rate-compatible punctured Turbo Codes, it is shown, that this claim arises solely due to the limitations of rate-compatible puncturing tables and that the *unoptimised* ordering strategy performs equally well or better than the rate-compatible puncturing tables proposed by Rowitch and Milstein[109] that are *optimised for a particular pair of generator polynomials and blocksize*.

Chapters 3 and 4 introduced the order in which bits should be transmitted. The performance results of these chapters, however, were based on the presumption that 1 bit accurate adjustment of the number of bits could be achieved. Of course, this is not a presumption that holds up in real life. **Chapter 5** considers the effect of packet size on the performance of a transmission system. The relationship between the transmission of the first packet and further incremental packets. Straightforward transmission of fixed packet sizes leads to suboptimal performance at high SNR and a prohibitively high delay due to the transmission of many increments. Two algorithms are developed. The first one simply aggregates a number of increments into the initial packet in order to address the delay issue. In order to also achieve good rate performance at high channel SNR, an algorithm is developed that uses the information gathered from previous transmissions in order to adapt the size of all transmitted packets.

Finally, **chapter 6** gives a summary of this thesis and provides an outlook on further areas of improvement and interest.

# Chapter 2

# Error Control for Noisy Channels

Efficient transmission of data over noisy channels has been the topic of intensive research since Shannon showed that reliable transmission is always possible [116]. This chapter introduces concepts of information theory, and coding theory in particular, as they are relevant to this thesis - paraphrasing Newton the giants on which this thesis stands. **Section 2.1** introduces some notation and basic concepts. **Section 2.2** presents the three transmission channel models that are used in this thesis. Afterwards the prevention of transmission errors through error control coding is presented. The use of Recursive Systematic Convolutional,used in chapter 3, and Turbo Codes,used in chapter 4, to achieve Forward Error Correction is presented in **section 2.3**. Preventing errors that affect the receiver is only one of two possibilities. The use of error detection so that erroneous data can be retransmitted through Automatic Repeat reQuest schemes is presented in **section 2.4**. After it was realised that the performance of ARQ was often not satisfactory and FEC was unable to provide sufficient system reliability [14], the two concepts were combined to form Hybrid ARQ transmission systems. **Section 2.5** introduces and reviews hybrid ARQ and how the feedback channel can be used to adapt ARQ to conditions on the transmission channel. Finally, **section 2.6** explains in detail how puncturing can be used to form ensembles of code and presents its logical extension to a transmission system with variable, adaptive rate. The material in this chapter is only meant as a brief introduction to the presented topics with a narrow focus on the relevance to this thesis. For further, in-depth treatment of the presented topics the reader is referred to textbooks on the subject such as [41, 60, 75, 96].

## 2.1 Preliminaries

This section presents some of the fundamentals of information theory relevant to this thesis and useful in better understanding the concepts presented later in this chapter. It also provides notation and definition that are used in the later chapters.

### 2.1.1 Signal-to-Noise Ratio

The signal-to-noise ratio of a transmission channel is the ratio of energy per bit of a signal transmitted over a channel to the spectral density $N_0$ of the noise on the transmission channel. Two separate but connected definitions exist. The ratio $\frac{E_s}{N_0}$ is defined with respect to the energy of *each channel input* bit $w$ (the energy per symbol). The second ratio $\frac{E_b}{N_0}$ is defined as the energy per *input bit* $x$ (the energy per information bit). Using a channel code of rate $r$, the two ratios relate such that

$$\frac{E_s}{N_0} = r \cdot \frac{E_b}{N_0}. \tag{2.1}$$

In the context of variable rate transmission, described in section 2.6.4, the ratio of $\frac{E_b}{N_0}$ cannot be used on the abscissa, since it depends on the transmission rate $r$ which is not known beforehand. Therefore, whenever signal-to-noise ratio or SNR is mentioned in this thesis, it refers to the ratio $\frac{E_s}{N_0}$.

### 2.1.2 Log-Likelihood Ratios

Working with binary random variables, $w \in \{+1, -1\}$, is often more convenient if the random variables are represented as log-likelihood ratios rather than probabilities [51]. The magnitude of Log-Likelihood Ratio (LLR) values provides an intuitive measure of the reliability of the data and the sign of the LLR provides the hard-decision, should it be required. Moreover, algorithms that work on LLR rather than probabilities often avoid numerical problems that can arise when working with very small probabilities. The Log-Likelihood Ratio of $w$ is defined with respect to the probabilities of $w$ as

$$\lambda(w) = \log_e \left( \frac{p(w=+1)}{p(w=-1)} \right). \tag{2.2}$$

If the variable $w$ is conditioned on another variable $\hat{w}$, the conditional log-likelihood ratio is defined as

$$\begin{aligned}\lambda(w|\hat{w}) &= \log_e\left(\frac{p(w=+1|\hat{w})}{p(w=-1|\hat{w})}\right)\\ &= \log_e\left(\frac{p(w=+1)}{p(w=-1)}\right)+\log_e\left(\frac{p(\hat{w}|w=+1)}{p(\hat{w}|w=-1)}\right)\\ &= \lambda(w)+\lambda(\hat{w}|w)\,.\end{aligned} \tag{2.3}$$

It follows from Bayes rule that if $p(w=+1)=p(w=-1)=1/2$

$$\lambda(\hat{w}|w)=\lambda\left(\frac{p(\hat{w}|w=+1)}{p(\hat{w}|w=-1)}\right)=\lambda\left(\frac{p(w=+1|\hat{w})}{p(w=+1|\hat{w})}\right). \tag{2.4}$$

Two channel coding techniques are greatly simplified by log-likelihood ratios. Firstly, when puncturing code bits, the receiver can replace punctured code bits with an LLR of 0. Since an LLR of 0 indicates that no information about the bit is available, the algorithm can remain unmodified and unaware of the puncturing of bits. Secondly, code combining proposed by Chase[20] can be easily achieved using log-likelihood ratios. Given two statistically independent transmissions of the equiprobable binary variable $w\in\{+1,-1\}$

$$p(w|\hat{w}_1,\hat{w}_2)=\frac{p(\hat{w}_1|w)\cdot p(\hat{w}_2|w)\cdot p(w)}{p(\hat{w}_1,\hat{w}_2)} \tag{2.5}$$

with

$$p(\hat{w}_i|w)=\frac{1}{\sqrt{2\pi}\cdot\sigma}\cdot e^{-\frac{1}{2\sigma^2}\cdot(\hat{w}_i-a_i\cdot w)^2}, \tag{2.6}$$

then

$$\begin{aligned}\lambda(w|\hat{w}_1,\hat{w}_2) &= \log_e\frac{p(\hat{w}_1|w=0)\cdot p(\hat{w}_2|w=0)}{p(\hat{w}_1|w=1)\cdot p(\hat{w}_2|w=1)}+\lambda(w)\\ &= \log_e\frac{p(\hat{w}_1|w=0)}{p(\hat{w}_1|w=1)}+\log_e\frac{p(\hat{w}_2|w=0)}{p(\hat{w}_2|w=1)}\\ &= \lambda(w|\hat{w}_1)+\lambda(w|\hat{w}_2)\,,\end{aligned} \tag{2.7}$$

where the last line follows only for equiprobable input bits $w$.

One problem when working with log-likelihood ratios is the addition of two log-likelihood ratios $\lambda_1+\lambda_2$. The straight-forward formula

$$\lambda_{1+2}=\log_e\left(e^{\lambda_1}+e^{\lambda_2}\right) \tag{2.8}$$

returns to the probability domain via $e^{\lambda_1}$ and $e^{\lambda_2}$. However, equation (2.8) can be solved using the Jacobian logarithm [107] such that

$$\lambda_{1+2}=\log_e\left(e^{\lambda_1}+e^{\lambda_2}\right)=\max(\lambda_1,\lambda_2)+\log_e\left(1+e^{-|\lambda_1-\lambda_2|}\right)\triangleq\lambda_1\boxplus\lambda_2. \tag{2.9}$$

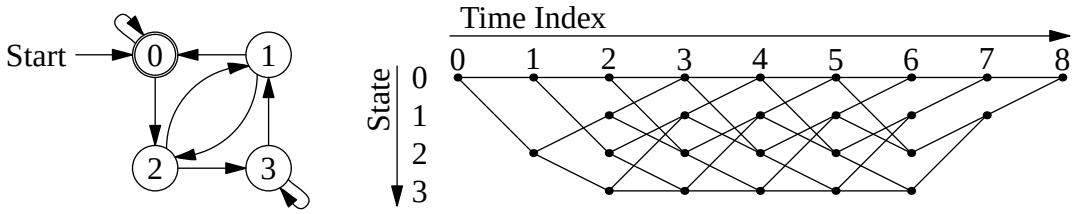



Figure 2.1: Example state machine and corresponding trellis of all valid transition sequences, if the state machine ends in state 0 after 8 state transitions.

Using the short-hand notation $\lambda_1 \boxplus \lambda_2$ from equation (2.9), the summation over multiple LLR values is defined as

$$\sum_{i=1}^{N}\boxplus \lambda_i = ((((\lambda_1 \boxplus \lambda_2) \boxplus \lambda_3) \boxplus \cdots) \boxplus \lambda_N) \tag{2.10}$$

Equations (2.9) and(2.10) play an important role in implementing an efficient version of the BCJR algorithm, presented in section 2.3.2.

### 2.1.3 Trellis Representation of State Machines

Closely related to both convolutional encoders (see section 2.3.1) and ISI channels (see e.g. [125]) is the concept of a Finite State Machine (FSM). An important concept that facilitates the working with and understanding of the processes of finite state machines is that of a trellis, introduced by Forney[35]. If a state machine undergoes $i$ state transitions, then it is possible to draw a two dimensional representation of *all possible* sequences of states $s_0, s_1, \ldots, s_i$, called a trellis. A trellis is basically a grid of nodes, one for each state $s$ per time index $t$. Conventionally, the states are stacked vertically. This stack is replicated vertically as often as it is required. Two states $s_t$ and $s'_{t+1}$ are joined by a directed edge in the trellis, if $s \rightarrow s'$ is a valid state transition of the finite state machine. The trellis is thus a directed, acyclic graph and the sequence of states $s_0, s_1, \ldots, s_i$ can be described as a path in the trellis. Figure 2.1 shows a state machine and the corresponding trellis for 8 transitions.

### 2.1.4 Soft Decoding – The BCJR Algorithm

A received sequence, $\hat{\mathbf{w}}$, often needs to be processed by the receiver. It is advantageous to process the data with soft values (i.e. taking into account the reliability of the received data, e.g. by using probabilities or LLR values) if possible [47]. The Binary Input AWGN (BIAWGN) channel which is introduced in section 2.2.1, for example, experiences a maximum performance degradation of 2 dB, if the reliability of the incoming data is ignored [43]. The two most wide-spread algorithms are the Viterbi algorithm [129] in its soft-decision version developed by Hagenauer and Hoeher [50] and the Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm [5]. Both algorithms are optimal although the optimality criterion differs between them. The Viterbi algorithm is a maximum likelihood decoding algorithm that minimises the probability for *codeword* errors whereas the BCJR algorithm is a maximum a posteriori probability decoding algorithm that minimises the error probability of an *information bit*. The Viterbi algorithm is simpler to implement since it only requires a forward recursion to operate. However, Turbo Codes require an iterative decoding scheme (Figure 2.19) in which the two constituent decoders exchange information. Due to the interleaver present in the decoding system, neither decoder can be certain about how the *its* decoding of each individual bit influences the decoding process of the *other* decoder. Since the BCJR algorithm minimises the error probability of *each* information bit as opposed to the probability of codeword errors as it is the case for the Viterbi algorithm, the BCJR algorithm is typically used in Turbo decoding (see also [75]). Due to this suitability for Turbo Codes, the BCJR algorithm is used throughout this thesis in order to compare the results of RSC and Turbo Codes, obtained in chapters 3 and 4, respectively.

Only an overview of the BCJR algorithm is presented here. Section 2.3.2 presents a version that uses LLR values, section 2.2.2 presents its use as a Maximum A-Posteriori (MAP) equaliser, suitable for the equalisation of ISI channels such as the ones used in section 2.2.2 and of particular interest since a MAP equaliser can provide scaled L-Values to the decoder. Douillard *et al.*[30] showed, that the Turbo principle that is used successfully to improve decoding performance can also be used to increase the performance of equalisation by using the soft output of the equaliser and performing equalisation and decoding iteratively. This process, called Turbo equalisation, however, used with Turbo Codes, quickly leads complexity, since the decoder works iteratively itself. For this reason, Turbo equalisation is not used in this thesis.

The BCJR algorithm can be easily visualised by considering the state transitions of a finite state machine as a trellis (see section 2.1.3). The algorithm consists of three separate probabilities. The forward probability $\alpha_t(s)$ of a state in the trellis is the probability that this state will be reached given all inputs $w_1 \dots w_t$. Similarly, the backward probability $\beta_t(s)$ is the probability that this state will be reached starting from the end of the trellis with reversed edges and considering the (reversed) input $w_{s_w} \dots w_{t+1}$. The third probability is the probability of observing a particular transition $b : s \to s'$ in the trellis, $\gamma_t(s,s')$, which of course depends on the observed *bit*, $\hat{w}$. $\gamma_t(s,s')$ is typically context-dependent (its calculation is explained in section 2.2.2 for use in an equaliser and in section 2.3.2 for use in a channel decoder). $\alpha_0(s)$ and $\beta_{s_w}(s)$ need to be initialised, since they cannot be calculated. Typically, either all $\alpha_0(s)$ have the same value, for exactly one state $s^*$, $\alpha_0(s^*) = 1$ and for all other states $\alpha_0(s) = 0$. The same applies to $\beta_{s_w}(s)$ respectively. Knowing $\gamma_t(s,s')$ and the initialisations of $\alpha_0(s)$ and $\beta_{s_w}(s)$, the forward recursion of the BCJR algorithm is

$$\alpha_t(s) = \sum_{s'} \gamma_t\left(s',s\right) \cdot \alpha_{t-1}\left(s'\right) \tag{2.11}$$

and the backward recursion is

$$\beta_t(s) = \sum_{s'} \gamma_t\left(s,s'\right) \cdot \beta_{t+1}\left(s'\right). \tag{2.12}$$

The probability of a particular transition $b : s \to s'$ in the trellis at time index $t$, i.e. the output of the BCJR algorithm, while observing the *entire sequence* $\hat{\mathbf{w}}$ is then given by

$$p_t\left(s,s',\hat{\mathbf{w}}\right) = \alpha_{t-1}(s) \cdot \gamma_t\left(s,s'\right) \cdot \beta_t\left(s'\right). \tag{2.13}$$

### 2.1.5 Digital Modulation

In order to transmit digital information, an analogue carrier that can exhibit different states is required. The digital information then causes the analogue carrier to change into one particular configuration for the duration of transmission of one symbol. This process of the digital information triggering changes to the analogue carrier is called digital modulation. The change of state of the carrier can be detected at the receiver, which then is able to deduce the digital information by reversing the modulation process. The (finite) number of distinct states into which the analogue carrier can be modulated is called the modulation alphabet, $M$.

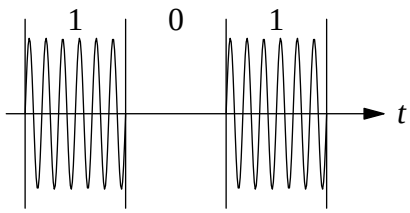



Figure 2.2: Example of a on-off keying modulated transmission.

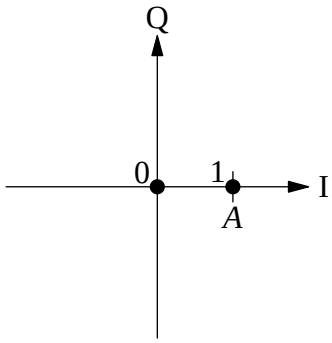



Figure 2.3: Constellation diagram of on-off keying.

Digital modulation can be categorised by the aspect of the analogue carrier that is changed. Consider an analogue carrier of the form

$$S(t) = A\cos(2\pi f \cdot t + \phi), \tag{2.14}$$

where $A$ is the amplitude of the signal, $f$ the carrier frequency and $\phi$ the phase of the carrier. Fundamentally, the carrier can be changed with respect to these three parameters, $A$, $f$ and $\phi$. In Amplitude Shift Keying (ASK), each state of the carrier corresponds to one particular amplitude $A_i$. A simple example is on-off keying, where a digital 1 corresponds to the amplitude $A_1$ and a digital 0 corresponds to an amplitude of 0. The transmission of the digital sequence modulated using on-off keying is shown in Figure 2.2. Since there are two states, amplitude $A$ and amplitude 0, the modulation alphabet, $M$ has cardinality 2 and can be summarised by a constellation diagram, showing the position of each state as a scatter plot of the complex plane (Figure 2.3).

Instead of adjusting the amplitude, $A$, the frequency, $f$, of the carries signal can be adjusted. The resulting modulation scheme is called Frequency Shift Keying (FSK). For example, a digital 1 can be assigned to some frequency $f_1$ and a digital 0 can be assigned to the frequency $f_0 = f_1/2$. The transmission of the digital sequence 101 is

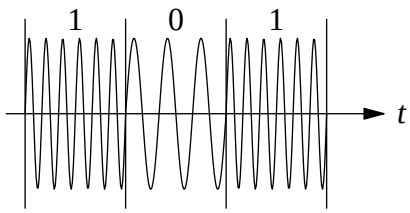



Figure 2.4: Example of a frequency shift keying modulated transmission.

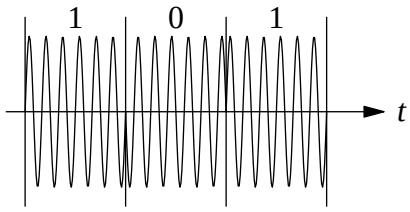



Figure 2.5: Example of a binary phase shift keying modulated transmission.

shown in Figure 2.4. A constellation diagram such as the one shown for on-off keying can obviously not be shown, since all different states agree in amplitude and phase.

Finally, the phase, $\phi$, of the analogue carrier can be modulated by the digital information giving rise to Phase Shift Keying (PSK). In this thesis the binary version of PSK, Binary Phase Shift Keying (BPSK) with a modulation alphabet, $M$, of cardinality two is used. Typically, antipodal states for the phase are chosen (i.e. a $180^\circ$ shift) for the two states. This can be represented by the constellation diagram shown in Figure 2.6, which closely resembles the on-off keying constellation diagram. This is not surprising, since a shift of $180^\circ$ effectively inverts the amplitude of the analogue carrier (see

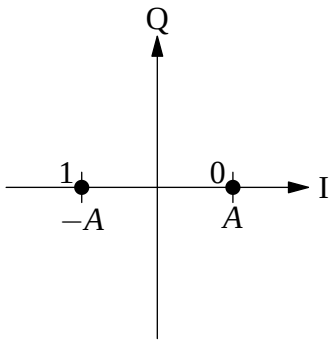



Figure 2.6: Constellation diagram for Binary Phase Shift Keying.

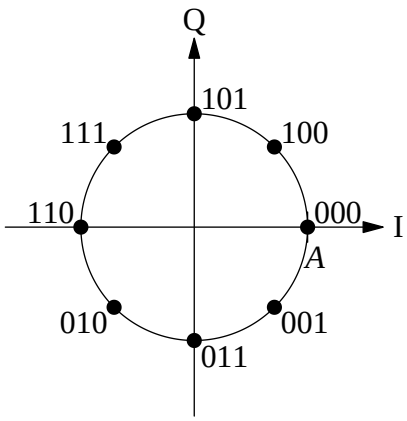



Figure 2.7: Constellation diagram for 8-PSK, using Gray coding.

Figure 2.5 for an illustrative example). However, the advantage of phase shift keying is that the cardinality of the modulation alphabet can be increased more easily than with amplitude shift keying. In the constellation diagram for 8-PSK (Figure 2.7), all symbols are still transmitted with the same amplitude, thus the energy that is transmitted remains constant (which is not the case for ASK modulation).

The modulation alphabet of 8-PSK modulation has a cardinality of 8. Thus, more than 1 digital bit can be transmitted per symbol of the modulation scheme. In fact, any modulation scheme can convey $n = \log_2(|M|)$ bits per symbol, where $|M|$ is the cardinality of the modulation alphabet, $M$. This yields an improved efficiency of the transmission, provided that the receiver can properly separate the different transmitted states. In fact, if it is desired to convey the information in many digital bits per transmitted symbol, it is possible to modulate using both the amplitude, $A$, and the phase, $\phi$, of the analogue carrier. These digital modulation schemes are called Amplitude Phase Keying (APK) or Quadrature Amplitude Modulation (QAM)[1] The well known constellation diagram for 16 APK (4,8,4), more commonly known as 16 QAM is shown in Figure 2.8.

[1] typically, the term quadrature amplitude modulation is used for the square constellation subset of amplitude phase keying[41].

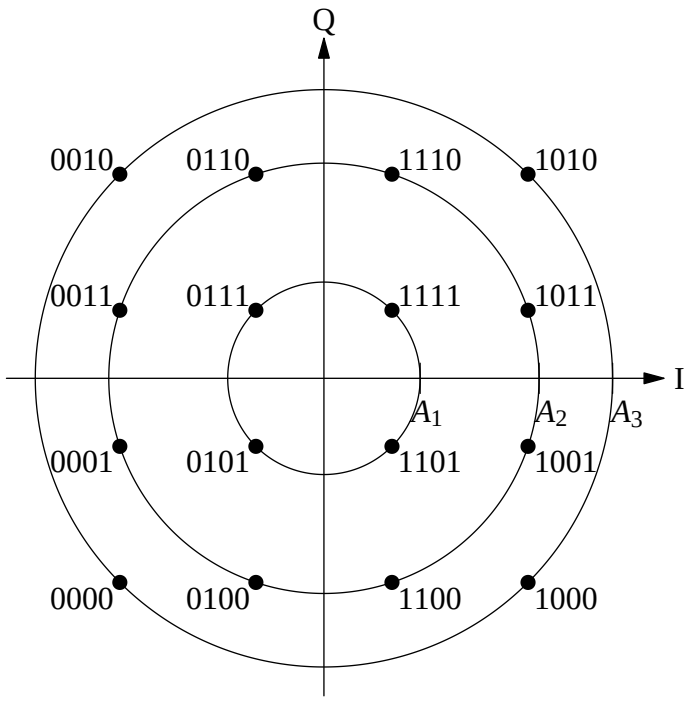



Figure 2.8: Constellation diagram for 16 APK (4,8,4) also known as 16 QAM. Gray coding is used for the assignment of input bits to transmission symbols.

### 2.1.6 Adaptive Modulation

In every communication system, the energy available for transmission of data is limited. Thus it is obvious, that increasing the cardinality of the modulation alphabet, $M$, decreases the distance that separates the individual transmission symbols. Hence, the probability that the receiver will detect symbols in error increases.[2] In other words, when requiring a fixed Bit Error Ratio (BER), the SNR of the transmission channel, i.e. its quality, needs to be better for 8-PSK than, say, BPSK. Glover and Grant[41] list the SNR required for a BER of $p_e = 10^{-6}$, reproduced here in Table 2.1.

Channels typically do not exhibit constant channel parameters. In order to increase the average throughput, it is possible to switch to more efficient modulation schemes when the channel exhibits good quality and use less efficient, but more error resilient modulation scheme when the channel quality decreases. The basic concept of an adaptive modulation system is shown in Figure 2.9. One example of such a system is burst-by-burst Adaptive Quadrature Modulation (AQAM)[52], which uses a selection

[2] Assuming, of course, that all possible symbols from $M$ are used.

| **Modulation Scheme** | $E_b/N_0$ **in dB** |
|---|---|
| BPSK (2-PSK) | 10.6 |
| QPSK (4-PSK) | 10.6 |
| 4-QAM | 10.6 |
| 8-PSK | 14.0 |
| 16-PSK | 18.3 |
| 16-QAM | 14.5 |
| 32-QAM | 17.4 |
| 64-QAM | 18.8 |

Table 2.1: Required signal-to-noise ratio for different modulation schemes for a bit-error-rate of $p_e = 10^{-6}$, presented by Glover and Grant[41].

of different cardinality QAM schemes.

The reason that adaptive modulation is not considered in this thesis for several reasons. Firstly, the adaptability of switching the modulation scheme is limited by the number of available modulation schemes. Increasing the number of modulation schemes clearly increases the complexity of the receiver. Furthermore, the receiver has to be notified of the modulation scheme either by explicit signalling or by blindly detecting the scheme (see Hanzo *et al.*[53] for an overview). For the case of explicit signalling, increasing the number of schemes increases the number of bits required for signalling. In the case of blind detection, increasing the number of modulation schemes can lead to higher uncertainty about the currently used scheme. Secondly, it is reasonable to argue that higher order modulation schemes are only beneficial, if their potential to transmit more than one bit per symbol can be realised. This benefit, however, is only relevant to channels with comparatively good quality, which are not considered in this thesis. Finally, the choice of modulation scheme affects the error probabilities of the symbols and thus the error probability of adjacent input bits is correlated. Thus the modulation scheme opens up a wide range of new parameters that can be adjusted and hence need to be investigated. A thorough investigation of all of them, however, is far beyond the scope of this thesis.

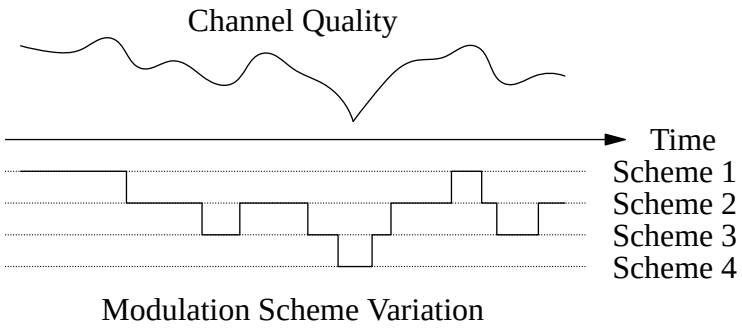



Figure 2.9: Illustration of the basic concept of a transmission system employing adaptive modulation with 4 modulation schemes. Scheme 1 is the scheme which allows the highest throughput but requires the best channel quality to work. The variation of the modulation scheme follows the instantaneous quality of the channel.

## 2.2 Transmission Channels

An important aspect of the simulation of a data transmission system such as the one in Figure 1.2 is the model of the transmission channel. The channel model for a CD recording where errors (scratches) appear in continuous blocks is significantly different from the channel model for satellite communications where white noise affects a weak transmission signal, leading to independent errors. This chapter presents the channel models that are used in the course of this thesis.

### 2.2.1 The Additive White Gaussian Noise (AWGN) Channel

One of the most commonly used channel models is the AWGN channel. This channel, depicted in Figure 2.10, makes two simple assumptions. Firstly, the input symbol $w$ is transmitted with a maximum average power $P$. Secondly, during transmission the channel adds to $w$ a statistically independent noise sample $z$ with zero mean and variance $\sigma_z^2$, distributed according to the Gaussian distribution

$$p_G(z) = \frac{1}{\sqrt{2\pi\sigma_z^2}} e^{-\frac{z^2}{2\sigma_z^2}} \tag{2.15}$$

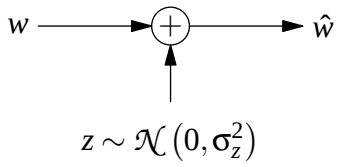



Figure 2.10: Discrete-time AWGN channel.

### Capacity of the AWGN Channel

The capacity of the AWGN channel in bits/channel-use, achieved by a Gaussian input distribution, is given by [25] as

$$C = \frac{1}{2}\log_2\left(1+\frac{P}{\sigma_z^2}\right). \tag{2.16}$$

It is often useful to use a single parameter, to describe the amount of noise on a channel. The SNR is typically used as this parameter. For a band-limited, continuous time AWGN channel with noise variance $\sigma_z^2 = \frac{N_0}{2T_s}$ ($T_s = \frac{1}{2B}$, where $B$ is the bandwidth of the channel) and the symbol energy $E_s = PT_s$, the signal to noise ratio is

$$\frac{E_s}{N_0} = \frac{PT_s}{2\sigma_z^2 T_s} = \frac{P}{2\sigma_z^2}. \tag{2.17}$$

Equation (2.17) applies to the discrete channel under the assumption of normalised transmission period $T_s = 1$. Using (2.17) in (2.16) yields the capacity as

$$C = \frac{1}{2}\log_2\left(1+2\frac{E_s}{N_0}\right). \tag{2.18}$$

### Capacity of the Binary Input AWGN (BIAWGN) Channel

The capacity of the AWGN channel can only be achieved by using a Gaussian input distribution. In practical communication systems, however, a fixed modulation symbol set is usually used. A common set is the BPSK modulation set with normalised transmission power, such that $w \in \{+1,-1\}$. The capacity of the BIAWGN channel is given by [43] as

$$C = \frac{1}{2}\sum_{w\in\{+1,-1\}}\int_{-\infty}^{\infty} p(\hat{w}|w)\log_2\left(\frac{2\cdot p(\hat{w}|w)}{p(\hat{w}|w=1)+p(\hat{w}|w=-1)}\right)d\hat{w} \tag{2.19}$$

with

$$p(\hat{w}|w) = \frac{1}{\sqrt{2\pi\sigma_z^2}}\, e^{-\frac{1}{2\sigma_z^2}(\hat{w}-w)^2}, \tag{2.20}$$

$$\sigma_z^2 = \frac{N_0}{2E_s}. \tag{2.21}$$

Note that the power term $P$ is missing from equation (2.21) because the transmit power is assumed to be normalised to 1 in equation (2.19). The variance of the noise is accordingly scaled with the inverse of the power constraint. The capacity of the BIAWGN channel is shown in Figure 2.13.

#### LLR Values for the BIAWGN Channel

A BIAWGN channel output can easily be expressed as an LLR value. For a BIAWGN channel with equiprobable[3] input alphabet $w \in \{+1,-1\}$ the LLR of the received signal $\hat{w}$ is

$$\begin{aligned}
\lambda(w|\hat{w}) &= \log_e\left(\frac{p(w=+1|\hat{w})}{p(w=-1|\hat{w})}\right) \\
&= \log_e\left(\frac{p(\hat{w}|w=+1)}{p(\hat{w}|w=-1)}\right) + \lambda(w) \\
&= \log_e\left(\frac{\sqrt{\frac{E_s}{\pi N_0}}e^{-(\hat{w}-1)^2\frac{E_s}{N_0}}}{\sqrt{\frac{E_s}{\pi N_0}}e^{-(\hat{w}+1)^2\frac{E_s}{N_0}}}\right) \\
&= -(\hat{w}-1)^2\frac{E_s}{N_0} - \left(-(\hat{w}+1)^2\frac{E_s}{N_0}\right) \\
&= 4\cdot\frac{E_s}{N_0}\cdot\hat{w}
\end{aligned} \tag{2.22}$$

### 2.2.2 Inter-Symbol Interference (ISI) Channels

A class of simple channels with memory is that of Inter-Symbol Interference (ISI) channels. The channel output $\hat{w}$ depends on the previous inputs to the channel and a random Gaussian noise sample. The channel output for time index $t$ is given by [96]:

$$\hat{w}_t = \sum_{i=0}^{m} a_i w_{t-i} + z, \tag{2.23}$$

where $w \in \{+1,-1\}$, the noise $z$ is assumed to be white and $a_0, a_1, \ldots, a_m$ are fixed, real parameters called the channel coefficients. A graphical representation of an ISI channel is shown in Figure 2.11.

[3] Because of this equiprobability, $\lambda(w) = 0$.

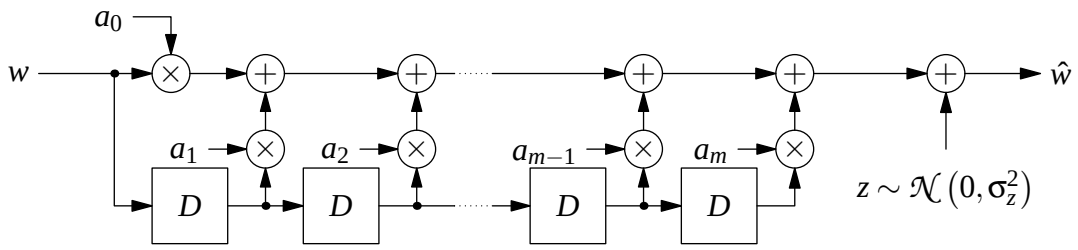



Figure 2.11: Graphical depiction of an ISI channel.

It is noteworthy to point out that the structure of an ISI channel is similar to that of the RSC encoder, presented in section 2.3.1, the only difference being the additions, which are not modulo 2. Analogous to RSC codes, there are $2^m$ states that the "encoder", i.e. the channel, can be in. Each such state corresponds to a sequence of past channel inputs $w_{-m}, w_{-(m-1)}, \ldots, w_{-1}$, and can be assigned a single "effect" value $e(s) = \sum_{i=1}^{m} a_i w_{-i}$. Due to this structure, the BCJR algorithm presented in section 2.1.4 can be used as an MAP equaliser for such a channel [125].

#### Capacity of Inter-Symbol Interference Channels

Arnold and Loeliger[3] propose an algorithm that can be used to calculate the capacity of ISI channels. Let $b : s \rightarrow s'$ be a transition in the trellis (see section 2.1.3) of the ISI channel from start state $s$ to state $s'$. Now define the operators $lst(b) = s$ and $rst(b) = s'$. This allows the definition of the branch metric $\mu(b)$ as

$$\mu(b) = p(b|lst(b))\, p(\hat{w}|b), \tag{2.24}$$

where, assuming uniform input and Additive White Gaussian Noise on the channel

$$p(b|lst(b)) = \frac{1}{2}. \tag{2.25}$$

Each of the $2^m$ possible states of the channel corresponds to a sequence of past channel inputs $w_{-m}, w_{-(m-1)}, \ldots, w_{-1}$, as previously described. Let

$$e(s) = \sum_{i=1}^{m} a_i w_{-i} \tag{2.26}$$

be the effect on the output bit that is given by the state $s$, with corresponding channel inputs $w_{-m}, w_{-(m-1)}, \ldots, w_{-1}$. Likewise, each branch $b$ corresponds to exactly one

value of the current input $w_{-0} \in \{+1,-1\}$. The expected channel output for a given branch, disregarding noise is then

$$e(b) = a_0 w_{-0} + e(lst(b)), \tag{2.27}$$

where $w_{-0}$ is the channel input value corresponding to branch $b$. Using equation (2.27), $p(\hat{w}|b)$ is then

$$p(\hat{w}|b) = \frac{1}{\sqrt{2\pi\sigma_z^2}} e^{\frac{-(\hat{w}-e(b))^2}{2\sigma_z^2}}. \tag{2.28}$$

The state metric $\mu(s)$ is then defined using equation (2.24) as

$$\mu(s) = \sum_{b:rst(b)=s} \lambda_t \mu(lst(b))\,\mu(b), \tag{2.29}$$

where $\lambda_t$ is a scaling constant such that for each time index $t$ the sum of the time-$t$ state metrics equals 1. For the ISI channel with white, Gaussian noise with variance $\sigma_z^2 = \frac{1}{2}\frac{E_s}{N_0}$, [25] gives the conditional entropy

$$h(\hat{\mathbf{w}}|\mathbf{w}) = \frac{1}{2}\log_2\left(2\pi e\sigma_z^2\right) \tag{2.30}$$

and according to [3]

$$\frac{1}{s_{\mathbf{w}}}\sum_{t=1}^{s_{\mathbf{w}}} \log_2(\lambda_t) = -\frac{1}{s_{\mathbf{w}}}\log_2(p(\hat{\mathbf{w}})) \to h(\hat{\mathbf{w}}). \tag{2.31}$$

The capacity of the ISI channel is then

$$C = h(\hat{\mathbf{w}}) - h(\hat{\mathbf{w}}|\mathbf{w}) = \frac{1}{s_{\mathbf{w}}}\sum_{t=1}^{s_{\mathbf{w}}} \log_2(\lambda_t) - \frac{1}{2}\log_2\left(2\pi e\sigma_z^2\right). \tag{2.32}$$

It is easiest to describe the algorithm by imagining a trellis of the channel for a number of time steps $s_{\mathbf{w}}$. For each time step $t$, the state metrics $\mu(s)$ of all states are calculated according to equation (2.29), disregarding $\lambda_t$. Then $\lambda_t$ is calculated such that the sum of the metrics equals 1 and the metrics are scaled appropriately. After $\lambda_t$ has been calculated this way for all $t$, equation (2.32) is used to calculate the capacity. Arnold and Loeliger investigated the channels listed in table 2.2 and report good convergence for blocksizes of $s_{\mathbf{w}} \geq 10^6$ [3]. The capacities for the channels listed in table 2.2 are shown in Figure 2.13.

| Channel Name | Normalised Impulse Response |
|---|---|
| DICODE | $a(D) = \left(1 - D^1\right)/\sqrt{2}$ |
| EPR4 | $a(D) = \left(1 + D^1 + D^2 + D^3\right)/2$ |
| E2PR4 | $a(D) = \left(1 + 2D^1 - 2D^3 - D^4\right)/\sqrt{10}$ |
| CH6 | $a(D) = 0.19 + 0.35D^1 + 0.46D^2 + 0.5D^3 + 0.46D^4 + 0.35D^5 + 0.19D^6$ |

Table 2.2: Impulse responses of selected channels investigated by Arnold and Loeliger[3].

### Equalisation and LLR Values for ISI Channels

The received sequence $\hat{\mathbf{w}}$ needs to be equalised before it can be processed further by the receiver. In this thesis the BCJR algorithm, described for this purpose in e.g. [125], is used as a MAP equaliser. While other possibilities, such as Turbo equalisation exist (see [98, 125]), the BCJR algorithm proved to be satisfactory in its performance with regard to its run-time complexity. The forward recursion of the BCJR algorithm is, in fact, equivalent to the calculation of the state metrics in equation (2.29). The backward recursion is similarly defined as

$$\overleftarrow{\mu}(s) = \sum_{b:lst(b)=s} \overleftarrow{\lambda_t}\, \overleftarrow{\mu}(rst(b))\, \mu(b), \tag{2.33}$$

where $\overleftarrow{\lambda_t}$ is the equivalent of the scaling constant, $\lambda_t$, for the backward recursion and, correspondingly, scales $\overleftarrow{\mu}(s)$. The output of BCJR algorithm is then

$$p(w|\hat{w}) = \sum_b \mu(lst(b)) \cdot p(\hat{w}|b) \cdot \overleftarrow{\mu}(rst(b)), \tag{2.34}$$

where the sum is over all branches which correspond to the input bit $w$ and $p(\hat{w}|b)$ is taken from equation (2.28).

This equalisation already provides probabilities. A scaling as in the case of the BI-AWGN channel is therefore not necessary, since the LLR values can be obtained directly from equation (2.34) as

$$\lambda(\hat{w}) = \log_2\left(\frac{p(w=+1|\hat{w})}{p(w=-1|\hat{w})}\right). \tag{2.35}$$

### 2.2.3 The Rayleigh Fading Channel

The Rayleigh distribution is commonly used in mobile radio channels to describe the time-varying nature of the received signal [99]. The Rayleigh probability density function (PDF) is given by [96] as

$$p_R(r) = \frac{r}{\sigma^2} e^{-\frac{r^2}{2\sigma^2}}, \quad r \geq 0. \tag{2.36}$$

This PDF is obeyed by the random variable $R$ if

$$R = \sqrt{X_1^2 + X_2^2}, \tag{2.37}$$

where $X_1$ and $X_2$ are zero-mean, independent Gaussian random variables with variance $\sigma^2$. The variance $\sigma^2$ is chosen to be $\sigma^2 = 0.5$ in this thesis in order for the channel not to add any power to the received signal. Thus

$$p_R(r) = 2re^{-r^2}, \quad r \geq 0. \tag{2.38}$$

with

$$R = \sqrt{X_1^2 + X_2^2}, \quad X_1 \sim \mathcal{N}(0, 0.5), X_2 \sim \mathcal{N}(0, 0.5) \tag{2.39}$$

The Rayleigh Fading Channel with AWGN is shown in Figure 2.12.

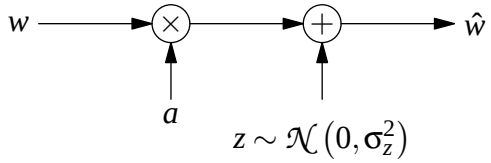



Figure 2.12: Discrete-time Rayleigh fading channel with AWGN. $a$ is Rayleigh distributed and generated by $a = \sqrt{b_1^2 + b_2^2}$, where $b_1$ and $b_2$ are drawn from independent, zero-mean Gaussian distributions with variance $\sigma^2 = 0.5$.

The factor $a$ is called the Rayleigh Fading Factor and is major importance when considering the *instantaneous* channel quality. Thus, $a$ determines the coherence time of the channel, i.e. the time for which the channel's response characteristics are essentially invariant. If the $a$ is kept constant over a significant portion of the "lifetime" of the channel, e.g. for the transmission of one data packet, the channel is said to be slow fading. If, on the other hand, $a$ varies for a block of or even for each individual symbol, the channel is said to be fast fading. In this thesis, unless mentioned otherwise,

a Rayleigh fading channel refers to a fast fading channel where $a$ changes for each input symbol $w$. Equalisation of the Rayleigh fading channel is possible only if the statistics of the channel are known (see e.g. [21]). For a fast fading channel this cannot necessarily be assumed.

### Capacity of the Rayleigh Channel

The capacity of a fast fading Rayleigh channel is given by [124] as

$$C = \mathbb{E}_a \left\{ \log_2 \left( 1 + a^2 \frac{E_s}{N_0} \right) \right\}, \tag{2.40}$$

where the input to the channel, $w$ is assumed to be Gaussian distributed. For BPSK signalling, where $w \in \{+1, -1\}$, the capacity can be calculated like that of the BIAWGN channel, taking into account the Rayleigh fading coefficients, such that

$$C = \frac{1}{2} \sum_{w \in \{+1,-1\}} \mathbb{E}_a \left\{ \int_{-\infty}^{\infty} p(\hat{w}|w) \log_2 \frac{2 \cdot p(\hat{w}|w)}{p(\hat{w}|w=1) + p(\hat{w}|w=-1)} d\hat{w} \right\} \tag{2.41}$$

with

$$p(\hat{w}|w) = \frac{1}{\sqrt{2\pi\sigma_z^2}} e^{-\frac{1}{2\sigma_z^2}(\hat{w}-aw)^2}, \tag{2.42}$$

$$\sigma_z^2 = \frac{N_0}{2E_s}. \tag{2.43}$$

In practice it is more convenient to exploit the ergodicity of the channel. The channel is simulated for $s_{\mathbf{w}}$ equally likely input bits $w_t \in \{+1, -1\}$, with channel realisations $a_t$ and $z_t$, hence $\hat{w}_t = a_t w_t + z_t$. The capacity can then be calculated as

$$C = \frac{1}{s_{\mathbf{w}}} \sum_{t=1}^{s_{\mathbf{w}}} \log_2 \frac{2 \cdot p(\hat{w}_t|w_t)}{p(\hat{w}_t|w_t=1) + p(\hat{w}_t|w_t=-1)}, \tag{2.44}$$

with $p(\hat{w}_t|w_t)$ calculated according to equation (2.42). The capacity for a Rayleigh fast-fading channel with a coherence length of 1 bit (a new Rayleigh distributed fading factor $a$ for each channel input) is shown in Figure 2.13.

### LLR Values for the Rayleigh Channel

The derivation of the LLR scaling constant for a Rayleigh channel is similar to that of the BIAWGN channel, with the difference that the channel input $w \in \{+1, -1\}$ is

scaled with the Rayleigh fading factor $a$ before being subjected to channel noise. Thus the LLR of the received signal $\hat{w}$ of a Rayleigh fading channel with equiprobable input $w \in \{+1,-1\}$ is

$$\begin{aligned}
\lambda(w|\hat{w}) &= \log_e\left(\frac{p(w=+1|\hat{w})}{p(w=-1|\hat{w})}\right) \\
&= \log_e\left(\frac{p(\hat{w}|w=+1)}{p(\hat{w}|w=-1)}\right)+\lambda(w) \\
&= \log_e\left(\frac{\sqrt{\frac{E_s}{\pi N_0}}e^{-(\hat{w}-a)^2\frac{E_s}{N_0}}}{\sqrt{\frac{E_s}{\pi N_0}}e^{-(\hat{w}+a)^2\frac{E_s}{N_0}}}\right) \\
&= -(\hat{w}-a)^2\frac{E_s}{N_0}-\left(-(\hat{w}+a)^2\frac{E_s}{N_0}\right) \\
&= 4\cdot a\cdot\frac{E_s}{N_0}\cdot\hat{w} \qquad (2.45)
\end{aligned}$$

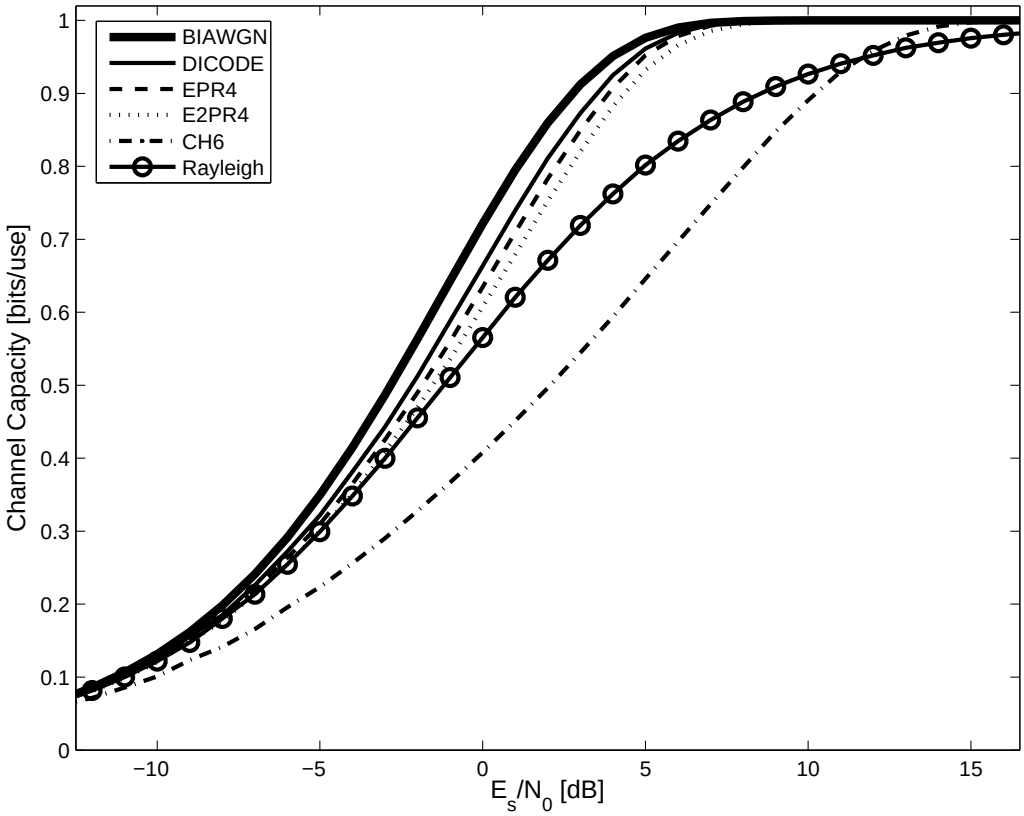



Figure 2.13: Capacity of the BIAWGN, ISI and Rayleigh channels. The coefficients of the ISI channels can be found in table 2.2.

## 2.3 Forward Error Correction

Shannon states, in his seminal paper [116], that any channel can be characterised by its channel capacity $C$ and that as long as the rate of information $r$ (in bits/second) satisfies $r < C$, the error rate can be reduced to any desired level by using proper channel coding. *Reliable* transmission is thus *possible*. Unfortunately, Shannon's proof is not constructive and so no statement can be drawn from it as to how such a transmission can be achieved in practice. This non-constructiveness ultimately gave rise to the research discipline of coding theory, which investigates methods to achieve reliable transmission through the use of channel codes. The objective of coding theory is to come as close as possible to the capacity predicted by Shannon.

The benefit to transmit as close as possible to channel capacity, $C$, is one of efficiency. Each improvement towards the channel capacity results in either less energy expended per information bit or more information transmitted per time. The achievements of coding theory are impressively visualised by considering the coding advances of the NASA space program which needed to transmit reliably over vast distances [24] and the recent advances made by Turbo and LDPC codes, coming within tenths of decibels of the capacity limit(see Figure 2.14).

In order to exploit channel codes, the transmitter needs to be extended to perform the encoding of the input data. Likewise the receiver needs to be able to decode the received data in order to benefit from the error correcting properties of the used channel code. Figure 2.15 shows a communication system that uses channel coding.

### 2.3.1 Recursive Systematic Convolutional Codes

Convolutional codes were first introduced by Elias [32] in 1955 as an alternative to block codes. Contrary to block codes, which are very well suited to correct burst errors, convolutional codes deal very well with independent errors such as those caused by white Gaussian noise [60]. The basic principle of convolutional codes is simple. The output of the code is generated by adding up (modulo 2) some digits of a temporary, binary word that is stored by the encoder. Which digits are added up is defined by the *generator polynomials* of the encoder, which can be seen as a bitmask over the temporary memory word. A binary input sequence $\mathbf{x}$ is then shifted bit-wise into the

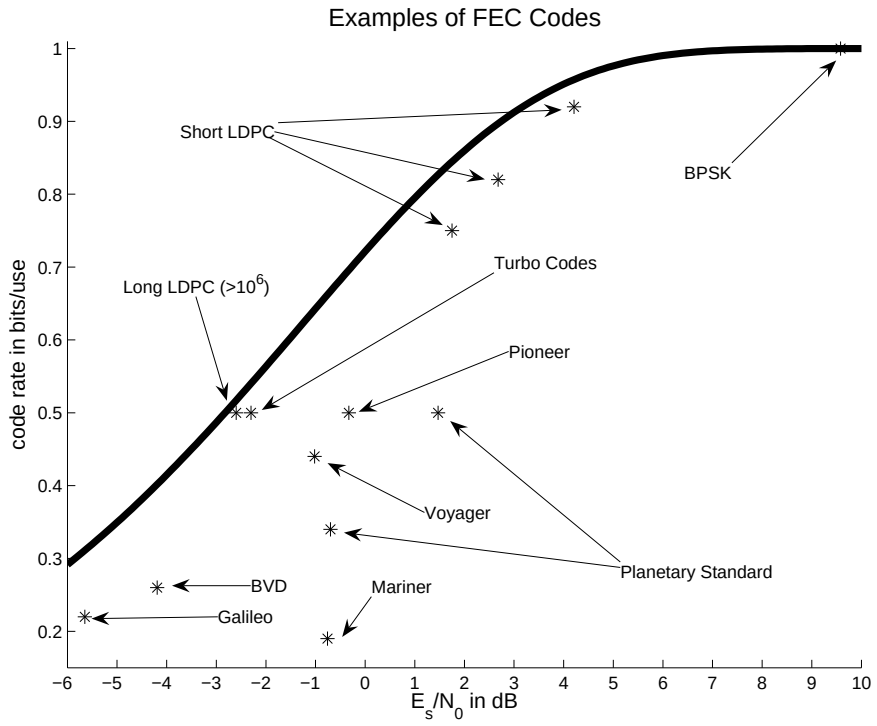



Figure 2.14: Performance of the error correcting codes used in the NASA space program against the capacity of a BIAWGN channel (bold). Adapted to $E_s/N_0$, originally published by Costello *et al.*[24], Turbo Codes and LDPC codes added from data presented by Lin and Costello[75] and Kou, Lin and Fossorier[67].

encoder memory and the output calculated for each bit, resulting in as many *parity bits* as the encoder has generator polynomials. A special case of convolutional encoders are Recursive Systematic Convolutional encoders, where one of the generator polynomials, typically $g^{(0)}$, does not produce any output but instead modifies the input data before it gets shifted into the encoder memory. To compensate for the missing code bit that would have been produced by $g^{(0)}$, the input bit taken directly as the first output of the encoder, $u^{(0)}$, the so-called *systematic* bit. Figure 2.16 shows a generalised RSC encoder with 2 generator polynomials, thus leading to a rate of $r = 1/2$ code, i.e. two bits for each input bit. An RSC encoder with $g^{(0)} = 1+D+D^2$, $g^{(1)} = 1+D^2$ with the corresponding state transitions and trellis is shown in Figure 2.17. $D$ indicates a delay element, i.e. the term $D$ indicates the last input bit, $D^2$ the bit before that.

Contrary to ARQ, introduced in section 2.4, no feedback channel is required in order to implement Forward Error Correction. However, both the transmitter and the receiver need to be able to handle the same channel code, typically resulting in a predefined,

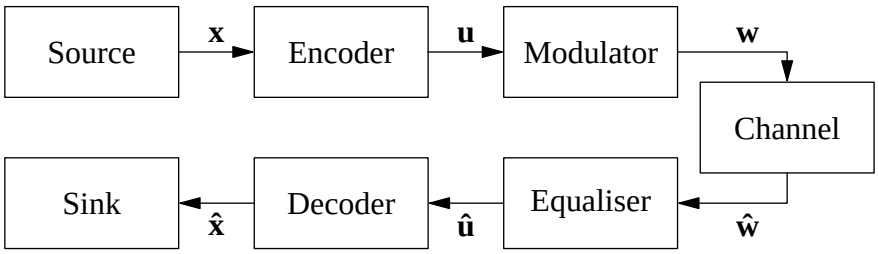



Figure 2.15: Communication system that uses channel coding.

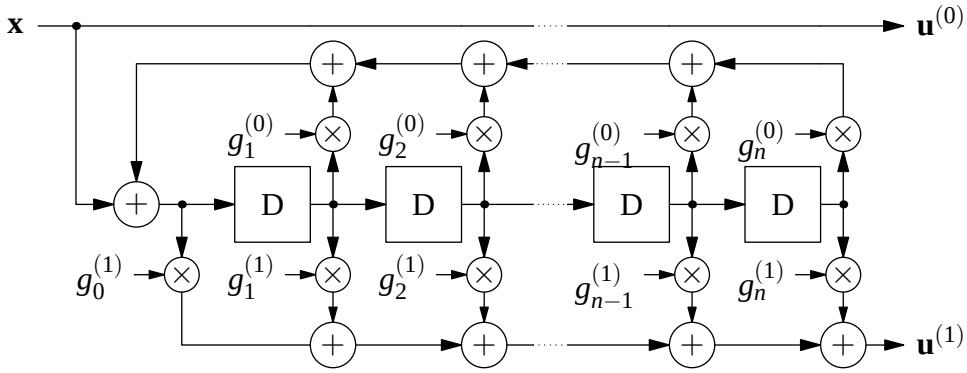



Figure 2.16: Generalised rate $1/2$ Recursive Systematic Convolutional encoder with generator polynomials $g^{(0)}(D) = 1 + g_1^{(0)}D^1 + g_2^{(0)}D^2 + \cdots + g_{n-1}^{(0)}D^{n-1} + g_{n-1}^{(0)}D^{n-1}$ (the feedback polynomial) and $g^{(1)}(D) = g_0^{(1)} + g_1^{(1)}D^1 + g_2^{(1)}D^2 + \cdots + g_{n-1}^{(1)}D^{n-1} + g_n^{(1)}D^n$, where $D$ indicates a Delay element, typically a memory cell to store the current value and make it available at the next cycle. One input bit $x$ is encoded to 2 code bits $u^{(0)} = x$ (the systematic bit) and $u^{(0)}$ (the parity bit). The order or constraint length $n$ of the encoder is determined by the amount of memory, i.e. the number of delay elements. For the case of binary input, i.e. $x \in \{0,1\}$, the additions are modulo 2. Typically, but not necessarily, the initial value of all delay elements is set to $0$.

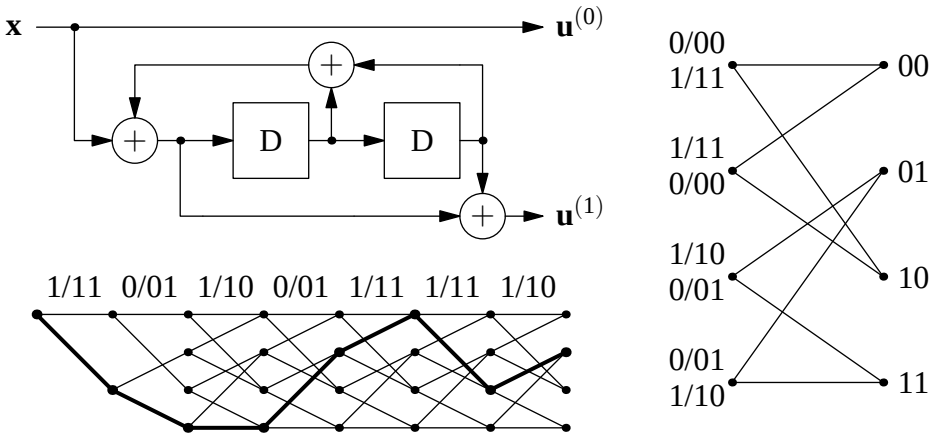



Figure 2.17: RSC encoder with $g^{(0)}(D) = 1+D+D^2$, $g^{(1)}(D) = 1+D^2$ (top-left). The associated state transitions are shown on the right (labelled $[x/u^{(0)}u^{(1)}]$, state labels are on the target (right) state). On the bottom left is shown a sample trellis path for the input sequence $\mathbf{x} = 1\,0\,1\,0\,1\,1\,1$, resulting in the encoded output $\mathbf{u} = 11\,01\,10\,01\,11\,11\,10$. Each transition is labelled with $[x_t/u_t^{(0)}u_t^{(1)}]$.

fixed encoding scheme and code rate. As long as the channel code is strong enough to deal with the error patterns that occur on the channel, this is sufficient and the lack of a feedback channel isn't a problem. In fact transmission using FEC can have advantages with regard to latency, since packets are not, and potentially cannot be, retransmitted. The lack of a feedback channel becomes problematic when considering system reliability. Every channel code has a residual bit error rate, i.e. certain error patterns that it cannot correct. Once such an error pattern occurs, the transmission system is unable to recover the transmitted data correctly. In addition, convolutional codes, typically, do not offer error detection capabilities. Thus, the receiver can only assume that the decoded data is error-free. Ideally, the channel code is sufficiently strong to correct every error that occurs. If, however, only few errors occur, i.e. the channel code is too strong, a throughput penalty is caused by the transmission of the parity bits generated by the encoder.

### 2.3.2 The LLR-BCJR (Log-MAP) Algorithm for RSC codes

The BCJR algorithm, introduced in [5] (see section 2.1.4) works with probabilities. Practical implementations of the BCJR algorithm, however, face the problem of limited accuracy of the data types and the very low probabilities that occur in the BCJR algorithm can become problematic in terms of numerical stability of the algorithm [51]. Robertson, Hoeher and Villebrun [107] presented a version of the BCJR algorithm in the log domain (the so called Log-MAP algorithm). This section presents this algorithm for the case of decoding a rate $r = 1/N$ RSC code.

Let $w_t^{(i)} \in \{+1, -1\}$ denote the BPSK symbol produced by encoder tap $i$ (see Figure 2.16) at time step $t$. Then, given the LLR value of the prior probability $\lambda_{\text{pri}}(w_t)$, the (logarithmic) probability of a state transition $s \to s'$ is

$$\gamma_t(s, s') = \frac{1}{2}\lambda_{\text{pri}}(w_t)\, q_{s\to s'}^{(0)} + \frac{1}{2}\sum_{i=0}^{N-1} \lambda\left(\hat{w}_t^{(i)}\right) q_{s\to s'}^{(i)}, \tag{2.46}$$

where $q_{s\to s'}^{(0)}, q_{s\to s'}^{(1)}, \ldots, q_{s\to s'}^{(N-1)}$ is the sequence of bits that the *encoder* produces for the state transition $s \to s'$, or, put differently, the output of the encoder given it is in state $s$ and receives the input $x_t$ that causes it to transition to state $s'$. $\hat{w}_t^{(i)}$ is the received LLR value after transmitting $w_t^{(i)}$ and $\lambda(\cdot)$ is calculated according to the channel model (see sections 2.2.1, 2.2.2 and 2.2.3).

The forward recursion of the algorithm (equation (2.11) in section 2.1.4) to determine the forward state probability can be calculated as

$$\alpha_t(s) = \sum_{s'}^{\boxplus} \gamma_t(s', s)\, \alpha_{t-1}(s'), \tag{2.47}$$

where the sum is over all states $s'$ such that a branch $b$ exists which links $s'$ to $s$, i.e. $b : s'_{t-1} \to s_t$. The sum $\sum^{\boxplus}$ is calculated according to equation (2.10). Likewise, the backward recursion of the algorithm (equation (2.12) in section 2.1.4) to determine the backward state probability can be calculated as

$$\beta_t(s) = \sum_{s'}^{\boxplus} \gamma_t(s, s') \cdot \beta_{t+1}(s'), \tag{2.48}$$

where the sum is over all states $s'$ such that a branch $b$ exists which links $s$ to $s'$, i.e. $b : s_t \to s'_{t+1}$. The decoder output at time index $t$ is then

$$\lambda(\hat{x}_t) = \sum_{\substack{b:s\to s' \\ s.t.\, q_0^{(0)}=+1}}^{\boxplus} \alpha_{t-1}(s) \cdot \gamma_t(s, s') \cdot \beta_t(s') - \sum_{\substack{b:s\to s' \\ s.t.\, q_0^{(0)}=-1}}^{\boxplus} \alpha_{t-1}(s) \cdot \gamma_t(s, s') \cdot \beta_t(s'), \tag{2.49}$$

where the sums correspond to all those transitions $b : s \rightarrow s'$ for which the encoder input has been 0 (modulated to $+1$) and 1 (modulated to $-1$), respectively. Note that the code is systematic so the input to the encoder $x = q^{(0)}$.

Iterative decoding of Turbo Codes (see Figure 2.19 in section 2.3.4) require passing extrinsic information between the constituent decoders. Given the output $\lambda(\hat{\mathbf{x}}_t)$ and the prior information $\lambda_{\text{pri}}(w_t)$, this extrinsic information is

$$\lambda_{\text{ext}}(\hat{x}_t) = \lambda(\hat{x}_t) - \lambda_{\text{pri}}(w_t) - \lambda\left(\hat{w}_t^{(0)}\right). \tag{2.50}$$

A practical implementation of the Log-MAP algorithm first computes $\gamma_t(s,s')$, according to equation (2.46) for all triples $(t,s,s')$ and then $\alpha_t(s)$ and $\beta_t(s)$, according to equations (2.47) and (2.48), respectively. The memory requirement of the simulation, as well as the runtime is thus bound by the calculation of $\gamma_t(s,s')$, i.e. $O\left(L \cdot s^2\right)$. In this thesis, a C-implementation of the Log-MAP algorithm with double precision is used. Since the codes investigated all have a binary trellis, not all $\gamma_t(s,s')$ had to be calculated, since not all pairs $(s,s')$ are connected. The complexity then reduces to $O(L \cdot s)$, since only 2 states, $s'$ need to be considered for each starting state, $s$.

It is worth noting that for efficiency reasons the algorithm can be simplified by dropping the corrective term $\log_e\left(1 + e^{-|\lambda_1 - \lambda_2|}\right)$ from equation (2.9). The resulting algorithm is called the Max-Log-MAP algorithm and is equivalent in performance to the soft output Viterbi algorithm, as shown by Fossorier*et al.*[36], "Theorem 3.1: The modified SOVA achieves the same decoding as the Max-Log-MAP decoding algorithm". Doing this will cause a performance loss of $\approx 0.5\,\text{dB}$ with regard to the Log-MAP algorithm[75].

### 2.3.3 Capacity-Approaching Codes

Today two classes of capacity-approaching codes are recognised. The first class is that of Turbo Codes introduced by Berrou, Glavieux, and Thitimajshima[12] in 1993, described in more detail in 1996[11] and presented in this thesis in section 2.3.4. The other class is that of low-density parity check (LDPC) codes described by Gallager [38, 39]. LDPC codes, although already described in 1962 where ignored until the 1990s (with the exception of Tanner's work [120]). They perform very close to the Shannon limit [105] and bear the promise of outperforming Turbo Codes, if con-

structed correctly [104]. Most notably, the error floor occurs at a much lower BER. The reasons that Turbo Codes are used in this thesis are that Turbo Codes have a simple encoder structure and can be decoded with relatively few iterations, thus greatly reducing the complexity required to obtain simulation results. Furthermore Turbo Codes are specified in several industry standards and thus implementations are readily available. Also, work by Sejdinovic *et al.*[114] indicates that the performance of punctured LDPC codes could be highly dependent on the used base code, further complicating the investigation of the effect of puncturing. Finally the structure of Turbo Codes allows some discussion of the effect of puncturing bits, whereas the best LDPC codes are thought to be irregular [104], making this kind of discussion difficult if not impossible. Another advantage of Turbo Codes is their systematic structure which allows the receiver to bypass decoding completely, if the channel is good enough, thus reducing the decoding complexity and delay in that situation.

### 2.3.4 Turbo Codes

Turbo codes, invented by Berrou *et al.* [12] consist, fundamentally, of two ideas. The first is the use of a code that has random-like properties. Shannon has shown in his work, that it is very desirable to have a code with random-like properties, yet decoding of these codes has been considered too complex [75]. The second idea, the Turbo principle, is the use of an iterative decoding process using soft-output values and passing extrinsic information between the decoders. It is this Turbo principle that allows efficient decoding (see [48] for an introduction). Figure 2.18 shows the encoder structure of a rate $r = 1/3$ Turbo Code. The input $\mathbf{x}$ is encoded twice with a constituent RSC encoder (the constituent encoders are typically identical, but they do not have to be [86]).[4] The second constituent encoder, however, does not encode the input $\mathbf{x}$ directly, but an interleaved version $\pi(\mathbf{x})$.[5] The interleaver is an integral part of the design of a Turbo Code, and some of its random-like properties of the code. Pseudo-random interleavers perform best for Turbo Codes [75].

According to Figure 2.18 the minimum rate of a Turbo Code is $r = 1/3$ since in addi-

[4]Other codes such as block codes can be used as well. For convolutional codes, recursive, i.e. feedback, encoders outperform non-recursive, i.e. feedforward, encoders [75].

[5]The presence of this interleaver made theoretical evaluation difficult until Benedetto and Montorsi[9] developed the uniform interleaver approach.

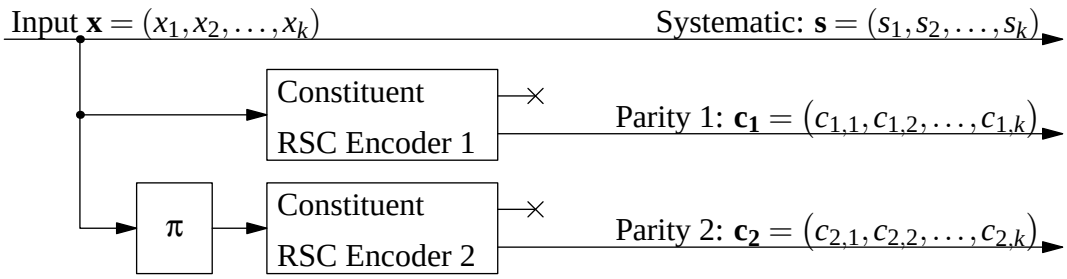



Figure 2.18: Schematic depiction of a rate 1/3 parallel Turbo encoder.

tion to the systematic bit, two encoders are required which each produce one parity bit $c_{i,t}$. In order to obtain lower rates, parity bits (but potentially also systematic bits [85]) are typically punctured (see section 2.6) to obtain the desired rate.

The decoding of Turbo Codes is where the Turbo principle and hence the name of the coding scheme comes from. The state complexity of the Turbo Code as a whole is extremely large due to the presence of the interleaver, making trellis-based decoding impossible [131]. However, the decoding of the codes produced by the constituent encoders is easily possible with the Log-MAP algorithm presented in section 2.3.2. Furthermore, since the constituent codes are systematic, the systematic sequence, **s**, can be shared between the two decoders. The input for the first decoder is then $\hat{\mathbf{w}}^{(0)} = \hat{\mathbf{s}}$ and $\hat{\mathbf{w}}^{(1)} = \hat{\mathbf{c_1}}$. For the second decoder the input is $\hat{\mathbf{w}}^{(0)} = \pi(\hat{\mathbf{s}})$ and $\hat{\mathbf{w}}^{(1)} = \hat{\mathbf{c_2}}$ ($\hat{\mathbf{s}}$ indicates that the sequence **s** has been distorted by the transmission channel). Clearly this separate decoding is sub-optimal. However, the constituent encoders inform each other about the information they have gained in the decoding process, the so called extrinsic information. A constituent decoder can use this extrinsic information of the *other* constituent decoder, interleaved properly, as a starting point, i.e. its prior information, in order to refine the decoding result. This iterative decoding method with soft-values often converges to within a few tenths of a decibel of overall MAP decoding [75] (for an in-depth review of Turbo coding with focus on iterative decoding, see [110]). The described Turbo decoder is shown in Figure 2.19. The output of the Turbo decoder is typically the output of the second constituent decoder but it is also possible to use the output of the first decoder or an average of both without affecting performance [75].

The iterative loop between the two constituent decoders is potentially endless. Thus, a stopping criterion has to be defined. Typically, this is a fixed number of iterations,

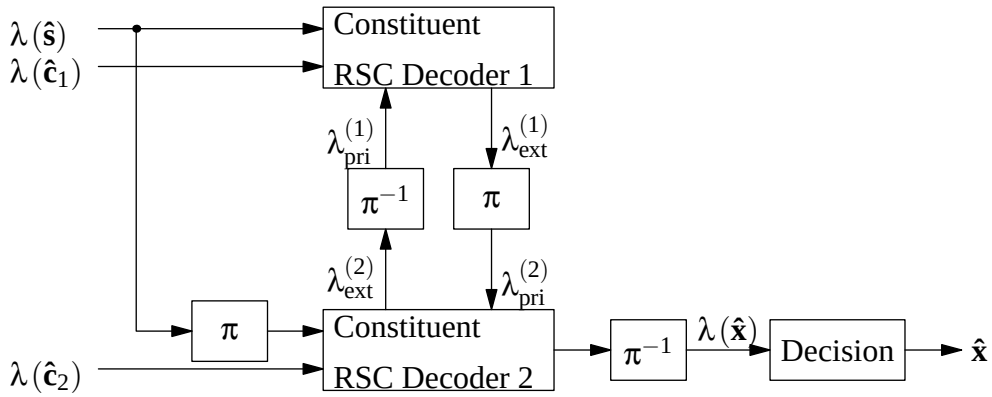



Figure 2.19: Schematic depiction of a decoder for a parallel rate 1/3 Turbo Code.

usually in the order of 10-20 iterations,[6] or some other reliability measure (e.g. by estimating the BER based on the LLR values of the decoder output, see e.g. [71, 72]).

The BCJR algorithm for MAP decoding typically assumes that the encoder returns to the 0 state after processing the input (this is typically ensured by placing appropriate tail bits at the end of the input). Due to the interleaver, this behaviour cannot be guaranteed for the second constituent decoder. Fortunately, this has little effect on performance for large block lengths [75].[7] The BCJR algorithm can be easily adapted to cope with this situation (see section 2.1.4). Finally it is worth making note of two drawbacks of Turbo Codes, although they are largely inconsequential for this thesis. Firstly, the blocklength of Turbo Codes is considerably larger than the blocklength of convolutional codes. Berrou *et al.* use 65536 input bits in their paper presenting Turbo Codes[12]. And blocksizes above $10^4$ bits are generally thought of as required[75].[8] This large blocklength and the iterative decoding results in a considerable coding delay.[9] Furthermore, Turbo Codes experience an error floor, where gains in SNR do not improve performance significantly.

[6] In this thesis, for reasons of complexity and thus simulation time, only 5 iterations are simulated, the results nevertheless are comparable to those found in the literature.

[7] It is also possible to modify the encoder to terminate both constituent decoders.

[8] This is in contrast to convolutional codes (but not in contrast to LDPC codes), where not the blocklength, but the constraint length of the encoder was the decisive variable to improve performance.

[9] The blocksize requirement of LDPC codes, the main contester with Turbo Codes, however, is even larger and $10^6$ bits are required in order to surpass the performance of Turbo Codes.

### 2.3.5 Notation of Convolutional Codes

Convolutional codes can be characterised mathematically by their generator polynomials, $g^{(0)}(D)\cdots g^{(n)}(D)$. A convenient notation for convolutional codes simply lists the generator polynomials, e.g. $\left(1+D+D^2,1+D^2\right)$. In order to avoid confusion, a recursive systematic convolutional code can be expressed as the fraction of parity-generating polynomials $g^{(1)}(D)\cdots g^{(n)}(D)$ and the feedback polynomial $g^{(0)}(D)$, e.g. $\left(1,\frac{1+D^2}{1+D+D^2}\right)$. The blocklength is added when describing simulation results in order to allow for better comparison. Turbo Codes typically use RSC codes, so the fractional expression is not used in order to provide for better readability. Thus the "original" Turbo Code by Berrou, Glavieux and Thitimajshima[12], in this notation, is expressed as the $\left(1+D+D^2+D^3+D^4,1+D^4,65536\right)$ Turbo Code.

#### Octal Representation of Generator Polynomials

Generator polynomials for convolutional codes are frequently represented as octal numbers in order to save space. The generator coefficients $g^{(0)}\ldots g^{(n)}$, when written as ones and zeros form a binary number, which is then converted into octal. Two different ways to arrange the coefficients are possible. Proakis [96] as well as Massey and Costello [84] arrange the polynomial with lowest power first and then read the binary sequence as an octal number. Thus the polynomial $g=1+D^2+D^4+D^5$ has a binary representation of $101011_2$ which is $53_8$. Expressed in octal notation, the Turbo Code introduced by Berrou *et al.* is $(37_8,21_8,65536)$. This method (the least significant bit on the left) is the generally used expression for generator polynomials of convolutional codes. However, when specifying generator polynomials of binary primitive BCH codes[75], Lin and Costello place the least significant bit, i.e. the lowest term, on the right. In order to avoid confusion, in this thesis, all generator polynomials are specified in their polynomial description and not represented in octal notation.

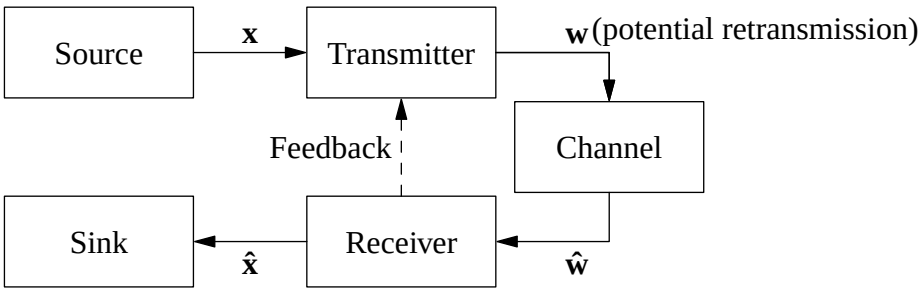



Figure 2.20: Communication system with automatic repeat requests (ARQ).

## 2.4 Automatic Repeat Request (ARQ)

The aim of any transmission system is to achieve reliable transmission over a channel. This implies that there has to be some method to detect, and if possible correct, errors that occur during transmission. Error detection can come in many form and shapes, but one commonly used method is the use of CRC sums (see e.g. [42, 75] for details of implementing cyclic codes). In fact, CRC sums are often part of industry standards (e.g. IEEE802.3 [121], IEEE802.11 [73], ISO-3309 [57] or ITU-T-v42 [58]).[10] Once errors are detected and a feedback channel exists[11], the receiver can inform the transmitter that the received sequence $\hat{\mathbf{w}}$ contains errors. The transmitter can then retransmit $\mathbf{w}$, hoping that, this time, the transmission is successful. Once the transmission is successful the receiver can, but does not need to (see e.g. Jeon and Jeong[59]), inform the receiver. These two messages are typically called ACKnowledgement (ACK) for successful transmission and Negative AcKnowledgement (NAK) to indicate transmission error. The protocol as a whole is called ARQ.

Several distinct ARQ protocol types exist. The simplest one, shown in Figure 2.21, is stop-and-wait ARQ. The transmitter waits until the receiver acknowledges correct reception of the transmitted data and proceeds to transmit new data or retransmits in case of a NAK. Other options are go-back-n ARQ where the transmitter does not wait for an ACK but continues to transmit data. Once an error occurs, the receiver *restarts* transmission with the packet that triggered a NAK (packets that are in transit but are

[10] A CRC sum is of course not the only available method. More creative ideas are e.g. the use of special markers combined with the error-*intolerance* of arithmetic coding and then using the garbled, decoded marker to detect the presence of errors[33].

[11] Note that the availability of a feedback channel does not increase the capacity of that channel, as shown by Alajaji[1].

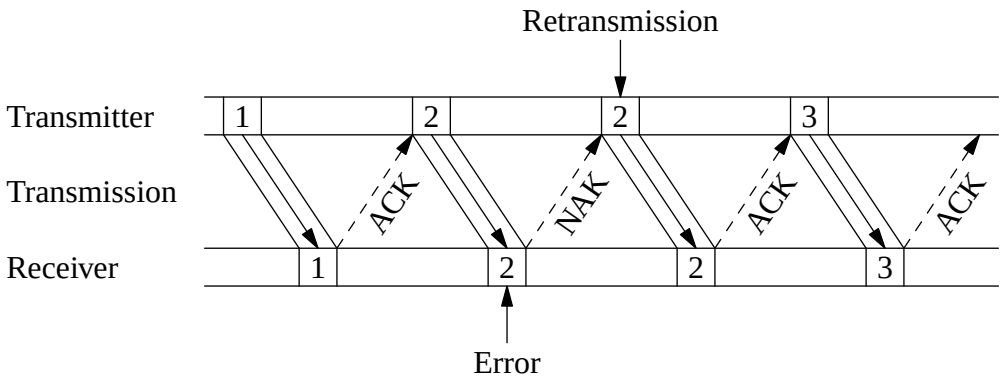



Figure 2.21: Visualisation of the stop-and-wait ARQ protocol. The transmitter waits until either receiving a positive or negative acknowledgement of reception of the message (ACK or NAK).

out of order are discarded at the receiver). Finally in selective repeat ARQ only those packets are retransmitted that trigger a NAK. The receiver is responsible for buffering packets and passes them on to the data consumer in the correct order.

The three ARQ schemes do not differ in their system reliability. The main distinctive feature when comparing ARQ systems is the throughput efficiency [14], defined as the ratio of the number of all correctly received and accepted bits over the number of bits that could have been transmitted in the same time period. Selective repeat outperforms go-back-n and stop-and-wait, but requires a potentially infinite buffer at both the receiver and transmitter [78]. Mixed mode ARQ has been proposed [88] for the case where the buffer size is too small and a buffer overflow would occur otherwise.

The different ARQ schemes have arisen in an attempt to improve the throughput performance of ARQ [76]. While the overhead of an ARQ system is fairly low [75], once the error rate of the channel increases, the throughput of any ARQ system drops rapidly. The main problem, that all error-detection-only systems face, is that there is no guarantee that a retransmission is error free (although code combining could be used, see [16]). If the errors appear in bursts, this is not a problem, however if errors are spread such that a few errors occur in nearly every packet, an ARQ system faces severe throughput degradation. The same effect occurs for the length of the transmitted data, as ARQ system face throughput degradation if the block size gets too large [90].

Despite their shortcomings for high channel error probabilities, ARQ systems have the advantage that they can provide system *reliability*. Thus, the receiver either supplies correct data or no data at all (as opposed to a FEC system, introduced in section 2.3). To achieve this reliability, ARQ systems require a feedback channel. The effect of errors on the feedback channel, however, while investigated in the literature (see [10, 92, 15]), is not considered erratumAddClarificationin this thesis. Instead it is assumed in this thesis that the feedback channel has a low throughput requirement such that any data can be protected by a sufficiently strong error correcting code (see section 2.3), effectively making the feedback transmission reliable.

## 2.5 Hybrid ARQ

Error control consist of two fundamental techniques. Error detection with retransmissions (ARQ), discussed in section 2.4, and error correction through the use of channel codes (FEC) as explained in section 2.3. The major advantage of ARQ over FEC is that of lower complexity of the implementation and a low overhead, while achieving high system reliability. However, ARQ schemes have in common that the throughput deteriorates rapidly once the channel error rate increases[76]. On the other hand, FEC systems enjoy very good throughput for channels with a fairly constant level of noise but suffer from the possibility of uncorrectable error patterns. In general, the probability of an uncorrectable error is much greater than the that of an undetectable error, making coding difficult and expensive to implement [75]. The reliability of the transmission system therefore cannot be guaranteed, since the output of the decoding process has to be passed to the application regardless of its correctness, since retransmissions are not built into such a transmission system. Burton and Sullivan[14] conclude:

> While we have seen that ARQ systems currently in use may not be satisfactory in future applications, the solution lies not in FEC systems, but rather in a better ARQ technique.

A communication system designed to properly combine the two systems can overcome the drawbacks of ARQ-only or FEC-only systems. These forms of transmission systems are called Hybrid ARQ (HARQ) transmission systems [13] and have been shown to be superior to pure ARQ or FEC systems [108].

The fundamental design principle of a HARQ system can be described by a single question: “Why does the receiver need to throw away the received packet, even if it is not error-free?” Typically, only a few errors occur during transmission, hence most of the bits are error-free. The simplest form of a HARQ system does not even need to use channel coding at all. Simply combining the received packets might be sufficient.[12] Sindhu[118] proposes an algorithm for determining the transmitted message from two or more corrupted copies when the channel over which data are transmitted is affected by dependent errors. Chakraborty *et al.* use an XOR combination of the two received packets to detect the bits that are different between the retransmission and the original and then use a brute-force approach of checking all possible combinations of these bits until the frame check sequence (a CRC sum) passes [18, 19]. In 1985, Chase developed a method of combining an arbitrary number of packets [20], that has since then been used (see section 2.1.2 or e.g. [27] which uses trellis-coded modulation with code combining). If more than one retransmission is available, packet combining can become complex, since it is theoretically possible that an arbitrary combination of the received packets will lead to correct decoding. Kallel and Leung investigated this problem for a multiple-copy transmission system[65, 66] and concluded that somewhat of a gain can be achieved by evaluating all possible combinations but that this was not worth the prohibitive complexity of this scheme.[13]

Hybrid ARQ systems, which employ some form of combining the information of the received packets, can be considered as a form of transmission system that exploits the time diversity of a channel (in particular this is true for fading channels, of course). This is achieved by exploiting to channel realizations with the expectation, that errors do not “overlap” in the transmitted packets. Zhao and Valenti[142] extended this concept of diversity to space diversity by generalising the Hybrid-ARQ design to incorporate the presence of multiple listening nodes which are capable of acting as a relay and of listening to the communication between the source and destination nodes. Such a setup is frequently encountered in wireless transmission systems. Assuming that the SNR is correlated with the distance from a transmitter to a receiver, it is obvious that under this consideration the nodes closer to the transmitter are able to decode the transmitted data before the receiver can do so and thus can act as a relay and im-

[12] An approach that is also worth mentioning here is the one taken by Lee and Lin[74], which does not adjust the coding scheme but instead changes the modulation from QPSK for the first packet to BPSK for the second packet.

[13] There is also work by Uhlemann *et al.* on a concept of Turbo combining [126] that provides benefits of better combining at the expense of additional complexity.

prove performance. In this thesis, however, only a direct-link transmission system is considered in order to obtain results independent of routing and relaying strategies.

### 2.5.1 Type-I Hybrid ARQ

The straightforward implementation of a Hybrid ARQ system, now called type-I HARQ, is simply to send an FEC protected data sequence to the receiver and use ARQ to guarantee system reliability should uncorrectable errors occur. This system, shown in Figure 2.22, is well suited for communication systems with fairly constant levels of noise and outperforms an ideal selective-repeat ARQ system [76]. The first codes to be employed in such a system were the well understood block codes and it has been shown by Sastry[112] that, depending on the environment, there is an optimum error correction capacity of the used code. The use of block codes in HARQ has been the topic of extensive research. Sastry investigated generalised burst trapping codes in 1976 [113]. Krishna and Mogera[89, 68] use a new linear code (which they call "KM" code). Wicker investigated Reed-Solomon codes for mobile-phone environments and the Rayleigh channel in particular [135, 134] and later, together with Bartz, investigated Reed-Muller [136] and MDS codes [137] (Sakakibara also investigated MDS codes with particular emphasis on multicast transmission [111]). Rice[101] analysed generalised minimum distance decoding for BCH codes and Rasmussen[100] dealt with trellis codes. From this treatment of codes, several concatenated, or cascaded, schemes were investigated by e.g. Takata [119] and Deng [28]. In 1993 Wang and Lin presented a cascaded system that separated the error detection and error correction concerns in 2 separate codes [133]. In 1994 de Alfaro and Meo [26] used this to develop an encoding scheme in which error correction can be achieved by *any two* received packets, making this scheme better suited to deal with packet losses.

An alternative to block codes are convolutional codes. In 1971 Fang showed that the reliability function of a transmission channel with a variable-length convolutional code can be bounded by the channel capacity [34] and provided a repeat-request scheme for it. Yamamoto and Ito[140] demonstrated an improved scheme in 1980 which yields a significant reduction of the number of retransmissions compared to Fang's approach. Reportedly, Yamamoto and Ito's scheme achieve twice the reliability of an FEC system using the Viterbi algorithm. In order to obtain this reliability, a likelihood ratio test is

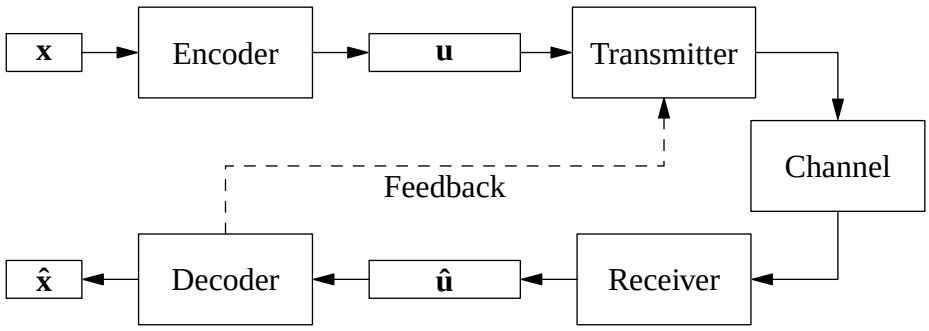



Figure 2.22: Progression of a packet through a type-I hybrid ARQ system. First, the packet is encoded (thus increasing the number of bits required to transmit it) and then sent over the channel, potentially multiple times due to retransmissions. Clearly, an overhead is caused by the parity bits of the for initially successful packets.

used to determine retransmissions. It took nearly 15 years until in 1994 Hashimoto[55] was able to show that this test, which before was generally considered suboptimal, is "effectively optimal".

Convolutional codes do not have a straightforward error detection mechanism. However, reliability measurements can be used within the Viterbi algorithm to provide a retransmission criterion (see e.g. Kudryashow [70]). As an alternative, properties of the decoding process can be used, such as in Drukarev's work [31] which extends Kallel and Haccoun's work on time-out conditions [63]. Another interesting approach comes from Hashimoto [54].[14] It is well known that reduced complexity decoding of convolutional codes typically causes bursts of errors of considerable length. By placing markers in the input sequence and looking for the presence of these markers, Hashimoto was able to achieve a small probability of undetected errors. Wang and Lin[133] separated error correction and error detection by using two separate codes and thus made it easier to us HARQ with convolutional codes. Turbo codes, of course, can also used in type-I HARQ schemes. Narayanan and Stuber[91], for example, use the MAP-decoding soft-value output of the previous packet as prior information when decoding the retransmission.

Type-I Hybrid ARQ systems, while well suited for transmission over channels with constant noise [76], use a fixed rate channel code. The problem of potentially wasting transmission time if the channel code is not adjusted to the channel conditions still

[14] It is mentioned in [54] that Kudryashow has done similar work for block codes in [69].

occurs, albeit with somewhat less severity for the average case, since a weaker channel code can be used. If an uncorrectable error pattern occurs, the penalty of retransmitting error-free bits is still paid by type-I HARQ systems.

### 2.5.2 Type-II Hybrid ARQ

In order to overcome the limitations of a fixed code rate, Mandelbaum[82] investigated the use of punctured Reed-Solomon codes. Not the entire codeword[15] was transmitted, but a punctured version (see also section 2.6), leading to a transmission system with *incremental redundancy*.[16] This approach of holding back some of the encoded bits, now called type-II HARQ, is shown in Figure 2.23. It was used by Lin and Yu[77] to devise a transmission system of a half-rate invertible code that alternatively transmits the parity data. One half of the encoded data is sent first. If errors are detected, the omitted parity bits would be transmitted. These parity bits allow the receiver to perform error correction. If errors persist, the original bits are retransmitted and error correction performed again, continuing until the packet can be decoded without errors. This scheme was improved on by Wang and Lin one year later [133] by separating error detection and error correction, thus making the use of convolutional codes possible. Type-II HARQ schemes overcome a significant shortcoming of type-I systems, namely the reduced throughput for low noise transmissions. Given a *suitable* puncturing and low noise on the channel, a type-II HARQ system performs exactly like a "pure" ARQ system. Only when errors do occur are parity bits provided to the receiver. In fact, the first transmission of parity bits does *not* necessarily contain all remaining, available parity bits. Hagenauer in 1988 [46] developed a systematic puncturing method called Rate-Compatible Punctured Convolutional (RCPC) codes (see section 2.6.3) that allow for a more fine-grained control over the transmission of parity bits. Kallel[64] suggested an alternative construction method and Barbulescu and Pietrobon extended punctured convolutional codes to Turbo Codes[7], with work on the design of good rate-compatible codes by e.g. Babich, Montorsi and Vatta [4], Row-

[15] Metzner[87] worked on reducing the complexity of the decoding process by coding subblocks of the input word but since computing capacity even for mobile devices has improved dramatically in the past 30 years, this is largely inconsequential.

[16] Note that the concept of incremental redundancy does not only apply to channel coding alone. Hagenauer, Dutsch, Barros and Schaefer[49], proposed to extend the redundancy management through an ARQ like mechanism to joint source and channel coding, allowing the encoder to decrement redundancy for already redundant *input data*.

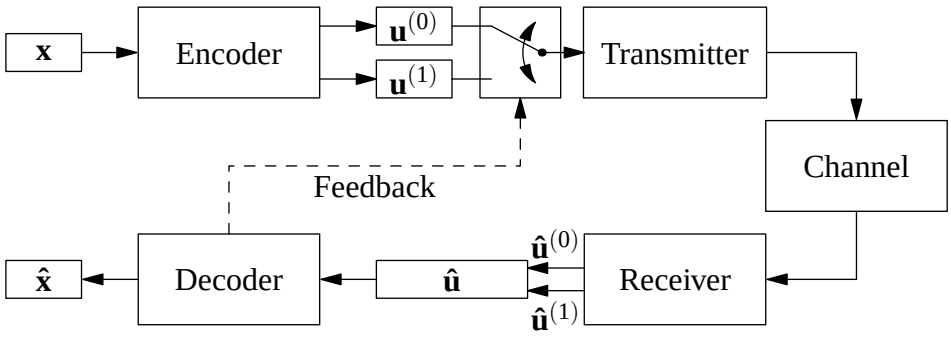



Figure 2.23: Progression of a packet through a type-II hybrid ARQ transmission system. The packet is first encoded but then split into several segments before it is sent over the channel (2 packets are shown, but any number is possible). Whenever a NAK is received the next packet to be transmitted changes from $\mathbf{u}^{(0)}$ to $\mathbf{u}^{(1)}$ (and back for the next NAK). The receiver assembles $\hat{\mathbf{u}}^{(0)}$ and $\hat{\mathbf{u}}^{(1)}$ to $\hat{\mathbf{u}}$ which is passed to the decoder. The latest received copy of $\hat{\mathbf{u}}^{(i)}$ is used. If $\hat{\mathbf{u}}^{(i)}$ has not yet been received, it is replaced with LLR values of 0. For RSC codes in the parity retransmission system, described by Lin[77], $\mathbf{u}^{(0)}$ contains the systematic bits and $\mathbf{u}^{(1)}$ contains the parity bits.

itch and Milstein[109] or Xie and Tang [139]. Full, granular control over the amount of redundancy can be achieved by defining a transmission order through the use of an interleaver [83] (shown in section 2.6.4). The problem of finding a suitable order is investigated in this thesis.

### 2.5.3 Channel Adaption Through Hybrid ARQ

Transmission channels are not necessarily stationary in their transmission characteristics. Nevertheless, hybrid ARQ schemes can be used successfully for transmissions over them (e.g. [79, 130]). The availability of a feedback channel in an ARQ transmission system allows the transmitter to obtain information about the quality of the channel. In the simplest form, this can be achieved by simply counting the number of NAK packets received by the transmitter [2, 141], but it can also be achieved by observing an external source like Pursley and Sandberg show[97]. The aim of these observations is to increase the error correcting capability of the *first* packet to be transmitted. Rice and Wicker[103, 102] adjust a Reed-Solomon code, Shiozaki increases the error correction of BCH codes [117] and Kallel uses rate-compatible codes [62].

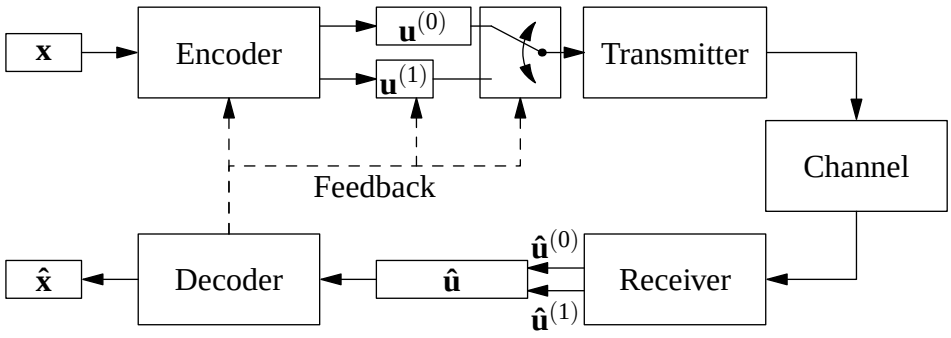



Figure 2.24: Progression of a packet through an adaptive type-II hybrid ARQ transmission system. The information received by the transmitter through feedback is not only used to switch to the next packet but also to adjust other parameters to the channel conditions, e.g. the encoding or the length of the next packet.

Ideally, the channel code that is used to protect the first packet is strong enough to correct most errors that occur on the channel. Figure 2.24 shows this graphically. In this aim, an adaptive system is similar to a type-I hybrid ARQ system. The additional benefit when using type-II hybrid ARQ systems is that the second packet can potentially be much smaller than the first packet. Chapter 5 deals with aspects of packet sizes and their effect on throughput and delay.

## 2.6 Puncturing, Code Ensembles and Variable Rate

Consider the communication system shown in Figure 2.25. The transmitter applies FEC in order to achieve reliable transmission, protecting $\mathbf{x}$ with parity bits to form $\mathbf{u}$ from which $\hat{\mathbf{x}}$ is recovered. The benefit of FEC stems from the fact that it is possible to recover the input data ($\hat{\mathbf{x}} = \mathbf{x}$), even when $\mathbf{u}$ is distorted by a transmission channel (see section 2.3). The main drawback of FEC is the requirement of sending $1/r$ bits per data bit. Typically $r = K/N$ requiring the transmission of $N$ code bits for every $K$ data bits. Assuming that a transmission system can send a constant number of *symbols* per unit of time, the selection of the rate $r$ determines the efficiency of a reliable transmission system in terms of how many *data bits* can be transmitted per unit of time. An efficient transmission system ideally transmits at a rate $\check{r}$ that is just sufficient for the decoder to obtain the output $\hat{\mathbf{x}} = \mathbf{x}$ from $\hat{\mathbf{u}}$. For $r < \check{r}$ parity bits are transmitted that are not necessary to recover the input data. By definition these parity bits do not contain any

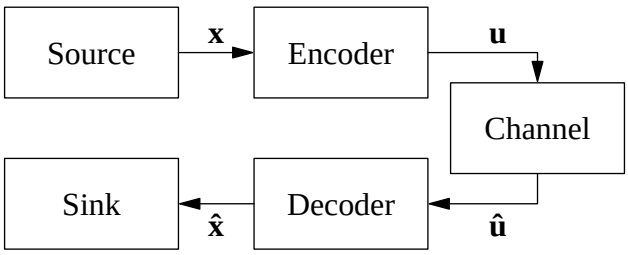



Figure 2.25: A communication system using Forward Error Correction to protect the input data **x** from distortions caused by the transmission channel.

information useful to the receiver (except for potentially increasing the confidence of the decoder). For $r > \check{r}$ the transmitted information is not adequately protected against errors caused by the transmission over the channel. Residual bit errors can potentially cause catastrophic failure at the data consumer (see Figure 1.1) or trigger a retransmission (in e.g. TCP). Retransmissions reduce the number of new information bits that can be transmitted, thus also reducing the efficiency of the overall transmission.

A prerequisite for a transmission system to achieve $\check{r}$ is the availability of a code of rate $\check{r}$. If such a code cannot be obtained, either because it does not exist or existing codes are too complex to implement, two strategies are feasible. Firstly, a higher rate code can be used repetitively. It is well known, however, that repetition codes are unable to provide any coding gain[22].[17] The other option available to come close to $\check{r}$ is to use a lower rate code and puncture a sufficient number of bits from the encoded sequence **u**. This section discusses strategies to obtain a code of arbitrary rate $\mathring{r}$ from a base code of rate $1/N$ within $[1/N, 1]$.

### 2.6.1 Puncturing

The easiest method of obtaining a code with $\mathring{r} < r_{\text{base}}$ is to omit some of the parity bits produced by the base code. The exact strategy of which bits to leave out needs to be known to both the encoder and the decoder, such that the decoder can take appropriate steps to deal with punctured bits. Typically, the puncturing pattern used is a regular one. A well known example of this is the Turbo Code used by Berrou [12], that

[17] Repetition codes are equivalent to transmitting bits for a longer time period, thus only increasing the energy per transmitted symbol and hence the SNR.

| Sequence 1 | | | Sequence 2 | | | Sequence 3 | | | Sequence 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_1$ | $c_{1,1}$ | $c_{2,1}$ | $s_2$ | $c_{1,2}$ | $c_{2,2}$ | $s_3$ | $c_{1,3}$ | $c_{2,3}$ | $s_4$ | $c_{1,4}$ | $c_{2,4}$ |

Figure 2.26: Bit Pattern for a systematic code of rate $r = 1/3$. For each input bit, one sequence consisting of a systematic and two parity bits is produced.

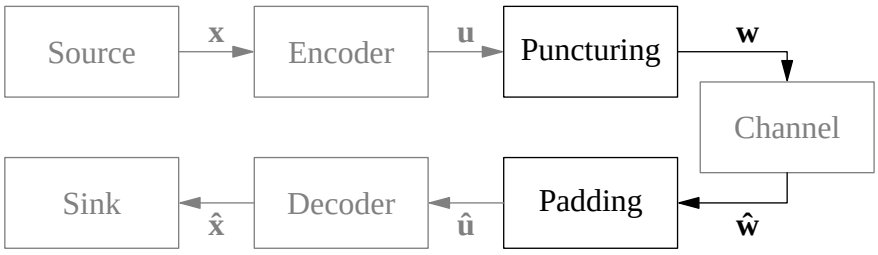



Figure 2.27: Transmission system with punctured input to the transmission channel.

achieves rate $1/2$ by alternately puncturing a code bit from the constituent encoders. Consider a systematic code as shown in Figure 2.26. Each input bit that is fed into the encoder results in a sequence of 3 bits, the input (systematic) bit, followed by two parity bits. Alternately puncturing parity bits, leads to a $1/2$ rate code with the sequence $s_1\, c_{1,1}\, s_2\, c_{2,2}\, s_3\, c_{1,3}\, s_4\, c_{2,4} \ldots$.

The decoding of a punctured code is particularly easy when soft-decoding using L-values is performed. After receiving the punctured sequence $\hat{\mathbf{w}}$, the receiver fills in the holes created by the encoder with an L-Value of 0, thus informing the soft-decoder that nothing is known about this bit (see section 2.1.2). The advantage of this approach is that the decoder does not need to be specially adjusted for punctured input and can remain unchanged and unaware of the puncturing. Figure 2.27 summarises the resulting communication system in which puncturing is performed at the transmitter and the receiver pads the punctured bits with 0s.

Type-II Hybrid ARQ transmission systems (see section 2.5) benefit from puncturing because it naturally partitions the bit sequence into two parts. If the transmitter is notified of a transmission failure,[18] it may decide to send those previously punctured bits to the receiver instead of repeating the previously transmitted sequence.

[18] Assuming that the transmission failure is not not caused by packet loss.

| Time Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Systematic | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ | $s_8$ |
| Parity 1 | $c_{1,1}$ | $c_{1,2}$ | $c_{1,3}$ | $c_{1,4}$ | $c_{1,5}$ | $c_{1,6}$ | $c_{1,7}$ | $c_{1,8}$ |
| Parity 2 | $c_{2,1}$ | $c_{2,2}$ | $c_{2,3}$ | $c_{2,4}$ | $c_{2,5}$ | $c_{2,6}$ | $c_{2,7}$ | $c_{2,8}$ |

Figure 2.28: Code matrix for a systematic code of rate $r = 1/3$. For each input bit, one sequence, shown as a column, of a systematic and two parity bits is produced.

### 2.6.2 Puncturing Tables

Consider the output sequence of a systematic code of rate $r_{\text{base}} = 1/3$, called the base code, shown in Figure 2.26. This sequence can be represented as a matrix, **U**. Each row represents an output tap of the encoder (in this case systematic, parity 1 and parity 2), each column corresponds to one time index, i.e. input bit (Figure 2.28).

A (repetitive) puncturing pattern of period $P$ for a code of rate $1/N$ is expressed as a $N \times P$ matrix **P** called a puncturing table with puncturing period $P$. Like for the code matrix, **U**, shown in Figure 2.28, each row corresponds to one output tap of the encoding process, the columns correspond to time shifts. The puncturing table **P** is applied to the code matrix **U** as a mask, only allowing those bits where **P** contains a 1 to be transmitted. If **U** contains more columns than **P**, **P** is applied repetitively, shifted by $P$ time indices each time. The puncturing table for the rate $1/2$ punctured Turbo Code described earlier is thus

$$\mathbf{P}_{binary} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{P}_{octal} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}, \tag{2.51}$$

where a 1 indicates transmission of and a 0 indicates puncturing of the corresponding bit. An octal notation of the puncturing table is formed like the octal representation of generator polynomials, described in section 2.3.5, by reading each row as a binary number, the first column containing the most significant bit. Figure 2.29 shows an example of applying the puncturing table (2.51) to the code matrix from Figure 2.28.

The rate of the punctured code is determined by the number of ones in the puncturing table, $m$, and the puncturing period $P$. An input sequence **x** of length $L$ encoded with rate $r = 1/N$ results in an encoded sequence **u** of length $L' = NL$. If **u** is punctured

| Time Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Systematic | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ | $s_8$ |
| Parity 1 | $c_{1,1}$ | $\times$ | $c_{1,3}$ | $\times$ | $c_{1,5}$ | $\times$ | $c_{1,7}$ | $\times$ |
| Parity 2 | $\times$ | $c_{2,2}$ | $\times$ | $c_{2,4}$ | $\times$ | $c_{2,6}$ | $\times$ | $c_{2,8}$ |

Figure 2.29: Code matrix for a systematic code of rate $r = 1/3$ after puncturing with the puncturing table provided in (2.51). A punctured bit is marked by $\times$.

with **P**, the resulting sequence **w** has length

$$\mathring{L}' = \frac{L'm}{NP} = \frac{NLm}{NP} = \frac{m}{P} \cdot L, \tag{2.52}$$

hence the code rate of the punctured code is

$$\mathring{r} = \frac{L}{\mathring{L}'} = \frac{P}{m}. \tag{2.53}$$

In order to ease presentation the operator $\mathbf{u} \circledcirc \mathbf{P}$ is used to indicate that a block **u** of length $L'$ is punctured by **P** (the puncturing table is applied repetitively if necessary) to obtain a sequence of bits **w** of length $\mathring{L}' = (L'm)/(NP)$. For a rate $1/3$ systematic code and **P** given in (2.51), $m = 4$, $P = 2$, and thus $\mathring{r} = 2/4 = 1/2$.

Puncturing divides the bit sequence in transmitted and punctured bits, the latter of which can be used for retransmissions in incremental redundancy type-II hybrid ARQ systems. Puncturing tables provide control over the amount of transmitted bits by allowing the number of ones in the table, $m$ to be adjusted. Additional protection of the input sequence can be achieved by increasing $m$. In order for the added protection to be useful in an incremental transmission system, an additional rate-compatibility constraint needs to be placed on a sequence of puncturing tables.

### 2.6.3 Rate-Compatible Punctured Codes

Hagenauer[46] used a set of puncturing tables for a base code of rate $1/N$ to obtain an ensemble of codes with rates of $P/(P+l)$, where $l$ is an integer that can be varied between 1 and $(N-1)P$. Thus, the obtainable code rate varies between $P/(P+1)$ and $1/N$. In order for these codes to be suitable for incremental redundancy transmission, a rate-compatibility restriction is placed on the tables. This restriction states that a new

| Time Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Systematic | × | × | × | × | × | × | × | × |
| Parity 1 | × | $c_{1,2}$ | × | $c_{1,4}$ | × | $c_{1,6}$ | × | $c_{1,8}$ |
| Parity 2 | × | × | × | × | × | × | × | × |

Figure 2.30: Code matrix for a systematic code of rate $r = 1/3$, punctured with $\mathbf{P}_\Delta = \mathbf{P}_3 - \mathbf{P}_2$, given in (2.55). A punctured bit is marked by $\times$.

puncturing table $\mathbf{P}_{l+1}$ can be obtained from $\mathbf{P}_l$ only by adding a single 1 to $\mathbf{P}_l$ where $\mathbf{P}_l$ contains a 0. This ensures that any puncturing table $\mathbf{P}_l$ contains all 1s of higher rate puncturing tables. In turn, a codeword that is punctured with $\mathbf{P}_{l+1}$ will contain all bits of the same codeword punctured with $\mathbf{P}_l$. In an incremental redundancy transmission system, only the additional bits need to be transmitted, or equivalently, the codeword punctured with $\mathbf{P}_\Delta = \mathbf{P}_{l+1} - \mathbf{P}_l$. Hagenauer varies the integer $l$ from 1 to $(N-1)P$, resulting in a maximum code rate of $r = \frac{P}{P+1}$. In order to compare rate-compatible punctured codes with variable rate transmission, introduced in section 2.6.4, it is necessary to obtain codes of rate $r = 1$. In this thesis, Hagenauer's method is therefore extended to allow $l$ to vary from 0 ($\mathbf{P}_0$, which contains $P$ 1s) to $(N-1)P$ ($\mathbf{P}_{(N-1)P}$, which contains all 1s) and thus achieve a code rate between 1 and $1/N$.

Consider a rate $1/3$ systematic code such as shown in Figure 2.28 and a series of rate-compatible puncturing tables

$$\mathbf{P}_{0\ldots4} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{2.54}$$

to obtain an ensemble of codes of rate $1, \frac{2}{3}, \frac{1}{2}, \frac{2}{5}, \frac{1}{3}$ $\left(\frac{2}{2}, \frac{2}{3}, \frac{2}{4}, \frac{2}{5}, \frac{2}{6}\right)$. The 4th transmitted packet (the 3rd increment) then contains the bits resulting from the puncturing pattern

$$\mathbf{P}_\Delta = \mathbf{P}_3 - \mathbf{P}_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{2.55}$$

which results in the packet $c_{1,2}\, c_{1,4}\, c_{1,6}\, c_{1,8} \ldots$ (see also Figure 2.30). Hagenauer did not present a strategy by which to obtain good puncturing tables. Optimal puncturing tables are still being researched, e.g. [4, 109, 139].

### 2.6.4 Variable Rate Transmission

Puncturing tables and sequences of puncturing tables that yield rate-compatible codes are primarily packet based, since they make statements about collections of output bits. A single puncturing table splits the available bits into the sets of initially punctured and initially sent bits. A set of rate-compatible puncturing codes has a granularity of $L/P$ bits for each successive puncturing table. No statement is made about the order of the bits within each packet. However, the optimal rate $\check{r}$ is not necessarily close to one of the rates provided by rate-compatible puncturing tables. A reduction of the granularity of rate-compatible punctured codes can be obtained by increasing the puncturing period, $P$, thus increasing the number of available rates and allowing a transmission closer to $\check{r}$. In the extreme case, where $P = L$, the granularity is 1 bit, and adding a 1 in the puncturing table $\mathbf{P}$ corresponds to adding precisely one bit in the encoded sequence $\mathbf{u}$. In this case, it is no longer necessary to store a sequence of puncturing tables, since that sequence of tables essentially only defines a transmission order of the sequence $\mathbf{u}$. Instead, it is more efficient to assign to each bit in $\mathbf{u}$ its relative transmission position. Effectively, this assignment is a permutation, $\prod$, applied to $\mathbf{u}$, resulting in the reordered sequence $\mathbf{v}$. The transmitter can now form a package of size $s_{\text{packet}}$, by selecting for transmission the first $s_{\text{packet}}$ bits of $\mathbf{v}$ (i.e. simply truncating $\mathbf{v}$) and transmitting the resulting sequence $\mathbf{w}$. Assuming soft-decision LLR decoding at the receiver, all that is required at the receiver is to append $\mathbf{0}$ of length $s_{\mathbf{w}} - s_{\mathbf{v}}$ to $\hat{\mathbf{w}}$ to form the sequence $\hat{\mathbf{v}}$ of length $s_{\mathbf{w}} = L/r$. Next, the receiver applies the inverse permutation $\prod^{-1}$ to $\hat{\mathbf{v}}$ to obtain the sequence $\hat{\mathbf{u}}$, suitable for decoding. Figure 2.31 illustrates the thus modified communication system. Note that this transmission system, from the receiver point of view, is nothing different than a combination of a bit-erasure channel (the puncturer) with detected erasures followed by the original transmission channel. Thus the code used should be able to handle bit-erasures reasonably well.

Enabling the transmitter to send an arbitrary number of bits increases the precision with which a transmission system can achieve the optimal transmission rate $\check{r}$. Consider an input block of length $L$ which is encoded and then truncated to length $\mathring{L}'$ to achieve rate $L/\mathring{L}'$. The closest neighbour rates are obviously $L/(\mathring{L}'-1)$ and $L/(\mathring{L}'+1)$, respectively. The distances to these neighbour rates are

$$\frac{L}{\mathring{L}'-1} - \frac{L}{\mathring{L}'} = \frac{L\mathring{L}' - L\mathring{L}' + L}{\mathring{L}'\left(\mathring{L}'-1\right)} = \frac{L}{\mathring{L}'\left(\mathring{L}'-1\right)}, \tag{2.56}$$

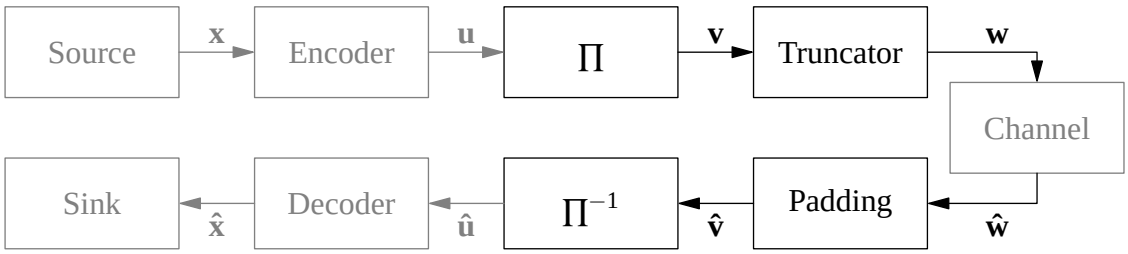



Figure 2.31: Communication system with variable rate input to the channel.

and $L/\left(\mathring{L}'\left(\mathring{L}'+1\right)\right)$, respectively. The maximum error, $\varepsilon_{\max}$, to $\check{r}$ is thus

$$\varepsilon_{\max} = \frac{1}{2}\frac{L}{\mathring{L}'\left(\mathring{L}'-1\right)} \approx \frac{1}{2}\frac{L}{\mathring{L}'^2} = \frac{1}{2L}\cdot\frac{L^2}{\mathring{L}'^2} = \frac{\mathring{r}^2}{2L}. \tag{2.57}$$

The corresponding calculation for an ensemble of rate-compatible punctured codes yields $\varepsilon_{\max} \approx P/\left(2\cdot(P+l)\cdot(P+l-1)\right) = \mathring{r}^2/(2P)$. Since typically $P \ll L$, the rate precision of an arbitrary number of bits is, unsurprisingly, better than that of rate-compatible codes (with the extreme case of $P = L$, when both are identical).

The transmission system shown in Figure 2.31 can be used in an incremental redundancy system. The first packet of size $s_{\text{packet}}$ is formed as described by selecting the first $s_{\text{packet}}$ bits from $\mathbf{v}$. An increment of size $s_{\text{inc}}$ is formed likewise. Given that $n_{\text{sent}}$ bits have been transmitted, the transmitter uses bits $n_{\text{sent}}+1$ to $n_{\text{sent}}+s_{\text{inc}}$ (i.e. the next $s_{\text{inc}}$ bits) from $\mathbf{v}$. The receiver accumulates all $n_{\text{sent}}$ received bits $\hat{\mathbf{w}}$ by appending them in packet order,[19] before padding and applying $\Pi^{-1}$.

### Mapping Rate-Compatible Puncturing Tables to Transmission Order

In this thesis, rate-compatible puncturing tables are converted into a transmission order such that the same simulation setup can be used to simulate a variable transmission order and rate-compatible puncturing tables. This has the benefit of increased confidence in the comparability of the results, since the developed algorithm can be checked independently of the rest of the transmission simulation. Algorithm 2.1, developed for this thesis, punctures the identity sequence $\mathbf{I} = [1,2,3,...,s_{\mathbf{u}}]$ with each of the delta

[19] It is assumed that some protocol in the transmission stack provides packet ordering (e.g. TCP or UDP+RTP) or that only one packet is in transmission at any given time so that packet ordering can be maintained.

tables $\mathbf{P}_\Delta$. The resulting sequences are concatenated to form the transmission order $\prod$. The ordering of the individual segments of length $\mathring{L}_\Delta = L/P$ is left undefined by the puncturing table. Algorithm 2.1 therefore randomises the ordering of bits of the individual segments (lines 3 and 7). However, this step is not strictly required to obtain a transmission order from the sequence of puncturing tables. Algorithm 2.1 assumes that the permutation table $\mathbf{P}_0$, required to achieve rate 1, exists and that each consecutive puncturing table adds a single 1, up to $\mathbf{P}_{(N-1)P}$, defined to be $\mathbf{1}$, a matrix filled with 1s. It is straightforward to extend algorithm 2.1 to sequences of puncturing tables that add more than one 1, however, if the last tables is not $\mathbf{1}$, the transmission order is undefined for the remaining bits, leaving only the choices of random selection or appending all resulting bits in the order in which they appear in $\mathbf{u}$.

**Algorithm 2.1** Map rate-compatible puncturing table to reordering permutation.

1: $\mathbf{I} = [1, 2, ..., L/r]$
2: $\prod = \mathbf{I} \circledcirc \mathbf{P}_0$
3: (optionally) randomly permute $\prod$
4: **for** $l = 1$ to $(N-1)P$ **do**
5: $\quad \mathbf{P}_\Delta = \mathbf{P}_l - \mathbf{P}_{l-1}$
6: $\quad \mathring{\mathbf{U}} = \mathbf{I} \circledcirc \mathbf{P}_\Delta$
7: $\quad$ (optionally) randomly permute $\mathring{\mathbf{U}}$
8: $\quad$ append this sequence to $\prod$
9: **end for**
10: **return** $\prod$

# Chapter 3

# Variable Rate Transmission of RSC Codes

Section 2.6.4 introduced a communication system that is capable of transmitting at a finely variable, arbitrary rate. This was made possible by reordering the encoder output, **u**, by a reordering permutation, $\prod$, and then removing bits from the end. While from this transmission setup it is clear that the bits placed at the end are best chosen to be the least important ones, section 2.6.4 does not specify how to construct an effective transmission order $\prod$. This chapter is devoted to this transmission order for RSC codes. RSC codes are investigated in this thesis for two reasons. Firstly, RSC codes, like other convolutional codes are well understood codes of relatively low complexity. Secondly, RSC codes are used as constituent codes for the capacity achieving Turbo Codes, which is why RSC codes are considered in this thesis rather than other convolutional codes. The investigation of RSC codes in this chapter thus lays the foundation for the treatment of Turbo Codes in chapter 4. **Section 3.1** deals in detail with possible transmission ordering strategies, starting with the performance of no reordering and random reordering. Both methods prove unsatisfactory and new strategies based on the properties of RSC codes are developed in order to overcome their shortcomings. Following this development, **section 3.2** discusses the reasons for the performance exhibited by the transmission schemes and provides an upper bound on the ability of RSC codes to recover erased bits. It will be shown that the transmission of parity or systematic bits exclusively in the first packet significantly influences the performance and reasons are presented why this is the case and how this can aid in the search for

good generator polynomials. **Section 3.3** shows that the input block length, as expected, does not influence the qualitative performance of the presented transmission schemes and neither does the constraint length of the used code. The results obtained in section 3.1 are based on the AWGN channel. This is expanded to ISI and Rayleigh channels in **sections 3.4 and 3.5**, respectively. Finally, **Section 3.6** concludes this chapter and summarises the results.

## 3.1 Development of an Effective Ordering Strategy

This section develops construction strategies for the transmission order, $\prod$. The performance of the individual transmission strategies is measured by simulating the communication system shown in Figure 3.1 for 10,000 input packets, $\mathbf{x}$, of 2048 equiprobable[1] bits each. For every new packet a new transmission order $\prod$ is generated according to the evaluated strategy. It is well known, that soft-value decoding provides significant gains over hard decoding ($\approx 2\,$dB, see section 2.1.4). Furthermore, Turbo Codes rely on soft-value decoding, in particular soft-in soft-out decoding to function properly. Since the investigation of RSC codes serves to better understand Turbo Codes, the constituent decoder in chapter 4 is used as the RSC encoder. In order to ease presentation and discussion of the ordering strategies an AWGN channel is used. This model of a transmission channel, typically, does not approximate realistic conditions well, since it ignores multi-path and fading effects which are commonly experienced. Different channel models are considered in sections 3.4 and 3.5. Another major assumption is that the replies needed for ARQ to work correctly, the ACK and NAK signalling, is error-free. This can typically be guaranteed by protecting these bits with very strong codes, since they only contain 1 bit of information. Moreover, assuming that no erroneous NAK packet is treated as an ACK packet, the presence of errors only increases the number of increments that need to be transmitted, which is shown by Malkamäki and Leib[81], who also provide upper bounds for the number of required transmissions under noisy feedback.

[1]The reason for assuming equiprobable input bits is so that the transmission system is not limited to any specific source content (with possible correlations in the source). Instead, perfect source coding is assumed for all sources, such that all correlation in the source signal is removed and hence the information content of each bit is highest. The necessary consequence of this is that source bits are equiprobable.

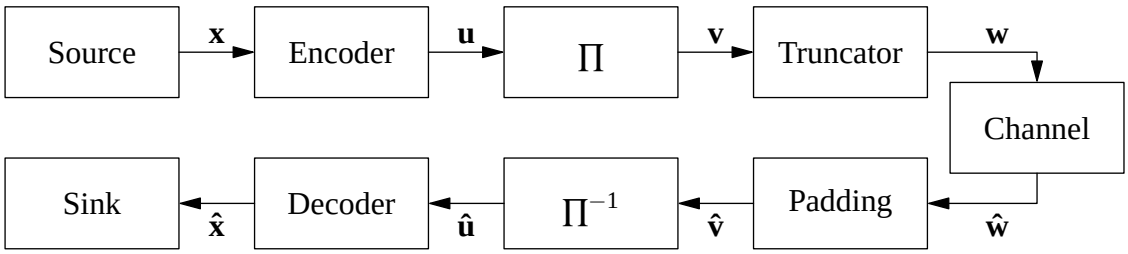



Figure 3.1: Communication system used in simulations. See table 3.1 for common parameters. The parameters for constructing the transmission order $\Pi$ are detailed in the respective sections.

The strategies are compared by their highest, error-free rate and how close this rate comes to the capacity of the BIAWGN channel. In order to determine this optimal rate, $\check{r}$, the minimum number of bits required for error-free transmission, $\check{s}_{\mathbf{w}}$, needs to be determined for the respective SNR of the channel. The standard approach is to simulate all possible packet sizes. However, this is prohibitively expensive in terms of simulation runtime. Instead, a binary search over the space of all possible packet sizes, $s_{\mathbf{w}}$, is performed to determine $\check{r} = s_{\mathbf{x}}/\check{s}_{\mathbf{w}}$.

The decoder output for the minimum number of bits, $\check{s}_{\mathbf{w}}$, will have a very small reliability (i.e. very small L-values). Therefore it is possible that additional bits with a strong noise sample added to them can "confuse" the decoder, such that more bits cause bit-errors to reoccur. Searching for these situations, it was in fact observed that additional bits can cause more errors. However, it was observed, that the number of bits required for the decoder to produce a stable, error-free output is only in the tens of bits. The impact on performance for a block size of 2048 bits is negligible.

Table 3.1 lists all common parameters for the performed simulation (unless stated otherwise in the respective section). The polynomials $g^{(0)}$ and $g^{(1)}$ were used because they had the smallest Bit Error Ratio in a preliminary brute-force comparison of all possible polynomials of order 6 over an AWGN channel. The decoding is performed with the BCJR algorithm, using LLR values in order to avoid numerical problems (see section 2.3.2).

| Simulation Parameters | |
|---|---|
| **General** | |
| Simulated packets | 10000 |
| **Source** | |
| Type | Memoryless |
| Samples per packet ($s_{\mathbf{x}}$) | 2048 |
| $p(x=1)$ | 0.5 |
| **Encoder** | |
| Type | Recursive Systematic Convolutional (Fig. 2.16) |
| $g^{(0)}$ (feedback) | $1+D^1+D^2+D^3+D^6$ |
| $g^{(1)}$ (parity) | $1+D^2+D^3+D^5+D^6$ |
| **Channel** | |
| Type | Additive White Gaussian Noise |
| **Decoder** | |
| Type | Soft-In Soft-Out BCJR Algorithm (Sec. 2.3.2) |

Table 3.1: Common parameters for the simulated transmission system shown in Figure 3.1 (unless stated otherwise).

### 3.1.1 No (Identity) Ordering, Preliminaries

The most naive transmission order is no order at all, i.e. $\prod$ is the identity permutation $\mathbf{I}$. The sequence of bits is simply the one that the encoder produces (see Figure 2.26). This ordering scheme, as shown in Figure 3.2, provides no gain over a transmission system that simply transmits the entire sequence $\mathbf{u}$. Clearly a more sophisticated approach is required if a rate-gain is to be achieved. The reason that the identity order is unable to achieve a rate gain with increasing channel quality stems from the fact that the code needs to correct the erasures caused by the puncturing. Section 3.2 discusses this problem in some more detail. For high rate codes the systematic and parity bits of a continuous section of the sequence $\mathbf{u}$ are punctured, the decoder has no possibility of recovering the respective input bit.

In order to ease presentation of the other ordering strategies, the rate performance of the base code is omitted because the performance of the identity order is virtually identical to the rate of the base code, $r_{\text{base}}$, the minimal rate at which the transmission system is expected to function. Note that the performance of the identity ordering is not perfectly straight and horizontal, even though Figure 3.2 might suggest that. The improvements of performance as the SNR increases are in the single bit range, however, and thus largely negligible. Figure 3.2 also shows the capacity bound of a BIAWGN channel, calculated using equation (2.19) and marks three regions of interest. The "maximum rate region" is the region of high SNR values, where the transmission order reaches its maximum possible rate, $r_{\max}$. In the maximum rate region, a good ordering strategy will achieve a rate of $r_{\max} = 1$ or at least $r_{\max} \approx 1$, such that efficient communication is possible for high SNR channels. Reducing the SNR, the performance of the ordering strategy leaves the maximum rate and enters the "adaptive region". In the adaptive region, changes in the channel SNR result in a changed, or adapted, rate. In the adaptive region the merit of the ordering strategy is determined by the distance to the ideal, theoretical, capacity of the BIAWGN channel. Finally, for low[2] SNR, the code reaches termination, where the base code is no longer able to correct errors. As explained, the optimal rate is determined from the number of bits for which no errors occur. At low SNR some packets will have a residual BER after decoding even for the maximum number of bits, $s_{\mathbf{u}}$. For these bits, the number of errors is calculated. Once the BER, $p_{\varepsilon}$, averaged over all packets exceeds $10^{-5}$, the plot ends since a comparison

[2] Low in this context is relative and depends on the base code.

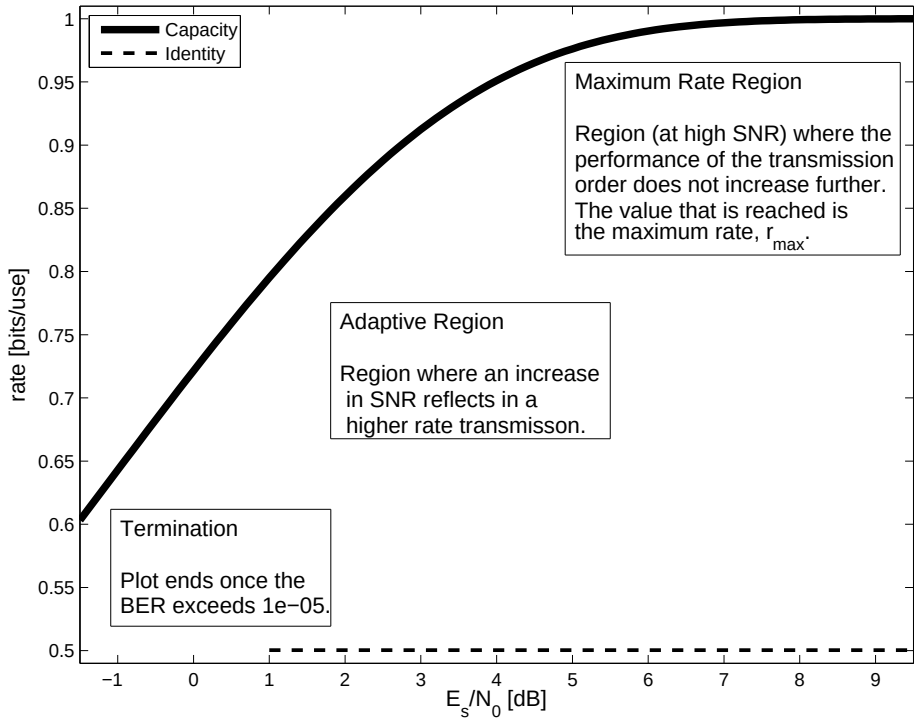



Figure 3.2: Performance of the identity ordering strategy (dashed) against the theoretical capacity (bold). Also shown are three regions of interest. In the maximum rate region, a transmission order reaches its best possible rate. In the adaptive region, the rate is adapted to the channel SNR. At termination, the base code cannot correct all errors and the BER exceeds $10^{-5}$. Note that the dotted line is neither perfectly straight nor perfectly horizontal.

with the error free transmission required at high SNR cannot be reasonably made. The BER was chosen over the packet-error-rate, since it is a typical measurement when evaluating the performance of coding schemes. However, it has to be noted that it is only relevant as a termination criterion. For sufficiently large SNR, the packet and bit error rates in the simulation are 0 by design, so the difference is of no significant consequence to the results presented in this thesis.

### 3.1.2 Random Ordering

The naive approach of sending the code sequence $\mathbf{u}$ in its original order does not provide any benefit, due to puncturing the information provided by systematic and parity bits unevenly. In order to ensure that the code sequence, $\mathbf{u}$, does not contain large gaps of information, the sequence $\mathbf{u}$ can be randomised.[3] A random permutation $\prod$ of length $s_{\mathbf{u}}$ can be constructed by taking the identity sequence $\mathbf{I}$ of length $s_{\mathbf{u}}$ and repetitively generating two indices, $i_1$ and $i_2$ within the interval $(1, s_{\mathbf{u}})$ and exchanging the contents of the cells at $i_1$ and $i_2$.

A randomised transmission order, shown in Figure 3.3, performs reasonable well at low SNR values. It is able to adapt the transmission rate to the different channel conditions, varying the rate between 0.6 and 0.75. In particular, the termination rate is higher than the base code rate, $r_{\text{base}}$, by about 0.1. However, despite the fact that the bits are randomly punctured and thus information is, on average, evenly "taken out" of the transmitted sequence $\mathbf{w}$, the random ordering strategy fails to achieve $r_{\max} \approx 1$. Instead, Figure 3.3 shows the maximum attainable rate, obtained empirically, to be $r_{\max} = 0.7507$. Section 3.2 discusses the problem of decoding erasures, particularly randomly chosen ones (Figure 3.13 in particular is of interest in this context).

### 3.1.3 Code Structure Based Ordering

In order to overcome the shortcomings of the random ordering, namely the inability to achieve a maximum rate $r_{\max} \approx 1$ requires knowledge of the structure of the encoder output, $\mathbf{u}$. It was shown in Figure 2.26 that the encoder produces the sequence

[3] Randomisation is in fact the *only* possibility if for some reason the code structure is unknown or fluctuates intractably.

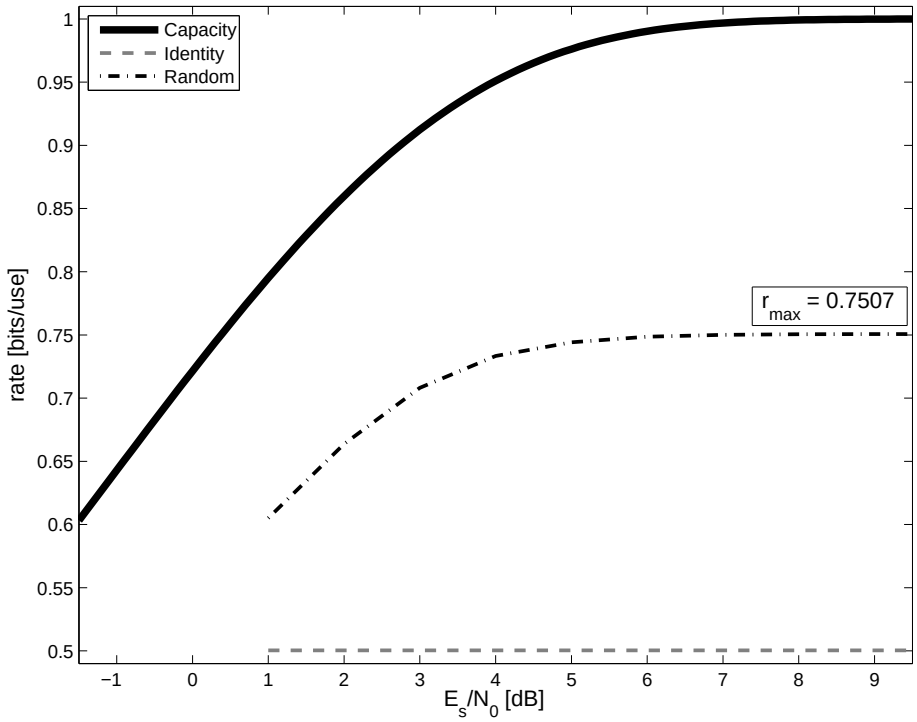



Figure 3.3: Performance of the random (dash-dot) ordering strategy compared to theoretical capacity (bold) and the rate of the base code, $r_{\text{base}}$, indicated by the performance of the identity ordering (dashed). Random ordering shows some promise in the adaptive region but only achieves an empirically determined maximum rate of $r_{\text{max}} = 0.7507$.

$s_1\, c_{1,1}\, s_2\, c_{1,2}\, s_3\, c_{1,3}\, s_4\, c_{1,4} \ldots$ from the input $x_1\, x_2\, x_3\, x_4 \ldots$ (due to the nature of the encoder $s_i = x_i$). In matrix form, the encoder output of the rate $1/2$ RSC code can be expressed as

$$\mathbf{U} = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 & \\ c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & \ldots \end{bmatrix}. \tag{3.1}$$

The identity permutation causes the code to be transmitted column-wise. It is well known, that the decoding process of a convolutional code has a window length of approximately six times the constraint length of the convolutional code [41]. Considering the trellis of the code, the essential consequence is that for the current trellis transitions, those transitions that are far away (in terms of the number of transitions) do not influence the current transition. Typically this argument is used for certain simplifications on the memory requirement of a decoder implementation. It is illustrative to consider the consequences of this simplification. More specifically, because this simplification is possible without affecting decoding performance significantly, it can be concluded, that the convolutional code does not need, or for all practical purpose use, information outside of this decoding window. Those bits do not contribute any useful information to the decoding, or put differently, the only bits that contain information required for successful decoding are those bits within one window length of the bit that is to be decoded. Using this decoding window argument, it is obvious why the identity permutation performs poorly. The bits are read column-wise, hence after $s_{\mathbf{x}}$ bits have been sent to obtain a rate of 1, only half of the columns have been used. It is safe to assume that $s_{\mathbf{x}}/2 \gg n$. Thus, for a significant number of bits at the end of the input no information is available.

A solution to the problem of missing information at the end of the input is to read $\mathbf{U}$ row-wise instead of column-wise.[4] The transmission sequence resulting from this operation, $s_1\, s_2\, s_3\, s_4 \ldots\, c_{1,1}\, c_{1,2}\, c_{1,3}\, c_{1,4} \ldots$, ensures that each column is used at least once.[5] The performance of this structured approach, shown in Figure 3.4, is very good for high SNR values. Because each input bit is "represented" once in the rate $r = 1$ codeword, all bits can be recovered, if very low distortions occur. An increase in distortion on the channel, however, leads to a rapid degradation of rate-performance. The rate of the structured transmission order crosses below the the maximum rate of the random transmission order of $r_{\max} = 0.7507$ at $6.4\,dB$. For lower SNR, the random order outperforms the structured order significantly.

[4] This is essentially just a transpose operation on $\mathbf{U}$ or, equally put, $\prod$ is a Block Interleaver.

[5] Given a packetisation of $s_{\mathbf{x}} + n$, this is the classical type 2 hybrid ARQ with 2 packets.

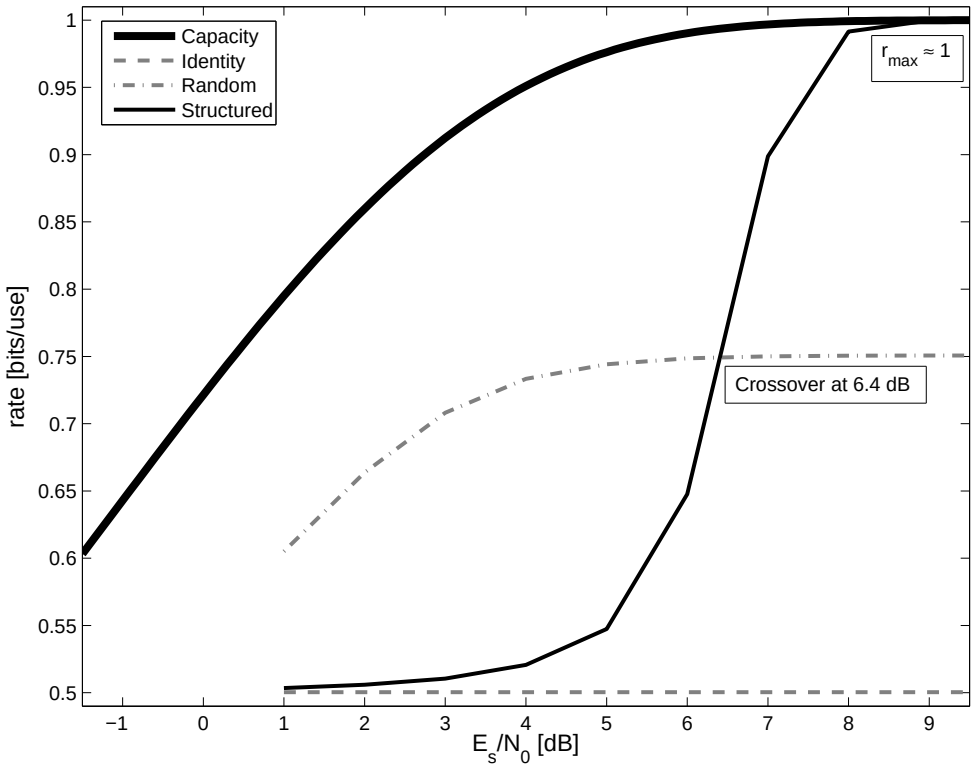



Figure 3.4: Performance of the structured (solid) and random (dash-dot) ordering strategies compared to the theoretical capacity (bold). The rate of the base code, $r_{\text{base}}$, is indicated by the performance of the identity ordering (dashed). Structured ordering achieves a maximum rate $r_{\text{max}} \approx 1$ and thus outperforms the random ordering strategy for high SNR. In the adaptive region, however, the structured strategy falls off sharply such that for low SNR, the random strategy outperforms the structured strategy with a crossover observed at $6.4\,\text{dB}$.

### 3.1.4 Block-Randomised Code Structure (BRCS) Ordering

The next logical improvement to the used ordering strategy is to merge the structured and the random approach to try to obtain good performance across all signal-to-noise ratios. Both the random ordering (section 3.1.2) and the structured approach (section 3.1.3) have their merits. The random ordering is superior for correcting strong distortion whereas the structured approach is superior for little to no distortion at high SNR. The structured approach overcame the shortcomings of the identity permutation, namely an inability to make available to the decoder information about the "later" input bits, by essentially transmitting the input bits (in the form of systematic bits) first. The argument of the limited window length of a decoder for convolutional codes made in section 3.1.3 however, still applies. The structured approach only reduced the problem but did not eliminate it. Given $s_{\mathbf{x}}$ input bits which are independent and equally distributed, a decoder cannot possibly recover the input sequence $\mathbf{x}$ from a code sequence $\mathbf{u}$ of less than $s_{\mathbf{x}}$ bits.[6] This in turn causes the first $s_{\mathbf{x}}$ bits to be sent, regardless of the used strategy. The order within this block of $s_{\mathbf{x}}$ bits does not affect the performance of the system, only the selection of bits that forms this set of $s_{\mathbf{x}}$ initial bits does.

Total randomness has been shown to perform poorly for high SNR values (see section 3.1.2). A transmission order needs to ensure that each input bit is "represented" in the first $s_{\mathbf{x}}$ bits in order to overcome the problems of a total random approach. However, a degree of randomness needs to be retained in order to achieve good error protection, since it is impossible to predict the exact location of errors. The next logical improvement is thus to merge the two previous approaches, i.e. exploit the structure of the RSC code while still retaining a degree of randomness in the selection of bits. Consider the encoded sequence $\mathbf{u}$, expressed as the matrix

$$\mathbf{U} = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 & \dots \\ c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & \end{bmatrix}. \tag{3.2}$$

Each column $i$ represents a sequence of bits that is generated by the encoder for the respective input bit $x_i$. Each row represents one of the encoder output taps. The encoder taps thus naturally partition the available bits into two blocks, one containing all the systematic bits and one containing all parity bits. In order to reap the benefit of randomness without sacrificing this structural distinction, each block is randomised *individually* before being transmitted, leading to block-randomised ordering based on

[6] In other words, each input bit has entropy 1.

the structure of the code, or block-randomised code structure (BRCS) ordering for short. The new code matrix is constructed such that

$$\mathbf{U}' = \pi_{\text{rows}}(\mathbf{U}) = \begin{bmatrix} \pi_{\text{rows},1}(\mathbf{U}_{\text{row}\,1}) \\ \pi_{\text{rows},2}(\mathbf{U}_{\text{row}\,2}) \\ \pi_{\text{rows},3}(\mathbf{U}_{\text{row}\,3}) \\ \pi_{\text{rows},4}(\mathbf{U}_{\text{row}\,4}) \\ \vdots \end{bmatrix} \tag{3.3}$$

where each interleaver $\pi_{\text{rows},i}$ only interleaves a single row $\mathbf{U}_{\text{row}\,i}$.[7]

The performance of the Block-Randomised Code Structure ordering strategy is shown in Figure 3.5. BRCS successfully combines the advantages of random ordering and the structured approach and is able to take the promising behaviour of the random ordering at low SNR and extend it to a maximum rate $r_{\max} \approx 1$.

### 3.1.5 Probabilistic Bit Selection of BRCS Ordering

The permutation design in section 3.1.4 was limited to only using randomness within each row. However this approach causes *all* systematic bits to be sent first. Thus the decoder receives no parity information in the first $s_{\mathbf{x}}$ bits, essentially preventing it from working. A possible solution to remedy this problem is to randomise the intra-column order, such that

$$\begin{aligned} \mathbf{U}'' &= \pi_{\text{rows}}(\pi_{\text{columns}}(\mathbf{U})) \\ &= \pi_{\text{rows}}\left(\begin{bmatrix} \pi_{\text{col},1}(\mathbf{U}_{\text{col}\,1}) & \pi_{\text{col},2}(\mathbf{U}_{\text{col}\,2}) & \pi_{\text{col},3}(\mathbf{U}_{\text{col}\,3}) & \pi_{\text{col},4}(\mathbf{U}_{\text{col}\,4}) & \dots \end{bmatrix}\right), \end{aligned} \tag{3.4}$$

where each interleaver $\pi_{\text{col},i}$ only interleaves a single column $\mathbf{U}_{\text{col}\,i}$. The interleaver $\pi_{\text{rows}}$ is constructed as described in equation (3.3). Note that the columns need to be randomised *prior* to randomising the rows to ensure that every input bit $x_i$ is still represented by either its systematic bit $s_i$ or one of the parity bits, $c_{j,i}$. A uniform randomisation of the columns, leads to a uniform distribution on the type of bit (Systematic or Code) in the first $s_{\mathbf{x}}$ bits. In the case of the rate $r = 1/2$ code employed in the simulations, a distribution of $0.5:0.5$.

[7] For ease of presentation, every row is interleaved, even though interleaving the first row does not affect performance, as described earlier in this section.

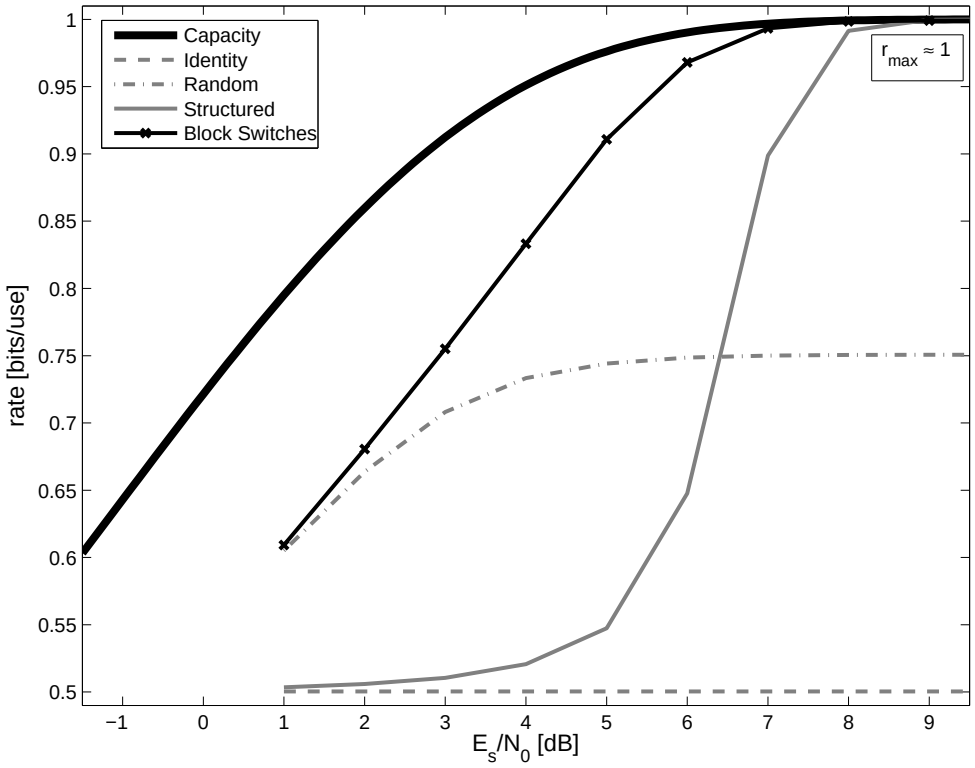



Figure 3.5: The Block Randomised Code Structure transmission order (cross) successfully combines the advantages of the structured (solid) and random (dash-dot) ordering strategies. In the adaptive region it comes significantly closer to the theoretical capacity (bold) and still reaches rate $r_{\text{max}} \approx 1$. It thus provides a significant gain over the rate of the base code, $r_{\text{base}}$, indicated by the performance of the identity ordering (dashed).

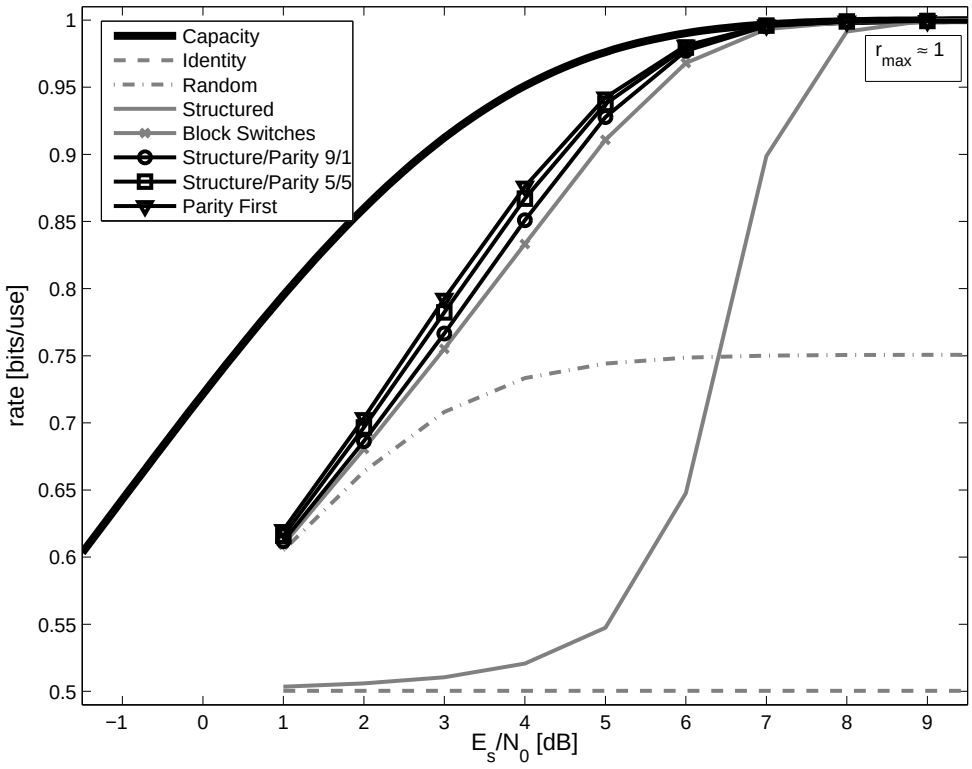



Figure 3.6: Probabilistic selection of the relative order of the systematic and parity bits produced for each input bit. Parity first (triangles) refers to the strategy where the parity bit produced for an input bit is always transmitted before the systematic bit is transmitted (the exact opposite of the block switches strategy indicated by crosses). The probability of transmitting the systematic bit before the parity bit was adjusted over the range $1.0$ to $0.0$, shown here are $p\,(systematic) = 0.9$ (circles) and $p\,(systematic) = 0.5$ (squares). The remaining values of the sequence $0.0, 0.1, 0.2, \ldots, 0.9, 1.0$ exhibit the same trend towards an optimum at transmitting parity first ($p\,(systematic) = 0.0$) but are omitted here for clarity. Keeping the parity information complete and transmitting systematic bits as supporting information achieves an improvement towards capacity (bold) of about $0.5\,$dB. Also shown for comparison are the random (dash-dot) ordering strategy, the structured approach without randomisation (solid) and the rate of the base code, $r_{\text{base}}$, indicated by the performance of the identity ordering (dashed).

A distribution of $0.5:0.5$ is not the only possible distribution of types of bits in the columns. Figure 3.6 compares the results of the distributions $1.0:0$ (the structured approach from section 3.1.4), $0.9:0.1$ (mostly systematic bits with one code bit for every 10 bits), $0.5:0.5$ (equal distribution of systematic and code) and $0.0:1.0$ (parity bits are transmitted first). The results indicate that transmitting all parity bits augmented by an incomplete selection of systematic bits outperforms transmitting all systematic bits augmented by an incomplete selection of parity bits in the adaptive region.[8] This observation is treated in more detail in section 3.2, a brief overview is presented here.

The superiority of transmitting parity bits first is of particular interest to type-II hybrid ARQ systems, since it leads to a computing overhead vs. performance trade-off. A type-II hybrid ARQ system that makes use of RSC codes as its error correcting code typically transmits the systematic bits in the first packet. The main advantage of this strategy is a potential reduction of the decoding overhead at the receiver. The systematic bits are, in fact, input bits (see section 2.3), hence the systematic bits can be used *directly*, without decoding, to check for the correctness of the received message if an error detecting code such as a CRC sum is used. Transmitting all parity bits as the first packet, on the other hand, allows for a performance gain of about $0.5\,\mathrm{dB}$ (the difference between the star and triangular markers in Figure 3.6). To achieve this gain, decoding is required regardless of the observed errors, because the value of any code bit depends on the input bit *and the state of the encoder having observed all previous input bits*.

The advantage of transmitting the parity bits first is due to the properties of the employed RSC code, in particular the transition labels under specific puncturing patterns. The difference between the code bits is discussed in detail in section 3.2.4. The $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}\right)$ code produces transitions labels that, given the knowledge of systematic bits, let the decoder, for a bit that occurs later in the trellis, infer the parity bit from the corresponding systematic bit. Sending this parity bit is wasteful, if the systematic bit is intact and hence, on average, more bits are required to correct errors.

[8] All integer distributions $1.0:0.0,\ 0.9:0.1,\ 0.8:0.2, \ldots 0.1:0.9,\ 0.0:1.0$ were simulated and exhibit the same trend. Most of the curves were omitted in Figure 3.6 to allow for a clearer presentation.

### 3.1.6 Binary Tree Based Ordering

The reason for randomising the individual rows in section 3.1.4 was to ensure that each bit receives some "coverage" by the transmitted sequence so that errors that occur during transmission can be corrected. Clearly, the block-randomised code structure ordering achieves this goal by providing at least the systematic or parity bit in the first packet. However, due to the randomisation of the blocks, the placement of additional bits is not guaranteed to cover all bits equally. In fact it is possible that two adjacent parity bits are transmitted consecutively, potentially "over-protecting" this part of **x**. The next logical step is thus to impose more structure onto the transmission order to ensure that the protecting bits are evenly spaced throughout the reordered sequence **v**. As will be shown in this section, an ordering strategy based on a binary tree emerges naturally when considering bit-by-bit transmission.

Under the assumption that errors occur with uniform distribution on an unpredictable location of the sequence **w**, no one transmission order can be selected that is optimal in all possible cases. Any protection of some data sequence **x** is only as strong as the weakest point of protection, since the least protected bit is most likely to cause a bit-error. Thus, since it is impossible to predict where and how strong the channel affects the individual bits of **w**, to cover the sequence **x** with $k$ additional parity bits, the ideal placement of the parity bits is an equal spacing of these parity bits. The number of additional parity bits, however, depends on the placement of these same parity bits as well as the current channel conditions. A hypothetical, optimal transmitter thus already needs to know that a given number of equally spaced parity bits will protect the transmission sufficiently when choosing the number of additional parity bits.

It is illustrative to compare the code bits to a bridge-building exercise, with each protective parity bit representing a pillar to support the bridge. An equal spacing of pillars (parity bits) is desirable, so that the weight of the cars is evenly distributed and the bridge itself has sufficient support without bending. This bending, when too strong will break the bridge much in the same way that too much bend (noise) in the received codeword will cause the decoding trellis to break (switch to a wrong path). Puncturing tables already provide an equal spacing of bits, since, by their design, they are applied repetitively. Furthermore, the sequence optimal puncturing tables developed by Rowitch and Millstein[109], see section 4.7, that additional ones in the puncturing tables are optimally placed to close large gaps in the punctured codeword, **w**.

The main problem with the requirement of equally spaced bits is that this order is fixed for a particular number of bits to be transmitted. Consider the sequence $x_1x_2x_3x_4x_5x_6$ that is to be transmitted with one additional parity bit. Equal spacing of the parity bits requires it to be placed in the middle, such that the sequence to be transmitted is $s_1s_2s_3\mathbf{c_{1,3}}s_4s_5s_6$.[9] However, if the number of parity bits is increased to two, the sequence to be transmitted is $s_1s_2\mathbf{c_{1,2}}s_3s_4\mathbf{c_{1,4}}s_5s_6$. This is essentially the problem that gives rise to the rate-compatibility criterion introduced by Hagenauer[46]. In an iterative transmission system, if the parity bit $c_{1,3}$ has been transmitted, it is not possible to "get this bit back" from the receiver and replace it with another bit, now determined to be more suitable.

If the requirement to select an arbitrary number of bits is kept, a strategy needs to be found that places *one additional bit at a time*. Considering the RSC code, it is worth to note, that it *returns to its initial state* through the use of tail bits. Therefore, 2 states of the encoder are known with certainty to the decoder, namely $s_0$ and $s_t$. The BCJR algorithm then employs a forward and a backward recursion starting from each of these known states. Given these two recursions, it is obvious that the bit in the middle is the one that the decoder knows least about, since it is removed furthest from a certain state. Thus the first parity bit is placed in the middle of the sequence $\mathbf{x}$. The next bit needs to be placed assuming that this bit is fixed. Two unprotected sequences of equal[10] length exist, separated by the first parity bit. By the same argument as before, the point of highest uncertainty will be in the middle of one of these sequences. Picking one at random, the next bit will be placed in the middle of the other one.

The 2 requirements placed on the transmission order, namely to consider each bit individually and consecutively and the requirement to place the bit such as to minimise the length of a sequence of code bits for which no protection has been transmitted yet is shown in Figure 3.7. From the way that Figure 3.7 presents the ordering scheme, it is obvious, that it resembles a binary tree, since one bit is placed at a time, thus splitting an existing unprotected sequence into 2 sub-sequences (e.g. placing bit 4 as the second bit splits the unprotected sequence 1-7 into the unprotected sequences 1-3 and 5-7).[11]. This tree-based approach ensures rate-compatibilitysince each previously placed bit is

[9] For ease of exposition systematic bits are all present and parity bits are used to augment them, since that is the conventional approach of incremental redundancy.

[10] Assuming that $s_{\mathbf{x}}$ is divisible by 2.

[11] In particular, the transmission order is equal to the way that a binary tree can be stored in an array (see [23]) with all non-existing nodes removed.

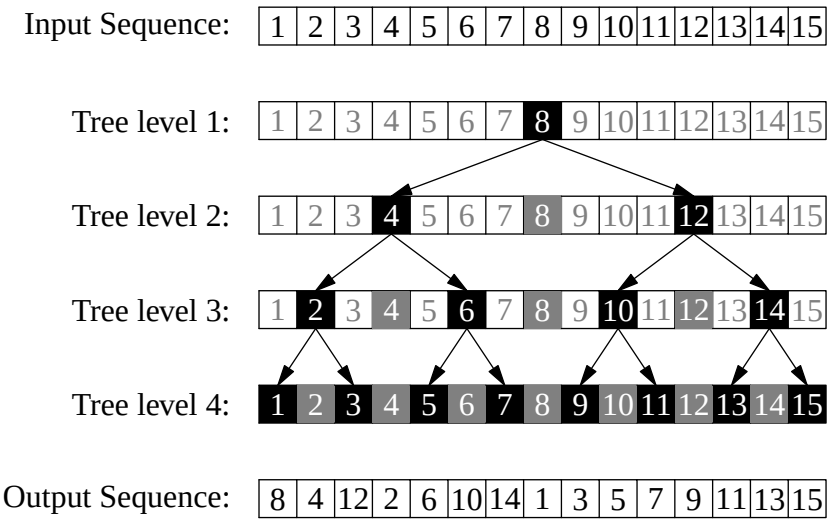



Figure 3.7: Schematic depiction of the binary tree ordering strategy for an input of length 15. Black filled boxes indicate that the index is used on the respective level, grey filled boxes indicate that the block has already been used in a higher level of the tree. The reordered output is the standard way of storing binary trees in arrays.

considered fixed, while spacing the parity bits as closely to equidistant as possible, given that bits, once transmitted, cannot be moved..[12]

The complexity of constructing the tree based strategy is comparable to that of the block-randomised strategy. Clearly the number of randomisations is dependent on the number of bits, for which a random order is going to be generated. This number does not change significantly, since all bits but the first (the root) still need to be assigned a random position. The only difference in complexity is then the partitioning of the available bits into levels of the tree. This again is linearly dependent on the number of bits, since when generating a tree from an input, every bit is only used once. Thus the overall complexity remains at $O(s_{\mathbf{u}})$.

The order of transmission of the siblings of each level of the binary tree remains undefined, however, the same problem of "over-protection" discussed in section 3.1.4 apply, namely that some parts of the input receive more protection than necessary at the expense of other parts of the input sequence, particularly at the last level of the tree, which contains every other parity bit. To remedy this problem, the order of the siblings

[12]The spacing of parity bits is optimal, i.e. equivalent to an equal spacing of bits, if the number of parity bits is $2^i - 1, \forall i \in \mathcal{N}$.

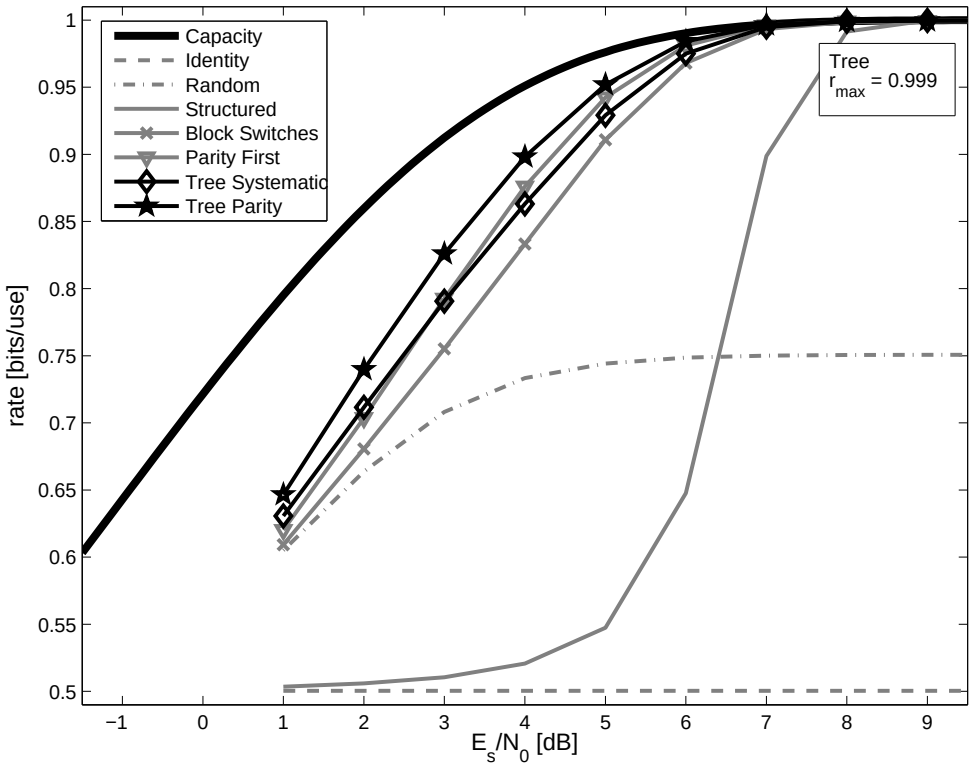



Figure 3.8: Tree based selection of bits. Tree Systematic (diamond) indicates that systematic bits are transmitted first, Tree Parity (stars) indicates that all parity bits are transmitted first. The probability of transmitting the systematic bit first, $p(\text{systematic})$, was adjusted over the range $1.0$ (systematic first) to $0.0$ (parity first), but the results are omitted for clarity. Only the extreme points of performance, $1.0$ (minimum) and $0.0$ (maximum), are shown since the remaining values of the sequence $0.0, 0.1, 0.2, \ldots, 0.9, 1.0$ monotonously decrease in performance as the probability of transmitting the systematic bit increases. Keeping the parity information complete and transmitting systematic bits as supporting information achieves an improvement towards capacity (bold) of about $0.5\,\text{dB}$. Also shown for comparison are the random (dash-dot) ordering strategy, the structured approach without randomisation (solid), the structured approach with randomisation (triangles for parity first and crosses for systematic first) and the rate of the base code, $r_{\text{base}}$, indicated by the performance of the identity ordering (dashed).

is randomised for each level. The tree based approach, shown in Figure 3.8, performs consistently better than the structured approach with randomisation. Again, as also seen in Figure 3.6, transmitting the parity bits before transmitting the systematic bits is significantly better.

A probabilistic selection of systematic and parity bits is also possible. The procedure is similar to the one used in section 3.1.5, with the modification, that $\pi_{\text{rows}}$ produces a tree ordering for each row. The results, omitted from Figure 3.8 for clarity of presentation, follow the same pattern as in Figure 3.6, decreasing in performance as the probability of transmitting the systematic bit, $p(\text{systematic})$ is increased from 0.0 (transmit parity first) to 1.0 (transmit systematic first).

Sometimes the randomisation of bits can lead to problems. In particular, when implementing hardware solutions, a random interleaver poses a significant bottleneck. It cannot be implemented in hardware directly, since the connections between input and output of the interleaver are random, the only possibility is to implement random interleaving in DSPs. This implementation in DSPs, however, requires that the DSP has enough memory available to perform the switching and secondly it introduces an additional delay. It is therefore interesting to compare the performance of the randomised tree with a deterministic version, in which the order of the siblings relative to each other is maintained.[13] Such a deterministic tree version is shown in Figure 3.9.

It is interesting to note that the performance of the deterministic version degrades with an increasing number of bits. An explanation for this is provided by the number of bits on each level of the tree. It is well known, that the number of elements of each level of a binary tree is $2^l$, where $l = 0,1,2,\ldots$ is the depth of the level, the root of the tree being on level 0. Assuming that the tree contains a total of $2^{l+1}-1$ elements, the lowest level, $l$ will contain roughly as many elements as the rest of the tree. In the tree based ordering, this level contains every other bit. Thus the performance of this last level alone can be likened to the performance of the structured, unrandomised approach shown in Figure 3.4. The reason why the deterministic tree outperforms the structured approach is that the lower levels have created a transmission order that already spans the entire codeword. In particular at the last layer, every second bit has already been transmitted and the remaining bits just fill the gaps left by this "scaffolding". It is obvious from the increasing number of bits on each level that as the number

[13] This certainly is not optimal and could be further improved.

of transmitted bits increases and more levels are used, the discrepancy between the randomised and non-randomised approach increases.

#### Possible Further Improvements

The tree based ordering contains randomisations that could be optimised, since these randomisations do not take into account the constraint length of the code. In particular at the lower levels of the tree, where the bits are reasonably close together, it could be beneficial to place the bits such that the decoding window "around" that bit overlaps the least with already placed bits. Ultimately, this might give rise to a "nested tree" approach, where the more numerous levels of the tree are ordered by a tree-based approach. However, it is unlikely that a deterministic version will outperform the randomised tree. Since randomisations are not a problem for the simulations in this thesis, this line of optimisation is not pursued further.

## 3.2 Correcting Erasures

Figures 3.2–3.8 show that a given transmission order does not necessarily reach rate $r_{\max} = 1$ even for high signal-to-noise ratios. Disregarding the channel effect of high SNR channels as minimal, the problem of reaching $r_{\max} \approx 1$ can be recast as the ability of a code to handle bit erasures. In this setting erasures are not caused by transmission over an erasing channel but by the puncturing of the to be transmitted bits. In this section this problem of correcting erasures is treated independently of the particular RSC code employed, therefore the results are not tied to one particular RSC code. Furthermore the simulation results in this section are of much reduced complexity compared to the simulation of the entire transmission system. It is thus possible to use the approach developed in this section to investigate higher order RSC codes, since, in particular for Turbo Codes, implementations are limited around a polynomial order of $n = 6$. This discussion will also provide an answer to the issue of performance difference when the parity or systematic bits are transmitted first, observed in section 3.1.5.

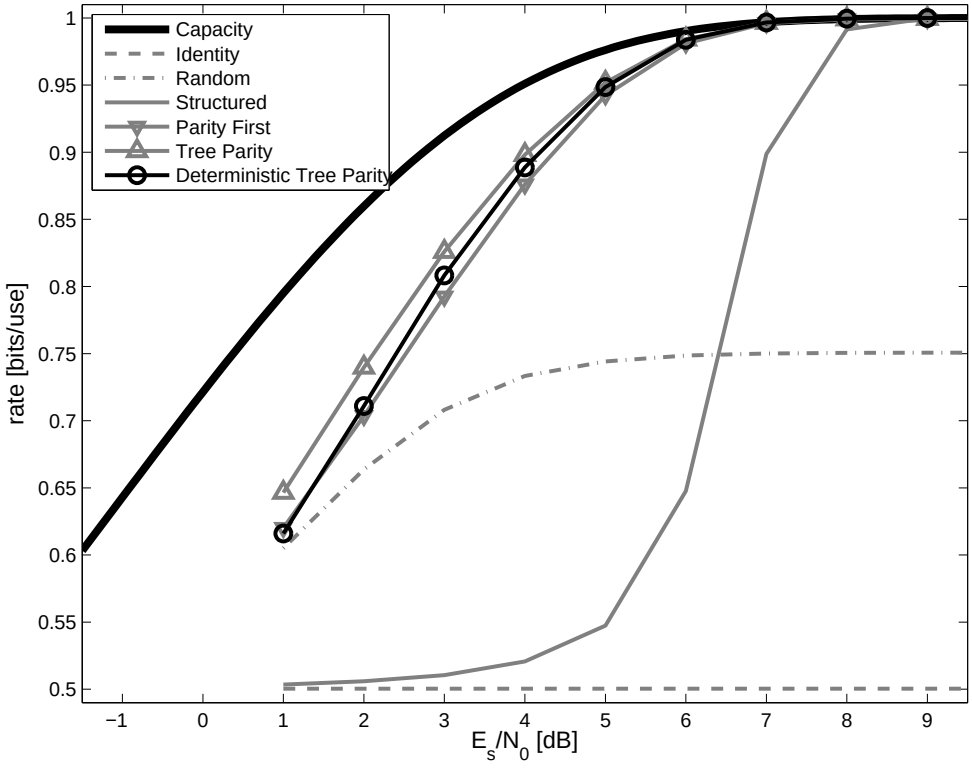



Figure 3.9: Deterministic tree based selection of bits (circles) compared to the randomised version (triangles pointing up) and the block-randomised code structure ordering (triangles pointing down). For simplicity, only the parity first ordering is shown as it has proven to perform best. Increasing the number of transmitted bits (lowering the rate) causes the deterministic and randomised version to move further apart. This is due to the increased number of bits on each level of the tree that are sent in order. Also shown for comparison are the random (dash-dot) ordering strategy, the structured approach without randomisation (solid), the structured approach with randomisation and the rate of the base code, $r_{\text{base}}$, indicated by the performance of the identity ordering (dashed).

The encoding process of an RSC code is a finite state machine (see sections 2.1.3 and 2.3.1). The decoding process observes the output of that state machine and tries to recover the state transitions and thus the input bits to the encoding process. The ability to recover punctured bits is equivalent to recovering a path in the trellis (and hence a sequence of states) from a given starting state, $s$, to a target state, $s'$, without observing the entire output of the encoding process between the start and target state, since the corresponding output can be (completely) punctured. The first considerations to be made therefore deal with the structure of the trellis.

**Lemma 3.1.** *An encoder of an RSC code of constraint length n can transition from any starting state s to any final state t with at most n transitions.*

*Proof.* The states are an enumeration of the different configurations of the memory cells in the shift registers of the RSC encoder. The contents of the last memory cell is discarded during a shift. It clearly follows that with $n$ shifts, the content of all memory cells is replaced. Since the input bit and hence the value of the bit that is shifted into the first memory cell is arbitrary, $n$ shifts lead to a final state depending only on the shifted-in values and not on the starting state. Thus with an appropriate sequence of input bits, it is possible to reach any state in at most $n$ transitions. □

Thus, in the trellis, every state can be reached starting from any other state, given a sufficient number of time steps. The structure of the trellis, however, can be narrowed down further.

**Lemma 3.2.** *A full trellis segment of length n of a binary input RSC code of constraint length n has exactly* $2^{2n}$ *unique paths.*

*Proof.* An RSC code of constraint length $n$ has $2^n$ states. Each of these states has exactly 2 transitions leaving that state corresponding to an input bit with a value of 0 and 1, respectively. Thus for each path leading to state $s$ with length $l$ there are exactly two paths of length $l+1$ following the 0 and 1 transition, respectively. Hence, for each starting state $s$, the number paths with a length of $k$ is $2^k$ and, accordingly, $2^n$ paths of length $n$. Since there are $2^n$ starting states, the total number of paths of length $n$ is $2^{2n}$. Since these states differ in either starting state or final state (which follows from lemma 3.2), the paths are unique. □

The ability to recover erasures is closely tied to the knowledge of the state of the encoder. Taking into account lemma 3.1 and 3.2, an upper limit can be placed on the maximum length of a continuous puncture, that the code is able to correct, given perfect information otherwise, i.e. knowledge of the state of the encoder before the puncture and after the puncture.

**Theorem 3.3.** *An RSC code of constraint length n can correct a gap of at most n transitions, provided that the start and final state of the encoder is known.*

*Proof.* An RSC encoder of constraint length $n$ has $2^n$ possible start states and $2^n$ possible final states. Thus there are $(2^n)^2 = 2^{2n}$ combinations of start and end state $(s, s')$. In a gap of $n$ transitions, there are exactly $2^{2n}$ possible paths. From the fact that the RSC encoder can transition from any state $s$ into any state $t$ with at most $n$ transitions, it follows that each path in the gap has to correspond to exactly one pair $(s, s')$. Thus if the pair $(s, s')$ is known, the path can be identified and the transitions and hence the input bits can be recovered. □

An RSC code of constraint length $n$ can thus obtain $n$ input bits from the knowledge of both the start state and the final state. A rate $1/2$ RSC code produces $2n$ bits for a trellis segment of length $n$, thus, given theorem 3.3, it is possible from these $2n$ bits to recover the start state $s$ and the final state $s'$.

**Lemma 3.4.** *For a binary input RSC code of constraint length n, the two branches that lead to any state $s'$ cannot produce the same encoder output.*

*Proof.* In order to lead to state $s'$, the two initial states $s_1$ and $s_2$ differ only in exactly the last memory cell of the encoder. Furthermore, the input bit that triggers the transitions $s_1 \rightarrow s'$ and $s_2 \rightarrow s'$, respectively needs to be the same due to the systematic nature of the RSC code. At least one generator polynomial needs to contain the term $D^n$ in order for it to be an order $n$ code. If the feedback polynomial $g^{(0)}$ contains this term, then $s_1$ and $s_2$ cannot transition to $s'$ with the same input bit. If any other generator polynomial, $g^{(i)}$ contains the term $D^n$, then the output $u^{(i)}$ will differ for the two transitions $s_1 \rightarrow s'$ and $s_2 \rightarrow s'$. □

In order to prove theorem 3.6, let the label of a path in the trellis be defined as the concatenation of the encoder output of all trellis branches that make up the path.

**Lemma 3.5.** *A rate $r = 1/2$ RSC code of constraint length n produces unique path labels for each path of length n.*

*Proof.* Suppose that for two distinct paths of length $n$, $p$ and $q$, the same path labels are produced. The paths cannot merge, since for length $n$ paths, every possible pairs of states has a single unique path that links them (lemma 3.5). The paths $p$ and $q$ cannot originate from the same state, since if $p$ and $q$ start from the same state, which follows from the presence of the systematic component of RSC codes. Lemma 3.4 states that $p$ and $q$ cannot enter the same state at any given time index, therefore $p$ and $q$ must be distinct paths in the trellis with separate starting states $s_1$, $s_2$ and final states $s'_1$, $s'_2$. Now, if the labels for $p$ and $q$ are the same, the states become indistinguishable. However, if the states are indistinguishable they can be merged into one state without ill effect. This however reduces the number of states which contradicts the fact that an order $n$ RSC code has $2^n$ states.[14] □

Given lemma 3.5, it is possible now to show that a rate $r = 1/2$ RSC encoder can recover state information from the input bits.

**Theorem 3.6.** *A rate $r = 1/2$ RSC code of constraint length n can recover the state information given the encoder output for n consecutive input bits.*

*Proof.* According to lemma 3.5, each path of length $n$ has a unique path label. A rate $r = 1/2$ code of order $n$ can enumerate $2^{2n}$ different paths. Since there are exactly $2^{2n}$ paths and each has a unique path label, knowledge of the path label, represented by the output of the encoder is thus sufficient to recover the transitions and states involved in producing the encoder output. □

The main effort of the decoder is to estimate a series of state transitions of the encoder. Theorems 3.3 and 3.6 illuminate how the decoder is able to trade state information for knowledge about transitions and how from parity information, i.e. a complete transition label, information about the state of the encoder can be recovered. When considering the correction of erasures, it makes sense to place two additional requirements on a rate $r = 1/2$ code, namely that the parity generating polynomial, $g^{(1)}$, contains the term $D^0$ and that *either* $g^{(0)}$ or $g^{(1)}$ contains the term $D^1$.

[14] Note that this happens e.g. when the generator polynomials if $\mathbf{GCD}\left(g^{(0)}, g^{(1)}\right) \neq 1$.

**Lemma 3.7.** *Every rate $r = 1/2$ RSC code, for which $g^{(1)}$ contains the term $D^0$, can be partitioned into two partitions, one where each state only produces transitions for which $s_i = c_{1,i}$ and one partition where each state produces transitions for which $s_i \neq c_{1,i}$. These partitions have equal size.*

*Proof.* The generator polynomial $g^{(1)}$ contains the term $D^0$. This requirement causes the code bit $c_{1,i}$ to depend on the state as well as on the input bit, i.e. it has a different value for $x = 0$ and $x = 1$. Since the only possible values for $c_{1,i}$ are 0 and 1, the code bit is either the same, or different from the systematic bit. Since the transition labels are equally distributed, the two partitions need to have the same size. □

The existence of these two partitions allows the decoder to determine a set of $2^{n-1}$ starting states from observing $u_t^{(0)}u_t^{(1)}$.

**Lemma 3.8.** *For every rate $r = 1/2$ RSC code, for which $g^{(1)}$ contains the term $D^0$ and* either *$g^{(0)}$ or $g^{(1)}$ contains the term $D^1$, the two transitions possible for each state do not connect to the same partition.*

*Proof.* On a state transition at time index $t$, the oldest bit is shifted out of the encoder memory and a new bit shifted into the encoder memory. This bit can have a value of either 0 or 1. When calculating the transition label of the next transition, $t+1$, this newly shifted-in bit is used by the term $D^1$ and is the only position in which the potential states at $t+1$ differ. Since $D^1$ occurs exactly once in the code, the two possible final states produce either a different parity bit (if $D^1$ occurs in $g^{(1)}$) or a different input into the shift register (if $D^1$ occurs in $g^{(0)}$) which then modifies the code bit because $g^{(1)}$ contains the term $D^0$. □

**Theorem 3.9.** *For a $1/2$ rate RSC code for which lemma 3.8 holds, an erasure at time index t can be corrected by observing the next state transition fully, i.e. observing $u_{t+1}^{(0)}u_{t+1}^{(1)}$.*

*Proof.* Consider the consequence of a code that does not obey lemma 3.8. If the transition at time index $t$ is punctured completely, the parity bit of the next transition (at $t+1$) *does not contain any useful information*, since in both states that the encoder can be in at $t+1$ can lead to a transition with the pattern $u_{t+1}^{(0)}u_{t+1}^{(1)}$. On the other hand, a

code that follows lemma 3.8 is able to use the parity bit to prune one of the two possible paths and recover the punctured bit. □

Theorem 3.9 uses a *known state before* the beginning of the erasure plus *known bits after* the erasure to recover from the erasure. Similarly, it is possible to reverse the look on the encoder in the previous lemmas and thus show that a decoder can correct an erasure with *known bits before* and a *known state after* the erasure, if both generators contain the term $D^n$ and *either* polynomial contains the term $D^{n-1}$. All optimised polynomials that the author has encountered (see e.g. section 4.6) fulfil these requirements. Namely, the terms $D^0$ and $D^n$ are always contained in *both* generator polynomials and the terms $D^1$ and $D^{n-1}$ are always contained in *only one* of the generator polynomials.

The aim of the decoder is to reconstruct the state sequence of the encoder in order to determine the input bits. Ideally, $s_{\mathbf{x}}$ bits are sufficient (e.g. the systematic bits), which readily follows from the structure of the RSC code. Thus, once the state sequence to a given time index $t$ is known, any additional bits, systematic or parity, in the time interval $[0..t]$ are not beneficial to the recovery of erasures, since no additional knowledge can be gained about the state sequence (an improvement in confidence is disregard due to the high channel SNR). Put constructively, useful bits to the decoding process are those that are not within a known state sequence or within a recoverable gap.

Theorem 3.3 showed how a decoder of a convolutional code can use state information to recover information about the input to the encoder. Likewise, theorem 3.6 showed how the reverse process can take place, i.e. how parity information can be used to recover the state of the encoder and finally theorem 3.9 showed how an erasure can be immediately corrected. These three theorems together form the basis for an evaluation of the ordering strategies, presented next.

### 3.2.1 State and Transition Certainty

The position of useful bits in the correcting process can be approximated by considering the number of potential states that the observed encoding process can be in. Let $\tilde{n}_{\text{states}}(t)$ be the number of possible states at time index $t$ that a rate $r = 1/2$ RSC encoder of constraint length $n$ can be in. Observing the transition $t \rightarrow t+1$

$$\overrightarrow{\tilde{n}_{\text{states}}}(t+1) = \begin{cases} \max\left(1, \overrightarrow{\tilde{n}_{\text{states}}}(t)/2\right) & \text{if systematic } \textit{and} \text{ parity bit are known} \\ \overrightarrow{\tilde{n}_{\text{states}}}(t) & \text{if systematic bit } \textit{or} \text{ parity bit is known} \\ \min\left(\overrightarrow{\tilde{n}_{\text{states}}}(t) \cdot 2, 2^n\right) & \text{if neither bit is known} \end{cases}, \tag{3.5}$$

with $\overrightarrow{\tilde{n}_{\text{states}}}(0) = 1$. Similarly let $\overleftarrow{\tilde{n}_{\text{states}}}(t)$ be the number of possible states having observed the backward transitions, $t \dashleftarrow s_{\mathbf{x}}$, for an input block of size $s_{\mathbf{x}}$ such that for the backward transition $t \leftarrow (t+1)$.

$$\overleftarrow{\tilde{n}_{\text{states}}}(t) = \begin{cases} \max\left(1, \overleftarrow{\tilde{n}_{\text{states}}}(t+1)/2\right) & \text{if systematic } \textit{and} \text{ parity bit are known} \\ \overleftarrow{\tilde{n}_{\text{states}}}(t+1) & \text{if systematic bit } \textit{or} \text{ parity bit is known} \\ \min\left(\overleftarrow{\tilde{n}_{\text{states}}}(t+1) \cdot 2, 2^n\right) & \text{if neither bit is known} \end{cases}. \tag{3.6}$$

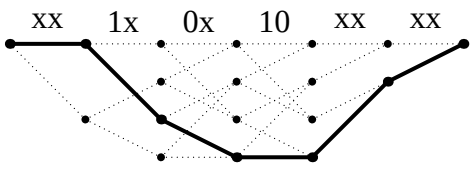
with $\overleftarrow{\tilde{n}_{\text{states}}}(0) = 1$ if the encoder terminates in a defined state and $\overleftarrow{\tilde{n}_{\text{states}}}(0) = 2^n$ if the encoder does not return to a predefined state.

It is immediately obvious that if $\tilde{n}_{\text{states}}(t) = 1$, knowing both the systematic and code bit of the transition $t \rightarrow (t+1)$ wastes one bit, since *either systematic or code bit is sufficient* to maintain $\tilde{n}_{\text{states}}(t+1) = 1$.

Equations (3.5) and (3.6) both define a measure of uncertainty about the current state of the encoder. Clearly, only the forward or backward iteration over the trellis need to lead to the knowledge of the state and, ideally, it is possible that the information from the forward and backward iteration complement each other to allow for reconstructing the state of the encoder. The certainty, $c_s(t)$, about the current state $s$ that the encoder is in at time index $t$ is thus defined as

$$c_s(t) = \min\left(2n - \log_2\left(\overrightarrow{\tilde{n}_{\text{states}}}(t)\right) - \log_2\left(\overleftarrow{\tilde{n}_{\text{states}}}(t)\right), n\right). \tag{3.7}$$

Equation (3.7) calculates the state and transition certainty strictly within the interval $[0, n]$. The reason for this is that the state certainty depends on the number of possible




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Forward certainty ($\overrightarrow{c_s}$): | 2 | 1 | 1 | 1 | 2 | 1 | 0 |
| Backward certainty ($\overleftarrow{c_s}$): | 0 | 1 | 1 | 1 | 0 | 1 | 2 |
| State certainty ($c_s$): | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

Figure 3.10: Example run of algorithm 3.1. For ease of exposition a concrete RSC code with memory $n = 2$, shown in Figure 3.12 is used to show the encoding trellis and the encoder outputs. A punctured bit is marked by $\times$, otherwise the proper encoded bit is shown. Shown are the intermediate calculations for the forward ($\overrightarrow{c_s}(t)$) and backward ($\overleftarrow{c_s}(t)$) certainty used in the algorithm as well as the resulting state certainty $c_s(t)$ according to equation (3.7).

states that the observed encoder can be in. Thus, the number of states cannot exceed $2^n$ (for which $c_s = 0$) nor fall below 1 ($c_s = n$). As an aside, this includes the fact that knowledge of the states can be redundant, if $\overrightarrow{\tilde{n}_{\text{states}}} + \overleftarrow{\tilde{n}_{\text{states}}} < 2^n$, e.g. if the state can be reconstructed from both the forward *and* the backward iteration.

The definition of the transition certainty $c_b(t \to t+1)$ for the transition $t \to t+1$ follows from the state certainty as

$$c_b(t \to t+1) = \min\left(c_s(t), c_s(t+1)\right). \tag{3.8}$$

A certainty value of $c_b = n$ indicates that the bit can be decoded, a certainty value of $c_b = 0$ indicates that nothing is known about the transition.

For this thesis, algorithm 3.1 was developed in order to compute the branch certainty (equation (3.8)) for a block of bits. It is similar to the BCJR algorithm[5] in concept, requiring a forward and backward recursion over the trellis of the encoder, similar to the $\alpha$ and $\beta$ calculations in the BCJR algorithm. An example of algorithm 3.1 can be seen in Figure 3.10.

**Algorithm 3.1** Transition Certainty Calculation.

{Forward Iteration}
$\overrightarrow{c_s}(0) = n$
**for** $t = 1$ to $s_{\mathbf{x}}$ **do**
  **if** systematic bit and parity bit are both available **then**
    $\overrightarrow{c_s}(t) = \min(\overrightarrow{c_s}(t-1)+1, n)$
  **else if** only the systematic bit or the parity bit is available **then**
    $\overrightarrow{c_s}(t) = \overrightarrow{c_s}(t-1)$
  **else**
    $\overrightarrow{c_s}(t) = \min(\overrightarrow{c_s}(t-1)-1, 0)$
  **end if**
**end for**
{Backward Iteration}
**if** Encoder terminates in defined state **then**
  $\overleftarrow{c_s}(s_{\mathbf{x}}+1) = n$
**else**
  $\overleftarrow{c_s}(s_{\mathbf{x}}+1) = 0$
**end if**
**for** $t = s_{\mathbf{x}} - 1$ down to 0 **do**
  **if** systematic bit and parity bit are both available **then**
    $\overleftarrow{c_s}(t) = \min(\overleftarrow{c_s}(t+1)+1, n)$
  **else if** only the systematic bit or the parity bit is available **then**
    $\overleftarrow{c_s}(t) = \overleftarrow{c_s}(t-1)$
  **else**
    $\overleftarrow{c_s}(t) = \min(\overleftarrow{c_s}(t+1)-1, 0)$
  **end if**
**end for**
{State Certainty Calculation}
**for** $t = 0$ to $s_{\mathbf{x}}$ **do**
  $c_s(t) = \min(\overrightarrow{c_s}(t) + \overleftarrow{c_s}(t), n)$
**end for**
{Transition Certainty Calculation}
**for** $t = 0$ to $s_{\mathbf{x}} - 1$ **do**
  $c_b(t \rightarrow t+1) = \min(c_s(t), c_s(t+1))$
**end for**

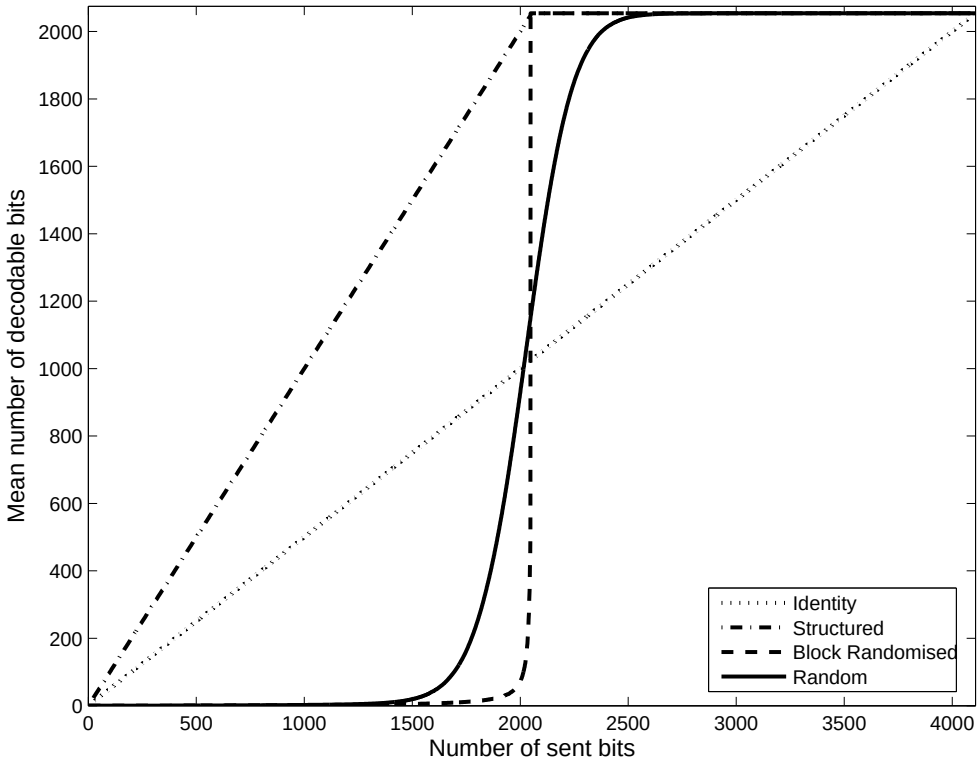



Figure 3.11: Number of decodable bits of transmission order construction strategies (averaged over 10000 instances for structured and random): Identity (dotted), Structured without intra block randomisation (dash-dot), Structured with intra block randomisation (dashed) and Random ordering (solid). The tree ordering performs almost identically to the block randomised approach and is omitted for clarity of presentation.

### 3.2.2 Evaluation of Existing Ordering Strategies

Algorithm 3.1 provides an evaluation of a transmission order which is independent of any particular input bit pattern. This makes it possible to evaluate the reordering strategies described in section 3.1 for a large number of possible instantiations of the strategy without needing to pay the computational cost of the BCJR algorithm. The transition certainty provides a way of determining if a given input bit, $x$ is decodable. If $c_b(t) = n$, then it follows from equation (3.8) that $c_s(t) = n$ and $c_s(t+1) = n$ and hence the states at either end of the transition are known. The input bit which triggered the transition can hence be recovered.

**Definition 3.10** (Decodable Bit)**.** *For a rate $r = 1/2$ RSC code with constraint length $n$, an input bit $x$ at time index $t$ is defined as decodable, if $c_b(t) = n$.*

Figure 3.11 shows the number of decodable (according to definition 3.10) bits calculated using algorithm 3.1. The identity ordering strategy transmits all code bits for a given input bit before proceeding to the next input bit. This is why the number of decodable bits increases only with a slope of the rate, $r$, in this case $r = 1/2$. The structured approach, on the other hand, achieves a slope of 1 due to sending the systematic bits first. The behaviour of the block-randomised and tree-based[15] ordering strategies clearly show the effect of randomisation. Bits are placed without necessarily forming coherent blocks. Once the number of sent bits approaches $s_{\mathbf{x}}$, the received bits start to link. Hence, a single bit can increase the transition certainty for a large number of bits due to it being the single bit that bridges the last gap. Thus, a significant increase is seen, once the first packet of $s_{\mathbf{x}}$ is sent. The random ordering strategy, unsurprisingly, also shows this randomisation effect. However, since the random ordering strategy takes bits from the entire code sequence and does not limit itself to particular types of bits, an earlier increase in the number of decodable bits is seen, since occasionally, parity and systematic bit for a given input bit is available, allowing the correction of adjacent erasures. However, the increase in the number of decodable bits flattens out once $s_{\mathbf{x}}$ bits have been transmitted. Choosing bits at random, the ordering strategy is unable to select those bits that are required to recover all bits and instead selects bits that do not contribute to the still unknown bits. Additional bits are required to obtain $s_{\mathbf{x}}$ decodable bits.

The data obtained with algorithm 3.1 for the random ordering strategy suggests that the maximum rate, determined by the average rate at which the number of decodable bits reaches $s_{\mathbf{x}}$, of $r_{\max} = 0.8235$. However, as shown in Figure 3.3, the random ordering strategy only achieves a maximum rate of $r_{\max} = 0.7505$. The reason for this discrepancy is that some puncturing patterns, depending on the code, can lead to ambiguous, irreducible paths which are the topic of the next section.

[15] The tree-based ordering strategy is omitted from Figure 3.11 for clarity.

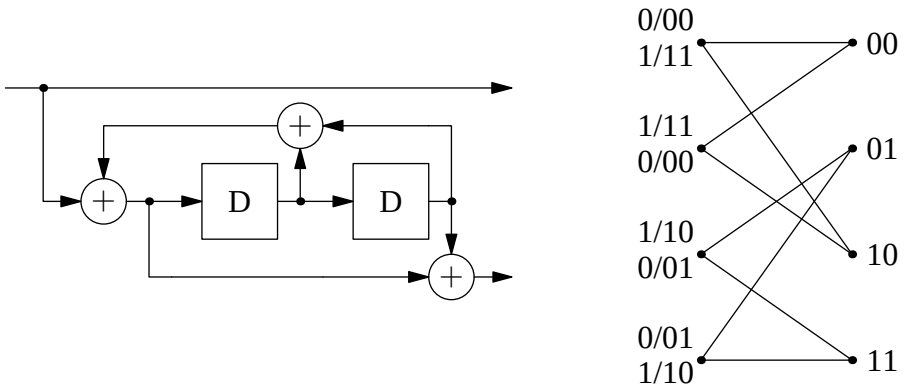



Figure 3.12: RSC encoder with $g^{(0)} = 1 + D + D^2$, $g^{(1)} = 1 + D^2$ and associated state transitions (input bit/output bits). State labels (on the right) correspond to memory cell assignments.

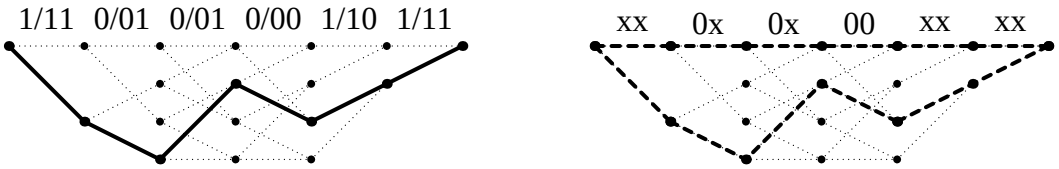



Figure 3.13: Trellis for 100011 input of $\left(1 + D + D^2, 1 + D^2\right)$ RSC encoder. Displayed transitions are known (solid line) or possible (dashed line).

### 3.2.3 Irreducible Paths

The definition of $\tilde{n}_{\text{states}}(t+1)$ in section 3.2.1 states that if both the systematic bit and the code bit are known, then $\tilde{n}_{\text{states}}(t+1) = \tilde{n}_{\text{states}}(t)/2$. However, puncturing patterns exist, for which this is not true. Consider the RSC encoder shown in Figure 3.12. The trellis for the input sequence 100011, resulting in the output 110101001011, is shown in Figure 3.13 on the left. Now consider the puncturing pattern $\times \times s_2 \times s_3 s_4 c_{4,1} \times \times \times \times$ to obtain a rate 1 code.[16] This leads to the observed sequence shown in Figure 3.13 on the right. Given this puncturing pattern, the original bit sequence cannot be recovered, since two paths in the trellis exist that produce the observed pattern. Of particular interest is the transition $3 \rightarrow 4$. Given the observed bits, the encoder is either in state 0 or in state 1. Both states are in the same partition of the trellis, i.e. they share the same output labels, in this case the parity bit is the same as the systematic bit. Thus

[16] The last two input bits are tail bits that are added by the encoder to make it return to the 0 state. Since they are not data bits, they can be punctured without losing information.

knowledge of the parity bit *fails to reduce the number of possible paths*, rendering knowledge of the parity bit useless. Algorithm 3.1, on the other hand, works under the assumption that this parity bit reduces the number of possible paths and thus predicts that all bits are recoverable, making the output of algorithm 3.1 an upper bound for the maximum rate $r_{\max}$. There is no possibility of addressing the problem of irreducible paths to remedy this problem without losing codeword independence. A pessimistic solution is to limit confidence increases to those places that occur right before and after a complete puncture of both systematic and parity bit, leading to a lower bound, since under some circumstances, it is possible to recover complete erasures even when the additional, required bit is not directly adjacent to the puncture.

### 3.2.4 Unequal Siblings – The Difference Between Code Bits

Figure 3.13 showed a puncturing pattern that is fulfilled by 2 paths in the trellis. Note that for transitions $1 \rightarrow 2$ and $2 \rightarrow 3$, the parity bits were punctured. Trying to replace the systematic bits with the respective parity bits, i.e. puncturing with the pattern $\times \times \times c_{2,1} \times c_{3,1} s_4 c_{4,1} \times \times \times \times$ allows a unique sequence of states to be recovered. The two bits by themselves, i.e. if the other bit is punctured, thus have a different effect on the decoder. Consider Figure 3.14 which shows all possible paths of length 3 in a trellis for an order $n = 2$ RSC code (the encoder for this particular trellis is shown in Figure 3.12). The number of possible paths is $n_{\text{paths}} = 2^2 \cdot 2^3 = 32$, hence 8 per starting state, or 2 for each pair of starting and final state. Considering the RSC encoder shown in Figure 3.12 and the puncturing pattern $\times \times s_2 \times \times \times$. The knowledge of $s_2$ reduces the number of paths to 1 per state pair. E.g. for a starting state of 0 and $s_2 = 0$ 3.14.1, 3.14.3, 3.14.6 and 3.14.8, for $s_2 = 1$ 3.14.2, 3.14.4, 3.14.5 and 3.14.7. Now consider the puncturing pattern $\times \times \times c_{2,1} \times \times$. There are still 4 possible paths per starting state when fixing $c_{2,1}$. Again, starting from 0 these paths are 3.14.1, 3.14.3, 3.14.5 and 3.14.7 for $c_{2,1} = 0$ and 3.14.2, 3.14.4, 3.14.6 and 3.14.8 for $c_{2,1} = 1$, respectively. Notice that the pairs 3.14.5, 3.14.7 and 3.14.6, 3.14.8 switch places from $s_2 = 0$ to $c_{2,1} = 1$ and $s_2 = 1$ to $c_{2,1} = 0$, respectively. The consequence of this is that for half the state pairs there are two possible paths, for the other half no possible path exists in the trellis that satisfies $c_{2,1}$.

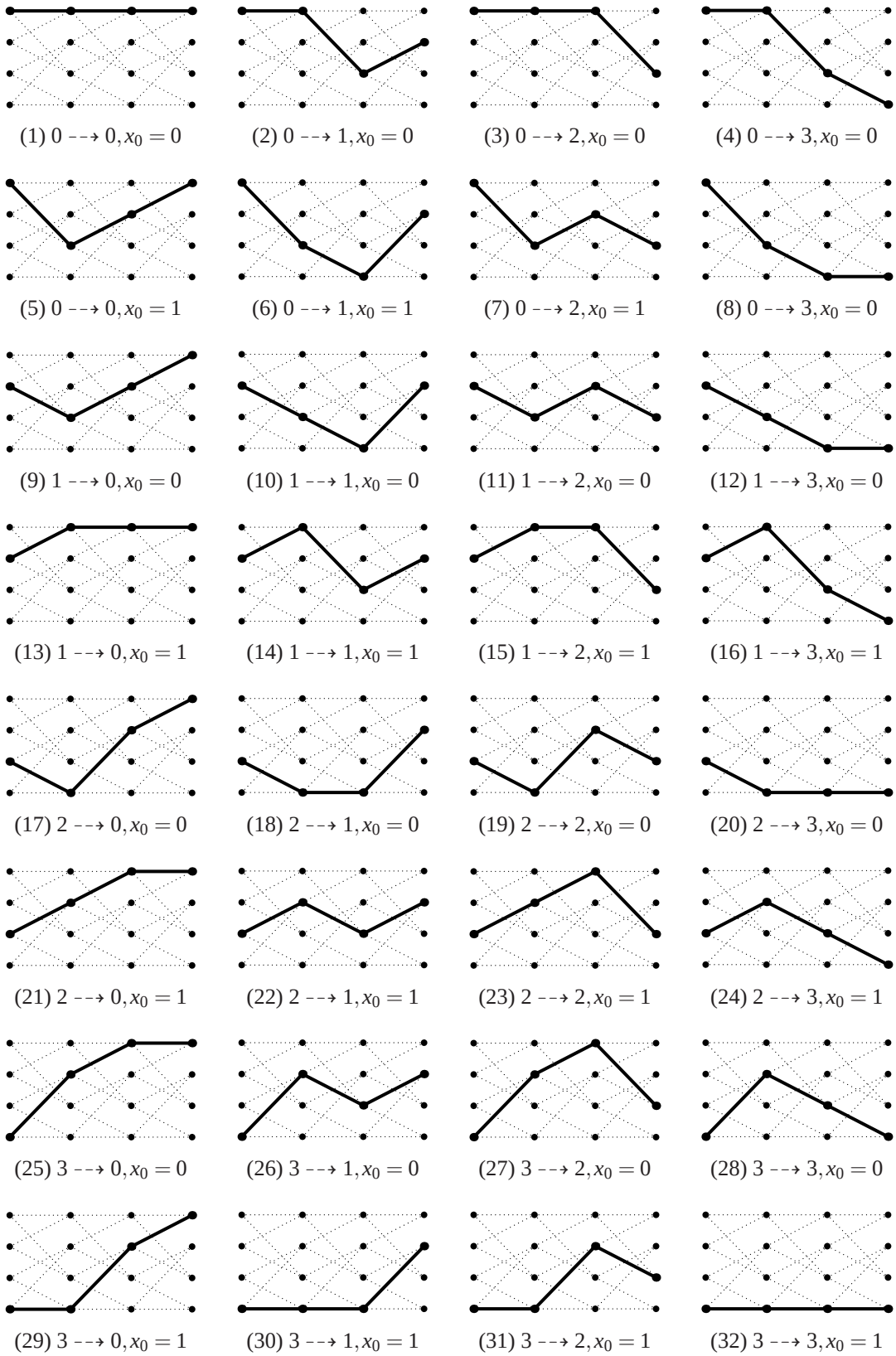



Figure 3.14: Trellis paths of length 3 for the $\left(1+D+D^2, 1+D^2\right)$ RSC code of constraint length 2 shown in Figure 3.12.

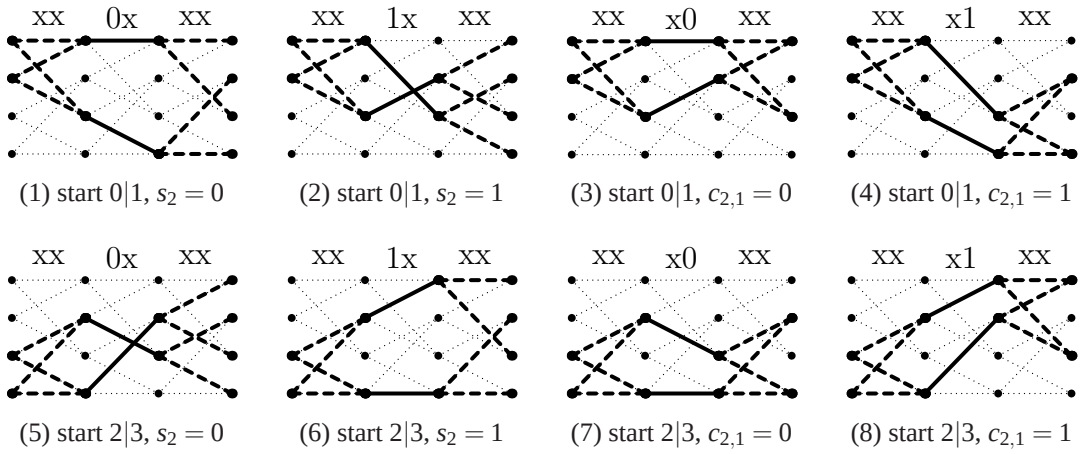



Figure 3.15: Different paths of length 3 in the trellis for the $\left(1+D+D^2, 1+D^2\right)$ RSC code of constraint length 2 shown in Figure 3.12. The paths are constraint by the knowledge of the systematic bit, $s_2$ or the parity bit $c_{2,1}$, respectively. For the code from Figure 3.12, pairs of start and final state exist for which two paths satisfy $c_{2,1}$ whereas other pairs cannot satisfy $c_{2,1}$.

Consider Figure 3.15 which summarises the different paths that can satisfy $s_2$ and $c_{2,1}$, respectively. The difference between the two code bits is clearly visible. While the systematic bit keeps the state in separate partitions of the graph (see lemma 3.7), the parity bit causes the two possible states to *converge into one partition*.

The difference between the two bits is not inherent in the distinction between systematic and parity bits. Exchanging the generator polynomials of the encoder shown in Figure 3.12 results in the encoder shown in Figure 3.16, with a different pattern of state transitions, also shown in Figure 3.16. This different pattern of state transitions leads to a reversed situation as far as the constraints of the systematic and parity bits are concerned. Consider Figure 3.17 which summarises the possible paths constraint by $s_2$ and $c_{2,1}$, respectively, for the $\left(1+D^2, 1+D+D^2\right)$ RSC code from Figure 3.16. In Figure 3.15 the knowledge of the systematic bit could be satisfied by any pair of start and final state. In Figure 3.17, on the other hand, it is the *parity bit* that can be satisfied by any pair of start and final state.

Once the encoder can be in two state of the same partition, lemma 3.7 states that the possible transition labels will be the same for these states. This is a significant problem when trying to correct erasures or correcting errors. Because the only possible states that the encoder can be in are in the same partition, knowledge of both code bits *does*

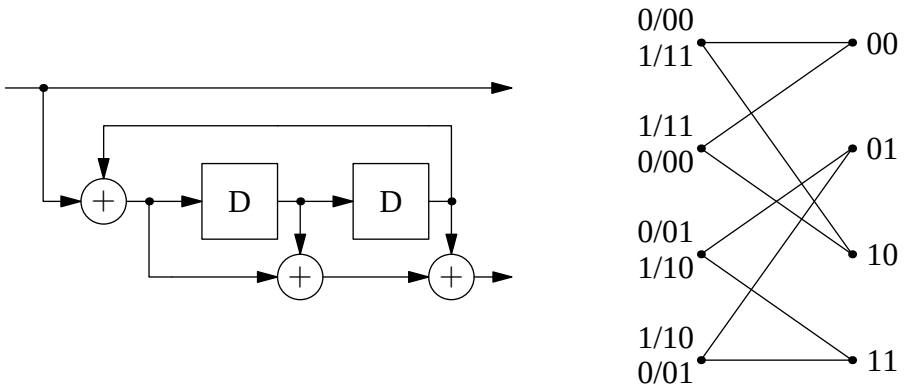



Figure 3.16: RSC encoder with $g^{(0)} = 1+D^2$, $g^{(1)} = 1+D+D^2$ and associated state transitions (input bit/output bits). State labels (on the right) correspond to memory cell assignments.

*not provide any additional knowledge gain*. Referring to Figure 3.15, if the decoder observes $c_{2,1} = 0$ for the $\left(1+D+D^2, 1+D^2\right)$ RSC code and $c_{3,1}$, then it immediately knows $s_3$ if the starting state is known. Similarly, knowledge of $s_3$ and the starting state implies knowledge of $c_{3,1}$. On the other hand, if $s_2$ is observed, knowing both $s_3$ and $c_{3,1}$ is beneficial to the decoder, since it can exclude paths in the trellis.

In order to explain the performance difference between selecting the parity or systematic bits for first transmission, observed in section 3.1 (Figure 3.6), consider the effect of transmitting the entire sequence of parity and systematic bits, respectively for the $\left(1+D+D^2, 1+D^2\right)$ RSC code. If all systematic bits are transmitted first, the error correcting ability of the code is achieved by *additional parity bits*. Following the previous discussion, parity bits up to 2 time steps away[17] are able to aid in the error correction. On the other hand, if all parity bits are transmitted, *additional systematic bits* provide the error correction. The knowledge of the parity bits in this case, however, already contains information about the systematic bits. Therefore not all systematic bits within 2 time steps are useful for error correction (refer to Figure 3.15). Transmission of systematic bits first is thus expected to outperform transmission of parity bits first. Conversely, for the $\left(1+D^2, 1+D+D^2\right)$ RSC code, it can be expected that transmission of parity bits first outperforms transmission of systematic bits first. It can be seen from Figure 3.18 that this expected behaviour is indeed the case.

[17] In general, for a code with constraint length $n$, the parity bits are most effective if maximally $n$ time steps away. Thus in this case the best parity bits are adjacent.

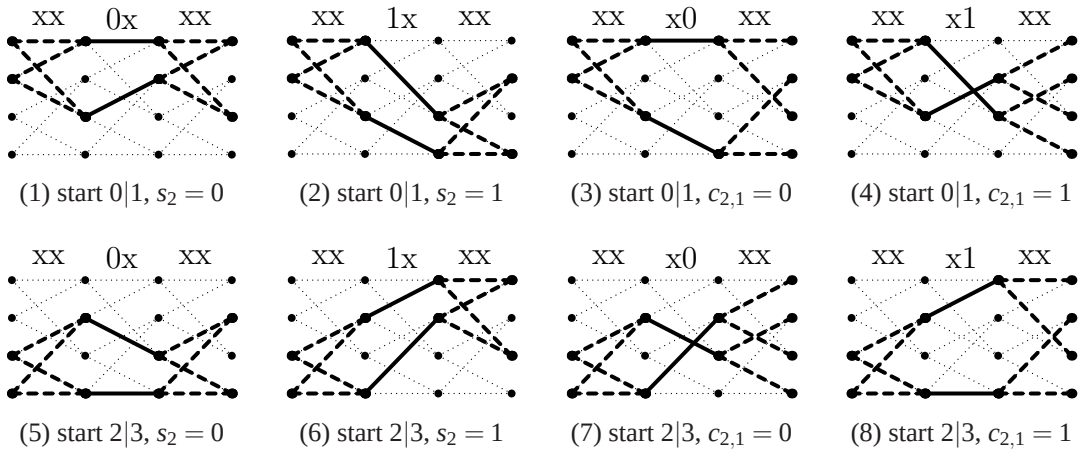
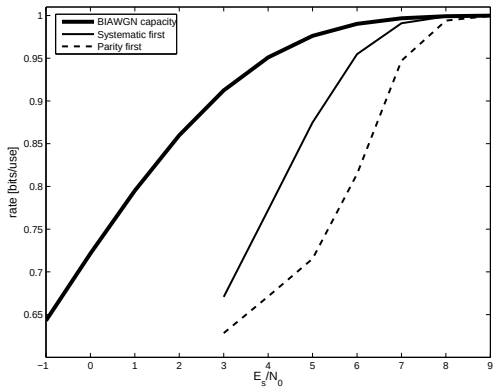


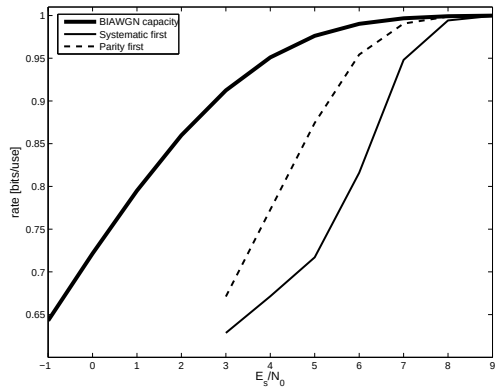

Figure 3.17: Different paths of length 3 in the trellis for the $\left(1+D^2, 1+D+D^2\right)$ RSC code of constraint length 2 shown in Figure 3.12. The paths are constraint by the knowledge of the systematic bit, $s_2$ or the parity bit $c_{2,1}$, respectively. For the code from Figure 3.16, contrary to the $\left(1+D^2, 1+D+D^2\right)$ RSC code, pairs of start and final state exist for which two paths satisfy $s_2$ whereas other pairs cannot satisfy $s_2$. The role of the systematic and code bit is thus reversed.

In Figure 3.18.1, which displays the performance of the $\left(1+D+D^2, 1+D^2\right)$ code, transmitting the systematic bits first produces a superior performance. The situation is reversed in Figure 3.18.2, which displays the $\left(1+D^2, 1+D+D^2\right)$ code. Using this switched code, transmitting the parity bits first is the superior ordering strategy.

The superiority of either transmitting the systematic or the parity bits first depends on the code and thus has an interesting consequence for Turbo Codes. Section 4.2 will show that transmitting the systematic bits first is always superior for Turbo Codes, even though the constituent code that is used is the code from section 3.1 that performs superior if parity bits are transmitted first. Considering Figure 3.18, it is possible to select the superiority of systematic or parity bits simply by switching the generator polynomials without incurring a performance loss to obtain a new code $\left(1, \frac{g'(1)}{g'(1)}\right)$ with $g'(0) = g^{(1)}$ and $g'(1) = g^{(0)}$. Figures 3.19, 3.20 and 3.21 illustrate this process for the generator polynomial pairs proposed by Benedetto *et al.*[8], Berrou *et al.*[12] and Perez *et al.*[93], respectively. Shown for each generator pair are the state transition diagram and the consequence of selecting for transmission the systematic or the parity bit first, exemplified by showing the existence of 2 separate paths that cannot be distinguished by the respective decoder. Finally, the performance of the transmission order of systematic bits first and parity bits first are shown for the proposed genera-




(1) $\left(1, \frac{1+D^2}{1+D+D^2}\right)$ code.




(2) $\left(1, \frac{1+D+D^2}{1+D^2}\right)$ code.

Figure 3.18: Performance difference of transmitting the systematic or parity bits first. Figure (1) shows the effect using the $\left(1, \frac{1+D^2}{1+D+D^2}\right)$ code whereas the $\left(1, \frac{1+D+D^2}{1+D^2}\right)$ code is shown in Figure (2). Switching the generator polynomials switches the superiority of systematic or parity bits of the two codes.

tor pair and the reverse pair, i.e. the $\left(1, g^{(0)}/g^{(1)}\right)$ code. It can be readily observed from Figures 3.19, 3.20 and 3.21 that interchanging the assignment of generator polynomials, i.e. turning the $\left(1, g^{(1)}/g^{(0)}\right)$ into the $\left(1, g^{(0)}/g^{(1)}\right)$ code does indeed cause a switch of the superiority of systematic or parity transmission without incurring a performance loss. In terms of Turbo Codes, this result is of particular interest. In chapter 4 it is shown that transmitting systematic bits first is necessary for turbo codes to achieve $r_{\max} \approx 1$. Thus it is beneficial, if the constituent RSC codes, which *both* get supplied with the systematic bits work best with the systematic bits transmitted first. When determining suitable polynomials, the steps outlined in this section make it possible to exclude half of the polynomials simply by determining the superiority of parity or systematic transmission, potentially saving resources (e.g. simulation time). In line with this reasoning, it is no coincidence that the generator polynomials selected by Benedetto *et al.*[8] and Perez *et al.*[93], which are optimised for Turbo Codes, perform best when the systematic bits are transmitted first, as can be observed in Figures 3.19 and 3.21.

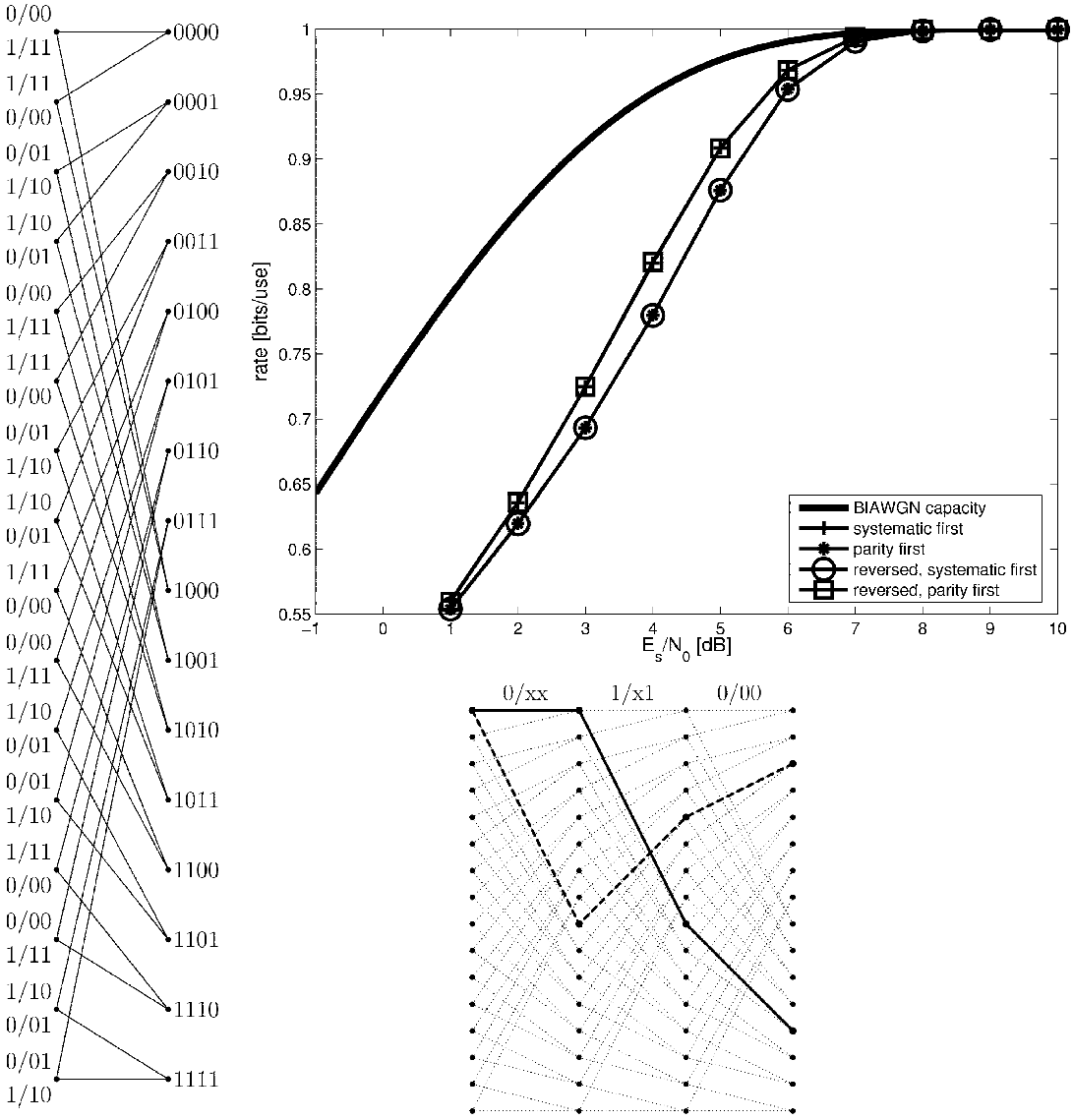



Figure 3.19: Performance of the $\left(1, \frac{1+D^1+D^3+D^4}{1+D^1+D^4}\right)$ code proposed by Benedetto *et al.* [9]. On the left the state transition diagram is shown with the transition labels of the states. On top the performance of the block-randomised ordering strategy with transmission of parity and systematic bits first, respectively, alongside the results for the reversed code $\left(1, \frac{1+D^1+D^4}{1+D^1+D^3+D^4}\right)$, clearly showing that the superiority of systematic-first or parity-first transmission depends on the assignment of the polynomials to $g^{(0)}$ and $g^{(1)}$. The problem of transmitting the parity bit first is demonstrated on the bottom. With the given puncturing pattern the decoder is unable to distinguish the shown, partial paths, since both states reached at $t = 2$ (0100 and 1000) contain the transition labels 00 and 11.

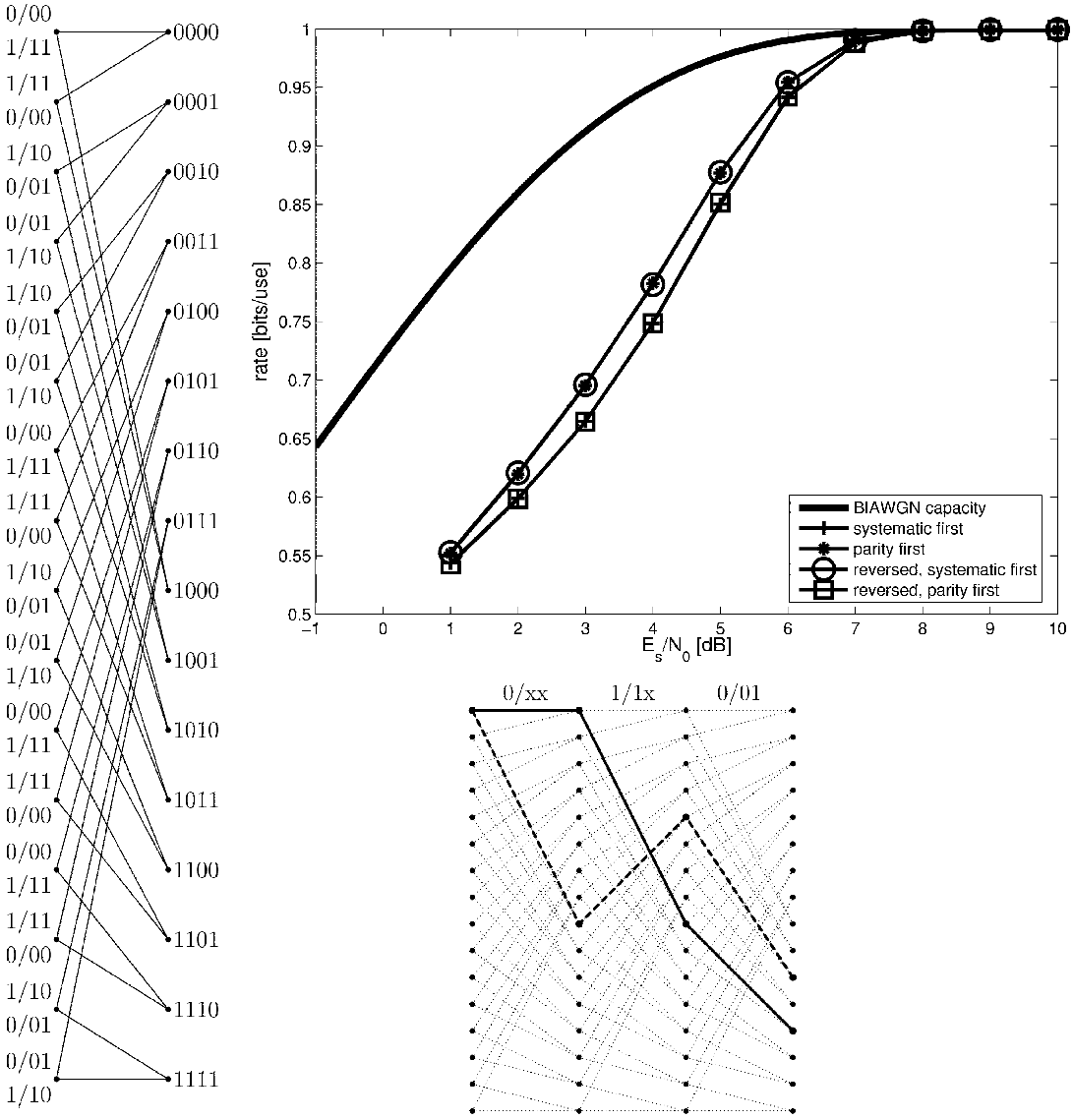



Figure 3.20: Performance of the $\left(1, \frac{1+D^4}{1+D^1+D^2+D^3+D^4}\right)$ code proposed by Benedetto *et al.* [12]. On the left the state transition diagram is shown with the transition labels of the states. On top the performance of the block-randomised ordering strategy with transmission of parity and systematic bits first, respectively, alongside the results for the reversed code $\left(1, \frac{1+D^1+D^2+D^3+D^4}{1+D^4}\right)$, clearly showing that the superiority of systematic-first or parity-first transmission depends on the assignment of the polynomials to $g^{(0)}$ and $g^{(1)}$. The problem of transmitting the systematic bit first is demonstrated on the bottom. With the given puncturing pattern the decoder is unable to distinguish the shown, partial paths, since both states reached at $t = 2$ (0100 and 1000) contain the transition labels 01 and 10.

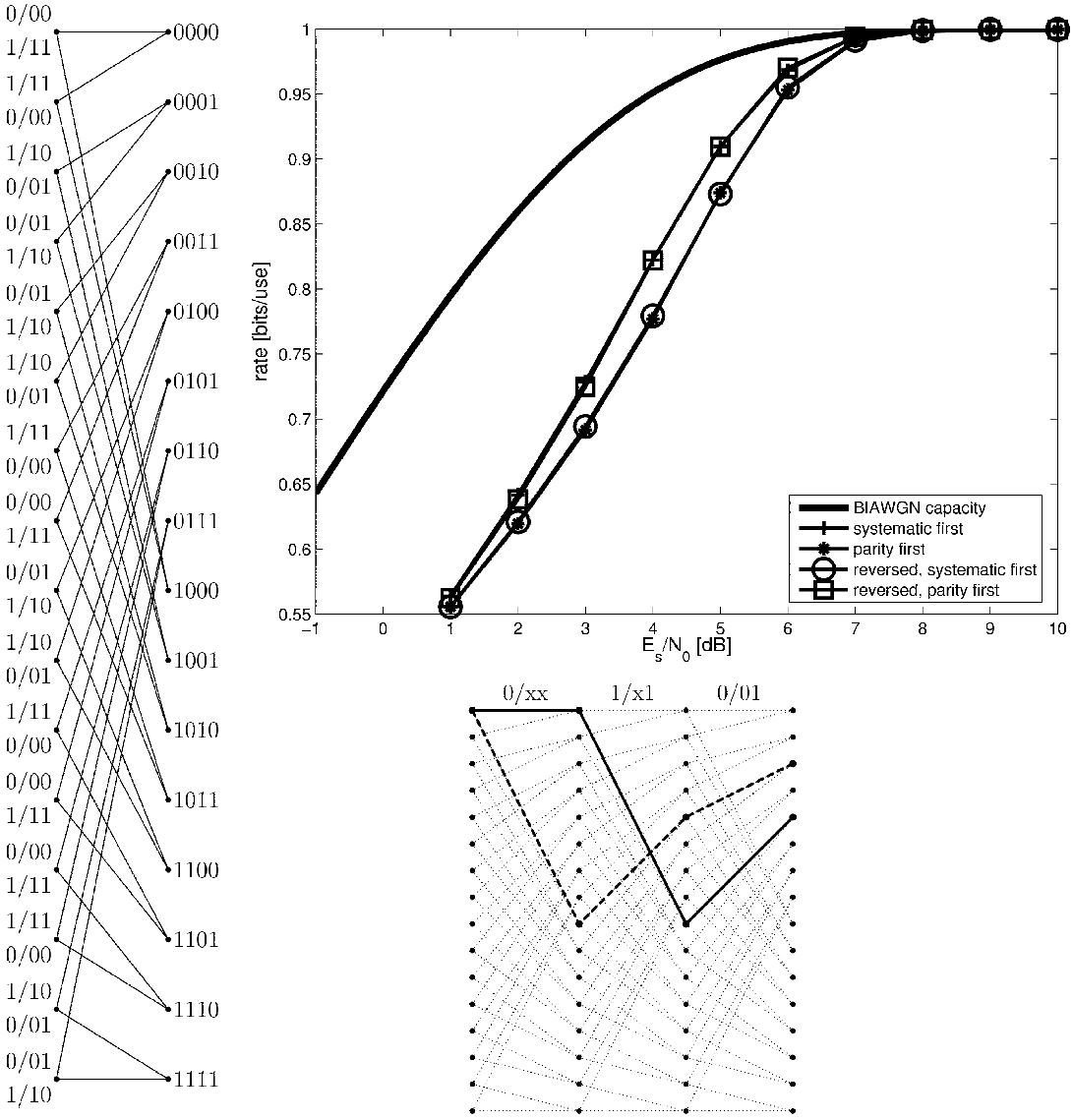
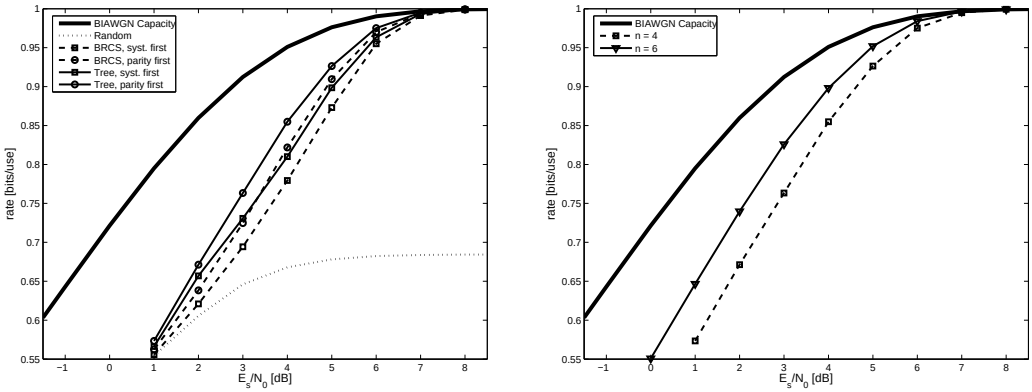



Figure 3.21: Performance of the $\left(1, \frac{1+D^1+D^2+D^4}{1+D^3+D^4}\right)$ code proposed by Benedetto *et al.* [93]. On the left the state transition diagram is shown with the transition labels of the states. On top the performance of the block-randomised ordering strategy with transmission of parity and systematic bits first, respectively, alongside the results for the reversed code $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}\right)$, clearly showing that the superiority of systematic-first or parity-first transmission depends on the assignment of the polynomials to $g^{(0)}$ and $g^{(1)}$. The problem of transmitting the parity bit first is demonstrated on the bottom. With the given puncturing pattern the decoder is unable to distinguish the shown, partial paths, since both states reached at $t = 2$ (0100 and 1000) contain the transition labels 01 and 10.




(1) Ordering strategy comparison

(2) Comparison of $n = 4$ and $n = 6$ codes.

Figure 3.22: Simulation results of the $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}, 2048\right)$ RSC code. Figure (1) shows the performance of the different ordering strategies. No qualitative degradation of the ordering strategies can be observed, although a performance penalty due to the reduced memory of the encoder is present. Figure (2) compares the binary tree based transmission ordering strategy with transmission of parity bits first of the $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}, 2048\right)$ and $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$ codes with a clearly observable performance degradation.

## 3.3 Blocksize and Constraint Length

The transmission ordering strategy developed in section 3.1 was evaluated in the context of the $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$ code of memory $n = 6$. This section investigates the blocksize and constraint length of the RSC code, showing that, while the qualitative behaviour of the transmission strategies does not change, the overall performance is dependent on the memory of the encoder and on the chosen blocksize.

### 3.3.1 Constraint Length

It is well known that the performance of a RSC code depends on the memory of the encoder. As a comparison to the memory $n = 6$ code used in section 3.1, all possible combinations of two generator polynomials of memory $n = 4$ were enumerated. From these, the polynomial was picked that achieved the lowest BER over 100000 packets of size $s_{\mathbf{x}} = 2048$ for an SNR of 2 dB. The surviving polynomial combination, $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}, 2048\right)$ is used as the memory $n = 4$ code.

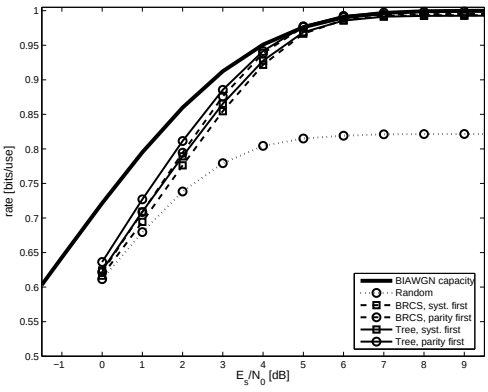
Figure 3.22.1 shows the simulation results of the ordering strategies for the RSC code with memory $n = 4$. The performance of the ordering strategies with respect to each other has not changed. The tree based strategy still outperforms the block-randomised strategy and parity transmission before systematic bits are transmitted is still superior. The comparison of the tree based transmission order with parity first transmission is shown in Figure 3.22.2. With decreasing channel quality, measured by a falling SNR, the code with memory $n = 4$ performs increasingly worse than the memory $n = 6$ code. This is expected, since the main criterion of the performance of a convolutional code is the memory size of the encoder [75].

### 3.3.2 Input Blocksize

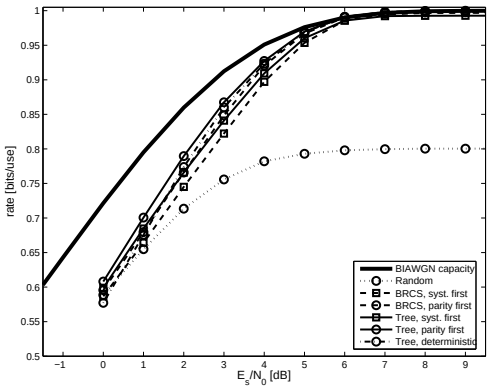
An RSC code depends on its memory to achieve low error probabilities and not on the input blocksize [75]. Conversely, the performance of an RSC code with fixed memory $n$ should not depend (much) on the blocksize. Nevertheless, it is worth investigating, if the performance of the *ordering strategy* depends on the input blocksize. For example, a large blocksize might somewhat mitigate the advantage of the tree based ordering strategy once the deepest level of the tree is reached.

Consider the performance of the $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$ code with memory $n = 6$ given different input blocksizes, shown in Figure 3.23. Fundamentally, the performance of the ordering strategies has not changed, that is the relative performance order for the individual ordering strategies is maintained. However, a close inspection of Figure 3.23 reveals that as the blocksize increases, the performance of the ordering strategies *decreases across the board*.

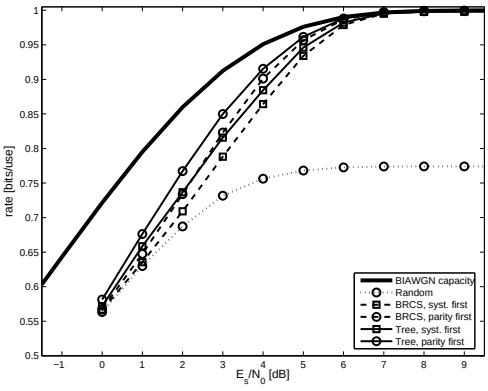
It is a well known weakness of the ARQ protocol, that long blocksizes are detrimental to the performance of ARQ. Since longer blocks increase the probability that *at least one error* will occur, longer blocks directly lead to more bits being retransmitted. In other words, a short blocksize allows an ARQ system to narrow down the occurrence of the error to the short block, whereas for long blocks, the single error could be anywhere in the block. A transmission system with variable rate faces a similar problem. This time the penalty is not time wasted through retransmissions, but transmission of additional parity bits. In an ideal code, every code bit $u$ contains an equal amount of information about *each* input bit $x$. A bit ordering in that case would be unnecessary.



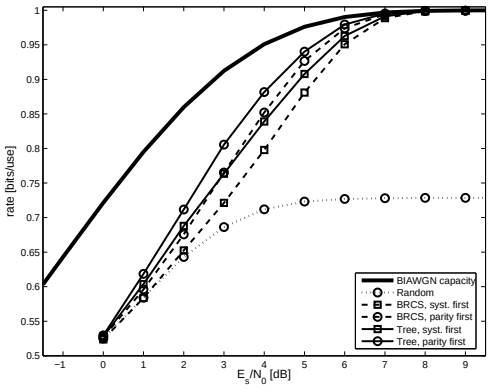

(1) $s_{\mathbf{x}} = 256$ bits



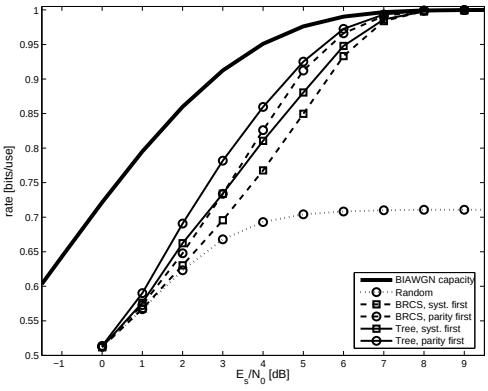

(2) $s_{\mathbf{x}} = 512$ bits



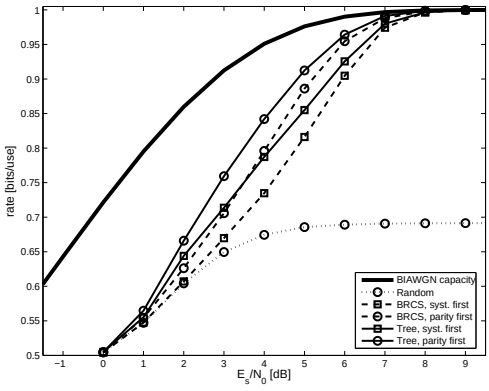

(3) $s_{\mathbf{x}} = 1024$ bits




(4) $s_{\mathbf{x}} = 4096$ bits




(5) $s_{\mathbf{x}} = 8192$ bits



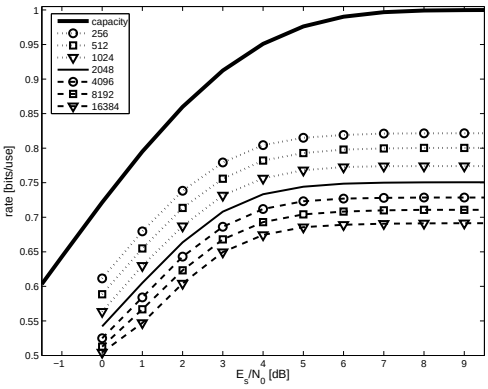

(6) $s_{\mathbf{x}} = 16384$ bits

Figure 3.23: Performance of the $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}\right)$ code for different input sizes. Notice that with increasing blocksize, $s_{\mathbf{x}}$, the performance of all employed ordering strategies decreases. 1000 packets were simulated for each input size.

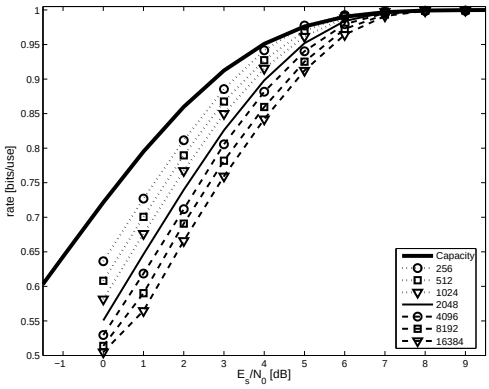
This however is not the case for convolutional codes, due to the decoding window of the code. For short blocksizes, it is easy to pick few bits that will "cover" the entire length of the block. However, the larger the blocksize, the larger the chance of picking a bit that cannot improve the reliability of the bit in error sufficiently.

There is, however, a second consideration to be made. The discussion thus far did not focus on the overhead that might be required per packet. The, not unreasonable, assumption made so far was that the input data contains the required bits to perform error detection or that some other form is available at the consumer. For evaluating the performance of different ordering strategies given the same blocksize, this consideration was not important, since the overhead applies equally to all ordering strategies. Now that the discussion has turned to varying blocksizes, the argument of overhead needs to be made. Consider Figure 3.24, which summarises the performance of the random and the tree based ordering strategies for different input blocksizes. It also shows the effect of reserving 16 bits of the input for use as a CRC sum. Even for this moderate requirement of only 16 bits, the performance of the small blocksizes drops significantly at high SNR (Figure 3.24.4). Therefore at high SNR, large blocksizes are preferred, since the overhead caused by the error detection codes is the significant one. Once the SNR drops, however, the parity bits required to maintain error free transmission become more numerous in comparison to the error detection bits. This effect can also be seen when comparing the random ordering strategy. Because the random ordering strategy is non-optimal for high SNR, the presence of an overhead caused by error detection is negligible, and shorter blocksizes are preferable.

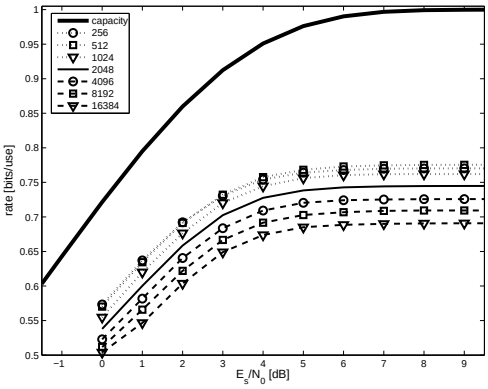
Thirdly, the residual error probability needs to be taken into account. In the type-II hybrid ARQ system used for the simulations, code combining was not performed. Likewise, discarding packets that exhaust the number of available parity bits was also not considered, since it has no impact on the transmission order. However, a better localisation of errors by limiting the block size comes at the cost of a greater "outage" of packets, i.e. packets which are subjected to an error pattern that cannot be corrected by the channel code. A code with large blocksize is better suited to correct these "unfortunate" error patterns where errors accumulate on few bits. Figure 3.25 shows the residual BER of the different blocksizes at 0 dB and 1 dB. At 0 dB the code is about to break down and bit- errors occur at a similar rate for all investigated blocksizes. Considering the same channel at 1 dB, however, shows that small blocksizes not only have a higher chance of transmitting packets that need little protection through parity



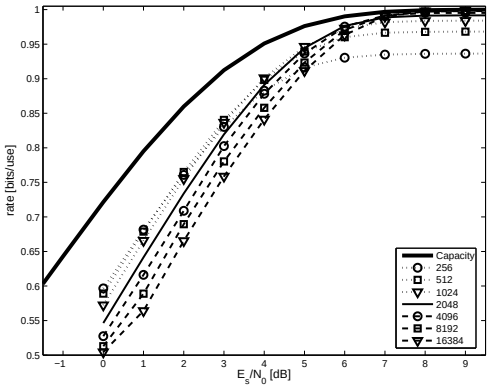

(1) Random, no overhead




(2) Tree, parity first, no overhead




(3) Random, 16 bits overhead




(4) Tree, parity first, 16 bits overhead

Figure 3.24: Performance of the $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}\right)$ code for the random and the tree based strategy with transmission of parity bits first, considering 0 and 16 bits overhead. For high SNR, the effect of overhead is significant, leading to an advantage for large blocksizes. However, for lower SNR, the presence of parity bits somewhat mitigates the impact of the overhead, since the additional parity bits cause the main performance degradation and small blocksizes are preferred. 1000 packets were simulated for each input size.

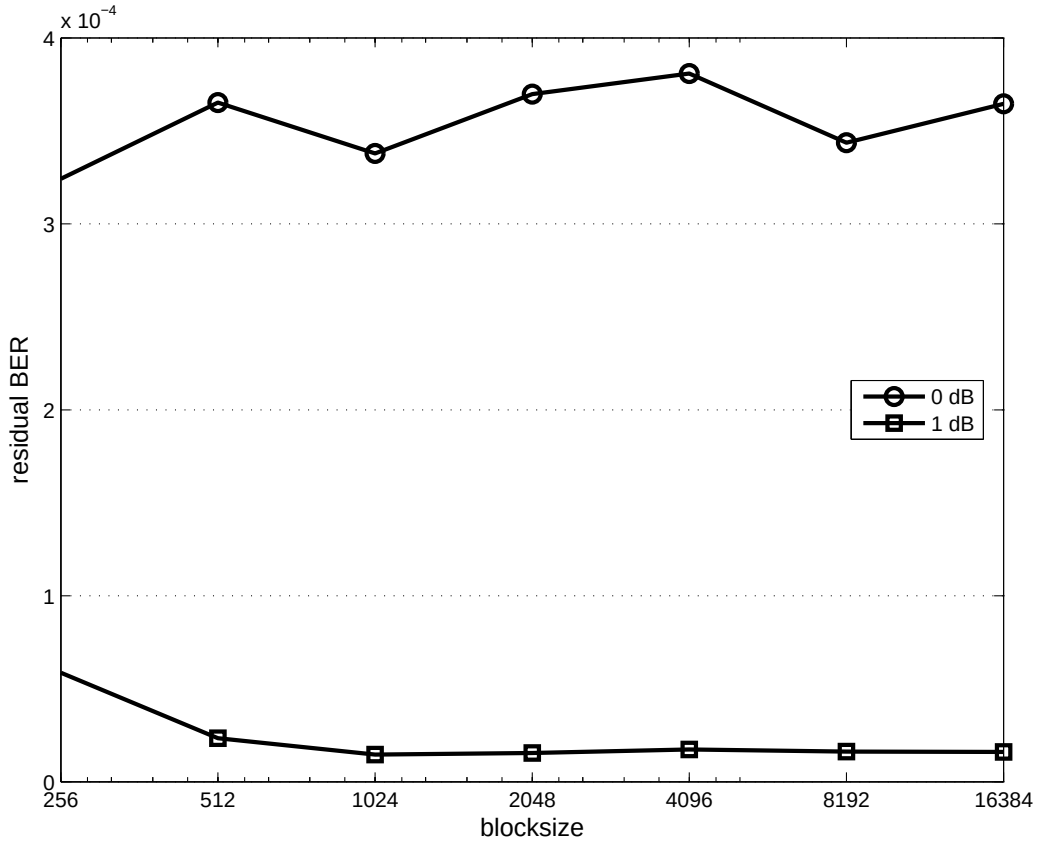



Figure 3.25: Residual BER of the tree tree based approach for different blocksizes. At 0 dB (circles) the residual BER is fairly constant, indicating immanent failure of the code. At 1 dB (squares) the residual BER is significantly higher for shorter blocklength, indicating that their performance advantage is, at least in part, due to allowing more errors to be passed to the application.

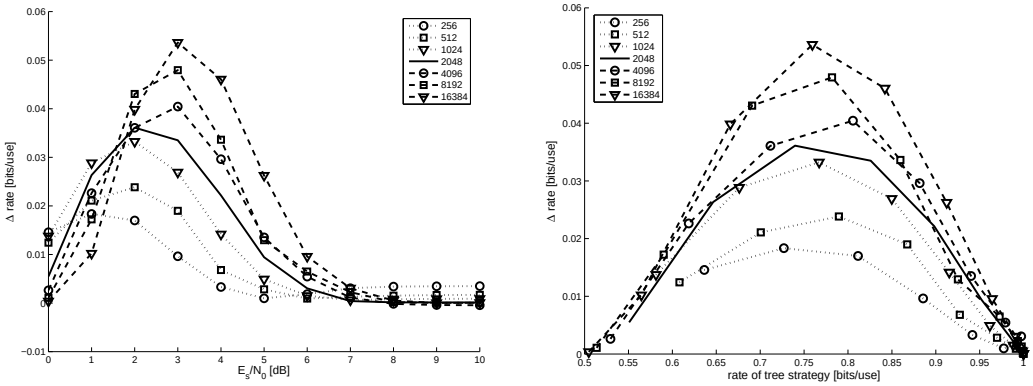
bits, there is also a higher probability that packets experience an *uncorrectable* error pattern, thus requiring additional retransmission.

This result contradicts the observation of improvements due to shorter blocksizes only somewhat. Consider the performance of the larger blocksizes in Figure 3.24.2 and the associated errors at 1 dB in Figure 3.25. Even though the performance decreases considerably, the residual bit-error rate remains reasonably constant. Therefore, while *very short* blocks have disadvantages with regard to "outage" of parity bits and uncorrectable errors, as soon as the blocksize becomes reasonably large, say more than 1000 bits, increasing the blocksize further reduces the rate performance without noticeable improvement of the residual bit-error rate.

#### The Effect of Blocksize on the Tree Based Ordering Strategy

Earlier in this section it was outlined, that the relative order of the transmission ordering strategies does not change given a changing blocksize. What does change, however, is the relative efficiency. Consider the tree based ordering strategy with the transmission of parity bits first. The advantage of the tree based approach is its ability to select bits appropriately for a large blocksize. However, the deepest level of the tree contains as many bits as the rest of the tree. Therefore one would expect that the advantage of the tree ordering strategy diminishes as this last level is tapped into. A slight benefit might still exist, but essentially, at this level, the tree based approach randomises the position of one quarter the bits of the encoded sequence, $\mathbf{u}$, the block-randomised approach, on the other hand, randomises the position of half of the bits, only twice as many as the tree (assuming a $1/2$ rate code). With an increasing blocksize, the deepest level becomes larger, thus narrowing the distinction between the two strategies.

Evaluating the rate difference between the tree-based approach and the block randomised approach seems to suggest that indeed, the tree-based approach performs worse with an increasing blocksize. Consider Figure 3.26.1 which shows the difference in rate between the two approaches over the range of previously investigated SNR. At low SNR, the performance advantage of the larger blocksizes (dashed) drops below that of the smaller blocksizes (dotted). However, at the given SNR values, the rate of the smaller blocksizes is significantly larger, resulting in fewer bits that were transmitted. Figure 3.26.2 takes these differences into account and plots the difference of the

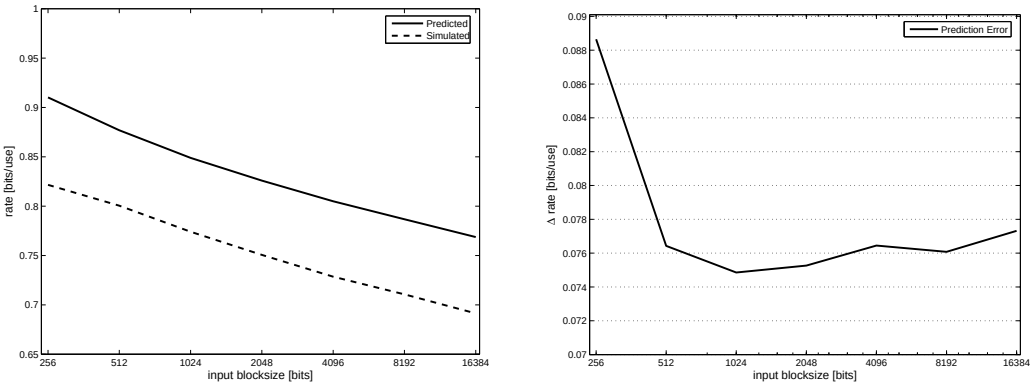



(1) Comparison based on $\frac{E_S}{N_0}$ (2) Comparison based on rate

Figure 3.26: Performance difference between the tree based approach and block-randomised approach. While Figure (1) shows that at low SNR the tree based approach loses ground, this is due to the overall worse performance of larger blocksizes at low SNR as Figure (2) shows.

two approaches with respect to the rate of the tree-based one. The picture is now completely different. It shows that, regardless of the blocksize, the tree-based approach is always superior to block randomisation. Figure 3.26.2 shows a peak at $r \approx 0.75$. The reason for this can be found in the constraints on the ordering schemes. For $r >> 0.75$ few additional bits are sent, whereas for $r << 0.75$ few additional bits are *left out*. It is obvious that as the rate approaches $r \approx 0.5$, all ordering schemes perform identically, since nearly all available bits have been transmitted. Equally obvious is the fact that all ordering schemes that cover each input bit at least once in the first $s_{\mathbf{x}}$ bits perform nearly identically for $Rate \approx 1$, since only very few additional bits need to be transmitted. The superiority of any ordering scheme thus has to manifest in between. This observation alone makes it likely that the greatest difference is at $r \approx 0.75$, since $r = 0.75$ is exactly the midpoint between these two extremes. In addition at $r \approx 0.75$ the tree ordering, having transmitted roughly half of the available parity bits, has not started the last tree level. This structural constraint of the tree ordering leads to an equal, an arguably optimal, spacing of parity bits at $r \approx 0.75$, contributing further to the performance advantage. [18]

[18] Interestingly, this is the rate the random ordering strategy fails to surpass. In both cases randomness is the reason for this, but the author believes that they are not causally related.




(1) Predicted and observed performance (2) Prediction error

Figure 3.27: Prediction of the performance of the random ordering strategy with increasing blocksize. Figure (1) displays the performance predicted by algorithm 3.1 using only the ordering permutation, $\prod$, (solid line) and the maximum rate, $r_{\max}$, observed in simulations of the entire transmission system (dashed line). Figure (2) shows the performance error.

### Performance Prediction of the Random Ordering Strategy

Algorithm 3.1, presented in section 3.2.1, can be used to predict the performance of the different ordering strategies. Of particular interest is the random ordering strategy in this context. Notice that in Figure 3.24.1, in addition to a decreasing performance at low SNR, the performance *at high SNR decreases with an increasing blocksize*. Consider Figure 3.27.1 which compares the performance predicted by algorithm 3.1 with the performance achieved in simulations. It was outlined in section 3.2.1, that the performance of algorithm 3.1 only provides an upper bound on the achievable rate. This point is observed for all blocksizes. However, the performance *loss* caused by the increasing blocksize ($\approx 0.022$ for doubling the blocksize, $s_{\mathbf{x}}$) is accurately predicted by the algorithm (see Figure 3.27.2). This leads to the conclusion, that the performance drop experienced by the random ordering strategy is entirely due to the poor properties of randomisation. Furthermore, it appears that the prediction error of algorithm 3.1, caused by the presence of irreducible puncturing patterns, is *a fixed offset* of $\approx 0.08$.

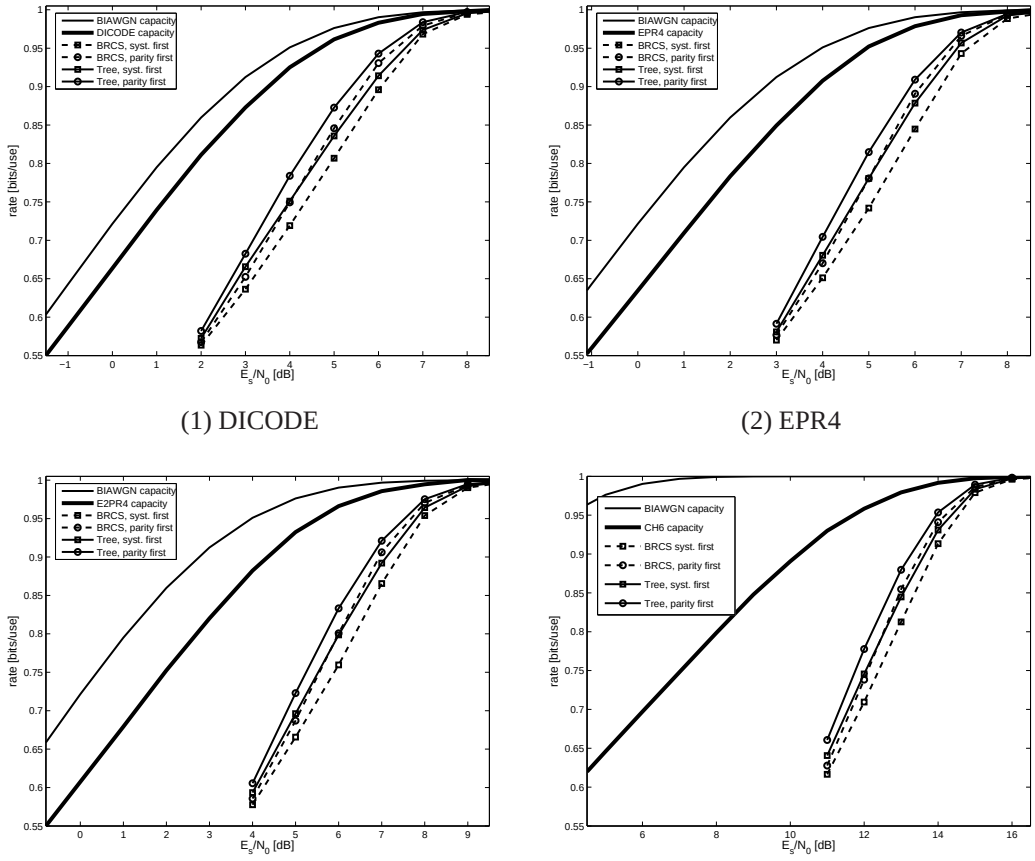



Figure 3.28: Comparison of selected ISI channels (the channel coefficients can be found in table 3.2). The block randomised code structure strategy (dashed, see section 3.1.4) is compared to the tree-based strategy (solid, see section 3.1.6 for transmitting systematic bits first (square markers) and parity bits first (round markers). The respective channel capacity is shown in bold together with the BIAWGN capacity (solid without marker) for reference. The code used is the $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$ RSC code with memory $n = 6$ from section 3.1, however only 1000 packets were simulated and not 10000. For all four simulated channels, the results are consistent with the qualitative results from section 3.1 in that transmission of the parity bits first outperforms transmission of the systematic first. It is also worth noting that even the cross-over of the tree strategy with systematic bits first and the BRCS strategy with parity bits first is retained, indicating that ISI transmission channels do not affect the selection of an ordering strategy.

## 3.4 Inter-Symbol Interference Channels

The results obtained in sections 3.1 and 3.3 were based on the AWGN channel. This section presents the results obtained for a simulation of the memory $n = 6$ RSC code $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$, used in section 3.1, and the memory $n = 4$ RSC code $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}, 2048\right)$, used in section 3.3.1, for transmission over inter-symbol interference channels. Figures 3.28 and 3.29 compare the block-randomised code structure transmission ordering with the binary tree based ordering strategy, each with transmitting either the parity or the systematic bits first. Only 1000 packets were simulated compared to the 10000 simulated in section 3.1. Comparing the first 1000 simulated packets to all 10000 simulated packets for the simulations performed in section 3.1 showed little no no difference in the average performance. Thus the number of simulated packets was reduced to speed up the simulation process while still remaining statistically practicable. The channels that were used for comparison are the well-known and widely used DICODE, EPR4, E2PR4 and CH6 channels, the coefficients of which are shown in table 3.2. It is assumed that the receiver has perfect knowledge of these channel parameters and performs MAP equalisation (see section 2.2.2) before passing the data to the decoder. No other method of ISI mitigation is used. Equalisation is essential for effective communication to take place over these channels. If equalisation is omitted, the RSC codes are unable to recover the transmitted data without errors, making a discussion about ordering schemes pointless. The advantage by using these channels is that Arnold and Loeliger[3] demonstrated an algorithm to calculate the capacity of these channels, which is used as a measure of the effectiveness of the ordering strategies. Both the tree based ordering strategy and the block-randomised code structure ordering strategy qualitatively perform similar to their respective performance for the AWGN channel. The best performing transmission ordering strategy remains the binary tree based selection with transmission of the complete parity information before the systematic information is transmitted. Even the partial superiority of BRCS with parity first over the tree based approach with transmission of systematic bits first, a minor detail, is retained. It can therefore be concluded that inter-symbol interference on the transmission channel does not affect the choice of the transmission ordering strategy. Figure 3.30 shows the performance of the selected ISI channels. As is expected, the performance degrades with an increase of the severity of the inter-symbol interference observed on the channel. This degradation is stronger than the difference between

| Channel Name | Normalised Impulse Response |
|---|---|
| DICODE | $a(D) = \left(1 - D^1\right)/\sqrt{2}$ |
| EPR4 | $a(D) = \left(1 + D^1 + D^2 + D^3\right)/2$ |
| E2PR4 | $a(D) = \left(1 + 2D^1 - 2D^3 - D^4\right)/\sqrt{10}$ |
| CH6 | $a(D) = 0.19 + 0.35D^1 + 0.46D^2 + 0.5D^3$ $+ 0.46D^4 + 0.35D^5 + 0.19D^6$ |

Table 3.2: Impulse responses of selected channels investigated by Arnold and Loeliger[3].

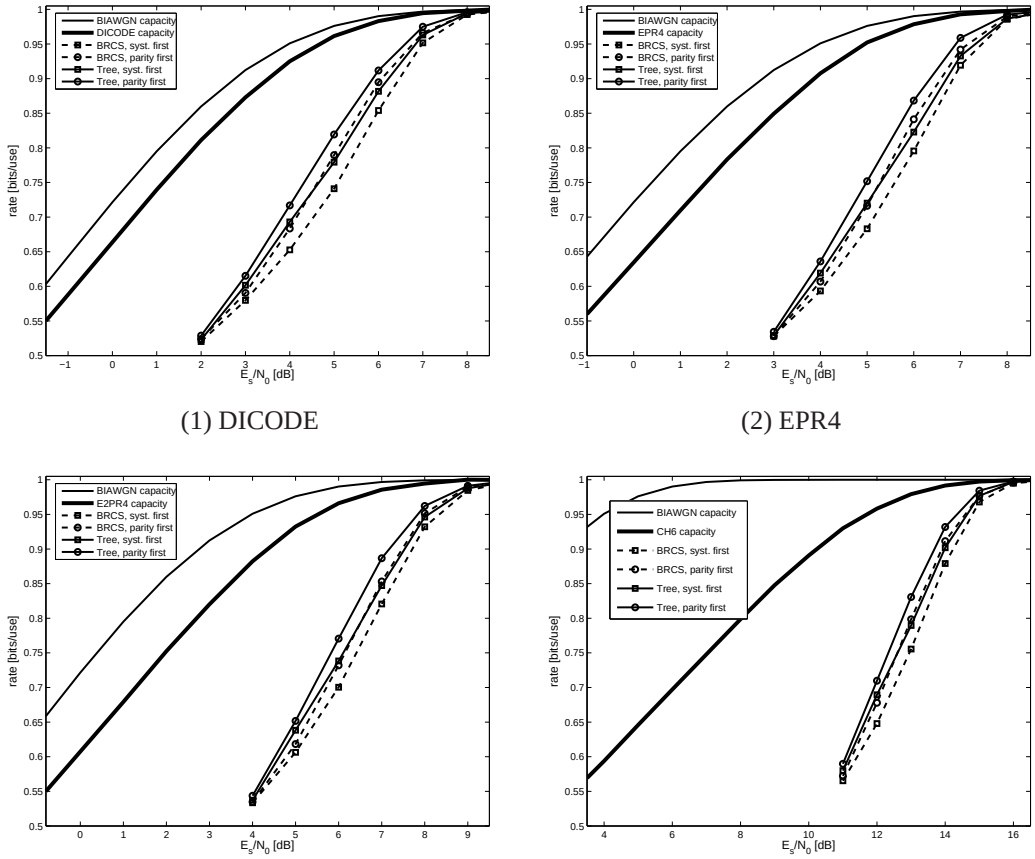
the calculated capacity of the ISI channels and the BIAWGN channel. However, as can be observed from Figure 3.30, the general performance characteristics and the relative superiority among the different ordering schemes is unaffected. This suggests that the ordering schemes presented in section 3.1 are also suitable for use on ISI channels.

## 3.5 Rayleigh Channels

The Rayleigh channel, presented in section 2.2.3, is a popular channel model for wireless transmission, where no direct line-of-sight component is available. It is an important model in the context of type-II hybrid ARQ systems, since it is most likely that channel coding is used in wireless transmission systems. Therefore the performance of the ordering strategies under the Rayleigh channel is investigated in this section.

The Rayleigh channel requires the knowledge of the Rayleigh fading factor, $a$, in order to convert the received signal level into LLR values. However, in the case of a fast-fading Rayleigh channel, where a different $a$ is simulated for every input bit, knowledge of this factor is impossible. Nevertheless, it is possible, in simulations, to assume knowledge of the fading factor, $a$, in order to show the performance penalty that ignorance of the fading factor bears.

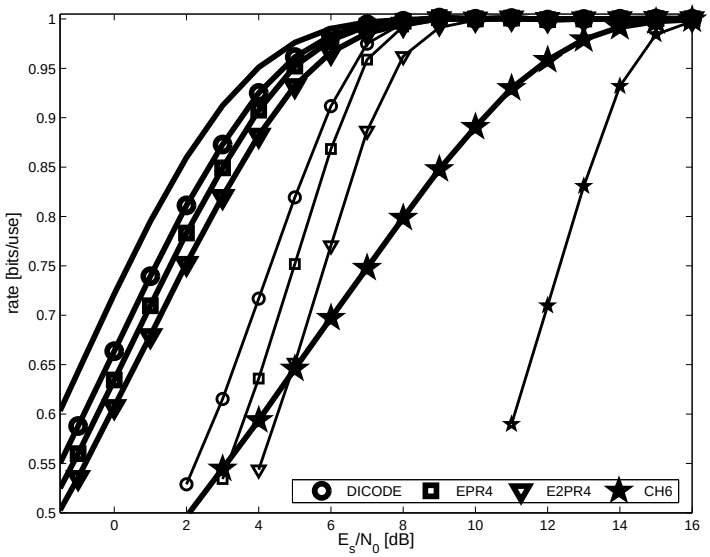
Figure 3.31 shows the performance of the different ordering strategies over the Rayleigh channel, both under the assumption of perfect knowledge of the fading factor, $a$, and in ignorance of the fading factor. The same observations that are made for the inter-symbol interference channels also apply to the Rayleigh fading channel. The relative



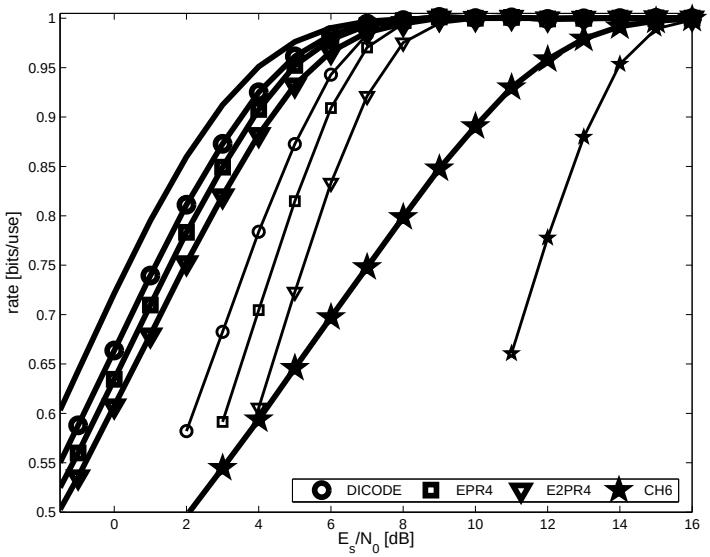

(1) DICODE (2) EPR4

(3) E2PR4 (4) CH6

Figure 3.29: Comparison of selected ISI channels (the channel coefficients can be found in table 3.2). The block randomised code structure strategy (dashed, see section 3.1.4) is compared to the tree-based strategy (solid, see section 3.1.6 for transmitting systematic bits first (square markers) and parity bits first (round markers). The respective channel capacity is shown in bold together with the BIAWGN capacity (solid without marker) for reference. The code used is the $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}, 2048\right)$ memory $n = 4$ code used in section 3.3.1, 1000 packets were simulated. As is expected from section 3.3.1 and Figure 3.28, the results are qualitatively consistent with the transmission over an AWGN channel. The performance of the $n = 4$ code is , as expected, worse than that of the $n = 6$ code shown in Figure 3.28 and the BER is significantly larger at low SNR. However, all ordering strategies perform as expected and no indication is available that an ISI transmission channel requires a specialised transmission order.

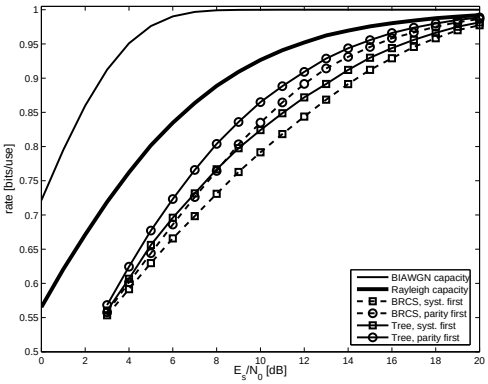



(1) Memory $n = 4$

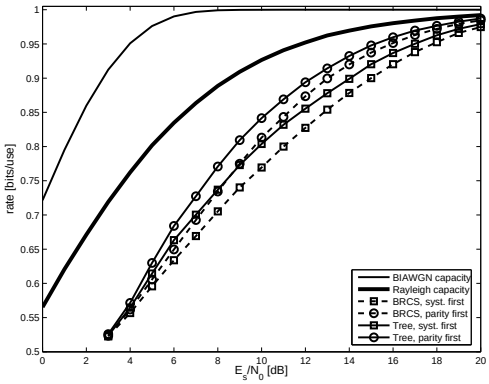



(2) Memory $n = 6$

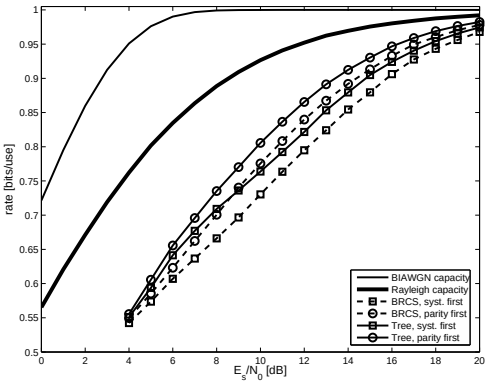
Figure 3.30: Comparison of the tree based ordering strategy for selected ISI channels (the channel coefficients can be found in table 3.2). 1000 packets were simulated. On the left is the $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}, 2048\right)$ memory $n = 4$ code used in section 3.3.1, on the right is the $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$ RSC code with memory $n = 6$ from section 3.1. 1000 packets were simulated. As expected a higher memory causes significant improvements. Capacities are shown in bold and simulation results in solid lines. The different channels used are DICODE (circles), EPR4 (squares), E2PR4(triangles) and CH6(stars). The capacity of the BIAWGN channel (plain bold line) is also shown for comparison.



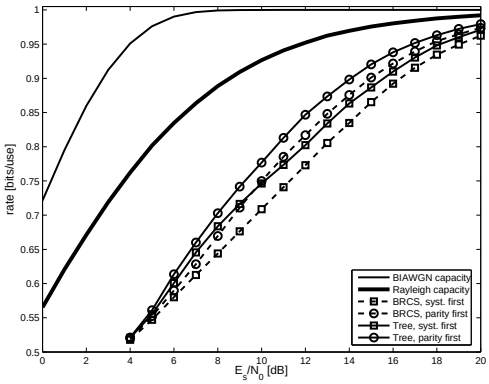

(1) $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$, scaled




(2) $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$, unscaled




(3) $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}, 2048\right)$, scaled




(4) $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}, 2048\right)$, unscaled

Figure 3.31: Performance of the transmission ordering strategies over a fast-fading Rayleigh channel. Shown are the tree based strategy (solid) and block randomised (dashed) ordering strategies with transmission of systematic bits first (square markers) or transmission of parity bits first (round markers). The $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$ and $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}, 2048\right)$ codes are investigated for the ideal case where the fast-fading factor $a$ is known and the LLR values can be appropriately scaled and the case where ignorant of the fading factor is assumed and the LLR values remain unscaled. 1000 packets were simulated.

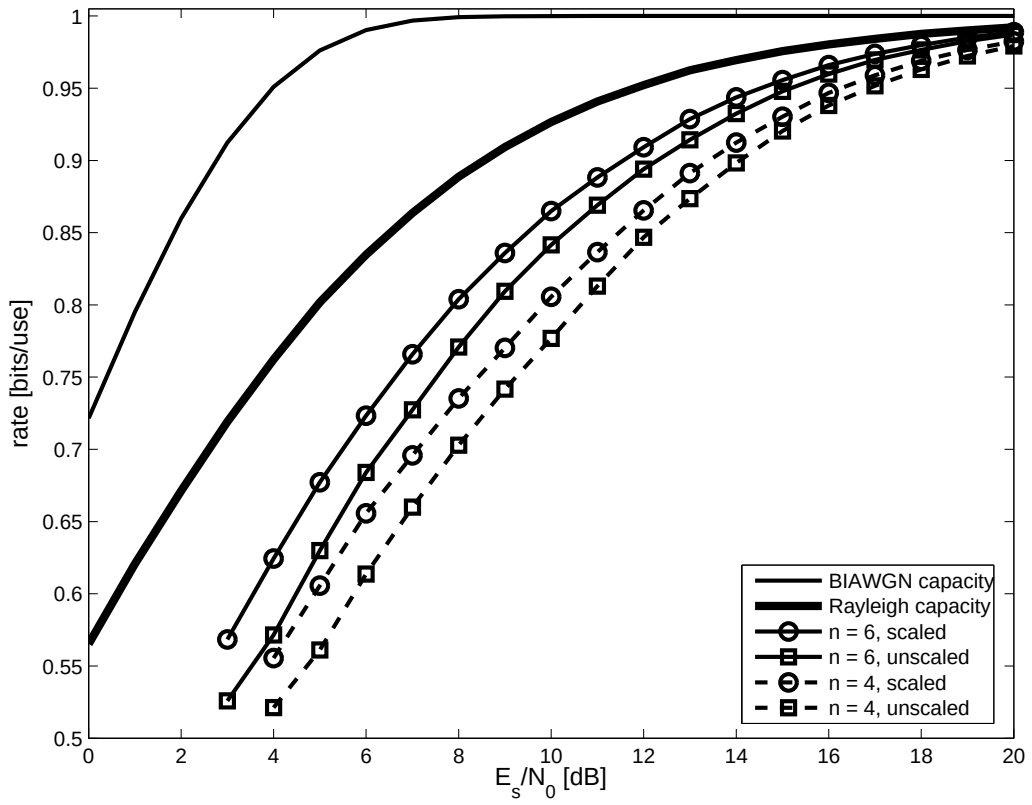
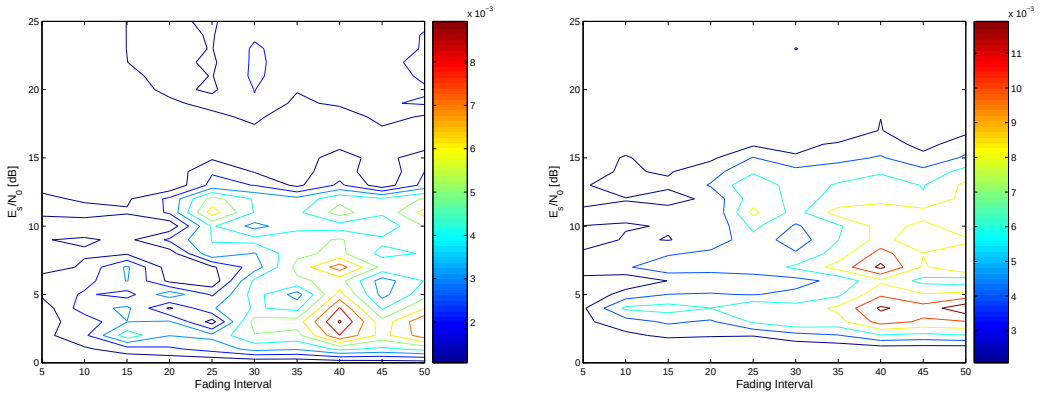



Figure 3.32: Comparison of the tree-based ordering strategy over a fast-fading Rayleigh channel. Shown the $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$ and $\left(1, \frac{1+D^3+D^4}{1+D^1+D^2+D^4}, 2048\right)$ codes with and without scaling of with the Rayleigh fading factor. 1000 packets were simulated.

order of the transmission ordering strategies does not change. Figure 3.32 summarises Figure 3.31 for the performance of the tree-based ordering strategy, allowing a more direct comparison. As expected, the memory of the used code is significant for the performance of the code, as is the knowledge of the scaling factor, whose ignorance incurs a performance penalty of roughly 1 dB.

The Rayleigh channel was also simulated for different fading intervals, i.e. the number of bits that the Rayleigh fading factor, $a$, remains constant. Simulated were fading intervals from 5 to 50 bits in increments of 5 bits. No significant difference was observed in the performance of the ordering strategies, regardless of the knowledge of the fading factor. Figure 3.33 shows the rate differences of the tree-based ordering strategy towards a Rayleigh channel with a fading interval of 1. As can be observed, the difference is $\lesssim 10^{-2}$, regardless of knowledge of the fading factor, $a$.




(1) Scaled (2) Unscaled

Figure 3.33: Difference in performance of the tree-based transmission ordering strategies over block-fading Rayleigh channel to the flat-fading Rayleigh channel for perfect knowledge (Figure (1)) and ignorance (Figure (2)) of the Rayleigh fading factor, $a$.

## 3.6 Concluding Remarks

In this chapter a transmission system that can transmit at a variable rate through reordering and truncating of the encoded sequence, $\mathbf{u}$, was investigated in order to determine how to construct a good reordering strategy, $\prod$. Section 3.1 showed that a straightforward truncation of the bits of $\mathbf{u}$ or a random selection of bits thereof produces poor results and developed a method for constructing a transmission order, $\prod$, that takes into account the unpredictability of errors on the transmission channel and the structure of the RSC code used in the transmission system. This reordering strategy, based on a binary tree, allowed the transmission system to effectively adapt to the channel conditions and achieve a maximum rate, $r_{\max} \approx 1$ at high SNR. Sections 3.4 and 3.5, demonstrated the suitability to inter-symbol interference and Rayleigh-fading transmission channels and it was shown in section 3.3 that this ordering strategy works regardless of the selection of constraint length, $n$, or blocksize, $s_{\mathbf{x}}$. However, an increase in blocksize makes the transmission less efficient, since the errors that occur on the transmission channel cannot be localised as well as they can be for small blocksizes. On the other hand, smaller blocksizes experience a “parity outage” where some packets cannot be delivered error free even when exhausting the available parity. Still, blocksizes of 2048 bits seem not to suffer from an increased residual bit-error rate, yet show a noticeable performance advantage over larger blocksizes. In addition, a blocksize of 2048 bits is reasonably resilient to rate degradation due to overhead caused by

the error-detection scheme or the transmission protocol.

In order to explain the different performance of transmitting either the parity or the systematic bits in the first $s_{\mathbf{x}}$ bits, the problem of correcting erasures was considered in section 3.2. An algorithm was developed to provide a performance bound on the performance of a particular transmission order, regardless of the actual input, by considering the recoverability of erasures. It is this problem of recoverability that explains how, due to the assignment of transition labels of the code, the systematic and parity bits differ and allowed the explanation of the performance difference observed in section 3.1. Building on this, a simple method was devised in order to determine the appropriate order from the transition diagram of the used RSC code. This is useful when investigating optimal generator polynomials for turbo codes, since the importance of systematic bits for turbo codes allow for 50% of the generator pairs (those with superior performance when the parity bits are transmitted first) to be eliminated.

Based on this chapter, the general recommendation for designing RSC encoded type-II hybrid ARQ transmission system is to employ the tree based ordering strategy described in section 3.1.6 and a blocksize of $s_{\mathbf{x}} > 1024$, if possible, to reduce the overhead of the transmission protocol. The relative order of transmission of systematic and parity bits depends on the generator polynomials and needs to be determined as described in section 3.2. Finally, it is interesting to note that the ordering strategies have essentially defined a measure of importance for the individual bits. In some situations, this information can be useful in designing the communication system. Maerkle and Sundberg[80] who place bits in a channel with a non-uniform interference pattern use rate-compatible puncturing tables in order to place the bits. The strategy developed in this thesis is thus directly applicable to their setup.

# Chapter 4

# Variable Rate Transmission of Turbo Codes

Turbo Codes, introduced by Berrou *et al.*[12], provide significant performance improvements over RSC codes. Since Turbo Codes use RSC codes as constituent encoders, they are a natural progression from RSC codes. Due to this construction of Turbo Codes, the results from chapter 3 provide a basis and a comparison for the results in this chapter. In particular, the results from chapter 3 make it possible to identify those cases, where the ordering strategies for Turbo Codes degrade them to RSC codes. While many of the insights that were obtained from the analysis of RSC codes in chapter 3 apply, the straightforward application of the principles from chapter 3 leads to poor performance and additional constraints need to be placed on a Turbo Code ordering strategy. **Section 4.1** follows section 3.1 and adapts the ordering strategies of RSC codes to Turbo Codes. The main new new dimension to consider when working with Turbo Codes is the availability of an additional code bit. **Section 4.2** deals with this code bit caused by the presence of the additional constituent RSC encoder and investigates the interplay between the two encoders. In particular it shows that the claim made by Rowitch and Milstein[109], that some systematic bits should be punctured in favour of parity bits is merely arising from a limitation of rate-compatible puncturing tables and not inherent in punctured Turbo Codes. Following this structural discussion of Turbo Codes, **section 4.3** considers the effect of the input blocksize on the performance of the ordering strategies, whereas **sections 4.4 and 4.5** consider the inter-symbol interference and Rayleigh channels, respectively. In all three cases

it is shown that the merit of the ordering strategy does not diminish. The choice of generator polynomials is the topic of **section 4.6**, which compares the performance of several pairs of generator polynomials that were suggested in the literature. Following suit, **section 4.7** compares the performance of the tree-based ordering strategy to the performance of rate-compatible punctured Turbo Codes, using optimised puncturing tables proposed by Rowitch and Milstein[109]. Finally, **section 4.8** offers a summary of the results of this chapter.

## 4.1 Adapting the RSC Ordering Strategy to Turbo Codes

The simulation that is used for Turbo Codes is very similar to that of the RSC codes, described in section 3.1. Instead of the RSC encoder and decoder, a parallel Turbo encoder and decoder pair, described in Figure 2.18 is used, with a random interleaver separating the two identical, constituent RSC encoders. The assumption of equiprobable bits as well as that of an AWGN channel is kept. In addition, following the argument made in section 3.1, the assumption of an error-free feedback for ARQ signalling is also kept. Each transmitted packet is assigned a newly generated transmission order $\prod$ and the minimum number of bits that are required for error free transmission, $\check{s}_{\mathbf{w}}$, determined by performing a binary search over the space of all possible packet sizes $s_{\mathbf{w}}$. However, since decoding of Turbo Codes is significantly more complex than decoding RSC codes,[1] the number of packets is reduced to 1000. The generator polynomials of the RSC simulations were reused. These polynomials are probably not optimal for use in Turbo Codes[2], however reusing them allows for a direct link to the results of the plain RSC codes from chapter 3. In fact the qualitative results of the simulations are largely unaffected by the choice of a particular pair of generator polynomials as shown in section 4.6.[3]

[1] Disregarding the interleaver, decoding a Turbo Code with 5 iterations is equivalent to decoding an RSC code of the same blocksize 10 times.

[2] A lot of research effort has gone into the design of optimal constituent codes, e.g. [8, 56, 122]

[3] Of course it is assumed here, that a non-catastrophic pair of generator polynomials is chosen. It is certainly possible to pick a pair of generator polynomials with known, poor performance.

| Simulation Parameters | |
|---|---|
| **General** | |
| Simulated packets | 1000 |
| **Source** | |
| Type | Memoryless |
| Samples per packet ($s_{\mathbf{x}}$) | 2048 |
| $p(x=1)$ | 0.5 |
| **Encoder** | |
| Type | Parallel Turbo (Fig. 2.18) |
| Interleaver | Random |
| Constituent Encoders | Identical RSC Encoders |
| **Constituent Encoders** | |
| Type | Recursive Systematic Convolutional (Fig. 2.16) |
| $g^{(0)}$ (feedback) | $1+D^1+D^2+D^3+D^6$ |
| $g^{(1)}$ (parity) | $1+D^2+D^3+D^5+D^6$ |
| **Channel** | |
| Type | Additive White Gaussian Noise |
| **Decoder** | |
| Type | Iterative Turbo (Fig. 2.19) |
| Iterations | 5 |
| **Constituent Decoder** | |
| Type | Soft-In Soft-Out BCJR Algorithm (Sec. 2.3.2) |

Table 4.1: Common parameters for the simulated transmission system shown in Figure 4.1 (unless stated otherwise).

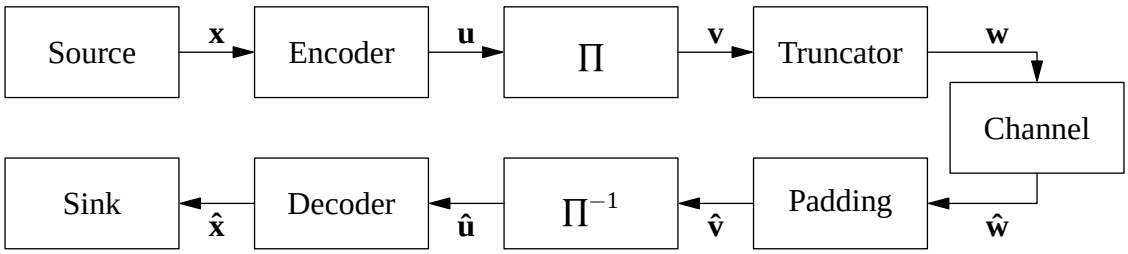



Figure 4.1: Communication system used in simulations. See table 4.1 for common parameters. The parameters for constructing the transmission order $\Pi$ are detailed in the respective sections.

### 4.1.1 Identity and Random Ordering

The identity ordering is, similar to RSC codes, the most naive approach to transmission ordering and, as expected, performs worst of all ordering schemes. It is interesting to note, however, that the rate of the identity ordering is very slightly better compared to the rate of the base code, $r_{\text{base}}$. The base code rate is $r_{\text{base}} = 2048/((2048+6)*3) \approx 0.3324$, whereas the rate obtained by the identity ordering is $r_{\max} \approx 0.3498$. This is not an exploitable difference, however, it is the first sign of a fundamental difference between RSC and Turbo Codes. Considering the existence of this interleaver, however, one would expect a much higher rate for the identity ordering. The reason why the identity ordering for RSC codes performed so poorly was that not all input bits were "covered" by the transmitted code bits, $\mathbf{w}$ (see section 3.1.1). Given the existence of the interleaver $\pi$, this argument is no longer true since bits from the end of the input sequence $\mathbf{x}$ can be early input bits to the second constituent RSC encoder. However, as can be seen from Figure 4.2, this is clearly not the case. This is due to the fact that systematic bits play an important role in the decoding of Turbo Codes (see section 4.1.3 in addition to the work by ten Brink [123]).

Figure 4.2 also shows the performance of the random ordering strategy. The maximum achieved rate, $r_{\max} = 0.8237$ is significantly better than that of the RSC code under the random ordering strategy ($r_{\max} = 0.7507$, see section 3.1.2), yet it is still far short of reaching a maximum rate of $r_{\max} = 1$. Like for RSC codes, the code structure needs to be exploited to achieve $r_{\max} \approx 1$.

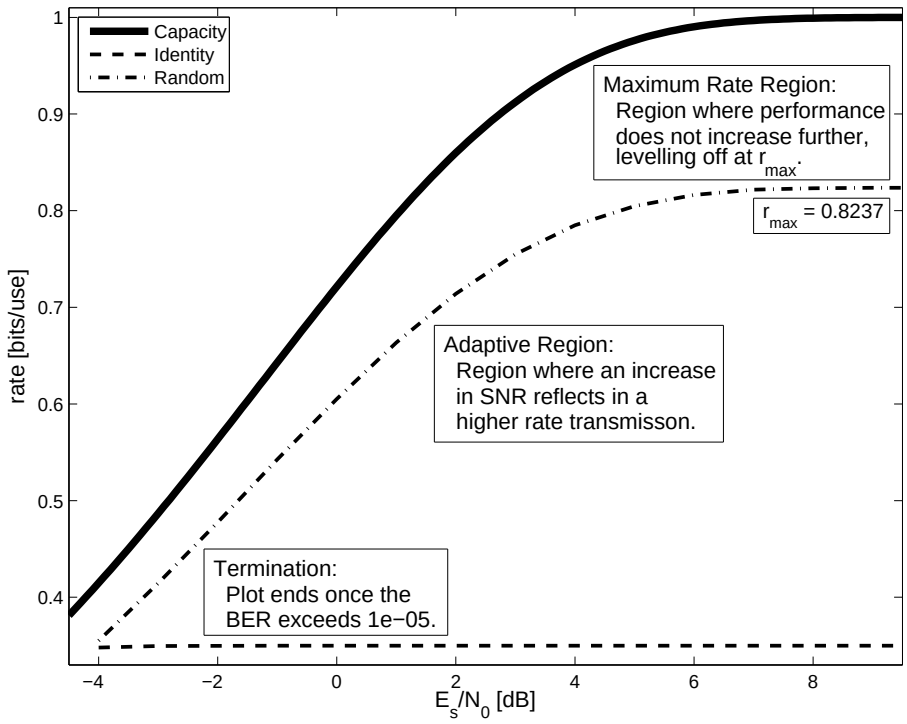



Figure 4.2: Performance of the identity (dashed) and random (dash-dot) ordering strategies against theoretical capacity (bold). Also shown are three regions of interest. In the maximum rate region, a transmission order reaches its best possible rate. In the adaptive region, the rate is adapted to the channel SNR. At termination, the base code cannot correct all errors and the BER exceeds $10^{-5}$.

### 4.1.2 Exploiting Code Structure

Section 3.1.3 described a method of exploiting the code structure of the base code to achieve a maximum rate of $r_{\max} = 1$. The same method can be used for Turbo Codes, since the encoder output of a Turbo Codes is similar to the output of an RSC code, simply adding an additional row to $\mathbf{U}$ to yield

$$\mathbf{U} = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 & \\ c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & \dots \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & \end{bmatrix}. \tag{4.1}$$

Reading this matrix row-wise leads to a similar performance change as was the case for RSC codes (compare Figure 3.4), block-randomisation as well behaves similarly to what is to be expected from RSC codes (compare Figure 3.5). Consider Figure 4.3 which displays the performance of the structured and block-randomised strategies for Turbo Codes. If compared to Figures 3.4 and 3.5, it is immediately clear that these two strategies are suboptimal, since the performance of the Turbo Code is *virtually identical* to the performance of the RSC code (where data for both codes is available). Only when the rate drops *below the rate of the RSC code* does the Turbo Code begin to outperform the RSC code significantly. A row-wise reading of $\mathbf{U}$ leads to the transmission of the bit sequence $s_1\, s_2\, s_3\, s_4 \dots\, c_{1,1}\, c_{1,2}\, c_{1,3}\, c_{1,4} \dots\, c_{2,1}\, c_{2,2}\, c_{2,3}\, c_{2,4} \dots$ which is identical to the sequence transmitted for the RSC code. This is not surprising, since the additional row is read last. This approach leads to the Turbo Code *degrading* to a plain RSC code, since the 2nd constituent decoder *does not receive any bits* and is therefore unable to improve the decoding reliability. This also explains, why the random ordering strategy (dash-dot in Figure 4.3) outperforms the block-randomised version (dashed, black in Figure 4.3), since the random ordering strategy obviously selects some bits of both encoders. Once the bits of the first constituent encoder and the systematic bits have all been transmitted (at rate $r \approx 0.5$) the second constituent decoder starts receiving bits thus putting into effect the Turbo principle with the respective benefits to transmission reliability. This effect can be seen clearly in Figure 4.3 where a significant improvement in performance occurs at $r \approx 0.5$.

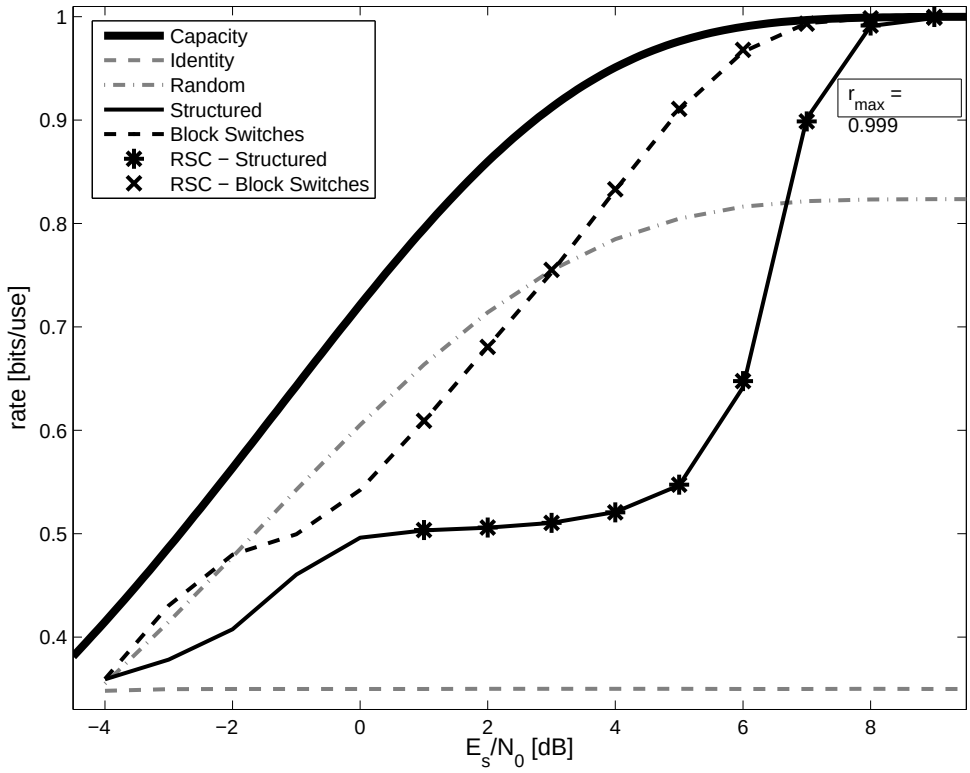



Figure 4.3: Performance of the structured (solid) and block-randomised (black dashed) ordering strategies against theoretical capacity (bold). Using the structured and block-randomised ordering schemes, the performance of the Turbo Code degrades significantly at high SNR, performing no better than the constituent RSC code on its own (the performance of the RSC code is indicated by stars and crosses for the structured and block-randomised ordering schemes, respectively). The reason for this merging of performance is the fact that the second constituent decoder does not receive any parity bits at high SNR and therefore cannot improve performance. Also shown for comparison are the identity (grey dashed) and random (dash-dot) ordering strategies.

### 4.1.3 Probabilistic Selection and Systematic Bits

Section 3.1.5 suggested randomising the intra-column order of $\mathbf{U}$ before randomising the rows in order to improve performance and determined that sending parity information first to be the best strategy. Given the successful use of row randomisation in section 4.1.2, it is reasonable to apply this approach to Turbo Codes. However, as can be seen in Figure 4.4 (round markers), this approach dramatically fails to deliver the expected performance, in fact, it performs worse than the random approach. The probabilistic selection of bits from each column was based on the belief that each bit plays an equal part in the decoding process. This, however, is not true as was already shown in section 3.1.5. In the case of the RSC code, omitting systematic bits in favour of parity bits improved the performance of the code. In the case of Figure 4.4 however, the performance decreased. In fact, as ten Brink argued[123], systematic bits are required to get Turbo decoding started. In the case of puncturing bits, there is another explanation. An additional systematic bit *immediately benefits both constituent decoders*. A parity bit, on the other hand, is only directly useful to one of the constituent decoders which needs to communicate this information indirectly to the second decoder via the extrinsic information passed between the constituent decoders (see Figure 2.19). The position of the systematic bits as the first bits in the transmitted bit sequence $\mathbf{v}$ thus needs to be preserved. In order to do this, the column interleaving $\pi_{columns}(\mathbf{U})$ is modified to keep the systematic bit of each column at the top of that column. The modified code matrix $\mathbf{U}''$ which is read row-wise to obtain the to be punctured sequence $\mathbf{v}$ is thus

$$\begin{aligned}\mathbf{U}'' &= \pi_{rows}\left(\pi^{*}_{columns}(\mathbf{U})\right) \\ &= \pi_{rows}\left(\begin{bmatrix} s_1 & s_2 & s_3 & \dots \\ \pi_{col,1}\left(\mathring{\mathbf{U}}_{col\,1}\right) & \pi_{col,2}\left(\mathring{\mathbf{U}}_{col\,2}\right) & \pi_{col,3}\left(\mathring{\mathbf{U}}_{col\,3}\right) & \dots \end{bmatrix}\right), \quad (4.2)\end{aligned}$$

where $\mathring{\mathbf{U}}$ is obtained by removing the top row (containing the systematic bits) from $\mathbf{U}$ and the size of the column interleaving operations $\pi_{col,i}\left(\mathring{\mathbf{U}}_{col\,i}\right)$ is adjusted accordingly. The resulting performance is shown in Figure 4.4 with square markers. By satisfying both decoders it is possible to obtain the benefits of Turbo decoding throughout the adaptive region of the code (from about $-4$ dB to about 7 dB), staying within 1 dB distance of the theoretical capacity limit of a binary input AWGN channel.

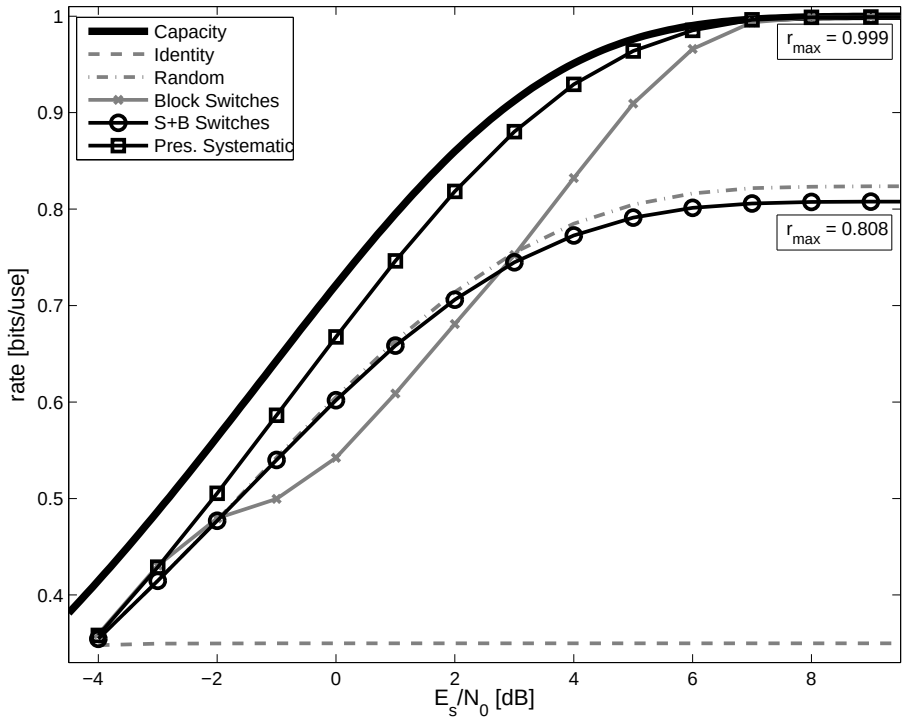



Figure 4.4: Performance of the probabilistic block-randomised ordering strategy (circles) against the theoretical capacity (bold) and the effect of preserving the systematic bits as the first bits to be transmitted (squares). Not preserving the systematic bits leads to a performance worse than a purely random approach. Preserving the systematic bits however leads to a performance within $1\,\mathrm{dB}$ of the theoretical capacity throughout the adaptive region. Also shown for comparison are the identity (dashed) and random (dash-dot) ordering strategies as well as the block-randomised structured strategy (crosses).

### 4.1.4 Binary Tree Based Ordering

RSC codes showed a significant performance increase when a tree-based ordering strategy was used (see section 3.1.6). Like for RSC codes, using a tree-based approach to distribute the parity bits, helps to increase the performance of the Turbo Code, however, the increase is by far not as pronounced. The tree-based ordering is constructed similarly to the block-randomised version with preservation of systematic bits. Consider the code matrix

$$\mathbf{U}'' = \pi_{tree}\left(\begin{bmatrix} s_1 & s_2 & s_3 & \\ \pi_{col,1}\left(\mathring{\mathbf{U}}_{col\,1}\right) & \pi_{col,2}\left(\mathring{\mathbf{U}}_{col\,2}\right) & \pi_{col,3}\left(\mathring{\mathbf{U}}_{col\,3}\right) & \cdots \end{bmatrix}\right), \tag{4.3}$$

where $\pi_{tree}(\cdot)$ reorders each row, such that the resulting row is the standard array representation of a balanced binary tree, as described in section 3.1.6. Figure 4.5 shows the performance of the tree-based approach. No significant gains are obtained through the use of the binary tree ordering (This is a consequence of the encoder memory. Section 4.6 will show that for memory $n = 4$ constituent RSC codes, the benefit is more pronounced).

The comparatively small performance increase from using the tree-based approach that worked well for RSC codes suggests that the selection of the bits for the Turbo Code is not as important as it was for the RSC code. In other words, parity information about $x_t$ is available from a larger number of bits. The performance of the deterministic tree, where the order of the individual entries in one particular level of the tree is not randomised supports this theory (Figure 4.6). However, this better use of the parity information does not lead to an arbitrary selection of the parity information. Also shown in Figure 4.6 is a naive approach of only selecting parity information at random, i.e.

$$\mathbf{U}'' = \begin{bmatrix} s_1 & s_2 & s_3 & \\ \pi_{col,1}\left(\mathring{\mathbf{U}}_{col\,1}\right) & \pi_{col,2}\left(\mathring{\mathbf{U}}_{col\,2}\right) & \pi_{col,3}\left(\mathring{\mathbf{U}}_{col\,3}\right) & \cdots \end{bmatrix}. \tag{4.4}$$

This approach, although better than the structured approach shown in Figure 4.3 (because both decoders work), fails to come even close to the capacity limit for higher SNR values. Some randomisation or spreading needs to be performed, in order to approach capacity.[4]

[4]Note that the S-Random, or parity spreading, interleavers will also work in this case.

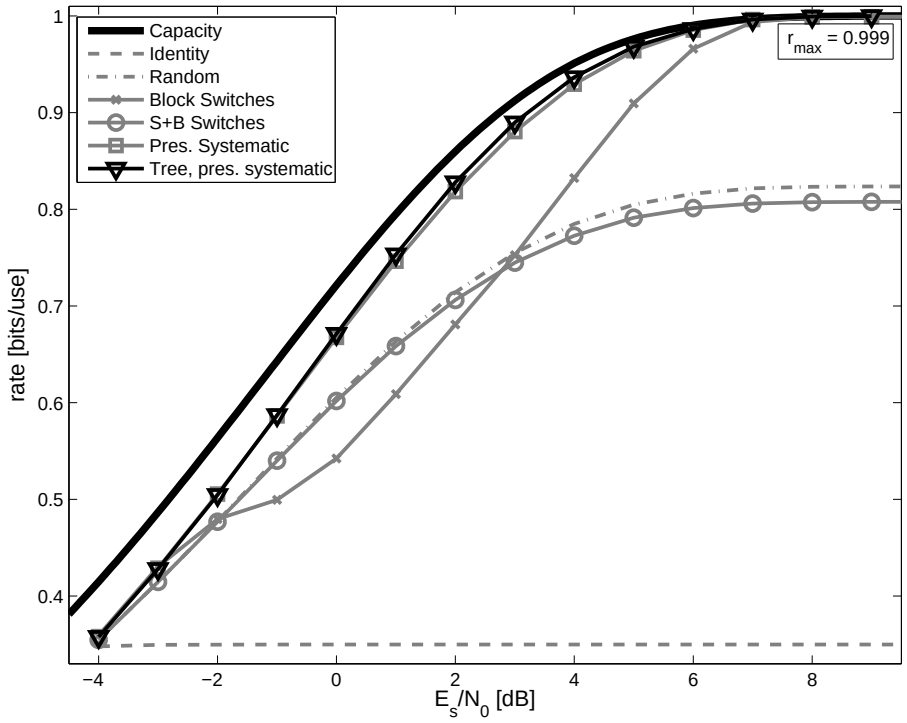



Figure 4.5: Performance of the tree-based ordering strategy (triangles) against the theoretical capacity (bold) and performance of the block-randomised ordering strategy with preservation of systematic bits (square). Only for high SNR values is the tree-based approach able to marginally outperform the block-randomised strategy. Also shown for comparison are the identity (dashed) and random (dash-dot) ordering strategies as well as the block-randomised structured strategy (crosses).

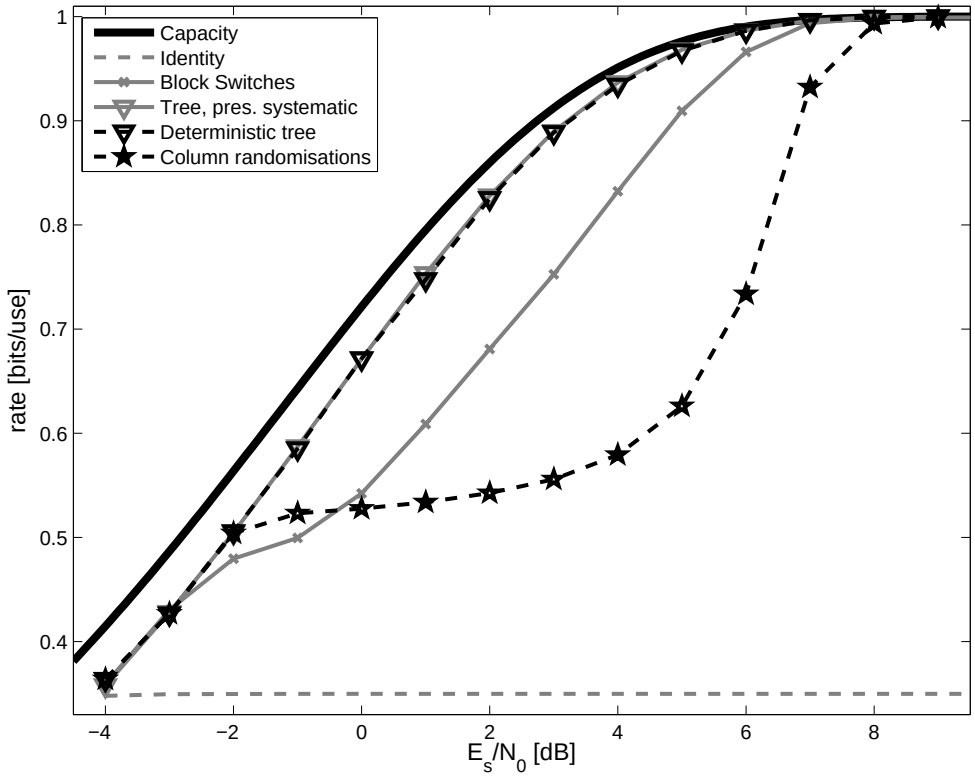



Figure 4.6: Performance of the deterministic tree ordering strategy (dashed, triangles) against theoretical capacity (bold) and the tree based approach (solid, triangles). Without randomisation *the block*, the performance of the tree is not significantly degraded. However, the tree based approach is still necessary as shown by the performance of a structured approach without block-randomisation (dashed, stars). Also shown for comparison is the identity (dashed) ordering strategy as well as the block-randomised structured strategy (crosses).

### 4.1.5 Summary

It was shown that the strategies developed for RSC codes in section 3.1 can also be used for Turbo Codes if it is ensured that both constituent decoders of the Turbo Code are supplied with parity bits. Otherwise, the performance of Turbo Codes degrades to that of single RSC codes and their full potential cannot be exploited. In fact, degrading a Turbo Code in this way is worse than using a single RSC code in terms of the complexity of the decoding process. Since a Turbo decoder uses RSC constituent decoders and, in this thesis, performs 5 iterations, the time required to perform decoding roughly ten times that of the single RSC code. If the Turbo Code is fed properly with parity information, it shows a remarkable resilience to certain randomisation types. Whereas the deterministic tree ordering for RSC codes significantly decreased performance, the same cannot be said for the Turbo Code. This seems to be a significant advantage of Turbo Codes, since randomisation of the ordering strategy is not required and one, fixed ordering permutation can be used. However, it has to be noted, that the Turbo Code, by its design, already contains a random element, the interleaver, so that any Turbo coding system needs to be capable of randomised reordering.

#### Potential Improvements

The ordering strategies for Turbo Codes already come very close to the capacity limit. The same considerations as for the RSC ordering strategies can be made, though. Given that an initial size of a packet is known, the ordering strategy could be improved by evenly spreading the parity bits in this initial packet. Each of these initial bits can be considered as the root of an individual tree, or, alternatively, they can be considered to be linked by a virtual parent node. The placement of the remaining bits then follows the procedure outlined in section 4.1.4. Given the small improvement observed from the use of the tree-based approach, gains from this approach are likely to be minimal.

## 4.2 Transmission Priority of Systematic and Parity Bits

The Turbo Code, like the RSC code, offers different types of bits. For Turbo Codes of base rate $r_{\text{base}} = 1/3$, these types are the systematic bits and parity bits from either encoder. Consider the matrix representation of a rate $r = 1/3$ systematic Turbo Code $\mathbf{U}$, such that

$$\mathbf{U} = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 & \\ c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & \cdots \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & \end{bmatrix}. \tag{4.5}$$

There are a total of 6 different possibilities in which order the bits in each column can be arranged and hence 6 different possibilities of the *relative* order of transmission of bits for each time index, $t$. Let $\mathcal{F}$ be a random variable that defines the selection of one of the systematic and parity bits to be transmitted before the other two. Let $p_0 = \Pr\{\mathcal{F} = \text{systematic}\}$, $p_1 = \Pr\{\mathcal{F} = \text{parity 1}\}$ and $p_2 = \Pr\{\mathcal{F} = \text{parity 2}\}$, be the probabilities that the systematic, first or second parity bit, respectively, is selected to be transmitted before the other two. Clearly $p_0 + p_1 + p_2 = 1$, since there are only the systematic and two parity bits available.

For ease of discussion, only a restricted subset of probability arrangement is considered. Let the second position be filled with one of the remaining bits, according to their *relative* probability, thus if the parity bit of the first decoder was selected to be transmitted first, the probability of transmitting the systematic bit next is $p_0/(p_0 + p_2)$, the probability of transmitting the parity bit of the second decoder next is $p_2/(p_0 + p_2)$. For the special case, where $p_0 = p_2 = 0$, the probabilities are set to $p_0 = 0.5$ and $p_2 = 0.5$ (the same applies to other combinations respectively). Moreover, a probability of 0 indicates that the bits of this type get transmitted as the *last* bits of the transmission. For ease of presentation, these three probabilities will be expressed as $p_0 : p_1 : p_2$, making it possible to capture the probability of relative transmission of bits in a single expression. Defined this way, the probabilities $p_0$, $p_1$ and $p_2$ also define the average composition of an initial packet of size $s_{\mathbf{x}}$, which, if decodable, achieves rate $r = 1$. Note that this approach to ordering bits is somewhat limited, since there is no possibility to define the block-wise transmission presented in section 4.1.2 other than specifying $(1-\varepsilon) : \varepsilon : 0$, with $\varepsilon$ being a very small number. However, as was shown in Figure 4.3, this type of transmission strategy performs poorly. The inability of properly specifying it is therefore not important in the context of this section.

Section 4.1.3 presented the block-randomised approach, adapted to Turbo Codes, and showed that if systematic bits are preserved as the first bits to be transmitted relative to their respective parity bits, rate $r \approx 1$ can be achieved. Rowitch and Milstein[109], assert that under some situations it is beneficial to puncture systematic bits despite acknowledging

> In general, puncturing *systematic* code symbols of a Turbo Code leads to a greater performance loss than when puncturing non-systematic code symbols. In fact, there are no examples in the literature, known to the authors, in which puncturing of the systematic symbols of a Turbo Code is advocated.

Consider Figure 4.7 which shows several probabilities of transmitting the systematic bits first. Note that this probability directly corresponds to the average composition of the first $s_{\mathbf{x}}$ bits. Thus for $0.8:0.1:0.1$, the first $s_{\mathbf{x}}$ will contain 80% systematic bits and 20% parity bits (10% for each decoder). As the fraction of systematic bits decreases, the maximum attainable rate, $r_{\max}$, achieved by the transmission system for high SNR channel conditions decreases as well.

Equal distribution of the parity bits to the decoders is not necessarily the only possibility. Consider Figure 4.8 which shows the asymmetric distributions $0.8:0.15:0.05$, $0.8:0.05:0.15$, $0.9:0.1:0.0$ and $0.9:0.0:0.1$. A probability distribution in favour of the first decoder allows the transmission system to achieve a higher maximum rate, $r_{\max}$. It is possible to cut out one decoder completely and doing so for the second decoder (Figure 4.8.2) effectively makes the transmission system equivalent to an RSC code transmission system that can fall back on the bits of the second constituent decoder when necessary. The resulting performance loss has already been discussed in section 4.1.2.

Figure 4.8.2 illuminates why Rowitch and Milstein[109] propose the puncturing of systematic bits when designing a series of rate-compatible puncturing tables for Turbo Codes. Consider the design of a rate-compatible puncturing table, $\mathbf{P}$, of period $P$. Thus, if rate $\mathring{r} = 1$ is to be achieved, $P$ 1s have to be placed in the table. Therefore a dilemma arises of where to place these 1s, either selecting parity bits or systematic bits. In [109], the highest rate puncturing table has a rate of $8/9$ and Rowitch and Milstein concluded, that for this rate, it is beneficial to allocate one 1 to each decoder, thus omitting a single 1 from the row in the puncturing table $\mathbf{P}$ that corresponds to the systematic bits. The

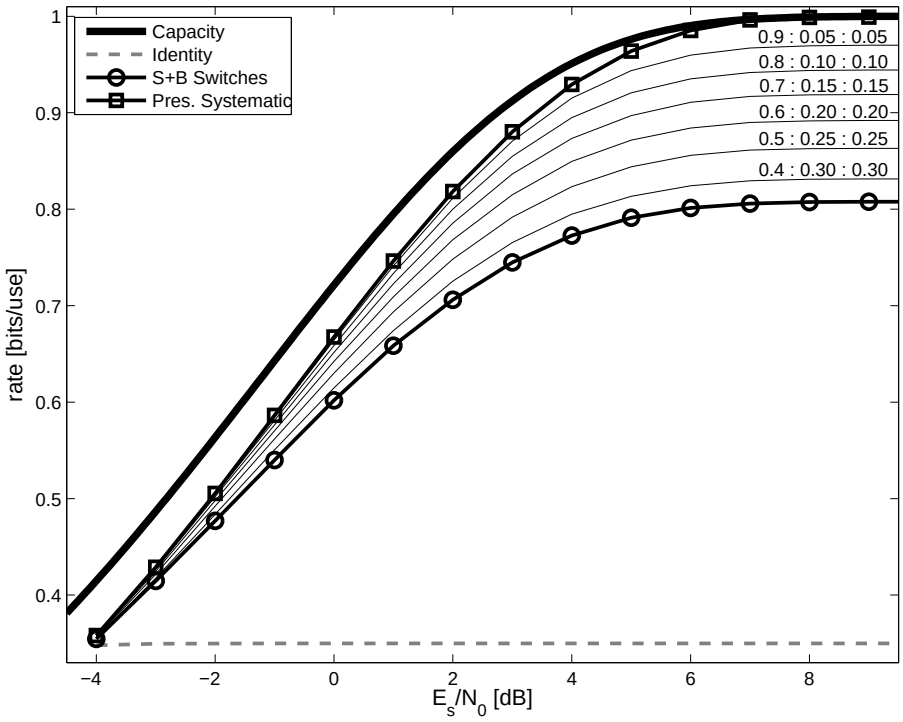
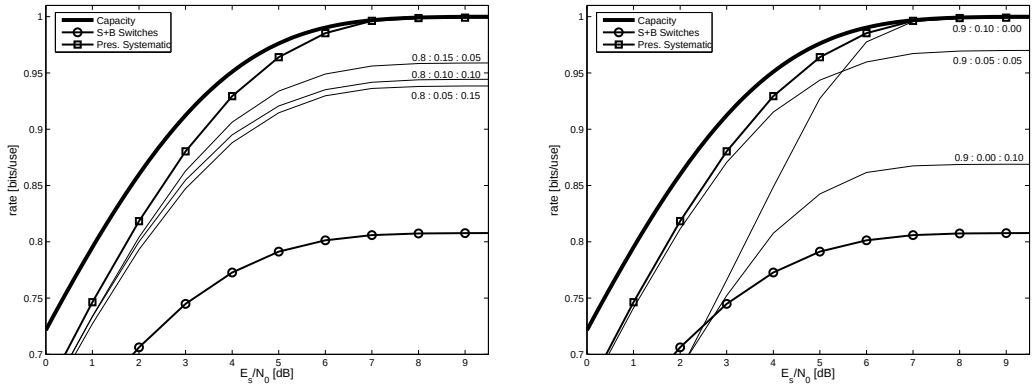



Figure 4.7: Comparison of different probabilities of transmitting the systematic bit first. As the fraction of systematic bits decreases, so does the maximum attainable rate, $r_{\max}$, achieved by the transmission system for high SNR channel conditions.




(1) Asymmetric distribution (2) Removing one decoder

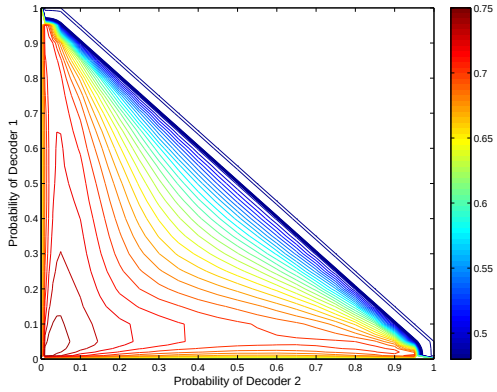
Figure 4.8: Performance comparison of an asymmetric distribution of selecting probability to the constituent decoders.

1 for the systematic bits is then supplied in the next lower rate table of rate $8/10$. The systematic bits are thus incomplete in the range of $r \in \left[0.8, 0.\overline{8}\right]$. It is clear from the comparison of Figures 4.8.1 and 4.8.2, that having both decoders supplied with bits outperforms the case of having only a single decoder working.

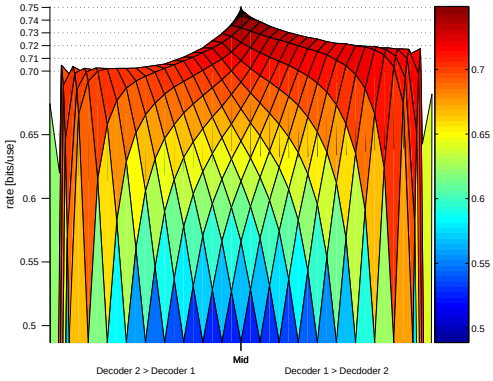
Puncturing systematic bits incurs a performance penalty for high rate transmission, as indicated by Figure 4.7 (disregarding the undesirable situation of omitting one decoder completely and thus reverting the system the performance of an RSC one). The benefit of puncturing systematic bits, observed by Rowitch and Milstein, arises solely due to a limitation caused by the use of rate-compatible puncturing tables and the coarse granularity in selecting the bits that they provide.

An evaluation of the performance of possible combinations of the probabilities of the individual constituent decoders is shown in Figure 4.9 for 1 dB and 9 dB. Immediately obvious is the apparent asymmetry of the two constituent decoders. For 9 dB, a mix of systematic bits and parity bits from the first decoder is able to achieve $r_{\max} = 1$, whereas the reverse, a mix of systematic bits and parity bits from the second decoder, shows a significant performance drop. This is surprising at first, however, the two constituent decoders are separated by an interleaver.[5] It is the effect of this interleaver that causes the performance drop at high rate. The input bits are reordered before being encoded by the constituent encoder. Following the argument of correcting erasures made in section 3.2, best possible performance is achieved when each input bit is covered

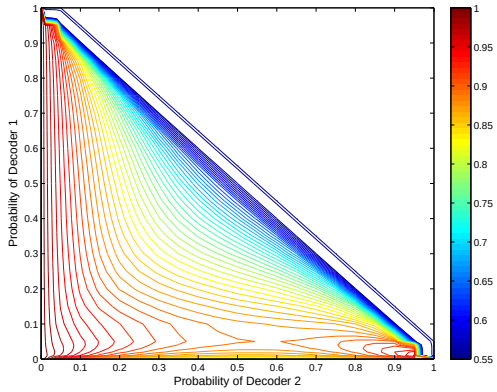
[5]There is also the difference that one of the two decoders has termination bits, thus returning to a specific state. However the difference in performance is only marginal.



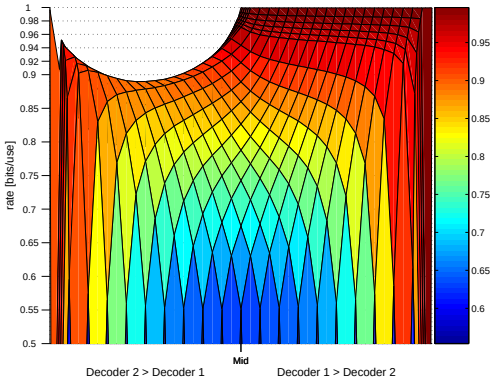

(1) 1 dB, contour




(2) 1 dB, view from (1,1) onto (0,0)




(3) 9 dB, contour




(4) 9 dB, view from (1,1) onto (0,0)

Figure 4.9: Distribution of the probability of selecting the bit of the first or second constituent decoder to be transmitted first. The probability of transmitting the systematic bit first is implicitly calculated from the probabilities of the constituent decoders.

by either a parity or a systematic bit. Clearly, given the presented selection strategy, this is not achieved. At 9 dB, the maximum rate $r_{\max} = 1$ can be achieved by assigning a probability of $(1-x):x:0$. Any other assignment will lead to a degradation of the maximum rate. Considering the performance at 1 dB (Figure 4.9.1), any assignment but $1:0:0$ provides suboptimal results. These results confirm that there is *no benefit* of puncturing systematic bits at any point in the transmission process.

### 4.2.1 Correcting for the Turbo Interleaver

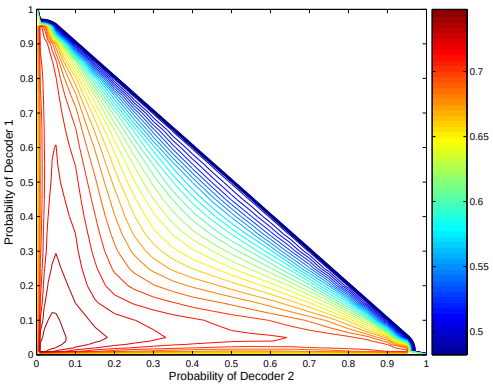
The performance of the two constituent decoders in Figure 4.9.3 appears asymmetric due to the presence of the interleaver. This interleaver causes a mixture of systematic bits and bits from the second constituent decoder to cover some input bits twice, i.e. due to the interleaving, a parity bit that is produced for a given input bit can be selected in addition to the parity bit. Furthermore, some bits may not be covered by either the systematic bit *or* the parity bit that was produced for the respective the input bit. The randomisation caused by the interleaver thus has an effect similar to that of the purely random ordering strategy, presented in section 4.1.1. Fortunately, the presence of the interleaver can be corrected for, as long as the permutation $\pi$, that the interleaver applies to the input bits before passing them to the second constituent encoder, is known. Consider the modified code matrix

$$\mathbf{U}^* = \begin{bmatrix} & s_1 & s_2 & s_3 & s_4 & \cdots & \\ & c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & \cdots & \\ \pi^{-1}( & c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & \cdots & ) \end{bmatrix}. \tag{4.6}$$

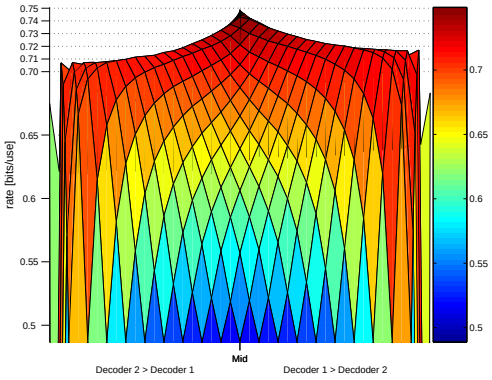
If selection method that was described earlier is applied to $\mathbf{U}^*$, the correct parity bits of the second constituent decoder are chosen. This modification allows a combination of systematic bits and bits from the second constituent encoder to achieve $r_{\max} = 1$ (see Figure 4.10). However, it has to be noted that this modification does not improve the performance of a *mixture* of parity bits from both decoders. In fact the performance shown in Figure 4.7 does not change. The preservation of systematic bits as the first bits to be transmitted is still essential to optimal performance of the Turbo Code.

## 4.3 Input Blocksize

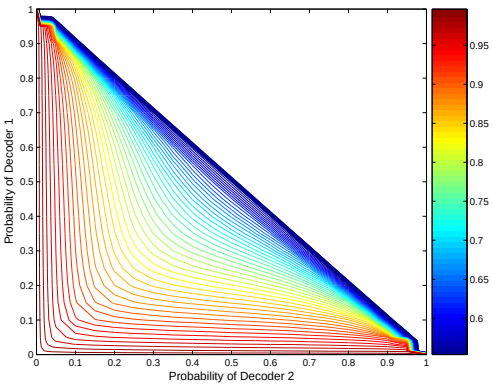
During the development, the input blocksize was fixed to $s_{\mathbf{x}} = 2048$ bits. This was done because it allows for a direct comparison to the RSC codes investigated in chapter 3, the simulation complexity of 2048 bits is still reasonable and the blocksize is reasonably large. Ideally, the blocksize, $s_{\mathbf{x}}$, satisfies $s_{\mathbf{x}} > 10^4$ in order to maintain good error performance (see [75]). Consider Figure 4.11 which shows the performance of some of the considered ordering strategies. No significant difference in performance, as far as *rate* is concerned, can be observed. Thus, the choice of $s_{\mathbf{x}} = 2048$ bits for the input



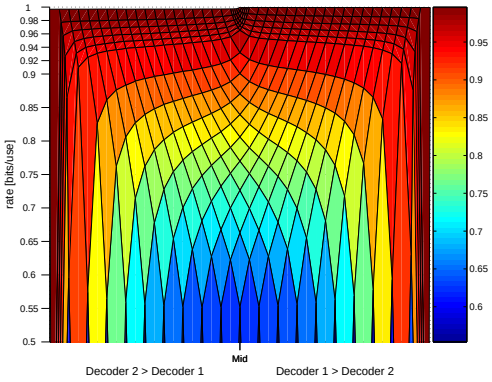

(1) 1 dB, contour




(2) 1 dB, view from (1,1) onto (0,0)




(3) 9 dB, contour




(4) 9 dB, view from (1,1) onto (0,0)

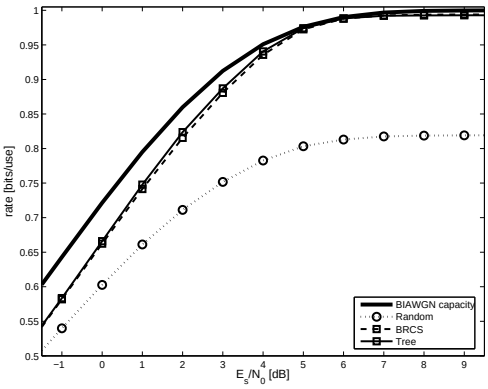
Figure 4.10: Distribution of the probability of selecting the bit of the first or second constituent decoder to be transmitted first while correcting for the Turbo interleaver. The probability of transmitting the systematic bit first is implicitly calculated from the probabilities of the constituent decoders.

blocksize does not have a negative impact on performance, even though $s_{\mathbf{x}} < 10^4$.

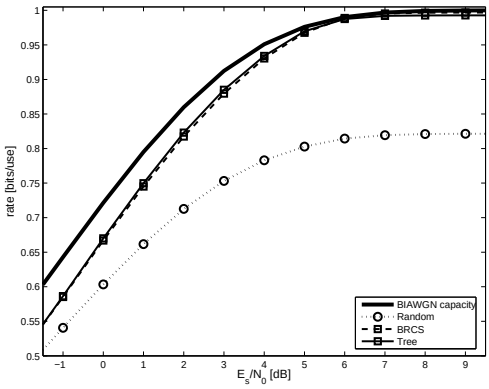
It was observed in section 3.3.1 for RSC codes, that the change in blocksize does not change the relative order between the ordering strategies. Figure 4.11 shows that this observation can be made as well for Turbo Codes. The major difference between RSC codes and Turbo Codes, however, is that RSC codes are very sensitive to the input blocksize, $s_{\mathbf{x}}$. An increase in blocksize led to a significant decrease in performance for the RSC codes (see Figure 3.23 in section 3.3.1). This led to a potential trade-off between the performance at high-SNR vs. low-SNR channels. This sensitivity on the input blocksize cannot be observed for Turbo Codes. Consider Figure 4.12 in which the performance of the random, block-randomised and tree-based ordering strategies are compared against the performance of the $s_{\mathbf{x}} = 2048$ code for different blocksizes $s'_{\mathbf{x}}$. The difference in performance is well below $10^{-2}$ and thus well below 1% of performance. The insensitivity to blocksize, and hence the small performance difference, can most easily be explained by considering the structure of the Turbo encoding process, notably the interleaver. The RSC code, lacking this interleaver only considers bits adjacent to a potentially erroneous bit. The further away one gets from this bit, the less can be deduced about the state of the bit in question. In a way. the parity bits "contain" less information about this bit. In Turbo Codes, this bit "adjacency" gets changed by the interleaver for the second encoder. Thus, information about bits that are too far away for the *single* RSC decoder to use is available through the detour of prior information and the second decoder.

A slight dependence on blocksize is also observed for the performance gain if the tree-based ordering compared to the block-randomised ordering strategy. Similar to the performance behaviour of the blocksize in general, the advantage of the tree-based strategy are highest for low blocksizes (256 bits), hit a minimum for 512 bits and then starts increasing, as blocksize increases.

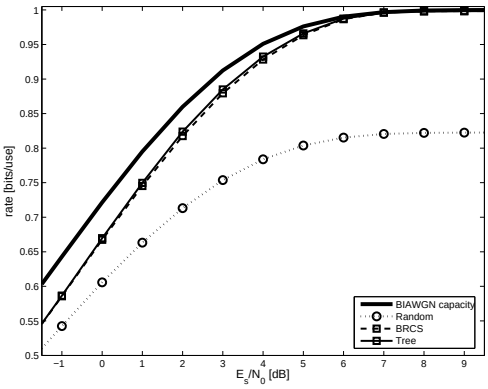
Turbo Codes require a somewhat large blocksize compared to RSC codes. Lin and Costello[75] suggest a minimum blocksize of $s_{\mathbf{x}} > 10^4$. The results presented in this section lead to the conclusion, that the presented ordering schemes work well regardless of the input blocksize, $s_{\mathbf{x}}$ and hence other considerations of the transmission system, such as coding delay or complexity of implementation will constrain the blocksize rather than the transmission order.



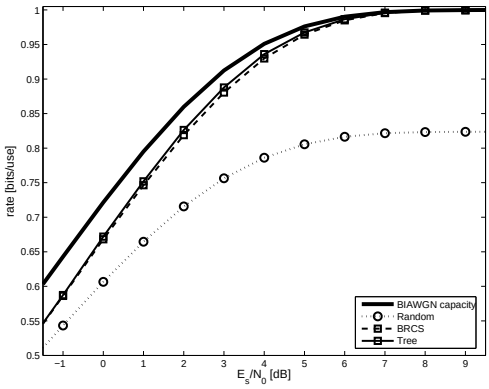

(1) $s_{\mathbf{x}} = 256$ bits



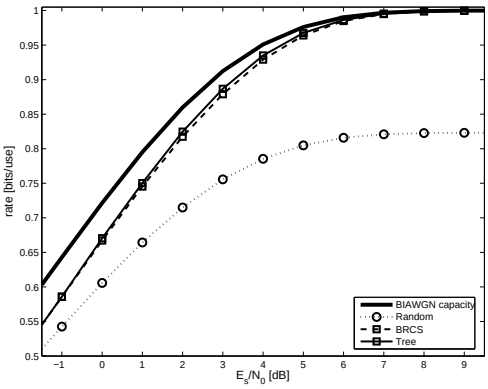

(2) $s_{\mathbf{x}} = 512$ bits



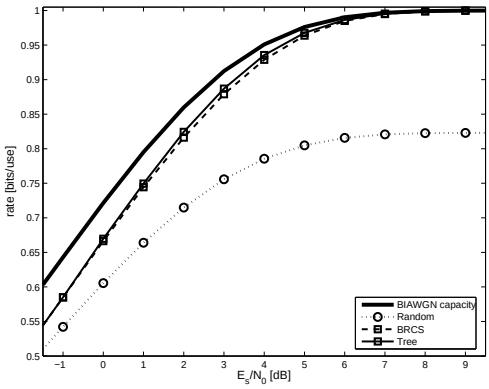

(3) $s_{\mathbf{x}} = 1024$ bits




(4) $s_{\mathbf{x}} = 4096$ bits

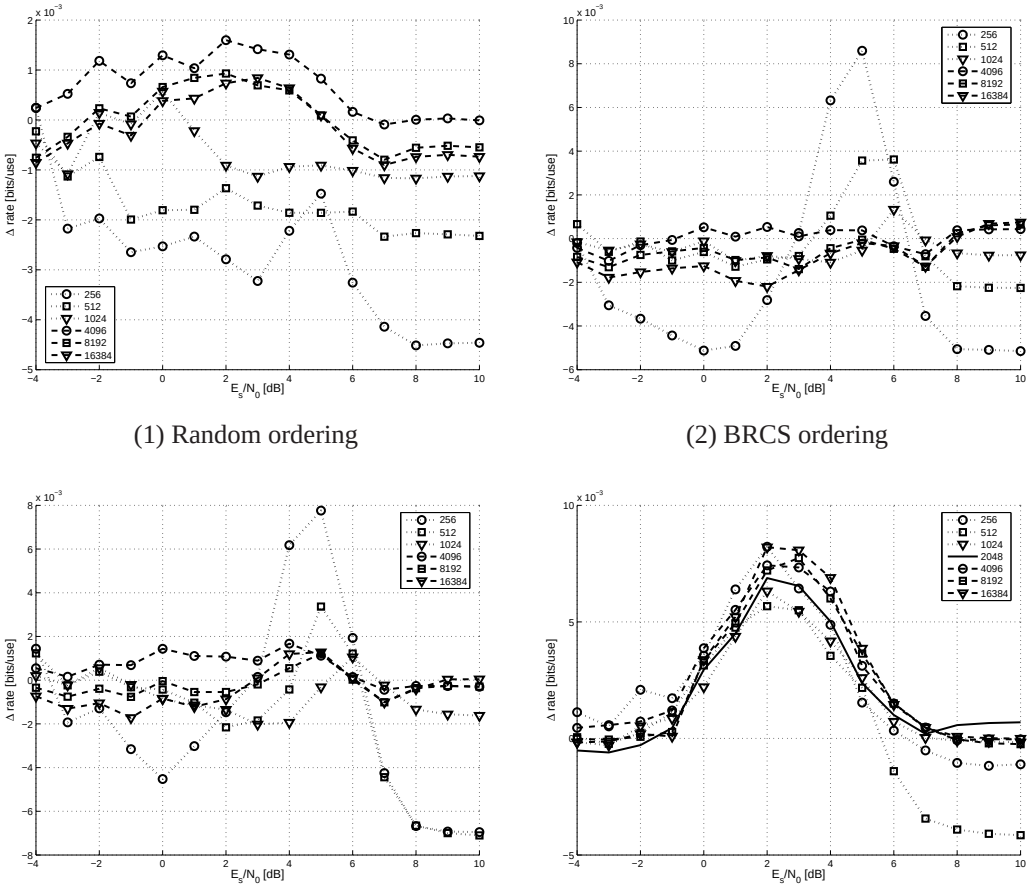



(5) $s_{\mathbf{x}} = 8192$ bits




(6) $s_{\mathbf{x}} = 16384$ bits

Figure 4.11: Performance of the $\left(1+D^1+D^2+D^3+D^6, 1+D^2+D^3+D^5+D^6\right)$ Turbo Code for different input sizes. Notice that, contrary to RSC codes, an increasing blocksize, $s_{\mathbf{x}}$, does not affect the performance of the employed ordering strategies significantly and that the relative order of the ordering strategies is also unaffected. 1000 packets were simulated for each input size.




(1) Random ordering

(2) BRCS ordering

(3) Tree ordering

(4) Performance gain of Tree vs. BRCS

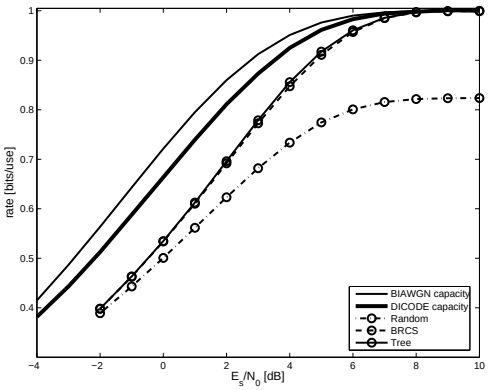
Figure 4.12: Performance difference with regard to a blocksize of $s_{\mathbf{x}} = 2048$ of the random, BRCS and tree-based ordering strategies given different input blocksizes (Figures (1), (2) and (3)) and the relative advantage of using the tree-based ordering strategy over the BRCS ordering strategy (Figure (4).

| Channel Name | Normalised Impulse Response |
|---|---|
| DICODE | $a(D) = \left(1 - D^1\right)/\sqrt{2}$ |
| EPR4 | $a(D) = \left(1 + D^1 + D^2 + D^3\right)/2$ |
| E2PR4 | $a(D) = \left(1 + 2D^1 - 2D^3 - D^4\right)/\sqrt{10}$ |
| CH6 | $a(D) = 0.19 + 0.35D^1 + 0.46D^2 + 0.5D^3 + 0.46D^4 + 0.35D^5 + 0.19D^6$ |

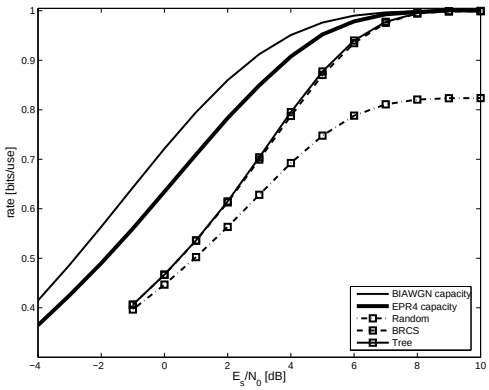
Table 4.2: Impulse responses of selected channels investigated by Arnold and Loeliger[3].

## 4.4 Inter-Symbol Interference Channels

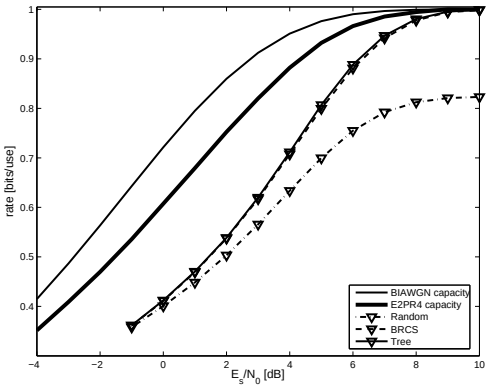
The investigation of data transmission over inter-symbol interference channels using Turbo Codes results in the same general observations that were made for the use of RSC codes in section 3.4. The channel plays an important role in determining the achieved rate of transmission, however, the relative ordering of the transmission strategies remains unchanged. Figure 4.13 shows the performance of the DICODE, EPR4, E2PR4 and CH6 channels (see table 4.2 for the respective channel coefficients). Like for the AWGN channel (Figure 4.5), the tree-based ordering strategy only marginally outperforms the block-randomised strategy (both with preservation of the systematic bits). Figure 4.14 compares the performance of the tree-based strategies for the Turbo and a single (constituent) RSC code over the the DICODE, EPR4, E2PR4 and CH6 channels. As mentioned, the performance of the Turbo and RSC codes is greatly affected by the choice of channel memory. However, Figure 4.14 shows that the performance difference between RSC and Turbo Code is reduced slightly with an increase in channel memory. This suggests that Turbo Codes are affected slightly more severely than RSC codes are. It is difficult to proof why this should be the case, however, it is the conjecture of the author, that the reason for this effect lies again with the interleaver. Without channel memory, the interleaver causes “clean” information to be supplied to the constituent decoder, since the order of the bits changes. With an increase in channel memory, it gets increasingly difficult to obtain information from bits that are not intertwined (by the channel) with other bits in the bitstream, hence the increased degradation due to channel memory.



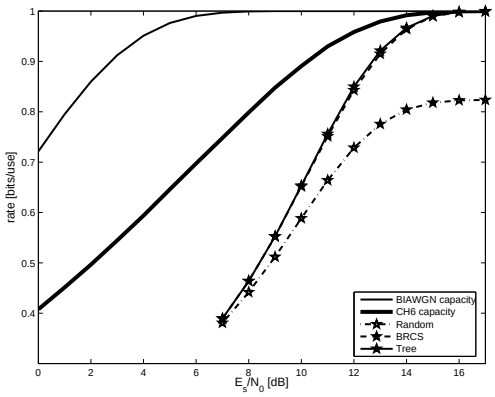

(1) DICODE




(2) EPR4




(3) E2PR4




(4) CH6

Figure 4.13: Comparison of selected ISI channels (the channel coefficients can be found in table 4.2). The block randomised code structure strategy (dashed, see section 4.1.3) is compared to the tree-based strategy (solid, see section 4.1.4), each with preservation of systematic bits. Also shown is the performance of the purely random strategy (dash-dot). The respective channel capacity is shown in bold together with the BIAWGN capacity (solid without marker) for reference. The results are qualitatively consistent with the transmission over an AWGN channel and no indication is available that an ISI transmission channel requires a specialised transmission order.

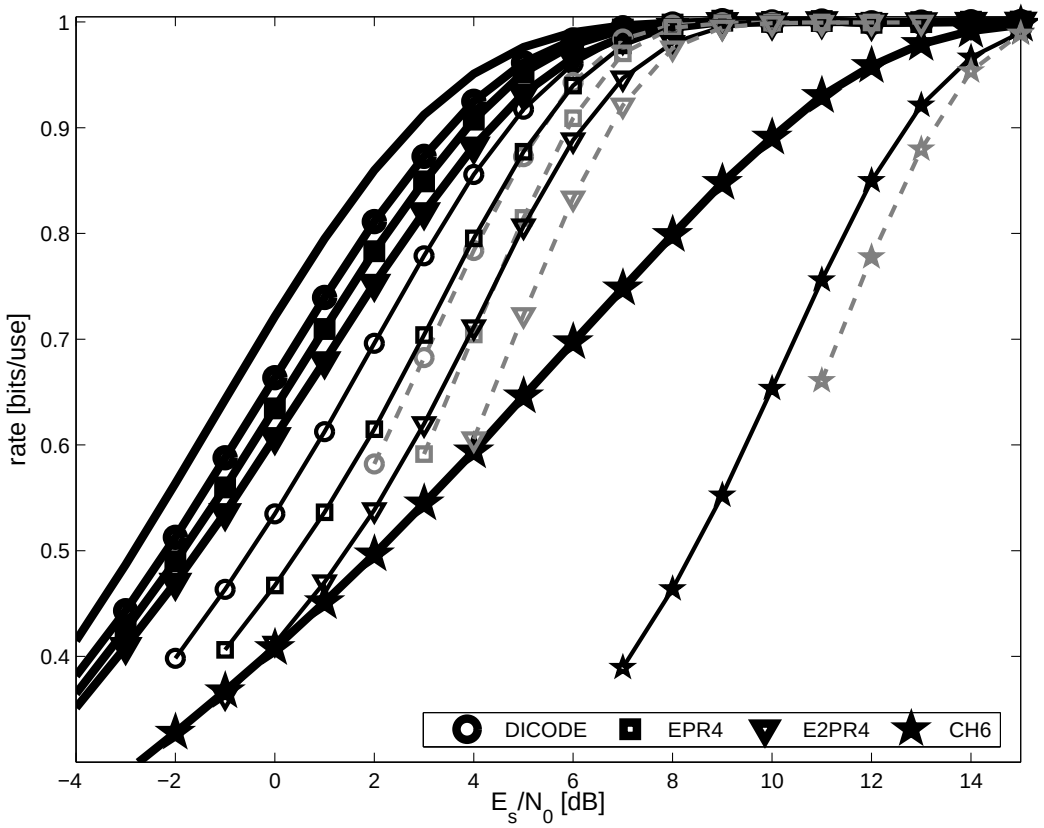



Figure 4.14: Comparison of the tree based ordering strategy for selected ISI channels (the channel coefficients can be found in table 4.2). Capacities are shown in bold and simulation results in solid lines. The different channels used are DICODE (circles), EPR4 (squares), E2PR4(triangles) and CH6(stars). Also shown for comparison is the performance of the $\left(1, \frac{1+D^2+D^3+D^5+D^6}{1+D^1+D^2+D^3+D^6}, 2048\right)$ RSC code with the tree based ordering strategy with transmission of parity bits first (grey, dashed lines).

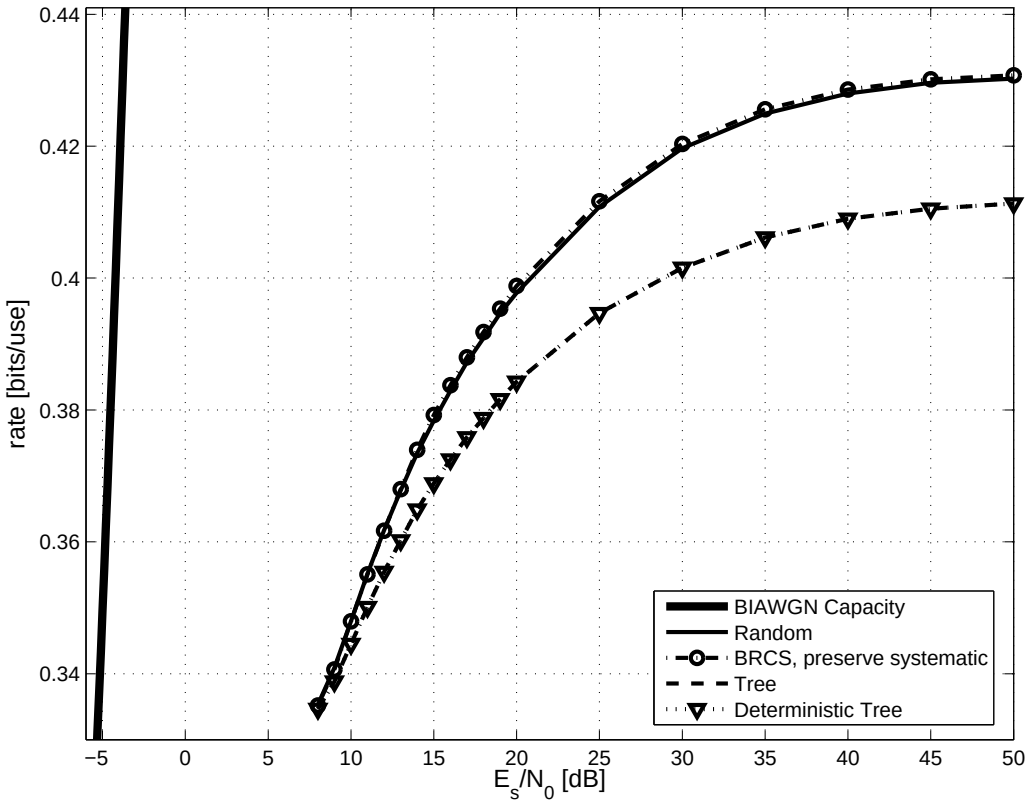
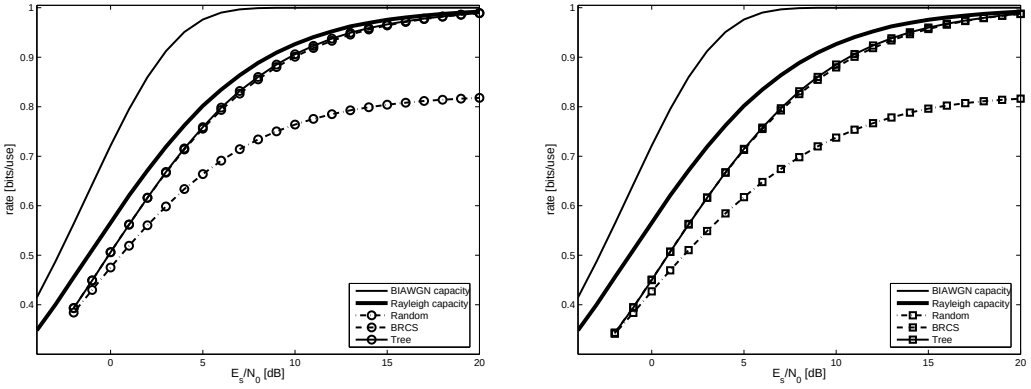



Figure 4.15: Performance of the Turbo Code ordering schemes over the unequalised DICODE channel (The used code was unable to recover the transmitted data over the EPR4, E2PR4 or CH6 channels). It is noteworthy that the two primarily random strategies, the random strategy (solid line) and the block-randomised approach (circles) outperform the more structured tree based approach (dashed) and the completely non-random deterministic tree (triangles).

Equalisation is essential for error-free transmission of data. For all but the DICODE channel, error-free recovery of the transmitted data was not possible, but even for the DICODE channel, performance, shown in Figure 4.15, remains very poor. It is interesting to note, however, that the tree ordering performs worse than the block-randomised ordering strategy and that the purely random ordering strategy outperforms the other ordering strategies. Given the poor performance altogether and thus the high number of bits that are transmitted, it is difficult to pinpoint a reason to why this could be the case. A possible explanation is the nature of the DICODE channel having memory 1 and thus bits that are 2 positions apart are not as effective as other combinations.

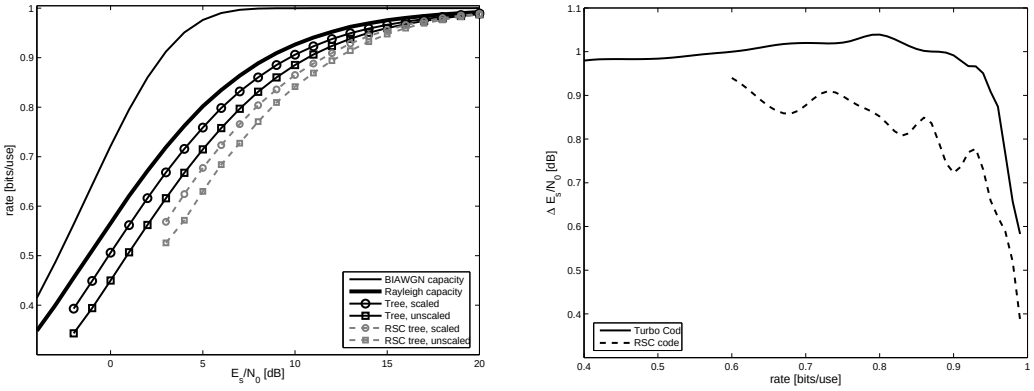
## 4.5 The Rayleigh Fading Channel

In section 3.5, the performance of RSC codes over a Rayleigh channel was evaluated. Similar to RSC codes, the performance of the ordering strategies, relative to each other, is not affected. Figure 4.16 shows the performance of Turbo Codes over the Rayleigh fast-fading channel, given total knowledge (Figure 4.16.1) or total ignorance (Figure 4.16.2) of the fading factor, $a$. Figure 4.17 compares the scaled and unscaled case for the tree-based ordering strategy and compares it to the RSC performance. Remarkably, the ignorance of the scaling factor causes a performance penalty of about 1 dB for Turbo and RSC codes (Figure 4.17.2). However, the Turbo Code maintains this advantage over a greater range. Due to saturation, the difference cannot be reliably determined as the code reaches rate $r \approx 1$, the maximum rate possible for BPSK signalling. Overall, the performance over the Rayleigh fast-fading channel is much closer to the theoretical capacity limit than it is the case for inter-symbol interference codes. This is to be expected due to the interleaving used in Turbo Codes. This makes them able to handle a single high-severity noise input better than constant interference. This single high-severity noise sample occurs for fast-fading Rayleigh channels when the fading parameter, $a$, is low such that little signal energy reaches the receiver.




(1) Scaled (2) Unscaled

Figure 4.16: Comparison of the performance of the random (dash-dot), block-randomised (dashed) and tree-based (solid) ordering strategies over a flat-fading Rayleigh channel.




(1) Performance comparison (2) Scaling gain

Figure 4.17: Comparison of the performance of the tree-based ordering strategy over the Rayleigh channel with perfect knowledge (circles) and ignorance (squares) of the scaling factor the scaled, $a$ (Figure (1)) and the scaling gain obtained through perfect knowledge of $a$ (Figure (2)). The RSC code performance is shown in both Figures as a reference as well (greyed in Figure (1) and dashed in Figure (2)).

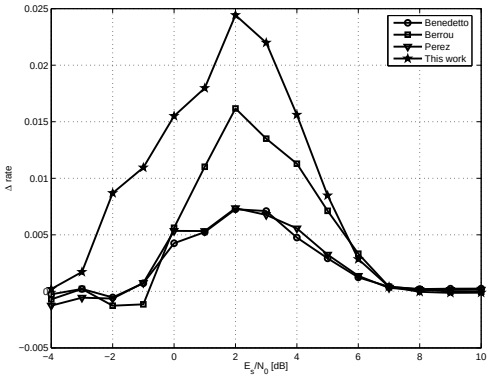
| Source | Memory | Generators |
|---|---|---|
| Benedetto *et al.* [9] | 4 | $g^{(0)} = 1 + D^1 + D^4$<br>$g^{(1)} = 1 + D^1 + D^3 + D^4$ |
| Benedetto *et al.* [9] | 6 | $g^{(0)} = 1 + D^1 + D^4 + D^5 + D^6$<br>$g^{(1)} = 1 + D^3 + D^4 + D^6$ |
| Berrou *et al.* [12] | 4 | $g^{(0)} = 1 + D^1 + D^2 + D^3 + D^4$<br>$g^{(1)} = 1 + D^4$ |
| Perez *et al.* [93] | 4 | $g^{(0)} = 1 + D^3 + D^4$<br>$g^{(1)} = 1 + D^1 + D^2 + D^4$ |
| section 3.3.1 | 4 | $g^{(0)} = 1 + D^1 + D^2 + D^4$<br>$g^{(1)} = 1 + D^3 + D^4$ |
| section 3.1 | 6 | $g^{(0)} = 1 + D^1 + D^2 + D^3 + D^6$<br>$g^{(1)} = 1 + D^2 + D^3 + D^5 + D^6$ |

Table 4.3: Selected generator polynomials for Turbo Codes.

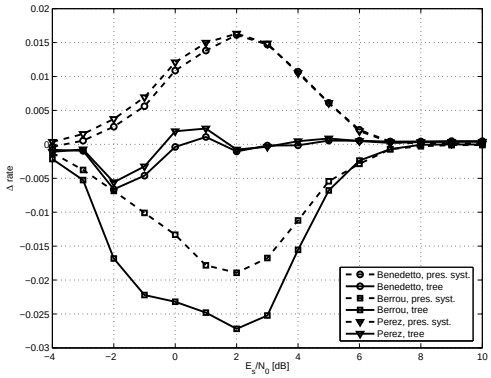
## 4.6 Generator Polynomials

In section 4.1, the generator polynomials of the RSC code from section 3.1, were used as the constituent encoders in order to compare the performance of the Turbo and RSC codes better. These polynomials, $g^{(0)} = 1 + D^1 + D^2 + D^3 + D^6$, $g^{(1)} = 1 + D^2 + D^3 + D^5 + D^6$, were not selected with regard to their optimality in Turbo Codes. In their original paper that introduced Turbo Codes, Berrou *et al.*[12] use $g^{(0)} = 1 + D^1 + D^2 + D^3 + D^4$, $g^{(1)} = 1 + D^4$. Other options are possible, though. Benedetto and Montorsi[9] propose $g^{(0)} = 1 + D^1 + D^4$, $g^{(1)} = 1 + D^1 + D^3 + D^4$ for $n = 4$ and $g^{(0)} = 1 + D^1 + D^4 + D^5 + D^6$, $g^{(1)} = 1 + D^3 + D^4 + D^6$ for $n = 6$, while Perez, Seghers and Costello[93] propose $g^{(0)} = 1 + D^3 + D^4$, $g^{(1)} = 1 + D^1 + D^2 + D^4$. Table 4.3 summarises the selected generator polynomials.

It has been suggested by Banerjee, Vatta, Scanavino and Costello, that nonsystematic Turbo Codes have desirable BER performance [6]. However, the systematic bits form an essential part of the Turbo Code and are of particular interest when puncturing Turbo Codes (see section 4.2). In fact, Banerjee *et al.* require *some* systematic bits in order to ensure convergence at the decoder. Nonsystematic constituent codes are therefore not considered in this thesis.




(1) Performance gain of tree-based ordering.

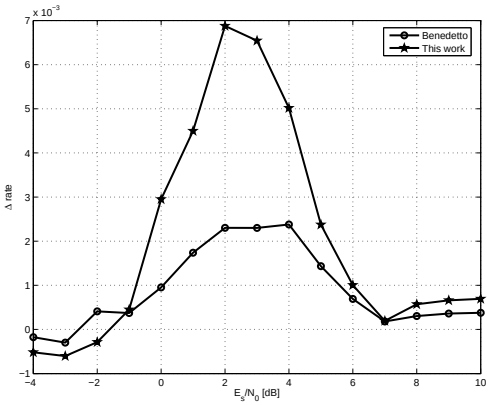



(2) Performance difference of selected generators with respect to the generators used in section 3.3.1.

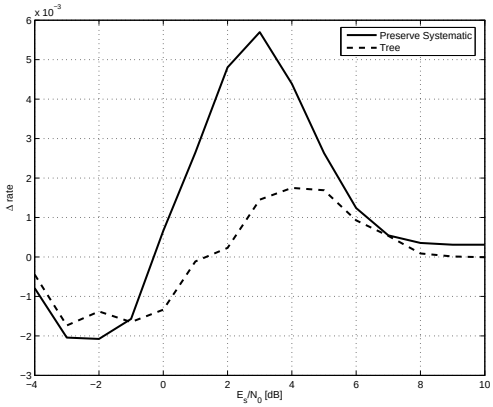
Figure 4.18: Comparison of the performance of the generator polynomials with memory $n = 4$ suggested by Benedetto *et al.* [9] (circles), Berrou *et al.* [12] (squares) and Perez *et al.* [93] (triangles). Figure (1) compares the performance gain obtained by the tree-based ordering strategy over the block-randomised ordering strategy with preservation of systematic bits. Figure (2) show the performance difference of the proposed generators to those used in section 3.3.1. 1000 packets were simulated.

Figure 4.18 compares the performance of the proposed pairs of generator polynomials of memory $n = 4$. Two observations can be drawn. Firstly, the *rate* performance of the Turbo Code depends only little on the employed generator polynomial pair. The maximum rate difference is $< 0.02$ which is $\approx 2\%$. The block-randomised ordering strategy seems to be more sensitive to the generator polynomial. If the tree-based ordering strategy is employed, the difference between the generator polynomial pairs becomes smaller (with the exception of the generator polynomials used by Berrou *et al.*). The second observation that can be made is that the tree-based ordering strategy outperforms the block-randomised approach only for slightly higher SNR values. At low SNR values both strategies perform about equally.

The same observations that were made for the generator polynomials with memory $n = 4$ can be made when considering the generator polynomials proposed by Benedetto *et al.*[9] ($g^{(0)} = 1 + D^1 + D^4 + D^5 + D^6$, $g^{(1)} = 1 + D^3 + D^4 + D^6$) and the one used during the development of the ordering strategies in section 3.1, $g^{(0)} = 1 + D^1 + D^2 + D^3 + D^6$, $g^{(1)} = 1 + D^2 + D^3 + D^5 + D^6$. The difference between the two generator polynomial pairs is $< 10^{-2}$. Notably, the difference between the tree-based ordering and the block-randomised approach *also diminished* to $< 10^{-2}$. This effect was only observed for




(1) Performance gain of tree-based ordering.




(2) Performance difference of selected generators with respect to the generators used in section 3.1.

Figure 4.19: Comparison of the performance of the generator polynomials with memory $n = 6$ suggested by Benedetto *et al.* [9] and section 3.1. Figure (1) compares the performance gain obtained by the tree-based ordering strategy over the block-randomised ordering strategy with preservation of systematic bits. Figure (2) show the performance difference of the proposed generators to those used in section 3.1. 1000 packets were simulated.

Turbo Codes. For RSC codes, the difference between the tree-based ordering and the block-randomised approach remained almost unchanged (see section 3.3.1). This indicates that the choice of generator polynomials for Turbo Codes can be made independently of the choice of the ordering strategy. This allows the reuse of existing generator polynomials with proven optimality conditions with at worst a minimal reduction in performance.

## 4.7 Comparison with Rate-Compatible Puncturing Tables

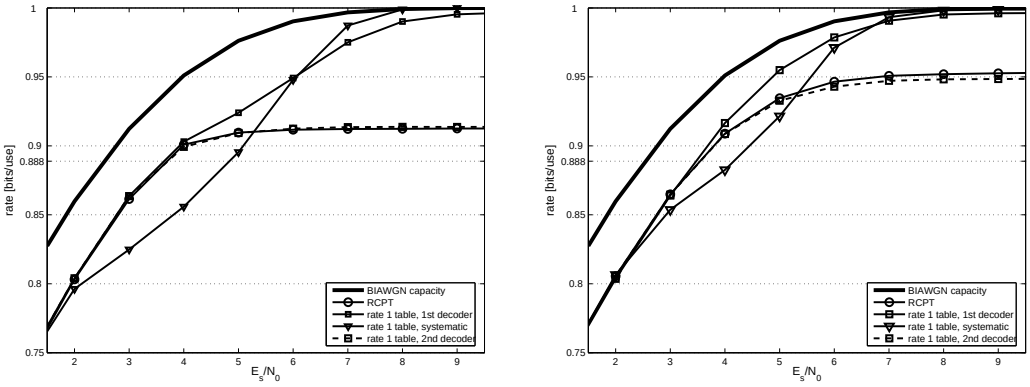
Rate-compatible punctured convolutional codes were introduced by Hagenauer[46]. While originally designed for RSC codes, the principle applies to Turbo Codes equally well (see section 2.6.3 for an introduction to rate-compatible puncturing). Rowitch and Milstein[109] proposed a series of rate-compatible puncturing table with puncturing period $P = 8$ for the codes of memory $n = 4$ proposed by Benedetto *et al.*[9],

| **Type of code** | **Punctured Code Rate** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\frac{8}{9}$ | $\frac{8}{10}$ | $\frac{8}{11}$ | $\frac{8}{12}$ | $\frac{8}{13}$ | $\frac{8}{14}$ | $\frac{8}{15}$ | $\frac{8}{16}$ | $\frac{8}{17}$ | $\frac{8}{18}$ | $\frac{8}{19}$ | $\frac{8}{20}$ | $\frac{8}{21}$ | $\frac{8}{22}$ | $\frac{8}{23}$ | $\frac{8}{24}$ |
| Benedetto *et al.* | 376 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 |
| proposed in [9] | 002 | 002 | 002 | 042 | 052 | 052 | 252 | 252 | 253 | 253 | 253 | 353 | 373 | 373 | 377 | 377 |
| $s_{\mathbf{x}} = 1024$ | 001 | 001 | 011 | 011 | 011 | 051 | 051 | 071 | 071 | 073 | 173 | 173 | 173 | 177 | 177 | 377 |
| Berrou *et al.* | 376 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 |
| proposed in [12] | 002 | 002 | 006 | 006 | 016 | 016 | 016 | 056 | 076 | 076 | 076 | 176 | 177 | 177 | 377 | 377 |
| $s_{\mathbf{x}} = 1024$ | 002 | 002 | 002 | 006 | 006 | 016 | 036 | 036 | 036 | 037 | 137 | 137 | 137 | 177 | 177 | 377 |
| Berrou *et al.* | 376 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 |
| proposed in [12] | 002 | 002 | 006 | 006 | 016 | 016 | 016 | 056 | 076 | 076 | 276 | 276 | 376 | 376 | 377 | 377 |
| $s_{\mathbf{x}} = 4096$ | 004 | 004 | 004 | 014 | 014 | 034 | 074 | 074 | 074 | 174 | 174 | 374 | 374 | 376 | 376 | 377 |
| Perez *et al.* | 376 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 | 377 |
| proposed in [93] | 020 | 020 | 024 | 024 | 224 | 224 | 264 | 264 | 265 | 265 | 265 | 275 | 375 | 375 | 377 | 377 |
| $s_{\mathbf{x}} = 1024$ | 020 | 020 | 020 | 024 | 024 | 224 | 224 | 264 | 264 | 274 | 374 | 374 | 374 | 375 | 375 | 377 |

Table 4.4: Octal representation (see section 2.3.5) of selected rate-compatible puncturing tables as proposed by Rowitch and Milstein[109].

Berrou *et al.*[12] and Perez *et al.*[93], respectively (see table 4.3 in section 4.6). In section 4.2, the transmission priority of systematic and parity bits was investigated with the result that systematic bits should not be punctured under any circumstances. Rowitch and Millstein, however, do propose puncturing tables (replicated in table 4.4) that do puncture systematic bits in order to allow parity bits to be allocated to both decoder. The tables proposed by Rowitch and Milstein closely follow the rate-compatibility approach proposed by Hagenauer, insofar as the maximum achievable rate is $\mathring{r} = 8/9$. The consequence of this is that the allocation of the first $s_{\mathbf{x}} \cdot 9/8$ bits lacks a good definition. Consider Figure 4.20. The performance of the rate-compatible puncturing tables is evaluated by using a binary search to determine the minimum number of bits required for transmission, like it is done in section 4.1. In order to make the rate-compatible puncturing tables compatible with this approach, algorithm 2.1 is used to turn the series rate-compatible puncturing tables into a transmission ordering $\prod$.

The tables, as proposed by Rowitch and Milstein, lack a table that achieves rate $\mathring{r} = 1$. It can be clearly seen from Figure 4.20 that this incurs a performance penalty. Due to the ambiguity of placing the first $s_{\mathbf{x}}$ bits, the proposed tables only achieve a maximum




(1) unrandomised performance (2) performance with randomisation

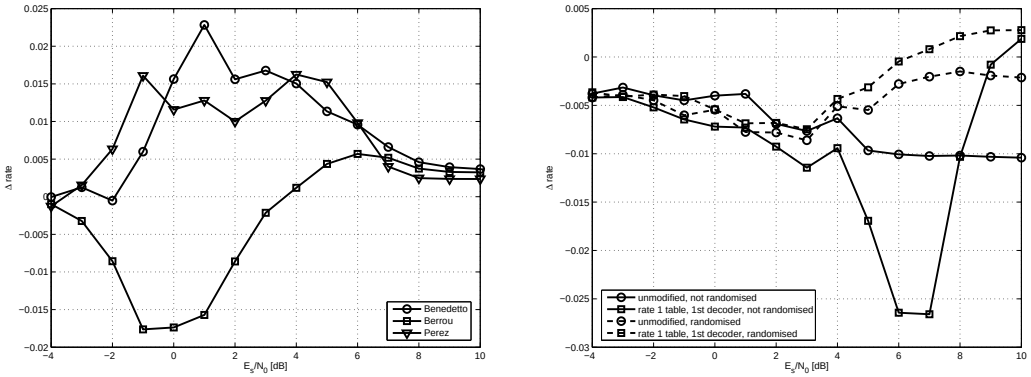
Figure 4.20: Comparison of the performance of the rate-compatible puncturing tables for the generator polynomials with memory $n = 4$ suggested by Berrou *et al.* [12]. Figure (1) shows the performance without randomisation, Figure (2) shows the performance with randomisation in algorithm 2.1. The tables, as proposed by Rowitch and Milstein[109], lack a table that achieves rate $r = 1$. The resulting uncertainty in transmission of the first bits shows as a performance penalty, and the table as proposed (circles) only obtain $r_{\max} = 0.91$ (randomisation improves this to $r_{\max} = 0.95$). Also shown are three possible modifications to obtain a rate $\mathring{r} = 1$ table: transmitting the systematic bits first (triangles), transmitting some parity bits of the first decoder (squares, solid) and transmitting some parity bits of the second decoder (squares, dashed). 1000 packets were simulated.

rate of $r_{\max} = 0.91$. If the bits are randomised, as suggested in section 2.6.4, the performance improves to $r_{\max} = 0.95$. In order to compare the performance of the rate-compatible puncturing tables to the ordering strategies in a fair manner, a table of rate $r = 1$ needs to be provided. Three alternatives are possible. The first two are simply the removal of one of the constituent decoders, i.e. taking the rate $\mathring{r} = 8/9$ table and setting either the second or third row to 000. The third possibility is to use a table of all systematic bits for $\mathring{r} = 1$ and add the first and then the second constituent decoder.[6] These three approaches are also shown in Figure 4.20. The results confirm what is already known about Turbo Codes. For the case of the systematic rate $\mathring{r} = 1$ table, a performance penalty is paid due to the lack of parity bits at the second decoder. Once more bits are available, and the second decoder starts working, performance improves. Allocating bits to the second decoder alone, while not ensuring that all bits receive at least either a systematic or a parity bits, leads to a diminished maximum rate, similar to the observations made in Figure 4.9.3, section 4.1.3. Allocating bits to the first decoder allows a maximum rate of $r_{\max} \approx 1$, at the cost of a reduced performance at high SNR. This performance cost is also observed by Garg and Adachi[40] when comparing type-II hybrid ARQ schemes based on rate-compatible punctured Turbo Codes. It has to be pointed out however, that randomisation largely mitigates this performance penalty.

Randomisation plays an important role for the performance of rate-compatible tables at high SNR. It can be readily observed from Figure 4.20, that randomisation improves performance. An even stronger argument in favour of randomisation is obtained by considering the effect of the input blocksize $s_{\mathbf{x}}$ in Figure 4.21.2. At high SNR, the performance of the rate-compatible puncturing tables *decreases*. This is due to the fact that the puncturing period, $P$, remains constant. An increase in the blocksize, $s_{\mathbf{x}}$, thus results in a larger number of bits punctured by each one in the puncturing table. Conversely, the individual order of the bits is less well defined and the bits are naively sent in the order in which they occur before puncturing. This is a similar effect that can be observed with the deterministic version of the binary tree ordering strategy (see section 4.1.4). Using randomisation can overcome this problem and largely mitigate the effects of increasing the blocksize.

Rate-compatible punctured-Turbo Codes that use optimised rate-compatible puncturing tables can perform better than the tree-based ordering strategy, as can be seen

[6] It is of course also possible to add the decoders the other way around, but this is not investigated due to the poor performance of the bits of the second decoder as seen in Figure 4.9.




(1) comparison to tree (2) blocksize

Figure 4.21: Performance difference of the rate-compatible punctured Turbo Codes compared to the tree-based ordering strategy (Figure (1)) and of the $s_{\mathbf{x}} = 1024$ and $s_{\mathbf{x}} = 4096$ versions of the puncturing tables proposed by Rowitch and Milstein[109] for the generators used by Berrou *et al.*[12].

in Figure 4.21.1. In case of the generator polynomials used by Berrou, Glavieux and Thitimajshima[12], $g^{(0)} = 1 + D^1 + D^2 + D^3 + D^4$, $g^{(1)} = 1 + D^4$, optimised rate-compatible puncturing tables do indeed provide a significant advantage over the tree-based strategy. Berrou *et al.* make no mention as to why they used that particular pair of generator polynomial. Indeed, when using a pair of generator polynomials that satisfy some optimality consideration, such as the ones proposed by Benedetto *et al.* or Perez *et al.*, the tree-based approach is superior even to optimised rate-compatible puncturing tables. Due to the comparatively coarse granularity of rate-compatible puncturing tables, however, the tree-based approach is always desirable at high-SNR levels.

The performance increase of the tree-based ordering strategy for the generator polynomials proposed by Benedetto *et al.* and Perez *et al.* seems insignificant. However, the rate-compatible puncturing tables suggested by Rowitch and Milstein[109] are *optimised to a particular pair of generator polynomials*. The tree-based approach, on the other hand, is a generic approach that is agnostic of the used polynomial and is not only experiencing only a slight performance penalty for the generator polynomials proposed by Berrou *et al.* but is also able to achieve just as much performance *gain* for the polynomials proposed by Benedetto *et al.* and Perez *et al.*, respectively.

## 4.8 Concluding Remarks

In this chapter the performance of Turbo Codes was considered. It is well known that Turbo Codes offer a significant performance increase over RSC codes and section 4.1 showed, that this performance advantage is maintained over the entire adaptive region, until both Turbo Codes and RSC codes saturate at $r_{\max} \approx 1$. It was demonstrated that the results from chapter 3 carry over to turbo codes insofar that when designing and ordering strategy, the structure of Turbo Codes needs to be taken into account and simple truncating or randomisation schemes do not perform well. Using the novel construction methods developed in chapter 3, it was shown that the best performing ordering strategy for Turbo Codes is the tree-based ordering strategy. However, the performance gain of the tree-based ordering strategy is not as pronounced as it was the case for RSC codes and only marginal gains are achieved by the tree-based ordering strategy over the block-randomised strategy. This difference between the strategies decreases further if the memory of the encoder is increased. Remarkably, for a memory $n = 6$ encoder, a deterministic version of the tree-based ordering strategy that is constructed without randomisation of the individual blocks performs almost identically to the tree-based strategy with randomisation. This could potentially simplify the design of a transmission system somewhat. Since the deterministic version of the tree ordering strategy does not use any randomisations, it is only dependent on the blocksize $s_{\mathbf{x}}$, the order $n$ and the rate, $r$, of the code, it can be implemented in advance for any transmission system, in which the aforementioned parameters are constant. Thus the need to maintain synchronous random number generators at the transmitter and receiver is no longer necessary. The Turbo Code has thus shown a certain robustness to the individual selection of bits, as long as they are spread over the length of the code.

While the choice of parity bits is less important for Turbo Codes, they are very sensitive to the lack of systematic bits. RSC codes offered the possibility to select the systematic or parity bit for each input bit at random, albeit with a slight performance degradation. Turbo codes, on the other hand, require that the systematic bits are transmitted first. Any other selection of the first bits causes either a lower maximum rate, $r_{\max}$, or a penalty in adapting to the channel. The best performance is obtained by transmitting all systematic bits first and supply both decoders with an equal amount of parity bits thereafter. It was demonstrated empirically, that the omission of systematic bits in favour of parity bits proposed by Rowitch and Milstein for RCPT codes in [109] is due

to the low granularity of the puncturing tables and not inherent to Turbo Codes.

Compared to RCPT codes, the tree-based ordering strategy performs slightly better, even if sufficient care is taken to turn the rate-compatible puncturing tables into an ordering permutation. In particular, randomisation is essential for RCPT codes in order to come close to the performance of the tree-based strategy. While, for the generator polynomial pair proposed by Berrou *et al.*, the optimised rate-compatible puncturing tables outperform the tree-based ordering strategy, the situation is reversed for the optimal generator polynomials proposed by Perez *et al.* and Benedetto *et al.*, respectively, where the tree-based approach is superior. However, it has to be mentioned that the tree-based ordering strategy is agnostic to the used generator polynomials whereas the rate-compatible puncturing tables were optimised for the respective code. Furthermore, at high SNR values, the rate-compatible puncturing tables prove to be too limited to allow for the fine grained control over the transmission order, required to achieve optimal performance.

Finally, the tree-ordering strategy performs very well for both inter-symbol interference and Rayleigh fading channels without requiring any special modification. The same observation can be made for every aspect of Turbo Codes considered. The input blocksize does not affect the rate performance and neither does the choice of generator polynomials. However, for some pairs of generator polynomials, a stronger gain of the tree-based approach was observed. The only property that noticeably affects the performance of the ordering strategies for Turbo Codes is the constraint length of the constituent encoders. The “ranking” of the ordering strategies does not change, but an increased encoder memory makes the Turbo Code increasingly robust towards the particular selection of bits in a given block.

In conclusion, Turbo Codes can easily be used for transmission systems that require an adaptive rate without significant modifications to an existing soft decoder. Due to the already random properties of Turbo Codes, an ordering scheme without randomisations can be employed as long as care is taken that the systematic bits, of paramount importance for Turbo Codes, are transmitted and that both constituent decoders are supplied with parity bits.

# Chapter 5

# Packetisation of Variable Rate Transmission

Hybrid-ARQ scheme with incremental redundancy can transmit at a variable rate. In order to work with optimal performance, two distinct issues of importance need to be addressed – which bits to transmit first and how large to make the individual segments that are sent to the receiver each time the receiver determines that it cannot successfully decode the data transmitted to it. Previously, chapter 3 has investigated a suitable ordering scheme for RSC codes, outlining the need for a structured approach, chapter 4 then extended this approach to Turbo Codes and outlined the additional constraints that need to be addressed when dealing with Turbo Codes. In these two chapters the performance was measured assuming that a 1-bit accuracy can be achieved. This allowed the development of the ordering strategies without requiring a particular packetisation strategy, essentially separating the problem of selection of bits, i.e. their ordering, from the number of bits in each transmitted packet.

This chapter addresses the other issue of hybrid-ARQ, thus far untreated in this thesis – how large should the individual packets be made in order to achieve good performance while, at the same time, only requiring a modest delay through the transmission of increment packets. First, **section 5.1** investigates the relationship between the size of multiple packets, in particular the relationship between the initial packet size and increment packets sizes, on the performance of a variable rate transmission system. It is shown, that even without 1-bit accuracy, a variable rate transmission system achieves

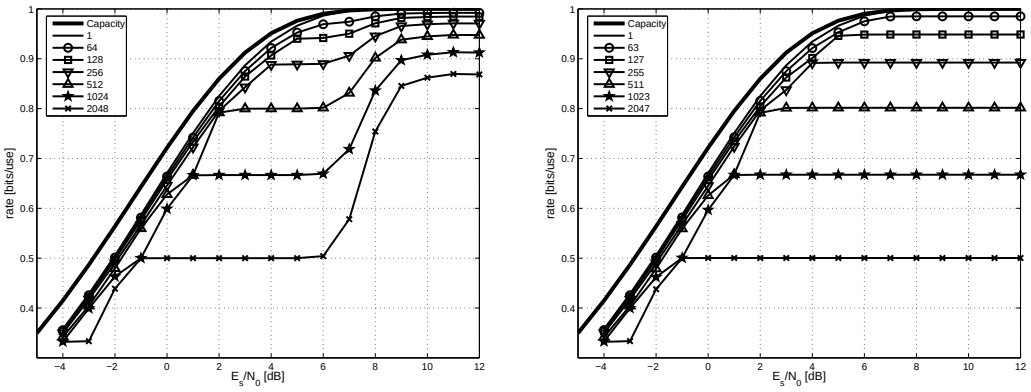
significant rate improvements. Fixed packet sizes, however, face a problem of excessive delay if chosen incorrectly. **Section 5.2** introduces two simple, yet effective, strategies for adapting the packet size to the channel conditions without the problem of excessive delay. The first strategy, merely aggregating packets of fixed sizes into larger packets performs very well for low channel SNR conditions. For use at high channel SNR, a mean based tracking algorithm is introduced that uses the average information from previous packets to select appropriate sizes for the initial and increment packets and comes within 0.5 dB of the 1-bit accurate version, while requiring $< 4$ increments. **Section 5.3** summarises the results, concluding this chapter.

## 5.1 Fixed Packet Size

Most protocols used for transmitting data packets contain the specification of a maximum payload and thus packet size. Any data that exceeds this maximum payload size needs to be split into a number of packets. It is beneficial to transmit at the maximum possible size, since this strategy minimises the overhead per packet. In a variable rate transmission system that tries to achieve efficient transmission by adjusting the rate of a channel code, it is important to consider these bounds. Consider Figure 5.1, which shows the performance of two types of packet sizes. In Figure 5.1.1, the (fixed) size of the individual packets is taken to be $s_{\text{packet}} = 2^n$. Thus a transmission of multiple packets is able to achieve a transmission rate of $r_{\max} = 1$, since $\sum_{n_{\text{packets}}} s_{\text{packet}} = s_{\mathbf{x}}$.[1] On the other hand, when selecting a packet size of $s_{\text{packet}} = 2^n - 1$ (shown in Figure 5.1.2), an additional packet needs to be transmitted. As an extreme, consider a packetisation of $s_{\text{packet}} = 2047$ for an input blocksize of $s_{\mathbf{x}} = 2048$. In order to transmit *at least* 2048 bits, 2 packets need to be transmitted, hence the number of transmitted bits, $n_{\text{sent}} = \sum_2 2047 = 4094$. Thus the maximum obtainable rate is limited to $r_{\max} = 2048/4094 \approx 0.5$.

Figure 5.2 compares all possible fixed packet sizes. The performance shown in Figure 5.2.1 exhibits a sawtooth pattern, outlined in Figure 5.2.2. For any given SNR, the performance degrades with increasing packet size, due to the need to transmit an additional packet when only few bits of that packet are actually needed. Once the size

[1] The Turbo Code does not achieve $r_{\max} = 1$ but only $r_{\max} \approx 1$. In this case, the Turbo Code needs, on average, 2050 bits instead of 2048, explaining the performance drop at high SNR for large packet sizes.



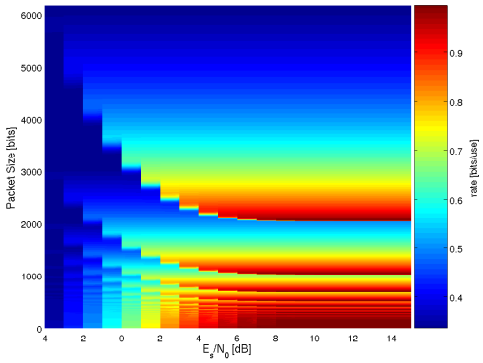

(1) Packet size $s_{\text{packet}} = 2^n$ (2) Packet size $s_{\text{packet}} = 2^n - 1$

Figure 5.1: Performance of two different strategies for splitting the input into packets. Figure (1) displays the performance of packets with a size of $s_{\text{packet}} = 2^n$, whereas Figure (2) displays the performance of packets with a size of $s_{\text{packet}} = 2^n - 1$.

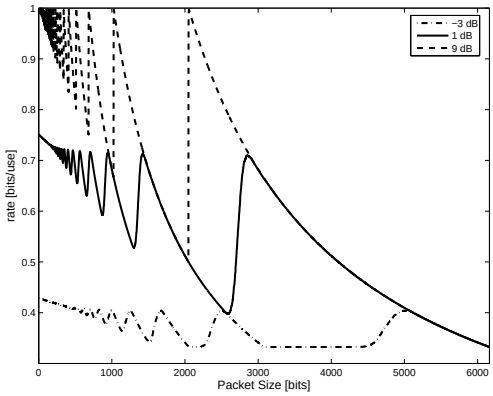
of the packet, $s_{\text{packet}}$, divides the average required number of bits, i.e. the packet size is enough to correct most errors without requiring a second packet, the performance increases significantly. A further increase of the packet size decreases the performance again. Naturally, the optimal size of the packet depends on the SNR of the channel, since the SNR determines the required number of bits for successful transmission.

### 5.1.1 Transmission of an initial packet

The performance of a fixed packetisation scheme, in which all packets have the same size, is highly dependent on the selected size (see Figure 5.1). Selecting an inappropriate size can lead to a significant performance degradation at high SNR, as shown in Figure 5.1.2 and Figure 5.2, in which a clear saw tooth pattern is visible. A solution comes naturally from the fact that there are clearly two separate goals that need to be achieved by a transmission system. The first goal, as the name suggests, is the *transmission* of some data. The second goal is the *protection* of this data. Clearly, the first goal, that of transmission, requires at least $s_{\mathbf{x}}$ bits to be sent, assuming that each bit is equiprobable. Any constellation of packet sizes that has not transmitted $s_{\mathbf{x}}$ bits clearly fails this goal. The goal of protection of the transmission is achieved by sending *additional* bits to *augment* the transmitted data.

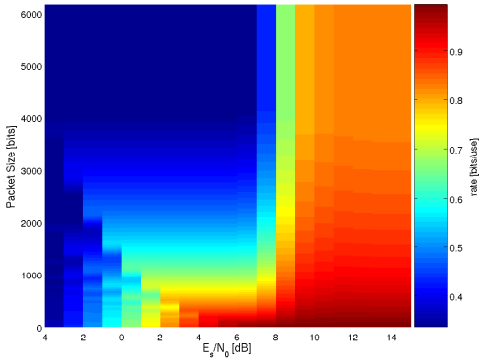



(1) Top view

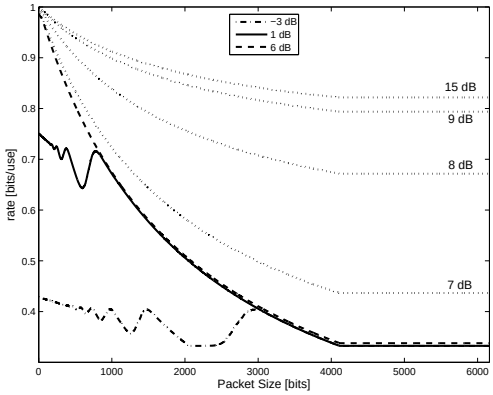



(2) Side view

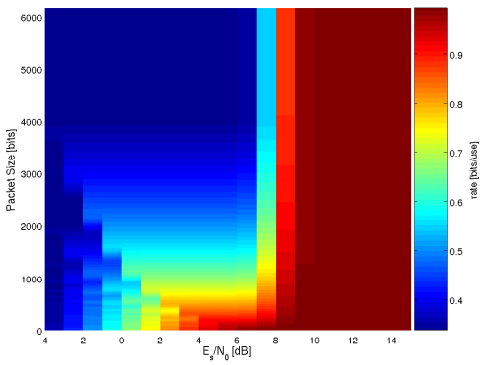
Figure 5.2: Performance of fixed packet sizes for a rate $r_{\text{base}} = 1/3$ Turbo Code under the tree-based ordering strategy.



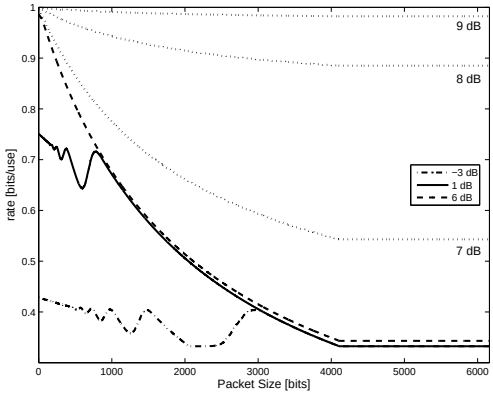

(1) $s_{\text{initial}} = 2048$




(2) $s_{\text{initial}} = 2048$, side view




(3) $s_{\text{initial}} = 2050$




(4) $s_{\text{initial}} = 2050$, side view

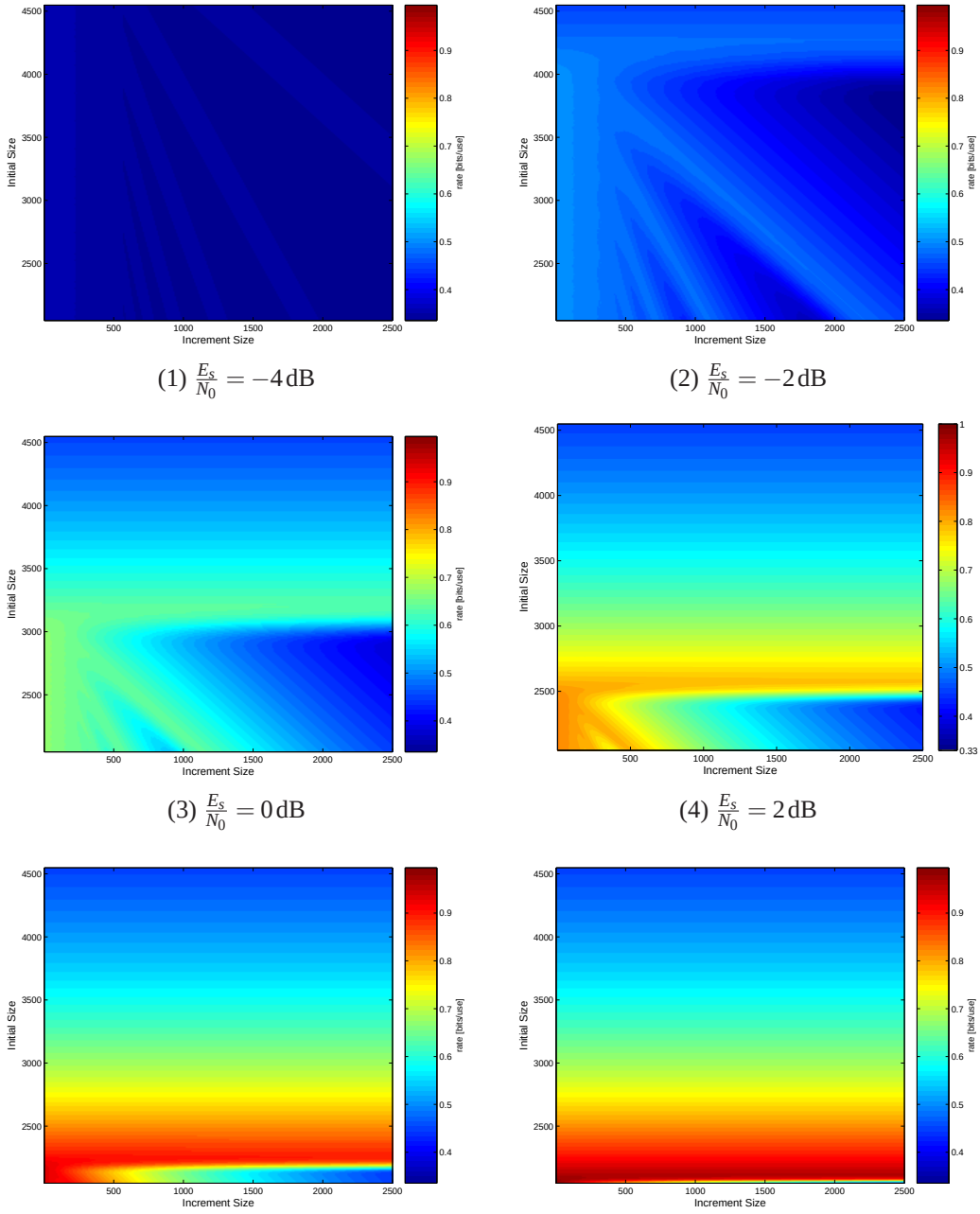
Figure 5.3: Performance of fixed increment packet sizes for a rate $r_{\text{base}} = 1/3$ Turbo Code under the tree-based ordering strategy. In Figures (1) and (2) an initial packet of $s_{\text{initial}} = 2048$ bits is transmitted, in Figures (3) and (4) an initial packet of $s_{\text{initial}} = 2050$ bits is transmitted.

Figure 5.3 shows the effect of transmitting an initial packet of $s_{\text{initial}} = 2048$ and $s_{\text{initial}} = 2050$. The initial packet solves the problem of the saw-tooth pattern, observed in Figure 5.2, for high SNR channels. However, once the channel quality starts to decay and the channel SNR becomes lower, the problem of selecting the right size still persists. For these low-SNR channel conditions, additional, protecting packets, called an *increment* packet needs to be transmitted. Similar to a fixed size for all packets, the size of the increment can lead to an over-correcting effect, if the remaining number of bits after accounting for the initial packet is only slightly larger than an integer multiple of the size of the increments, $s_{\text{inc}}$. This in turn requires an additional increment to be sent which requires more bits than fewer, larger increments. The effect of this over-correcting by the increment can be seen in Figures 5.3.1 and 5.3.3 as a sharp drop of performance when moving from high SNR on the right to low SNR on the left.

### 5.1.2 Initial and Increment Size

In a transmission system that does not determine packet sizes adaptively, two sizes need to be fixed: the size of the initial packet and the size of each increment. The distinction between initial and increment size is made for the simple reason, that, ideally, the initial size should be selected such that one initial packet and a small number of increment packets are able to ensure error-free transmission. This constraint, in turn, makes the size of the initial packet, $s_{\text{initial}}$ much larger than the size of the increment packets, $s_{\text{inc}}$, hence the distinction between the two packets.

Figure 5.4 shows the relationship between the size of the initial packet, $s_{\text{initial}}$, and the size of the increment packet, $s_{\text{inc}}$. The initial packet clearly has a limiting role, i.e. selecting an initial size such that the initial packet is always sufficient to correct the errors that occur on the channel essentially fixes the rate of the code. However, ensuring that the initial packet is always successful incurs a reduction of the achievable rate, since, obviously, the size needs to be set to correct even the worst case scenario. If the initial size, $s_{\text{initial}}$, is set low enough, the transmission of increment packets is required. From Figure 5.4 it can be seen that the optimal performance is achieved on several straight lines. These lines correspond to the transmission of multiple increments. If $n_{\text{req}}$ are required for successful transmission and $n_{\text{sent}} = s_{\text{initial}} + n_{\text{inc}} \cdot s_{\text{inc}}$ is the number of transmitted bits, then the performance will be best, if $n_{\text{sent}} \gtrapprox n_{\text{req}}$.

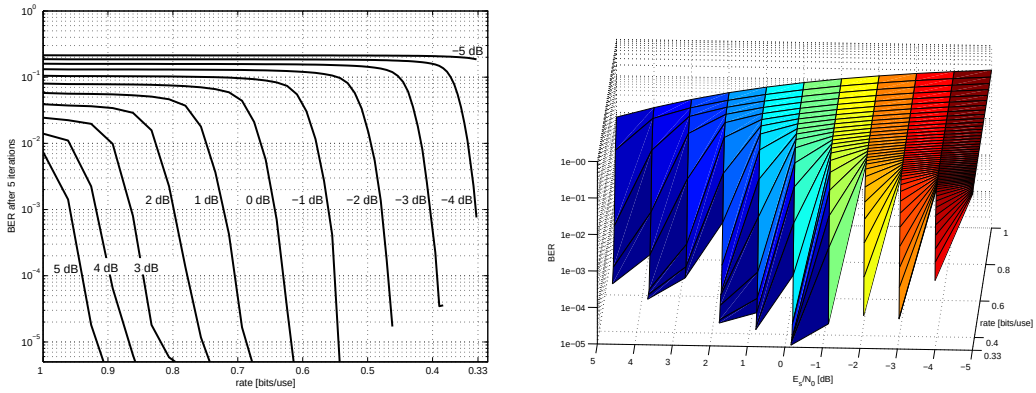



(1) $\frac{E_S}{N_0} = -4\,\text{dB}$ (2) $\frac{E_S}{N_0} = -2\,\text{dB}$

(3) $\frac{E_S}{N_0} = 0\,\text{dB}$ (4) $\frac{E_S}{N_0} = 2\,\text{dB}$

(5) $\frac{E_S}{N_0} = 4\,\text{dB}$ (6) $\frac{E_S}{N_0} = 6\,\text{dB}$

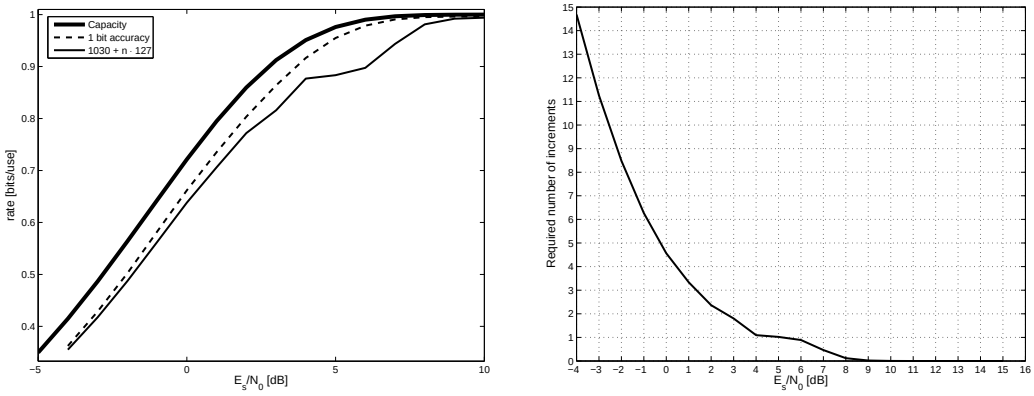
Figure 5.4: Performance of combinations of fixed initial and increment packet sizes for a rate $r_{\text{base}} = 1/3$ Turbo Code under the tree-based ordering strategy.




(1) flat view (2) 3D view

Figure 5.5: Bit-error rate performance vs. rate of the transmitted sequence, $\mathbf{w}$. 1000 packets were simulated. An increasing number of transmitted bits shows the same waterfall characteristics that Turbo Codes are known to experience with an increasing channel-SNR. Both waterfalls can be seen simultaneously in Figure (2).

Allowing the transmission of multiple increments creates the possibility that an additional increment "over-protects" the data packet and hence causes a performance penalty. This is seen in particular for the mid-range of SNR values (Figures 5.4.2, 5.4.3 and 5.4.4). However, for a reasonably small size of the increment size of $s_{\text{inc}} < 500$ bits, the individual lines of optimal performance seem to converge, such that only a small performance penalty is experienced. In other words, the granularity of increment packets with $s_{\text{inc}} < 500$ bits is good enough to achieve a reasonable performance. This argument is also supported by considering the bit-error rate with decreasing rate, shown in Figure 5.5. It is well known, that Turbo Codes experience a waterfall region with increasing channel SNR, causing the bit-error rate to drop significantly for comparatively small SNR improvements. The same behaviour can be observed not only with an increasing channel-SNR, but also with a decreasing rate, i.e. an increase in the number of transmitted bits, eventually leading to an error floor experienced by all Turbo Codes. It is because of this sharp drop in bit-error rate that the increment packets need to be relatively small. A large increment would cause the bit-error rate to drop to the error floor, where further improvement is almost insignificant.



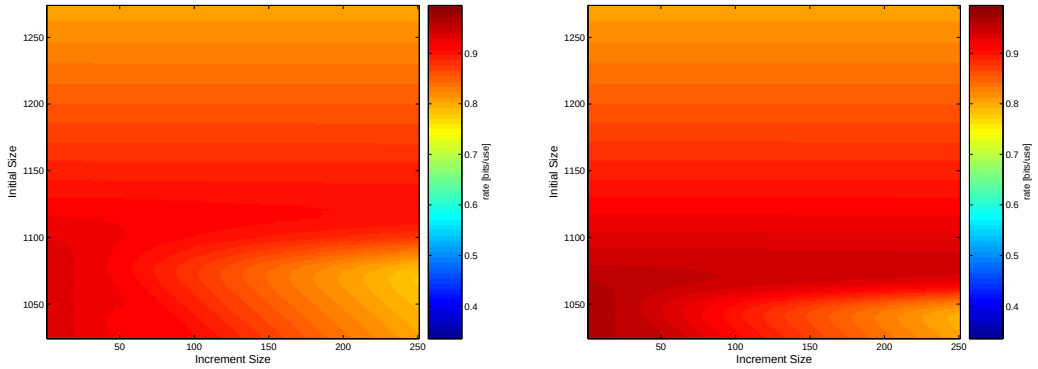

(1) Rate performance (2) Delay performance

Figure 5.6: Performance of packetisation of a rate-compatible punctured code with base rate, $r_{\text{base}} = 1/3$, an input blocksize of $s_{\mathbf{x}} = 2048$ and puncturing period $P = 8$. Due to the puncturing period and the requirement to transmit a rate $\mathring{r} = 1$ table, $s_{\text{initial}} = 1030$ (including 6 termination bits of the code) and $s_{\text{inc}} = 1030/8 \approx 127$. Thus $n_{\text{sent}} = 1030 + n_{\text{inc}} \cdot 127$. Figure (1) shows the rate performance of such a transmission system, Figure (2) shows the delay, measured as the number of transmitted increments, $n_{\text{inc}}$, of the system.

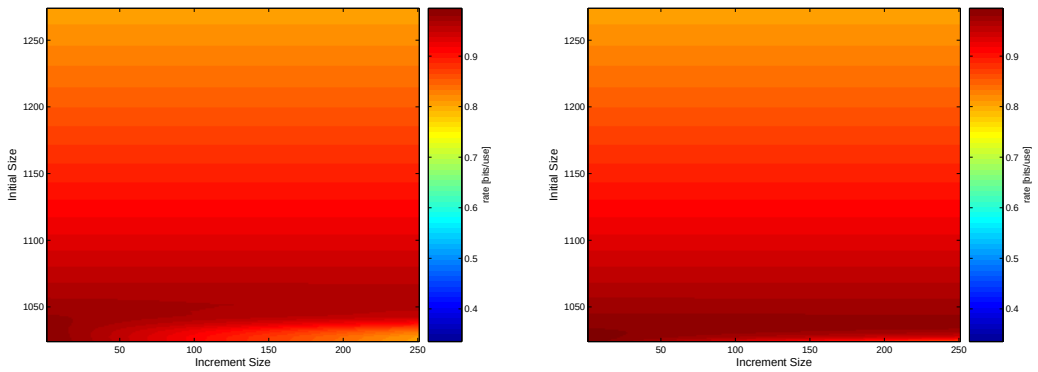
### Rate-Compatible Codes

In section 2.6.3 rate-compatible punctured codes where introduced, which have a straightforward initial and increment size, determined by the puncturing tables $\mathbf{P}_0$ and $\mathbf{P}_\Delta$, respectively. The optimised puncturing tables introduced by Rowitch and Milstein[109] are designed for an input size of $s_{\mathbf{x}} = 1024$ bits plus 6 termination bits, a base code rate $r_{\text{base}} = 1/3$ and have a puncturing period of $P = 8$. Therefore, the size of the initial packet is $s_{\text{initial}} = 1030$, the size of the increments are $s_{\text{inc}} = 1030/8 \approx 127$ bits.

The performance of such a transmission system is shown in Figure 5.6. At low SNR, the number of bits required for successful transmission, $n_{\text{req}}$, is relatively high and many increments need to be transmitted in order transmit this number. The delay of the transmission system this increases when the channel quality decreases (Figures 5.6.2). However, because many bits are transmitted, over-correcting with a packet of 127 bits, as it is done in Figure 5.6, does not negatively impact performance significantly. This is different when considering high-SNR channels. The delay of the hybrid-ARQ system is low, since only few packets are required to obtain $n_{\text{req}}$. However, the impact of over-correcting becomes more significant. In particular, consider the rate and delay




(1) $\frac{E_S}{N_0} = 4\,\text{dB}$ (2) $\frac{E_S}{N_0} = 5\,\text{dB}$




(3) $\frac{E_S}{N_0} = 6\,\text{dB}$ (4) $\frac{E_S}{N_0} = 7\,\text{dB}$

Figure 5.7: Performance close-up of possible combinations of fixed initial and increment packet sizes for a rate $r_{\text{base}} = 1/3$ Turbo Code with an input size of $s_{\mathbf{x}} = 1024$ under the rate-compatible ordering strategy. $s_{\mathbf{x}} = 1024$ was chosen to allow for a comparison with rate-compatible punctured codes.

performance shown in Figures 5.6.1 and 5.6.2, respectively, at 4 dB–6 dB. The delay performance *exhibits a plateau* between 4 dB and 6 dB. This indicates that the chosen packetisation scheme is too coarse and a reduction of $n_{\text{req}}$, due to an improving channel cannot be exploited by transmitting fewer increments. The rate performance (Figures 5.6) shows the corresponding effect. Between 4 dB and 6 dB the rate performance exhibits a similar plateau and performance only improves at higher channel SNR, where sometimes the initial packet, i.e. the protection by the 6 tail bits, is sufficient to protect the input.

The selection of an appropriate packetisation scheme becomes difficult once the channel quality becomes “almost good enough”. Figure 5.7 shows a close-up of the connection between initial size, $s_{\text{initial}}$, and increment size, $s_{\text{inc}}$. The overcorrection penalty

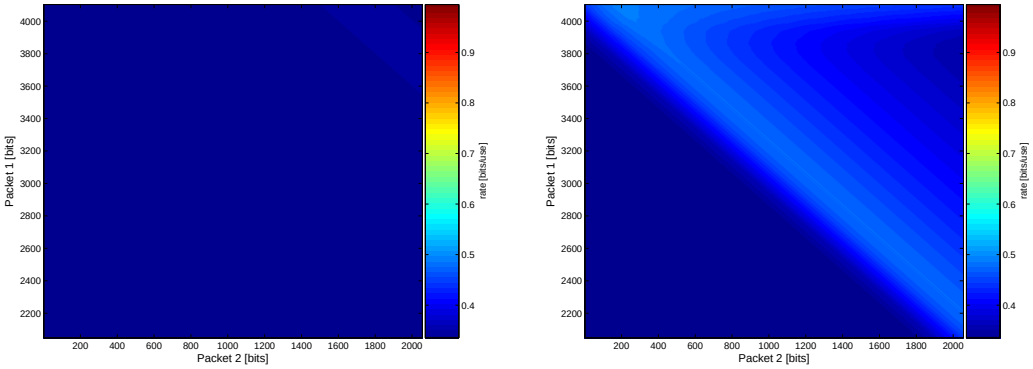
is a triangular shape that impacts performance for few initial bits with a large increment size. As the channel quality improves (Figures 5.7.2 and 5.7.3), this triangle moves closer towards low increment sizes until the packet sizes selected by the rate-compatible approach are affected by the over-correcting penalty. At 7 dB, the channel has become sufficiently good so that the selected initial size can correct some error patterns by itself. The rate performance thus starts to approach the theoretical performance of 1 bit accuracy.

### 5.1.3 Delay-limited Packetisation

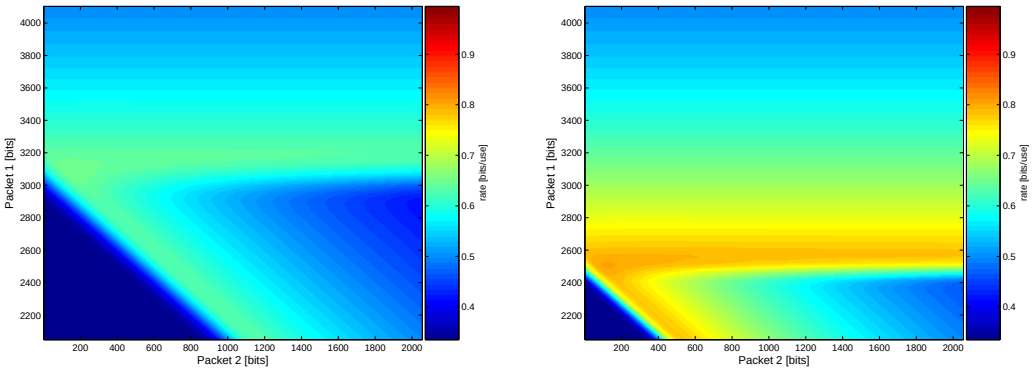
A major problem with incremental redundancy systems is the delay incurred by transmitting the increments, in particular for low-SNR channels (see Figure 5.6.2). If the delay constraint is known, it is possible to turn the last increment packet, i.e. the last packet that is sent before the delay constraint is reached, into a "catch all" packet. In this packet, instead of transmitting just a fraction of the parity data, *all remaining parity data* is transmitted to the receiver. For ease of presentation, consider the case where a transmission system is able to send 3 packets. The first of these packets will thus be the initial packet, with a size of at least the input size, $s_1 \geq s_{\mathbf{x}}$. The second packet will be an "ordinary" incremental packet, the third one will be the "catch-all" packet, such that $s_1 + s_2 + s_3 = s_{\mathbf{u}}$, where $s_{\mathbf{u}}$ is the size of the sequence produced by the encoder.

Figure 5.8 shows the resulting rate for the three packet sizes (only the first two packets need to be selected, since $s_3 = s_{\mathbf{u}} - (s_1 + s_2)$). A clear penalty of underestimating the required size of the first two packets can be observed. Whereas for fixed sizes with an arbitrary number of increments (see Figure 5.4) the delay increased significantly when underestimating $n_{\text{req}}$, the third packet massively over-corrects, leading to a penalty of the rate while not affecting the maximum delay.

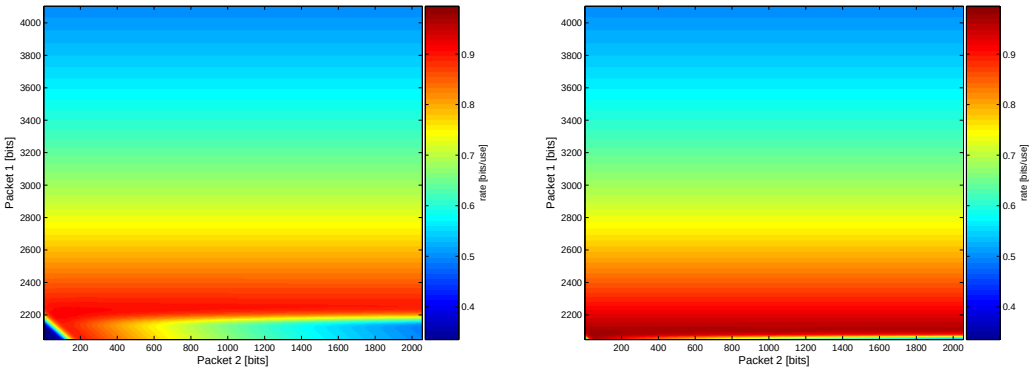
The optimum performance of a packetised transmission system with a limited delay is achieved when the initial packet can take care of most error patterns, with exceptional error patterns corrected by the non-last increments. The "catch-all" packet is then used as a last resort. This behaviour is not only desirable for the transmission system that is limited to 3 packets. In *any* transmission system it is beneficial to adjust the initial size to correct the average case with increments only dealing with exceptional cases. In fact, the transmission system with 3 packets does not perform much worse




(1) $\frac{E_S}{N_0} = -4\,\text{dB}$ (2) $\frac{E_S}{N_0} = -2\,\text{dB}$

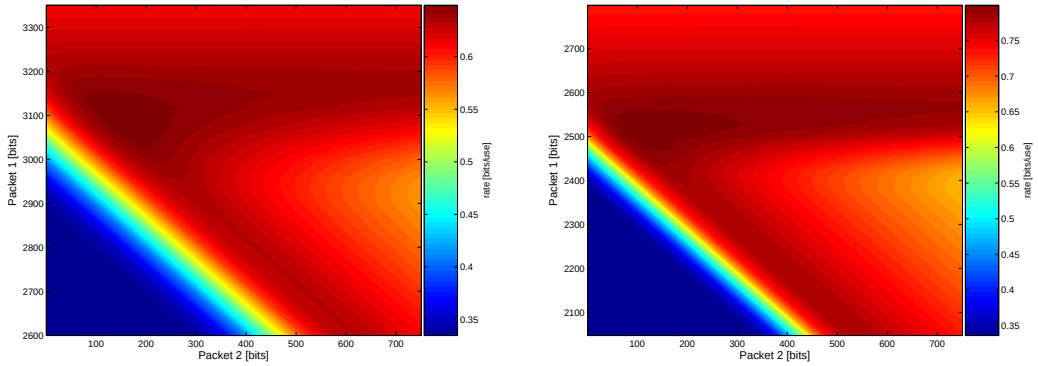



(3) $\frac{E_S}{N_0} = 0\,\text{dB}$ (4) $\frac{E_S}{N_0} = 2\,\text{dB}$




(5) $\frac{E_S}{N_0} = 4\,\text{dB}$ (6) $\frac{E_S}{N_0} = 6\,\text{dB}$

Figure 5.8: Performance of combinations of 3 packets for a rate $r_{\text{base}} = 1/3$ Turbo Code under the tree-based ordering strategy with an input blocksize of $s_{\mathbf{x}} = 2048$. The size of the third packet is calculated by subtracting the combined size of the first and second packet from the maximum number of bits, 6162. For ease of exposition, it is assumed that the first packet always contains, at a minimum, 2048 bits.




(1) $\frac{E_S}{N_0} = 0\,\text{dB}$ (2) $\frac{E_S}{N_0} = 2\,\text{dB}$

Figure 5.9: Close-up of the performance of possible combinations of 3 packets for a rate $r_{\text{base}} = 1/3$ Turbo Code under the tree-based ordering strategy with an input blocksize of $s_{\mathbf{x}} = 2048$. The size of the third packet is calculated by subtracting the combined size of the first and second packet from the maximum number of bits, 6162. For ease of exposition, it is assumed that the first packet always contains, at a minimum, 2048 bits.

than a transmission system which uses a fourth packet as the catch-all packet. This can be seen from the performance shown in Figures 5.3.2 and 5.3.4. In these Figures the local maxima correspond to a decreasing number of increments, $n_{\text{inc}}$. Since the performance of the last two maxima, corresponding to 2 and 1 increment, respectively, does not differ significantly, an additional increment, resulting in a total of 4 packets, is of little benefit to the transmission system.

## 5.2 Adaptive Packet Size

Knowledge about the effect of the transmission on the transmitted symbol frequently isn't available. In particular, sufficient information to determine the optimum sizes of the initial and increment packets are typically not available. However, a transmitter can use the information received on the feedback channel, namely if decoding at the receiver was successful or not, in order to adapt the packetisation for future packets. Such adaption serves to improve two aspects of the transmission system. Firstly, it allows the transmitter to select an appropriate initial packet size in order to reduce the number of increments, hence *reducing delay*. Secondly, an adaptive system can *reduce waste* that occurs due to excess bits in increment packets.

Consider Figure 5.9 which shows a close-up of the region of optimal rate for the transmission system with 3 packets, introduced in section 5.1.3 (note that the colours are overemphasised and reflect only the range of rates in the picture, not the entire spectrum from $\mathring{r} = 0.33$ to $\mathring{r} = 1.0$). The optimum constellation of packet sizes is clearly visible. In fact, the limitation to 3 packets provides a method to obtain this optimal rate. Due to the "catch-all" nature of the last packet that is transmitted, under-estimating the required number of bits, $n_{\text{req}}$, causes a significant performance penalty. A transmission system that increases the increments will thus first see an increase in performance, followed by the decrease due to over-correcting. Once this point is determined, this increment size can be added to the initial packet size, to come reasonably close to the optimum rate. From this point, the initial size is reduced again with a simultaneous increase in increment size, in order to obtain the optimal rate.

### 5.2.1 Aggregating Fixed Packets

The delay experienced by an incremental redundancy transmission system can be a significant problem (see Figure 5.6.2). If the size of the initial packet, $s_{\text{initial}}$, and the size of the increment packets, $s_{\text{inc}}$, have been fixed, a simple mechanism to reduce the delay is to aggregate the initial packet and several increments in the first packet of size $s_1 = s_{\text{initial}} + n_{\text{inc}} \cdot s_{\text{inc}}$, in the hope of saving the time required to transmit $n_{\text{inc}}$ increments. Of course, this is wasteful, if less than $n_{\text{inc}}$ increments need to be transmitted. The transmitter can then track the number of increments that were required to achieve successful transmission and set $n_{\text{inc}}$ accordingly for the next packet. By itself, this leads to the problem that the size of this first packet, $s_1$, only increases. Therefore, if the first packet is successful, the transmitter needs to reduce the number of increments aggregated into the first packet. A possible such adaption strategy is algorithm 5.1.

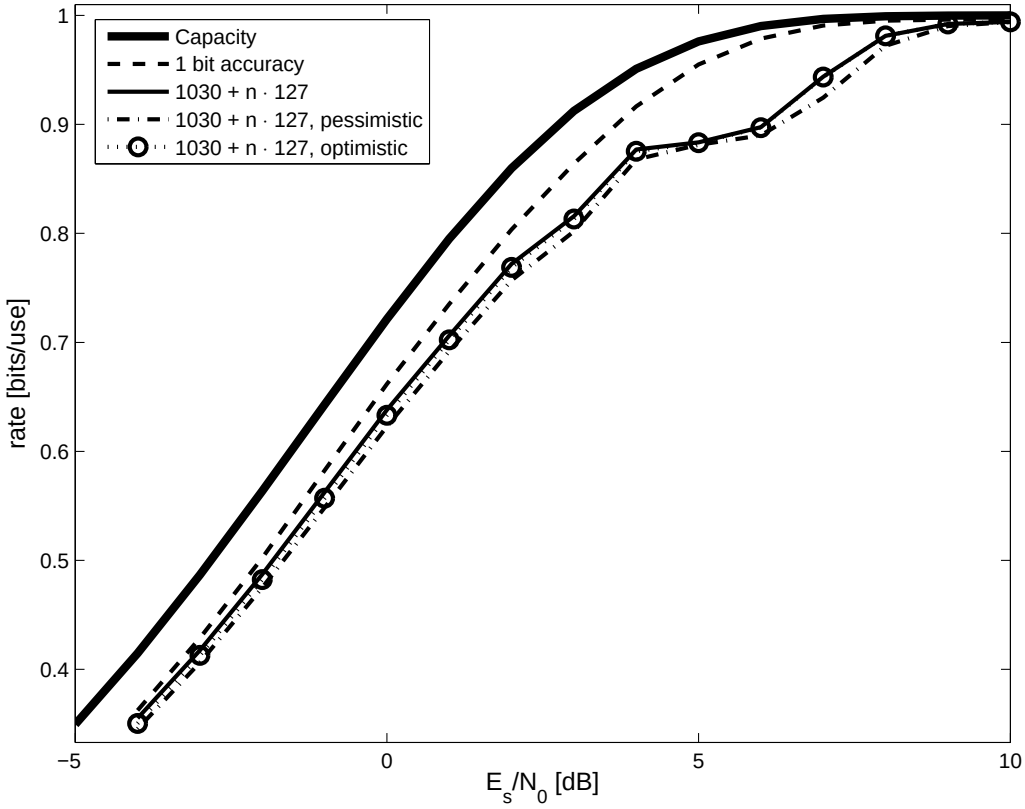
The strategy presented in algorithm 5.1 uses the information about the previous transmission to adjust the size for the next packet. This makes algorithm 5.1 suitable for not only adjusting to a channel, but also to track the progression of that channel, as long as the change in the channel is slow and gentle enough for the information about the last transmission to still be meaningful. Clearly though, the first data packet that is to be transmitted lacks a predecessor and no information is available for that packet. Therefore, the delay of this packet is potentially larger than that of the other packets.

**Algorithm 5.1** Channel adaption by packet aggregation.

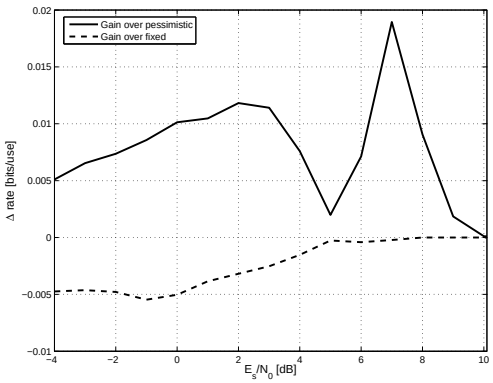
1: $n_{\text{bo}} = 0/1$ depending on selected algorithm parameters
2: $n_{\text{inc}} = 0$ or use channel state information to determine $n_{\text{inc}}$
3: **for** each packet to be transmitted **do**
4: Transmit packet 1 of size $s_1 = s_{\text{initial}} + n_{\text{inc}} \cdot s_{\text{inc}}$
5: **if** first packet is successful **then**
6: $n_{\text{inc}} = \max(n_{\text{inc}} - 1, 0)$
7: **else**
8: continue to transmit increments
9: let $n'_{\text{inc}}$ be the number of additional increments
10: $n_{\text{inc}} = n_{\text{inc}} + n'_{\text{inc}} - n_{\text{bo}}$
11: **end if**
12: **end for**

However, when transmitting a large amount of data, a priming packet or other channel state information can be used in order to establish an initial number of increments. One remark about algorithm 5.1 is necessary. In line 10, a back-off, $n_{\text{bo}}$, is used. This number determines the "optimism" of the algorithm, in the sense that it expects the noise observed on the channel to be better, i.e. less than it is now, thus reducing the number of bits for the next transmission ($n_{\text{bo}} = 1$). On the other hand, setting $n_{\text{bo}} = 0$ results in no reduction of transmitted bits, hence the algorithm expects the channel quality to be at least as bad (or worse, hence the pessimism) as it was presently observed. It is possible to set $n_{\text{bo}}$ to values different from 0 and 1. However, consider the ramifications. The aim of the adaptive algorithm is to send an initial guess and if that guess fails correct with hopefully only one increment. Thus setting $n_{\text{bo}} > 1$ only makes sense if it is known, that the channel produces error spikes and thus, if errors are observed, it can be argued that the next transmission is much better. This situation, however, is not considered in this thesis because the added complexity of error placement (i.e. which packet they affect) unnecessarily complicates the discussion of ordering schemes.

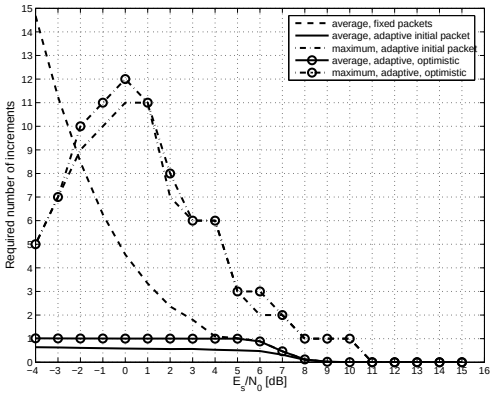
The performance of algorithm 5.1 is shown in Figure 5.10. Aggregating a number of increments with the initial packet to be transmitted first significantly reduces the *average* delay (Figure 5.10.3), while not having a significant impact on the rate performance (Figure 5.10.1). Figure 5.10.2 shows the difference in performance of the optimistic packaging strategy vs. the fixed packetisation strategy with the dashed line. The performance loss at its worst $\approx 0.005$ (at 1 dB) and decreases as the channel qual-




(1) Rate performance




(2) Optimistic performance gain




(3) Delay performance

Figure 5.10: Performance of adaptive packetisation according to algorithm 5.1 of a rate-compatible punctured code with base rate, $r_{\text{base}} = 1/3$, an input blocksize of $s_{\mathbf{x}} = 1024$ and puncturing period $P = 8$. Due to the puncturing period and the requirement to transmit a rate $\mathring{r} = 1$ table, $s_{\text{initial}} = 1030$ (including 6 termination bits of the code) and $s_{\text{inc}} = 1030/8 \approx 127$. Thus $n_{\text{sent}} = 1030 + n_{\text{inc}} \cdot 127$. Figure (1) shows the rate performance of such a transmission system, Figure (2) shows the rate advantage of the optimistic version of the algorithm with $n_{\text{bo}} = 1$, Figure (3) shows the delay, measured as the number of transmitted increments, $n_{\text{inc}}$, of the system.

ity improves. The reason for this is that a good channel quality requires few bits to correct errors and hence the two strategies differ in fewer packets on fewer occasions. However, a noticeable difference exists between the optimistic version of the algorithm with $n_{\text{bo}} = 1$ and the pessimistic version with $n_{\text{bo}} = 0$. Here the performance difference is as large as $\approx 0.02$ (at 7 dB). Naturally, as the channel quality improves, more reason exists to be optimistic, since it can reasonably be expected that errors are few and spread out. Notice, however, that there is a significant drop in performance difference at 5 dB, coinciding with a plateau of the rate performance. This difference, however, is caused by the fixed increment size. This effect overshadows the trend of the performance difference between the optimistic and pessimistic version. The pessimistic version, on the other hand, reduces the average delay to $\approx 0.55$ increments packets but suffers a rate performance penalty for doing so. Since the pessimistic version only reduces the rate if the first packet is successful, the probability of over-correcting with this packet is higher, hence the rate loss and delay reduction. It is worth make note of an additional characteristic of Figure 5.10.3, namely that there is an increase in the maximum delay of the adaptive methods from $-4\,\text{dB} - -0\,\text{dB}$, with a resulting maximum at 0 dB. The reason for this increase is that the code is constraint severely at $r \approx 0.33$ and $r \approx 1$. In these situations, either few additional bits are sent ($r \approx 1$) or few remaining bits are left out ($r \approx 0.33$). Both situations are very easy to adjust to. At 0 dB, the algorithm performs at $r \approx 0.66$. The channel quality is still not very good (otherwise the rate would be better) nor is the channel quality so bad that a lot of bits are required (otherwise the rate would be worse). In short, the midpoint between $r \approx 0.33$ and $r \approx 1$ is just that point where the code can potentially correct very easily (i.e. with few bits) but potentially needs many bits. Thus the range of potential packet sizes is at its largest and the maximum number of increments is highest.

One remaining problem of the adaptive approach is the possibility of an error pattern which requires a large number of bits to be corrected following a situation where algorithm 5.1 aggregates only few increments. In this case, the delay remains high, since a lot of increment packets are needed to obtain the required number of bits at the receiver. Consider table 5.1 which shows the distribution of the number of increments under algorithm 5.1 (not considering the first packet, for which no prior information exists). The majority of transmission requires an increment $< 6$, however, for some packets, a maximum of 12 increments need to be transmitted. A solution to this problem can be achieved by taking into account a delay limit, $n_{\text{lim}}$, and transmit a “catch-all” packet,

| **Algorithm** | **Number of Increments** | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **and SNR** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| pessimistic, 0 dB | 576 | 300 | 108 | 10 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| pessimistic, 1 dB | 566 | 319 | 108 | 3 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| optimistic, 0 dB | 299 | 445 | 232 | 15 | 4 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| optimistic, 1 dB | 244 | 540 | 200 | 11 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| pess., limited, 0 dB | 573 | 301 | 110 | 10 | 5 | - | - | - | - | - | - | - | - |
| pess., limited, 1 dB | 561 | 324 | 108 | 3 | 3 | - | - | - | - | - | - | - | - |
| opt., limited, 0 dB | 296 | 447 | 233 | 15 | 8 | - | - | - | - | - | - | - | - |
| opt., limited, 1 dB | 240 | 542 | 202 | 11 | 4 | - | - | - | - | - | - | - | - |

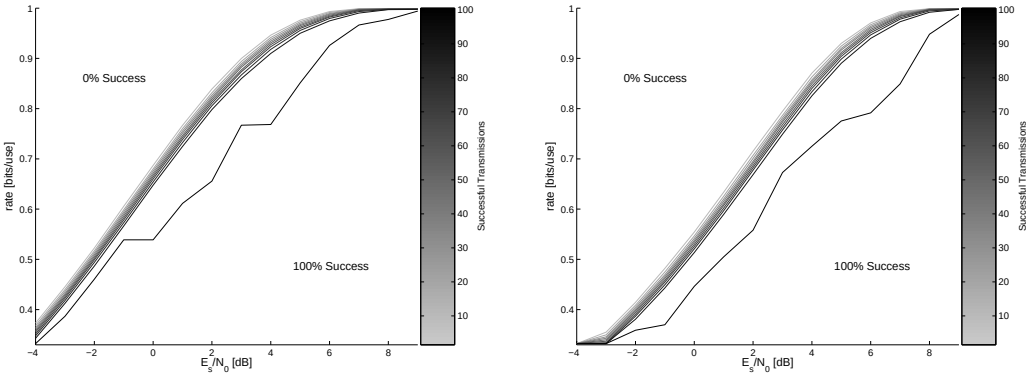
Table 5.1: Required number of increments for variations for aggregation of packets, described by algorithm 5.1. The pessimistic version has $n_{\text{bo}} = 0$, the optimistic version $n_{\text{bo}} = 1$. In the limited version, a catch-all packet is transmitted after the third increment. The number of increments is then only increased by 3. The rate performance difference between the limited and non-limited versions is $< 10^{-2}$. The first packet that is transmitted, for which no prior information exists, is omitted.

after the allowed number of increments is exhausted (refer to section 5.1.3). Such a modification of algorithm 5.1 is shown in algorithm 5.2. Since these catch-all packets are considered to be the exception, they should, accordingly, not used as a reference for future packets.

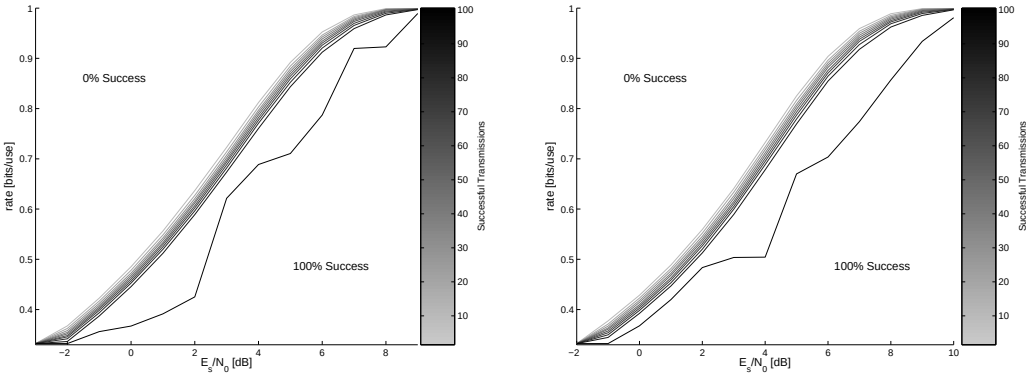
The delay performance of algorithm 5.2 is also shown in table 5.1. The rate penalty paid by occasionally over-correcting when the delay limit, $n_{\text{lim}}$, is reached is very small and the delay limited version performs within $\Delta < 10^{-2}$ of the non-limited version. The difference between the optimistic and pessimistic versions of algorithm 5.2 is again the back-off in line 14, $n_{\text{bo}}$. Note that while the number of catch-all packets (4 increments) is the sum of all increments $>= 4$ in the non-limited version, there is a slight redistribution for lower number of increments, since line 11 limits the increase in the number of aggregated increments.

**Algorithm 5.2** Delay-limited channel adaption by packet aggregation.

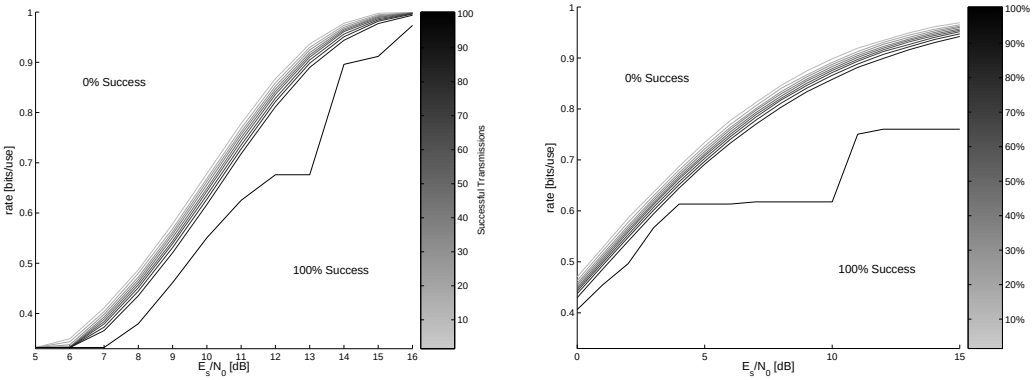
1: $n_{\text{bo}} = 0/1$ depending on selected algorithm parameters
2: transmit the first packet according to algorithm 5.1 to obtain initial $n_{\text{inc}}$
3: **for** each following packet to be transmitted **do**
4: Transmit packet 1 of size $s_1 = s_{\text{initial}} + n_{\text{inc}} \cdot s_{\text{inc}}$
5: **if** first packet is successful **then**
6: $n_{\text{inc}} = \max(n_{\text{inc}} - 1, 0)$
7: **else**
8: continue to transmit increments
9: **if** increment limit, $n_{\text{lim}}$, is reached **then**
10: transmit the catch-all packet
11: $n_{\text{inc}} = n_{\text{inc}} + n_{\text{lim}}$
12: **else**
13: let $n'_{\text{inc}}$ be the number of additional increments
14: $n_{\text{inc}} = n_{\text{inc}} + n'_{\text{inc}} - n_{\text{bo}}$
15: **end if**
16: **end if**
17: **end for**




(1) BIAWGN (2) DICODE




(3) EPR4 (4) E2PR4




(5) CH6 (6) Rayleigh

Figure 5.11: Percentage of successful transmissions for given rates and channel SNR. The majority of packets is contained in a fairly narrow band around the average rate performance. Only occasionally do packets require a significantly larger (or in fact lower) number of bits for successful transmission.

### 5.2.2 Beyond Aggregation – Variable Packet Sizes

The performance of the fixed packetisation scheme shown in Figure 5.10.1, while fairly good for low channel SNR, fails to approach the performance of a 1-bit accurate transmission system. This is due to the fact that for high channel SNR even a few unnecessary bits affect the puncture rate, $\mathring{r}$. Thus a truly adaptive transmission system needs to adjust both the size of the initial packet, $s_{\text{initial}}$, and the size of the increment, $s_{\text{inc}}$, beyond merely aggregating packets. Consider Figure 5.11, which shows the percentage of packets that are successful at a given rate and SNR. All investigated channels show a high percentage of packets which are successful for a similar rate. It thus is sensible to use the average of the successful transmissions as the initial packet and the width of the main band (10%–90%) for the increment packet. A "catch-all" packet can be used to take care of the few remaining that are not successful within the given delay limit.

The approach of averaging over the packets faces one significant problem, namely that the data about successful packets is limited. Consider a transmission which would require 2501 bits. If the initial packet is set to be 3000 bits, the transmission system has no information about how many bits *could have been left out*. The situation is better when increments need to be transmitted, since there are two boundaries set by the last increment, the sum of all packets transmitted before that increment, $n_{failed}$, and the number of bits including this last increment, $n_{successful}$. This information can be used to design an algorithm that adapts to the channel conditions of a stationary channel. Because sending more bits than necessary ensures correct decoding, it is practical to start off with too many bits. If the channel quality remains constant, determining the correct size can be approached with a binary search. This can be seen in algorithm 5.3 lines 1 to 5. The aim of this first phase of the algorithm is to find a packet size that fails. This gives a very rough lower bound on the minimum size of the initial packet. After this failing packet has been found, the algorithm uses the mean of the prior packets to determine the initial size (lines 6 to 8). Because the initial successful packets can be too large, the initial packets (Those packets that are sent before the first negative feedback is received) are not included in line 7.

The calculation of the increment size follows a similar two stage approach. When the first increment is requested (i.e. a packet has failed), the algorithm selects the increment size to be half of the difference of the last known successful packet and the initial size. This reinforces the notion of a binary search for the succeeding and failing

**Algorithm 5.3** Determine initial packet size

1: **if** first packet **then**
2: $size = (n_{max} + n_{min})/2$
3: **else if** no packet has failed **then**
4: $size = (n_{last} + n_{min})/2$
5: **else**
6: $\mu_{bad} = E\{n_{failed}\}$
7: $\mu_{good} = E\{n_{successful}\}$
8: $size = (\mu_{good} + \mu_{bad})/2$
9: **end if**
10: **return** *size*

packet sizes. After knowing at least one good and bad packet size, the algorithm takes into account the mean if succeeding and failing packet sizes. Note that in line 8 of algorithm 5.4 the size is multiplied by $2/3$. Since the initial packet size is the average of good and bad size means, an increment size of $2/3$ ensures that after the first increment packet the number of sent bits reaches at least the average of succeeding packets. In order to keep the delay low, the increment packet overshoots a bit. Empirical trials have shown, that in order to prevent a high delay due to small packet sizes, the increment size should be set to at least 200 bits, or about 10% of the input block size, $s_{\mathbf{x}}$.

**Algorithm 5.4** Determine increment size

1: **if** first packet **then**
2: $size = (n_{max} - n_{initial})/2$
3: **else if** no packet has failed **then**
4: $size = (n_{last} - n_{initial})/2$
5: **else**
6: $\mu_{bad} = E\{n_{failed}\}$
7: $\mu_{good} = E\{n_{successful}\}$
8: $size = (\mu_{good} - \mu_{bad}) \cdot 2/3$
9: **end if**
10: $size = MAX(size, 200)$
11: **return** *size*

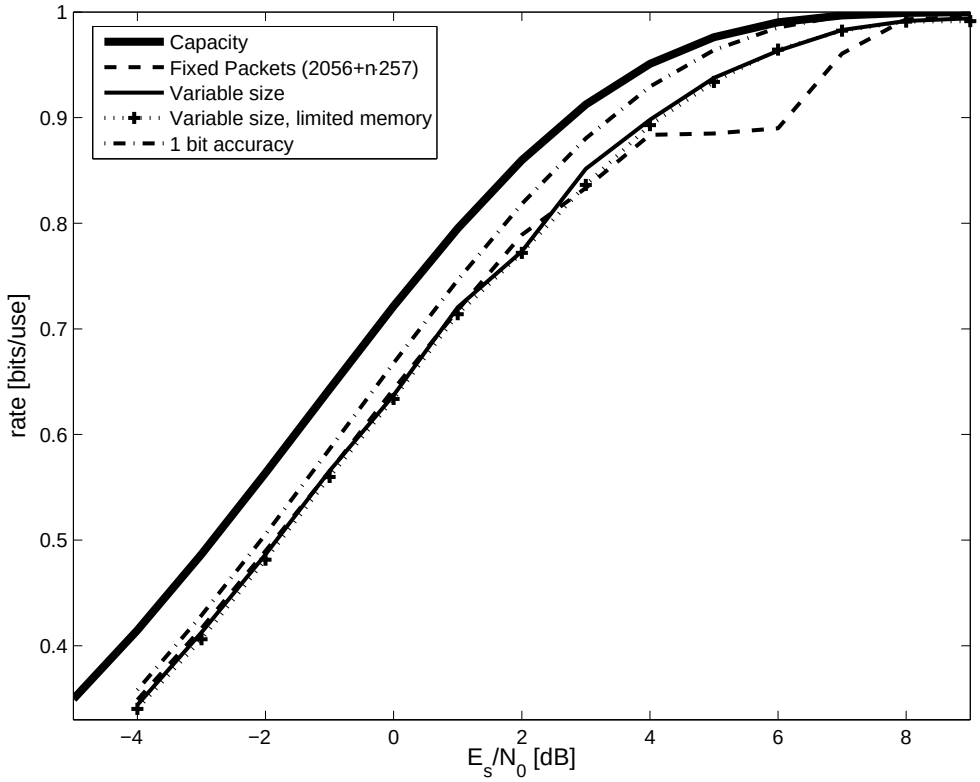



Figure 5.12: Performance of algorithms 5.3 and 5.4 for a Turbo Code consisting of 2 constituent RSC codes of order 6 and a random interleaver and an input blocksize of 2048 data bits, compared to the fixed packetisation strategy. The limited memory version only considers the past 10 packets when calculating $\mu_{good}$ and $\mu_{bad}$

| **SNR** | **Number of Increments** | | | | | $\mu$ |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | |
| $-4$ **dB** | 833 | 149 | 18 | 0 | 0 | 0.185 |
| $-2$ **dB** | 862 | 133 | 5 | 0 | 0 | 0.143 |
| 0 **dB** | 958 | 50 | 1 | 0 | 1 | 0.056 |
| 2 **dB** | 982 | 15 | 2 | 1 | 0 | 0.022 |
| 4 **dB** | 947 | 52 | 1 | 0 | 0 | 0.054 |
| 6 **dB** | 974 | 26 | 0 | 0 | 0 | 0.026 |
| 8 **dB** | 987 | 13 | 0 | 0 | 0 | 0.013 |
| 10 **dB** | 998 | 2 | 0 | 0 | 0 | 0.002 |

Table 5.2: Number of required retransmissions for variable packetisation

In order to adapt to a changing channel, an algorithm needs to work with time-relevant data. Thus the algorithm only considers the last $n$ packets. This also indicates that the algorithm is able to work with limited memory, since not all packets need to be saved for later processing. Figure 5.12 displays the performance for the described algorithm and its memory limited counterpart which considers the last 10 packets. As can be expected, not considering all packets slightly reduces performance. Overall the algorithm comes reasonably close to the best possible performance, with a good delay performance (see table 5.2.2), considering that IEEE802.11 allows 7 retransmissions.

Limiting algorithms 5.3 and 5.4 to a restricted number of past packets also allows the algorithm to adapt to slowly changing channels, since the information about the old channel conditions is eventually forgotten and not taken into account when selecting the initial size and increment size for the current channel conditions.

The performance of algorithms 5.3 and 5.4 is shown in Figure 5.12 and compared to a fixed packetisation resulting from using a rate-compatible puncturing table. The proposed variable packetisation algorithms not only outperform the fixed packet size strategy in terms of their delay performance, but show significantly better rate performance at high channel SNR. The proposed algorithms thus overcome the limitations of fixed packet sizes, of poor rate performance at either high channel SNR (large increments), or poor delay performance for low channel SNR (small increments). The only noticeable area where the fixed size packetisation is superior occurs at 2 dB, where the difference between good and bad decoding attempts is large (in terms of required bits

and in terms of predictability) and thus difficult to adapt to. However, the difference is negligible when compared to the gains possible at higher SNR.

## 5.3 Concluding Remarks

In chapters 3 and 4, the accuracy with which the transmission could be adjusted to the transmission channel was assumed to be 1 bit. Clearly, this assumption cannot be made for more realistic transmission systems. Therefore this chapter investigated the relationship between the sizes of the transmitted packets on the rate performance. It was shown that inappropriate packet sizes can lead to an excessive amount of useless parity bits, which are "over-correcting" the observed error pattern and that thus the size of the packets transmitting incremental redundancy needs to be reasonably small. The reason for this behaviour, as demonstrated in this chapter, is the existence of a waterfall region of Turbo Codes, where the BER drops rapidly with only a small increase in the number of transmitted bits, much like the waterfall region that Turbo Codes exhibit for increasing channel SNR. However, small, fixed packet sizes can result in an excessive delay for low channel SNR. In order to reduce this delay, a packetisation strategy was developed in this chapter that aggregates increments into the initial packet. This packetisation strategy was shown to perform within tenths of dB of the 1-bit accuracy for low channel SNR, even for the delay limited case, where the final packet transmitted before some delay threshold contains all remaining parity bits. However, at high SNR a severe performance degradation was experienced. Achieving good performance at high channel SNR was made possible by developing, in this chapter, an algorithm that uses the number of bits required for successful transmission of previous packets to adapt the packet sizes of the current transmission. Not being limited by mere aggregation, it was shown that this algorithm successfully adapts to the channel at high SNR, coming within tenths of dB of the performance of 1-bit accuracy without suffering the performance loss exhibited by the aggregation algorithm.

This chapter concludes the work of this chapter, showing that it is possible to use the results from chapters 3 and 4 in an adaptive transmission system that is delay limited and thus cannot adjust the amount of transmitted parity information at 1-bit accuracy. Obviously dropping 1-bit accuracy incurs a performance penalty, however, this chapter contributes a method to achieve channel adaption at only a small performance penalty.

# Chapter 6

# Conclusion

In this thesis the well known method of incremental redundancy in hybrid-ARQ systems was used to construct a transmission system that is able to transmit at an arbitrary rate by employing an interleaver to define a transmission order on the encoder output. Following the poor performance of simply randomising the order of bits, an upper bound was developed for the erasure-correcting ability of the RSC code. Possible transmission ordering strategies to overcome this problem were investigated. By reviewing Hagenauer's rate-compatibility criterion, a novel ordering strategy was developed for RSC codes that is agnostic of the constraint length of the code, quality of the channel, or blocksize of the input data and is able to transmit at a rate of $\mathring{r} \approx 1$ for high SNR, while adapting to the channel quality for lower SNR. By investigating the ability of RSC codes, an algorithm was developed to provide an upper bound on the performance of a transmission order regardless of the input to the encoder. This investigation contributed an insight in the difference of systematic and parity bits of the RSC encoder. Depending on the choice of generator polynomials of the RSC encoder different state transitions are assigned to parity and systematic bits. Because of this difference, transmission of either all systematic bits or all parity bits as the first bits to be transmitted provides a significant performance gain for RSC codes. The practical contribution of this result is that when searching for optimal generator polynomials for Turbo Codes, only those pairs need to be considered for which the systematic bits first provide superior performance, reducing the number of candidate pairs by about 50%.

Following the investigation of RSC codes, the capacity approaching Turbo Codes were investigated. The application of the ordering strategies that were developed for RSC codes is made more difficult by the added RSC encoder and the interleaver present in the Turbo encoder. It was shown that the availability of all systematic bits to the decoder is paramount for achieving good performance, particularly at high channel SNR. Thus empirical evidence was contributed, that the argument made by Rowitch and Milstein[109], that omitting systematic bits can be beneficial arises purely from the limitation of rate-compatible puncturing tables and the low selection granularity that they provide. It was demonstrated how rate-compatible puncturing tables can be modified to achieve a performance close to the one of the tree-based ordering strategy, however, the tree-based strategy remains superior with regard to its general applicability, being, like it was observed for RSC codes, agnostic of the generator polynomials, constraint length, input block size and channel quality.

The investigation of ordering strategies assumed a 1-bit accuracy when determining the performance of the ordering strategies. While nice conceptually, this is insufficient for achieving gains in practice, since the overhead by any transmission protocol prohibits the use of increments with a payload size of only 1 bit. The choice of packet sizes was investigated in chapter 5, which demonstrated, that Turbo Codes not only exhibit a waterfall region for increasing channel SNR at fixed rate, but *also for decreasing rate at fixed SNR*. Thus amount of redundancy transmitted to the receiver does not need to be very large, once this waterfall region is found. It was shown that a simple, fixed size transmission such as the transmission scheme that follows naturally from the use of rate-compatible puncturing tables, yields satisfactory rate performance at low channel SNR, as long as the increments are comparatively small, but requires a high number of increments in order to achieve that performance. For high channel SNR, the delay performance is adequate, however, the rate performance no longer is. In order to overcome the delay problem, an aggregation scheme was developed that combines the initial packet and the first increments into a single packet and keeps track of the required number of increments. Still, this scheme potentially requires a large number of increments. This problem can be addressed by making the last increment, transmitted before a given delay constraint is reached, contain all remaining parity information. It was shown that even such a delay limited system can perform well, within tenths of dB from the 1-bit accurate system. In order to address the lacking rate performance at high channel SNR, both the initial size and the increment size

needed to be adjusted more dynamically. An algorithm was developed that allows the transmission system to learn from the past successful and failed transmission sizes in order to adjust the packet size to the current channel conditions. It was shown that the proposed method is able to come, at worst, within 0.5 dB of the performance of a 1-bit accurate transmission system, while maintaining a modest delay requirement of, on average, $\approx 0.5$ increments.

This thesis aimed at addressing the two fundamental questions of an incremental redundancy, variable rate transmission system – how should the available bits be ordered for optimal partitioning into consecutive segments? and how large should the individual segments be? Answering the first question led to revisiting the rate-compatibility criterion, introduced by Hagenauer, and an optimal transmission ordering strategy based on the layout of a (partially randomised) binary tree. The second issue, how to split the bits into packets can be achieved by a simple, yet effective, algorithm which does not require any previous knowledge of the channel quality and simply takes into account average size of failed and successful transmissions of previous packets.

## Possible Improvements and Future Work

The work of this thesis raises some questions about the investigation of incremental redundancy schemes. As an example Frenger *et al.*[37], investigating HSDPA transmission systems, claim that incremental redundancy only provides small gains and that Chase combining can outperform incremental redundancy transmission systems for some fading channels. Unfortunately Frenger *et al.*do not illustrate their incremental redundancy scheme beyond stating that the retransmissions contain additional parity bits from the encoder. The results presented in this thesis indicate, that the selection of these parity bits is of supreme importance for achieving good performance.

An interesting question also arises when considering relay networks and the generalization of hybrid-ARQ systems to them[142]. In particular, if a relaying node should select a different ordering strategy than the original source node or if it should transmit those bits that the source would transmit next.

Some might argue, that the era of Turbo Codes is coming to an end, to be replaced by the also capacity-approaching low-density parity-check (LDPC) codes, which exhibit

a much lower error floor. Future work is required in order to find an ordering strategy suitable for LDPC. Of particular interest is the question if LDPC codes can be punctured with the same effectiveness as Turbo Codes, allowing LDPC codes to obtain $r_{\max} \approx 1$. Investigating the puncturing performance and potential ordering schemes for In their work on rate-compatible punctured low-density parity-check codes, Ha *et al.*[44][45] notice a severe performance[1] loss for high rate, punctured LDPC codes. In particular, random puncturing is problematic (as it is with Turbo Codes) since the punctured bits could contain a non-empty stopping set, which is not recoverable (see also [29]). Furthermore, Pishro-Nik and Fekri[94] observe that the maximum attainable rate is determined by a puncturing threshold, which explains the results by Ha *et al.*[44]. However, random puncturing can perform satisfactorily, if the desired rate increase is less than forty percent[95]. The design of puncturing schemes thus seems more difficult with LDPC codes. Vellambi and Fekri, working on puncturing schemes[127], base their results on the "intuition [...] to keep the punctured bits far from each other to ensure that their decoding neighbourhoods contain more unpunctured bit nodes", a similar intuition which was used in section 4.1, but even with a method to puncture additional bits[128] observe performance loss for high-rate (highly punctured) codes, concluding that "the range of achievable rates under the proposed scheme is limited". That said, the puncturing of Turbo Codes is far from well-understood. The arguments made in this thesis about the construction of ordering schemes are based on an observation of the structure of the codes and the decoding process, since the presence of random interleavers for Turbo Codes makes it difficult to analyse Turbo Codes. Therefore the optimality of the proposed transmission order cannot be guaranteed and it could potentially be further improved. In addition, schemes that increase the bandwidth-efficiency, such as turbo trellis-coded modulation[106], potentially do not lend themselves well to puncturing for higher rates, since puncturing needs to be done on a transmitted symbol resulting in the puncture of blocks of 2 systematic and 1 parity bit.

It is also possible, however, to use an entirely different approach and disconnect completely from an ordering strategy. Rateless coding (see e.g. [17]) can be used to decrease the rate arbitrarily. Sejdinovic *et al.*[114] combine LDPC and Fountain codes to achieve good rate adaption to overcome the inability of Fountain codes to come close to $r_{\max} \approx 1$, however, they experience a decrease in performance at low SNR compared

[1] Performance is defined as the distance of the achieved to the theoretical BER.

to Fountain codes alone. The significant advantage of Fountain Codes, and rateless codes in general, is that they can, potentially, achieve any rate with enough information transmitted, thus only requiring an answer to the second issue of this thesis about the size of the individual increments. An interesting approach to that end could come from the considerations made by Sejdinovic *et al.*[115] and Vukobratovic *et al.*[132] when considering a windowing technique for constructing Fountain codes in the context of unequal error protection. In their scheme, the Fountain code is based on the first $s$ bits of an input block of length $k$, the window size is set with some probability $p$ for each encoded symbol. Thus the first bits are protected the most, and, assuming that the input bits are ordered in decreasing importance, this scheme protects the most important bits the most. Two observations are important. Firstly, since these Fountain Codes only use a subset of the input, it could be possible that a punctured Fountain Code can be implemented by working on different subsets of bits of the input. However, this requires a notion of order on the transmitted bits which could be difficult to find. Due to the lack of systematic bits, the result from this thesis cannot be applied. Secondly, this scheme could be used in a Selective-Repeat ARQ scheme to protect those *packets* the most (by placing them at the beginning of a concatenation of packets) that are reaching their maximum delay, thus potentially increasing the efficiency of the transmission system while also improving reliability. Again, however, the open question remains if the good performance of punctured Turbo Codes, achieving high rate for good channel SNR *and* good adaption overall can be matched, in particular since Fountain Codes are more complex to implement than Turbo Codes and, in order that their use is is justified, need to provide a performance advantage.

Finally, the algorithm presented for obtaining the initial and increment sizes can deal with slowly varying channels. The adaption to channels with sudden changes of the channel quality, e.g. losing the line-of-sight component need to be investigated with respect to an optimal packetisation strategy.

# Bibliography


[1] F. Alajaji, "Feedback does not increase the capacity of discrete channels with additive noise," *IEEE Transactions on Information Theory*, vol. 41, no. 2, pp. 546–549, Mar. 1995.

[2] A. Annamalai, V. Bhargava, and W.-S. Lu, "On adaptive go-back-n ARQ protocol for variable-error rate channels," *IEEE Transactions on Communications*, vol. 46, no. 11, pp. 1405–1408, Nov. 1998.

[3] D. Arnold and H.-A. Loeliger, "On the information rate of binary-input channels with memory," in *Proc. IEEE International Conference on Communications (ICC '01)*, vol. 9, Helsinki, Finland, 11–14 Jun. 2001, pp. 2692–2695.

[4] F. Babich, G. Montorsi, and F. Vatta, "Design of rate-compatible punctured turbo (RCPT) codes," in *Proc. IEEE International Conference on Communications (ICC '02)*, vol. 3, New York, USA, 28 Apr. – 2 May 2002, pp. 1701–1705.

[5] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[6] F. Banerjee, A.and Vatta, B. Scanavino, and D. J. Costello, Jr., "Nonsystematic turbo codes," *IEEE Transactions on Communications*, vol. 53, no. 11, pp. 1841–1849, Nov. 2005.

[7] A. S. Barbulescu and S. S. Pietrobon, "Rate compatible turbo codes," *Electronics Letters*, vol. 31, no. 7, pp. 535–536, 1995.

[8] S. Benedetto, R. Garello, and G. Montorsi, "A search for good convolutional codes to be used in the construction of turbo codes," *IEEE Transactions on Communications*, vol. 46, no. 9, pp. 1101–1105, Sep. 1998.

[9] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 409–428, Mar. 1996.

[10] R. Benice and J. Frey, A., "An analysis of retransmission systems," *IEEE Transactions on Communication Technology*, vol. 12, no. 4, pp. 135–145, Dec. 1964.

[11] C. Berrou and A. Glavieux, "Near optimum error correcting and decoding: Turbo-codes," *IEEE Transactions on Communications*, vol. 44, pp. 1261–1271, Oct. 1996.

[12] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE International Conference on Communications (ICC '93)*, vol. 2, Geneva, Switzerland, 23–26 Mar. 1993, pp. 1064–1070.

[13] K. Brayer, "Error control techniques using binary symbol burst codes," *IEEE Transactions on Communication Technology*, vol. 16, no. 2, pp. 199–214, Apr. 1968.

[14] H. O. Burton and D. D. Sullivan, "Errors and error control," *Proceedings of the IEEE*, vol. 60, no. 11, pp. 1293–1301, Nov. 1972.

[15] R. Cam and C. Leung, "Throughput analysis of some ARQ protocols in the presence of feedback errors," *IEEE Transactions on Communications*, vol. 45, no. 1, pp. 35–44, Jan. 1997.

[16] R. Cam, C. Leung, and C. Lam, "A performance comparison of some combining schemes for finite-buffer ARQ systems in a Rayleigh-fading channel," in *Proc. IEEE International Conference on Selected Topics in Wireless Communications*, Vancouver, B. C., Canada, 25–26 Jun. 1992, pp. 88–92.

[17] J. Castura and Y. Mao, "Rateless coding and relay networks – benefits, challenges, and considerations for designing with rateless codes," *IEEE Signal Processing Magazine*, vol. 24, no. 5, pp. 27–35, Sep. 2007.

[18] S. S. Chakraborty, M. Liinabarja, and E. Yli-Juuti, "An adaptive ARQ scheme with packet combining for time varying channels," *IEEE Communications Letters*, vol. 3, no. 2, pp. 52–54, Feb. 1999.

[19] S. S. Chakraborty, E. Yli-Juuti, and M. Liinaharja, "An ARQ scheme with packet combining," *IEEE Communications Letters*, vol. 2, no. 7, pp. 200–202, Jul. 1998.

[20] D. Chase, "Code combining – a maximum-likelihood decoding approach for combining an arbitrary number of noisy packets," *IEEE Transactions on Communications*, vol. 33, no. 5, pp. 385–393, 1985.

[21] X.-M. Chen and P. A. Hoeher, "Reduced-complexity SISO equalization for Rayleigh fading channels with known statistics," in *Proc. IEEE 59th Vehicular Technology Conference (VTC '04-Spring)*, vol. 1, Milan, Italy, 17–19 May 2004, pp. 535–539.

[22] G. C. Clark and J. B. Cain, *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. New York: McGraw-Hill, Inc., 1990.

[24] D. J. Costello, Jr., J. Hagenauer, H. Imai, and S. B. Wicker, "Applications of error-control coding," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2531–2560, Oct. 1998.

[25] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, ser. Wiley Series in Telecommunication, D. L. Schilling, Ed. New York: John Wiley & Sons Inc., 1991.

[26] L. de Alfaro and A. Meo, "Codes for second and third order GH-ARQ schemes," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 899–910, Feb./Mar./Apr. 1994.

[27] R. H. Deng, "Hybrid ARQ scheme using TCM and code combining," *Electronics Letters*, vol. 27, no. 10, pp. 866–868, May 1991.

[28] ——, "Hybrid ARQ schemes employing coded modulation and sequence combining," *IEEE Transactions on Communications*, vol. 42, no. 6, pp. 2239–2245, Jun. 1994.

[29] C. Di, D. Proietti, I. Telatar, T. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

[30] C. Douillard, A. Picart, P. Didier, M. JÃl'zÃl'quel, C. Berrou, and A. Glavieux, "Iterative correction of intersymbol interference: Turbo-equalization," *European Transactions on Telecommunications*, vol. 6, no. 5, pp. 507–512, Sep. – Oct. 1995.

[31] A. Drukarev and D. J. Costello, Jr., "Hybrid ARQ error control using sequential decoding," *IEEE Transactions on Information Theory*, vol. 29, no. 4, pp. 521–535, Jul. 1983.

[32] P. Elias, "Coding for noisy channels," in *IRE National Convention Record*, 1955, pp. 37–46.

[33] G. F. Elmasry, "Joint lossless-source and channel coding using automatic repeat request," *IEEE Transactions on Communications*, vol. 47, no. 7, pp. 953–955, Jul. 1999.

[34] R. Fang, "Lower bounds on reliability functions of variable-length nonsystematic convolutional codes for channels with noiseless feedback," *IEEE Transactions on Information Theory*, vol. 17, no. 2, pp. 161–171, Mar. 1971.

[35] G. D. Forney, "Review of random tree codes," in *Appendix A. Study of Coding Systems Design for Advanced Solar Missions. NASA Contract NAS2-3637*. Codex Corporation, Dec. 1967.

[36] M. P. C. Fossorier, F. Burkert, S. Lin, and J. Hagenauer, "On the equivalence between SOVA and Max-Log-MAP decodings," *IEEE Communications Letters*, vol. 2, no. 5, pp. 137–139, May 1998.

[37] P. Frenger, S. Parkvall, and E. Dahlman, "Performance comparison of HARQ with chase combining and incremental redundancy for HSDPA," in *Proc. 54th IEEE Vehicular Technology Conference, 2001. VTC 2001 Fall.*, vol. 3, Atlantic City, NJ, USA, 7–11 Oct. 2001, pp. 1829–1833.

[38] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[39] ——, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[40] D. Garg and F. Adachi, "Rate compatible punctured turbo-coded hybrid ARQ for OFDM in a frequency selective fading channel," in *Proc. The 57th IEEE*

*Semiannual Vehicular Technology Conference (VTC '03-Spring)*, vol. 4, Jeju, Korea, 22–25 Apr. 2003, pp. 2725–2729.

[41] I. A. Glover and P. M. Grant, *Digital Communication*. Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England: Prentice Hall, 1998.

[42] N. Goertz, "Aufwandsarme Qualitätsverbesserungen bei der gestörten Übertragung codierter Sprachsignale," Ph.D. dissertation, Technische Fakultät der Christian-Albrechts-Universität zu Kiel, 1998.

[43] ——, *Joint Source-Channel Coding of Discrete-Time Signals with Continuous Amplitudes*, ser. Communications and Signal Processing. London: World Scientific Publishing, Imperial College Press, 2007, vol. 1.

[44] J. Ha, J. Kim, D. Klinc, and S. McLaughlin, "Rate-compatible punctured low-density parity-check codes with short block lengths," *IEEE Transactions on Information Theory*, vol. 52, no. 2, pp. 728–738, Feb. 2006.

[45] J. Ha and S. McLaughlin, "Optimal puncturing of irregular low-density parity-check codes," in *Proc. IEEE International Conference on Communications (ICC '03)*, vol. 5, Anchorage, AK, USA, 11–15 May 2003, pp. 3110–3114.

[46] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Transactions on Communications*, vol. 36, no. 4, pp. 389–400, 1988.

[47] ——, "Soft–in / soft–out: The benefits of using soft values in all stages of digital receivers," in *Proc. 3rd International Workshop on DSP Techniques Applied to Space Communications*, ESTEC, Noordwijk, Netherlands, Sep. 1992.

[48] ——, "The turbo principle: Tutorial introduction and state of the art," in *Proc. International Symposium on Turbo Codes*, Brest, France, 3–5 Sep. 1997, pp. 1–11.

[49] J. Hagenauer, N. Dutsch, J. Barros, and A. Schaefer, "Incremental and decremental redundancy in turbo source-channel coding," in *Proc. First International Symposium on Control, Communications and Signal Processing (ISCCSP '04)*, Hammamet, Tunesia, 21–24 Mar. 2004, pp. 595–598.

[50] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM '89)*, vol. 3, Dallas, TX, USA, 27–30 Nov. 1989, pp. 1680–1686.

[51] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[52] L. Hanzo, W. Webb, and T. Keller, *Single and Multicarrier Quadrature Amplitude Modulation: Principles and Applications for Personal Communications, WATM and Broadcasting*, 1st ed. Chichester, England: John Wiley - IEEE Press, 2000.

[53] L. Hanzo, L.-L. Yang, E.-L. Kuan, and K. Yen, *Single- and Multi-Carrier DS-CDMA*. Chichester, England: John Wiley - IEEE Press, 2003.

[54] T. Hashimoto, "A coded ARQ scheme with the generalized Viterbi algorithm," *IEEE Transactions on Information Theory*, vol. 39, no. 2, pp. 423–432, Mar. 1993.

[55] ——, "On the error exponent of convolutionally coded ARQ," *IEEE Transactions on Information Theory*, vol. 40, no. 2, pp. 567–575, Mar. 1994.

[56] M. S. C. Ho, S. S. Pietrobon, and T. Giles, "Improving the constituent codes of turbo encoders," in *IEEE Global Telecommunications Conference (GLOBECOM '98)*, vol. 6, Sydney, Australia, 8–12 Nov. 1998, pp. 3525–3529.

[57] *Information Processing Systems–Data Communication High-Level Data Link Control Procedure–Frame Structure*, International Organization for Standardization Std. IS 3309, Rev. 3rd, Oct. 1984.

[58] *Error-correcting Procedures for DCEs Using Asynchronous-to-Synchronous Conversion*, International Telecommunications Union Recommendation V.42, Rev. 1, 1994.

[59] W. S. Jeon and D. G. Jeong, "Improved selective repeat ARQ scheme for mobile multimedia communications," *IEEE Communications Letters*, vol. 4, no. 2, pp. 46–48, Feb. 2000.

[60] P. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*. New York: IEEE Press, 1999.

[61] *Advanced Video Coding for Generic Audiovisual Services*, ITU-T Rec. H.264 & ISO/IEC 14496-10, Version 4, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG Std. ITU-T Rec. H.264 & ISO/IEC 14 496-10, Version 4, Nov. 2007.

[62] S. Kallel, "Efficient hybrid ARQ protocols with adaptive forward error correction," *IEEE Transactions on Communications*, vol. 42, no. 234, pp. 281–289, Feb./Mar./Apr. 1994.

[63] S. Kallel and D. Haccoun, "Sequential decoding with ARQ and code combining: A robust hybrid FEC/ARQ system," *IEEE Transactions on Communications*, vol. 36, no. 7, pp. 773–780, Jul. 1988.

[64] ——, "Generalized type II hybrid ARQ scheme using punctured convolutional coding," *IEEE Transactions on Communications*, vol. 38, no. 11, pp. 1938–1946, Nov. 1990.

[65] S. Kallel and C. Leung, "Adaptive incremental redundancy selective-repeat ARQ scheme for finite buffer receivers," *Electronics Letters*, vol. 28, no. 7, pp. 664–666, Mar. 1992.

[66] ——, "Efficient ARQ schemes with multiple copy decoding," *IEEE Transactions on Communications*, vol. 40, no. 3, pp. 642–650, Mar. 1992.

[67] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Transactions on Information Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[68] H. Krishna and S. Morgera, "A new error control scheme for hybrid ARQ systems," *IEEE Transactions on Communications*, vol. 35, no. 10, pp. 981–990, Oct. 1987.

[69] B. D. Kudryashov, "Convolutional-block coding in channels with decision feedback," *Probl. Peredach. Inform.*, vol. 21, no. 1, pp. 17–27, 1985.

[70] ——, "Error probability for repeat request systems with convolutional codes," *IEEE Transactions on Information Theory*, vol. 39, no. 5, pp. 1680–1684, Sep. 1993.

[71] I. Land, "Reliability information in channel decoding," Ph.D. dissertation, Technische Fakultät der Christian-Albrechts-Universität zu Kiel, 2005.

[72] I. Land and P. A. Hoeher, "New results on monte carlo bit error simulation based on the a posteriori log-likelihood ratios," in *Proc. 3rd International Symposium on Turbo Codes and Related Topics*, Brest, France, 1–3 Sep. 2003.

[73] LAN/MAN Standards Committee of the IEEE Computer Society, *IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. IEEE 802.11, Rev. 2007, Dec. 2007.

[74] S.-K. Lee and M.-C. Lin, "An ARQ scheme using combined QPSK and BPSK transmissions," *IEEE Transactions on Communications*, vol. 43, no. 5, pp. 1917–1925, May 1995.

[75] S. Lin and D. J. Costello, Jr., *Error Control Coding*. Pearson Education International, 2004.

[76] S. Lin, D. J. Costello, Jr., and M. Miller, "Automatic-repeat-request error-control schemes," *IEEE Communications Magazine*, vol. 22, no. 12, pp. 5–17, 1984.

[77] S. Lin and P. Yu, "A hybrid ARQ scheme with parity retransmission for error control of satellite channels," *IEEE Transactions on Communications*, vol. 30, no. 7, pp. 1701–1719, Jul. 1982.

[78] J. A. Lockitt, A. G. Gatfield, and T. R. Dobyns, "A selective repeat ARQ system," in *Proc. 3rd International Conference on Digital Satellite Communications*, Kyoto, Japan, 11–13 Nov. 1975, pp. 189–195.

[79] L. R. Lugand, D. J. Costello, Jr., and R. H. Deng, "Parity retransmission hybrid ARQ using rate 1/2 convolutional codes on a nonstationary channel," *IEEE Transactions on Communications*, vol. 37, no. 7, pp. 755–765, Jul. 1989.

[80] R. Maerkle and C.-E. W. Sundberg, "Channel codes based on hidden puncturing for partial band interference channels," in *Proc. IEEE International Conference on Communications (ICC '03)*, vol. 4, Anchorage, AK, USA, 11–15 May 2003, pp. 2894–2898.

[81] E. Malkamäki and H. Leib, "Performance of truncated type-II hybrid ARQ schemes with noisy feedback over block fading channels," *IEEE Transactions on Communications*, vol. 48, no. 9, pp. 14 770–1487, Sep. 2000.

[82] D. Mandelbaum, "An adaptive-feedback coding scheme using incremental redundancy," *IEEE Transactions on Information Theory*, vol. 20, no. 3, pp. 388–389, May 1974.

[83] R. Mantha and F. R. Kschischang, "A capacity-approaching hybrid ARQ scheme using turbo codes," in *Proc. Global Telecommunications Conference (GLOBECOM '99)*, vol. 5, Rio de Janeiro, Brazil, 5–9 Dec. 1999, pp. 2341–2345.

[84] J. Massey and D. J. Costello, Jr., "Nonsystematic convolutional codes for sequential decoding in space applications," *IEEE Transactions on Communications*, vol. 19, no. 5, pp. 806–813, Oct. 1971.

[85] P. C. Massey and D. J. Costello, Jr., "Turbo codes with recursive nonsystematic quick-look-in constituent codes," in *Proc. IEEE International Symposium on Information Theory (ISIT '01)*, Washington, D.C., USA, 24–29 Jun. 2001, p. 141.

[86] ——, "New developments in asymmetric turbo codes," in *Proc. 2nd International Symposium on Turbo Codes*, Brest, France, 4–7 Sep. 2000, pp. 93–100.

[87] J. Metzner, "Improvements in block-retransmission schemes," *IEEE Transactions on Communications*, vol. 27, no. 2, pp. 524–532, Feb. 1979.

[88] M. Miller and S. Lin, "The analysis of some selective-repeat ARQ schemes with finite receiver buffer," *IEEE Transactions on Communications*, vol. 29, no. 9, pp. 1307–1315, Sep. 1981.

[89] S. D. Morgera and V. K. Oduol, "Soft-decision decoding applied to the generalized type-II hybrid ARQ scheme," *IEEE Transactions on Communications*, vol. 37, no. 4, pp. 393–396, Apr. 1989.

[90] J. Morris, "Optimal blocklengths for ARQ error control schemes," *IEEE Transactions on Communications*, vol. 27, no. 2, pp. 488–493, Feb. 1979.

[91] K. R. Narayanan and G. L. Stuber, "A novel ARQ technique using the turbo coding principle," *IEEE Communications Letters*, vol. 1, no. 2, pp. 49–51, Mar. 1997.

[92] V. K. Oduol and S. D. Morgera, "Performance evaluation of the generalized type-II hybrid ARQ scheme with noisy feedback on Markov channels," *IEEE Transactions on Communications*, vol. 41, no. 1, pp. 32–40, Jan. 1993.

[93] L. C. Perez, J. Seghers, and D. J. Costello, Jr., "A distance spectrum interpretation of turbo codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1698–1709, 1996.

[94] H. Pishro-Nik and F. Fekri, "Results on punctured ldpc codes," in *Proc. IEEE Information Theory Workshop, 2004*, San Antonio, TX, USA, 24–29 Oct. 2004, pp. 215–219.

[95] ——, "Results on punctured low-density parity-check codes and improved iterative decoding techniques," *IEEE Transactions on Information Theory*, vol. 53, no. 2, pp. 599–614, Feb. 2007.

[96] J. G. Proakis, *Digital Communications*, 4th ed. New York: McGraw-Hill, 2000.

[97] M. B. Pursley and S. D. Sandberg, "Variable-rate hybrid ARQ for meteor-burst communications," *IEEE Transactions on Communications*, vol. 40, no. 1, pp. 60–73, Jan. 1992.

[98] D. Raphaeli and Y. Zarai, "Combined turbo equalization and turbo decoding," *IEEE Communications Letters*, vol. 2, no. 4, pp. 107–109, Apr. 1998.

[99] T. S. Rappaport, *Wireless Communications*, 2nd ed. Pearson Education, 2002.

[100] L. K. Rasmussen and S. B. Wicker, "Trellis-coded, type-I hybrid-ARQ protocols based on CRC error-detecting codes," *IEEE Transactions on Communications*, vol. 43, no. 10, pp. 2569–2575, Oct. 1995.

[101] M. Rice, "Application of generalized minimum distance decoding to hybrid-ARQ error control," *IEEE Transactions on Communications*, vol. 42, no. 234, pp. 640–647, Feb./Mar./Apr. 1994.

[102] M. Rice and S. B. Wicker, "Adaptive error control for slowly varying channels," *IEEE Transactions on Communications*, vol. 42, no. 234, pp. 917–926, Feb./Mar./Apr. 1994.

[103] ——, "A sequential scheme for adaptive error control over slowly varying channels," *IEEE Transactions on Communications*, vol. 42, no. 234, pp. 1533–1543, Feb./Mar./Apr. 1994.

[104] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[105] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[106] P. Robertson and T. Wörz, "Bandwidth-efficient turbo trellis-coded modulation using punctured component codes," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 206–218, Feb. 1998.

[107] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *European Transactions on Telecommunications*, vol. 8, pp. 119–125, Mar. – Apr. 1997.

[108] E. Y. Rocher and R. L. Pickholts, "An analysis of the effectiveness of hybrid transmission schemes," *IBM J. Res. Dev.*, vol. 7, pp. 426–33, Jul. 1970.

[109] D. N. Rowitch and L. B. Milstein, "On the performance of hybrid FEC/ARQ systems using rate compatible punctured turbo (RCPT) codes," *IEEE Transactions on Communications*, vol. 48, no. 6, pp. 948–959, 2000.

[110] W. E. Ryan, *Convolutional Codes and Iterative Decoding*, ser. Wiley Encyclopedia of Telecommunications, J. G. Proakis, Ed. New York: John Wiley, 2002.

[111] K. Sakakibara and M. Kasahara, "A multicast hybrid ARQ scheme using MDS codes and GMD decoding," *IEEE Transactions on Communications*, vol. 43, no. 12, pp. 2933–2940, Dec. 1995.

[112] A. Sastry, "Performance of hybrid error control schemes of satellite channels," *IEEE Transactions on Communications*, vol. 23, no. 7, pp. 689–694, Jul. 1975.

[113] A. Sastry and L. Kanal, "Hybrid error control using retransmission and generalized burst-trapping codes," *IEEE Transactions on Communications*, vol. 24, no. 4, pp. 385–393, Apr. 1976.

[114] D. Sejdinovic, V. Ponnampalam, R. J. Piechocki, and A. Doufexi, "The throughput analysis of different IR-HARQ schemes based on fountain codes," in *Proc. IEEE Wireless Communications and Networking Conference*, Las Vegas, NV, USA, 31 Mar.– 3 Apr. 2008, pp. 267–272.

[115] D. Sejdinovic, D. Vukobratovic, A. Doufexi, V. Senk, and R. J. Piechocki, "Expanding window fountain codes for unequal error protection," in *Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, 2007. ACSSC 2007.*, Pacific Grove, CA, USA, 4–7 Nov. 2007, pp. 1020–1024.

[116] C. E. Shannon, "A mathematical theory of communication," *Bell Systems Technical Journal*, vol. 27, pp. 379–423, 1948.

[117] A. Shiozaki, "Adaptive type-II hybrid broadcast ARQ system," *IEEE Transactions on Communications*, vol. 44, no. 4, pp. 420–422, Apr. 1996.

[118] P. Sindhu, "Retransmission error control with memory," *IEEE Transactions on Communications*, vol. 25, no. 5, pp. 473–479, May 1977.

[119] T. Takata, T. Fujiwara, T. Kasami, and S. Lin, "An error control system with multiple-stage forward error corrections," *IEEE Transactions on Communications*, vol. 38, no. 10, pp. 1799–1809, Oct. 1990.

[120] R. M. Tanner, "Recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, pp. 533–547, Sep. 1981.

[121] Technical Commitee on Computer Communications of the IEEE Computer Society, *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*, IEEE Std. IEEE 802.3, 2005.

[122] S. ten Brink, "Design of serially concatenated codes based on iterative decoding convergence," in *Proc. 2nd International Symposium on Turbo Codes & Related Topics*, Brest, France, 4–7 Sep. 2000, pp. 319–322.

[123] ——, "Code doping for triggering iterative decoding convergence," *IEEE International Symposium on Information Theory*, vol. 1, p. 235, 2001.

[124] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[125] M. Tüchler, "Turbo equalization," Ph.D. dissertation, Technische Universität München, 2003.

[126] E. Uhlemann, T. M. Aulin, L. K. Rasmussen, and P.-A. Wiberg, "Packet combining and doping in concatenated hybrid ARQ schemes using iterative decoding," *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, vol. 2, pp. 849–854, Mar. 2003.

[127] B. N. Vellambi and F. Fekri, "Rate-compatible puncturing of finite-length low-density parity-check codes," in *Proc. 2006 IEEE International Symposium on Information Theory*, Seattle, WA, USA, 9–14 Jul. 2006, pp. 1129–1133.

[128] ——, "Finite-length rate-compatible ldpc codes: a novel puncturing scheme," *IEEE Transactions on Communications*, vol. 57, no. 2, pp. 297–301, Feb. 2009.

[129] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, Apr. 1967.

[130] B. Vucetic, "An adaptive coding scheme for time-varying channels," *IEEE Transactions on Communications*, vol. 39, no. 5, pp. 653–663, 1991.

[131] B. Vucetic and J. Yuan, *Turbo Codes*. Boston, Mass.: Kluwer Academic, 2000.

[132] D. Vukobratovic, V. Stankovic, D. Sejdinovic, L. Stankovic, and Z. Xiong, "Expanding window fountain codes for scalable video multicast," in *2008 IEEE International Conference on Multimedia and Expo*, Hannover, Germany, 23–26 Apr. 2008, pp. 77–80.

[133] Y.-M. Wang and S. Lin, "A modified selective-repeat type-II hybrid ARQ system and its performance analysis," *IEEE Transactions on Communications*, vol. 31, no. 5, pp. 593–608, 1983.

[134] S. B. Wicker, "Reed-Solomon error control coding for Rayleigh fading channels with feedback," *IEEE Transactions on Vehicular Technology*, vol. 41, no. 2, pp. 124–133, May 1992.

[135] S. Wicker, "High-reliability data transfer over the land mobile radio channel using interleaved hybrid-ARQ error control," *IEEE Transactions on Vehicular Technology*, vol. 39, no. 1, pp. 48–55, Feb. 1990.

[136] S. Wicker and M. Bartz, "The design and implementation of type-I and type-II hybrid-ARQ protocols based on first-order Reed-Muller codes," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 979–987, Feb./Mar./Apr. 1994.

[137] ——, "Type-II hybrid-ARQ protocols using punctured MDS codes," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 1431–1440, Feb./Mar./Apr. 1994.

[138] T. Wiegand and G. J. Sullivan, "The H.264/AVC video coding standard [standards in a nutshell]," *IEEE Signal Processing Magazine*, vol. 24, no. 2, pp. 148–153, Mar. 2007.

[139] W. Xie and J. Tang, "Rate compatible punctured turbo code scheme over optical wireless channel," in *Proc. IEEE International Conference on Communications (ICC '03)*, vol. 5, Anchorage, AK, USA, 11–15 May 2003, pp. 3130–3134.

[140] H. Yamamoto and K. Itoh, "Viterbi decoding algorithm for convolutional codes with repeat request," *IEEE Transactions on Information Theory*, vol. 26, no. 5, pp. 540–547, Sep. 1980.

[141] Y.-D. Yao, "An effective go-back-n ARQ scheme for variable-error-rate channels," *IEEE Transactions on Communications*, vol. 43, no. 1, pp. 20–23, Jan. 1995.

[142] B. Zhao and M. C. Valenti, "Practical relay networks: a generalization of hybrid-ARQ," *IEEE Transactions on Communications*, vol. 23, no. 1, pp. 7–18, Jan. 2005.