Chapter 2: Error Detection and Correction Codes

Lectures 4-5

Check sum (or check codeword)

To send the message MATH3411,
we encode each character in ASCII
and add a binary-parity check sum
(this is a burst code).

Here, the check sum is 0010111,
ASCII for the symbol $\overline{\text{ETB}}$.
The message sent is then MATH3411$\overline{\text{ETB}}$.

Example
Burst noise affects consecutive bits.
For instance, it might send each bit to 1.

	ASCII
M	1 0 0 1 1 0 1
A	1 0 0 0 0 0 1
T	1 0 1 0 1 0 0
H	1 0 0 1 0 0 0
3	0 1 1 0 0 1 1
4	0 1 1 0 1 0 0
1	0 1 1 0 0 0 1
1	0 1 1 0 0 0 1
$\overline{\text{ETB}}$	0 0 1 0 1 1 1

Check sum (or check codeword)

To send the message MATH3411,
we encode each character in ASCII
and add a binary-parity check sum
(this is a burst code).

Here, the check sum is 0010111,
ASCII for the symbol $\overline{\mathrm{ETB}}$.
The message sent is then MATH3411$\overline{\mathrm{ETB}}$.

Example
Burst noise affects consecutive bits.
For instance, it might send each bit to 1.

	ASCII
M	1 0 0 1 1 0 1
A	1 0 0 0 0 0 1
T	1 0 1 0 1 0 0
O	1 0 0 1 1 1 1
s	1 1 1 0 0 1 1
4	0 1 1 0 1 0 0
1	0 1 1 0 0 0 1
1	0 1 1 0 0 0 1
$\overline{\mathrm{ETB}}$	0 0 1 0 1 1 1

Check sum (or check codeword)

To send the message MATH3411, we encode each character in ASCII and add a binary-parity check sum (this is a burst code).

Here, the check sum is 0010111, ASCII for the symbol $\overline{\text{ETB}}$. The message sent is then MATH3411$\overline{\text{ETB}}$.

We could also use 8-bit even parity ASCII.
We can then detect and correct 1 error.
We can often detect several errors
- but we cannot correct them.

	ASCII
M	0 1 0 0 1 1 0 1
A	0 1 0 0 0 0 0 1
D	1 1 0 0 0 1 0 0
H	0 1 0 0 1 0 0 0
1	0 0 1 1 0 0 0 1
4	1 0 1 1 0 1 0 0
1	1 0 1 1 0 0 0 1
1	1 0 1 1 0 0 0 1
$\overline{\text{ETB}}$	0 0 0 1 0 1 1 1

Check sum (or check codeword)

To send long messages, we can partition them into 8-character blocks.

Example

The message MATH3411␣IS␣FUN. can be split into the 2 blocks

MATH3411 and ␣IS␣FUN. .

M	0 1 0 0 1 1 0 1
A	0 1 0 0 0 0 0 1
T	1 1 0 1 0 1 0 0
H	0 1 0 0 1 0 0 0
3	0 0 1 1 0 0 1 1
4	1 0 1 1 0 1 0 0
1	1 0 1 1 0 0 0 1
1	1 0 1 1 0 0 0 1
$\overline{\text{ETB}}$	0 0 0 1 0 1 1 1

␣	1 0 1 0 0 0 0 0
I	1 1 0 0 1 0 0 1
S	0 1 0 1 0 0 1 1
␣	1 0 1 0 0 0 0 0
F	1 1 0 0 0 1 1 0
U	0 1 0 1 0 1 0 1
N	0 1 0 0 1 1 1 0
.	0 0 1 0 1 1 1 0
i	0 1 1 0 1 0 0 1

The encoded message is then MATH3411$\overline{\text{ETB}}$␣IS␣FUN.i .

Check sum (or check codeword)

To send long messages, we can partition them into 8-character blocks. This is the 9-character 8-bit ASCII.

- $8 \times 7 = 56$ information bits
- $8 + 8 = 16$ check bits
- $8 \times 9 = 72$ bits in total

Each check bit gives a check equation.

$$x_{11} + \cdots + x_{18} \equiv 0 \pmod 2$$

M	0 1 0 0 1 1 0 1
A	0 1 0 0 0 0 0 1
T	1 1 0 1 0 1 0 0
H	0 1 0 0 1 0 0 0
3	0 0 1 1 0 0 1 1
4	1 0 1 1 0 1 0 0
1	1 0 1 1 0 0 0 1
1	1 0 1 1 0 0 0 1
$\overline{\text{ETB}}$	0 0 0 1 0 1 1 1

Check sum (or check codeword)

To send long messages, we can partition them into 8-character blocks.
This is the 9-character 8-bit ASCII.

- $8 \times 7 = 56$ information bits
- $8 + 8 = 16$ check bits
- $8 \times 9 = 72$ bits in total

Each check bit gives a check equation.

$$x_{16} + \cdots + x_{96} \equiv 0 \pmod 2$$

M	0	1	0	0	1	1	0	1
A	0	1	0	0	0	0	0	1
T	1	1	0	1	0	1	0	0
H	0	1	0	0	1	0	0	0
3	0	0	1	1	0	0	1	1
4	1	0	1	1	0	1	0	0
1	1	0	1	1	0	0	0	1
1	1	0	1	1	0	0	0	1
$\overline{\mathrm{ETB}}$	0	0	0	1	0	1	1	1

Check sum (or check codeword)

To send long messages, we can partition them into 8-character blocks. This is the 9-character 8-bit ASCII.

- $8 \times 7 = 56$ information bits
- $8 + 8 = 16$ check bits
- $8 \times 9 = 72$ bits in total

Each check bit gives a check equation.

Note that

$$x_{91} + \cdots + x_{98} \equiv 0 \pmod 2$$

is the sum of the 8 first row equations
and is thus linearly dependent on them.

The 9-character 8-bit ASCII can be seen as a length 72 binary code with 72-bit codewords

$$\mathbf{x} = x_{11} \cdots x_{18} x_{21} \cdots x_{28} \cdots x_{91} \cdots x_{98}$$

Types of codes

- variable length code: codewords have different lengths
- block code: codewords have the same length
- t-error correcting code: code can always correct up to t errors
- systematic code: code with information digits and check digits distinct

Example

Morse code is a variable length code.
It is neither error correcting nor systematic.

Types of codes

- variable length code: codewords have different lengths
- block code: codewords have the same length
- t-error correcting code: code can always correct up to t errors
- systematic code: code with information digits and check digits distinct

Example

ISBN is a block code.
It is single-error detecting.
It is also systematic:

the 10th digit is a check digit;
the other 9 are information digits.

Types of codes

- variable length code: codewords have different lengths
- block code: codewords have the same length
- t-error correcting code: code can always correct up to t errors
- systematic code: code with information digits and check digits distinct

Example

ASCII (7-bit or 8-bit) is a block code.
It is not error correcting.
The 8-bit ASCII is systematic:

the 1st digit is a check bit;
the other 7 are information bits.

Types of codes

- variable length code: codewords have different lengths
- block code: codewords have the same length
- t-error correcting code: code can always correct up to t errors
- systematic code: code with information digits and check digits distinct

Example

9-character 8-bit ASCII is a block code.
It is single-error correcting.
It is also systematic,
with 16 parity/check bits and 56 information bits.

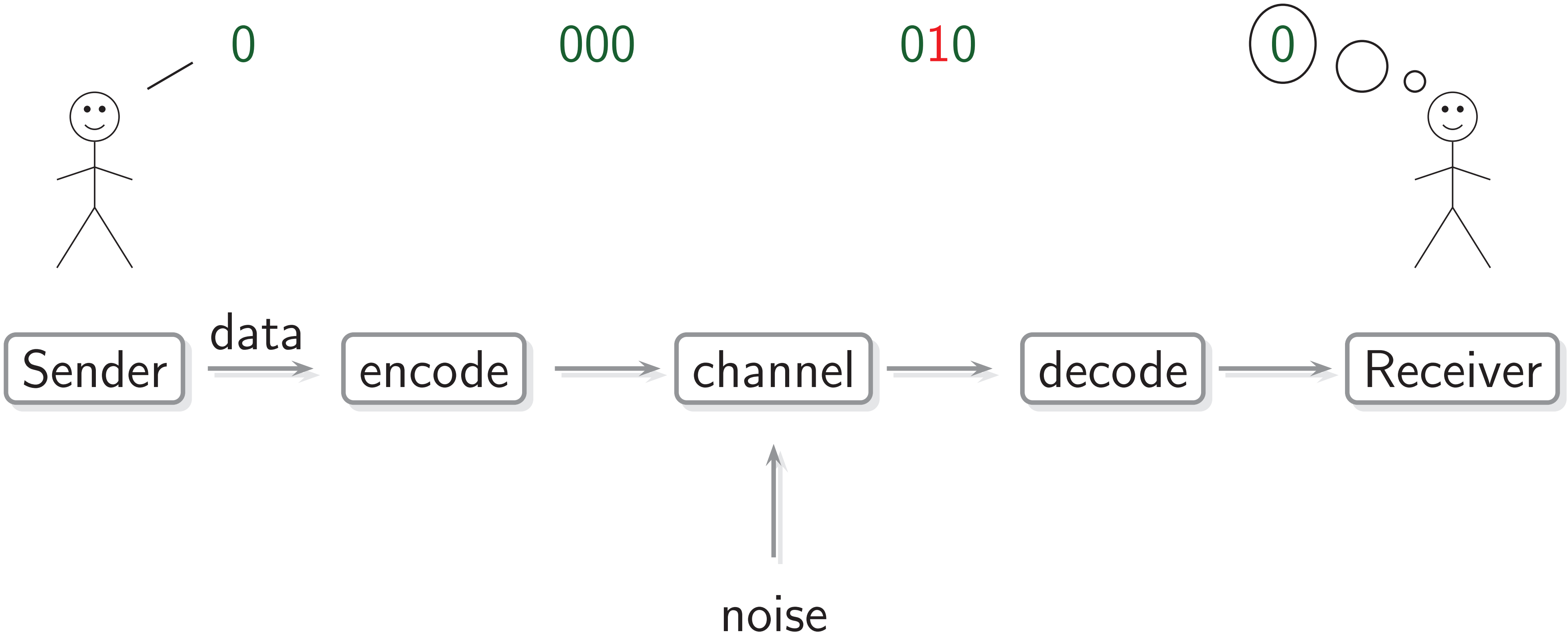
Binary Repetition Codes

- A binary r-repetition code encodes $0 \to \overbrace{0\cdots 0}^{r}$ and $1 \to \overbrace{1\cdots 1}^{r}$

Binary Repetition Codes

- A binary r-repetition code encodes $0 \to \overbrace{0 \cdots 0}^{r}$ and $1 \to \overbrace{1 \cdots 1}^{r}$

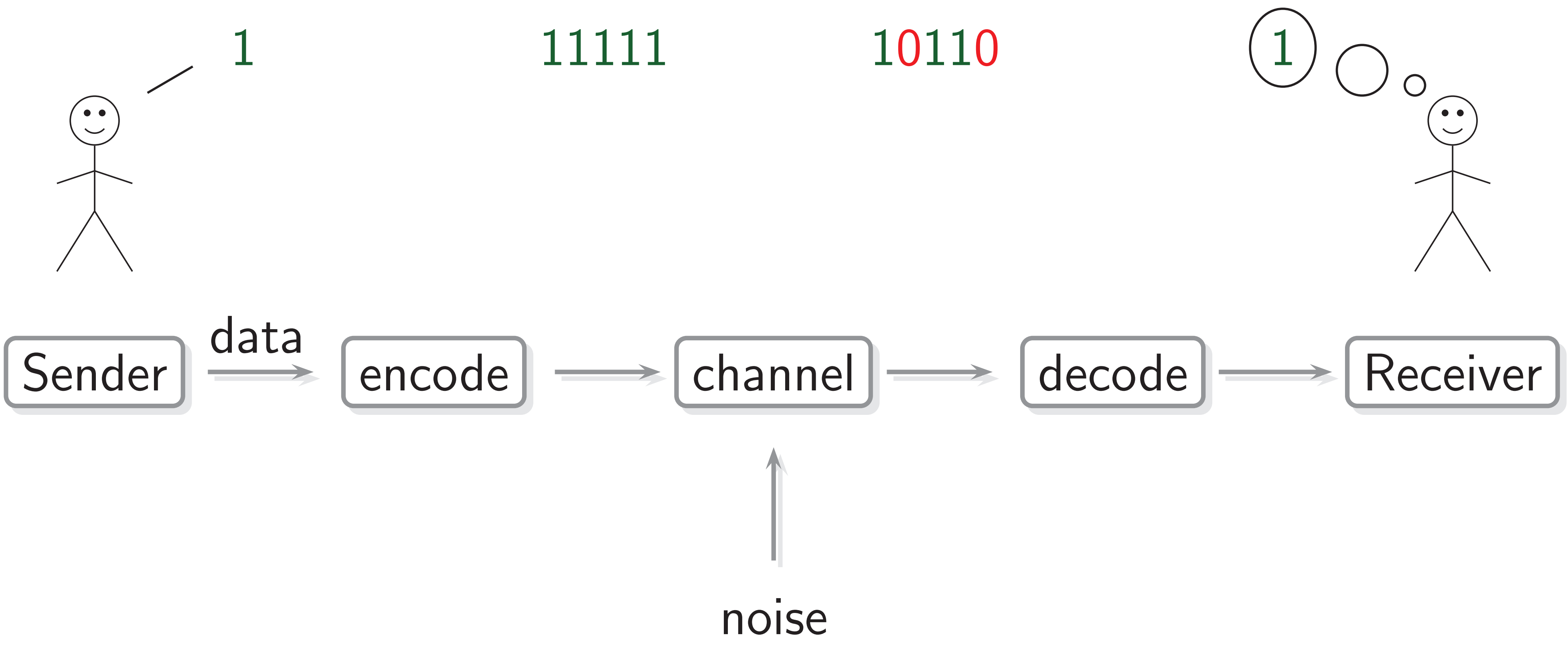
binary triple-repetition code



Binary Repetition Codes

- A binary r-repetition code encodes $0 \to \overbrace{0 \cdots 0}^{r}$ and $1 \to \overbrace{1 \cdots 1}^{r}$

binary 5-repetition code



Binary Repetition Codes

- A binary r-repetition code encodes $0 \to \overbrace{0\cdots0}^{r}$ and $1 \to \overbrace{1\cdots1}^{r}$

Theorem

The binary $(2t+1)$-repetition code is t-error correcting.
The binary $2t$-repetition code is $(t-1)$-error correcting & t-error detecting.

Example

We receive the corrupted binary 8-repetition encoded word 01101001.
Since there are equally many 0s and 1s, we cannot decode this word.
Our decoding therefore Fails.
However, we can see (detect) that there are 4 errors.

Binary Repetition Codes

Theorem

The binary $(2t+1)$-repetition code is t-error correcting.
The binary $2t$-repetition code is $(t-1)$-error correcting & t-error detecting.

Example

We receive the corrupted binary 8-repetition encoded word 01101001.
Since there are equally many 0s and 1s, we cannot decode this word.
Our decoding therefore Fails.
However, we can see (detect) that there are 4 errors.

Example

We receive the corrupted binary 8-repetition encoded word 01101101.
There are more 1s than 0s,
so it is natural to correct to 11111111 and decode to 1.
We write this as 01101101 $\rightarrow$ 1.

Binary Repetition Codes

Theorem

The binary $(2t+1)$-repetition code is t-error correcting.
The binary $2t$-repetition code is $(t-1)$-error correcting & t-error detecting.

There are many decoding strategies for decoding repetition codes.
We can choose any of these - but only one of these!

Example

For a 5-repetition code, we can choose from the following stategies:

STRATEGY 1

Correct up to 2 errors.

$$\begin{aligned} 00001 &\rightarrow 0 \\ 00011 &\rightarrow 0 \end{aligned}$$

Binary Repetition Codes

Theorem

The binary $(2t+1)$-repetition code is t-error correcting.
The binary $2t$-repetition code is $(t-1)$-error correcting & t-error detecting.

There are many decoding strategies for decoding repetition codes.
We can choose any of these - but only one of these!

Example

For a 5-repetition code, we can choose from the following stategies:

STRATEGY 2

Correct 1 error
or detect 2 or 3 errors.

$$\begin{array}{ccc} 00001 & \to & 0 \\ 00011 & \to & \mathsf{F} \\ 00111 & \to & \mathsf{F} \end{array}$$

Here, our decoding strategy failed for two of the words
- but we detected that there were 2 or 3 errors.

Binary Repetition Codes

Theorem

The binary $(2t+1)$-repetition code is t-error correcting.
The binary $2t$-repetition code is $(t-1)$-error correcting & t-error detecting.

There are many decoding strategies for decoding repetition codes.
We can choose any of these - but only one of these!

Example

For a 5-repetition code, we can choose from the following stategies:

STRATEGY 3
Detect up to 4 errors.

$$\begin{aligned} 00001 &\rightarrow \text{F} \\ 00011 &\rightarrow \text{F} \\ 00111 &\rightarrow \text{F} \\ 01111 &\rightarrow \text{F} \end{aligned}$$

Here, our decoding strategy was a complete failure
- but we did detect that there were errors.

Binary Repetition Codes

Theorem

The binary $(2t+1)$-repetition code is t-error correcting.
The binary $2t$-repetition code is $(t-1)$-error correcting & t-error detecting.

There are many decoding strategies for decoding repetition codes.
We can choose any of these - but only one of these!

Example

At best, we can correct 2 errors, so the code is 2-error correcting.

Example

For a 6-repetition code ($t = 3$), we can choose from these stategies:

STRATEGY 1 Correct up to 2 errors or detect up to 3 errors.

STRATEGY 2 Correct 1 error or detect up to 4 errors.

STRATEGY 3 Detect up to 5 errors.

		STRATEGY		
		1	2	3
000000	$\to$	0	0	0
000001	$\to$	0	0	F
000011	$\to$	0	F	F
000111	$\to$	F	F	F
001111	$\to$	1	F	F
011111	$\to$	1	1	F
111111	$\to$	1	1	1

Example

For a 7-repetition code ($t = 3$), we can choose from these stategies:

STRATEGY 1 Correct up to 3 errors.
STRATEGY 2 Correct up to 2 errors or detect up to 4 errors.
STRATEGY 3 Correct 1 error or detect up to 5 errors.
STRATEGY 4 Detect up to 6 errors.

		STRATEGY			
		1	2	3	4
0000000	$\rightarrow$	0	0	0	0
0000001	$\rightarrow$	0	0	0	F
0000011	$\rightarrow$	0	0	F	F
0000111	$\rightarrow$	0	F	F	F
0001111	$\rightarrow$	1	F	F	F
0011111	$\rightarrow$	1	1	F	F
0111111	$\rightarrow$	1	1	1	F
1111111	$\rightarrow$	1	1	1	1

Check sum (or check codeword)

To send the message MATH3411, we encode each character in ASCII and add a binary-parity check sum (this is a burst code).

Here, the check sum is 0010111, ASCII for the symbol $\overline{\mathrm{ETB}}$.
The message sent is then MATH3411$\overline{\mathrm{ETB}}$.

	ASCII
M	1 0 0 1 1 0 1
A	1 0 0 0 0 0 1
T	1 0 1 0 1 0 0
H	1 0 0 1 0 0 0
3	0 1 1 0 0 1 1
4	0 1 1 0 1 0 0
1	0 1 1 0 0 0 1
1	0 1 1 0 0 0 1
$\overline{\mathrm{ETB}}$	0 0 1 0 1 1 1