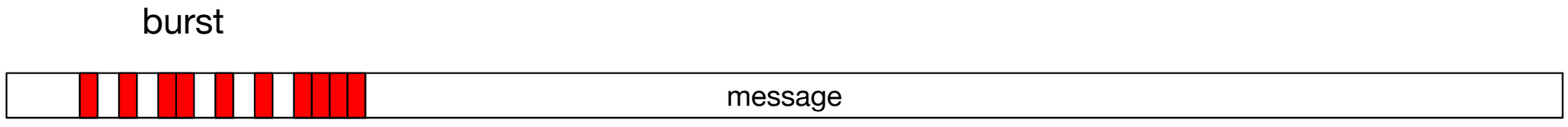
Error and Erasure Correction

Thomas Schwarz, SJ

Assumptions

- We have a message of n bits
  - Each bit has a chance of being corrupted and being interpreted the wrong way: an error
  - Error:
    - Burst errors: chance for corruption is larger for nearby bits



Assumptions

- The bit-error rate $p = \frac{e}{n}$ is proportion of # errors e over number of bits n

- If the bit-error rate $p = \frac{1}{2}$, then no data can be transmitted
- If the bit-error rate $p > \frac{1}{2}$: Flip bits (0 -> 1, 1 -> 0) and obtain a bit-error rate $p < \frac{1}{2}$
- Thus: assume $p < \frac{1}{2}$

Dealing with Errors

- Error Detection:
  - Detect that an error has occurred
    - Prevent action based on erroneous information
      - E.g. Error detecting DRAM:
        - Do not execute bad instructions and risk destroying more data
- In a communication channel:
  - Ask to retransmit corrupted packages

Dealing with Errors

- Forward Error Correction
  - Use redundancy in order to correct errors or declare that there are too many errors

Dealing with Errors

- Undetected errors:
  - The message contains errors but we do not see them
- Uncorrectable errors:
  - We do not have sufficient information to detect the errors

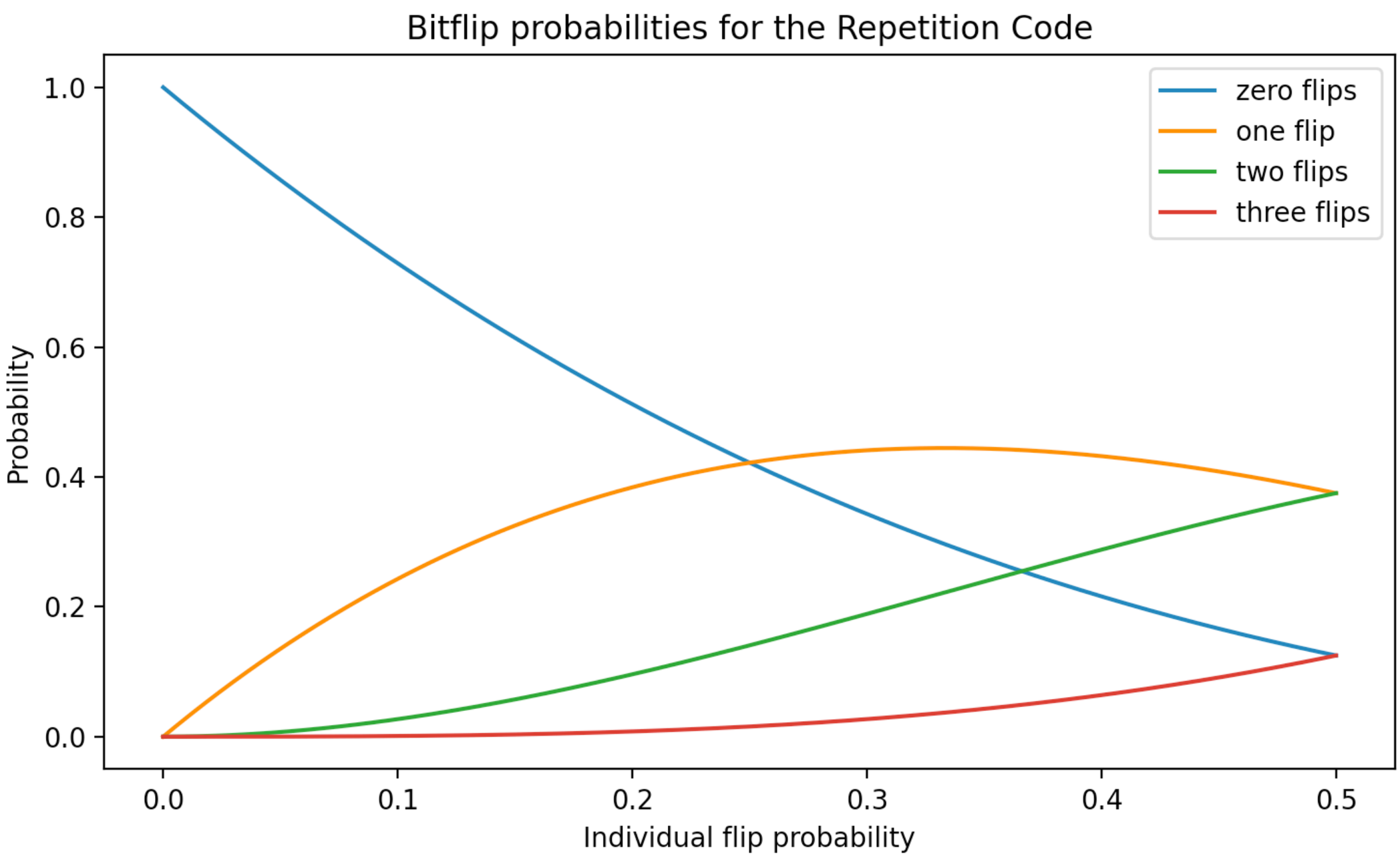
Simple Example

- The repetition code:
  - (User) message has length $m = 1$
    - Two possibilities: message is 0 or message is 1
  - We send 000, 111
    - Assume bits are flipped independently of other bits with probability $p < 0.5$
    -

Simple Example

- Assume we are sending 0
  - We receive 000 with probability $(1-p)^3$
  - We receive 001, 010, 100 with probability $(1-p)^2 p$ each
  - We receive 011, 101, 110 with probability $(1-p)p^2$ each
  - We receive 111 with probability p^3

Simple Example



Error Detection

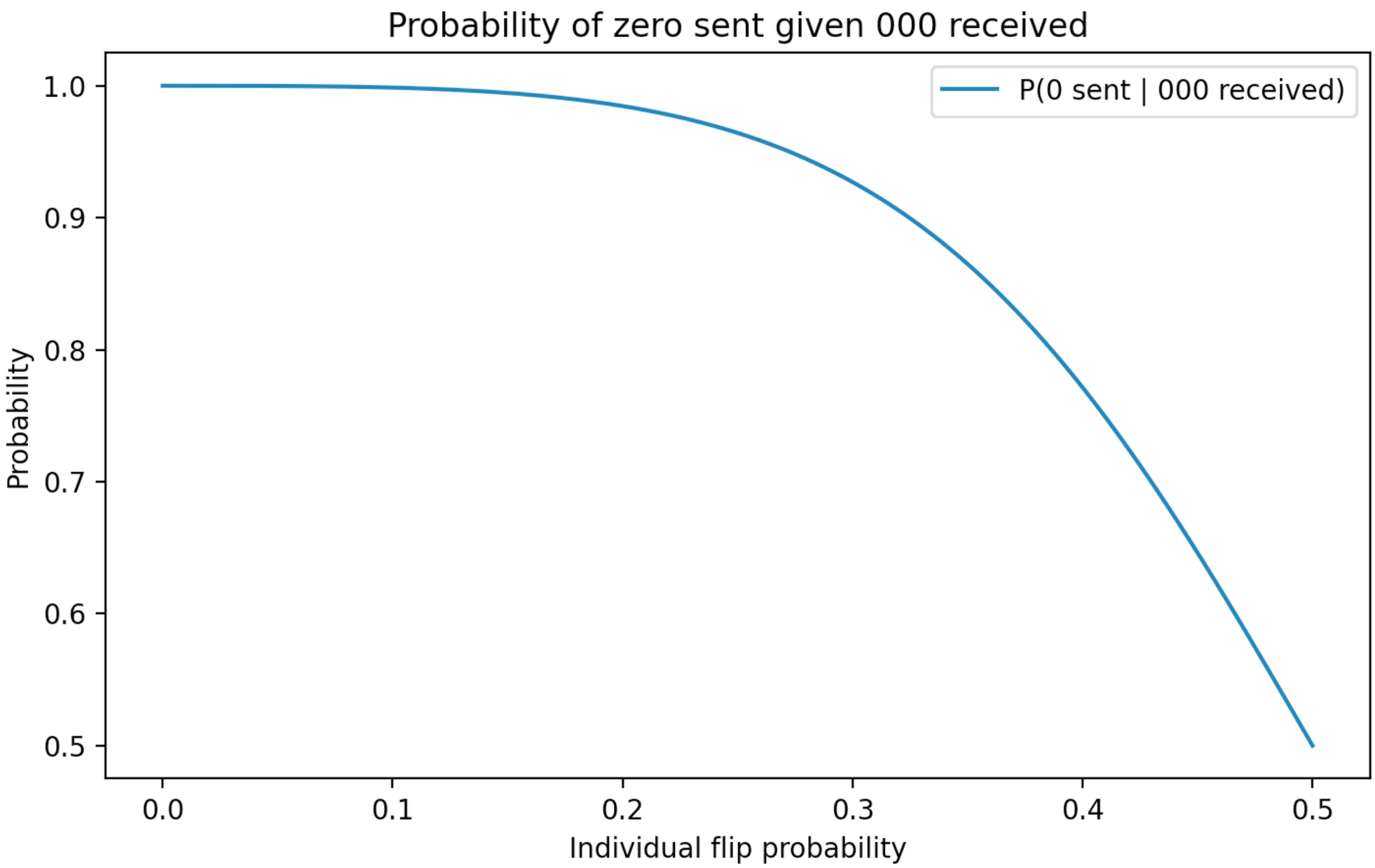
- How should we decode?
  - If we receive 000?

$$P(\text{000 got}) = P(\text{000 got} \mid \text{0 sent}) \cdot P(\text{0 sent}) + P(\text{000 got} \mid \text{1 sent}) \cdot P(\text{1 sent})$$

$$= 0.5 \cdot (1-p)^3 + 0.5 \cdot p^3$$

- We want to find the original message sent
- Calculate the conditional probability using Bayes Theorem
  - $P(\text{0 sent} \mid \text{got 000})$
  - $$= \frac{P(\text{got 000} \mid \text{0 sent}) \cdot P(\text{0 sent})}{P(\text{got 000})}$$
  - $$= \frac{(1-p)^3 \cdot \frac{1}{2}}{\frac{1}{2}(1-p)^3 + \frac{1}{2}p^3}$$ (assuming messages are equally likely)

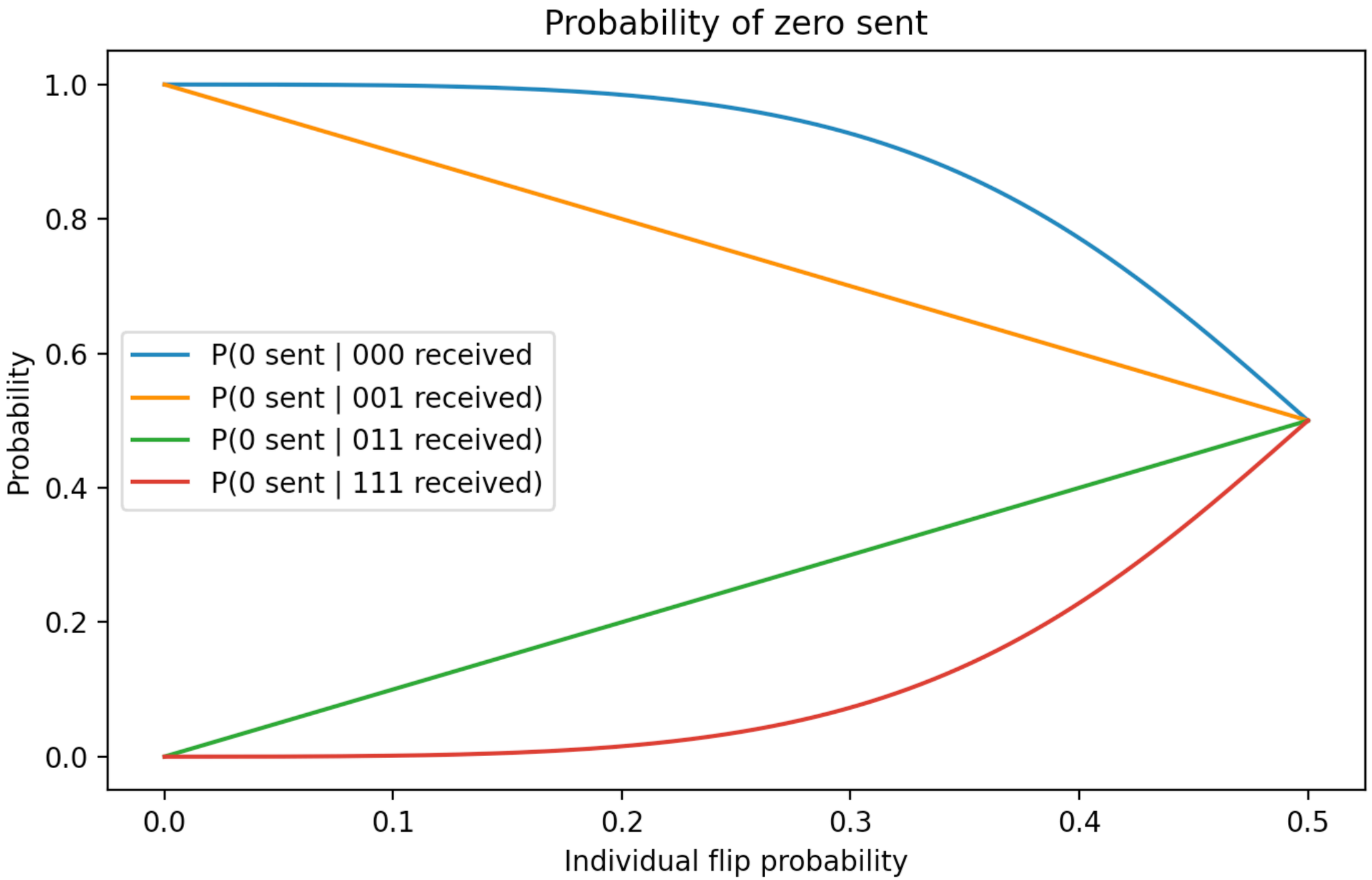
Error Detection



Simple Example

- Probability to receive "001"
  - $\frac{1}{2}(1-p)^2 p + \frac{1}{2}(1-p)p^2$
- Probability 0 was sent when 001 was received:
  - $P(\text{0 sent} \mid \text{001 received}) = \frac{(1-p)^2 p \cdot \frac{1}{2}}{\frac{1}{2}(1-p)^2 p + \frac{1}{2}(1-p)p^2}$
  - $= \frac{(1-p)}{(1-p)+p} = 1-p$

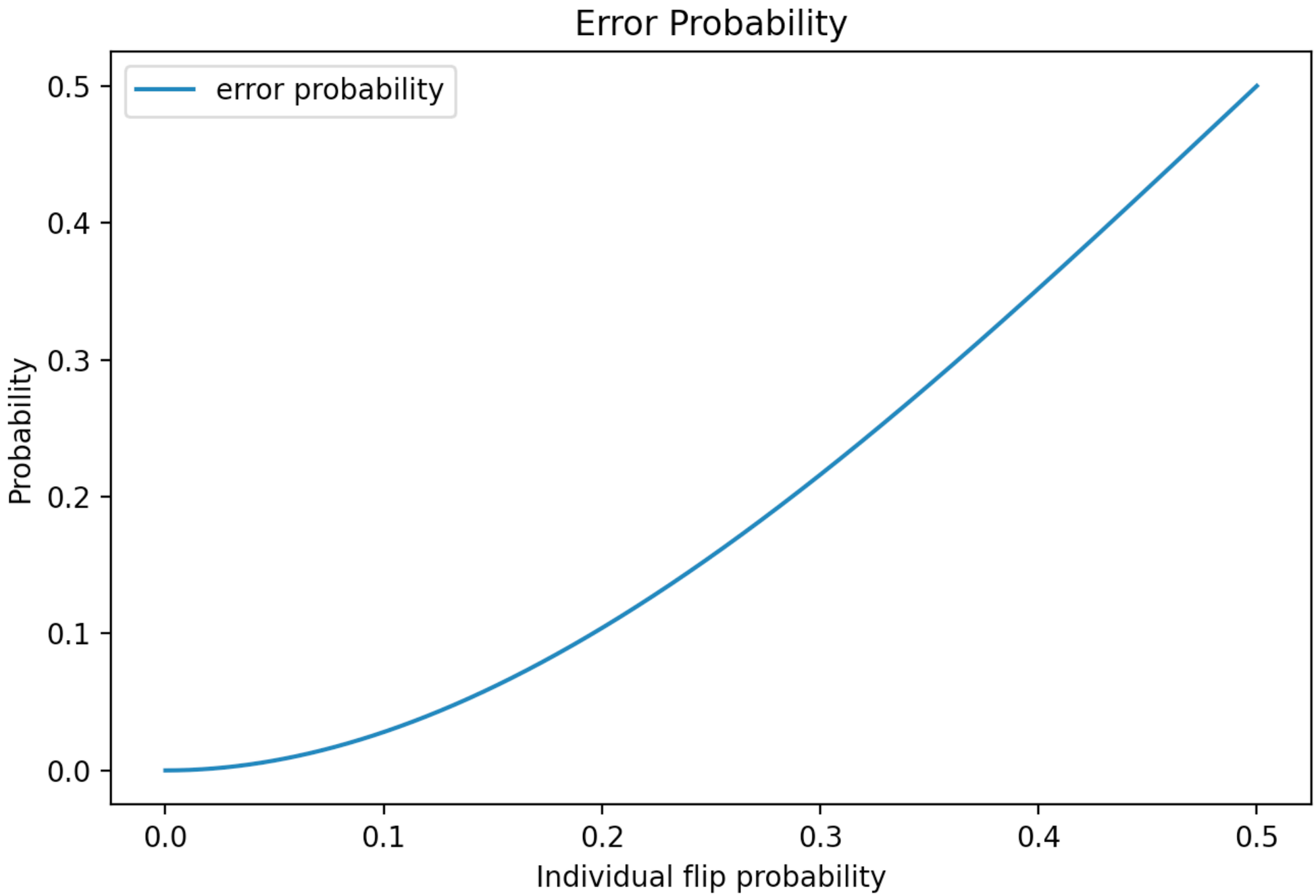
Simple Example



Simple Example

- Highest likelihood decoding:
  - If we see three or two zeroes:
    - Assume that message was 0
  - If we see one or no zeroes:
    - Assume that message was 1
- This still gives us opportunities for error

Simple Example



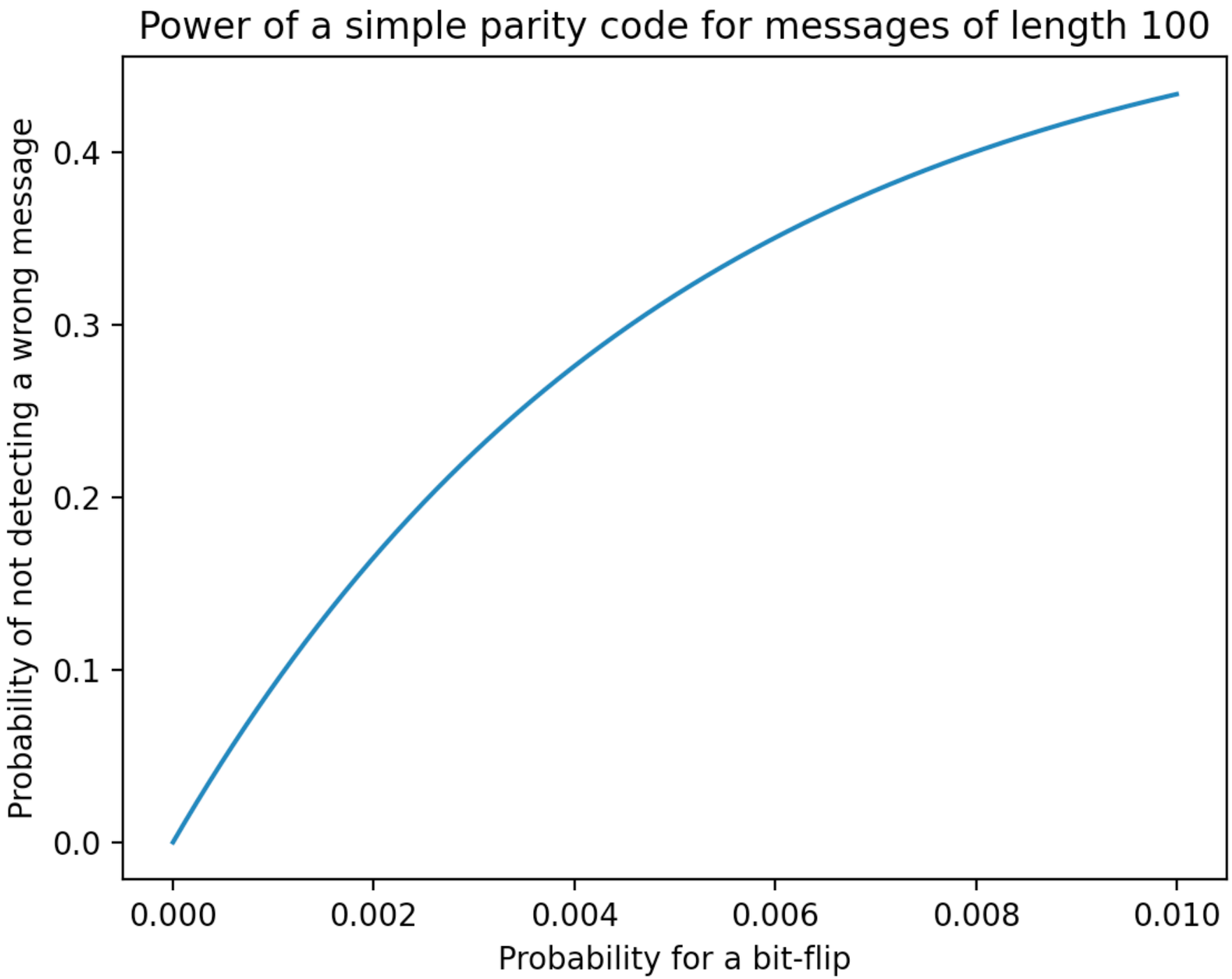
Simple Example

- If we expand this to a four-fold repetition code:
  - If we see two zeroes and two ones:
    - No way to decide between the two messages (unless one is more likely than the other)

Error Detection

- Single parity code:
  - Add the parity of the n message bits
  - $(x_1, x_2, \ldots x_n) \longrightarrow (x_1, x_2, \ldots x_n, p)$
    - With $p = x_1 \oplus x_2 \oplus x_3 \ldots \oplus x_n$
- Discovers 1, 3, 5, … errors
-

Error Detection



Cyclic Redundancy Codes

- Cyclic Redundancy Code
  - Uses calculations over the field $\mathscr{F}_2 = \{0,1\}$
  - Represent a bit string $x_0, x_1, x_2, \ldots, x_{n-1}$ as a polynomial
    $x_0 t^{n-1} + x_1 t^{n-2} + x^2 t^{n-3} + \cdots + x_{n-2} t + x_{n-1} \in \mathscr{F}_2[t]$
  - Example:
    - 0xa3 corresponds to 10100011 to $t^7 + t^5 + t + 1$

Cyclic Redundancy Codes

- We now use a "generator polynomial" $g(t) \in \mathscr{F}_2[t]$ to define the "code words"
  - Code words are all encoded messages that are good (have no errors)
  - For a CRC: code words are multiples of the generator polynomial
    - Take $g(t) = t^3 + t + 1$
    - Take length of encoded word to be 5

Cyclic Redundancy Codes

- Code words are
  - $0 \times (t^3 + t + 1)$ corresponds to 00000
  - $1 \times (t^3 + t + 1)$ corresponds to 01011
  - $t \times (t^3 + t + 1) = t^4 + t^2 + t$ corresponds to 10110
  - $(t + 1) \times (t^3 + t + 1) = t^4 + t^2 + t + t^3 + t + 1 = t^4 + t^3 + t^2 + 1$ corresponds to 11101

Cyclic Redundancy Codes

- How do we encode?
  - Start out with the message
    - Example 11
  - Pad with zeroes: 11000
  - Divide with remainder
    - $\begin{array}{r} 11000 \\ \underline{1011} \\ 0111 \end{array}$ $\qquad$ $\begin{array}{r} 11000 \\ \underline{10110} \\ 01110 \end{array}$

$$t^4 + t^3 + 0 \cdot t^2 + 0 \cdot t + 0 = t \times (t^3 + t + 1) + (t^3 + t^2 + t + 0)$$

t because I place the 1011 to the left.

Cyclic Redundancy Codes

$$\begin{array}{r} 11000 \\ 10110 \\ \hline 01110 \\ 1011 \\ \hline 0100 \end{array}$$

- $t^4 + t^3 = (t+1) \times (t^3 + t + 1) + (t^2 + t)$
- $t^3 + t^2 + t = 1 \cdot (t^3 + t + 1) + (t^2 + 1)$
  - Thus $t^4 + t^3 = t \times (t^3 + t + 1) + (t^2)$

- Use the remainder (110) in order to fill in the last three bits
  - 11110

Cyclic Redundancy Codes

- More elaborate example:
  - Same generator polynomial 1101, but $m = 11$
  - Remainder uses three bits, so 8 bits for information
  - Encode 0xa3 = 1010 0011.
    - Divide with remainder 1010 0011 000

Cyclic Redundancy Codes

```
101 0001 1000  = ____ * 1101 + ___
```

Cyclic Redundancy Codes

```
101 0001 1000  = 1____ * 1101 + ___
110 1
```

Cyclic Redundancy Codes

```
101 0001 1000  = 1____ * 1101 + ___
110 1
-----------
011 10
```

Cyclic Redundancy Codes

```
101 0001 1000   = 110__ * 1101 + ___
110 1000 0000
-------------
011 1001 1000
 11 0100 0000
-------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
```

Cyclic Redundancy Codes

```
101 0001 1000   = 1101__ * 1101 + ___
110 1000 0000
-------------
011 1001 1000
 11 0100 0000
-------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
    1101 0000
-------------
```

Cyclic Redundancy Codes

```
101 0001 1000   = 1101__ * 1101 + ___
110 1000 0000
-------------
011 1001 1000
 11 0100 0000
-------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
    1101 0000
-------------
     000 1000
```

Cyclic Redundancy Codes

```
101 0001 1000   = 11010__ * 1101 + ___
110 1000 0000
---------------
011 1001 1000
 11 0100 0000
---------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
    1101 0000
-------------
     000 1000
     000 0000
```

Cyclic Redundancy Codes

```
101 0001 1000   = 11010__ * 1101 + ___
110 1000 0000
-------------
011 1001 1000
 11 0100 0000
-------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
    1101 0000
-------------
     000 1000
     000 0000
-------------
      00 1000
```

Cyclic Redundancy Codes

```
101 0001 1000   = 110100__ * 1101 + ___
110 1000 0000
-------------
011 1001 1000
 11 0100 0000
-------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
    1101 0000
-------------
     000 1000
     000 0000
-------------
      00 1000
      00 0000
```

Cyclic Redundancy Codes

```
101 0001 1000   = 110100__ * 1101 + ____
110 1000 0000
-------------
011 1001 1000
 11 0100 0000
-------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
    1101 0000
-------------
     000 1000
     000 0000
-------------
      00 1000
      00 0000
-------------
       0 1000
```

Cyclic Redundancy Codes

```
101 0001 1000   = 1101000_ * 1101 + ___
110 1000 0000
---------------
011 1001 1000
 11 0100 0000
---------------
  0 1101 1000
  0 0000 0000
---------------
    1101 1000
    1101 0000
---------------
    000  1000
    000  0000
---------------
      00 1000
      00 0000
---------------
       0 1000
       0 0000
```

Cyclic Redundancy Codes

```
101 0001 1000   = 1101000_ * 1101 + ___
110 1000 0000
-------------
011 1001 1000
 11 0100 0000
-------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
    1101 0000
-------------
     000 1000
     000 0000
-------------
      00 1000
      00 0000
-------------
       0 1000
       0 0000
-------------
         1000
```

Cyclic Redundancy Codes

```
101 0001 1000   = 11010001 * 1101 + ___
110 1000 0000
-------------
011 1001 1000
 11 0100 0000
-------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
    1101 0000
-------------
     000 1000
     000 0000
-------------
      00 1000
      00 0000
-------------
       0 1000
       0 0000
-------------
         1000
         1101
-------------
          101
```

Cyclic Redundancy Codes

```
101 0001 1000   = 11010001 * 1101 + 101
110 1000 0000
-------------
011 1001 1000
 11 0100 0000
-------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
    1101 0000
-------------
     000 1000
     000 0000
-------------
      00 1000
      00 0000
-------------
       0 1000
       0 0000
-------------
         1000
         1101
-------------
          101
```

Cyclic Redundancy Codes

```
101 0001 1000   = 11010001 * 1101 + 101
110 1000 0000
-------------
011 1001 1000
 11 0100 0000
-------------
  0 1101 1000
  0 0000 0000
-------------
    1101 1000
    1101 0000
-------------
     000 1000
     000 0000
-------------
      00 1000
      00 0000
-------------
       0 1000
       0 0000
-------------
         1000
         1101
-------------
          101
```

- The message is
  - 101 0001 1101
- This corresponds to a polynomial divisible by the generator polynomial

Cyclic Redundancy Codes

- The factorization of the generator polynomial dictates the properties of the CRC
  - Only error patterns forming a polynomial that is a multiple of the generator polynomial are not detected
  - Creates a premium for irreducible (i.e. prime) polynomials

Cyclic Redundancy Codes

- Example:
  - Generator polynomial $t^3 + t + 1$
  - Errors 7 bits apart:

$$t^7 + 1 = (t + 1)(t^3 + t^2 + 1)(t^3 + t + 1)$$

  - Cannot be detected!

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{l}
10010000100000 \quad = \quad 1 \\
\underline{1101} \\
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
10010000010000 & = & 1 \\
\underline{1101} & & \\
01000 & &
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
10010000010000 & = & 11 \\
\underline{1101} & & \\
01000 & & \\
\underline{1101} & &
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
10010000010000 & = & 11 \\
\underline{1101} & & \\
01000 & & \\
\underline{1101} & & \\
0101 & &
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
10010000010000 & = & 11 \\
\underline{1101} & & \\
01000 & & \\
\underline{1101} & & \\
01010 & &
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
10010000010000 & = & 11 \\
\underline{1101} & & \\
01000 & & \\
\underline{1101} & & \\
1010 & &
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
1001000010000 & = & 111 \\
\underline{1101} & & \\
01000 & & \\
\underline{1101} & & \\
01010 & & \\
\underline{1101} & & \\
0111 & &
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
10010000010000 & = & 1111 \\
1101 & & \\
\hline
01000 & & \\
1101 & & \\
\hline
01010 & & \\
1101 & & \\
\hline
01110 & & \\
1101 & & \\
\hline
0110 & &
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

```
10010000010000    =   11110
1101
----------
01000
 1101
----------
 01010
  1101
----------
 01111
  1101
----------
 00110
  0000
```

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
1001000010000 & = & 11110 \\
\underline{1101} & & \\
01000 & & \\
\underline{1101} & & \\
01010 & & \\
\underline{1101} & & \\
01110 & & \\
\underline{1101} & & \\
00110 & & \\
\underline{0000} & & \\
1101 & &
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
1001000010000 & = & 111101 \\
\underline{1101} & & \\
01000 & & \\
\underline{1101} & & \\
01010 & & \\
\underline{1101} & & \\
01110 & & \\
\underline{1101} & & \\
00110 & & \\
\underline{0000} & & \\
01101 & & \\
\underline{1101} & & \\
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
1001000010000 & = & 111101 \\
\underline{1101} & & \\
01000 & & \\
\underline{1101} & & \\
01010 & & \\
\underline{1101} & & \\
01110 & & \\
\underline{1101} & & \\
00110 & & \\
\underline{0000} & & \\
01101 & & \\
\underline{1101} & & \\
0000 & &
\end{array}
$$

Cyclic Redundancy Code Example

- 1001000010 using the same generator polynomial

$$
\begin{array}{rcl}
1001000010000 & = & 1111010 \\
1101 & & \\
\hline
01000 & & \\
1101 & & \\
\hline
01010 & & \\
1101 & & \\
\hline
01110 & & \\
1101 & & \\
\hline
00110 & & \\
0000 & & \\
\hline
1101 & & \\
1101 & & \\
\hline
0000 & & \\
0000 & &
\end{array}
$$

Cyclic Redundancy Code Example

```
10010000010000    =  1111010
1101
-----------
01000
 1101
-----------
 01010
  1101
-----------
  01110
   1101
-----------
   00110
    0000
-----------
    1101
    1101
-----------
    0000
    0000
-----------
    0000
```

Cyclic Redundancy Code Example

```
10010000010000   =  111101000
1101
----
01000
 1101
----
 01010
  1101
----
  01110
   1101
----
   00110
    0000
----
    1101
    1101
----
    0000
    0000
----
    0000
    0000
----
    0000
    0000
```

Cyclic Redundancy Code Example

```
10010000010000    =  111101000
1101
----
01000
 1101
----
 01010
  1101
----
 01110
  1101
----
 00110
  0000
----
  1101
  1101
----
  0000
  0000
----
  0000
  0000
----
  0000
  0000
----
  0000
```

Cyclic Redundancy Code Example

```
10010000010000   =  1111010000
1101
----
01000
 1101
----
 01010
  1101
----
  01110
   1101
  -----
   00110
    0000
   -----
    1101
    1101
   -----
    0000
    0000
   -----
    0000
    0000
   -----
     0000
     0000
    -----
      0000
      0000
```

Cyclic Redundancy Code Example

```
10010000010000   =  1111010000
1101
----
01000
 1101
----
 01010
  1101
----
  01110
   1101
----
   00110
    0000
----
    1101
    1101
----
    0000
    0000
----
     000 0
     0000
----
     0000
     0000
----
     0000
     0000
----
      000
```

Cyclic Redundancy Code Example

$$1001000010000 = 1111010000 \cdot 1101 + 000$$

```
1001000010000
1101
____________
01000
 1101
____________
 01010
  1101
____________
  01110
   1101
____________
   00110
    0000
____________
     1101
     1101
____________
      0000
      0000
____________
       000 0
       0000
____________
        0000
        0000
____________
         0000
         0000
____________
          000
```

Since the remainder is 000, the polynomial did not change

Cyclic Redundancy Codes

- IEEE 802.3 Standard Generating Polynomial

$$t^{32} + t^{26} + t^{23} + t^{22} + t^{16} + t^{12} + t^{11} + t^{10} + t^8 + t^7 + t^5 + t^4 + t^2 + t + 1$$

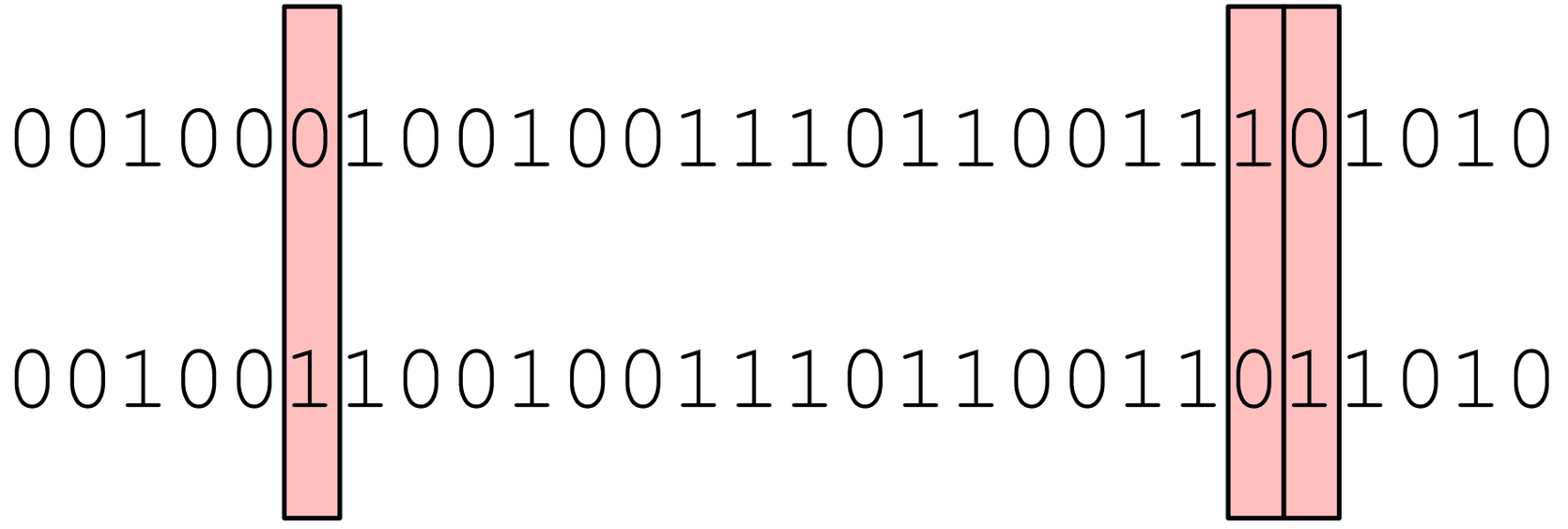
  - Generates 32 bit parity
  - Detects error burst of length up to 32
  - Guaranteed to detect any 3 errors in a message of 1MTU (1500 B)

- (But there are better polynomials known:

  • Philip Koopman: 32-Bit Cyclic Redundancy Codes for Internet Applications, DSN 2002

Error Correction

- We measure distance between bit-strings by the *Hamming* distance:
  - Number of positions in which two strings differ



Error Correction

- Hamming weight of a string:
  - Number of one-bits
  - $\delta_{\text{Hamming}}(\mathbf{x}, \mathbf{y}) = \text{weight}_{\text{Hamming}}(\mathbf{x} \oplus \mathbf{y})$
  - Example:

```
  00100010010011101100111010 10
⊕ 00100110010011101100110110 10
------------------------------
  00000 1 0000000000000000 11 0000
```

Error Correction

- Let $\mathbf{m}$ be the message sent (including parity bits)
- Let $\mathbf{r}$ be the message received
- The symmetric difference $\mathbf{e} = \mathbf{r} \oplus \mathbf{m}$ is the transmission error

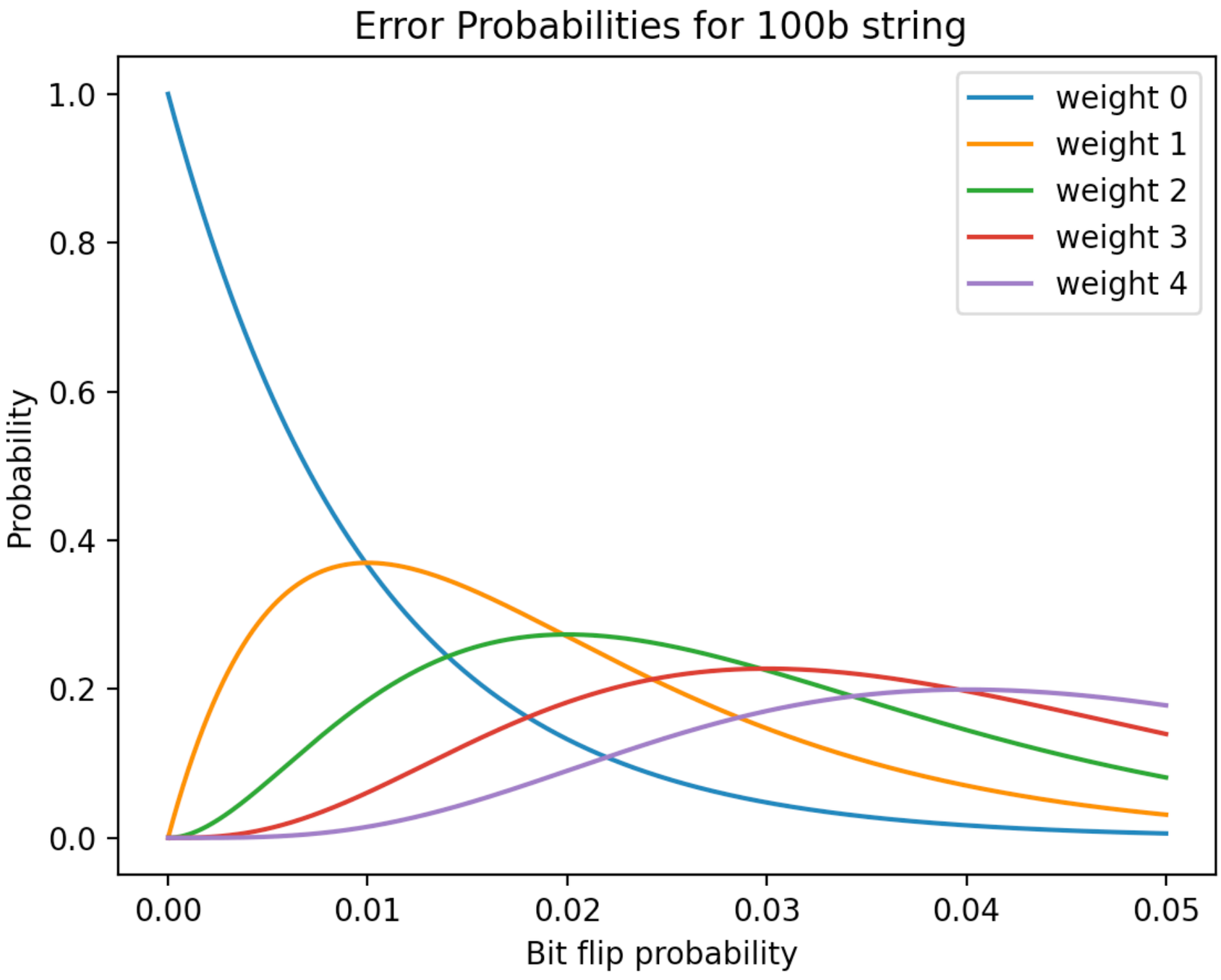
Error Correction

- Transmission error model:
  - All bits are flipped independently with equal probability p

Error Correction

- If $\mathbf{m} \in \mathcal{F}_2^n$ (n bits per message):
  - Probability of error vector with weight 0 (no error)
    - $(1-p)^n$
  - Probability of error vector with weight 1 (one error)
    - $\binom{n}{1} p(1-p)^{n-1}$
  - Probability of error vector with weight 2 (two errors)
    - $\binom{n}{2} p^2(1-p)^{n-2}$
  - et cetera: Binomial Distribution Probability Mass Function

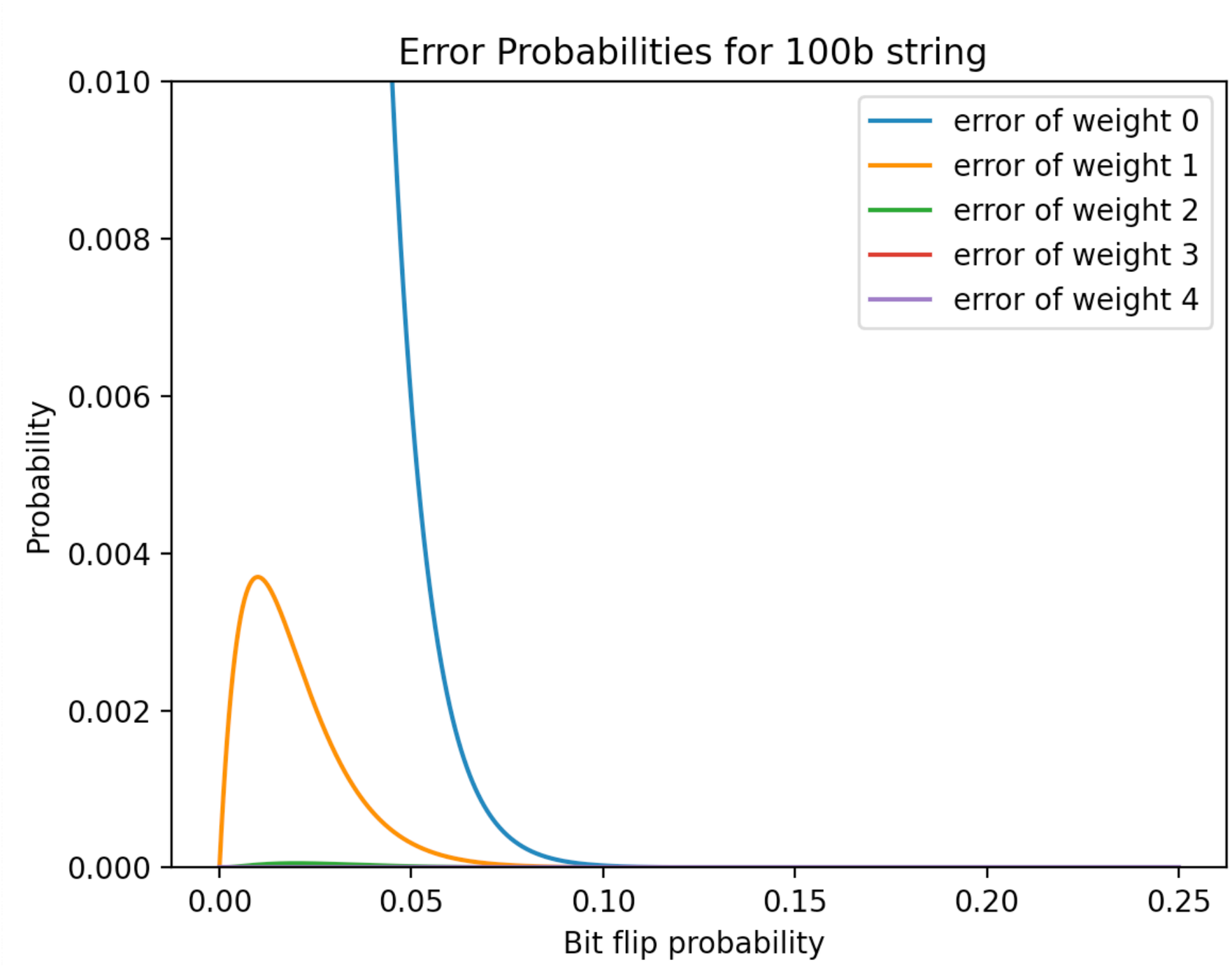
Error Correction



Error Correction

- This is very different from the probability of a given error vector
  - $w(\mathbf{e}) = 0$ with probability $(1-p)^n$
  - $w(\mathbf{e}) = 1$ with probability $(1-p)^{n-1}p$
  - $w(\mathbf{e}) = 2$ with probability $(1-p)^{n-2}p^2$
  - $w(\mathbf{e}) = 3$ with probability $(1-p)^{n-3}p^3$

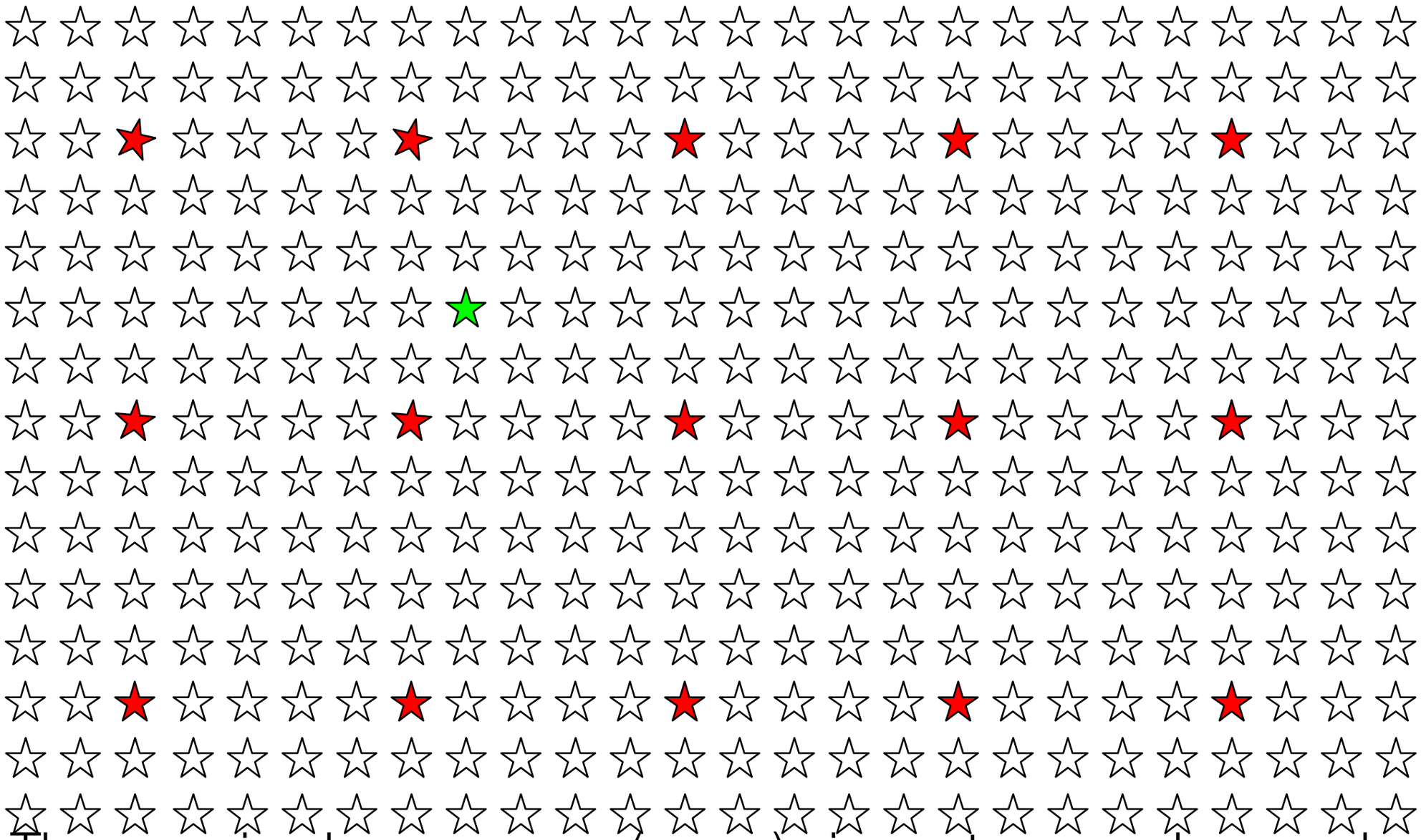
Error Correction



Error Correction

- Maximum Likelihood Decoding
  - If we receive a message $\mathbf{r}$
    - We want to distinguish between two cases:
      - Original message was $\mathbf{m}_1$ or was $\mathbf{m}_2$
      - Putative errors are $\mathbf{e}_1 = \mathbf{r} \oplus \mathbf{m}_1$ or $\mathbf{e}_2 = \mathbf{r} \oplus \mathbf{m}_2$
- We select the error vector with smaller weight as the more likely one

Error Detection



The received message (green) is not a code word, therefore an error has happened. Maximum likelihood encoding assumes the most likely error, which would be the one with minimum number of ones. Therefore, pick the closest code word.

Error Correction

- Error correcting codes:
  - Code words have minimum Hamming distance d.
  - Can correct up to $\lfloor \frac{d-1}{2} \rfloor$ bit errors.

Error Correction

- Systematic Codes
  - Take the message of length n and add p parity bits
  - Advantage:
    - If you can check the accuracy of the parity bits, you do not need to decode the message but just extract it.

Error Correction

- Linear Codes: (of length $n + p$)
  - Set of code words is a sub-vector-space of $\mathbb{Z}_2^{n+p}$
  - Defined by a *parity check matrix* $\mathscr{H} \in \mathbb{Z}_2^{p \times (n+p)}$
    - Code words $= \{\mathbf{x} \in \mathbb{Z}_2^{n+p} \mid \mathscr{H} \cdot \mathbf{x} = \mathbf{0}\}$
- Or by a generator matrix $\mathbb{G} \in \mathbb{Z}_2^{(n+p) \times n}$
  - Code words $\{\mathbb{G} \cdot \mathbf{x} \mid \mathbf{x} = \mathbb{Z}_2^n\}$

Error Correction

- If the minimum Hamming distance between two code words is $2d+1$
  - Then we can correct d errors

- The received word with d errors has distance d from the intended code word
- And distance at least $d+1$ from all other code words
- So, we always pick the correct code word

Error Correction

- Hamming bound:
  - Assume we have *n* bits of which *m* are information bits
  - Assume that the code can correct *t* errors
  - Then all spheres around code words of radius *t* are distinct
  - Each sphere has $\sum_{i=0}^{t} \binom{n}{i}$ elements in them
  - There are 2^m spheres
  - And a total of 2^n elements
  - Therefore : $2^m \sum_{k=0}^{t} \binom{n}{k} \leq 2^n$

Error Correction

- Codes that attain the Hamming bound are called perfect
- 1973: All perfect codes that are not trivial (e.g. repetition codes) are:
  - Hamming Code
  - The Golay Code

Error Correction

- Systematic Linear Codes:
  - Generator matrix has an identity matrix in it. Example:
  - $$\mathbb{G} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad \mathbb{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$
  - Encodings: $000 \to 0000000$, $001 \to 0010111$, $010 \to 0101011$, $011 \to 0111100$, $100 \to 1001101$, $101 \to 1011010$, $110 \to 1100110$, $111 \to 1110001$

Error Correction

- Minimum distance:
  - $\min(\{\delta(\mathbb{G} \cdot \mathbf{u}, \mathbb{G} \cdot \mathbf{v}) \,|\, \mathbf{u}, \mathbf{v} \in \mathbb{Z}_2^3\})$
  - $= \min(\{w(\mathbb{G} \cdot \mathbf{u} + \mathbb{G} \cdot \mathbf{v}) \,|\, \mathbf{u}, \mathbf{v} \in \mathbb{Z}_2^3\})$
  - $= \min(\{w(\mathbb{G} \cdot (\mathbf{u} + \mathbf{v})) \,|\, \mathbf{u}, \mathbf{v} \in \mathbb{Z}_2^3\})$
  - $= \min(\{w(\mathbb{G} \cdot \mathbf{u}) \,|\, \mathbf{u} \in \mathbb{Z}_2^3\})$
  - $= 4$

Error Correction

- We can:
  - Correct one error
  - Discover two errors
  - Miscorrect three errors
  - Not discover some cases of four errors

Error Correction

- One Error Detecting Hamming Codes
  - Create a check matrix by using all non-zero vectors
    - But move the unit vectors first
    - $\mathbb{H} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$

Error Correction

$$\mathbb{G} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{pmatrix}$$

Error Correction

- If you do not want to calculate by hand, use Python
  - Install numpy using a Terminal or Command window:
    - `pip3 install numpy`
  - Import numpy
    - `import numpy as np`
  - Define matrix and vector as numpy arrays

Error Correction

```
G = np.array( [ [1,0,0,0,0,0,0,0,0,0,0],
                [0,1,0,0,0,0,0,0,0,0,0],
                [0,0,1,0,0,0,0,0,0,0,0],
                [0,0,0,1,0,0,0,0,0,0,0],
                [0,0,0,0,1,0,0,0,0,0,0],
                [0,0,0,0,0,1,0,0,0,0,0],
                [0,0,0,0,0,0,1,0,0,0,0],
                [0,0,0,0,0,0,0,1,0,0,0],
                [0,0,0,0,0,0,0,0,1,0,0],
                [0,0,0,0,0,0,0,0,0,1,0],
                [0,0,0,0,0,0,0,0,0,0,1],
                [0,0,0,0,1,1,1,1,1,1,1],
                [0,1,1,1,0,0,0,1,1,1,1],
                [1,0,1,1,0,1,1,0,0,1,1],
                [1,1,0,1,1,0,1,0,1,0,1]
                ])
```

Error Correction

- Use the new @ operator in Numpy/Python

```
vec = np.array([0,0,1,0,0,1,0,0,1,0,1])
encoded = G @ [0,0,1,0,0,1,0,0,1,0,1]%2
print('encoding is', vec)
```

Error Correction

- Example:
  - Let the message be (00100100101)
  - $\mathbb{G} \cdot \mathbf{x}^t = (0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0)^t$
  - $\mathbb{H} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$

Error Correction

- Assume we receive
  - $\mathbf{y}^t = (0 \quad 0 \quad 1 \quad 0 \quad \mathbf{1} \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0)^t$
  - We calculate the *syndrome:*
  - $\mathbb{H} \cdot \mathbf{y}^t = (1 \quad 0 \quad 0 \quad 1)^t$

$$\mathbb{H} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Error Correction

- Find the corresponding column in $\mathbb{H}$
  - This is the location of the error
  - Flip the corresponding bit and the error is corrected.

Extensions

- So far, we only looked at bit-signals
  - This is generally not true
- We only look at symmetric error probabilities
- We only looked at errors, but not erasures: