# THE LINK LAYER

COMP 30650: NETWORKS AND INTERNET SYSTEMS

Dr. Gavin McArdle
Email: gavin.mcardle@ucd.ie
Office: A1.09 Computer Science

# TODAY'S PLAN

## Error Correction

- Hamming Codes
- Comparison of Correction V Detection

## Retransmission of lost frames

# ERROR DETECTION

Noise can cause errors in the bits that are received. How do we detect this?

- Parity
- Checksums
- CRCs

Detection will let us fix the error, for example, by retransmission.

# WHY ERROR CORRECTION IS HARD

**If we had reliable check bits we could use them to narrow down the position of the error**

- Then correction would be easy

**What if the error is in the check bits as well as the data bits!**

- The data itself might even be correct!

# INTUITION FOR ERROR CORRECTING CODE

Suppose we construct a code with a Hamming distance of at least 3

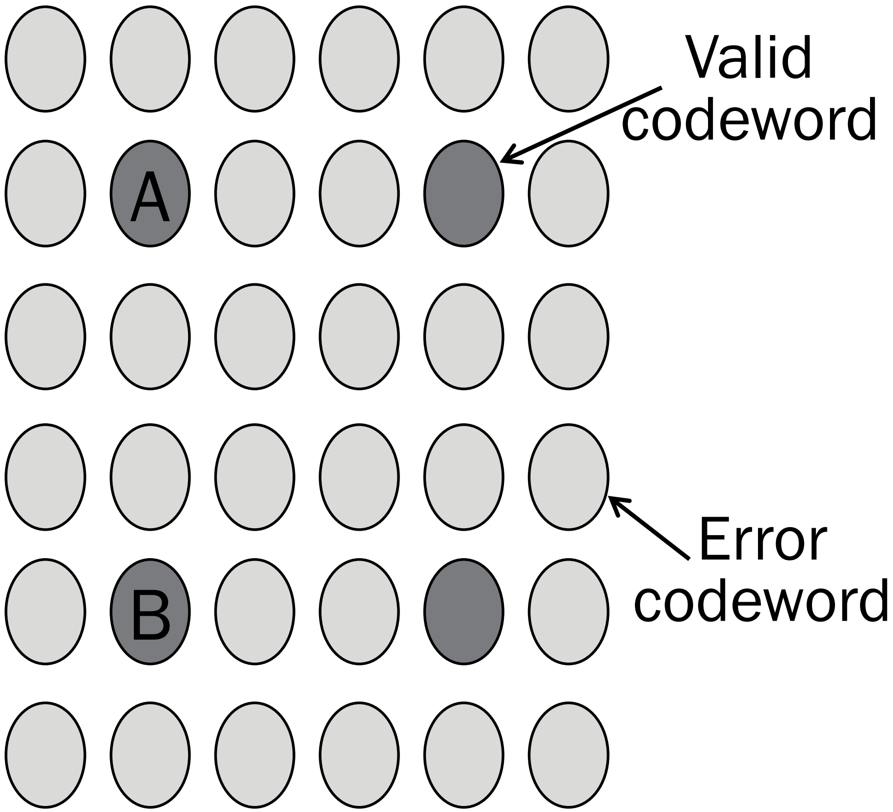
- Need ≥3 bit errors to change one valid codeword into another
- Single bit errors will be closest to a unique valid codeword

If we assume errors are only 1 bit, we can correct them by mapping an error to the closest valid codeword

- Works for d errors if Hamming Distance ≥ 2d + 1

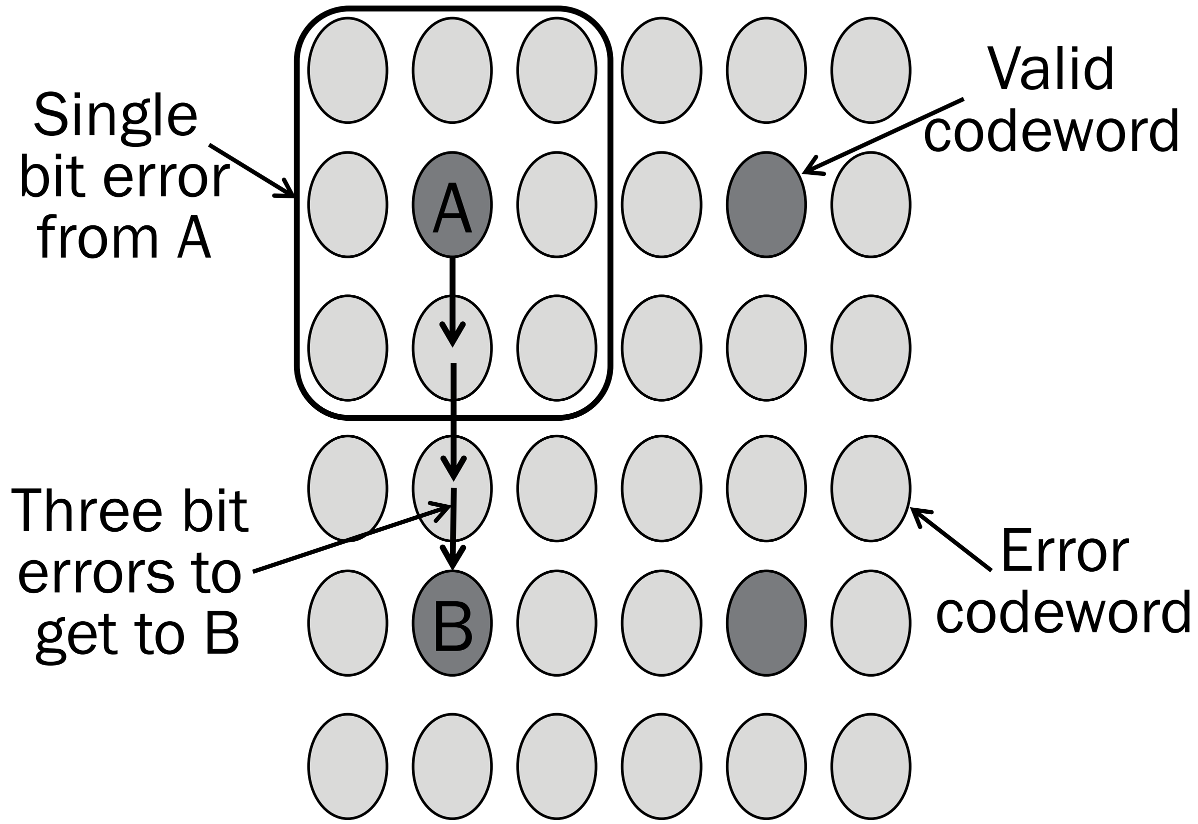
# INTUITION

Visualization of code:



# INTUITION

Visualization of code:



# HAMMING CODE

The example gives a method for constructing a code with a distance of 3

- Uses $n = 2^k - k - 1$, e.g., n=4, k=3
  - k = check bits
  - n = data
- Put check bits in positions p that are powers of 2, starting with position 1
- Check bit in position p is parity of positions with a p term in their values

Plus an easy way to correct

# HAMMING CODE

## Example: data=0101, 3 check bits

- 7 bit code, check bit positions 1, 2, 4
- Check 1 covers positions 1, 3, 5, 7
- Check 2 covers positions 2, 3, 6, 7
- Check 4 covers positions 4, 5, 6, 7

Parity:

0 for even
1 for odd

| _ | _ | _ | _ | _ | _ | _ |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# HAMMING CODE

## Example: data=0101, 3 check bits

- 7 bit code, check bit positions 1, 2, 4
- Check 1 covers positions 1, 3, 5, 7
- Check 2 covers positions 2, 3, 6, 7
- Check 4 covers positions 4, 5, 6, 7

| <u>0</u> | <u>1</u> | 0 | <u>0</u> | 1 | 0 | 1 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

$$p_1 = \underset{1}{*} + \underset{3}{0} + \underset{5}{1} + \underset{7}{1} = 0,$$

$$p_2 = \underset{2}{*} + \underset{3}{0} + \underset{6}{0} + \underset{7}{1} = 1,$$

$$p_4 = \underset{4}{*} + \underset{5}{1} + \underset{6}{0} + \underset{7}{1} = 0$$

# HAMMING CODE PATTERN

| Bit position | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Encoded data bits** | | p1 | p2 | d1 | p4 | d2 | d3 | d4 | p8 | d5 | d6 | d7 | d8 | d9 | d10 | d11 | p16 | d12 | d13 | d14 | d15 | |
| **Parity bit coverage** | p1 | ✘ | | ✘ | | ✘ | | ✘ | | ✘ | | ✘ | | ✘ | | ✘ | | ✘ | | ✘ | | |
| | p2 | | ✘ | ✘ | | | ✘ | ✘ | | | ✘ | ✘ | | | ✘ | ✘ | | | ✘ | ✘ | | … |
| | p4 | | | | ✘ | ✘ | ✘ | ✘ | | | | | ✘ | ✘ | ✘ | ✘ | | | | | ✘ | |
| | p8 | | | | | | | | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ | | | | | | |
| | p16 | | | | | | | | | | | | | | | | ✘ | ✘ | ✘ | ✘ | ✘ | |

# HAMMING CODE

## To decode:

- Recompute check bits (with parity sum including the check bit)
- Arrange as a binary number
- Value (syndrome) tells error position
- Value of zero means no error
- Otherwise, flip the bit to correct

# HAMMING CODE

## Decoding Example, continued

$$\longrightarrow \quad \underline{0} \quad \underline{1} \quad 0 \quad \underline{0} \quad 1 \quad 0 \quad 1$$

1 2 3 4 5 6 7

Check 1 covers positions 1, 3, 5, 7
Check 2 covers positions 2, 3, 6, 7
Check 4 covers positions 4, 5, 6, 7

$p_1$=
$p_2$=
$p_4$=

Syndrome =

Data =

# HAMMING CODE

## Decoding Example, continued

$$\longrightarrow \underline{0}\ \underline{1}\ 0\ \underline{0}\ 1\ 0\ 1$$

1 2 3 4 5 6 7

$p_1 = 0+0+1+1 = 0,$
$p_2 = 1+0+0+1 = 0,$
$p_4 = 0+1+0+1 = 0$

Syndrome = 000, no error
Data = 0 1 0 1

# HAMMING CODE

## Example, continued

$\longrightarrow$ $\underline{0}$ $\underline{1}$ 0 $\underline{0}$ 1 1 1

1 2 3 4 5 6
7

Check 1 covers positions 1, 3, 5, 7
Check 2 covers positions 2, 3, 6, 7
Check 4 covers positions 4, 5, 6, 7

$p_1$=
$p_2$=
$p_4$=

Syndrome =
Data =

# HAMMING CODE

## Example, continued

$\longrightarrow$ $\underline{0}$ $\underline{1}$ 0 $\underline{0}$ 1 1 1

1 2 3 4 5 6 7

Check 1 covers positions 1, 3, 5, 7
Check 2 covers positions 2, 3, 6, 7
Check 4 covers positions 4, 5, 6, 7

$p_1 = 0+0+1+1 = 0$,
$p_2 = 1+0+1+1 = 1$,
$p_4 = 0+1+1+1 = 1$

Syndrome = 1 1 0, flip position 6
Data = 0 1 0 1 (correct after flip!)

# DETECTION VS. CORRECTION

Which is the better choice will depend on the pattern of errors. For example:

- 1000 bit messages with a bit error rate (BER) of 1 in 10,000

Which has less overhead?

# DETECTION VS. CORRECTION

## Which is better will depend on the pattern of errors. For example:

- 1000 bit messages with a bit error rate (BER) of 1 in 10000

## Which has less overhead?

- It still depends! We need to know more about the errors

# DETECTION VS. CORRECTION

1. **Assume bit errors are random**

- Messages have 0 or maybe 1 error
- Most will have 0

**Error correction:**

- Need ~10 check bits per message
- Overhead: **10**

**Error detection:**

- Need ~1 check bits per message plus 1000 bit retransmission 1/10 of the time
- Overhead: 1 + 1000/10 = **101**

# DETECTION VS. CORRECTION

2. **Assume errors come in bursts of 100**

- Only 1 or 2 messages in 1000 have errors

**Error correction:**

- Need >>100 check bits per message
- Overhead: >**100?**

**Error detection:**

- Need 32? check bits per message plus 1000 bit resend 2/1000 of the time
- Overhead:32 + (2/1000 *1000) = **34**

# DETECTION VS. CORRECTION

**Error correction:**

- Needed when errors are expected
- Or when no time for retransmission

**Error detection:**

- More efficient when errors are not expected
- And when errors are large when they do occur