Computer Networks

Error Correction (§3.2.1)

David Wetherall (djw@uw.edu)
Professor of Computer Science & Engineering
UNIVERSITY of WASHINGTON

Topic

- Some bits may be received in error due to noise. How do we fix them?
  - Hamming code »
  - Other codes »

- And why should we use detection when we can use correction?

Why Error Correction is Hard

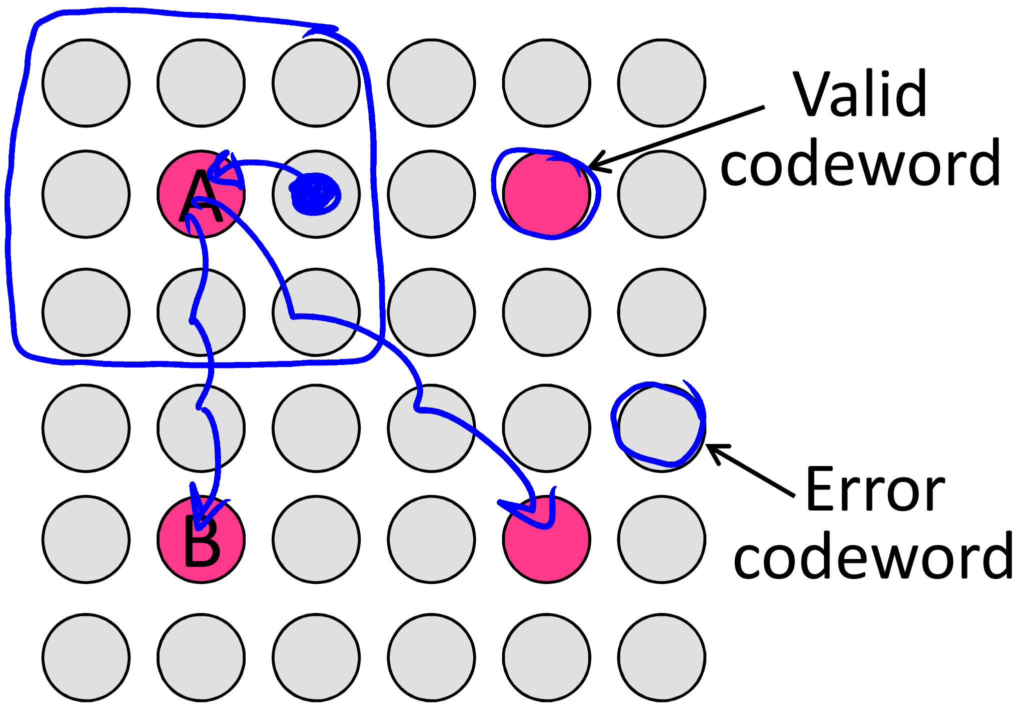
- If we had reliable check bits we could use them to narrow down the position of the error
  - Then correction would be easy
- But error could be in the check bits as well as the data bits!
  - Data might even be correct

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 HD ≥ 2d + 1

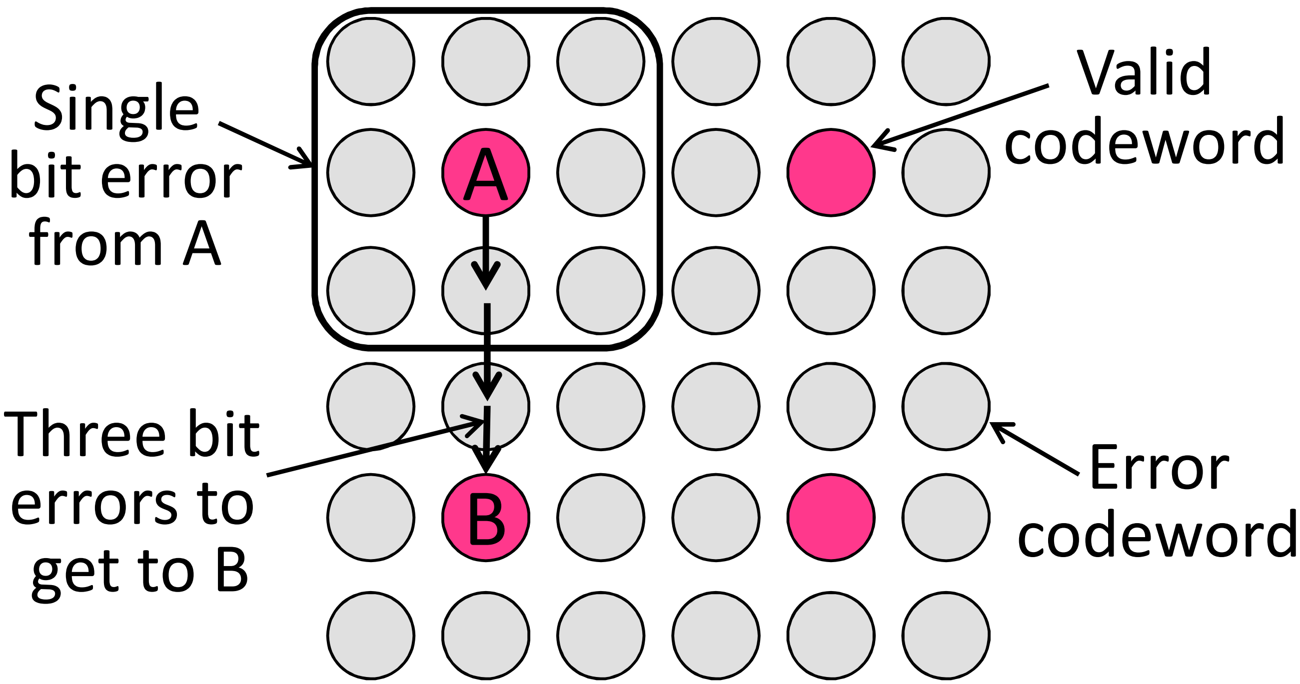
Intuition (2)

- Visualization of code:



Intuition (3)

- Visualization of code:



Hamming Code

- Gives a method for constructing a code with a distance of 3
  - Uses $n = 2^k - k - 1$, e.g., n=4, k=3
  - 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 [soon]

Hamming Code (2)

- 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

$p_1 = 0+1+1 = 0$
$p_2 = 0+0+1 = 1$
$p_3 = 1+0+1 = 0$

0	1	0	0	1	0	1
1	2	3	4	5	6	7

Hamming Code (3)

- 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

$$\begin{array}{ccccccc} \underline{0} & \underline{1} & 0 & \underline{0} & 1 & 0 & 1 \end{array} \longrightarrow$$
$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}$$

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

Hamming Code (4)

- 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 bit to correct

Hamming Code (5)

- 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 = 0101

Hamming Code (6)

- Example, continued

$$\longrightarrow \underset{1}{\underline{0}}\ \ \underset{2}{\underline{1}}\ \ \underset{3}{0}\ \ \underset{4}{\underline{0}}\ \ \underset{5}{1}\ \ \underset{6}{0}\ \ \underset{7}{1}$$

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 (7)

- Example, continued

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

1 2 3 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 = 110 → 6

Data = 0101

Hamming Code (8)

- Example, continued

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

1 2 3 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!)

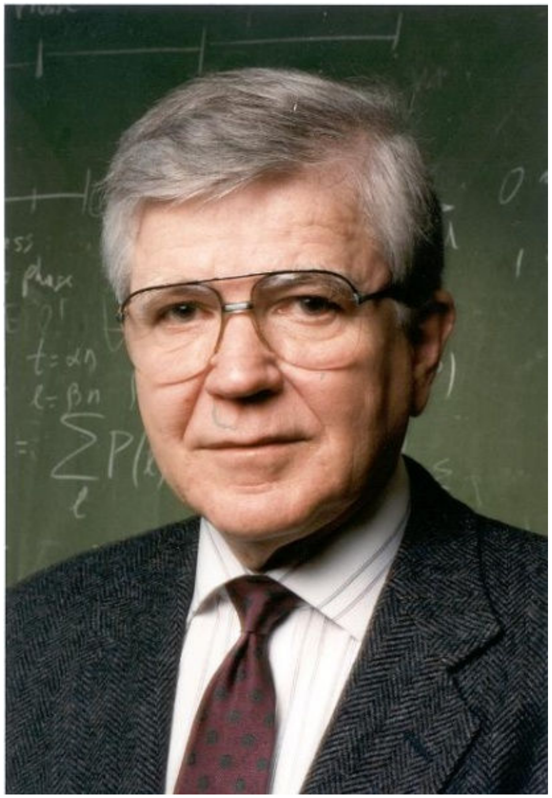
Other Error Correction Codes

- Codes used in practice are much more involved than Hamming

- Convolutional codes (§3.2.3)
  - Take a stream of data and output a mix of the recent input bits
  - Makes each output bit less fragile
  - Decode using Viterbi algorithm (which can use bit confidence values)

Other Codes (2) – LDPC

- Low Density Parity Check (§3.2.3)
  - LDPC based on sparse matrices
  - Decoded iteratively using a belief propagation algorithm
  - State of the art today
- Invented by Robert Gallager in 1963 as part of his PhD thesis
  - Promptly forgotten until 1996 …



Source: IEEE GHN, © 2009 IEEE

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?

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 (2)

1. Assume bit errors are random
   – Messages have 0 or maybe 1 error

- 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: $1 + \frac{1000}{10} \sim 101$ bits

Detection vs. Correction (3)

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: $32 + \frac{1000}{1000} \times 2 \sim 34$ bits

Detection vs. Correction (4)

- 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

Error Correction in Practice

- Heavily used in physical layer
  - LDPC is the future, used for demanding links like 802.11, DVB, WiMAX, LTE, power-line, …
  - Convolutional codes widely used in practice
- Error detection (w/ retransmission) is used in the link layer and above for residual errors
- Correction also used in the application layer
  - Called Forward Error Correction (FEC)
  - Normally with an erasure error model
  - E.g., Reed-Solomon (CDs, DVDs, etc.)

END