Computer Networks (06-05933) The Data-Link Layer

Rob Minson, rm. 134
R.Minson@cs.bham.ac.uk

Error Control
Correction and Detection Schemes

Error Control

- Split binary data stream in to *words*
  - Each word is m bits long
  - Add r extra bits in special, known positions
  - The r bits can be examined by the receiver to
    - Detect if bits have been incorrectly flipped
    - Locate and correct the flipped bits
  - The r bits are then removed and the m-bit data word is delivered

Error Detection with Parity Bits

- A Parity Bit makes the number of 1's in a word even, e.g. if $m = 3$ and $r = 1$

000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

Error Detection with Parity Bits

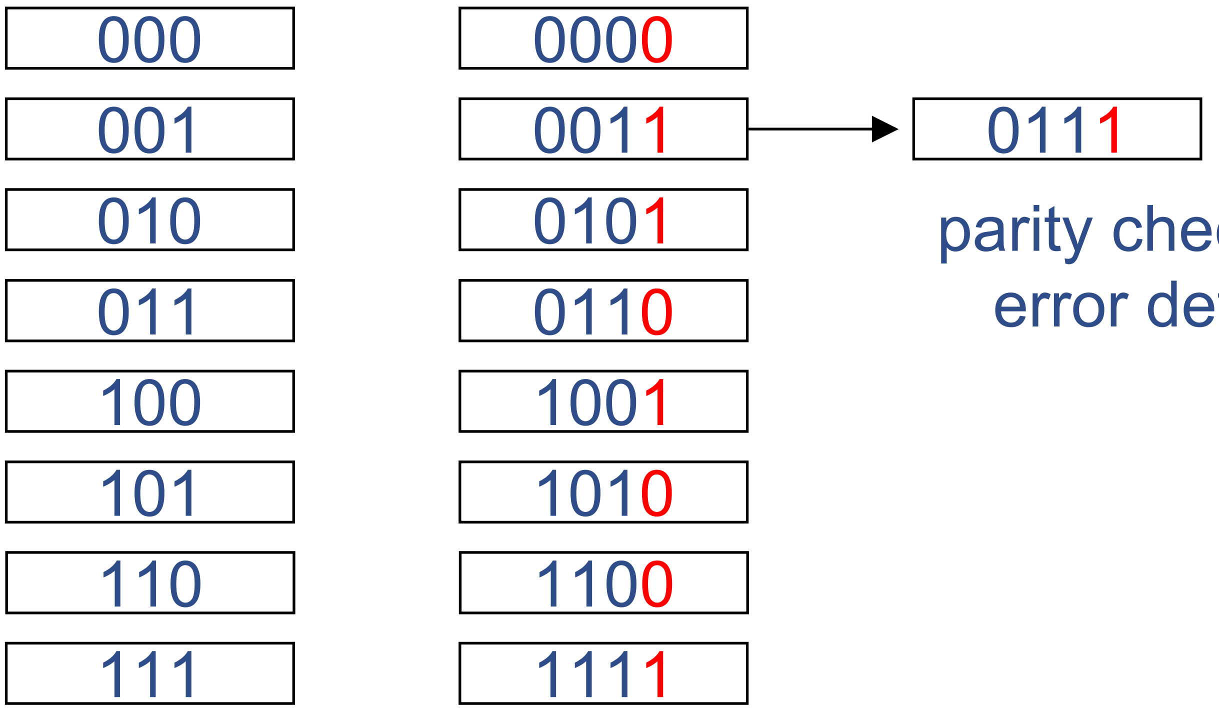
- A Parity Bit makes the number of 1's in a word even, e.g. if *m* = 3 and *r* = 1

000	0000	
001	0011	→ 0011
010	0101	
011	0110	
100	1001	
101	1010	
110	1100	
111	1111	

parity check OK

Error Detection with Parity Bits

- A Parity Bit makes the number of 1's in a word even, e.g. if *m* = 3 and *r* = 1



parity check not OK
error detected!

Error Detection with Parity Bits

- A Parity Bit makes the number of 1’s in a word even, e.g. if *m* = 3 and *r* = 1

000	0000	
001	0011 →	1111
010	0101	
011	0110	
100	1001	
101	1010	
110	1100	
111	1111	

parity check OK
error missed!!!

Error Detection with Parity Bits

- Parity Bits detect single-bit errors in a given data word
  - *m* can be arbitrarily large
  - But if 2 bits are flipped instead of one the parity check will pass and an error will be missed
  - Larger *m* increases likelihood of this

Hamming Theory

- For *any* values of *m* and *r* we can say some things about a given hypothetical code
  - A scheme produces 2^{m+r} receivable words
  - 2^m will be valid words
  - $2^{m+r} - 2^m$ will be invalid words
- An error detection scheme can tell the difference between a valid and an invalid word
- An error *correction* scheme can work out which of the 2^m valid words an invalid word must have been upon transmission.

Hamming Theory

- Some number of bit-flips = *d* will change a valid word in to another valid word
  - *d* is the *hamming distance* of the code
  - e.g. Our 4-bit parity scheme required just two flips to go from one valid word to another, $d = 2$
  - An error *detection* scheme to detect $\leq d$ errors must have a hamming distance = $d+1$
  - Why…?

Hamming Theory

- An error *detection* scheme to detect ≤ *d* errors must have a hamming distance = *d+1*

0000	with distance 2, if 1 error occurs we will always receive an invalid word	0001
0011		0010
0101		0100
0110		0111
1001		1000
1010		1011
1100		1101
1111		1110

Hamming Theory

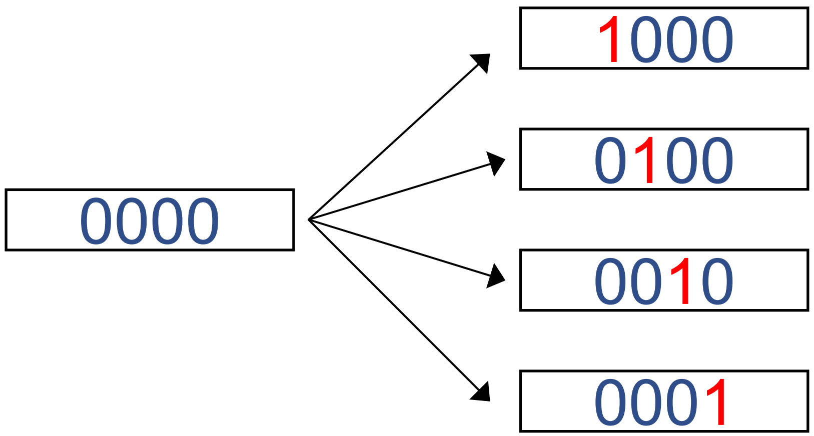
- To *correct* single errors, an *m*-bit scheme must have *r* bits such that:

$$(m + r + 1)2^{m} \leq 2^{m+r}$$

- Why?

Hamming Theory

- Explanation 1 (Reserve Invalid Words):
  - We have 2^m valid words (see our 4-bit parity code)
  - For each one, take each bit in turn and flip it
  - This generates *m+r* invalid words



Hamming Theory

- Explanation 1 (Reserve Invalid Words):
  - In total there are 2^{m+r} - 2^{m} invalid words
  - Each has distance = 1 from a valid original
  - The receiver will always be able to correct to the valid original if it receives one of these words, because no other valid word is within distance = 1
  - (This is the fundamental principle of error correcting codes)

Hamming Theory

- Explanation 1 (Reserve Invalid Words):
  - Because we need to keep this property, each valid word ‘reserves’ *m+r invalid* words
  - The total number of possible words is 2^{m+r}
  - Therefore:

$$2^m + (m+r)2^m \leq 2^{m+r}$$

all valid words + the words invalid words reserved by them ≤ total words available

$$2m + (m+r)2m = (m+r+1)2m$$

Hamming Theory

- Explanation 2 (Encode Enough Positions)
  - Start by rearranging the equation a bit:

    $(m+r+1)2^m \leq 2^{m+r}$

    $(m+r+1) \leq 2^r$ (divide both sides by 2^m)
  - 2^r is the number of distinct words we can make out of the *r* parity bits.
  - In order to *correct* errors, the parity word must tell us both
    - if an error has occurred
    - if one has, which bit position is wrong

Hamming Theory

- Explanation 2 (Encode Enough Positions)
  - To do this, we need to have 'parity words' for
    - No error,
    - Error in position 1
    - Error in position 2
    - …
    - Error in position *m+r*
  - Therefore 2^r must be able to express this many positions, and so

$$(m+r) + 1 \leq 2^r$$

(That is, we must have enough parity 'words' to express 'error' in each bit position + 1 word for 'no errors')

Hamming Codes

- Optimal error correction for single-bit errors
  - Use multiple parity bits ($r > 1$)
  - Each parity bit checks a subset of the data bits
  - Cross check which parities are wrong to determine which data bit has been flipped
- Central idea:
  - The set of parity bits *describe* the position of the incorrect bit

Hamming Codes

- A Simple Code
  - $m = 4$
  - $r = 3$
    - r^1 checks m1, m2, m3
    - r^2 checks m2, m3, m4
    - r^3 checks m1, m2, m4

0	0	0	0	0	0	0
r^1	m^1	m^2	r^2	m^3	m^4	r^3

Hamming Codes

- An Example:
  - For the given data bits ‘0011’
  - What should our 3 parity bits be?

?	0	0	?	1	1	?
r^1	m^1	m^2	r^2	m^3	m^4	r^3

$r^1 \longrightarrow m^1\ m^2\ m^3$

$r^2 \longrightarrow m^2\ m^3\ m^4$

$r^3 \longrightarrow m^1\ m^2\ m^4$

Hamming Codes

- Another Example:
  - For the given data bits ‘1010’
  - What should our 3 parity bits be?

?	1	0	?	1	0	?
r^1	m^1	m^2	r^2	m^3	m^4	r^3

$r^1 \longrightarrow m^1\ m^2\ m^3$

$r^2 \longrightarrow m^2\ m^3\ m^4$

$r^3 \longrightarrow m^1\ m^2\ m^4$

Hamming Codes

- An Example:
  - Flip **any one** data bit and ***at least two*** parity bits will be incorrect
  - Each data bit is checked by a different set of parity bits, so we always know which one was flipped

0	1	0	1	1	0	1
r^1	m^1	m^2	r^2	m^3	m^4	r^3

$r^1 \longrightarrow m^1\ m^2\ m^3$

$r^2 \longrightarrow m^2\ m^3\ m^4$

$r^3 \longrightarrow m^1\ m^2\ m^4$

Hamming Codes

- An Example:
  - The pattern of correct/incorrect in the parity bits form *parity words*
  - Each unique parity word describes the position of the error in the data word

r^1	r^2	r^3	error bit
✗	✓	✗	m^1
✗	✗	✗	m^2
✗	✗	✓	m^3
✓	✗	✗	m^4

$r^1 \longrightarrow m^1\ m^2\ m^3$

$r^2 \longrightarrow m^2\ m^3\ m^4$

$r^3 \longrightarrow m^1\ m^2\ m^4$

Hamming Codes

- But isn't there a problem here…?
  - r^1 checks m^1, m^2, m^3
  - r^2 checks m^2, m^3, m^4
  - r^3 checks m^1, m^2, m^4

0	0	0	0	0	0	0
r^1	m^1	m^2	r^2	m^3	m^4	r^3

Hamming Codes

- But isn't there a problem here…?
  - r^1 checks m^1, m^2, m^3
  - r^2 checks m^2, m^3, m^4
  - r^3 checks m^1, m^2, m^4…
  - …so who checks $r^{1,2,3}$?

0	0	0	0	0	0	0
r^1	m^1	m^2	r^2	m^3	m^4	r^3

Hamming Codes w/ Logarithmic Parity

- In a real Hamming code, the *r* bits are in positions $2^0, 2^1, 2^2, \ldots, 2^i$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r^1	r^2	m^1	r^3	m^2	m^3	m^4	r^4	m^5	m^6	m^7	m^8	m^9	m^{10}	m^{11}	r^5	m^{12}

- They check the data bits, but they also check *each other*…

Hamming Codes w/ Logarithmic Parity

- Each bit has its position expressed as a sum of powers of 2
  - e.g. m^4 in position 7 = 4 + 2 + 1 = $2^2 + 2^1 + 2^0$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r^1	r^2	m^1	r^3	m^2	m^3	m^4	r^4	m^5	m^6	m^7	m^8	m^9	m^{10}	m^{11}	r^5	m^{12}

  - A position is checked by the parity bits in the positions used to calculate its sum

Hamming Codes w/ Logarithmic Parity

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0
r^1	r^2	m^1	r^3	m^2	m^3	m^4	r^4	m^5	m^6	m^7	m^8

	1	2	3	4	5	6	7	8	9	10	11	12
r^1 (1)	✓		✓		✓		✓		✓		✓	
r^2 (2)		✓	✓			✓	✓			✓	✓	
r^3 (4)				✓	✓	✓	✓					✓
r^4 (8)								✓	✓	✓	✓	✓

Hamming Codes w/ Logarithmic Parity

1	2	3	4	5	6	7	8	9	10	11	12
?	?	0	?	0	1	0	?	1	1	0	0
r^1	r^2	m^1	r^3	m^2	m^3	m^4	r^4	m^5	m^6	m^7	m^8

	1	2	3	4	5	6	7	8	9	10	11	12
r^1 (1)	✓		✓		✓		✓		✓		✓	
r^2 (2)		✓	✓			✓	✓			✓	✓	
r^3 (4)				✓	✓	✓	✓					✓
r^4 (8)								✓	✓	✓	✓	✓

Hamming Codes w/ Logarithmic Parity

1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	1	0	1	0	0	1	1	0	0
r^1	r^2	m^1	r^3	m^2	m^3	m^4	r^4	m^5	m^6	m^7	m^8

	1	2	3	4	5	6	7	8	9	10	11	12
r^1 (1)	✓		✓		✓		✓		✓		✓	
r^2 (2)		✓	✓			✓	✓			✓	✓	
r^3 (4)				✓	✓	✓	✓					✓
r^4 (8)								✓	✓	✓	✓	✓

Hamming Codes w/ Logarithmic Parity

- error in m^2…

1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	1	***1***	1	0	0	1	1	0	0
r^1	r^2	m^1	r^3	m^2	m^3	m^4	r^4	m^5	m^6	m^7	m^8

	1	2	3	4	5	6	7	8	9	10	11	12
r^1 (1)	✓		✓		✓		✓		✓		✓	
r^2 (2)		✓	✓			✓	✓			✓	✓	
r^3 (4)				✓	✓	✓	✓					✓
r^4 (8)								✓	✓	✓	✓	✓

- …parity incorrect for r^1 and r^3

Hamming Codes w/ Logarithmic Parity

- error in m^6…

1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	1	0	1	0	0	1	***0***	0	0
r^1	r^2	m^1	r^3	m^2	m^3	m^4	r^4	m^5	m^6	m^7	m^8

	1	2	3	4	5	6	7	8	9	10	11	12
r^1 (1)	✓		✓		✓		✓		✓		✓	
r^2 (2)		✓	✓			✓	✓			✓	✓	
r^3 (4)				✓	✓	✓	✓					✓
r^4 (8)								✓	✓	✓	✓	✓

- …parity incorrect for r^2 and r^4

Hamming Codes w/ Logarithmic Parity

- error in r^4…

1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	1	0	1	0	*1*	1	1	0	0
r^1	r^2	m^1	r^3	m^2	m^3	m^4	r^4	m^5	m^6	m^7	m^8

	1	2	3	4	5	6	7	8	9	10	11	12
r^1 (1)	✓		✓		✓		✓		✓		✓	
r^2 (2)		✓	✓			✓	✓			✓	✓	
r^3 (4)				✓	✓	✓	✓					✓
r^4 (8)								✓	✓	✓	✓	✓

- …parity incorrect for r^4

Hamming codes w/ Logarithmic Parity

- General case…
  - Iterate over each parity bit…
  - If the bit is not in correct parity, add the value of its position to a counter…
  - At the end…
    - …if the counter == 0 there are no errors
    - …else, the counter’s value indicates the incorrect position
    - Why?
      - Because it is a sum of the incorrect parity positions
      - NOTE! Each position is described by a unique sum

Hamming codes w/ Logarithmic Parity

- Example:

1	2	3	4	5	6	7
0	1	0	1	0	1	0
r^1	r^2	m^1	r^3	m^2	m^3	m^4

1 = 1
2 = 2
3 = 1 + 2
4 = 4

5 = 4 + 1
6 = 4 + 2
7 = 4 + 2 + 1

  - Iterate over each parity bit…
  - If the bit is not in correct parity, add the value of its position to a counter…
  - At the end…
    - …if the counter == 0 there are no errors
    - …else, the counter's value indicates the incorrect position

Hamming codes w/ Logarithmic Parity

- Example:

1	2	3	4	5	6	7
0	0	0	1	0	0	1
r^1	r^2	m^1	r^3	m^2	m^3	m^4

1 = 1
2 = 2
3 = 1 + 2
4 = 4

5 = 4 + 1
6 = 4 + 2
7 = 4 + 2 + 1

- Iterate over each parity bit…
- If the bit is not in correct parity, add the value of its position to a counter…
- At the end…
  - …if the counter == 0 there are no errors
  - …else, the counter's value indicates the incorrect position

Hamming Code Summary

- Hamming codes correct single bit errors in a given data word
  - Embed *r* parity bits at positions 2^0, 2^1, 2^2, …
  - Each parity bit checks a unique subset of the other bit positions (and itself)
  - If a single bit error occurs a unique combination of the parity bits will be incorrect
  - This unique combination is used to locate and correct the flipped bit