COMP2011 -- **Networks**, Databases and Graphics

Dealing Errors at Bit Level

Dr. Shi Zhou
Department of Computer Science
University College London

Errors

- Errors are inevitable.
  - Delivery of a message which has been corrupted in some way, i.e. the bits received are not the same as the bits sent.
  - Non-delivery of a message that was submitted.
- Their occurrence can only be reduced not eliminated.

Errors and error rates

- Bit Error Rate (BER)
  - e.g. 10^{-12} is one error in 10^{12} bits (on average)
    - For 64Kbps, a BER of 10^{-12} gives one bit error every 6 months - negligible
- **Signalled error rate (SER)**:
  - detected errors
- **Residual error rate (RER)**:
  - errors that are not detected
- Error correction:
  - if either SER or RER is unacceptable
- Error detection:
  - if RER is unacceptable

Redundancy

- Here is a **srelling** mistake
- Not all combinations of letters make legal words:
  - if they did how would we detect error?
- Redundancy:
  - information that is not essential
  - helps in detecting errors
- To detect/correct errors, we need redundancy:
  - how much do we need?
  - what is the cost?

Error detection -- Hamming distance

- **Codewords**: size $c = m + r$
  - m is size of message
  - r is number of error control bits
- Example: 8-bit ASCII
  - 7-bits data, 1-bit even parity
  - 12.5% redundancy
- **Hamming distance**
  - Prof. Richard Hamming (1915-1998)
  - The number of bits in which two codewords differ
- To detect h-bit errors: (minimum) **Hamming distance > *h***

even parity bit

```
A   0 100 0001
B   0 100 0010
C   1 100 0011
```

Min. Hamming distance = 2

```
1-bit error:  0100 0101  ×
              0000 0010  ×
              1000 0011  ×

2-bit errors: 1100 0011  ?
              0110 0011  ?
              0100 0001  ?
```

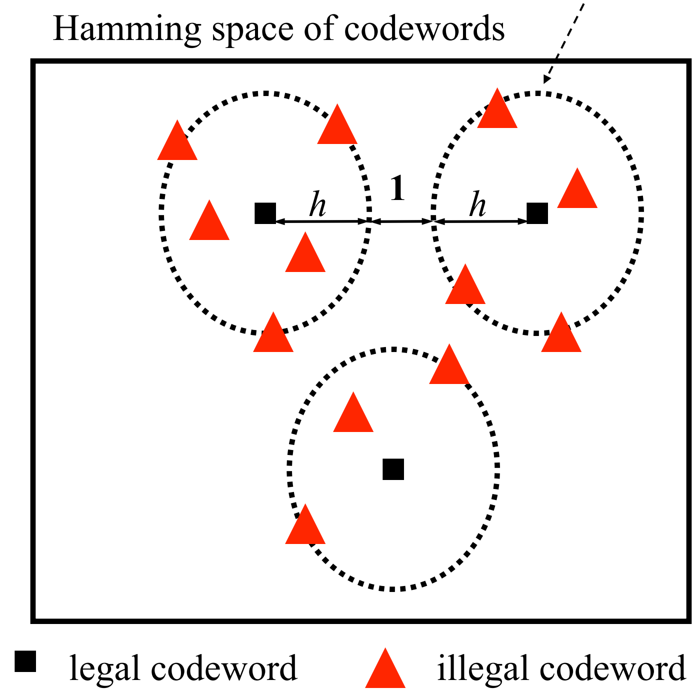
Error detection
-- Residual error rate

- Example: Transmission of 8-bit bytes
  - Raw BER of channel = p = 10^{-4}
  - P(a bit is in error) = p; P(a bit is not in error) = $1-p$
- No parity (eight bits)
  - P(no bit errors in a byte) = $(1-p)^8$
  - P(at least one bit error in byte) = $1 - (1-p)^8 = 8\text{x}10^{-4}$
- Single-bit parity (nine bits in all)
  - P(exactly two bits in error) = ${}^9C_2 p^2(1-p)^7 = 3.6\times10^{-7}$
    - ${}^9C_2$ is number of ways 2 bits can be chosen
    - $p^2(1-p)^7$ is probability of any of these occurring
- Single-bit parity reduces the undetected error rate by three orders of magnitude!

Error correction -- Hamming distance

- Can we correct h-bit errors?
- Yes, if
  - enough redundancy and
  - enough Hamming distance between codewords
    - what is “enough”?
- To correct h-bit errors:
  Hamming distance ≥ 2h + 1

Circles encompasses all codewords derivable from legal codeword by changing up to h-bits



2-bit error

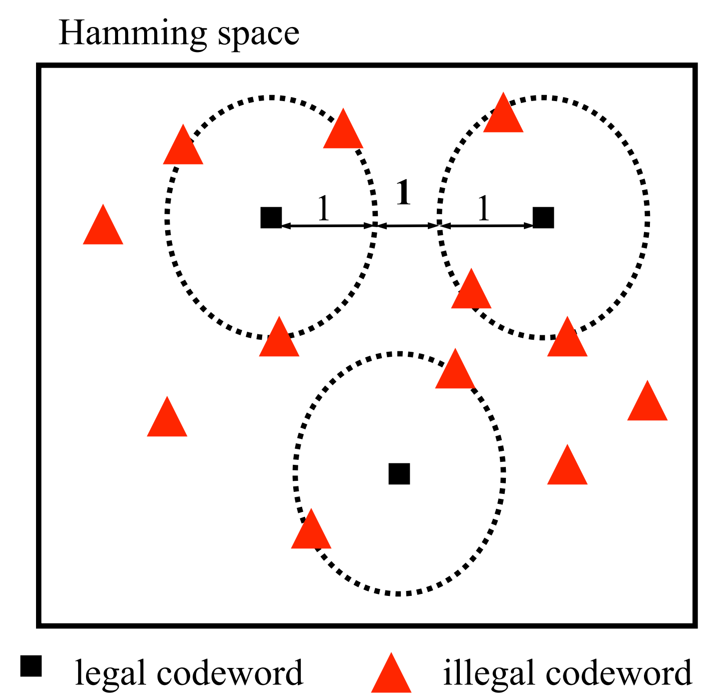
3-bit error

```
00000 00000      00000 10001
00000 11111                    00000 10001
11111 00000
11111 11111
```

Example: code words with min Hamming distance of 5

Error correction -- Redundancy

- $c = m + r$
  - Absolute total number of codewords is 2^c
- Suppose we want to correct all 1-bit errors
  - There are 2^m circles
  - Each circle has $1+c$ codewords
  - Total number of codewords in the circles: $(1+c)2^m$
- We should have $(1+c)2^m \leq 2^c$
  - $1+m+r \leq 2^r$, e.g. $m = 7$, $r = 4$
  - 36% redundancy



Error correction -- Hamming code

- Detection and correction of 1-bit errors
- 7 databits + 4 check bits

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

A = 1 0 0 0 0 0 1

11	10	9	**8**	7	6	5	**4**	3	**2**	**1**	
1	0	0	**1**	0	0	0	**0**	1	**0**	**0**	**Tx**
1	0	0	**1**								
				0	0	0	**0**				
1	0			0	0			1	**0**		
1		0		0		0		1		**0**	

Msg bit	Check bit 08	04	02	01
11	1	0	1	1
10	1	0	1	0
09	1	0	0	1
07	0	1	1	1
06	0	1	1	0
05	0	1	0	1
03	0	0	1	1

A check bit is set by an parity calculation on the msg bits identified by 1s in the column under the check bit, e.g.

bit 02 = parity(11, 10, 07, 06, 03)

i.e parity(02, 11, 10, 07, 06, 03) = 0

Assume bit 6 changed during transmission

11	10	9	8	7	6	5	4	3	2	1	
1	0	0	1	0	**1**	0	0	1	0	0	**Rx**

Check by repeating parity calculation

08	04	02	01	
0	1	1	0	=> bit 06

Note

- Hamming code is designed to detect 1 or 2-bit errors in codewords, and correct 1-bit errors.
  – All right for small data items.
- Modern communication networks, copper and (optical) fibre, have low error rates.
  – When errors occur, they are usually multiple bit errors in a small region of data
  – Error correction is not feasible, only error detection.
- Need some simple and fast test to detect the small number of blocks that have errors
  – Then deal with them by getting retransmission from Tx (not by correction at Rx)

Error detection in modern communication networks

- Add check bits at end of data block
  - They are easily calculated at both Tx and Rx
  - They tell Rx about errors in data block of even 10000s of bits in length.
- Check bits are produced by a division process
  - e.g.12345678912345/54321 ==> remainder 42312
  - Errors in the transmission of the message will give a different remainder
- Need a division process that can be done quickly in hardware!

Error detection based on CRC

- Cyclic redundancy check (CRC), also known as **Polynomial codes**
  - Based on solid math theory, cheap to implement in hardware
- General form of a polynomial: $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$
  - Degree n is the power of its highest term
  - When x=10, decimal number; when x=2, binary number

```
       24
    ______
123|3071
    246
     611
     492
     119
```

so: 3071/123 = 24 r 119
or: 3071 − 119 = (24×123)

```
                                    3.10 - 6
                  ____________________________
1.10²+2.10+3 |3.10³   +             7.10 + 1
              3.10³ + 6.10² + 9.10
              ____________________
                    - 6.10² - 2.10 + 1
                    - 6.10² -12.10 -18
                    ______________________
                             10.10 + 19
```

```
          8x⁴ - 5x² +       6
         ____________________
x² +1 | 8x⁶ + 3x⁴ +   x² + 1
        8x⁶ + 8x⁴
        _________
             -5x⁴ +   x²
             -5x⁴ + -5x²
             ___________
                     6x² + 1
                     6x² + 6
                     _______
                          -5
```

so: $8x^6+3x^4+x^2+1 \;/\; x^2+1 = 8x^4-5x^2+6$, r -5
or: $8x^6+3x^4+x^2+1 - (-5) = (8x^4-5x^2+6)(x^2+1)$

Polynomial codes [1]

- Use remainder (R) as check bits
- Sender “subtracts” remainder from message (M)
- Receiver re-computes remainder:
  - If non-zero ⇒ error

- Must chose “good” divisor, the **generator polynomial** (G)
- R always has fewer bits than G
- Based on mod 2 arithmetic - simple in hardware

+	1	0
1	0	1
0	1	0

-	1	0
1	0	1
0	1	0

```
          1100100
1101|1011010110   ⇒ 1100100 r 10
     1101
      1100
      1101
         1101
         1101
             10
```

Polynomial codes [2] - example

M 11011100 **message**, 8 bits, degree 7

G 1100 **generator polynomial**, 4 bits, degree 3, $x^3 + x^2$

M' 11011100000 **extended message** by max size of remainder, i.e. 8 bits message + 3 bits remainder

M_t 11011100100 **transmitted message**: remainder subtracted from extended message M'

Sender: divide *M'* by *G*:

```
         10010111
1100|11011100000
     1100
        1110
        1100
          1000
          1100
           1000
           1100
            1000
            1100
             100
```

R **100**, degree 2, 3 bits

```
M'   11011100000
R            100  –
Mt   11011100100
```

Receiver: divide M_t by *G*:

```
         10010111
1100|11011100100
     1100
        1110
        1100
          1001
          1100
           1010
           1100
            1100
            1100
               0
```

Polynomial codes [3]

- Polynomial codes are able to detect burst errors.
- Burst errors is equivalent to adding (mod 2) some random number to the transmitted message.

4-bit message 1101
Received message 1001, Error is 0100 → $E = x^2$
Received message 1111, Error is 0010 → $E = x^1$

M_t transmitted message, divisible by G
M_E received message including error E

$$M_E = M_t + E$$

$$\frac{M_E}{G} = \frac{M_t + E}{G} = \frac{M_t}{G} + \frac{E}{G}$$

E/G must give a non-zero remainder if E is non-zero.

For an 1-bit error, i.e. $E = x^n$

What form of G will detect all 1-bit errors?

Suppose G ends in a 1, then

$$\frac{E}{G} = \frac{x^n}{x^k + \ldots + 1}$$

will always have a remainder.
All 1-bit errors will be detected.
Rule 1: G should not have trailing zeros.

Polynomial codes [4]

Rule 2: if G has (x+1) as a factor it will detect all errors affecting an odd number of bits.

1. Assume E(x) has odd number of non-zero coefficients. Using mod 2 arithmetic, we have E(1) = 1.

2. If E(x) have (x + 1) as a factor, then we can write E(x) = (x + 1)F(x), so E(1) = (1+1)F(1)=0.

1 and 2 are contradictory!
Therefore we do not get a zero remainder when E has an odd number of bit errors and G has a factor (x + 1).

So, $G(x) = (x + 1)(x^k + \ldots + 1)$ will detect:

- all single bit errors
- all odd-number bit errors

Lots of other properties possible …
Two standard polynomials:

CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$x^{16} + x^{12} + x^5 + 1$

Both give 16-bit checksums which will detect:

- all 1 and 2 bit errors
- all error bursts of up to 16 bits in length
- all bursts affecting an odd number of bits
- 99.997% of 17 bit error bursts
- 99.998% of 18 and longer bursts

Notes

- When errors are detected at Rx, Rx can arrange for Tx to re-transmit the message
  - Using automatic repeat request (ARQ) protocols
  - We have discussed ARQ in the last lecture.

Other error correction techniques

- Forward error correction (FEC)
  - Correct error(s) at Rx
  - Computationally expensive
  - Require additional redundancy
  - May be useful for real-time communication or for channels with high end-to-end delays
  - Example technique: Convolutional codes
  - We do not examine these in this course

The End