Data Link Layer, Part 2
Error Detection and Correction

These slides are created by Dr. Yih Huang of George Mason University. Students registered in Dr. Huang's courses at GMU can make a single machine-readable copy and print a single copy of each slide for their own reference, so long as each slide contains the copyright statement, and GMU facilities are not used to produce paper copies. Permission for any other use, either in machine-readable or printed form, must be obtained from the author in writing.

Transmission Errors

- Causes: noises, attenuation, distortion, crosstalk, losing synchronization
- Error detection
  - Parity checks, cyclic redundancy codes, …
- Error correction
  - send redundant information with data
  - when receiving data incorrectly, the receiver makes "educated guess" about the original data
  - Ex. Hamming code

Parity Checks

- Add an extra bit to a string of bits in order to make the total number of 1's even (even parity) or odd (odd parity)
- Example (even parity): **0 1 1 0 1 1 0 1 1**
- Advantages
  - detects any single bit error
  - in fact, detects any error involving odd number of bits
- Disadvantages
  - only 50% chance of detecting **burst errors**
  - an *n*-bit burst error is a string of bits inverted during transmission

Cyclic Redundancy Codes (CRC)

- Basic idea: treat string of bits as coefficients of a polynomial that uses modulo 2 arithmetic
  - Ex. 1 0 1 0 0 1 represents $x^5 + x^3 + 1$.
- Additions and subtractions are equivalent to Exclusive-OR:

```
  1 0 0 1 1 0 1 1      1 1 1 1 0 0 0 0
+ 1 1 0 0 1 0 1 0    - 1 0 1 0 0 1 1 0
-----------------    -----------------
```

Method

- ❑ Sender:
  - – divide string (frame) by a **generator polynomial** *G(x)*
  - – tag the remainder (called a **checksum**) onto the frame when it is transmitted
- ❑ Receiver:
  - – divide the entire frame by *G(x)*
  - – a non-zero remainder indicates errors
- ❑ Example:
  - – data: 1010001101, G(x): 110101


```
1 1 0 1 0 1 ) 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0
              1 1 0 1 0 1
                1 1 1 0 1 1
                1 1 0 1 0 1
                    1 1 1 0 1 0
                    1 1 0 1 0 1
                        1 1 1 1 1 0
                        1 1 0 1 0 1
                            1 0 1 1 0 0
                            1 1 0 1 0 1
                              1 1 0 0 1 0
                              1 1 0 1 0 1
                                  0 1 1 1 0
```

Transmitted data:
1 0 1 0 0 0 1 1 0 1 0 1 1 1 0

6-bit generator produces 5-bit remainder

How CRC Works ?

- *D(x):* data
- *D'(x):* data with appended 0s
  - Let $D'(x) = P(x)G(x) + R(x)$
- *T(x):* transmitted bits; must be a multiple of *G(x)*

```
  1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 | D'(x)
-                     0 1 1 1 0 | R(x)
---------------------------------
  1 0 1 0 0 0 1 1 0 1 0 1 1 1 0 | T(x)=D'(x)-R(x)
```

- Suppose the received bits *R(x)=T(x)*, then *G(x)* also divides *R(x).*
- To detect errors, the receiver tests if $G(x) \mid R(x)$.

When Does CRC Fail ?

- *E(x):* error bits

```
  Transmitted: 1101010101000111 1
+      Errors: 00001000111000100
---------------------------------
     Received: 11011101101001011
```

- That is, $R(x) = T(x) + E(x)$
- *R(x)* is accepted by the receiver if $G(x)|R(x)$
- Hence, what is the relationship between *G(x)* and *E(x)* to cause the CRC to fail ?

Example of a CRC Failure

- From the earlier example:
  - G(x) = $x^5 + x^4 + x^2 + 1$ (110101)
  - D(x): 1 0 1 0 0 0 1 1 0 1
  - T(x): 1 0 1 0 0 0 1 1 0 1 0 1 1 1 0
  - E(x): 1 1 1 1
  - R(x): 1 0 1 0 0 0 1 0 1 0 0 1 1 1 1
  - Please check that E(x)=G(x)*(x^2+1)
  - You can check for yourself that R(x) will be (incorrectly) accepted by the receiver.

Generators Are Not Born Equal

- If *G(x)* contains two or more terms, then it can detect all single bit errors.
  - Why?

- detects all double errors if
  - x does not divide $G(x)$, and
  - $G(x)$ does not divide $x^k + 1$ for any $k < K$ where K is the frame length

  Why?

- detects all odd errors if $G(x)$ contains $x + 1$ as a factor. Why?

Other CRC Properties

- If G(x) satisfies all the above properties, then
  - all burst errors of length *r* or less are detected, where *r* is the degree of *G(x)*,
  - burst errors of length $r + 1$ are missed with probability $1/2^{r-1}$
  - burst errors of length r + 2 or more are missed with probability $1/2^{r}$

CS 656 13

Common Generators

- CRC-8: $x^8 + x^2 + x + 1$ (used with ATM)
- CRC-CCITT: $x^{16} + x^{12} + x^5 + 1$ (used with HDLC)
  - catch all single, double, and odd errors
  - catch all burst errors of length of 16 or less
  - catch 99.997% of burst errors of length 17
  - catch 99.998% of length 18 or more
- CRC –32: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ (used with Ethernet)

CS 656 14

Error Correcting Codes

- Frame consists of m data bits and r check bits.
- The resulting $n = m + r$ bit unit is called a **codeword**.
- The number of bits by which two codewords differ is called the **Hamming Distance**.
- To detect d-bit errors, we need distance of d + 1 between any pair of codewords.
  - codewords with single parity bit have a minimum distance of 2
- To correct d-bit errors, we need distance of $2d$ + 1.

An Example

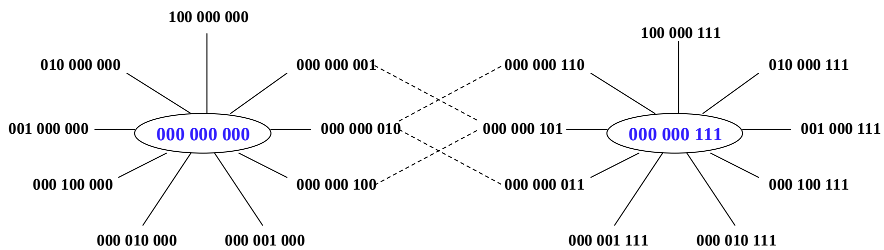
- Consider a (inefficient) coding scheme where each bit is simply repeated three times.

000	000	000	000
001	000	000	111
010	000	111	000
011	000	111	111
100	111	000	000
101	111	000	111
110	111	111	000
111	111	111	111

- The minimum Hamming distance among the above codewords is 3.
- This scheme can detect any 2-bit errors.

A Naïve Error Correction Method

- The above scheme can also be used to correct 1-bit errors.
- When receiving an invalid codeword, we assume that the original data is the closest, valid codeword.



To Correct 1-Bit Errors

- each of the 2^m legal codewords must have n + 1 bit patterns dedicated to it.
- that is, $(n + 1)2^m <= 2^n$
- divide both sides by 2^m to obtain

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

- Examples:
  - 11 data bits, how many check bits ?
  - 16 data bits, how many check bits ?
  - 32 data bits, how many check bits ?

Hamming Codes

Achieves the theoretical lower bound of check bits.

- number bits 1 to n
- power-of-2 positions are check bits
- the value of each check bit 2^k depends on the parity of the bits whose label contains that 2^k when written as the sum of powers of 2.
- to find out the incorrect bit, determine if check bits are correct
- add 2^k to a counter c if the check bit is one of the wrong parity.
- in the end, if $c = 0$, then it gives the position of the incorrect bit.

Example

original data: 1 0 1 0 1 1 0

codeword:	0	1	1	1	0	1	0	0	1	1	0
bit numbers:	1	2	3	4	5	6	7	8	9	10	11

0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11

Error Correction in Action

Error

received:	**0**	**1**	1	1	1	1	0	**0**	1	1	0
bit numbers:	1	2	3	4	5	6	7	8	9	10	11

Parity checks:

$2^0 = 1$, fail

$2^1 = 2$, pass

$2^2 = 4$, fail

$2^3 = 8$, pass

Denote pass by 0 and fail by 1.

We have: $\underset{2^3}{\mathbf{0}}\ \underset{2^2}{\mathbf{1}}\ \underset{2^1}{\mathbf{0}}\ \underset{2^0}{\mathbf{1}} = \mathbf{5}$

Discussion

- In a nutshell, error correction technologies are *educated guests* on the part of the receiver.
- Error corrections are typically used by applications that
  - can tolerate occasional errors
  - but cannot tolerate the delays of data retransmissions
- Example: multimedia playback