Error Detection and Correction

Now we are here

- OSI 7 layer model

Errors

Errors

- Definition of “error” in data link layer

- Sender sent bit 0
  - Receiver thinks sender sent bit 1, or
  - Receiver cannot figure out which bit was sent

- Sender sent bit 1
  - Receiver thinks sender sent bit 0, or
  - Receiver cannot figure out which bit was sent

Why do errors occur?

- attenuation, distortion, noise

촘촘할수록 오류가 많을 수 있음

- amplitude, frequency and phase of signal changes

no error

error

n-bit error

- Suppose sender sends a block of data “01101100”.
- Suppose the receiver receives “01101101”.
- Then we say a 1-bit error occurred.

- If the receiver receives “01011101” → 3-bit error

- The size of data block depends on the unit of error detection/correction

Error detection vs. correction

- Error detection 어느 비트가 틀렸는지 모름 -> 받은 내용을 버려야함
  - Know that an error has occurred
  - Don’t know how to correct

- Error correction 수정해서 사용할 수 있음
  - Know that an error has occurred
  - Can correct the error

Error detection

- How can we know that an error has occurred?
  - Need to insert additional bits to the original data

Block Coding for Error Detection/Correction

Block coding

- Block coding was used in physical layer to avoid continuous zeros

- Block coding is also used for error detection/correction

Block coding

- Terms
  - dataword: a block of original message
  - codeword: block-coded message

- Generate an n-bit codeword from a k-bit dataword
  - dataword: k bits
  - codeword: n bits
  - r: redundancy codeword와 dataword의 차이
  - $r = n - k$

Block coding

Error detection using block coding

- We can block-code the message as the table below
- What is k? 2
- What is n? 3
- What is r? 1
- What do we actually transmit if we want to send "01101100"?

비트를 늘려서 올 수 있는 경우, 없는 경우로 나눠서 검출

011101110000

Datawords	*Codewords*
00	000
01	011
10	101
11	110

이렇게 오는 건 오류가 아님

Error detection using block coding

- The sender wants to send “01”.
- The sender will send “011” using block coding.

- Suppose receiver received “011” (no error).
- What should the receiver do?

받은 codeword가 에러인지 판단하는 법 - 받은 codeword가 표에 없으면 에러

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Error detection using block coding

- The sender wants to send “01”.
- The sender will send “011” using block coding.

- Suppose receiver received “111” (1-bit error).
- What should the receiver do?

01을 전달할 때 이걸 codeword인 011로 바꿈
만약 011이 111로 한 자리가 바뀌면 111인 경우는 없으니 error detect
만약 011이 000로 두 자리가 바뀌면 000인 경우가 있어서 에러가 나오지는 않음
1비트가 다를 때는 검출할 수 있고 2비트가 다를 때는 검출을 못함
그럼 1비트만 다른 모든 경우의 수에서 error detect가 되는가? 네

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Error detection using block coding

- The sender wants to send “01”.
- The sender will send “011” using block coding.

- Suppose receiver received “000” (2-bit error).
- What should the receiver do? sender는 01을 보내고 싶었지만 2bit error로 receiver는 00으로 알아들음

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Error detection using block coding

- The sender sends “011”.
  - “011” received → receiver thinks dataword is “01”.
  - “111” received → receiver knows an error has occurred.
  - “000” received → receiver thinks dataword is “00” (wrong)

  → This codeword set can detect 1-bit error, but cannot detect 2-bit error.

  → Can we prove that this scheme always detect 1-bit errors?

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Error detection using block coding

- Another block coding scheme

- What is k, n, and r?

k = 2
n = 5
r = 3

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Error detection using block coding

- The sender sends codeword “01011”.
- The receiver receives “01001” (1-bit error).

- The receiver knows that there was an error.
- Can the receiver do more than that?

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Error detection using block coding

- The sender sends codeword “01011”.
- The receiver receives “01001” (1-bit error).

- Among the codewords, 01011 is “the closest” codeword to 01001.
- Thus, the receiver guesses that the dataword was 01.

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Error detection using block coding

- The sender sends codeword “01011”.
- The receiver receives “01001” (1-bit error).

- Let’s assume that there can be only 1-bit errors.
  - no 2 or more bits of error
- Then, this codeword set can correct the error.

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Error detection using block coding

- The sender sends codeword “01011”.
- The receiver receives “00001” (2-bit error).
- In this case, the closest codeword is “00000” (wrong)
  - This codeword cannot correct 2-bit error.
- However, the receiver knows that there is an error.
  - Because there is no codeword “00001”.
  - This codeword set can detect 2-bit errors.
  - How about 3-bit errors?

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Error detection using block coding

- What is the logic behind all this?

Hamming distance

- Number of <u>different bits</u> between two codewords
- d(000, 000) = 0
- d(000, 011) = 2
- d(0100, 0010) = 2
- d(10101, 11110) = 3

Minimum Hamming distance

- Given a codeword set, its MHD is the minimum of hamming distance between any two codeword in the set

- What is the MHD of this codeword set?

4C2 개의 경우의 수 중 min이 2 -> MHD 2

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Minimum Hamming distance

- What is the MHD of this codeword set?

3

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Hamming distance and error

- C_T: transmitted codeword
- C_R: received codeword

- If $d(C_T, C_R) = n$, this means there was an <u>n-bit error</u>

- Example
  - transmitted: 000
  - received: 011
  - d(000, 011) = 2, thus 2-bit error

MHD and error detection

- To detect an s-bit error, MHD of the codeword set must be larger than s.

- $d_{min} > s$

MHD and error detection

- To <u>detect</u> an s-bit error, MHD of the codeword set must be larger than s.

- $d_{min} > s$

- Suppose MHD of a codeword is s+1.
- Pick any codeword, and change s bits or less.
- This should not be in the codeword set, because then the MHD must be less than s+1.

MHD and error detection

- The MHD of this codeword set is 2.

Datawords	*Codewords*
00	000
01	011
10	101
11	110

- It means this codeword set can detect up to 1-bit errors.

MHD and error detection

- The MHD of this codeword set is 3.

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

- It means this codeword set can detect up to 2-bit errors.

MHD and error detection

dmin 미만으로 오류가 나는 경우들
어떤 오류가 나도 다른 codeword와 겹치지 않음 -> 오류 detect 보장

MHD and error correction

- To <u>correct</u> a t-bit error, MHD of the codeword set must be larger than 2t.

- $d_{min} > 2t$

ex 2bit 에러를 고치려면 MHD가 5여야 함

MHD and error correction

- To <u>correct</u> a t-bit error, MHD of the codeword set must be larger than 2t.

- $d_{min} > 2t$

- Suppose MHD of a codeword is 2t+1.
- Pick any codeword (C), and change t bits (C').
  - C' obviously is not in the codeword set.
- Choose the closest codeword to C'.
- It is C, since other codewords should have hamming distance of at least t+1.

MHD and error correction

MHD and error correction

- MHD of this codeword set is 3.

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

- This codeword set can detect up to 2-bit errors.
- <u>This codeword set can correct up to 1-bit errors.</u>

Example

- Suppose a codeword set has MHD=4.

- How many bit errors can this codeword set detect?

3

- How many bit errors can this codeword set correct?

2

Linear Block Codes

Linear block codes

- Pick any two codewords from the set
- XOR them
- The result codeword is in the codeword set

- If this is satisfied, this codeword set is called <u>linear block codes</u>

Linear block codes

- Is this linear block codes?

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Linear block codes

- Is this linear block codes?

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

MHD of linear block codes

- Exclude codeword with all zeros
- Count 1s for all other codewords
- The minimum number of 1s is the MHD of the codeword set

- What is the MHD of this codeword set?

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Simple Parity Check Codes

- k-bit dataword → (k+1)-bit codeword
  - Add 1 bit (parity bit)

- The parity bit should make the codeword have <u>even number of 1s.</u>

Simple Parity Check Codes

codeword의 1 숫자가 짝수가 되도록 뒤에 parity bit 붙임

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

- $n = k + 1$
- $d_{min} = 2$

Simple Parity Check Codes

Example

- Sender sends dataword ‘1011’ using parity check codes.

  1의 개수가 짝수가 되도록 뒤에 parity bit를 붙인 codeword를 전송

  – Codeword is ‘10111’.

- Receiver receives ‘10111’. What happens?

  통과

- Receiver receives ‘10011’. What happens?

  1비트가 다름, 오류 검출

- Receiver receives ‘10110’. What happens?

  1비트 오류가 났는데 parity bit가 오류가 난 상황, 오류 검출

Example

- Sender sends ‘10111’.

- Receiver receives ‘00110’. What happens?

2비트 오류, 오류 검출이 안됨

- Receiver receives ‘01011’. What happens?

3비트 오류, 10111(홀4) 01011(홀3) 홀짝이 달라서 오류 검출(운 좋은 케이스)

Performance of parity check codes

- Can parity check codes detect 1-bit errors?
- Can parity check codes detect 2-bit errors?
- Can parity check codes detect 3-bit errors?
- Can parity check codes detect 4-bit errors?

Performance of parity check codes

- Parity check codes can detect odd-bit errors.

- Pretty good!

2-dimensional parity

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Row parities

Column parities

parity 줄

a. Design of row and column parities

맨 뒤에 parity bit를 넣고, 몇번째 word 마다 한 줄을 다 parity로 채움

2-dimensional parity

b. One error affects two parities

2d parity로 1bit에러가 발생한 위치의 좌표를 구해서 수정할 수 있음

c. Two errors affect two parities

오류 위치를 특정할 수 없지만 오류를 검출할 수 있음

2-dimensional parity

d. Three errors affect four parities

오류 위치를 특정할 수 없지만 오류를 검출할 수 있음

e. Four errors cannot be detected

에러 검출이 안되는 케이스
4비트는 검출이 될 수도 안 될 수도

Performance of 2-D parity

- 1-bit errors: detect
- 2-bit errors: detect
- 3-bit errors: detect
- 4-bit errors: cannot detect

- 2-D parity can detect <u>some</u> of 4-bit errors, but cannot detect <u>all</u> 4-bit errors.

- If a codeword is said to detect n-bit errors, it should detect <u>all</u> of n-bit errors.

Hamming Code

- A family of error detection codes invented by Richard Hamming in 1950.

- $d_{min} = 3$
  - detect up to 2-bit errors
  - correct up to 1-bit errors

- Hamming (n,k) code
  - convert a k-bit dataword into an n-bit codeword

Hamming (7,4) code

- Convert 4-bit datawords into 7-bit codewords

※ Note

- There are different ways to generate Hamming (7,4) code
- If you have textbooks, the coding method will be different from the slides
- I suggest that you should follow the slides (not the textbook)

총 7개 비트 중 4개가 데이터

Hamming (7,4) code

- Number the bits from the left.
- If the index of the bit is a power of 2, it is a <u>check</u> bit.
  - 1st, 2nd, 4th bit
- If the index of the bit is not a power of 2, it is a <u>data</u> bit.
  - 3rd, 5th, 6th, 7th bit

2의 배수 자리수에는 체크용 비트를 넣고
아닌 자리수에는 전달할 데이터를 넣음

1	2	3	4	5	6	7
C_1	C_2	D_3	C_4	D_5	D_6	D_7

Hamming (7,4) code

- The 4 bits of the dataword go into the data bits
  - If dataword is “1011”
  - $D_3 = 1$
  - $D_5 = 0$
  - $D_6 = 1$
  - $D_7 = 1$

인덱스를 이진수로 나타냄

001	010	011	100			
1	2	3	4	5	6	7
C_1	C_2	D_3	C_4	D_5	D_6	D_7

Hamming (7,4) code

- Check bits
  - j^{th} bit: select k^{th} bits where j & k = j
    - &: bitwise AND

and해서 비교하는 비트의 자리수가 나오면 영향을 미침
ex. 001 & 011 = 001 즉 1번째 비트는 3번째 비트에 영향을 줌
그리고 1,2,4가 체크 비트이기 때문에 이들의 특징은 한 자리수만 1이다.
비교하는 상대 비트랑 같은 자리수에 1이 있으면 영향을 준다. 그러니까 굳이 and해서 비교 안해도 됨
ex. 001(1번)이랑 101(5번)을 비교할 때 1번이 맨 끝 비트가 1이고 5도 그러니 5번은 1번 영향을 받음

- 1st bit: XOR values of 3rd, 5th, and 7th bits
  - 001 & 011 = 001, 001 & 101 = 001, 001 & 111 = 001

1인덱스의 체크 비트의 영향을 미치는 비트는 3,5,7

Hamming (7,4) code

- Check bits
  - j^{th} bit: select k^{th} bits where j & k = j
    - &: bitwise AND

- 2nd bit: XOR values of 3rd, 6th, and 7th bits

3,6,7 번째 비트의 1의 개수가 홀수면 c2가 1, 짝수면 c2가 0

Hamming (7,4) code

- Check bits
  - j^{th} bit: select k^{th} bits where j & k = j
    - &: bitwise AND

- 4th bit: XOR values of 5rd, 6th, and 7th bits

Hamming (7,4) code

mhd가 3인 이유!
예를 들어 데이터가 1011인 걸 0011로 한 비트만 바꾸고 싶음
이 데이터를 hamming code로 나타냈을 때는 D3 부분이 바뀐건데(1011은 데이터니까) 이 때 영향을 미치는 c1, c2 또한 바뀌어서 code word 상에서 총 3 개가 바뀐다

Hamming (7,4) code

- Exercise

- 0000 → 0000000

- 0011 → 1000011

- 1010 → 1011010

- 1110 → 0010110

Hamming (7,4) code

- What is the MHD of Hamming (7,4) code?

3

- Why?

Hamming (7,4) code

- Why (7,4)?

- Suppose we use n-bit codeword, where $n = 2^m-1$ for a natural number m.

- Suppose the dataword is k bits, so that
- $k = n - \log_2(n+1)$

- Is there a better way to correct 1-bit errors?

Hamming (7,4) code

- To correct a 1-bit error in an n-bit codeword, the receiver should be able to correctly pick one of the following cases:
  - No error
  - 1st bit is error
  - 2nd bit is error
  - 3rd bit is error
  - …
  - $(n-1)^{th}$ bit is error
  - n^{th} bit is error

1 비트 오류가 있을 케이스 - 8 개(각 자리수 7 + 안바뀔 경우 1)
8개의 케이스를 표현하려면 최소 3비트가 필요함

- How many cases? n+1

Hamming (7,4) code

8 pieces of information (각 자리수 7 + 안바뀔 경우 1)

- For a n-bit codeword, n+1 pieces of information is needed in order to correct a 1-bit error.

- To embed n+1 pieces of information, we need $\log_2(n+1)$ bits

- Thus Hamming code uses minimum cost to correct 1-bit errors. There cannot be a better way.
  - Hamming code is described as a perfect code. 가장 최적화된 코드

Voting

- The simplest way to increase hamming distance
- Convert ‘0’ into N number of ‘0’s
  - If n = 3, ‘0’ → ‘000’, ‘1’ → ‘111’
- Makes hamming distance N
- If receiver receives ‘010’, it is interpreted as 0 because the codeword has more 0s than 1s → voting

2^3 - 1 , 2^3 - 1 -3

Voting (N=3) vs. Hamming (7,4)

- Efficiency of voting = 1/3 = 33.3%

code word 2^k-1
dataword 2^k-1 - k

codeword가 2^k-1 인 이유
최소한의 비트로 최대한의 1비트 에러를 체크할 수 있는 것이다
ex. codeword가 14여도 15랑 같이 4비트가 필요하다 하지만 16이면 5비트가 필요해짐
그래서 2^4 - 1개의 비트의 codeword

- Efficiency of hamming (7,4) = 4/7 = 57.1%

Hamming Codes

- Recall that to correct 1-bit error in 7-bit codeword, we need 3 control bits
  - 8 pieces of information

- In general, suppose we have a (2^k-1)bit codeword. Then, to correct 1-bit error we need k control bits

2^4-1, 2^4 -1 - 4

- If k=4, we obtain Hamming (15, 11) code
- If k=5, we obtain Hamming (31, 26) code

k를 늘려 해밍코드를 늘릴 때 장점 - 전송률이 좋아짐 26/31이 4/7보다 좋다
에러를 확인할 때 봐야하는 비트가 많아짐

- The efficiency rate increases with k

Hamming (15,11) Code

- Dataword is 10110001011
- Convert dataword into codeword using hamming (15,11) code

101101110001011

code word bit의 인덱스를 이진수로 표현했을 때, 같은 자리에 1값을 가지는 인덱스의 비트들을 xor(1의 홀짝 개수 파악 홀수면 1, 짝수면 0)

- What is the efficiency rate?
  - 11/15 = 73.3%

Hamming Codes: Summary

- Size of the control bits = k
- Size of the codeword = 2^k-1
- Size of the dataword = 2^k-1-k

- Efficiency rate = (2^k-1-k) / (2^k-1)

- As k increases, efficiency rate increases
- However, as the dataword becomes longer, the possibility of having 2 or more bit-errors increases → detection rate decreases

Checksum

Checksum

- An error detection method slightly more complex than parity check codes

Checksum

- Suppose we want to send five 4-bit numbers
  - 7, 11, 12, 0, 6

    0111 1011 1100 0000 0110

- Add all numbers and place the sum as the last number
  - 7, 11, 12, 0, 6, <u>36</u>

    프레임에 checksum을 위한 자리에 넣음

- The receiver checks for errors by adding all numbers and comparing with the last number
  - 7 + 11 + 12 + 0 + 6 = 36 → no error!

    보내고자 하는 데이터의 값을 다 더해서 값이 맞으면 에러 없음

Checksum

- We can also do this like the following.

컴퓨터는 다 더해서 0이 되는지를 비교하는 게 더 빠름

- Add minus sign to the sum
  - 7, 11, 12, 0, 6, <u>-36</u>

- The receiver simply adds all numbers and see if it is zero
  - 7 + 11 + 12 + 0 + 6 + (-36) = 0 → no error!

Wrapped Sum

- The problem is: 36 is not a 4-bit number

4비트로 만들기 위해 wrapped sum을 함

checksum은 값이 무한이 늘어날 수 있는데, checksum을 넣을 프레임은 정해져 있음 -> checksum을 4비트로 맞추자

- We calculate a "wrapped sum"
  - 36 is "100100" in binary representation
  - keep the lower 4 bits, and add the number to the higher 2 bits

```
   0100
 +   10
--------
   0110
```

  - The wrapped sum of 36 is 6

Wrapped Sum

- What is the 4-bit wrapped sum of 45?

- 45 → 101101 (binary)

```
    1101
 +    10
----------
    1111
```

- 4-bit wrapped sum of 45: 15

Wrapped Sum

- What is the 4-bit wrapped sum of 63?

- 63 → 111111 (binary)

```
    1111
 +    11
---------
   10010
```

- This is still over 4 bits!

Wrapped Sum

- In this case, wrap one more time

```
    0010
 +     1
---------
    0011
```

- 4-bit wrapped sum of 63: 3

Checksum

- Now we need to make the number negative (-6) using 4 bits
- Use 1's complement to represent -6
- 1's complement: toggle all bits
  - 6 = 0110
  - -6 = 1001
- '1001' is 9 in decimal representation

Checksum

- The sender place ‘9’ as the last number
  - 7, 11, 12, 0, 6, 9
- The receiver receives the numbers and adds them
  - 7 + 11 + 12 + 0 + 6 + 9 = 45
- Calculate a wrapped sum of 45 (101101)

```
    1101
 +    10
--------
    1111
```

- Find 1’s complement of 1111 = 0000 → correct!
  - If the final value is not 0, then there is an error

Checksum

Sender site
7
11
12
0
6
0
Sum
36
Wrapped sum
6
Checksum
9
1 0 0 1 0 0 36
1 0
0 1 1 0 6
1 0 0 0 9
Details of wrapping and complementing
7, 11, 12, 0, 6, 9
Packet
Receiver site
7
11
12
0
6
9
Sum
45
Wrapped sum
15
Checksum
0
1 0 1 1 0 1 45
1 0
0 1 1 0 15
1 0 0 0 0
Details of wrapping and complementing

Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check

- A complex and powerful error detection method
- Complexity and power of CRC can be controlled by selecting “divisor (generator)”

Modulo Arithmetic

- A basic arithmetic used in CRC and other coding methods

- Modulo-N: use N numbers [0, N-1]

- Example: modulo-5
  - 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4
  - 3 + 2 = 0
  - 4 + 3 = 2
  - 2 – 3 = 4

Modulo-2

- Use only 0 and 1

  xor과 동일

- $0 + 0 = 0,\ \ 0 + 1 = 1,\ \ 1 + 0 = 1,\ \ 1 + 1 = 0$

- $0 - 0 = 0,\ 0 - 1 = 1,\ 1 - 0 = 1,\ 1 - 1 = 0$

- Addition and subtraction lead to the same results
  - addition = subtraction

Modulo-2

- Addition and subtraction are equal to XOR

$0 \oplus 0 = 0 \qquad 1 \oplus 1 = 0$

a. Two bits are the same, the result is 0.

$0 \oplus 1 = 1 \qquad 1 \oplus 0 = 1$

b. Two bits are different, the result is 1.

$$
\begin{array}{cccccc}
 & 1 & 0 & 1 & 1 & 0 \\
\oplus & 1 & 1 & 1 & 0 & 0 \\
\hline
 & 0 & 1 & 0 & 1 & 0
\end{array}
$$

c. Result of XORing two patterns

Cyclic Redundancy Check

- Suppose we use the divisor ‘1011’
- We want to send the dataword ‘1001’
- Since the divisor is 4 bits, we add ‘000’ after the dataword → ‘1001000’
- Divide ‘1001000’ by the divisor ‘1011’
  - Use modulo-2 arithmetic

Cyclic Redundancy Check

- Dividing 1001000 by 1001 using modulo-2 arithmetic

나머지를 dataword 뒤에 보냄
수신자가 수신한 내용을 몫으로 나눴
을 때 나머지가 0이면 오류가 없다

Cyclic Redundancy Check

- Since the remainder is ‘110’, we add this to the dataword to make the codeword

  → <u>1001110</u>

- Send this codeword

Cyclic Redundancy Check

- The receiver receives the codeword and divide the number by the divisor
  - If remainder is 0, then there is no error

Cyclic Redundancy Check

- If the remainder is not 0, then error detected

Binary Polynomials

- We can represent a binary number as a binary polynomial

a. Binary pattern and polynomial

b. Short form

Binary Polynomials

- Division of binary polynomials

Cyclic Redundancy Check

- dataword(message): M(x)
- divisor: g(x)
- codeword: T(x)
- T(x) is divisible by g(x)
  - Remainder of T(x)/g(x) is 0

Cyclic Redundancy Check

- Sender sends $T(x)$
- Receiver receives $T'(x)$
- Receiver calculates $T'(x) / g(x)$
- If the remainder is zero, the receiver presumes $T(x) = T'(x)$ (no error)

Cyclic Redundancy Check

- When does CRC fail to detect an error?

ex!=0이고
ex / gx = 0일 때 오류 검출에 실패함

- An error has occurred but, still the modified codeword T'(x) is divisible by g(x)
  → error detection fails

- T'(x) = T(x) + e(x)
  - e(x) is the error
  - If T(x) + e(x) is divisible by g(x), then the receiver thinks the codeword is correctly received

- Error detection performance of CRC depends on g(x)

Performance of CRC

- Suppose the divisor has x^0 term, and has at least one other term
  - e.g. $g(x) = x^2 + 1$

맨 밑이 1이고 다른 자리에 1이 1 이상 있으면 1bit 에러를 다 알 수 있음

- Then, all 1-bit errors can be detected

- Why?

Performance of CRC

- Which of the following divisors can detect all 1-bit errors?

- a. $x + 1$
- b. x^3
- c. 1

Performance of CRC

- If degree of the polynomial is n
  - detects all burst errors of size n or smaller
  - If size of the burst is n+1, the error can be detected with probability $1 – (1/2)^{n-1}$
  - If size of the burst is larger than n+1, the error can be detected with probability $1 – (1/2)^{n}$

Burst Error

- Multiple bits are corrupted
- Length of the burst error: from the first corrupted bit to the last corrupted bit

Performance of CRC

- Suppose the divisor is $x^6 + 1$
  - All burst errors with size 6 or less are detected
  - If the length of the burst is 7, then error detection probability is 31/32
  - If the length of the burst is 8 or higher, then error detection probability is 63/64
- Powerful!

Performance of CRC

- CRC divisors used in practice

Name	*Polynomial*	*Application*
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

데이터 전송의 단위 프레임
한 프레임을 보낼 때 그 프레임에 대한 crc를 더함

End of Class