데이터통신

비트오류탐지 수정

충남대학교
컴퓨터공학과
이영석

오늘 공부할 것

- 음...
  - 대정시 유선구 국동 춘남대학교?

  - 대정시 vs 대전시
  - 유선구 vs 유성구
  - 군동 vs 궁동
  - 춘남대학교 vs 충남대학교

- 해결방법?
  - 사전에 있는 말과 유사도 매칭

비트 오류가 있는지 검사

- Parity
- CRC
- Checksum

기초용어

- 데이터워드
  - 원래 데이터 k bits
- 코드워드
  - 원래 데이터를 오류검사가 가능하게 코딩한 데이터 n = k + r bits

송신자는 데이터워드로부터 코드워드를 생성한다. 수신자에게 전송된 각 코드워드는 변형될 수 있다. 수신된 코드워드가 유호 코드워드와 같으면 용인되어 코드워드로부터 데이터워드를 추출한다. 수신된 코드워드가 유효하지 않으면 버려진다. 그러나 전송 도중 코드워드가 손상되었거나 수신된 코드워드가 여전히 유효 코드워드 중 하나라면 오류는 검출되지 않은 태로 남는다.

오류 검출 코드는 찾도록 설계된 오류만을 찾아낸다. 다른 오류는 검출되지 못한다.

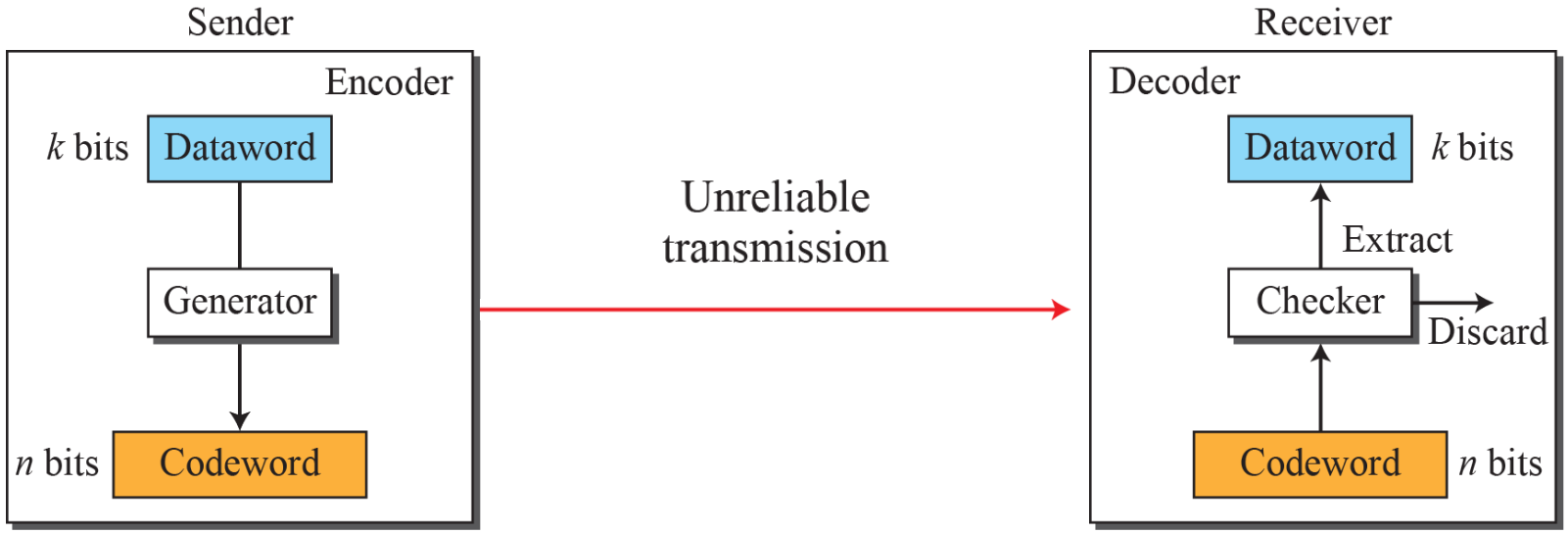


Figure 10.2: Process of error detection in block coding

기초용어2

- 해밍(Hamming) 거리
  - d(000, 011) -> 2
  - d(10101, 11110) -> 3

두 같은 크기의 워드 사이의 해밍거리는 서로 차이가 나는 해당 비트들의 갯수. 수신된 코드워드와 발신된 코드워드 사이의 해밍거리는 전송되는 동안에 오류가 생긴 비트들의 수이기 때문에 오류 검출에서 중요함. 해밍 거리가 0이 아니면 코드워드가 전송되는 동안에 오류가 발생한 것임.

- 최소 해밍 거리
  - 모든 코드워드간의 해밍거리 중에서 가장 작은 값

예

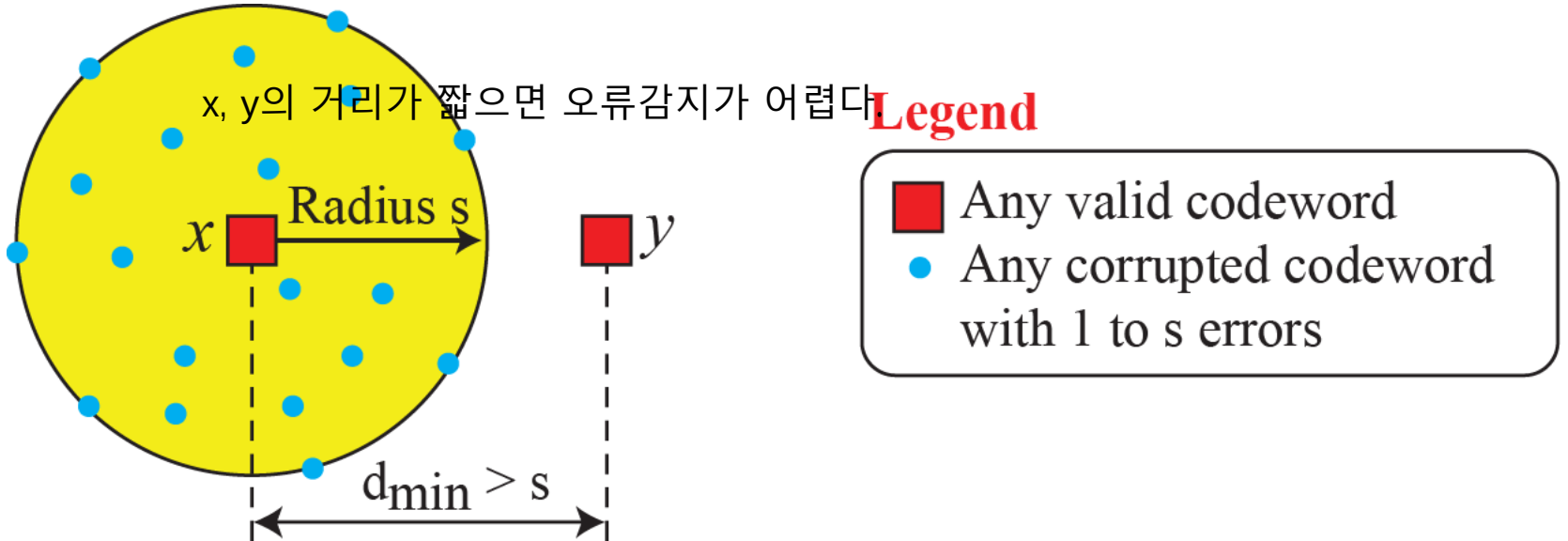
- 2비트 전송을 한다. 1비트를 추가해서 코드워드를 만든다.
  - 00 -> 000
  - 01 -> 011
  - 10 -> 101
  - 11 -> 110

- 이 때 수신받은 비트가 (01을 보낸다고 가정)
  - 011 이라면? OK (01)
  - 111 이라면? 오류(유효 코드가 아니므로 버려짐)
  - 000 이라면? OK(부정확한 데이터 워드인 00을 추출)

코드워드 비트 에러 탐지

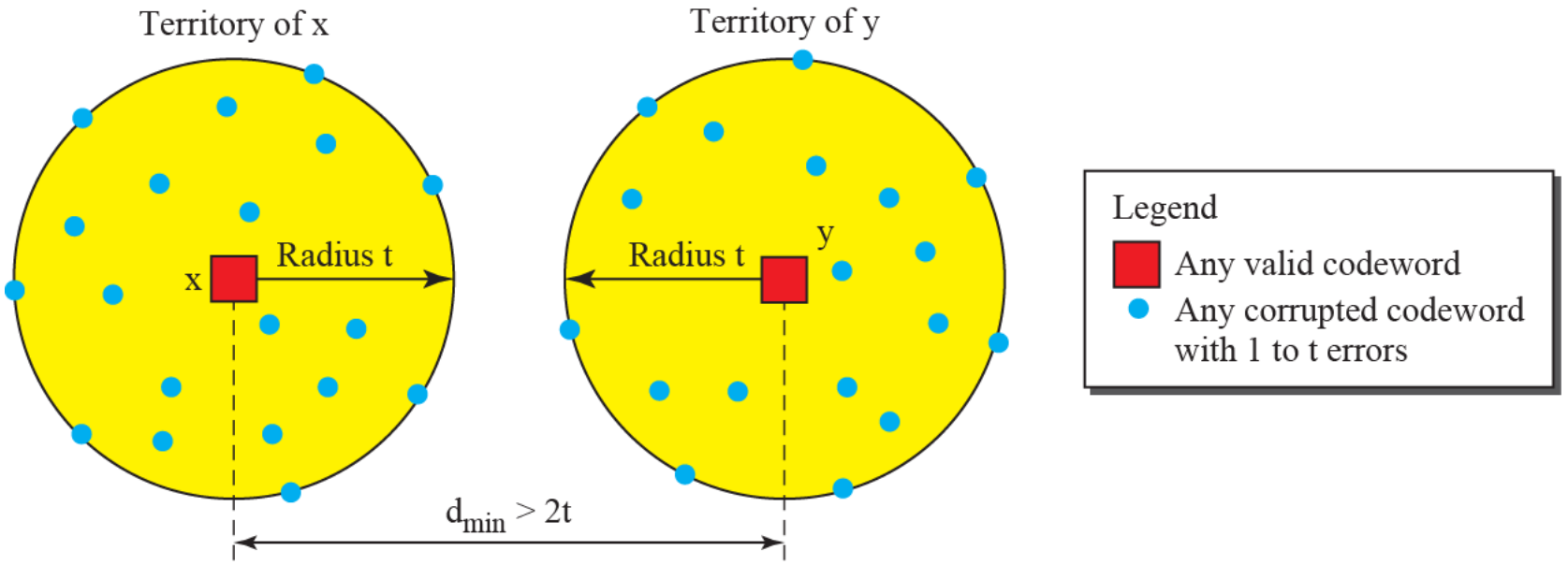
d = s+1인 코드는 특별한 경우에는s 개보다 많은 수의 오류가 발생해도 검출할 수 있으나, 오직 s개 이하의 오류가 발생한 경우에 오류를 검출할 수 있다는 것을 보장한다는 것이다.

- x 와 y 는 코드워드
  - $d_{min} = s + 1$ (손상된 채로 수신된 코드가 유효 코드 중의 하나가 될 수 없게 하기 위해. 다시 말해 s+1이면 수신된 코드는 잘못 다른 유효 코드로 인식되지 않음)
  - 유효한 코드워드와 에러가 있는 코드워드의 비트 오류가 발생했을 때
    - s 비트 만큼은 오류가 있다는 것을 알 수 있음



비트에러 탐지에서 수정까지 하려면?

- 오류가 없는 코드워드를 멀리 떨어뜨려야한다.
  - $d_{min} = 2t + 1$
  - t 비트 교정할 수 있음33



Parity Bit

- Even or Odd
  - 1의 개수 맞추기
- 오류탐지 능력
  - 짝수개 비트 오류 발생시?

7 비트 데이터	패리티 포함 8 비트	
	짝수	홀수
0000000 (0)	**0**0000000	**1**0000000
1010001 (3)	**1**1010001	**0**1010001
1101001 (4)	**0**1101001	**1**1101001
1111111 (7)	**1**1111111	**0**1111111

CRC

- n 비트 CRC 계산
  - (n+1)비트 나누기(XOR)
- 데이터
  - 11 0100 1110 1100

다항식	제수	비트수	CRC
$x^3 + x + 1$	1011	4비트	3비트

```
          1
     --------------------
1011 ) 11010011101100 000    <--- 오른쪽으로 3비트 후부터
       1011                  <--- 제수(divisor) 4비트 <= x³ +x+1
------------------
       01100011101100 000    <--- 결과
```

```
1101 XOR 1011 => 0 110
```

11 0100 1110 1100 + 100

```
   11110001111 100   <--- 몫은 별로 의미가 없는 중간 과정이다.
------------------
11010011101100 000   <--- 오른쪽으로 3비트 후부터
1011                 <--- 제수
01100011101100 000   <--- 결과
 1011                <--- 제수 ...
00111011101100 000
  1011
00010111101100 000
   1011
00000001101100 000
    0000             <--- 0인 경우 위의 계산결과가 바뀌지 않는다. 그대로 넘겨도 된다.
00000001101100 000
     0000
00000001101100 000
      0000
00000001101100 000
       1011
00000000110100 000
        1011
00000000011000 000
         1011
00000000001110 000
          1011
00000000000101 000
           101 1
00000000000000 100   <--- 왼쪽이 모두 00이면 여기서 끝내도 된다. 뒤에 오는 것은 0은 계산에 영향이 없다.
            00 00
00000000000000 100
             0 000
-----------------
00000000000000 100 <--- 나머지(remainder) 3비트, 앞에는 모두 00이 되고 뒤에 3비트가 최종 결론이다.
```

CRC

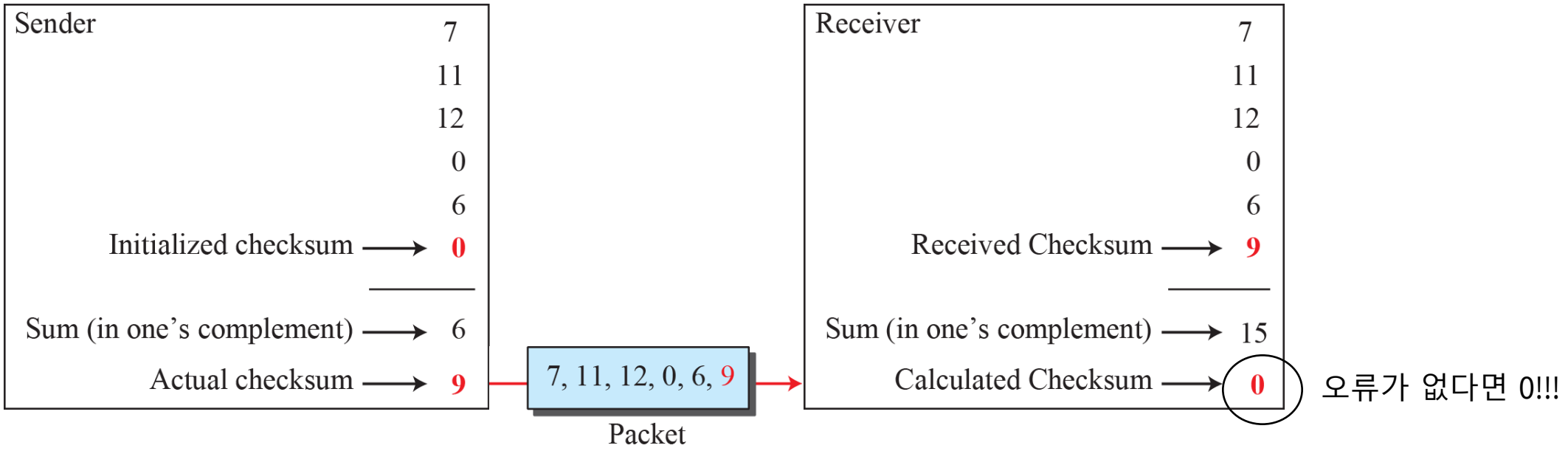
- 수신시 계산

```
11010011101100 100 <--- 입력과 CRC 체크값을 붙이고
1011               <--- 나누고
01100011101100 100 <--- 결과
 1011              <--- 나누고 ...
00111011101100 100


......

00000000001110 100
          1011
00000000000101 100
           101 1
----------------
               0 <--- 나머지
```

- Example
  - CRC-7: SD카드
  - CRC-16: Bluetooth
  - CRC-30: CDMA
  - CRC-32: 이더넷, MPEG2, Gzip, PNG

https://ko.wikipedia.org/wiki/%EC%88%9C%ED%99%98_%EC%A4%91%EB%B3%B5_%EA%B2%80%EC%82%AC

비트에러탐지코드: Checksum



- 4 비트 정수 표시
  - 7+11+12+0+6 = 36 -> $(10\mathbf{0100})_2$ -> $(10)_2 + (0100)_2 = (0110)_2 = 6$ 보수취하면 (1001) = 9
- 수신자 체크섬계산
  - 7 + 11 + 12 + 0 + 6 + 9 = 45 -> 10 + 1011 = 1111 -> 0000

IPv4 Header Checksum

- IPv4 헤더(볼드 폰트), 0xb861은 체크섬

4500 0073 0000 4000 4011 b861 c0a8 0001
c0a8 00c7 0035 e97c 005f 279f 1e4b 8180

- 어떻게 0xb861 체크섬이 나오나?

4500 + 0073 + 0000 + 4000 + 4011 + c0a8 + 0001 + c0a8 + 00c7 = 2479C

2 + 479C = 479E

  - 0x479E의 1의 보수는?
- 수신자 검사

4500 + 0073 + 0000 + 4000 + 4011 + b861 + c0a8 + 0001 + c0a8 + 00c7 = 2fffd
2 + fffd = ffff
0000 -> no error

Hamming Distance One-line Code

```
def hamming(a, b):
    return len([i for i in filter(lambda x: x[0] != x[1], zip(a, b))])
```