컴퓨터 네트워크

제 10 장 오류 제어와 흐름 제어

목차

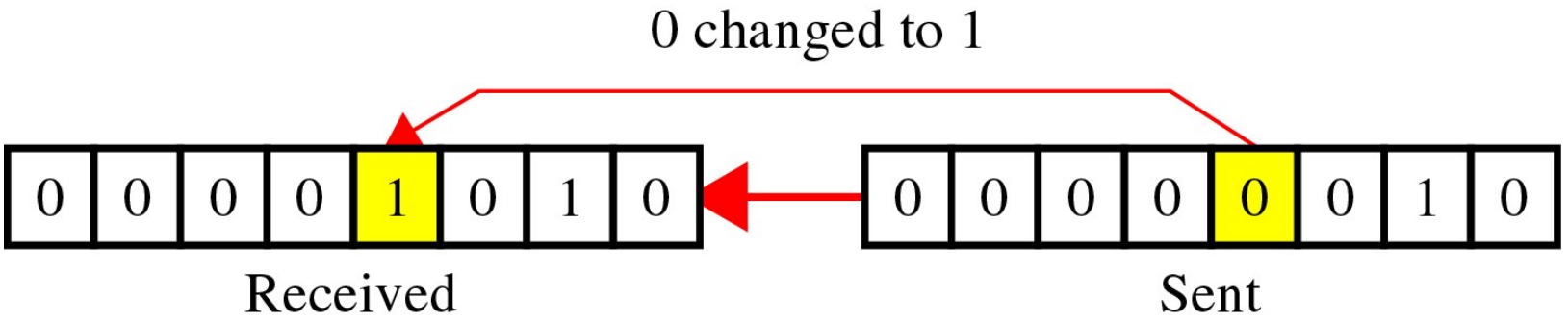
10.1 오류 제어 (Error Control)

10.2 흐름 제어 (Flow Control)

오류의 종류

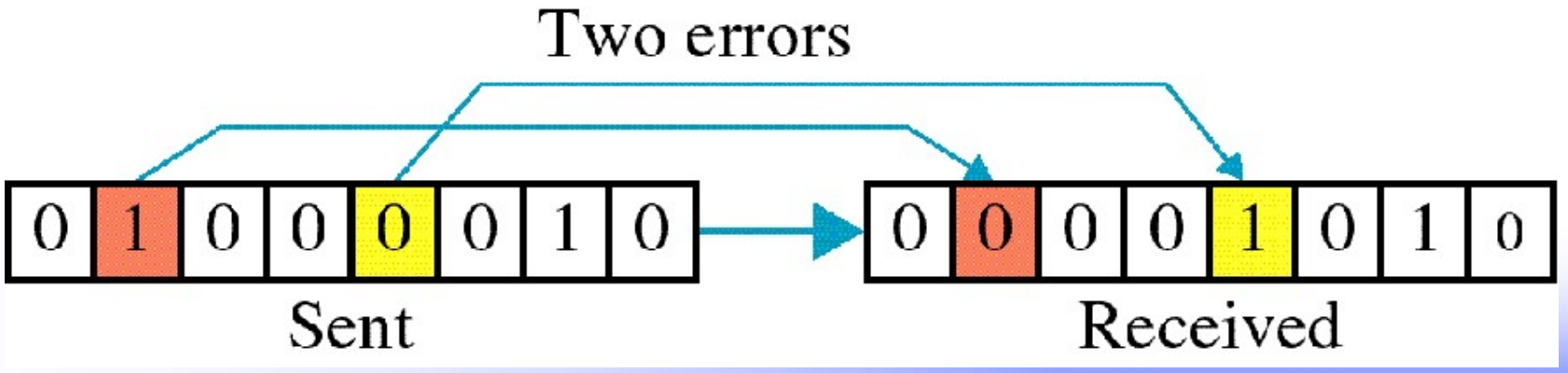
단일-비트 에러(Single-Bit Error)

- ✓ 데이터 부분의 한 비트만 변경



다중-비트 에러(Multiple-Bit Error)

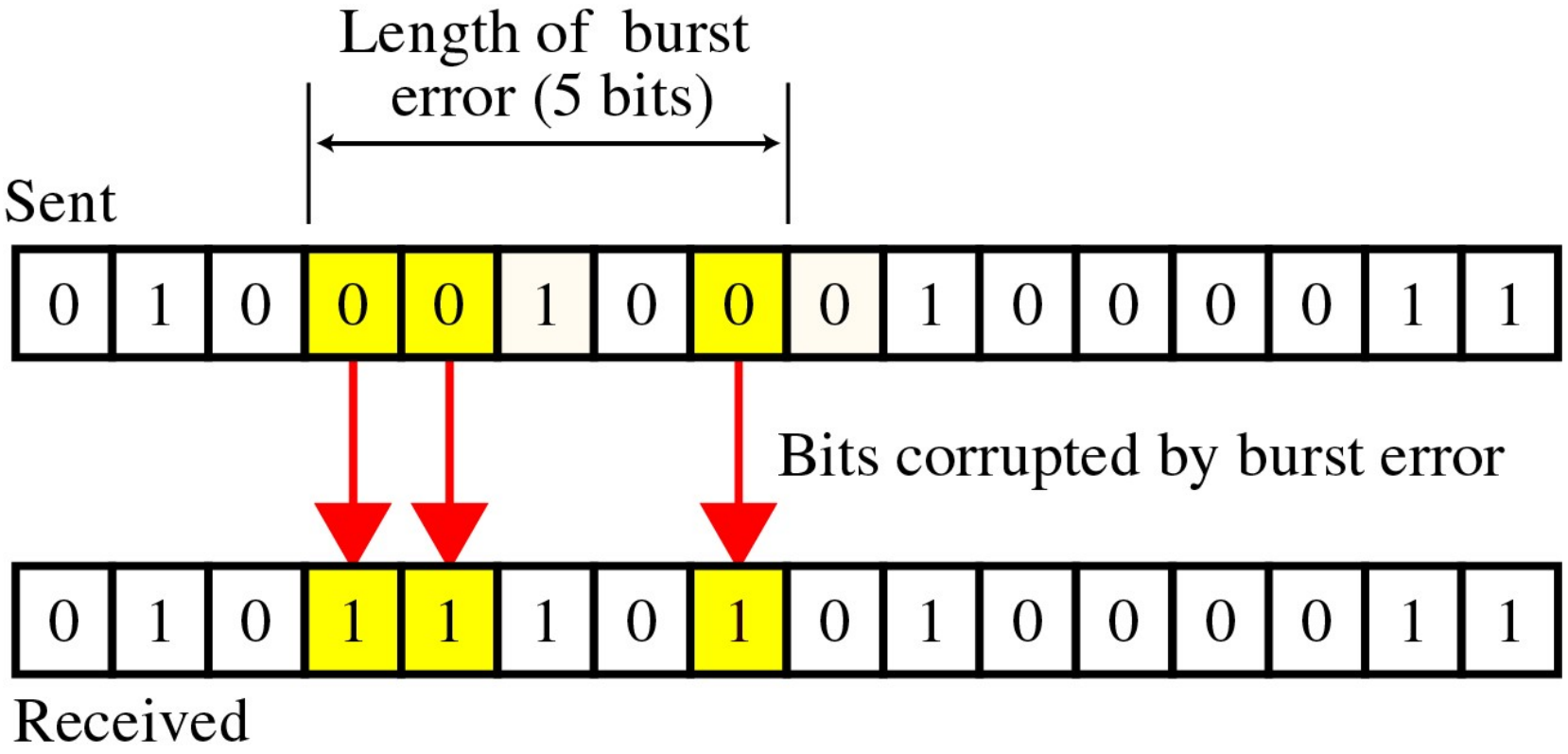
- ✓ 데이터 부분의 2개 또는 그 이상의 비연속적인 비트가 변경



오류의 종류

집단 오류(Burst Error)

- ✓ 데이터 부분의 2개 또는 그 이상의 연속적인 비트가 변경

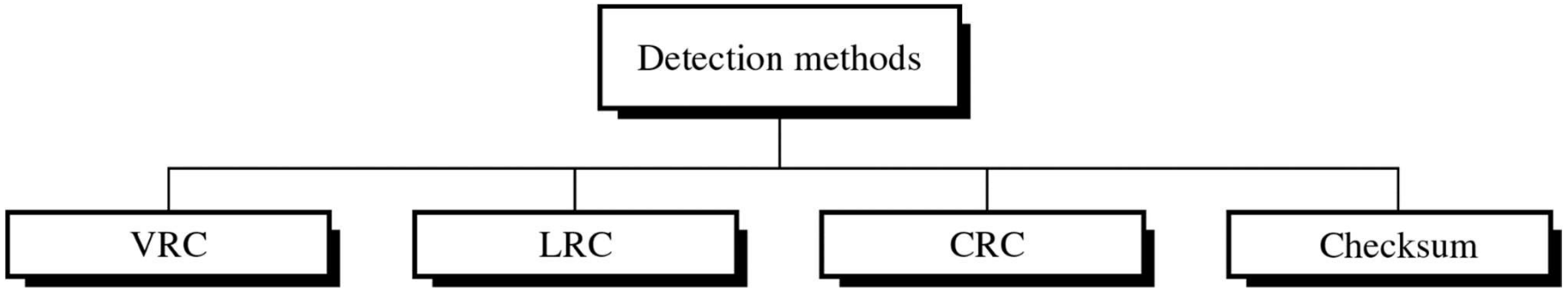


10.1 오류 제어(1/43)

오류 검출 (Error Detection)

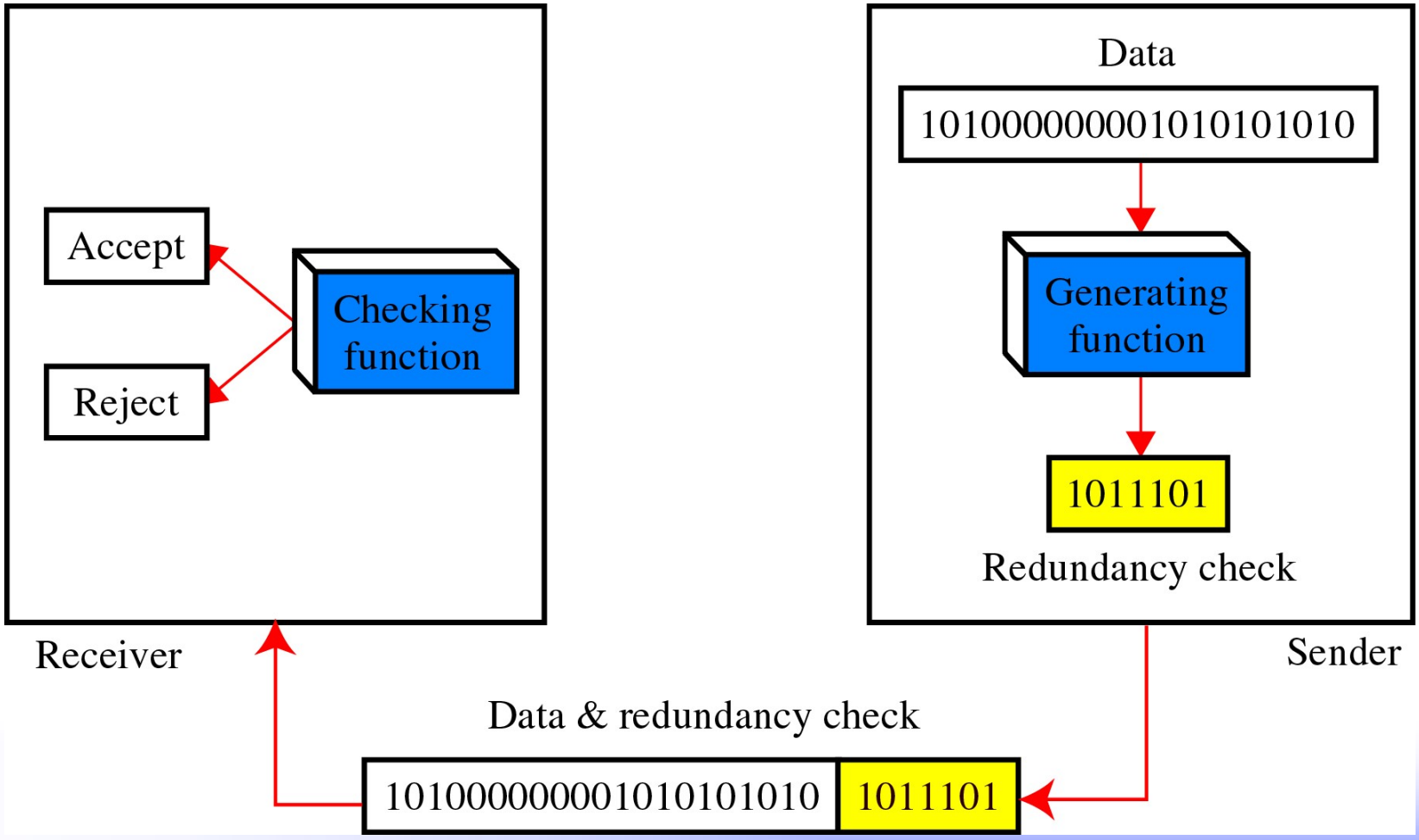
- ✓ 송신측에서 보내고자 하는 원래의 정보 이외에 별도로 잉여분의 데이터를 추가
- ✓ 수신측에서는 이 잉여(Redundancy) 데이터를 검사함으로써 오류검출이 가능

- ✓ 종류
  - ➢ 패리티 검사, 블록 합 검사, CRC(Cyclic Redundancy Check) , Checksum 등



- ➢ VRC(Vertical Redundancy Check), LRC(Longitudinal Redundancy)

잉여 정보(redundancy)



10.1 오류 제어(2/43)

패리티 검사(Parity Check)

- ✓ 한 블록의 데이터 끝에 한 비트 추가
- ✓ 구현이 간단하여 널리 사용
- ✓ 종류
  - ➢ 짝수 패리티 : 1의 전체 개수가 짝수개
  - ➢ 홀수 패리티 : 1의 전체 개수가 홀수개

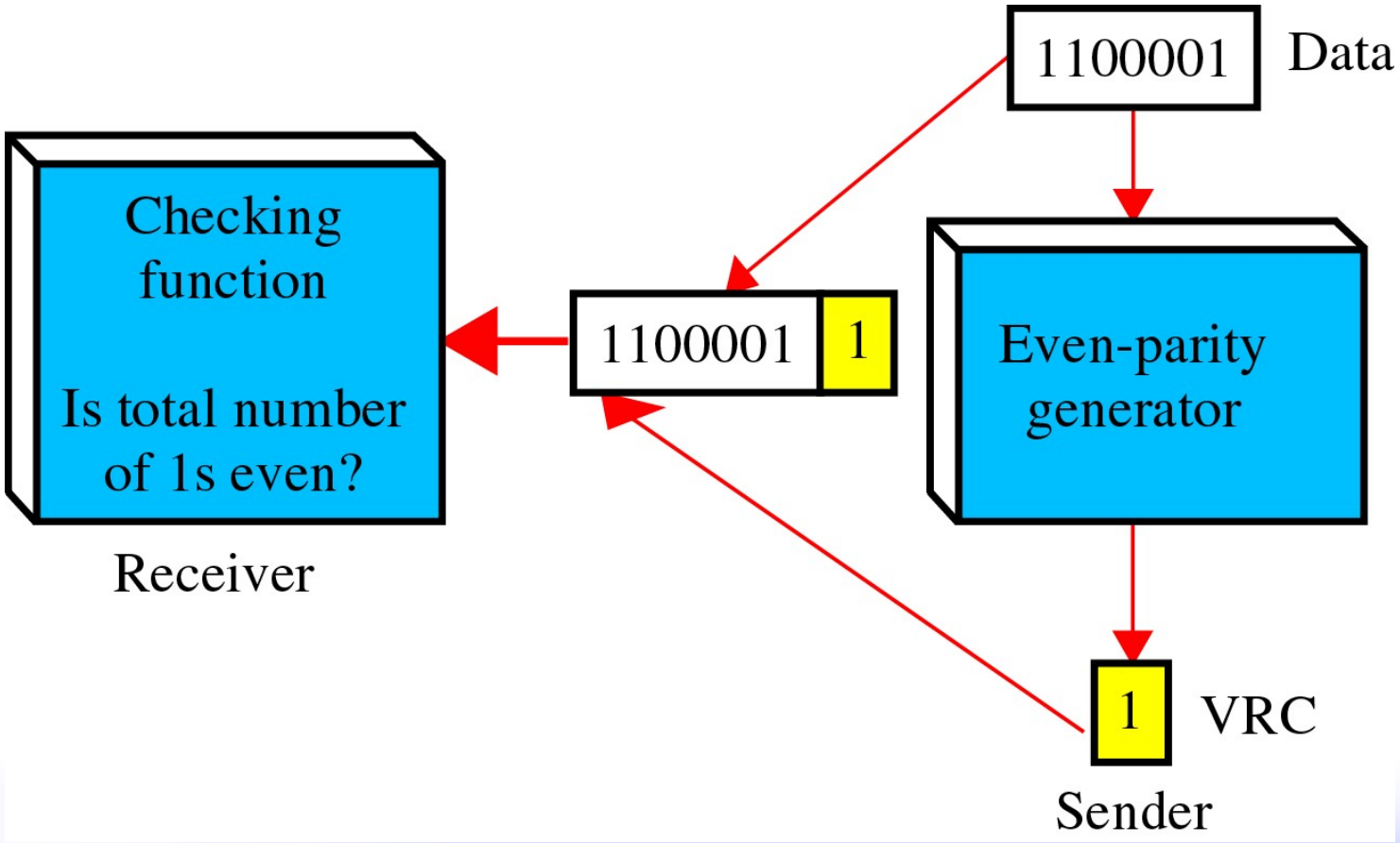
- ✓ 동작과정
  - ➢ 송신측
    - ▪ 짝수 또는 홀수 패리티의 협의에 따라 패리티 비트 생성
    - ▪ ASCII 문자(7bit) + 패리티 비트(1bit) 전송
  - ➢ 수신측
    - ▪ 1의 개수를 세어 오류 유무 판단(짝수 또는 홀수)
    - ▪ 맞지 않다면 재전송 요청

10.1 오류제어(3/43)

- 예
  - 홀수 패리티 사용
  - 전송하고자 하는 데이터 : 1101001
  - 1의 개수를 홀수로 하기 위해 패리티 비트를 1로 지정
  - 패리티 비트 추가한 최종 전송 데이터 : 11101001
  - 수신측은 패리티 비트를 포함한 데이터 내의 1의 개수를 세어 홀수인지 판단
  - 홀수가 아니면 재전송 요청

- 단점
  - 짝수개의 오류는 검출 불가
  - 예
    - 11011001 : 짝수개의 오류가 발생하여 1의 개수가 홀수인 경우

짝수 패리티 VRC(Vertical Redundancy Check)



10.1 오류제어(4/43)

블록 합 검사(Block Sum Check)

- ✓ 이차원 패리티 검사 : 가로와 세로로 두 번 관찰
- ✓ 검사의 복잡도를 증가
  - ➢ 다중 비트 오류와 폭주오류를 검출할 가능성을 높임
- ✓ 동작과정
  - ➢ 데이터를 일정크기의 블록으로 묶음
  - ➢ 각 블록을 배열의 열로 보고 패리티 비트를 계산하여 추가
  - ➢ 각 블록의 행에 대한 패리티 비트를 계산하여 추가한 후 전송

10.1 오류제어(5/43)

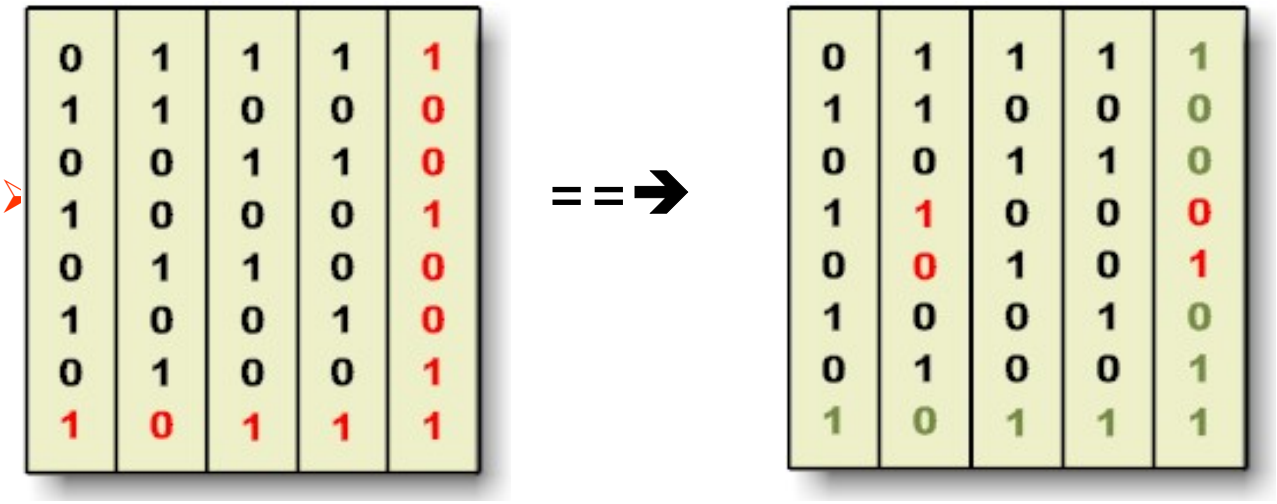
- 예
  - 전송하고자 하는 데이터

  - 그림의 최하위 비트들이 함께 더해지고 블록 합에 따라 짝수 혹은 홀수 중 하나의 패리티를 얻음
  - 두 번째 비트들이 더해지고 패리티 비트가 얻어짐
  - 블록 합의 마지막 비트는 블록 합 데이터 단위 자체를 위한 패리티 비트이고 블록 내의 모든 패리티 비트들을 위한 비트임
  - VRC(Vertical Redundancy Check), LRC(Longitudinal Redundancy)

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

10.1 오류제어(6/43)

- ✓ 단점
  - ➢ 하나의 블록에서 두 개에 오류가 생기고, 다른 블록의 동일한 위치에서 두 개의 오류가 발생한 경우 검출 불가



  - ➢ 두 번째 블록과 마지막 블록의 동일 위치에 각 두 개의 오류발생

10.1 오류제어(7/43)

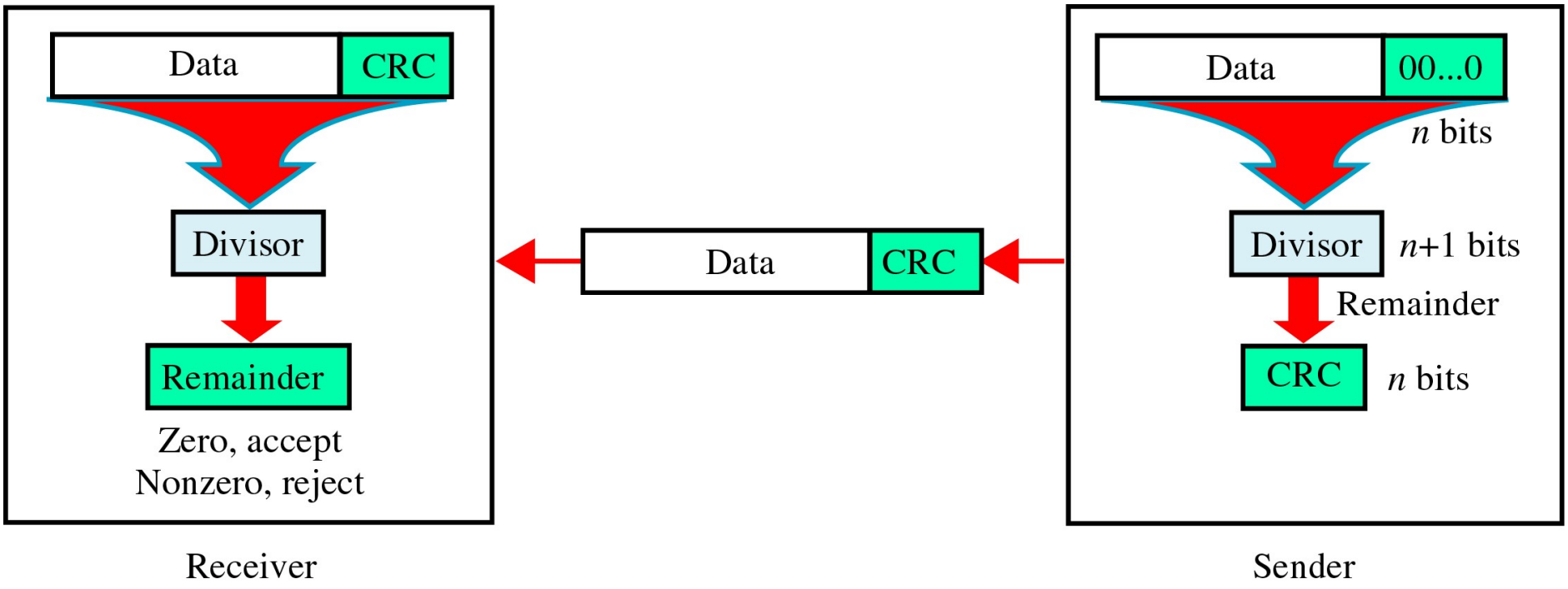
CRC(Cyclic Redundancy Check)

- ✓ 전체 블록검사
- ✓ 이진 나눗셈을 기반

- ✓ 계산 방법
  - ➢ 메시지는 하나의 긴 2진수로 간주
  - ➢ 특정한 이진 소수에 의해 나누어짐
  - ➢ 나머지는 송신되는 프레임에 첨부
    나머지를 BCC(Block Check Character)라고도 함
  - ➢ 프레임이 수신되면 수신기는 같은 제수(generator)를 사용하여 나눗셈의 나머지를 검사
  - ➢ 나머지가 0이 아니면 오류가 발생했음을 의미
  - ➢ 제수 : 생성 다항식(generator polynomial)

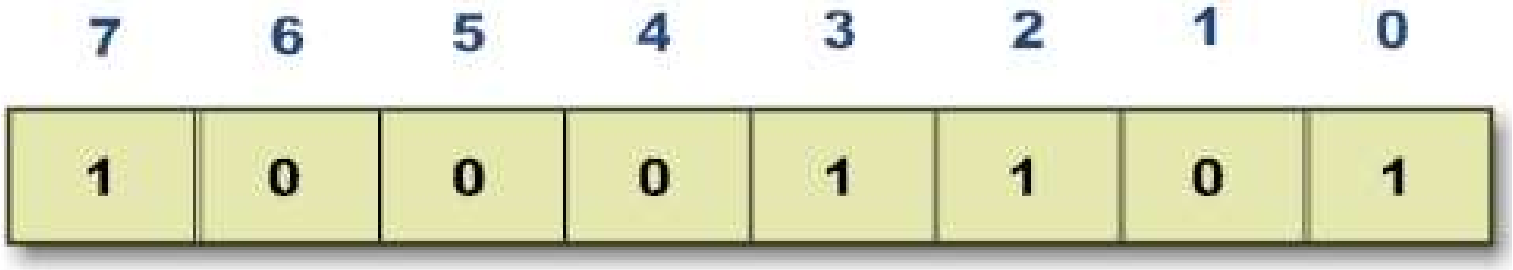
순환 중복 검사(CRC: Cyclic Redundancy Check)

- ✓ 2진 나눗셈을 이용



10.1 오류제어(8/43)

- ✓ 부호화 과정
  - ➢ 각 비트들의 값을 보면서 하나의 함수를 만드는 과정
  - ➢ 정보 비트를 전송비트의 다항식(polynomial)에 의한 표현으로 변환

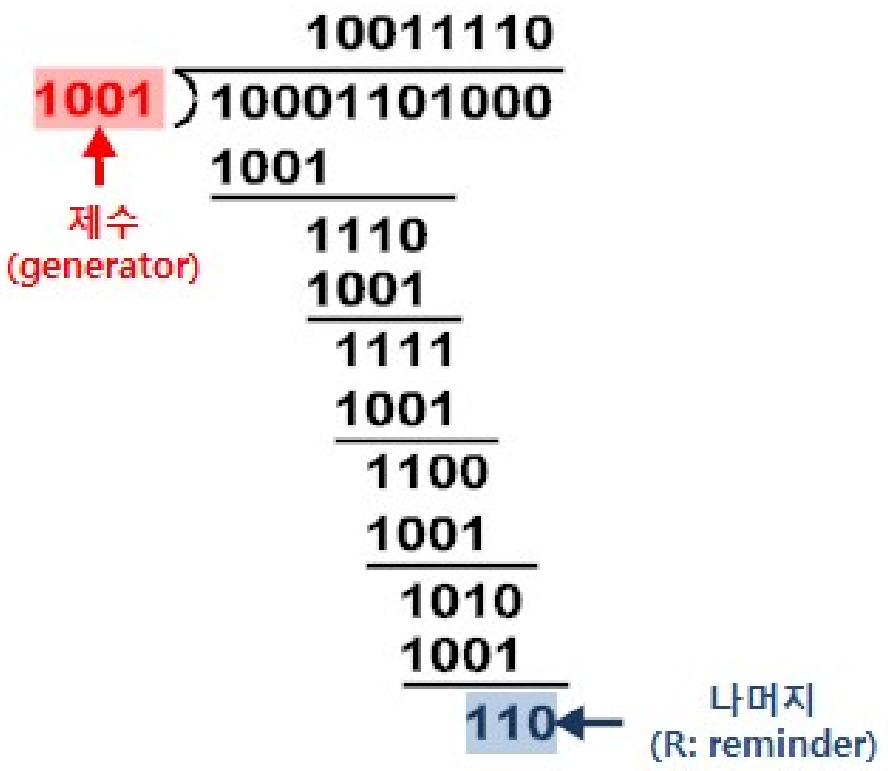


$$P(x) = x^7 + x^3 + x^2 + x^0 = x^7 + x^3 + x^2 + 1$$

10.1 오류제어(9/43)

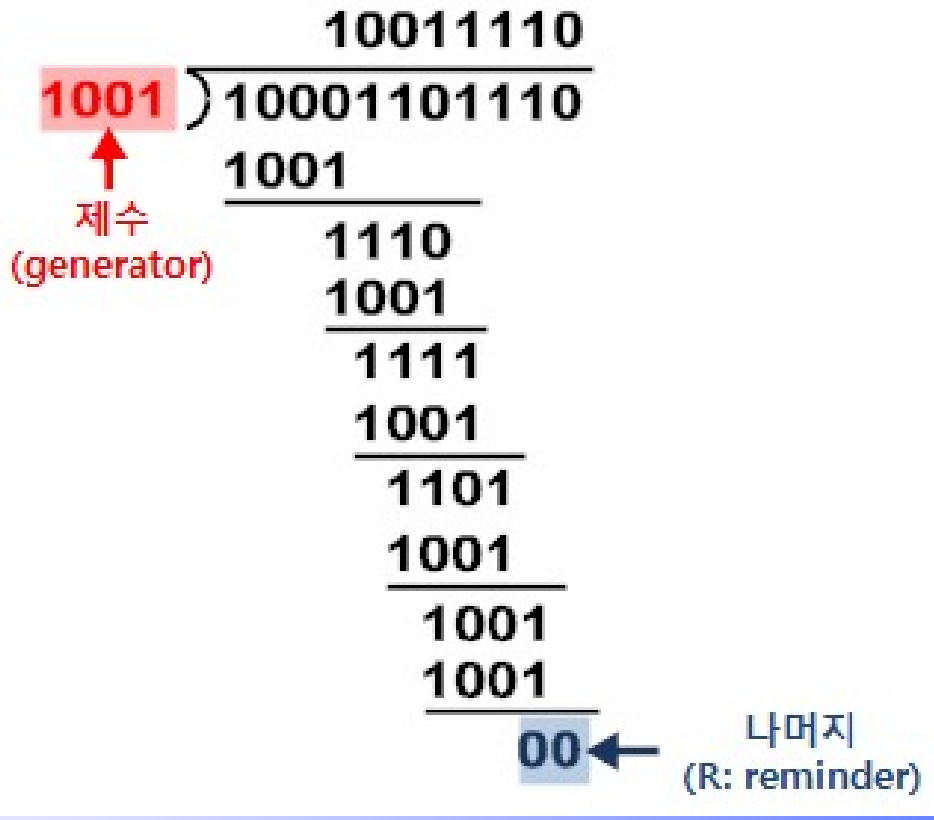
- ✓ CRC 비트의 생성
  - ➢ 캐리(Carry)가 없는 Modulo-2 연산
  - ➢ 전송 하고자 하는 데이터 : 10001101
  - ➢ 원하는 BCC 비트의 길이 : n
  - ➢ 제수의 길이 : n+1
  - ➢ 연산과정
    - 전송하고자 하는 데이터 뒤쪽에 n개의 0을 삽입
    - 제수로 나눔
    - 데이터 뒤의 n개의 0을 R로 대체
    - 전송

1111	1111
- 1001	⊕1001
0110	0110

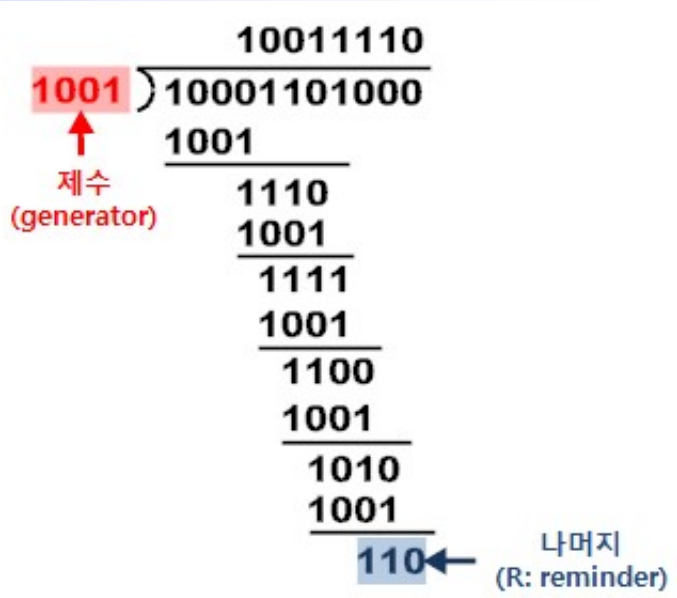


10.1 오류제어(10/43)

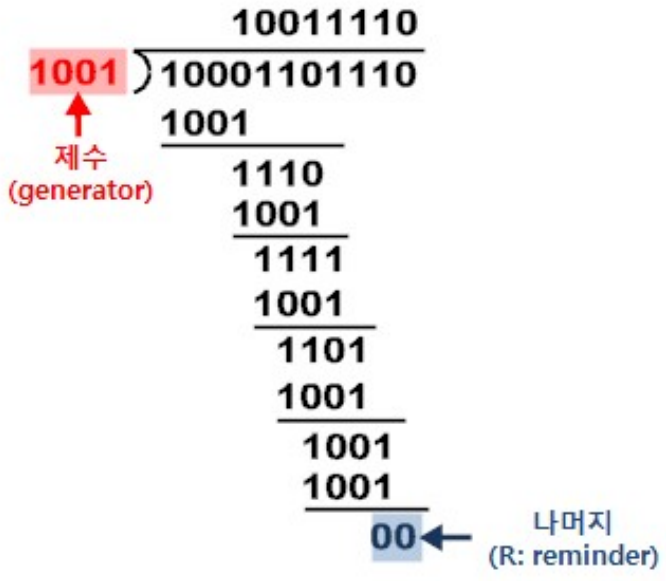
- ✓ 오류 검출 방법
  - ➢ 수신된 데이터를 송신측과 합의된 제수로 나눔
  - ➢ 연산결과 나머지가 0이면 오류 없음
  - ➢ 그렇지 않다면 재전송 요청



CRC 계산 수식



- 원래 데이터 : D(x) (k bits)
- 잉여 데이터 : F(x) (n-k bits)
- 전송 데이터 : T(x) (n bits)
- 미리 정의된 CRC 다항식 (Divisor) : P(x) (n-k+1 bits)
  - (n–k+1 bits)의 비트 패턴 : 생성 다항식
- P는 T를 나눌 수 있어야 함 (약수 즉, T/P의 나머지는 없음)



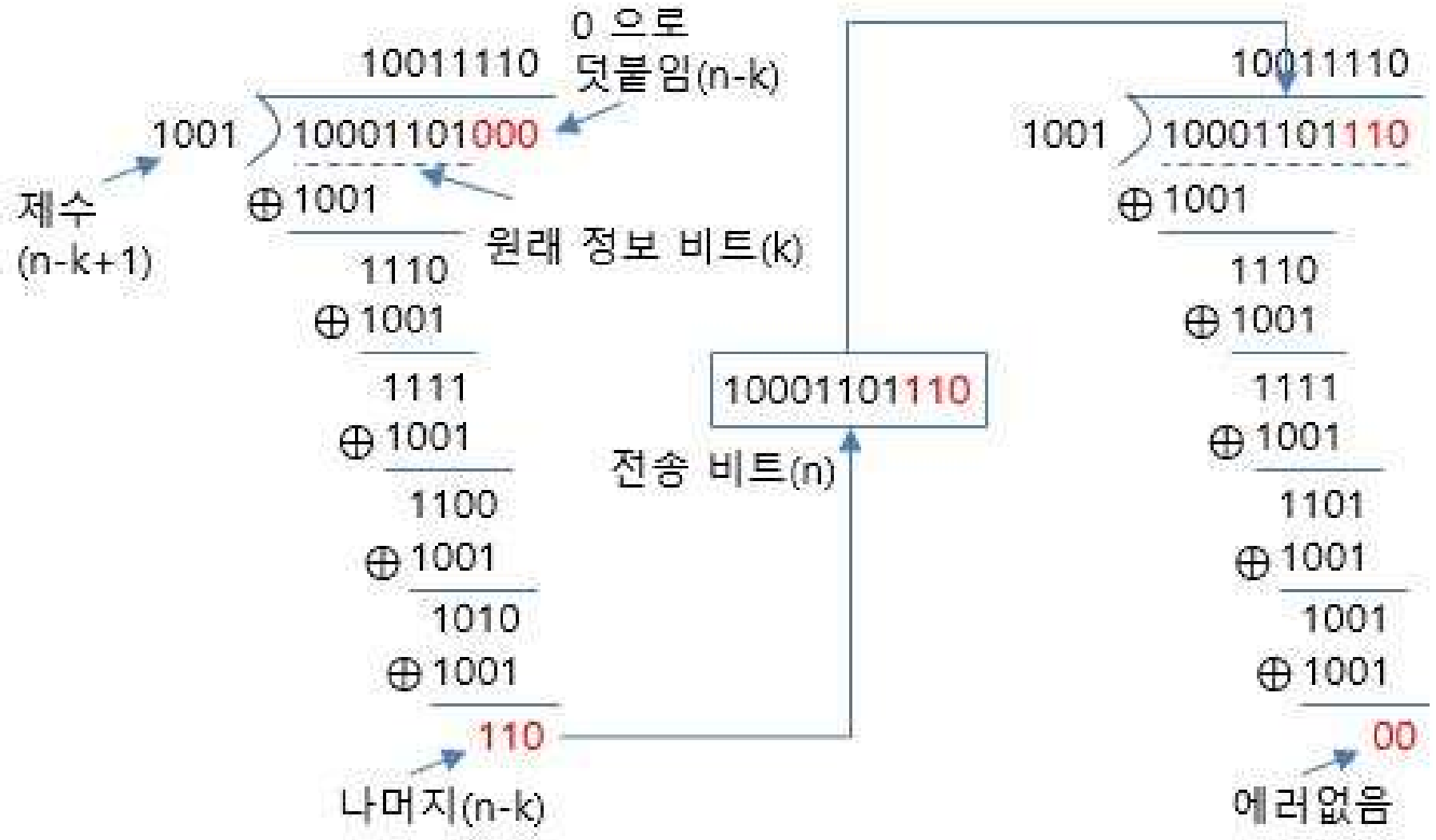
- 나눗셈 몫 (Quotient) : Q(x) (k bits) (버려짐)
- 나머지 (Reminder) : R(x) (n-k bits) (덧붙여짐)

CRC 계산 수식

CRC 계산을 위한 다항식표현

- ✓ 송신 데이터 $D'(x) = x^{n-k} D(x)$ (원래 데이터 : D(x) (k bits))
  - ➢ 원래 데이터에 생성 다항식의 가장 큰 차수를 곱함
  - ➢ 즉, 그만큼 0 값을 덧붙이는 것 (n bits)
- ✓ 생성다항식으로 나눔
  - ➢ $D'(x)/P(x) = x^{n-k}D(x)/P(x) = Q(x) + R(x)/P(x)$
  - ➢ $x^{n-k}D(x) = Q(x)P(x) + R(x)$

- ✓ $x^{n-k}D(x)$을 전송(T(x)) = $x^{n-k}D(x) + R(x)$
- ✓ $T(x) = x^{n-k}D(x) + R(x) = Q(x)P(x)$
  - ➢ 전송 비트 패턴 T(x)가 생성다항식 P(x)에 의해 정확히 나누어 떨어짐


0 으로
덧붙임(n-k)
10011110
1001) 10001101000
제수
(n-k+1)
⊕ 1001
원래 정보 비트(k)
1110
⊕ 1001
1111
⊕ 1001
1100
⊕ 1001
1010
⊕ 1001
110
나머지(n-k)
10001101110
전송 비트(n)
10011110
1001) 10001101110
⊕ 1001
1110
⊕ 1001
1111
⊕ 1001
1101
⊕ 1001
1001
⊕ 1001
00
에러없음

10.1 오류제어(11/43)

- ✓ 하드웨어로 구성된 CRC
  - ➢ 계산된 BCC(Block Check Character) 는 쉬프트레지스터 (Shift register)에 축적
  - ➢ 이 레지스터의 구성은 CRC 코드를 생성
  - ➢ 레지스터 중에 있는 각 구분은 생성다항식의 등급과 동일
  - ➢ Exclusive-OR요소들의 수도 또한 그 다항식과 관계되는 수
  - ➢ 종류
    - CRC-12
    - CRC-16
    - CRC-CCITT 등

10.1 오류제어(12/43)

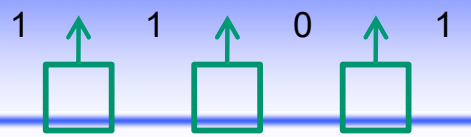
- CRC-12

  - 동기방식에서 사용
  - 6비트 캐릭터에 사용하며 이때의 BCC는 12비트
  - 생성다항식(제수)

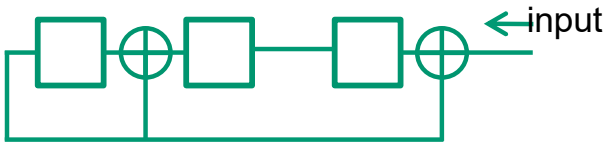
    : $x^{12}+x^{11}+x^3+x^2+x+1 = (x+1)(x^{11}+x^2+1)$

    $1100001000111_{(2)}$

  - 검출능력
    - 12비트보다 작거나 같은 집단오류 100%
    - 12비트보다 큰 집단오류의 경우는 99%이상의 확률



생성다항식(제수) : x^3+x^2+1, 정보 1100001의 경우



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

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

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

0 1 1 0 0 0 0 1 0 0 0

1 1 0 0 0 0 1 0 0 0

0 0 1 0 0 1 0 0 0

0 = 1 + 1 1 = 1 + 0

0 1 0 0 1 0 0 0

1 0 0 1 0 0 0

1 0 0 0 0 0

1 0 1 0 0

1 1 1 0

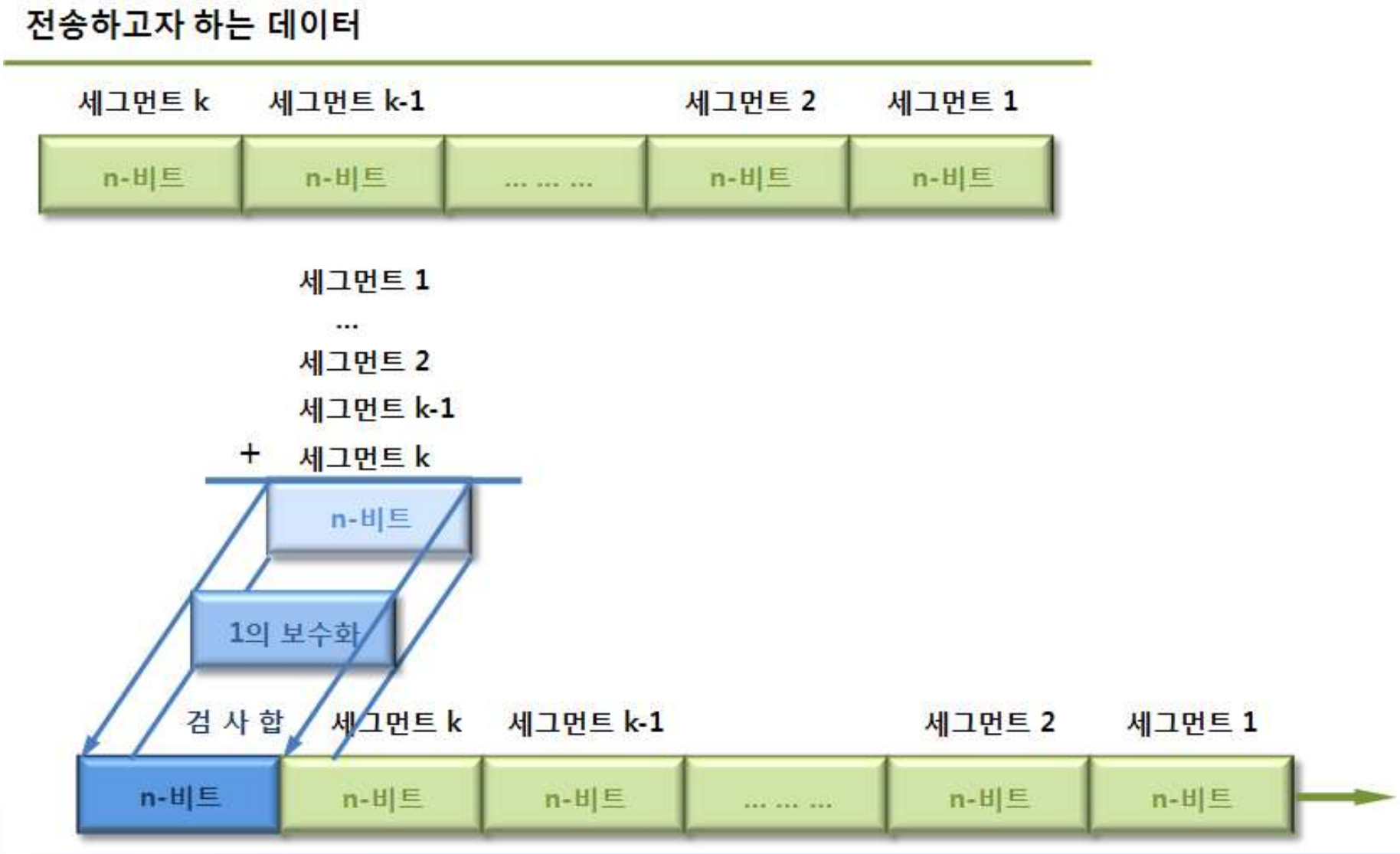
0 1 1

10.1 오류제어(19/43)

Checksum

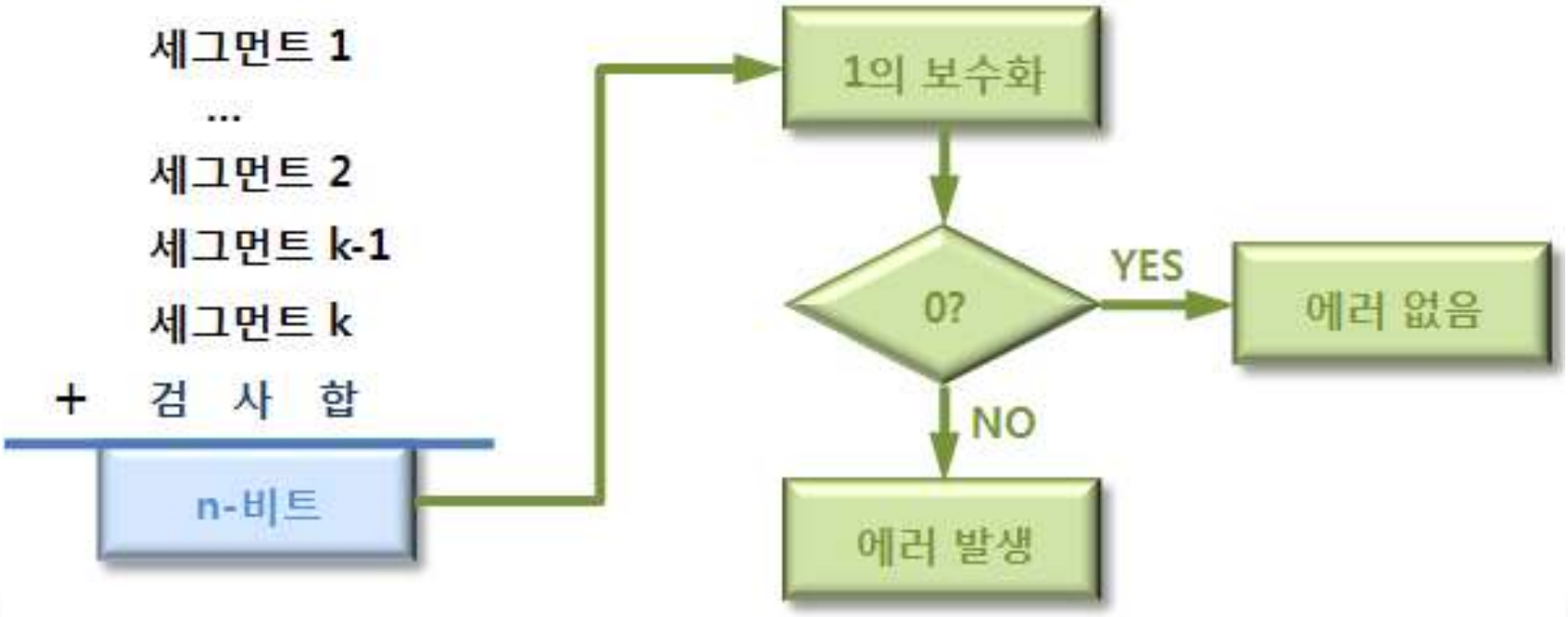
- ✓ 전송 데이터의 맨 마지막에 앞서 보낸 모든 데이터를 다 합한 합계를 보수화하여 전송
- ✓ 수신측에서는 모든 수를 합산하여 검사하는 방법
- ✓ 동작과정
  - ➢ 송신측
    - 데이터 단위를 n(보통은 16) 비트의 여러 세그먼트로 나눔
    - 이 세그먼트들은 전체 길이도 또한 n비트가 되도록 1의 보수 연산을 이용하여 합산
    - 전체 합은 보수화되고 원래 데이터 단위의 끝에 삽입
    - 이렇게 확장된 데이터 단위는 네트워크를 통해 전송

10.1 오류제어(20/43)



10.1 오류제어(21/43)

- 수신측
  - 검사 합을 포함하여 모든 세그먼트들을 더했을 때 모든 비트가 1이 나오지 않으면 오류
  - 송신측에서 검사 합을 최종 삽입할 때 1의 보수를 삽입하기 때문에 원래의 세그먼트들의 합을 더해주면 1이 되기 때문



10.1 오류제어(22/43)

- 송신측
  - 전송하고자 하는 데이터 : (11100101, 01100110, 11100110)

```
  11100101          01001011          10000011
+ 01100110        + 11100110        + 01100001
----------        ----------        ----------
  01001011          00110001          11100100
  10000011          1의 보수 값 = 11001110    10011010
```

  - 최종 데이터 (11100101, 01100110, 11100110, 10011010)

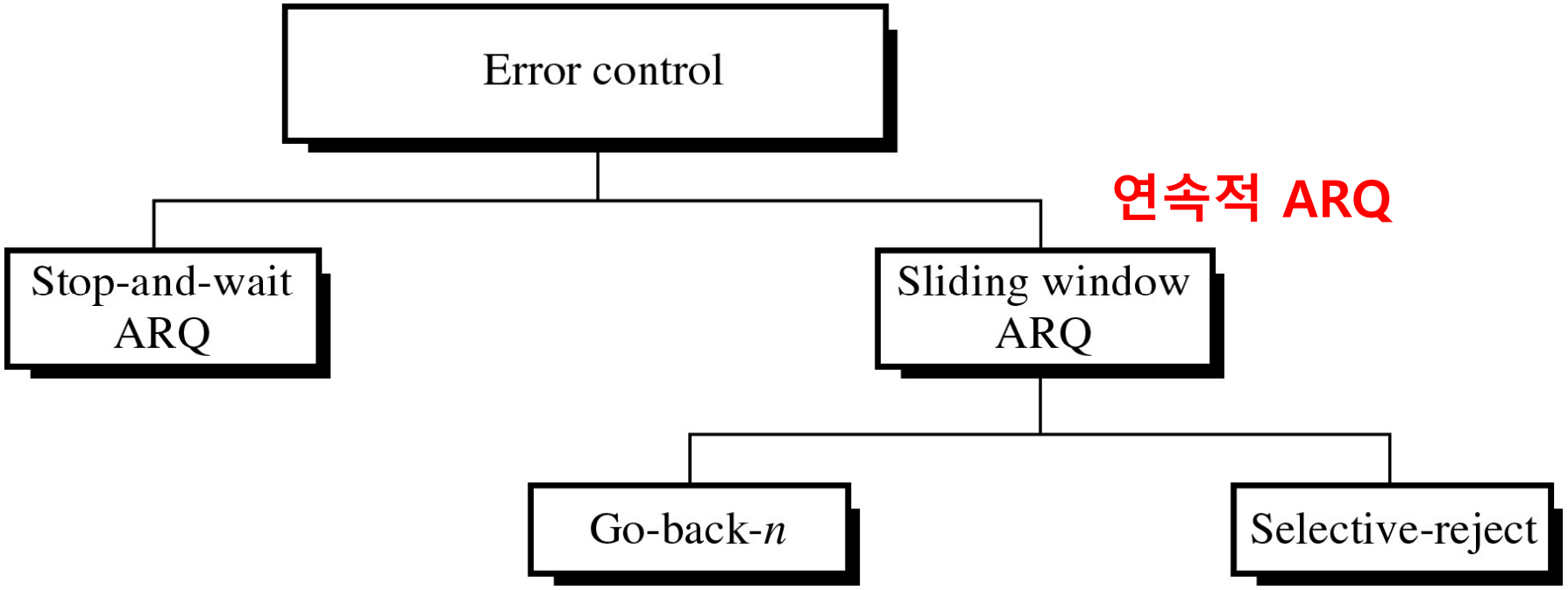
- 수신측
  - 수신된 데이터 : (11100101, 01100110, 11100110, 10011010)

```
  11100101          01001011          00110001
+ 01100110        + 11100110        + 11001110
----------        ----------        ----------
  01001011          00110001          11111111
```

  - 최종 합은 1이며, 오류 없음

10.1 오류 복구

오류 복구 분류



연속적 ARQ

ARQ : Automatic Repeat reQuest or Automatic Repeat-Query

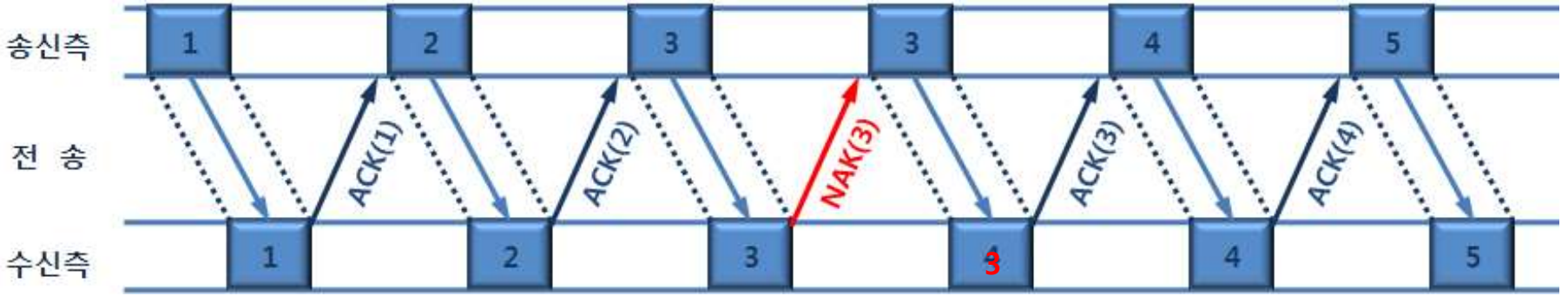
10.1 오류 복구(23/43)

오류 복구

- ✓ Stop-and-Wait ARQ
  - 송신측이 하나의 프레임을 전송
  - 수신측에서는 해당 프레임의 오류 유무를 판단
  - 오류가 없을 경우 송신측에게 ACK를 전송
  - 오류가 있는 경우 NAK를 전송하여 재전송 유도
  - 특징
    - 흐름제어 방식 중에 가장 간단한 형태
    - 한번에 한 개의 프레임만 전송
    - 한 개의 연속적인 블록이나 프레임으로 메시지를 전송할 때 효율적
    - 전송되는 프레임의 수가 한 개이므로 송신측이 기다리는 시간이 길어져 전송효율 저하
    - 송·수신측 간의 거리가 멀수록 각 프레임 사이에서 응답을 기다리는 데에 낭비되는 시간 때문에 효율 저하

10.1 오류 복구(24/43)

- 동작 과정
  - 송신측은 데이터 전송 후 ACK, NAK를 받을 때 까지 대기
  - 수신측은 수신 프레임에 대하여 오류검사를 수행
  - 오류가 없으면 ACK를 송신
  - 오류가 있으면 NAK를 송신
  - 송신측에서는 NAK를 수신할 경우 프레임을 재전송



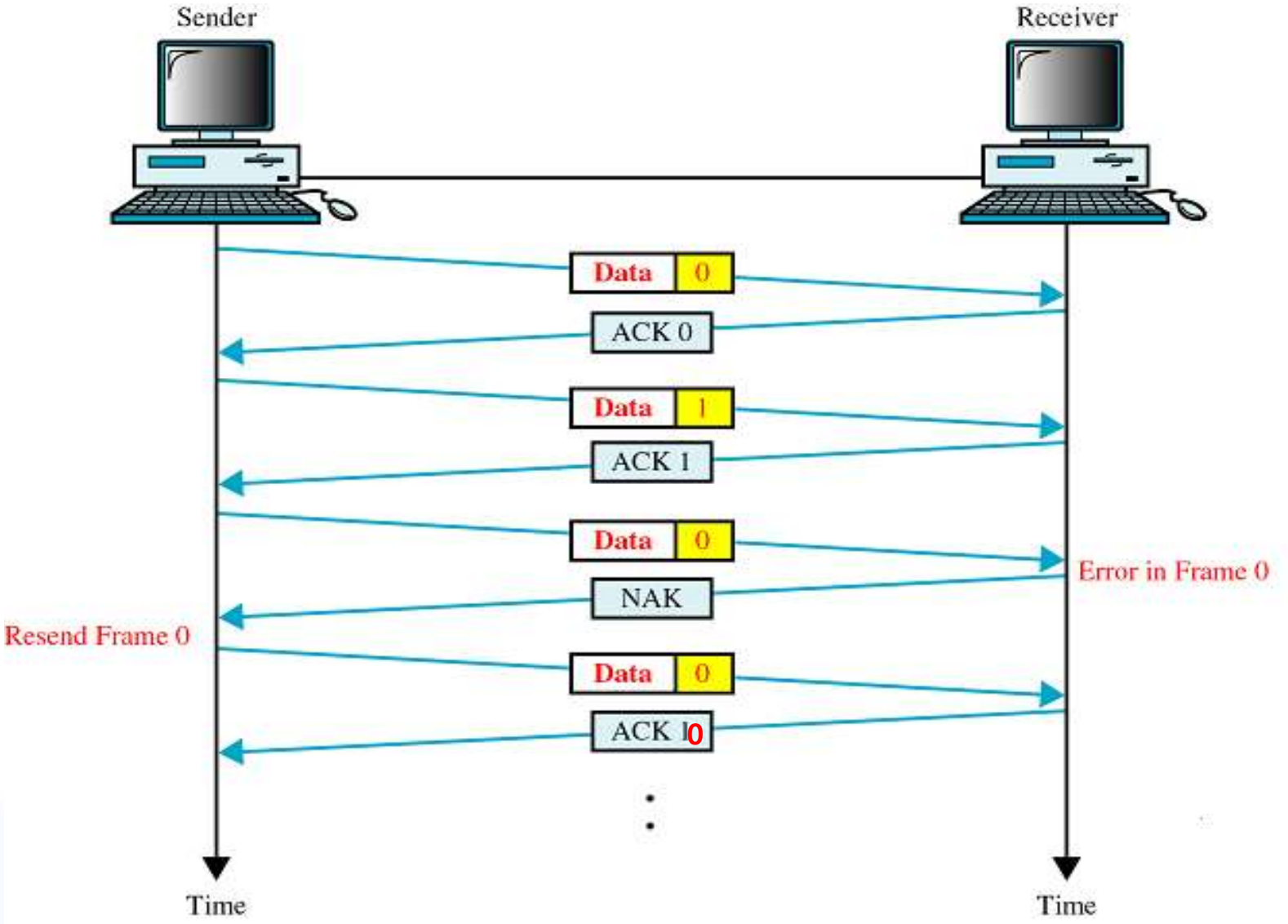
  - 3번 프레임의 전송 중 오류가 발생한 경우
  - 수신측은 NAK를 전송
  - 송신측은 NAK를 수신하면 3번 프레임을 재전송

정지/대기(Stop-and-Wait) ARQ

- ✓ 재전송을 위하여, 기본 흐름 제어 메커니즘에 4가지 특성이 추가
  - ➢ 송신측은 전송되어 분실된 프레임의 사본을 갖는다
  - ➢ 데이터 프레임과 ACK 프레임에 번갈아 0과 1을 부여한다
  - ➢ NAK 프레임(번호가 없는)
  - ➢ 타이머(송신측)

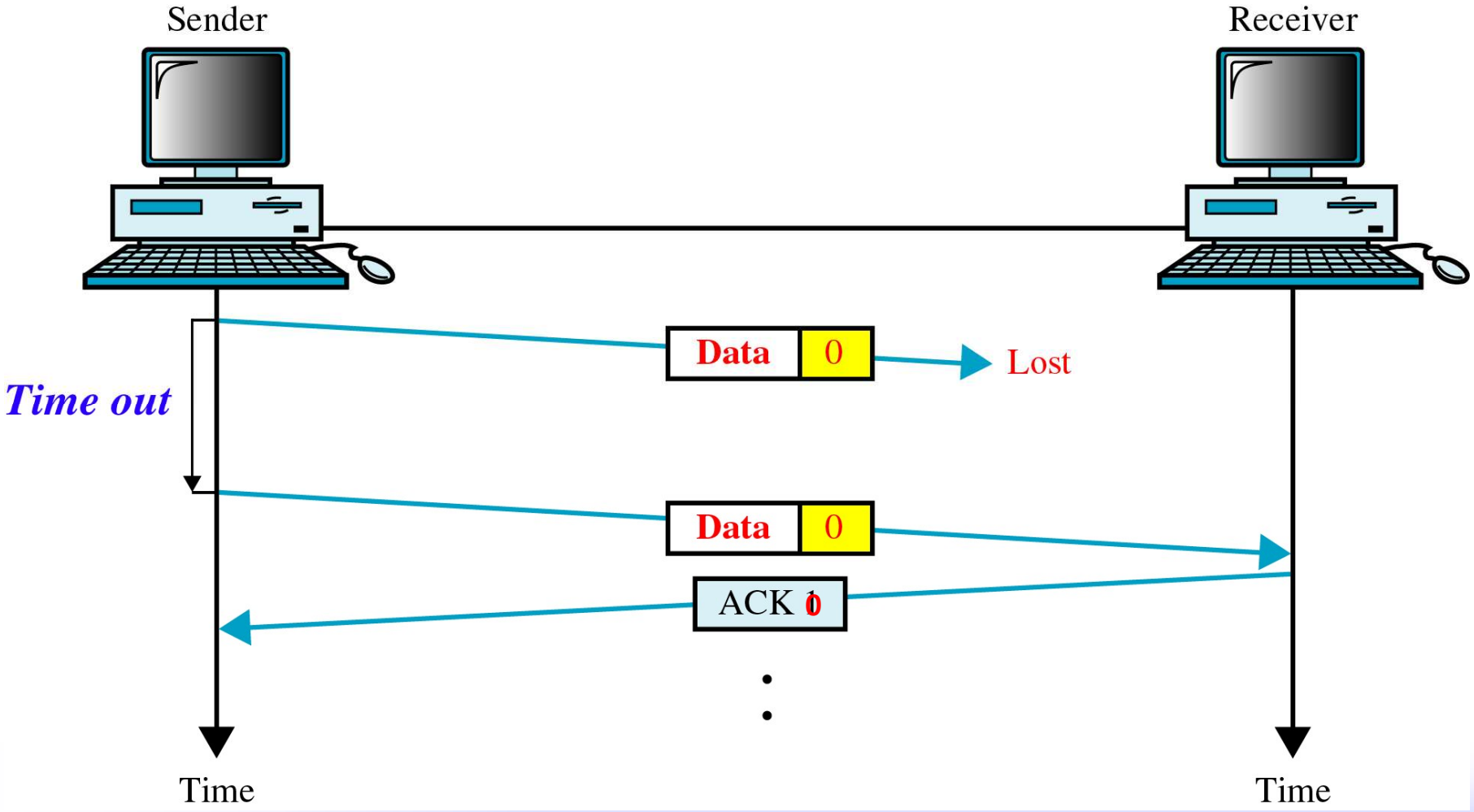
오류 제어(계속)

손상된 프레임



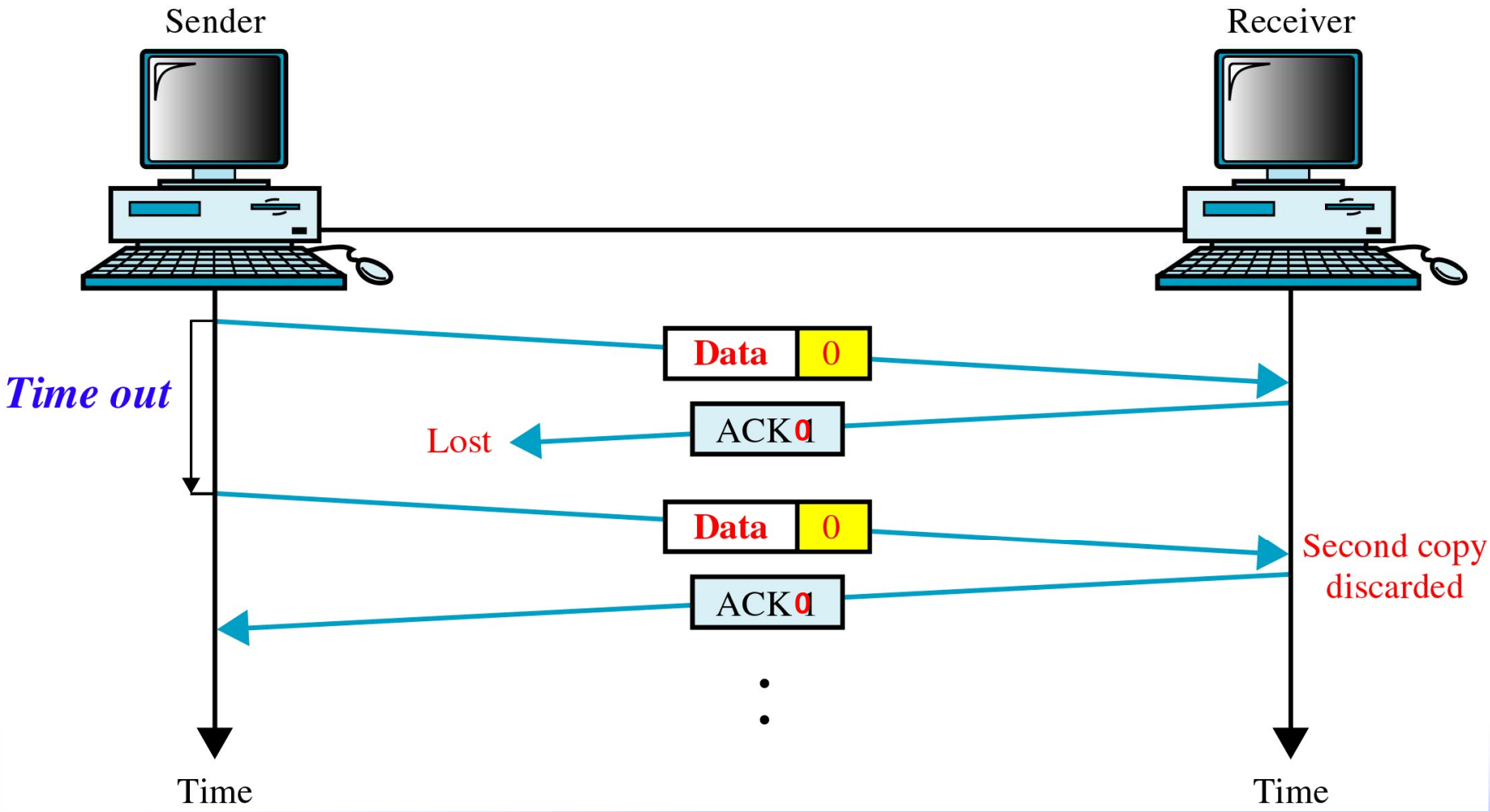
오류 제어(계속)

분실된 데이터 프레임

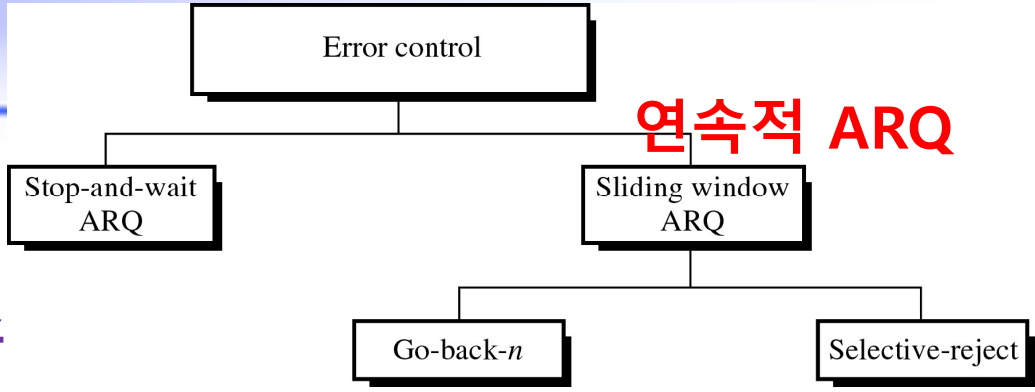


오류 제어(계속)

분실된 확인응답(Acknowledgment)

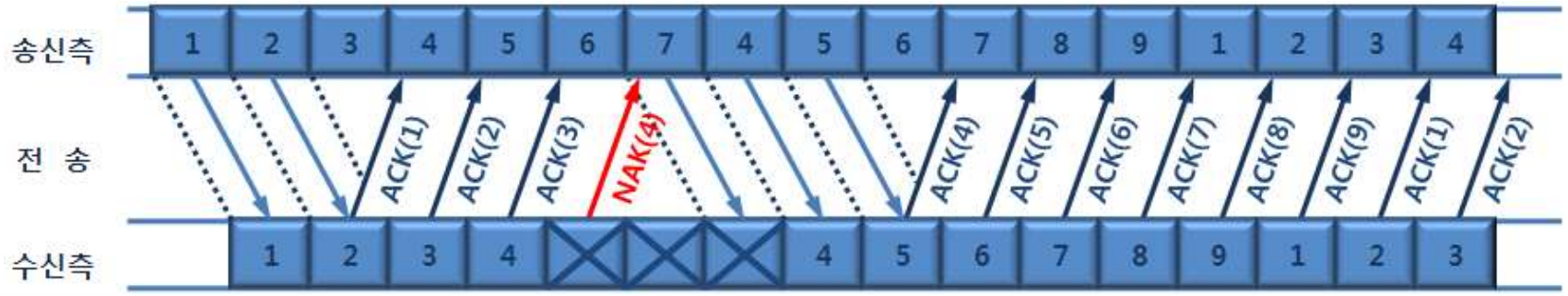


10.1 오류 복구(25/43)



연속적 ARQ

- ✓ Go-Back-N ARQ
  - ➢ 연속적 ARQ(continuous ARQ)
  - ➢ 수신응답 대기의 오버헤드 감소
  - ➢ 특징
    - 프레임의 수신은 순차적
    - 프레임에 순서번호를 삽입
    - 포괄적 수신확인
    - 오류 발생 프레임 부터 모두 재전송
  - ➢ 동작과정



- 수신측이 4번 프레임이 잘못되었음을 인지하고 NAK를 전송
- 송신측은 4번부터 모두 재전송

10.1 오류 복구(26/43)

- ✓ Selective Repeat
  - ➢ 연속적 ARQ
  - ➢ 오류가 발생한 프레임만 재전송
  - ➢ 특징
    - 송신측과 수신측은 동일한 크기의 Sliding-window를 보유
    - 수신측은 프레임의 순서에 상관없이 수신
    - 각각의 프레임에 대한 수신확인을 수행
  - ➢ 동작과정

- 오류가 검출된 4번 프레임에 대해서만 재전송 수행

10.1 오류 복구(27/43)

항목	Go-Back-N	Selective Repeat
Sliding-window	송신측만 갖고 있음 수신측은 수신버퍼 하나만 필요	송・수신측 모두 동일한 크기를 갖고 있음
수신확인	포괄적 수신확인 사용	개별적인 수신확인 사용
재요청방식	오류발생 또는 잃어버린 프레임 이후의 모든 프레임을 재요청 하거나 타임아웃으로 자동 재송신 됨	오류발생 또는 잃어버린 프레임에 대해서만 재요청 또는 타임아웃으로 인한 자동 재송신
프레임 수신 방법	프레임의 송신순서와 수신순서가 동일해야 수신	순서와 상관없이 윈도우 크기 만큼의 범위 내에서 자유롭게 수신
상위계층으로의 전달	수신 프레임이 순서적으로 들어올때 하나씩 상위계층으로 올려 보냄	순서에 상관없이 수신하여 일정수의 윈도우 만큼이 되면 전달
장단점	간단한 구현 적은 수신측 버퍼 사용량	구현이 복잡 버퍼사용량이 큼 보다 적은 재전송 대역폭

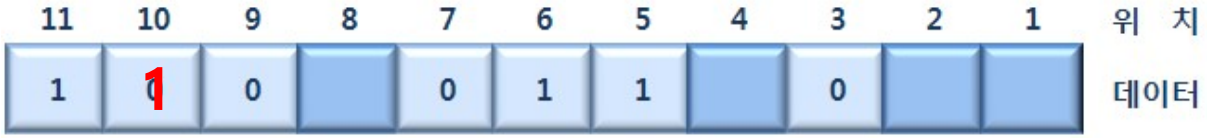
10.1 전진 오류 정정(28/43)

전진오류 정정(Forward Error Correction)

- ✓ 수신측에서 오류를 정정
- ✓ ARQ와 FEC
  - ➢ 재전송 요청하는 방법(ARQ) ; BEC
    - ▪ 구현이 단순
    - ▪ 재전송으로 인한 대역폭을 요구
  - ➢ 오류 정정 코드를 삽입, 수신측에서 직접 정정하는 방법(FEC)
    - ▪ 구현 복잡
    - ▪ FEC를 위한 별도의 코드 삽입으로 전송 시 대역폭을 요구
- ✓ cf) Backward Error Control

10.1 전진 오류 정정(29/43)

- ✓ 단일 비트 오류 정정(Single bit error correction)
  - ➢ 오류가 발생한 위치를 알아야 함
  - ➢ 7비트 데이터의 단일 비트 오류 정정을 위해서는 8가지의 상이한 상태를 구별할 패리티 비트가 필요
- ✓ 해밍코드
  - ➢ (11,7)code

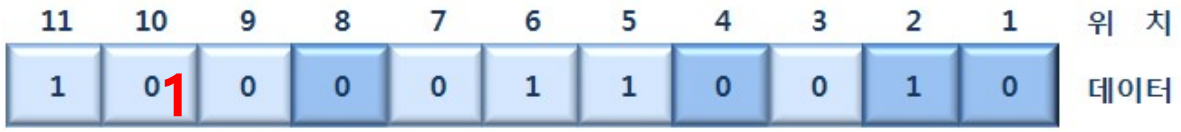


  - ➢ 연산방법
    - ▪ 1의 값을 가진 비트의 위치값을 이진수로 Exclusive-OR

Decimal	Binary
11	1011
10	1010
6	0110
5	⊕ 0101
Exclusive-OR	0010

10.1 전진 오류 정정(30/43)

- 최종형태(전송)



- 수신측 오류 검출 및 정정 방법
  - 오류가 없는 경우
    - 수신한 데이터내의 1의 값을 갖는 비트의 위치값을 계산

1011 ... 11							
1010 ... 10		0001					
0110 ... 6		0110		0111			
0101 ... 5		0101		0101		0010	
0010 ... 2	⊕	0010	⊕	0010	⊕	0010	
						0000	

※ 결과 값이 0000이므로 오류가 없다.

10.1 전진 오류 정정(31/43)

- 오류가 발생한 경우
  - 수신 프레임

위 치	11	10	9	8	7	6	5	4	3	2	1
데이터	1	01	0	0	0	1	1	0	0	1	0

위 치	11	10	9	8	7	6	5	4	3	2	1
데이터	0	1	0	0	0	1	1	0	0	1	0

에러

Decimal	Binary
10	1010
6	0110
5	0101
2	⊕ 0010
Exclusive-OR	1011

※ 결과 값이 1011이므로 11번째 비트에 오류가 있다.

  - 결과가 1011 이 나왔고, 십진수로 변환하면 11이라는 값이 나오므로 **11번째 비트에 오류 발생**
  - 오류정정 : 비트값이므로 0은 1로, 1은 0으로 변환

10.1 전진 오류 정정(32/43)

- 단일오류의 수정 및 이중오류의 검출이 가능

  n : 사용자 데이터의 크기 (위의 예에서는 7비트)
  k : 해밍코드의 크기 (위에서는 4비트)
  m : n-k, 잉여비트의 수

  $$2^m \geq n+1 = m+k+1$$

- n개의 가지 수 중에서 한 가지를 지정하는데 필요한 최소의 비트 수는 $\log_2 n$
- 오류가 없는 경우를 검출하기 위해 한 비트를 추가

  $$m \geq \log_2(n+1), \text{ 즉 } 2^m \geq n+1 = m+k+1$$

- 해밍코드 된 블록은 (n, n-m(k))으로 표현
- 해밍코드의 형태

  (15, 11) (511, 502) (4095, 4083) 등

- 블록사이즈와 효율은 비례
- 전송 중 3개 이상의 간헐적 오류에 의해 파괴될 가능성 수반

10.1 전진 오류 정정(33/43)

- ✓ 상승코드(Convolutional Code)
  - ➢ 길쌈코드(길쌈부호)
  - ➢ IS-95(CDMA)나 IMT-2000과 같은 이동통신에서 주로 사용
  - ➢ 응용 기술
    - ▪ 바이터비 코드(Viterbi code)
    - ▪ 터보 코드(Turbo code)
  - ➢ 현재의 입력이 과거의 입력에 대하여 영향을 받아 부호화되는 방법
  - ➢ 비블록화 코드
  - ➢ 구성 요소
    - ▪ 쉬프트 레지스터(shift register) : 정보를 암호화할 때 사용되는 일종의 기억장치
    - ▪ 생성다항식 : 쉬프트 레지스터와 결과 값을 연결할 때 사용(XOR)
  - ➢ 출력
    - ▪ 원시비트들을 쉬프트 레지스터에 통과시킴
    - ▪ modulo-2 가산기를 사용하여 전송비트 생성

10.1 오류제어(39/43)

	종류	특 징
오류 검출 기법	패리티 검사	- 한 블록의 데이터 끝에 한 비트를 추가하는 가장 간단한 방법 - 오류가 짝수 개 발생하게 되면 검출이 불가능
	블록 합 검사	- 각 비트를 가로와 세로로 두 번 관찰하여 데이터에 적용되는 검사의 복잡도를 증가시킴으로써 오류 검출능력 증대 - 하나의 데이터 단위 내에서 두 비트가 손상되고 다른 데이터 단위 내에서 정확히 같은 위치의 두 비트가 손상되면 블록 합 검사는 오 류를 검출하지 못함
	CRC	- 전체 블록 검사를 수행 - 이진 나눗셈을 기반으로 하므로 패리티 비트보다 효율적이고 오류검출 능력이 뛰어남
	Checksum	- 전송 데이터의 마지막에 앞서 보낸 모든 데이터를 다 합한 합계를 보수화하여 보냄 - 수신측에서는 모든 수를 합산하여 검사하는 방법
오류 정정 기법	단일 비트 오류 정정	- 오류를 정정하기 위해서는 오류위치를 파악해야 함 - 위치를 찾기 위한 패리티 비트를 추가
	해밍 코드	- 데이터와 패리티 비트간의 관계를 이용 - 각각 다른 데이터 비트들의 조합을 위한 패리티인 네 개의 패리티 비트 삽입
	상승 코드	- 현재 값과 과거 값 사이의 상관관계에 의한 값을 얻음 - 미리 약속된 디코딩 트리를 갖고 있어야 함 - 해밍거리를 이용하여 오류에 대한 신뢰성 보장

10.1 오류제어(40/43)

오류 복구 및 정정 기법의 응용

✓ FEC와 ARQ(BEC)

비 고	FEC	ARQ
장 점	- 수신측에서 오류를 정정 - 재전송을 하지 않아 적은 대역폭 사용	- 필드에 오류 정정 코드 불필요 - 간단한 구현 - 원시프레임의 크기에 CRC만 붙음
단 점	- 오류 정정 코드의 삽입으로 프레임 크기 증가 - 복잡한 구현	- 수신측이 자체 오류정정을 못함 - 재전송에 드는 대역폭 손실 큼

10.1 오류제어(41/43)

ARQ가 적절한 응용

- ✓ 주기적으로 인터럽트가 가능한 형식으로 전송하는 데이터의 전송응용
- ✓ FEC에 비해 경량
- ✓ 신속한 전송 가능
- ✓ 응용분야
  - ➢ 종이 혹은 자기테이프장치는 전송된 메시지의 사본을 만들 수 있어 ARQ에 적합
  - ➢ ARQ의 응용에서는 어떤 코드의 검출능력이 실질적으로 그것의 수정 능력보다 더 커야 한다. 즉 코드의 잉여도가 크지 않으면 ARQ기법이 FEC보다 유리
  - ➢ 오류가 집단적으로 사용되는 경우 ARQ는 데이터 자체에 대한 재전송을 요청하기 때문에 오류의 수와는 직접적인 상관관계가 없어 적절
  - ➢ 평균 비트 오류율이 현저하게 줄어든 현재의 네트워크에서는 경량의 ARQ가 FEC보다 적합
  - ➢ 간단한 패리티검사코드는 그 경제성으로 인하여 리얼타임 폴링에 사용

10.1 오류제어(42/43)

FEC가 적절한 응용

- ✓ 송수신측 사이에 인터럽트를 받지 않는 연속적인 형태로 데이터가 교환되는 응용에 적합
- ✓ 터미널이 버퍼가 없어도 됨
- ✓ 응용분야
  - ➢ 역 채널이 ARQ에 만족할 만한 사항이 아닐 경우
  - ➢ 채널의 전파지연시간이 너무 길어 Stop-and-Wait ARQ가 부적합한 경우에 적절
  - ➢ 4800bps이상 속도의 시분할 멀티플랙서들 사이에서의 전이중방식 전송은 데이터 전송과정에서의 인터럽트가 정상적으로는 허용될 수 없으므로 FEC의 사용이 적합

10.1 오류제어(43/43)

ARQ나 FEC 어느 것이나 이용될 수 있는 응용

- ✓ ARQ와 FEC의 설치에 필요한 모든 특성들이 만족되어야 함
- ✓ FEC는 특별한 채널이나 터미널기기의 특성에 대한 요구사항이 더 적음
- ✓ FEC를 선택하지 않는 주요한 이유는 기기에 소요되는 경비와 허용 가능한 오류율 등과 같은 사용자의 목적이나 또는 요구조건 때문
- ✓ ARQ가 선택되지 않는 주요 이유는 채널이나 터미널장비의 제약

10.2 흐름제어(1/9)

정의

- ✓ 송신기가 확인 응답을 기다리기 전에 보낼 수 있는 데이터의 양을 제한시키기 위해 사용되는 기법

목적

- ✓ 버퍼의 오버플로우(overflow)로 인한 데이터의 손실방지
- ✓ 네트워크 자원을 낭비하는 재전송 방지

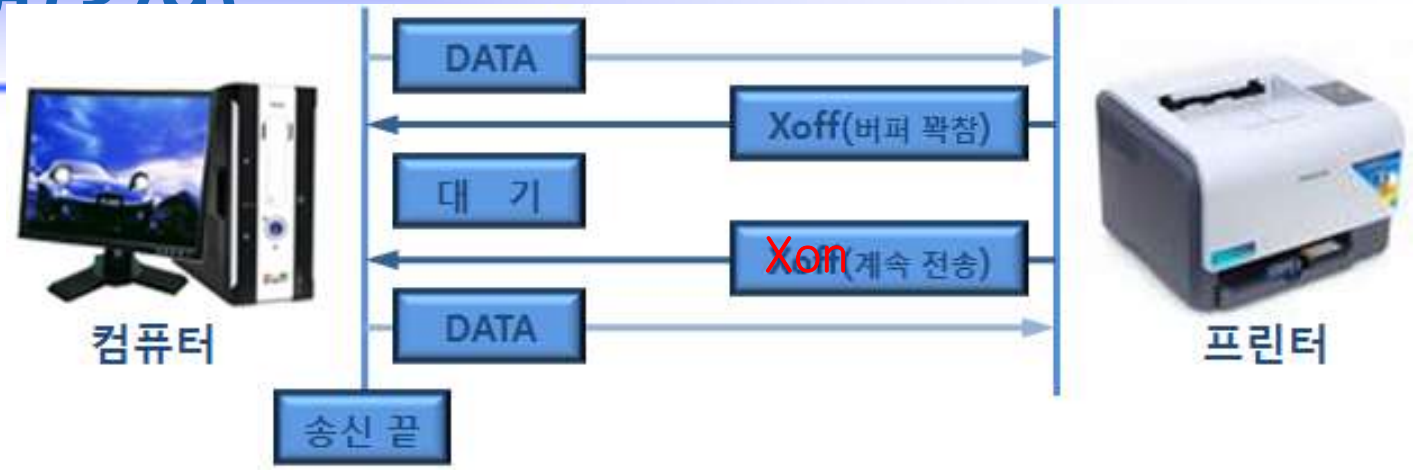
10.2 흐름제어(2/9)

Xon/Xoff

- ✓ 컴퓨터와 주변기기간의 비동기 통신 제어에 사용되는 프로토콜
- ✓ 모뎀에서 데이터의 흐름제어를 위해 사용
- ✓ 특징
  - ➢ 컴퓨터와 주변기기간의 상이한 전송속도로 인해 발생되는 통신상의 문제를 해결
  - ➢ 부호화된 문자로 비트통신에서 인식되지 않을 가능성
  - ➢ 수신측이 송신측의 데이터 송신을 제어

10.2 흐름제어(3/9)

동작과정



- ✓ 컴퓨터는 프린터에게 출력할 데이터를 전송
- ✓ 컴퓨터가 보내는 속도보다 프린터가 출력하는 속도가 느리기 때문에 프린터는 버퍼가 꽉 차게 되면 **Xoff** 를 보내어 컴퓨터의 송신을 잠시 멈춤
- ✓ 버퍼에 여유가 생기면 프린터는 다시 컴퓨터에게 송신을 하라는 **Xon**을 전송
- ✓ 컴퓨터는 데이터를 계속해서 송신
- ✓ 이러한 과정이 컴퓨터가 데이터를 모두 송신할 때 까지 계속
- ✓ "X"는 "transmitter"의 약자이므로, Xon 또는 Xoff 신호는 transmitter(송신장치)를 켜거나 끄기 위한 것
- ✓ Xon의 실제 신호는 아스키의 Ctrl-Q, Xoff는 Ctrl-S

10.2 흐름제어(4/9)

RTS/CTS

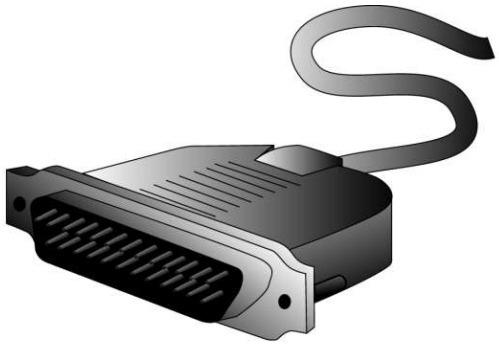
- RS-232C에 사용
- 모뎀을 이용한 통신에서 상이한 보오율을 조정
- 산업용 네트워크에서 사용

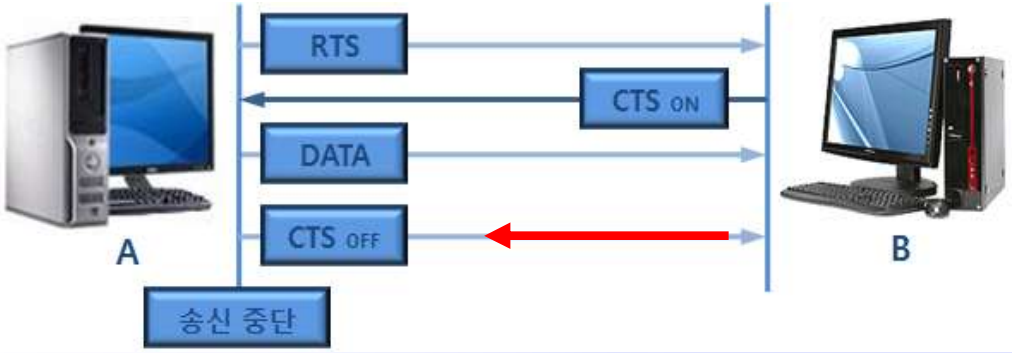
- 특징
  - 충돌방지를 위해 상호간 전송예비신호 전송
  - 특정 핀을 이용하여 원하는 신호 전달

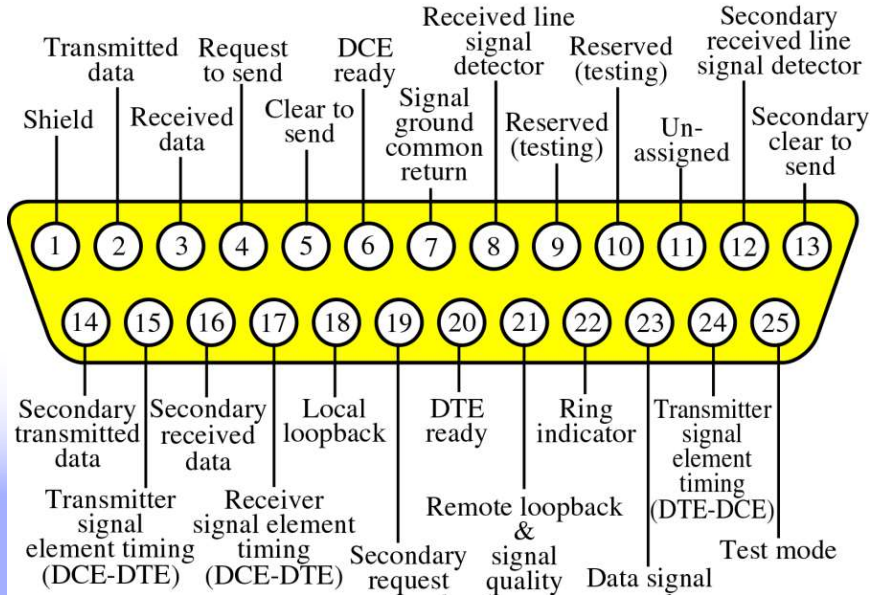
- RTS (Request to Send)와 CTS (Clear to Send)

10.2 흐름제어(5/9)

- 동작과정
  - A는 보내고자 하는 데이터가 있을 때 4번 핀을 이용하여 RTS신호 set(raising:1)
  - B는 이에 받을 준비가 되었다는 CTS를 5번 핀을 set(raising:1)
  - A는 응답을 받고, 2번 핀을 이용하여 실제 데이터를 전송
  - 이때 B측이 데이터를 더 이상 받지 않기를 원하면 5번 핀을 reset(lowering:0)
  - A는 이를 알아채고 데이터 송신 중단





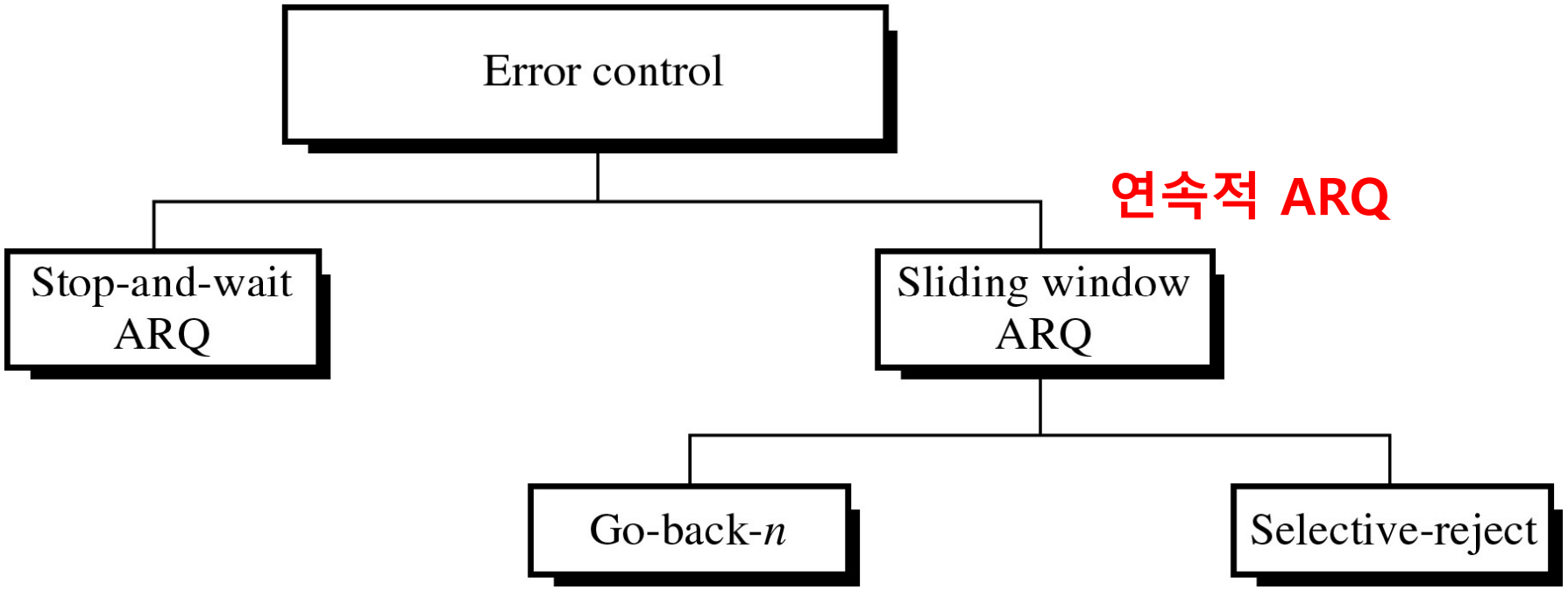


10.2 흐름제어(6/9)

✓ Xon/Xoff와 RTS/CTS의 비교

비 고	Xon/Xoff	RTS/CTS
공통점	데이터의 흐름을 제어 모뎀에서 송/수신 흐름 제어용으로 사용	
차이점	- 흐름제어를 문자형 프로토콜을 사용하여 제어 - 수신측이 주가 됨	- 흐름제어에 물리적 신호를 사용 - 송신측이 주가 됨

Sliding-window(연속적 ARQ)



연속적 ARQ

10.2 흐름제어(7/9)

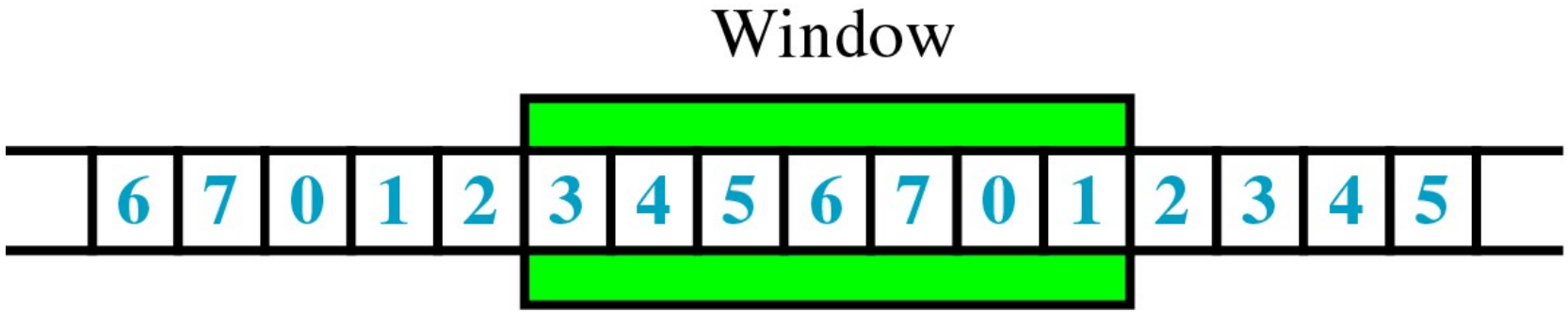
Sliding-window(연속적 ARQ)

- ✓ 송신측의 송신된 하나의 프레임과 수신측의 수신확인 프레임간의 1:1대응 방식을 탈피
  - ➢ 송수신측에 버퍼를 이용한 전송방식
  - ➢ 수신측의 응답방식은 포괄적 수신확인 허용

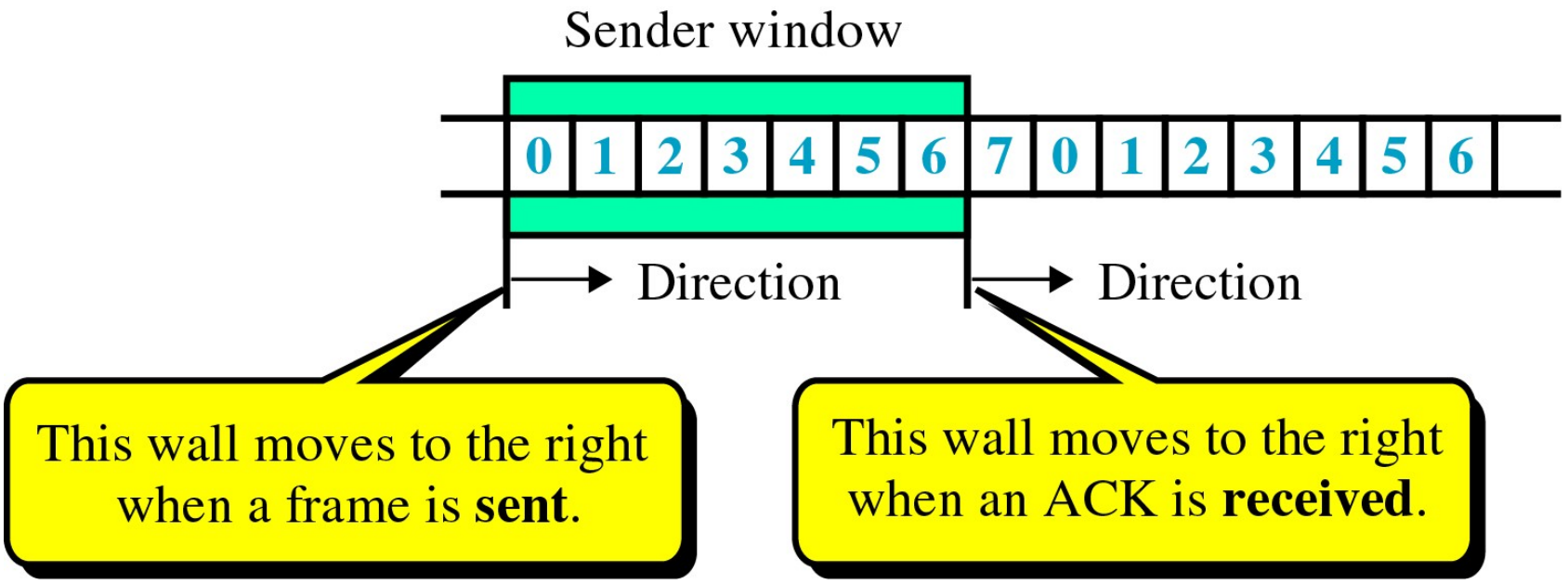
- ✓ 특징
  - ➢ 수신측으로부터 응답 메시지가 없더라도 미리 약속한 윈도우의 크기만큼의 데이터 프레임을 전송
  - ➢ 송수신 윈도우 사이즈 동일
  - ➢ 수신측에서는 확인 메시지를 이용하여 송신측 윈도우의 크기를 조절, 전송속도 제한

슬라이딩 윈도우(Sliding window)

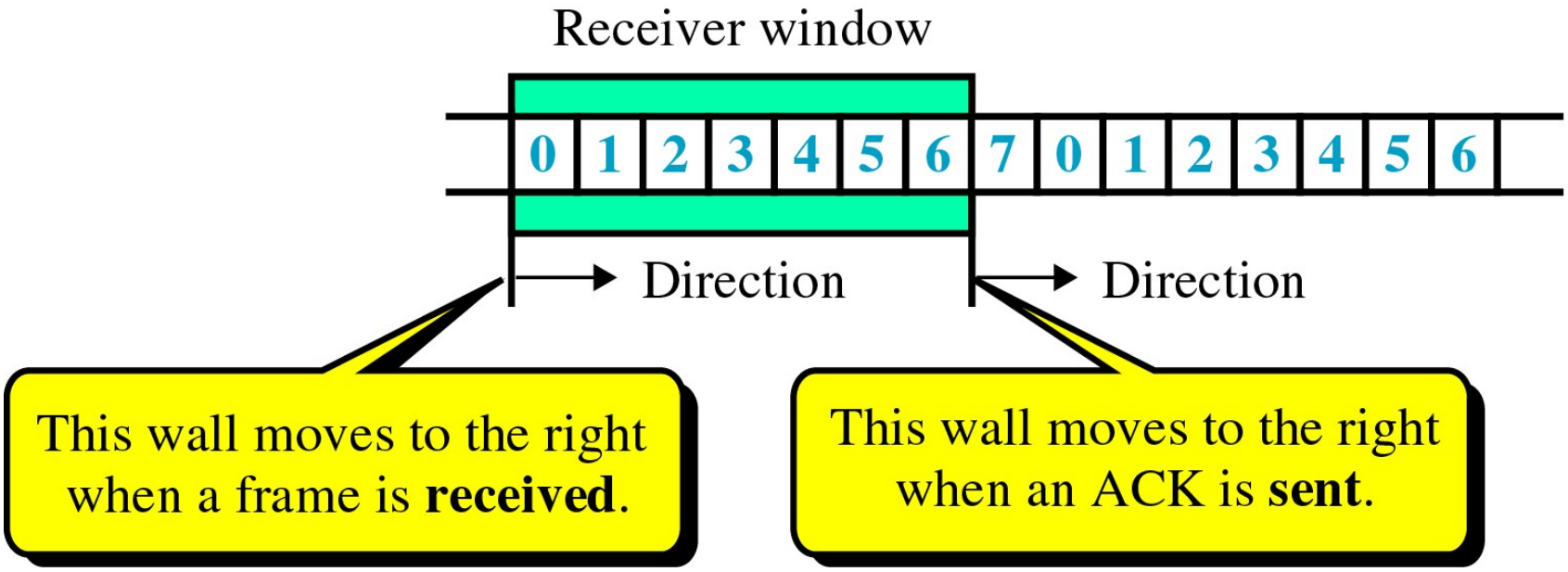
- ✓ 동시에 여러 개의 프레임을 전송할 수 있다



송신자 윈도우

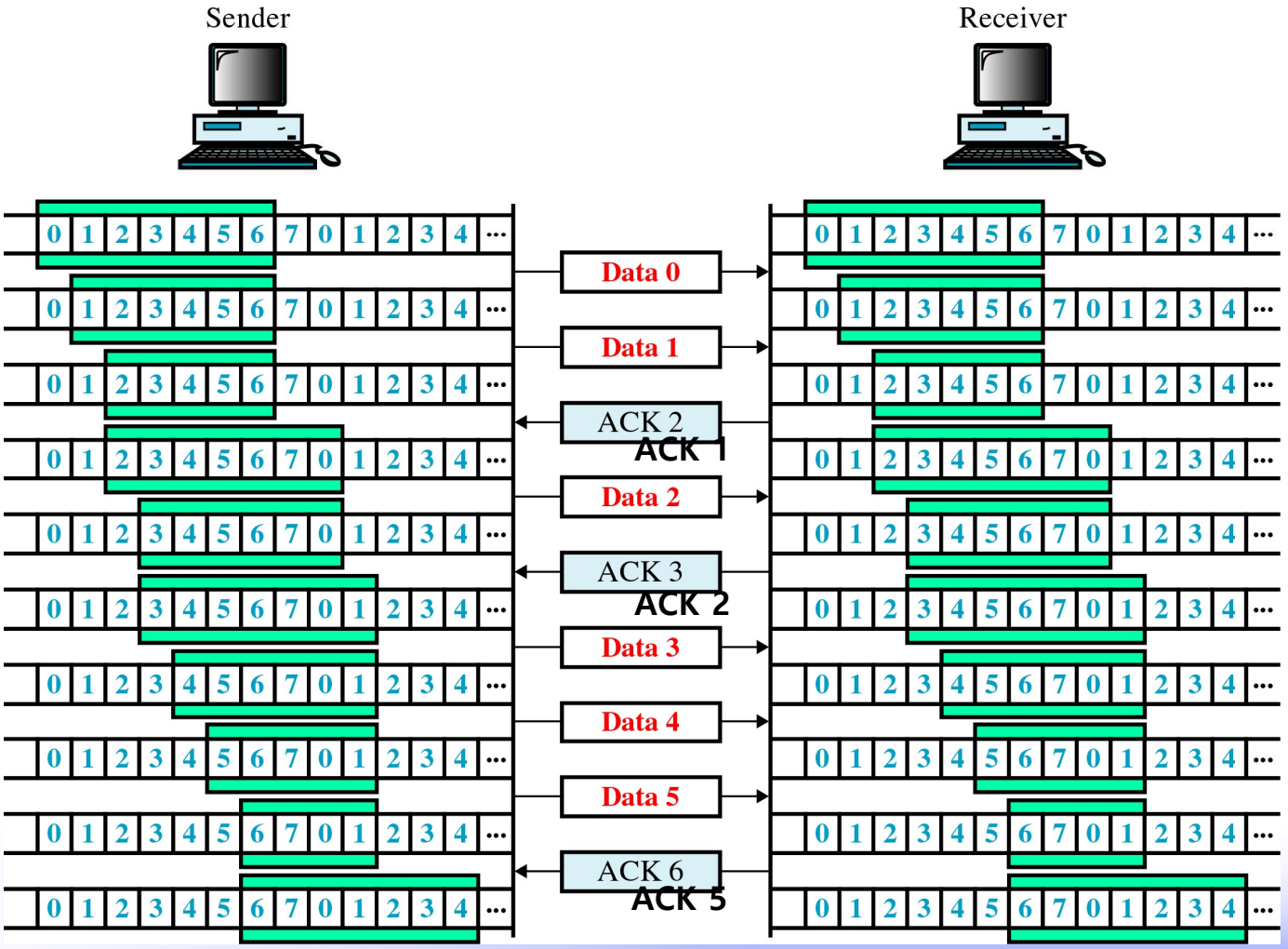

Sender window
0 1 2 3 4 5 6 7 0 1 2 3 4 5 6
Direction
Direction
This wall moves to the right when a frame is **sent**.
This wall moves to the right when an ACK is **received**.

수신자 윈도우


Receiver window
0 1 2 3 4 5 6 7 0 1 2 3 4 5 6
Direction
Direction
This wall moves to the right when a frame is **received**.
This wall moves to the right when an ACK is **sent**.

흐름 제어(계속)

예제

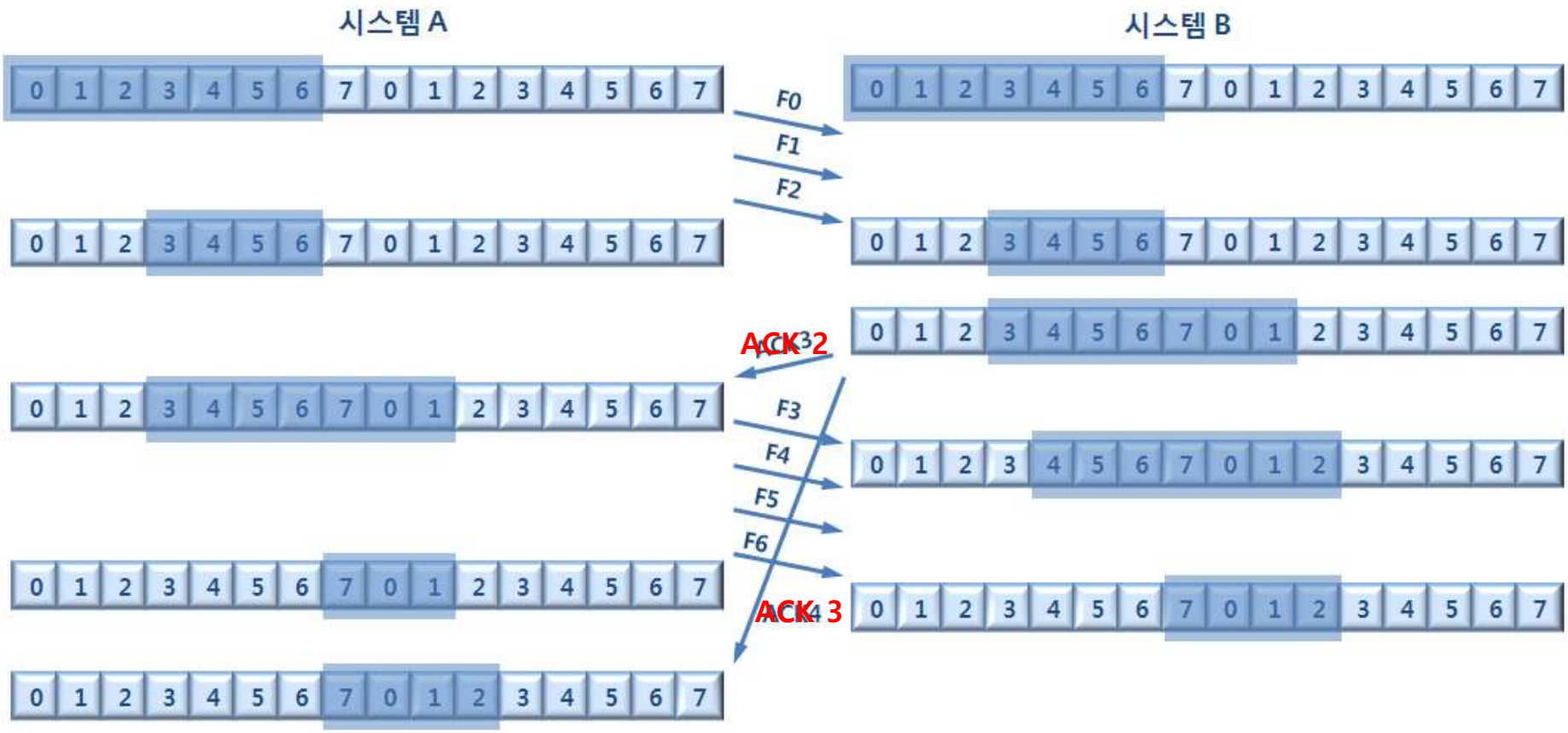

Sender
Receiver
Data 0
Data 1
ACK 2
ACK 1
Data 2
ACK 3
ACK 2
Data 3
Data 4
Data 5
ACK 6
ACK 5

10.2 흐름제어(8/9)

- 동작과정
  - 최대 윈도우 크기 : 7(0~7)
  - 빗금 친 부분
    - 송신측 : 전송 가능 범위
    - 수신측 : 수신 가능 범위

    - 처음에 시스템 A와 B는 프레임 0(F0으로 표현)에서 시작, 7개 프레임을 송신할수 있는 윈도우를 가짐
    - 시스템 A는 프레임(F0,F1,F2)을 전송
    - 윈도우 4개로 줄어듬
    - 시스템 B는 그러고 나서 ACK3(포괄적 수신확인)을 전송
    - 시스템 A는 프레임 3, 4, 5, 6의 전송을 진행
    - ACK4를 시스템 A에게 전송
    - 그러면 시스템A는 하나의 응답만을 받았으므로 윈도우를 하나 늘려 7번 프레임부터 2번 프레임까지 4개의 프레임을 전송할 수 있음

10.2 흐름제어(8/9)



10.2 흐름제어(9/9)

✓ Stop-and-Wait와 Sliding-window의 특징

종 류	특 징
Stop-and-Wait	- 송신측에서 각 프레임을 하나씩 보내고 수신측으로부터 확인 응답을 받는 방식 - 한번에 한 개의 프레임만 전송 - 송신측이 기다리는 시간이 길어져 전송효율이 낮음
Sliding-window	- 확인 응답 없이 한번에 약속된 윈도우 크기만큼 전송 - 한번에 윈도우 크기만큼 프레임 전송 - 여러 개의 프레임이 동시에 전송되므로 전송 효율이 높음