데이터통신과 네트워킹

Data Communication & Networking Ch. 6

동아대학교 컴퓨터AI 공학부
S03 301-01호실 (이양민 교수 연구실)
2024.10 (Kakao ID: yanwenry)

CHAPTER

06

데이터 링크 계층의 작업

Section

SECTION 01

프레임

1. 프레임 만들기

- 데이터 링크 계층data link layer의 역할은 물리 계층physical layer을 이용하여 LAN에 속해있는 노드들에게 데이터를 전송하는 것.
- 데이터 링크 계층은 네트워크 계층이 보내온 패킷에 데이터 링크 계층이 사용하는 헤더를 붙임. 이렇게 만들어진 데이터 단위를 **프레임**frame이라 부름.
- 데이터 전송방식은 예고 없이 데이터를 보내는 비동기식 전송과 데이터를 전송 한다고 먼저 알려준 후 프레임을 보내는 동기식 전송으로 나눌 수 있음. 현재는 동기식 전송 사용.

그림 6-1 비동기식 전송과 동기식 전송

프레임

- 동기식 전송에서는 데이터를 보내기 전에 통신시작을 알리는 신호가 **플래그**flag
- 프레임을 시작할 때와 마찬가지로 프레임이 언제 끝날지를 알려주어야 함.
- 앞에 붙이는 플래그를 **프리앰블**preamble이라 부르고 뒤에 붙이는 플래그를 **포스트앰블**postamble이라 부름.

그림 6-2 프리앰블과 포스트앰블

프레임

- 프레임 내에 플래그와 같은 패턴을 가진 데이터가 존재하는 경우 문제가 발생할 수 있음.
- 플래그와 같은 패턴이 있을 경우, 그 앞에 이스케이프 문자(ESC)를 삽입하여 전송.
- 프레임 안에 'ESC + 플래그' 패턴이 들어 있는 경우, 'ESC + 플래그' 앞에 ESC를 하나 더 붙여서 보냄.

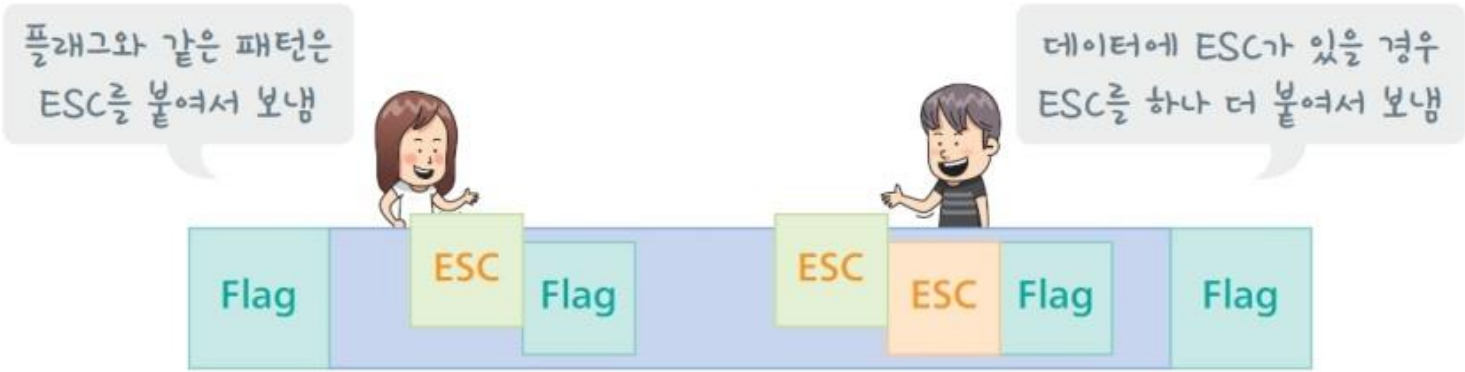


그림 6-3 프레임 만들기

프레임

2. 문자 프레임과 비트 프레임

- 문자 프레임에서는 데이터를 아스키 코드 형태로 전송.
- 문자 프레임에서 프리앰블은 DLE STX를 붙여서 사용하고 포스트앰블로 DLE ETX를 사용.
- STX는 전송 텍스트의 시작을 나타내는 제어문자이며, ETX는 전송 텍스트의 끝을 나타내는 제어문자.
- 데이터에 DLE ETX가 나타나는 경우에는 DLE ETX 앞에 DLE를 하나 더 붙여서 전송 -> **문자 스터핑**charater stuffing
- 문자 프레임은 현재 거의 사용하지 않음.

그림 6-4 문자 프레임과 문자 스터핑

프레임

- 대부분은 비트 프레임 방식을 사용.
- 비트 프레임에서는 프리앰블과 포스트앰블에 비트 패턴을 사용 -> 프리앰블과 포스트앰블 패턴은 똑같이 01111110 임. 0이 나온 후 1이 6개에 연달아 나타난 뒤, 맨 뒤에 0이 붙어서 총 8비트.
- 데이터에 플래그와 같은 패턴, 즉 01111110이 나타나면 문제가 발생.
- 연달아 나타나는 1의 다섯 번째 다음에 0을 하나 삽입 -> **비트 스터핑**bit stuffing

그림 6-5 비트 프레임과 비트 스터핑

SECTION 02

슬라이딩 윈도우 프로토콜

1. 데이터 전송 오류

- 송신 A가 보낸 프레임이 사라지는 문제가 발생 -> 송신 A는 처음 보낸 프레임이 사라졌다는 사실을 모른 상태에서 두 번째 프레임을 보냄.
- 수신 B의 입장에서는 처음 보낸 프레임이 사라진 것을 알 수 없음 -> 따라서 송신 A가 두 번째로 보낸 프레임을 첫 번째 프레임이라 착각하는 문제 발생.

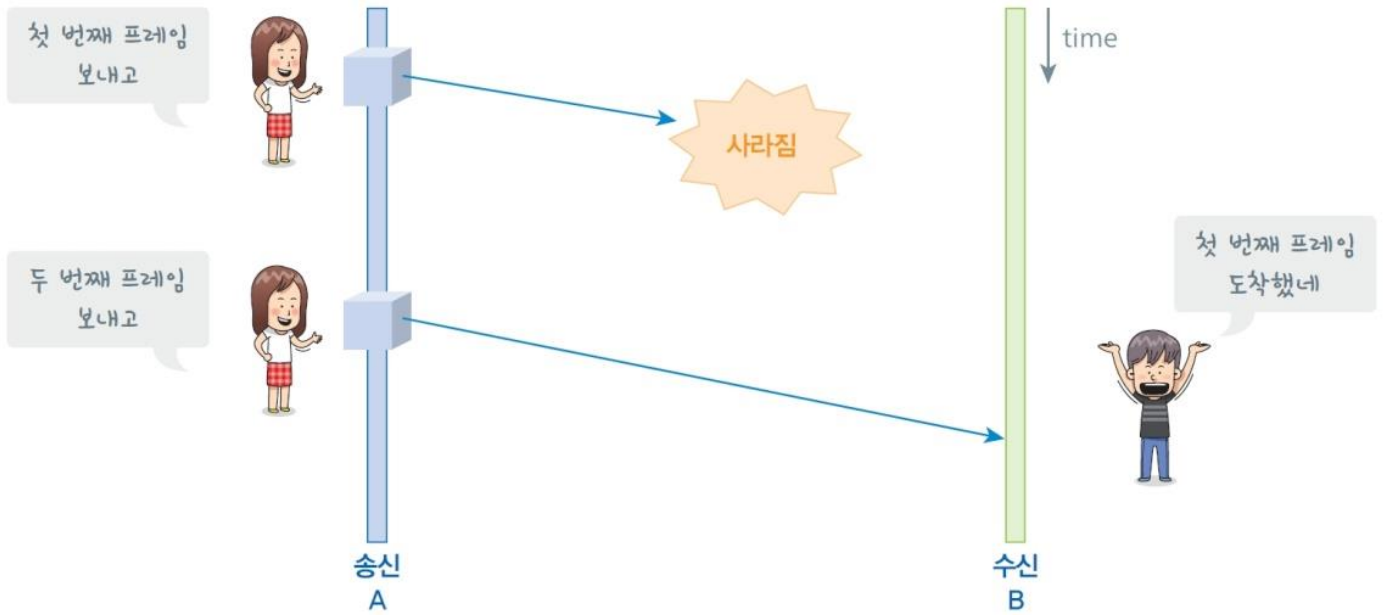


그림 6-6 오류가 있는 네트워크에서 프레임이 사라지는 문제

SECTION 02

슬라이딩 윈도우 프로토콜

- 에러가 있는 네트워크에서 데이터를 보내는 것은 마치 높은 벽 뒤로 물건을 던지는 것과 같음 -> 프레임을 보낸 쪽은 제대로 받았는지 확인 할 수 없음.
- 이러한 문제를 해결하는 방법은 물건을 받을 때 마다 '액(ACK)'이라고 소리치면 됨. 보내는 쪽에서는 '액(ACK)' 소리를 듣고 난 후에 다음 프레임을 보냄.

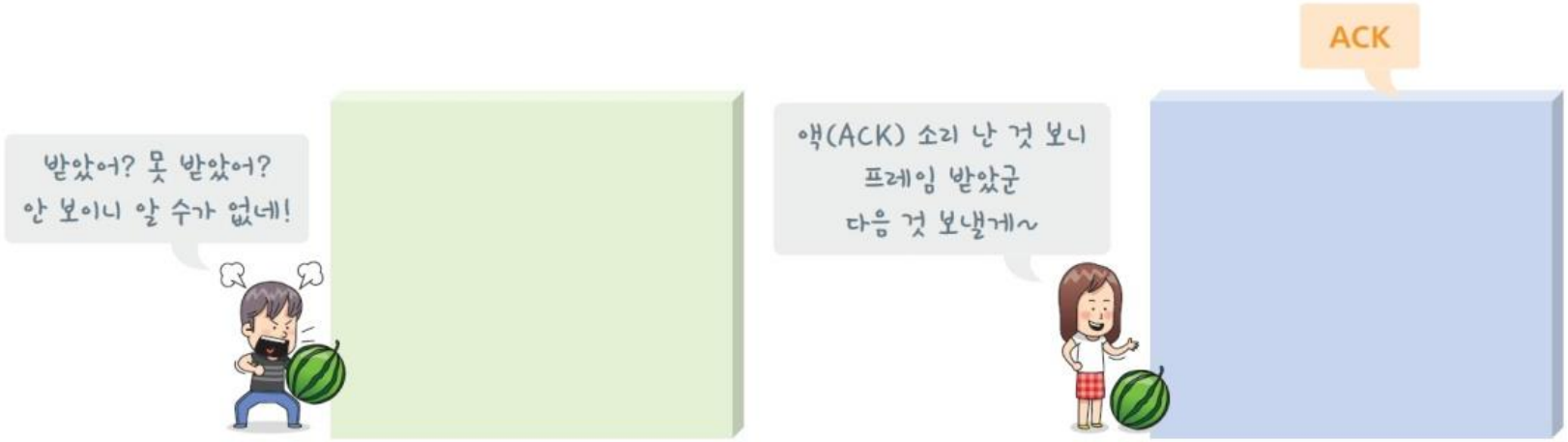


그림 6-7 액(ACK) 사용

슬라이딩 윈도우 프로토콜

- 수신 B가 프레임을 받은 후 ACK를 보냈는데 이 ACK가 사라졌음. 송신 A는 ACK를 받아야만 다음 프레임을 보낼 수 있음. 그런데 ACK가 사라졌기 때문에 송신 A는 새로운 프레임을 보낼 수 없음.
- 수신 B도 ACK를 보낸 후 다음 프레임을 하염없이 기다리게 됨. 결국 송신 A와 수신 B는 무작정 기다리기만 한다 -> 양쪽 모두 **타임아웃**timeout 사용.

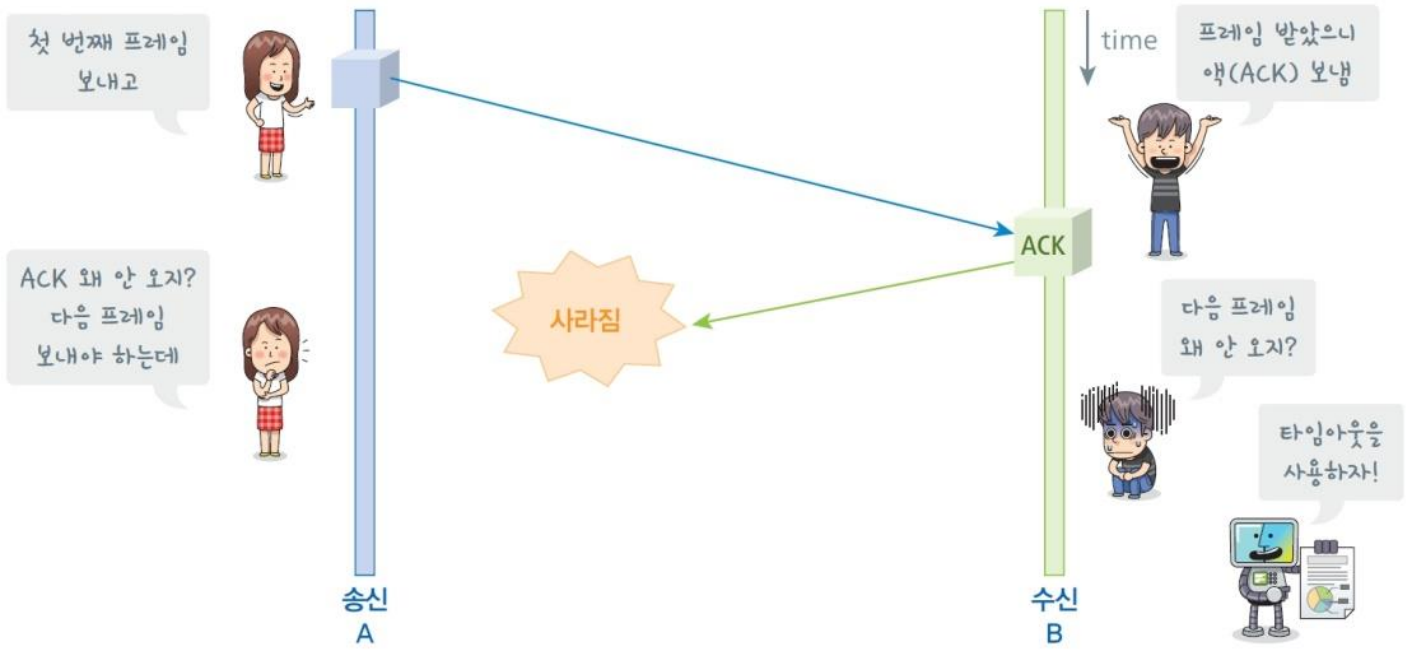


그림 6-8 ACK가 사라지는 경우

슬라이딩 윈도우 프로토콜

- 송신 A가 타임아웃이 걸려 같은 프레임을 2번 보낸 이후에 ACK가 도착.
- 수신 B는 ACK를 보낸 이후에 도착하는 프레임을 정상적인 프레임으로 생각하게 됨 -> 수신 B는 같은 프레임을 2번 받았지만, 서로 다른 프레임이라 착각하게 됨.

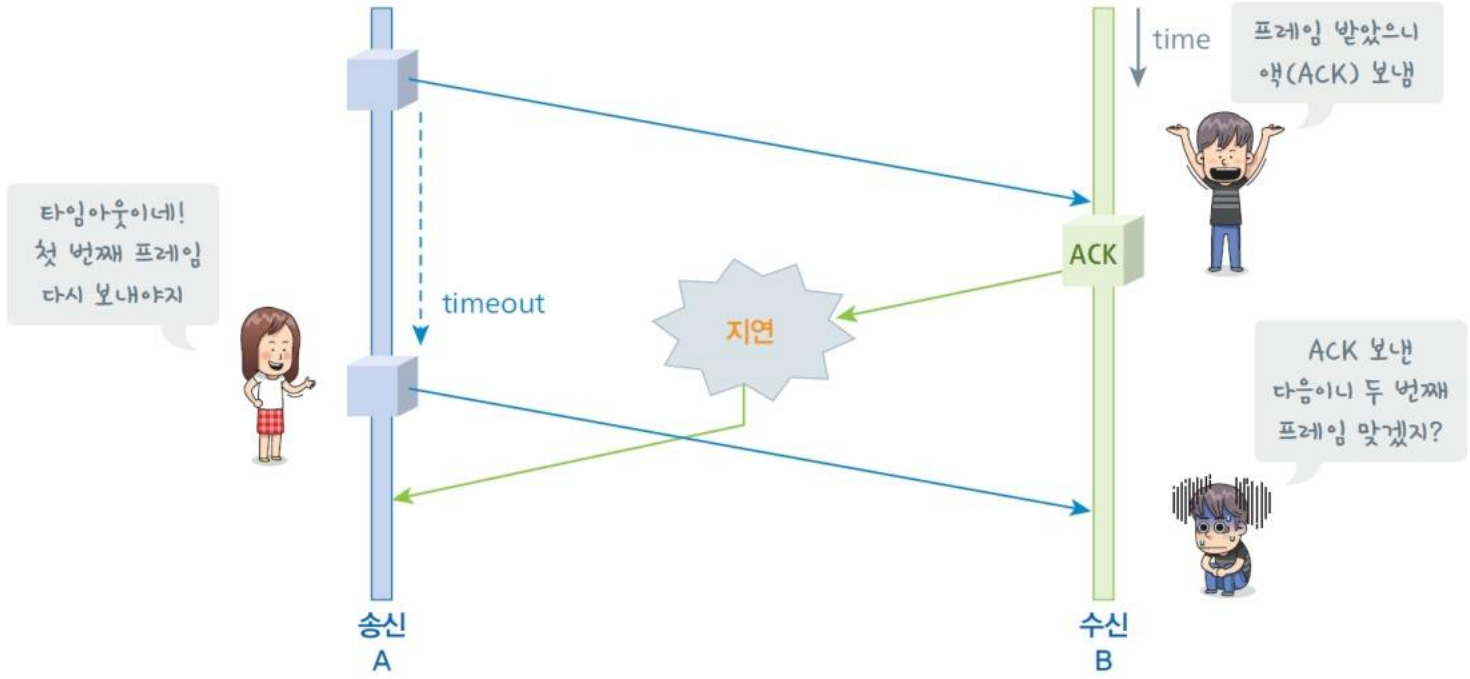


그림 6-9 타임아웃의 문제

슬라이딩 윈도우 프로토콜

- **일련번호**sequence number 사용
  - 일련번호를 붙이면 보내는 프레임이 몇 번째 프레임인지 정확하게 알게 됨. 따라서 프레임이 사라지거나 중복되는 경우에 이를 확인 할 수 있음.

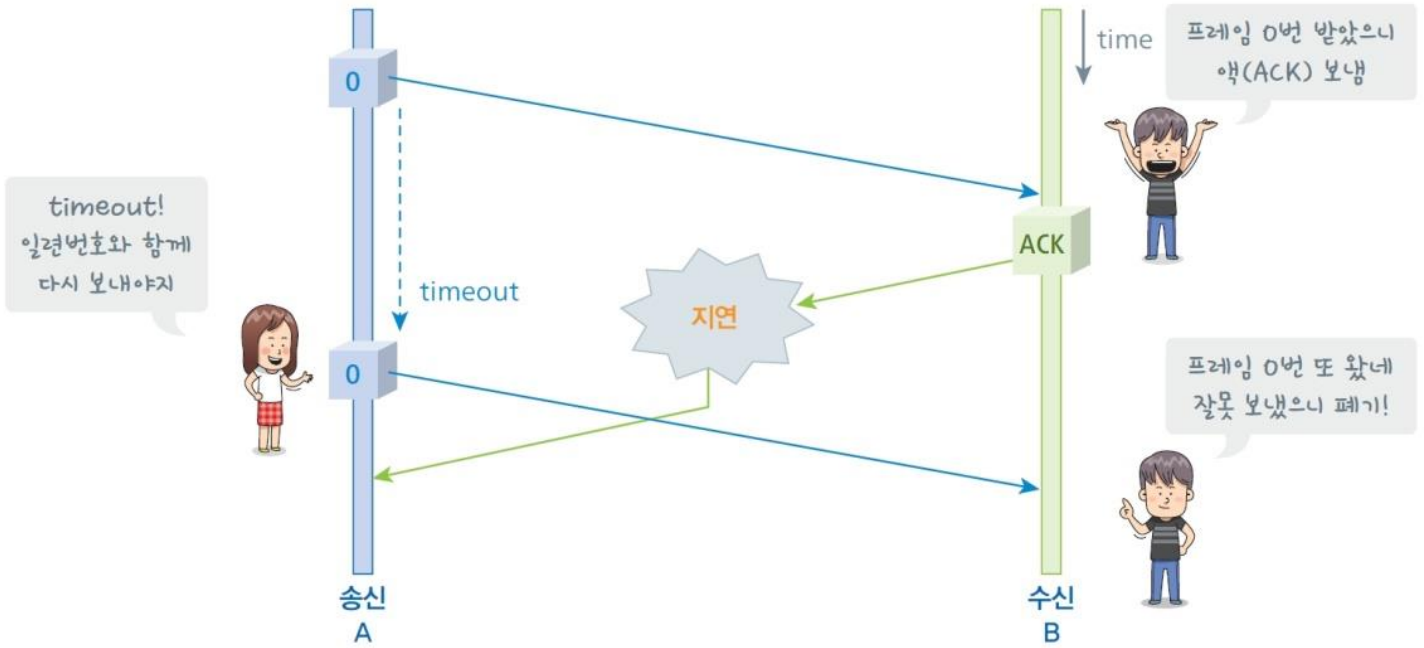


그림 6-10 프레임 일련번호의 사용

슬라이딩 윈도우 프로토콜

- 수신 B가 보낸 ACK가 늦게 도착하여 송신 A에서 타임 아웃이 걸렸음 -> 0번 프레임 다시 전송.
- 0번 프레임이 다시 전송된 후에 지연된 ACK가 도착 -> 송신 A는 다시 전송한 0번 프레임의 ACK라 생각하고 1번 프레임을 보냄.
- 1번 프레임 사라짐 -> 수신 B로부터 두 번째 0번 프레임에 대한 ACK 도착 -> 송신 A는 방금 보낸 1번 프레임에 대한 ACK이라 착각 -> 2번 프레임 전송.

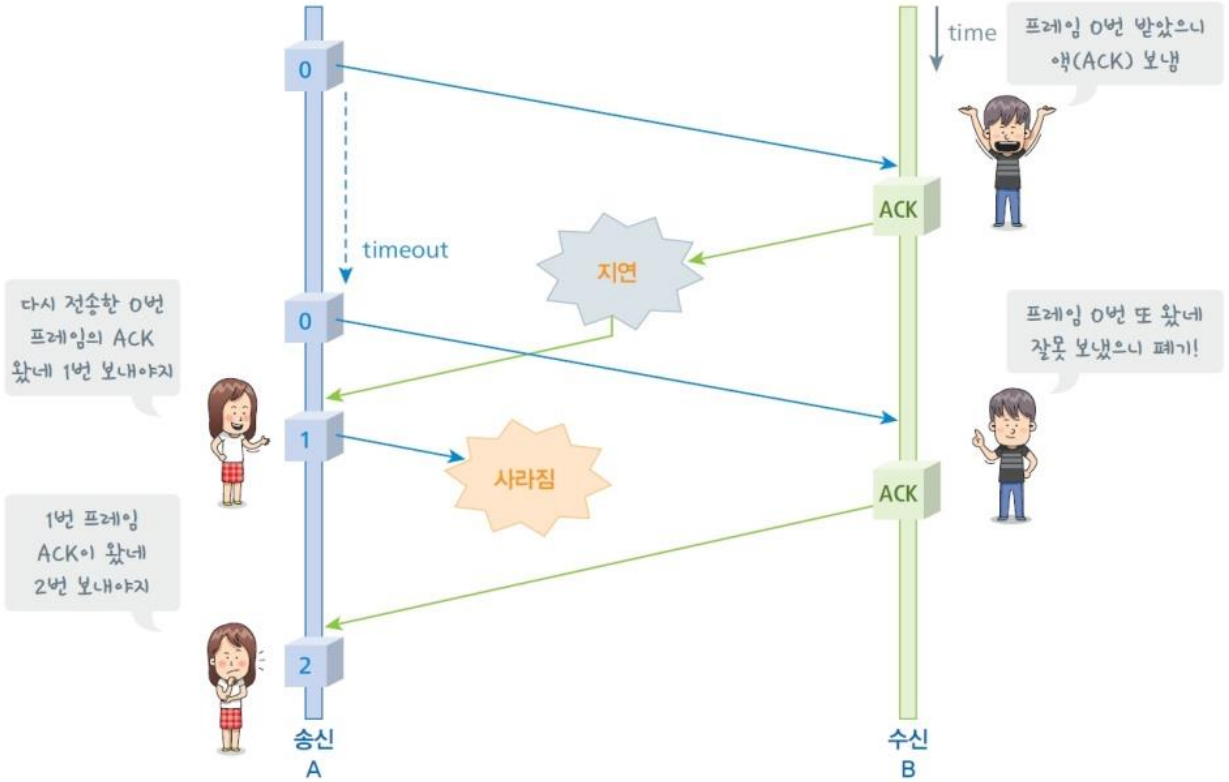


그림 6-11 ACK 일련번호가 없는 경우의 문제

슬라이딩 윈도우 프로토콜

- **ACK 일련번호**sequence number 사용
  - 송신 A는 0번 ACK가 두 번 도착하여 중복되었다는 사실을 알고 폐기.
  - ACK 일련번호를 사용하면 1번 ACK가 도착하기 전에 2번 프레임을 전송하는 문제 해결.

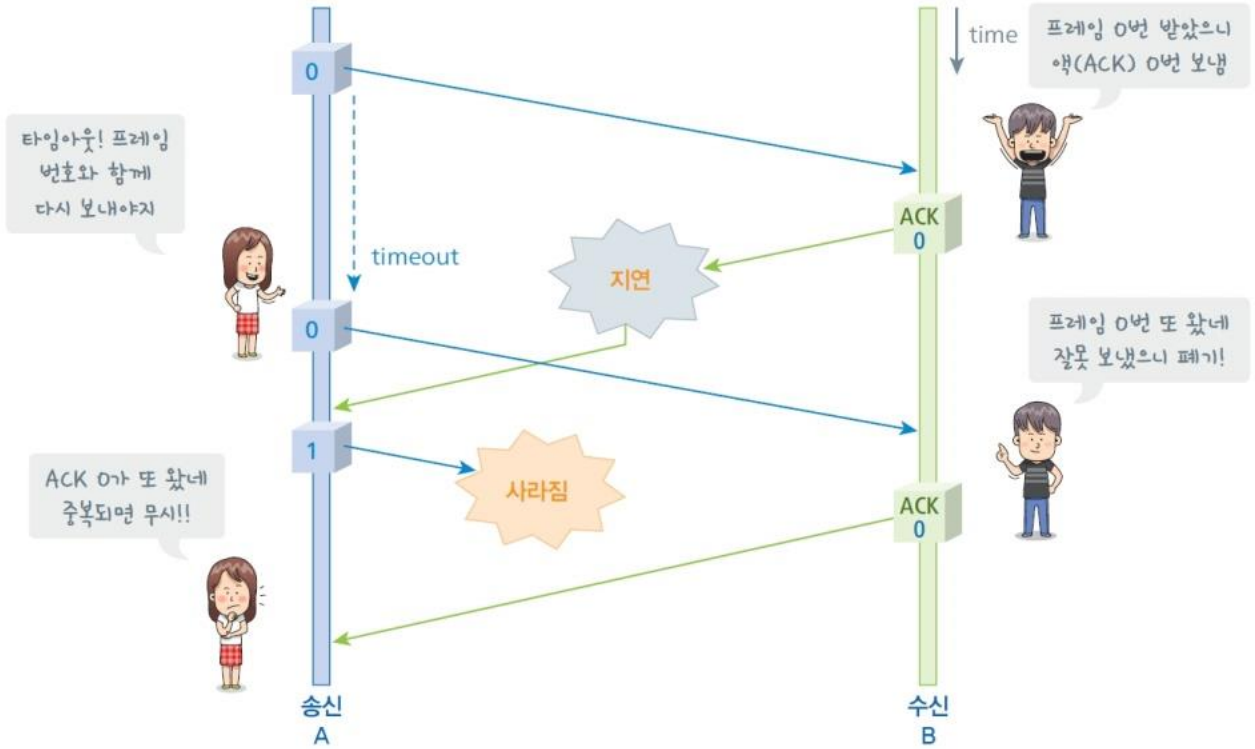


그림 6-12 ACK 일련번호 사용

슬라이딩 윈도우 프로토콜

- 데이터전송의 필수요소

> 데이터 전송 필수요소

응답 메시지(ACK) 사용

타임아웃 사용

프레임 일련번호 사용

ACK 일련번호 사용

- 데이터도 없이 ACK만 보내는 것은 낭비 -> 양방향 통신이기 때문에 양쪽에서 상대방에게 보내는 데이터가 존재 -> 전송되는 데이터에 ACK를 같이 넣어서 보내면 네트워크가 덜 혼잡해 짐.
- 이렇게 기존의 메시지에 ACK를 얹어서 보내는 방식이 **피기백킹**piggy-backing

슬라이딩 윈도우 프로토콜

2. 슬라이딩 윈도우 프로토콜

- 데이터를 보낸 후 멈추고(stop), ACK를 기다린다(wait)는 Stop-and-Wait 방식은 매우 느림.
- 에러 없는 네트워크에서 ACK를 매번 기다리는 것은 낭비.
- ACK 없이 한꺼번에 많은 양의 데이터를 보내면 전송속도가 올라감 -> **슬라이딩 윈도우**sliding window **프로토콜** 혹은 연속전송continues 프로토콜이라 부름.

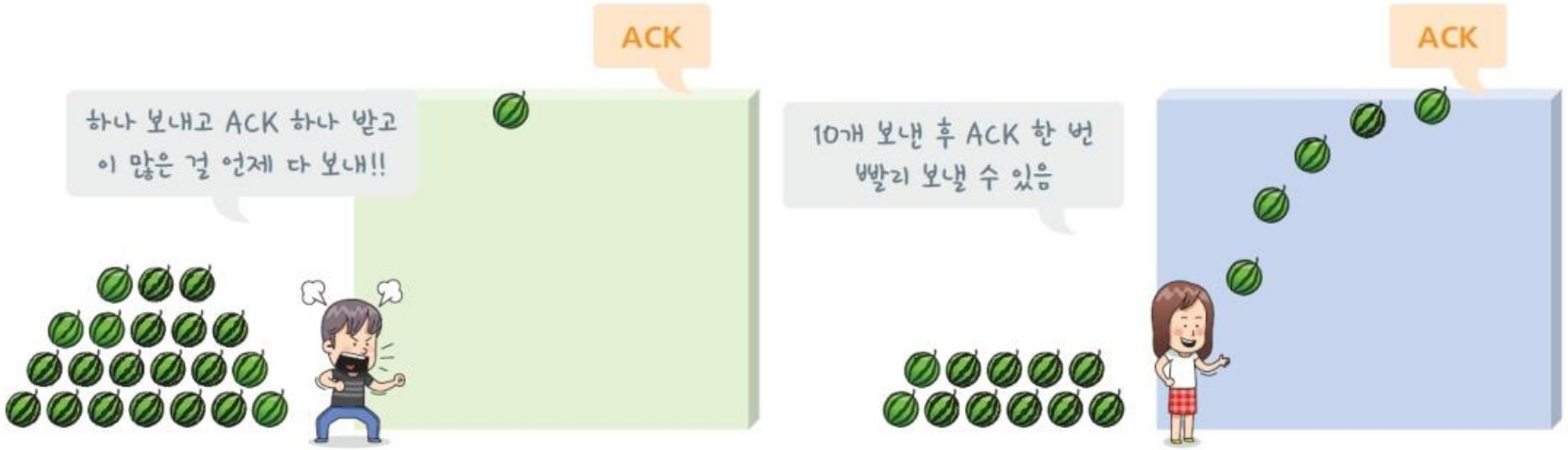


그림 6-13 여러 개의 프레임을 보내고 ACK 한 번 받기

SECTION 02

슬라이딩 윈도우 프로토콜

- 윈도우 크기window size : 슬라이딩 윈도우 프로토콜에서는 보내는 쪽과 받는 쪽에서 ACK 없이 보낼 수 있는 프레임의 개수.
- 윈도우 크기가 4라는 의미는 보내는 쪽에서 ACK 없이도 4개의 프레임을 연속적으로 보낼 수 있다는 의미.
- 받는 쪽에서는 마지막 4번째의 ACK만 보냄 -> ACK를 받으면 다음 번 4개의 프레임을 전송.

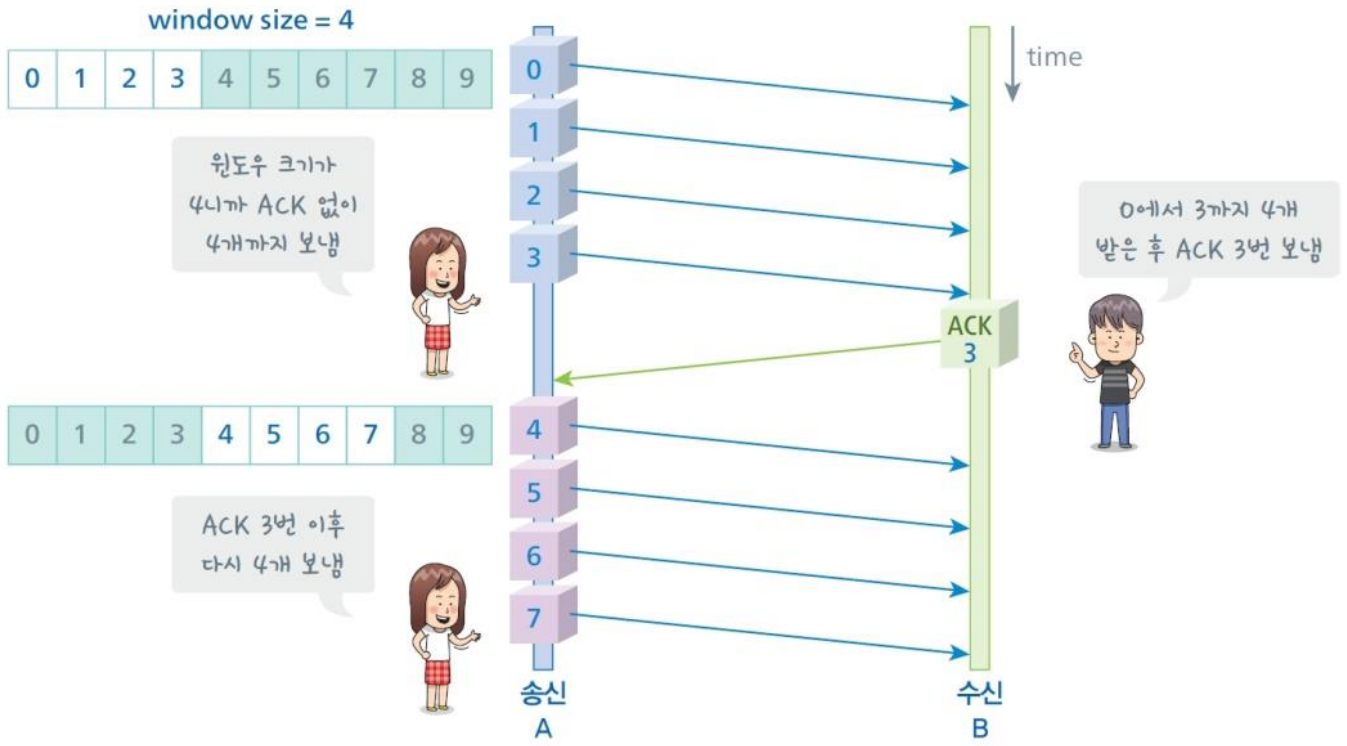


그림 6-14 슬라이딩 윈도우 프로토콜

슬라이딩 윈도우 프로토콜

- Stop-and-Wait 방식을 **Stop-and-Wait ARQ** 라 부름 : 윈도우 사이즈 1.
- 슬라이딩 윈도우 프로토콜에서 수신 B는 받지 못한 프레임에 대하여 부정응답(NAK)을 발송.
- **Go-Back-N ARQ**에서 2번 NACK을 받은 경우, 2번 프레임부터 시작하여 2, 3, 4, 5의 4개의 프레임을 다시 보냄.

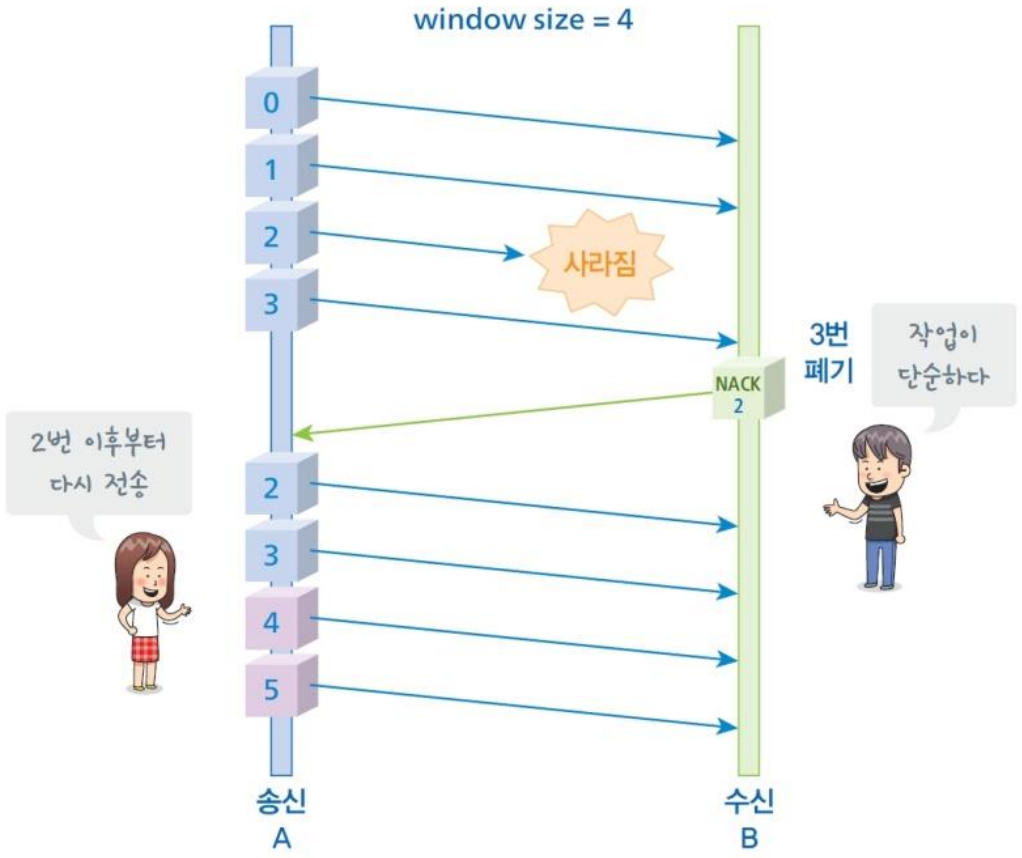


그림 6-15 Go-Back-N ARQ

슬라이딩 윈도우 프로토콜

- **Selective Repeat ARQ에**서는 NACK을 받은 2번 프레임만 다시 전송 -> 2, 4, 5, 6의 4개의 프레임이 전송.
- Selective Repeat ARQ를 사용하기 위해서는 수신 B가 버퍼에 프레임 3번을 저장했다가 프레임 2번을 받은 후 프레임을 순서대로 다시 조합해야 함 -> 회로도 복잡 -> Go-Back-N ARQ는 단순함.

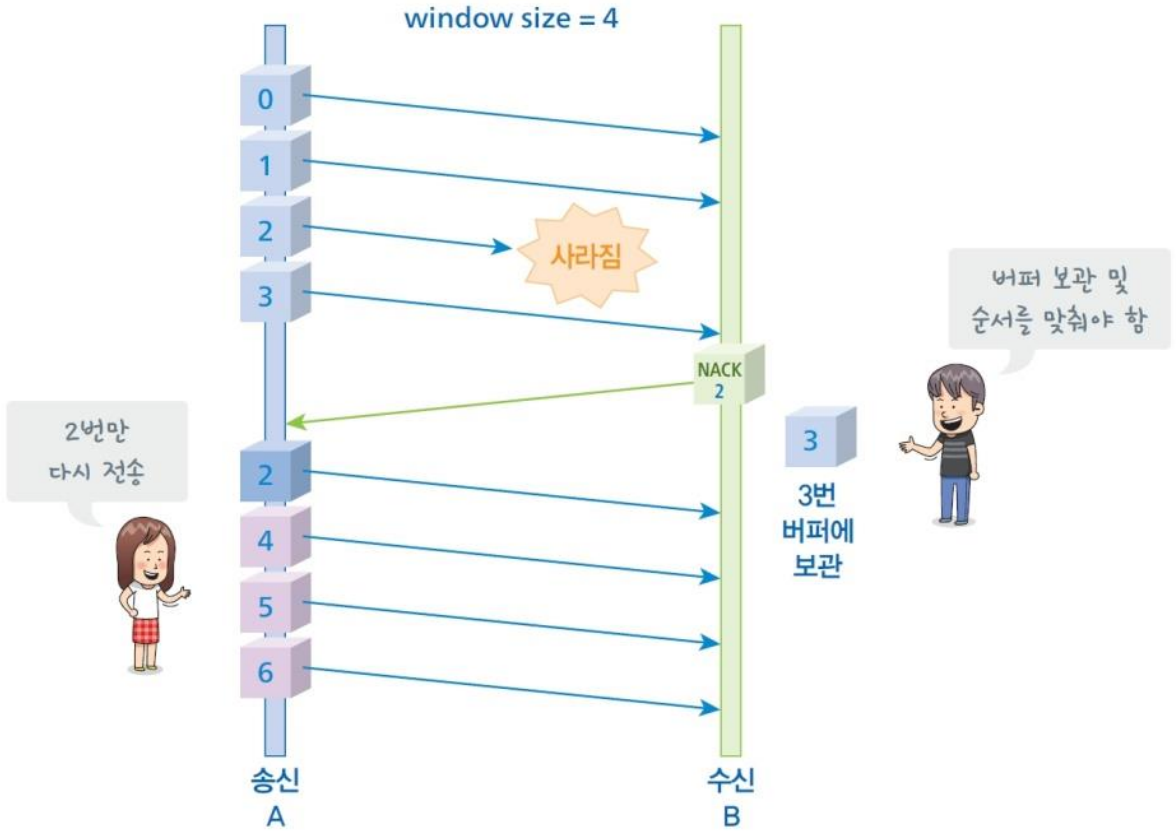


그림 6-16 Selective Repeat ARQ

슬라이딩 윈도우 프로토콜

- Adaptive ARQ에서는 네트워크가 안정적인 경우 프레임의 크기를 늘리고, 네트워크가 불안정하면 프레임의 크기를 줄임.
- Adaptive ARQ는 네트워크가 안정적인 때 최대한 큰 프레임을 보냄으로서 전송효율을 높이는 방식 -> 그러나 회로가 복잡하기 때문에 현재에는 잘 사용하지 않음.

표 6-1 슬라이딩 윈도우 프로토콜

ARQ	특징
Stop and Wait ARQ	window size = 1, 안정적이지만 느리다.
Go-Back N ARQ	연속 전송 방식, 작업이 단순하지만 패킷이 중복되는 단점이 있다.
Selective Repeat ARQ	연속 전송 방식, 받는 쪽이 재조합을 해야 하고 버퍼가 커야 한다.
Adaptive ARQ	채널의 상태에 따라 프레임의 크기를 조정하는 방식이다.

SECTION 03

오류 처리 코드

1. 오류 처리 코드의 종류

- 에러를 찾을 때 사용하는 코드를 에러 탐색 코드error detection code
- 에러를 찾을 뿐 아니라 원래의 값으로 보정해주는 코드가 에러 보정 코드error corection code

표 6-2 오류 처리 코드

방식	종류	비고
오류 탐색 코드	패리트 비트, CRC 코드, 검사합	오류를 찾는다.
오류 보정 코드	허밍 코드	오류를 찾고 보정, 오버헤드가 크다.

오류 처리 코드

2. 패리티 비트

- 가장 간단한 에러 탐색 코드가 **패리티 비트**parity bit
- 보내려는 데이터에 추가로 1비트를 만듦. 추가된 비트에 1이나 0을 넣어 전체 1의 개수가 짝수 혹은 홀수가 되도록 만드는 방식.
- 1의 개수가 짝수인 것을 짝수 패리티 비트even parity bit, 홀수인 것을 홀수 패리티 비트odd parity bit

그림 6-17 짝수 패리티 비트와 홀수 패리티 비트

오류 처리 코드

- 패리티 비트 방식은 간단하지만, 연속에러에 취약.
- 패리티 비트에서 에러가 짝수 개 발생하면 에러를 찾지 못함.

그림 6-18 패리티 비트에서 짝수 개의 오류는 찾기 어렵다

오류 처리 코드

- 여러 개의 에러를 찾기 위해 패리티 비트를 수직과 수평으로 배열할 수도 있음.
- 그러나 이 방법도 짝수개의 에러에 취약하기는 마찬가지.
- 보통의 경우 패리티 비트는 추가되는 1비트당 1개의 에러를 찾을 수 있음.

그림 6-19 수직-수평으로 배열한 짝수 패리티 비트

오류 처리 코드

3. CRC 코드

- CRC 코드는 적은 오버헤드로 많은 에러를 찾을 수 있음 -> 가장 많이 사용되는 에러 검출코드.
- 보내려는 쪽과 받으려는 쪽에서 똑같은 'CRC 코드 값'을 알고 있다. 그림에서 9.
- 보내는 쪽에서는 데이터를 CRC 코드 값으로 나누었을 때 0이 되도록 숫자를 추가하여 보냄.
- 보내려는 데이터는 12인 경우, CRC 계산을 위해 한 자리수가 추가되기 때문에 보내려는 데이터는 120 ~ 129 사이의 값 -> 120에 6을 더한 126 전송.
- CRC코드가 9인 경우, 9로 나누어 나머지가 0이 아닌 모든 수가 에러이므로 찾을 수 있는 에러의 개수는 8개.

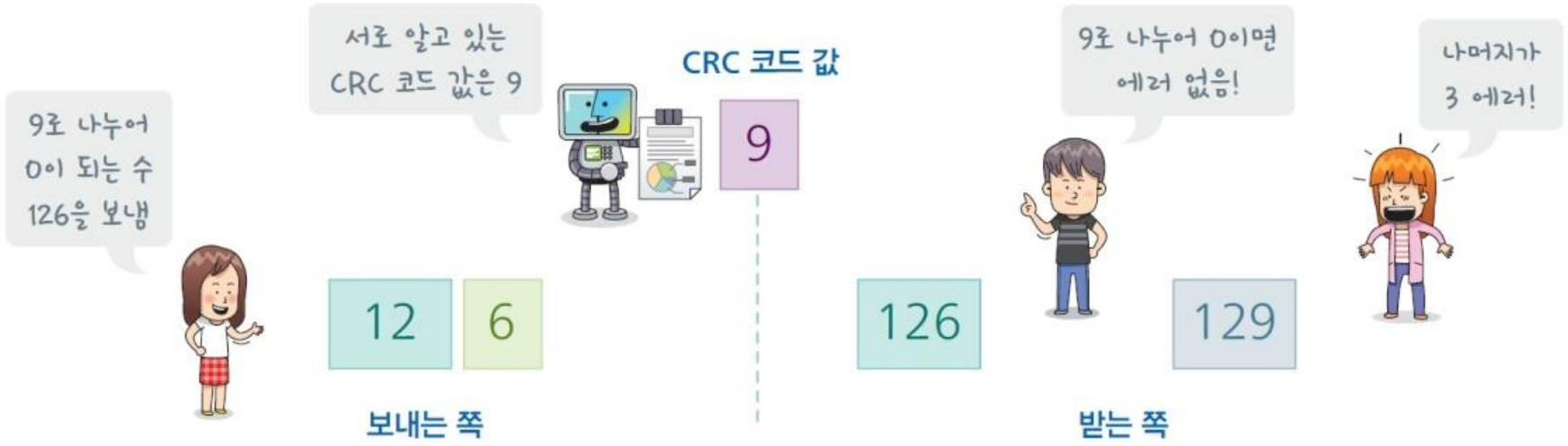


그림 6-20 CRC 코드 작동 원리

오류 처리 코드

- CRC코드에서 8비트로 찾을 수 있는 에러의 개수는 2^8 - 1인 255개.
- 16비트를 추가하는 경우, 패리티 비트는 16개를 찾을 수 있지만, CRC의 경우 2^{16} - 1개인 65535개를 찾을 수 있음.
- CRC 코드 값은 표준으로 정해져 있으며 CRC-16, CRC-32, CRC-64등이 사용 됨.
- CRC-32의 경우 이더넷 헤더에 사용. 동영상 포맷인 MPEG-2나 PKZIP과 같은 압축 파일, 그림 파일인 PNG 헤더에도 CRC-32가 사용 됨

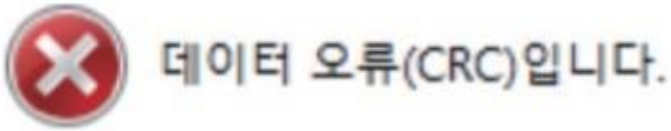


그림 6-21 CRC 오류 화면

오류 처리 코드

4. 검사합

- **검사합**checksum 혹은 체썸 코드 방식은 CRC와 유사하지만 좀 더 간단하게 에러 탐색을 할 수 있음.
- 검사합 방식에서는 보내려는 데이터를 일정 크기로 자르고 이를 더하여 에러 탐색 코드를 만듬.
- 4비트씩 잘려진 비트 덩어리를 1의 보수1's complement라 생각하고 모두 더함.
- 검사합 방식에서 4비트씩 잘라서 더하면 결과는 4비트 보다 커질 수 있음. 이렇게 커진 결과를 잘라서 4비트로 맞춤.
- 수신측에서는 검사합과 수신 받은 데이터의 최종합을 합산하여 모든 비트가 1로 나오면 에러가 없는 것임
- 변형되어 우연히 합이 같아진 경우는 이를 에러로 인지할 수 없음으로 CRC보다 찾을 수 있는 에러양이 적음

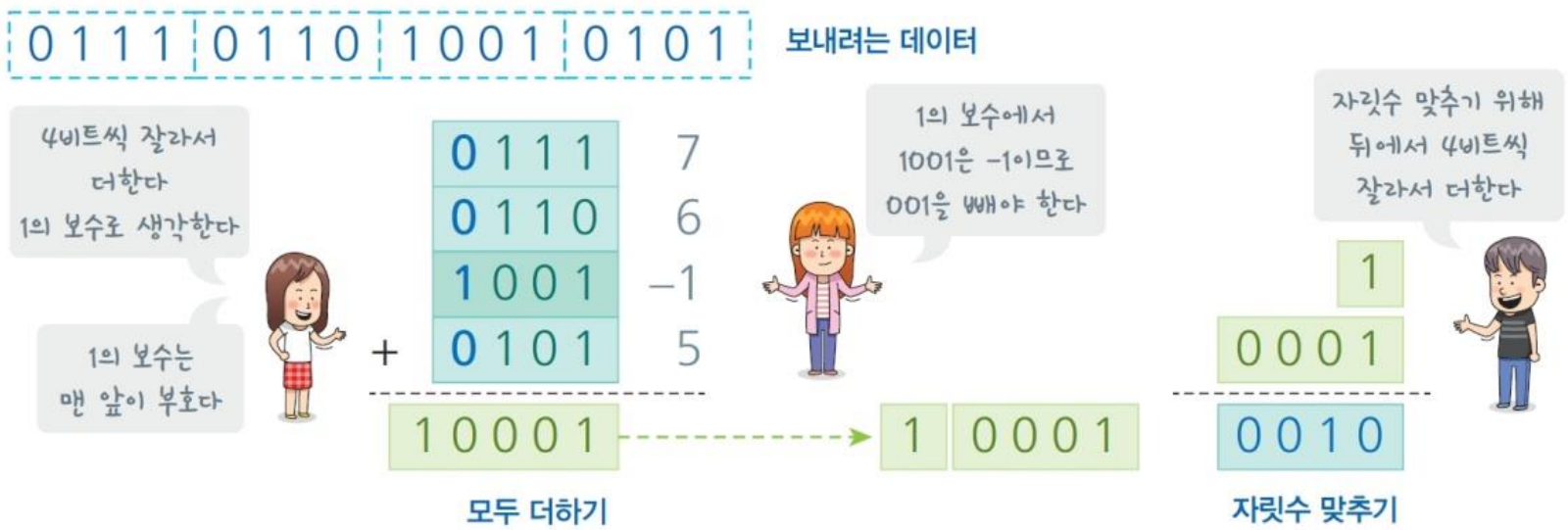


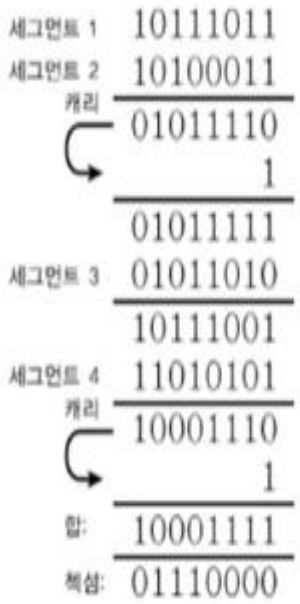
그림 6-22 검사합의 연산

오류 처리 코드

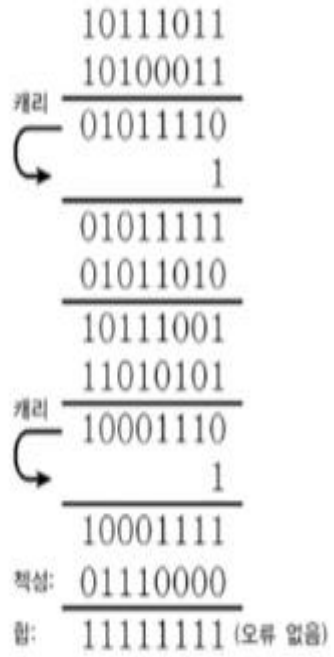
4. 검사합

- **검사합**checksum 계산 과정

data = 10111011 | 10100011 | 01011010 | 11010101



(a) 송신측의 체섬 계산



(b) 수신측의 체섬 검사

예제) 다음 16비트 데이터를 8비트 세그먼트로 분류할 때, 송신측에서 보내는 체섬값은 무엇인가?

1010100100111001

풀이) seg1 = 10101001

seg2 = 00111001

=> 11100010

답) 1의 보수(complement) -> 송신측에서 만든 체섬값 : 00011101

오류 처리 코드

4. 검사합과 CRC 비교

- CRC 방식은 나눗셈 필요: 회로 복잡
- 검사합 방식은 1의 보수 덧셈만 필요: 회로 단순, 연산 속도가 빠름
- 인터넷 프로토콜 IP와 TCP는 검사합 방식으로 오류 검출: "인터넷 검사합" 으로 불림
- 16bit 기준으로 CRC는 16비트로 나눈 나머지가 0이 아니면 모두 오류: 2^{16}-1
- 16bit 기준으로 검사합도 데이터를 더한 값이 검사합과 다를 경우 오류이기 때문에 2^{16}-1로 보임
  - 변형되어 우연히 합이 같아진 경우는 이를 오류로 인식하지 못함
  - Ex) 10진수로 7, 6, -1, 5 데이터에 오류가 발생하여 5, 7, -2, 7로 바뀌었다면 모두 17의 결과
  - 일부 찾을 수 없는 오류가 있으므로 CRC보다 찾을 수 있는 에러양이 적음.

오류 처리 코드

5. 오류 보정 – 해밍 코드

- 해밍코드에서는 보내려는 데이터를 홀수배로 늘려서 보냄.
- 해밍거리는 각 코드끼리 차이가 나는 최소값을 의미.
- 해밍코드가 홀수인 경우 보정할 수 있는 에러의 개수는 [허밍거리/2]이다. 여기서 []는 소숫점이하 버림을 의미.
- 오버헤드가 커서 일반적인 경우에는 해밍코드를 사용하지 않음.

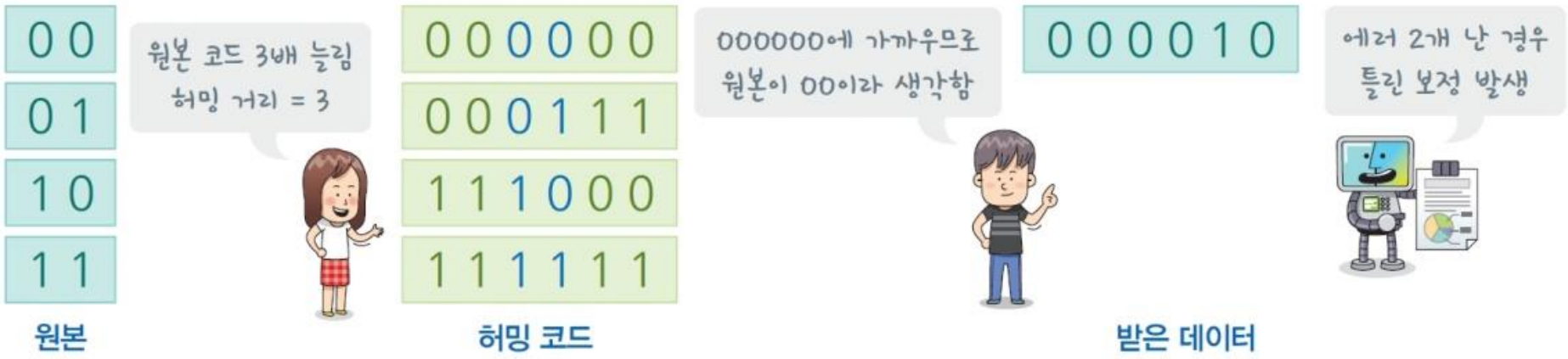


그림 6-23 허밍 코드

오류 처리 코드

6. CRC 자세히 알기

- 보내려는 데이터는 1100이고 CRC 코드 값은 1001.
- 나머지를 붙여서 보내는 비트만큼 0을 추가한 후 나눗셈을 함. -> 1100000을 대상으로 나눗셈 연산을 함.

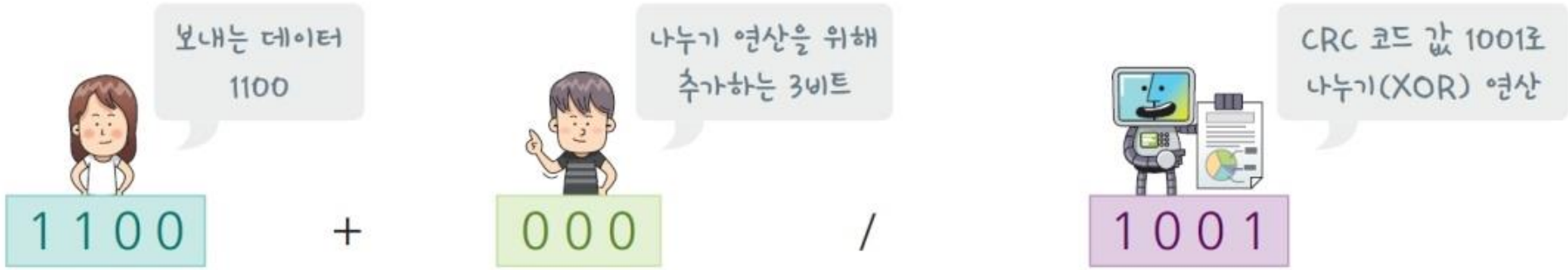


그림 6-24 CRC 나눗셈 연산 전 준비

오류 처리 코드

- CRC에서의 나눗셈은 정확하게는 모듈러modular 2 연산(XOR 연산).
- 나머지 101을 보내려는 1100에 붙여서 보냄.

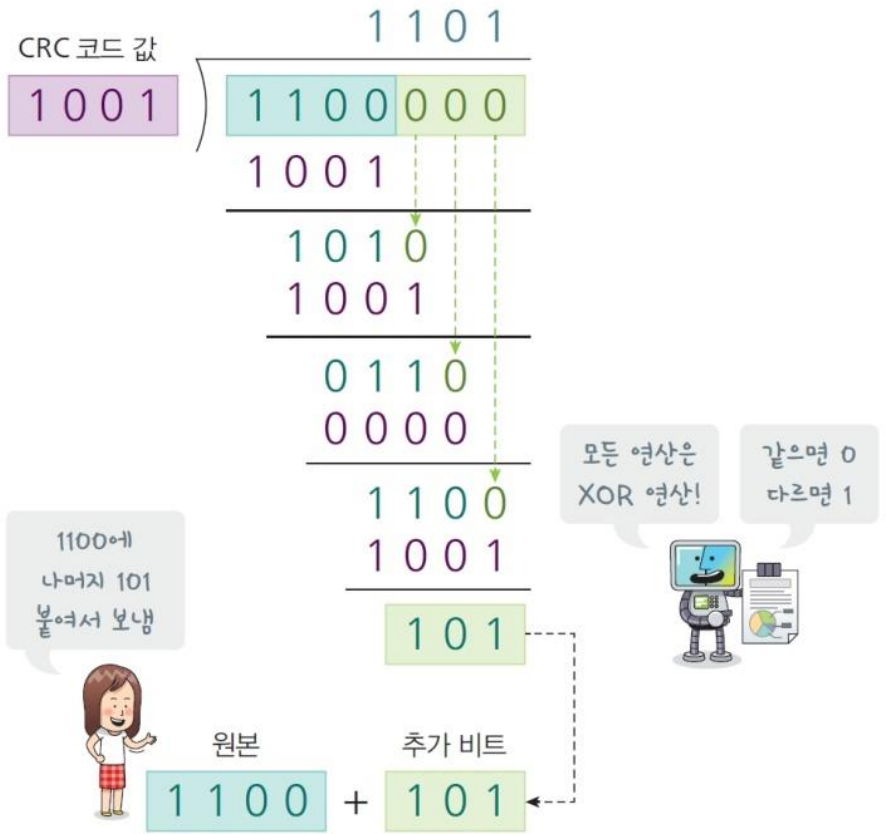


그림 6-25 CRC에서 보내는 쪽의 연산

오류 처리 코드

- 전송된 1100101을 CRC 코드 값 1001로 나눔.
- 나눗셈의 결과 000이 나왔음으로 전송된 데이터에 에러가 없음.

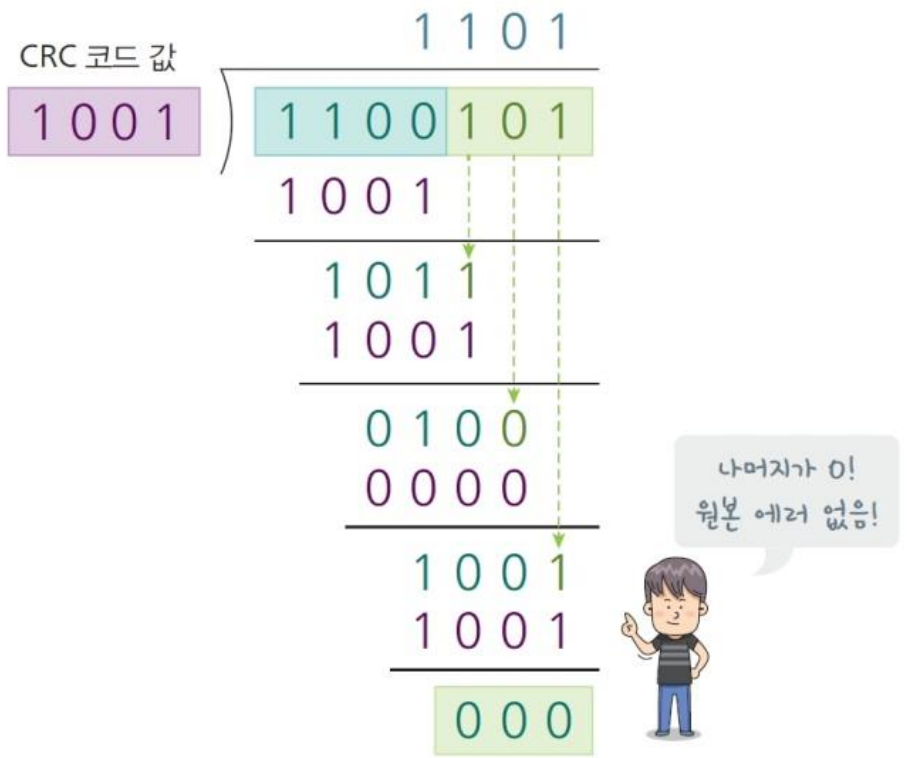


그림 6-26 CRC에서 받는 쪽의 연산

오류 처리 코드

- CRC-16의 경우 17개의 비트패턴이며, CRC-32의 경우 33개의 비트 패턴.

표 6-3 CRC 종류에 따른 비트 패턴

종류	비트 패턴	다항식
CRC-12	1100000001111	$x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
CRC-16	11000000000000101	$x^{16} + x^{15} + x^2 + 1$
CRC-32	10000010011000001000111010110111	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$