1. 프로토콜 스택에 HTTP 리퀘스트 메시지 남기기

이 동작은 애플리케이션이 write를 호출하여 송신 데이터를 프로토콜 스택에 건네 주는 곳부터 시작한다.

- 프로토콜 스택에서의 동작순서

➔ 처음 데이터를 받을 때 프로토콜 스택은 그 내용이 무엇인지 알 수 없다.

➔ 데이터를 받은 후 곧바로 송신하는 것이 아니라 일단 내부에 있는 송신용 버퍼 메모리 영역에 저장하고, 애플리케이션이 다음 데이터를 건네 주기를 기다린다.

➔ 이때 애플리케이션이 건네 주는 데이터 크기는 프로토콜 스택에서 제어할 수 없다.

- 애플리케이션으로부터 데이터를 받을 때마다 바로 송신할 수도 있지만 그럴 경우 크기가 작은 패킷이 많이 보내져 네트워크의 이용 효율이 저하될 수 있다.

데이터를 어느 정도까지 저장할 지 판단은 다음 요소를 바탕으로 한다.

1. 첫 번째는 한 패킷에 저장할 수 있는 데이터 크기이다.

➔ 프로토콜 스택은 MTU라는 매개변수를 바탕으로 한 패킷에 저장할 수 있는 데이터 크기를 판단한다.

- MTU : 한 개 패킷으로 운반할 수 있는 디지털 데이터의 최대 길이(보통 1500바이트)

➔ MTU의 맨 앞에는 헤더가 포함되므로 헤더를 제외한 것이 실제 데이터 길이 MSS가 된다.

- MSS : 헤더를 제외하고 한 개의 패킷으로 운반할 수 있는 TCP 데이터의 최대 길이

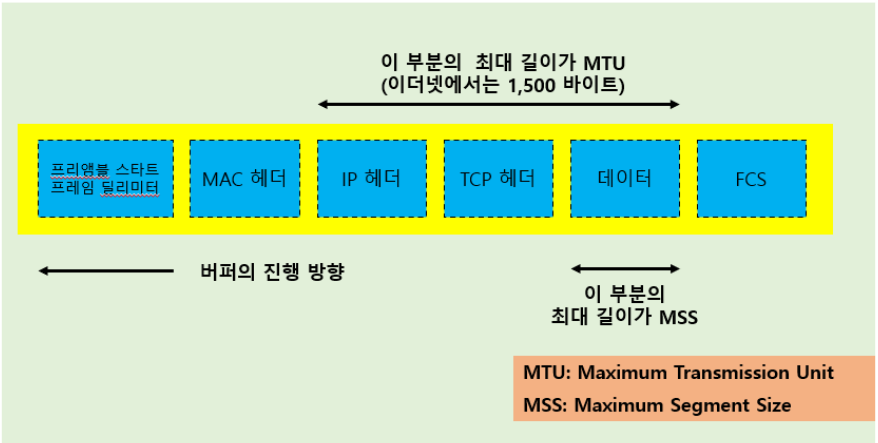

그림 1. MTU와 MSS

2. 두 번째는 타이밍이다.

➔ 애플리케이션의 송신 속도가 느려 지는 경우 MSS에 가깝게 데이터를 저장하면 여기에서 시간이 걸려 송신이 지연된다.

➔ 그렇기에 버퍼에 데이터가 모이지 않아도 적당한 곳에서 송신 동작을 실행해야 한다.

➔ 따라서 프로토콜 스택은 내부에 타이머가 있어 이것으로 일정 시간 이상 경과하면 패킷을 송신한다.

- 두 가지 판단 요소의 상반되는 문제점

➔ 데이터 크기를 맞추면 네트워크 이용 효율이 높아지지만 송신 동작이 지연될 수 있다.

➔ 타이밍을 고려할 경우 지연은 적어 지지만 네트워크 이용 효율이 떨어지게 된다.

2. 데이터가 클 때 분할하여 보내기

한 개의 패킷에 들어가지 않을 만큼 긴 데이터를 보낼 경우 송신 버퍼에 저장된 데이터는 MSS의 길이를 초과하므로 다음 데이터를 기다릴 필요가 없다.

따라서 송신 버퍼에 있는 데이터를 맨 앞부터 차례대로 MSS의 크기에 맞게 분할 후, 분할한 조각을 한 개씩 패킷에 넣어 전송한다.

이후 각 조각마다 맨 앞부분에 TCP헤더를 부가한다.

TCP 헤더가 부여된 조각 앞에는 다시 IP헤더, MAC헤더 등이 부가된다.

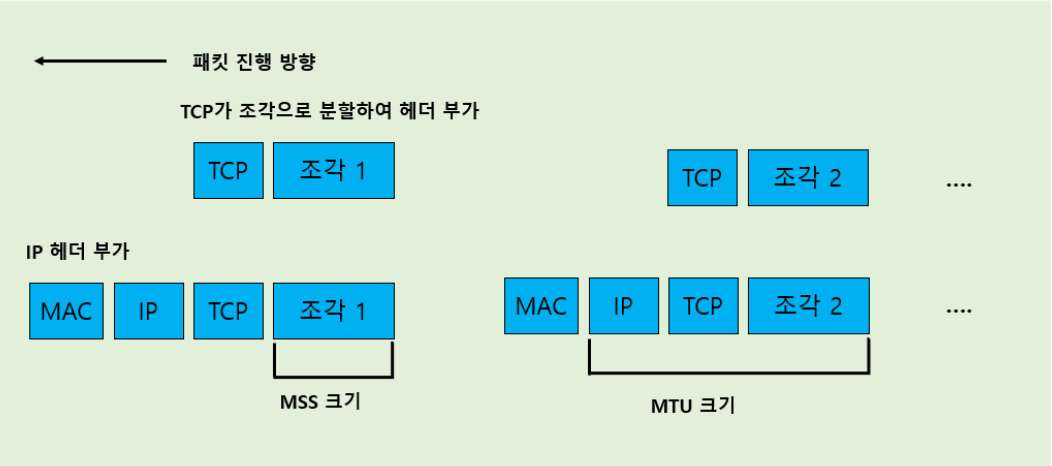

그림 2. 애플리케이션의 데이터 분할

3. ACK번호를 사용하여 패킷이 도착했는지 확인하기

TCP는 송신한 패킷이 상대에게 올바르게 도착했는지 확인하고, 도착하지 않았으면 다시 송신하는 기능 있으므로 패킷을 송신한 후에는 확인 동작으로 넘어간다.

TCP담당 부분은 데이터를 조각으로 분할할 때 조각이 통신 개시부터 따져서 몇 번째 바이트에 해당하는지 세어 둔다. 그리고 데이터 조각을 송신할 때 세어 둔 값을 TCP 헤더에 기록하는데, 이것이 **시퀀스 번호** 항목에 해당된다. 그림 3과 같은 방법을 통해 수신 측에서는 패킷이 누락되었는지 확인 할 수 있다.

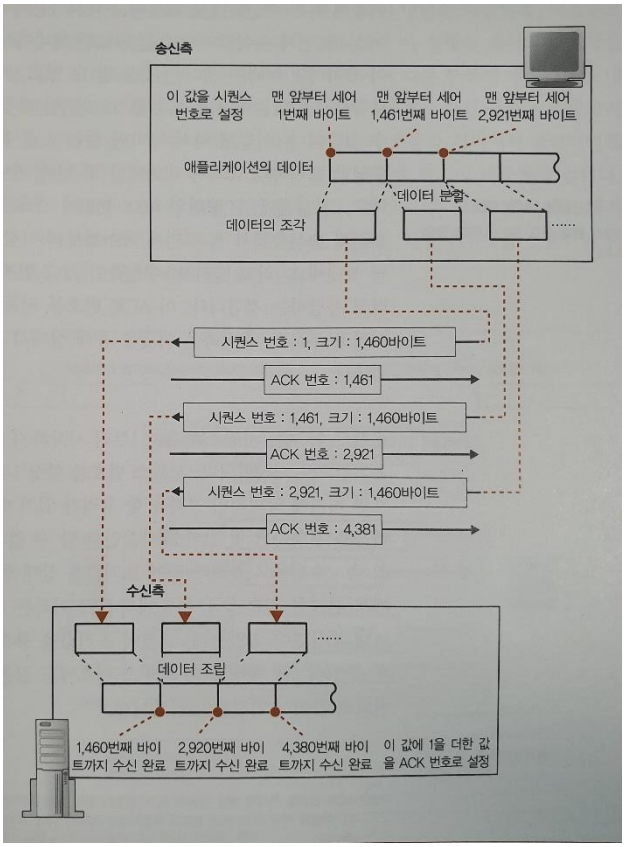

그림 3. 시퀀스 번호와 ACK 번호의 사용

(이미지 출처 : "성공과 실패를 결정하는 1%의 네트워크 원리")

수신 측에서는 누락없이 데이터를 확인하면 이전에 수신한 데이터와 합쳐서 데이터를 몇 번째 바이트까지 수신한 것인지 계산하고, 그 값을 TCP헤더의 **ACK 번호**에 기록하여 송신 측에 알려준다.

- TCP 통신 과정

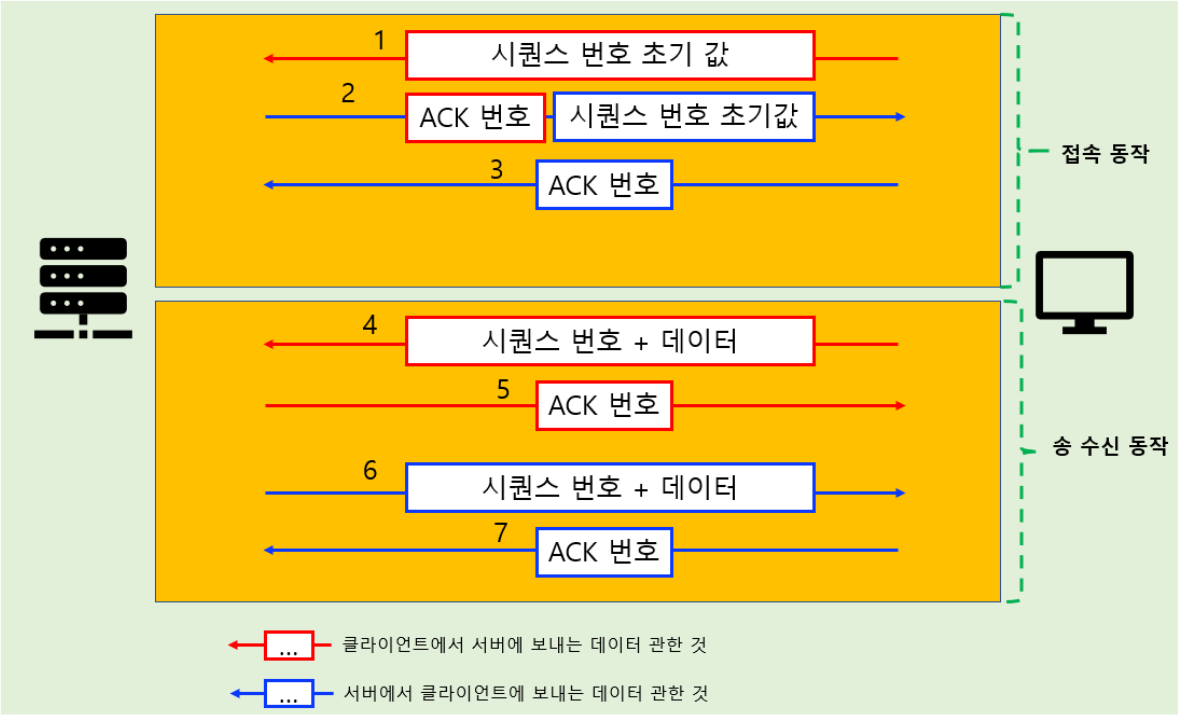

그림 4. 시퀀스 번호와 ACK 번호의 대화

이러한 구조로 인해 LAN어댑터, 버퍼, 라우터 모두 회복 조치를 취하지 않고 패킷을 다시 보내는 방식으로 회복 처리를 할 수 있다.

4. 패킷 평균 왕복 시간으로 ACK번호 대기 시간 조정하기

ACK번호가 돌아오는 것을 기다리는 대기 시간을 **타임아웃 값**이라고 한다.

네트워크가 혼잡하여 정체가 일어나면 ACK번호가 돌아오는 것이 지연되므로 이것을 예측하여 대기 시간을 어느정도 길게 설정해야 한다.

대기 시간을 적절한 값으로 설정하기 위해 TCP는 대기 시간을 동적으로 변경하는 방법을 취한다.

➔ 데이터 송신 동작을 실행하고 있을 때 항상 ACK번호가 돌아오는 시간을 계측해 둔다.

➔ 그리고 ACK 번호가 돌아오는 시간이 지연되면 이것에 대응하여 대기 시간도 늘린다.

➔ 반대로 짧아질 경우 대기 시간을 짧게 설정한다.

5. 윈도우 제어 방식으로 효율적으로 ACK번호 관리

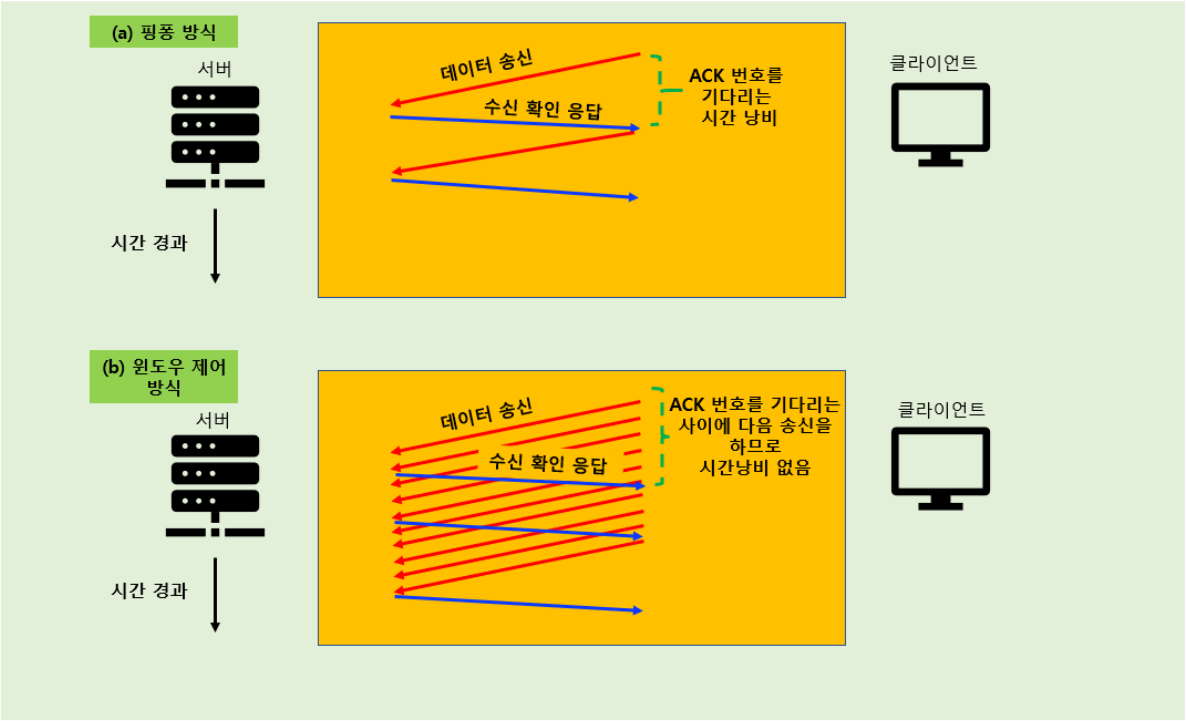

그림 5. 핑퐁 방식과 윈도우 제어 방식

그림 5의 (a)와 같이 한 개 패킷을 보내고 ACK번호를 기다리는 방법은 단순하고 이해가 쉽지만 ACK번호가 돌아올 때까지 아무 일도 하지 않고 기다리는 것은 시간 낭비가 된다.

이러한 낭비를 줄이기 위해 TCP는 그림5의 (b)와 같이 윈도우 제어라는 방식을 사용한다.

- ➔ 윈도우 제어는 한 개의 패킷을 보낸 후 ACK번호를 기다리지 않고 차례대로 연속해서 복수의 패킷을 보내는 방법이다.
- ➔ 단, ACK번호를 기다리지 않고 계속 보낼 경우 수신 측의 능력을 초과하여 패킷을 보내는 사태가 발생할 수 있다.
- ➔ 이를 해결하기 위해 수신 측에서 송신 측에 수신 가능한 데이터 양을 통지하고, 수신 측은 이 양을 초과하지 않도록 송신 동작을 실행한다.
- ➔ 수신 측의 빈 영역이 0이되면 송신을 중지하고, 빈 영역 값을 받으면 다시 송신한다.

수신 가능한 데이터 양의 최대 값을 **윈도우 사이즈**라고 부른다.

6. ACK번호와 윈도우를 합승

수신 측에서 애플리케이션에 데이터를 건네 주고 수신 버퍼의 빈 영역이 늘어나면 이것을 송신 측에 통지해야 하는데 이것이 **윈도우 통지 타이밍**이다.

수신 측에서는 ACK번호와 윈도우 통지 번호를 둘 다 보내야 하는데 두개의 패킷이 하나씩 따로 송신 측에 보내질 경우 패킷이 많아져 효율성이 저하 된다.

- 전송 패킷이 많아질 때 해결 방법

➔ ACK 번호를 송신할 때 윈도우 통지가 일어나면 ACK 번호와 윈도우를 한 개의 패킷에 합승 시켜서 통지하여 패킷의 수를 줄일 수 있다.

➔ 복수의 ACK 번호 통지가 연속해서 일어날 경우 최후의 것만 통지하고 도중의 것은 생략한다. 이렇게 해도 송신 측에서는 데이터를 어디까지 받았는지 확인 할 수 있게 된다.

➔ 윈도우 통지가 여러 개일 경우에도 마지막 것만 통지한다.

7. HTTP 응답 메시지를 수신

브라우저는 리퀘스트 메시지를 송신해 달라고 의뢰한 후, 서버에서 돌아오는 응답 메시지를 받기 위해 read 프로그램을 호출한다.

- 응답 메시지가 도착 후 프로토콜 스택의 동작과정

➔ 프로토콜 스택은 수신한 데이터 조각과 TCP 헤더의 내용을 조사하여 도중에 데이터가 누락되었는지 검사하고, 문제가 없으면 ACK 번호를 반송한다.

➔ 이후 데이터 조각을 수신버터에 일시 보관하고, 조각을 연결하여 데이터를 원래 모습으로 복원 후 애플리케이션이 지정한 메모리 영역에 옮겨 기록한 후 애플리케이션에 제어를 되돌려 준다.

➔ 이후 타이밍을 가늠하여 윈도우를 송신 측에 통지한다.