TCP(Connection-oriented Transport Layer Protocol)

이메일은 한자라도 틀리면 안된다는 특성 -> 신뢰성이 높아야 하는 데이터에 적합

1. Reliable : 무오류성. 에러가 없는. 신뢰성

무오류성을 보장하기 위해 TCP에 적용한 기술들

Checksum : 패래티 코드와 같이 오류 검출

#Acknowledge : 잘 받았다. 피드백 메시지

Timer : ACK를 적절한 시간에 보내기 위한 타이머

Sequence Number : Segment의 순서 유지를 위함

(Pipelining : segment 여러 개를 쭉쭉 보낸다. 그리고 한꺼번에(마지막 segment를 받고) ACK를 보낸다.)

2. Resent : 재전송을 통해 에러 제어

3. Delivers data in order sent : 순서 유지

4. Point to point : 한명의 sender, 한명의 receiver

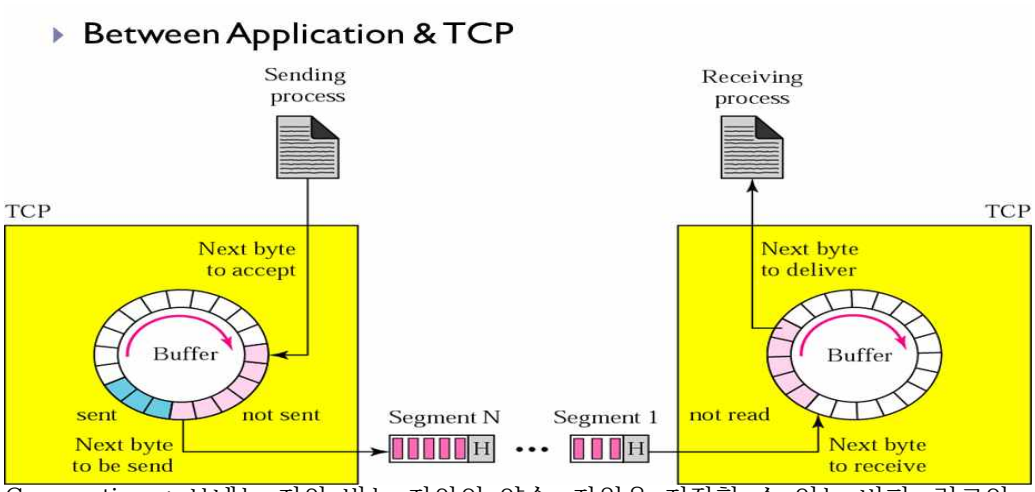
5. Full duplex data :양쪽 방향으로 데이터를 전달할 수 있는 프로토콜

6. Connection-oriented : UDP와 달리 반드시 서버와 connection을 맺고 통신(handshake)한다. 신뢰성 있는 전송을 위해서이다.

7. Congestion control : 만약 패킷의 전송 속도보다 빠른 속도로 패킷이 입력되고 대기 행렬이 쌓이게 되면 패킷 지연 시간이 기하급수적으로 증가하므로 이를 통제한다.

8. Flow control : 흐름 제어

TCP가 어떻게 connection을 맺는가? ->강의목표.시험범위



Connection : 보내는 자와 받는 자와의 약속. 자원을 저장할 수 있는 버퍼. 링크의 Bandwidth 등을 약속한다.

Application과 TCP 사이의 구조

왼쪽은 sender, 오른쪽은 receiver, Buffer는 linked list로 구동.

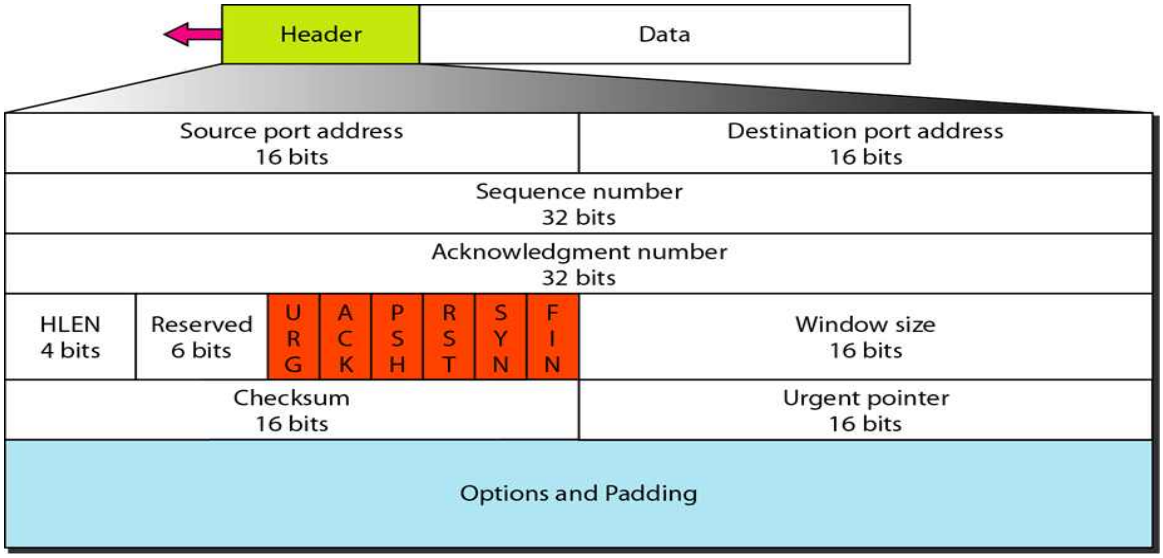
Sending process : 어플리케이션에 맞게 데이터를 생성하여 TCP에 byte단위로 던져준다. 버퍼의 한 칸이 바이트 단위로 저장됨. 바이트를 모아 Segment 단위로 데이터가 축적된 후 데이터가 나가게 된다.

그림에서 파란 부분 : 전송이 이미 된 부분도 써져 있다. Receiver가 정상적으로 데이터를 받지 못 했을 때를 대비.

Receiver측에서 어플리케이션이 분홍색 맨 위쪽부터 쭉쭉쭉 읽어 나감.

Segment 일련번호가 순서대로 오지 않았을 경우 : Buffer을 적당히 비워 두고 나중에 채워준다. ->모든 segment의 길이를 알아야 한다.

(Out of order segment : 3번이 먼저 들와야 하는데 4번이 먼저 들어온 경우 버퍼를 비워 놓았다가 3번이 오면 다 같이 올려 보내는 방식.)



TCP 헤더의 구조.

Acknowledge number : 잘 받은 segment들이 몇 번 까지 인지를 나타냄
다음번에 들어올 것으로 기대되는 segment의 Sequence number(Cumulative ACK : 누적된 ACK.)>

HLEN : TCP segment의 길이를 정의, option head의 끝을 알린다.

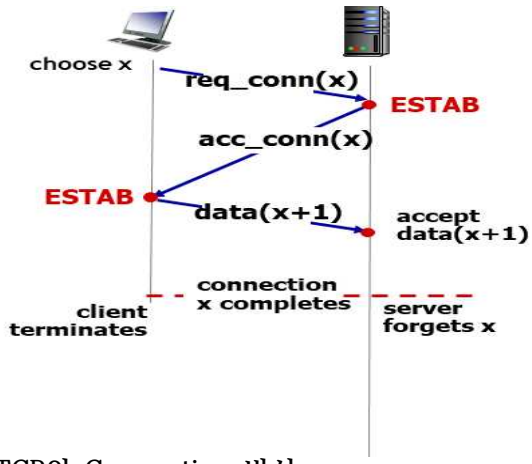
Reserved : 사용을 안하는 비트

Checksum : 오류검출

ACK : 이것이 ACK임을 알려주는 flag(Segment가 전 순서의 Segment의 Acknowledge number과 일치함을 알려주는 Flag) / 잘 받았다는 응답. 수신자 입장에서 오류가 있을 시 Acknowledge를 안 보내는 것이 오류 대처법

PSH : 데이터를 모아서 보내지 않고 모아지지 않았지만 빨리 보내야할 때 알려주는 flag(segment가 잘 모이지 않을 경우)

Sequence number : 세그먼트 데이터의 첫 번째 바이트의 number를 표시. 데이터 유실시 몇 번째 segment가 사라졌는지 알 수 있다. 1000바이트의 segment들을 보낸다고 가정하면 첫 번째 segment의 seq=1, 두 번째 segment의 seq=100



TCP의 Connection 방식

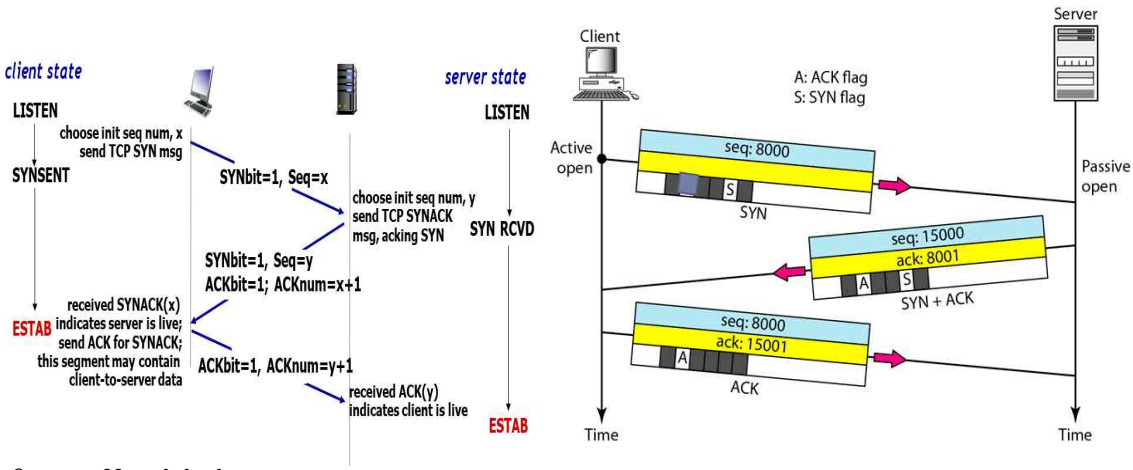
<2-way handshake>

Connection : 서로 상대를 확인을 하고 데이터 전송을 시작하자 약속을 맺는 것. UDP와 다른 점이다.

2-way handshake는 잘 사용하지 않는다.

1. Variable delay : delay가 일정하지 않다.
2. 버퍼의 낭비 (빨간 점 : 서버에서 버퍼를 할당하고 데이터 전송을 시작한다는 뜻)

->이러면 관심 없는 사람들은 집에 갔는데 전송을 시작해서 버퍼낭비 엄청 발생한다.



<3-way Handshake>

보내는 세 개의 segment는 connection을 맺기 위해 보내진다.

두 번째 패킷부터 ACK를 통해 잘 받았음을 확인시켜줌

나랑 통신 맺을래? -> 그래 데이터 줄 테니까 받을래? -> 응

Connection을 끝내는 경우 : 이 경우 또한 3-way handshake

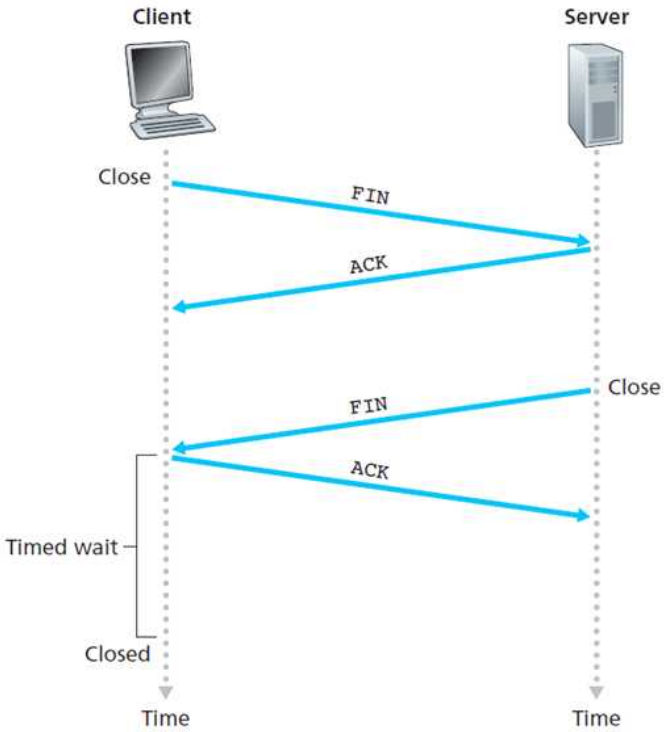


Figure 3.40 ♦ Closing a TCP connection

서버는 완전 수동적 맺자고 하는 것도 클라이언트, 끊자고 하는 것도 클라이언트
Finish flag를 set해서 보내고 서버는 그 즉시 ACK를 보냄.
근데 사실 서버는 보낼 데이터가 남아있어. ACK 를 보냈지만 더 줄게 있어. 그래서 데이터를 보내. 진짜 서버가 다 데이터를 보내면 FIN flag를 보내서 관계 청산을 하고 클라이언트가 ACK를 보내게 된다.
Time wait후에 자원(버퍼)를 해제한다.->서버의 데이터가 delay에 의해서 지연 되는 것을 조금 기다리는 용도

Application에 맞게 Transport layer 설정(UDP,TCP)
TCP : Error control, Flow control, Congestion control
Connection management : 커넥션을 맺는다.
Error Control : 리던던시의 추가(패리티 코드의 일종)