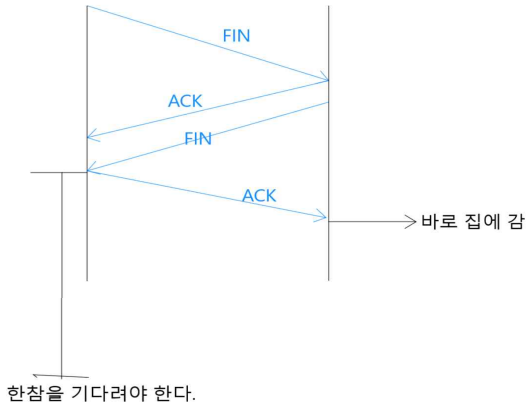
<인터넷 프로토콜>

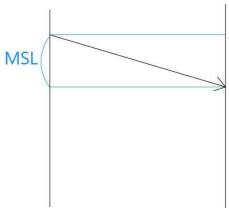
작성자_2018320161_송대선

작성일_05_23



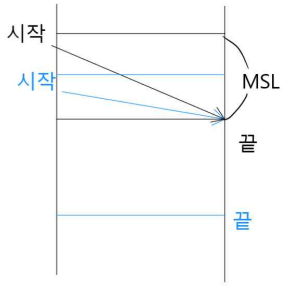

ACK가 도달하리라는 보장이 없다.
-> ACK가 안 오면 FIN을 다시 보낸다.
-> 기다리면서 올지도 모르는 FIN을 받아야 함.
(안 기다리면 저쪽이 계속 FIN을 보내면서 기다림)

결국 "TIME_WAIT"상태가 된다.




-> 패킷 하나가 배달될 때까지 걸리는 최악 시간이 1~2분이라고 생각했었다.
-> 그 최악 시간이 MSL이다.
-> 2*MSL만큼 기다린다.

"TIME_WAIT" 상태가 왜 필요한가? (심오한 이유가 있다.)
-> FIN을 기다리는게 전부인가?
-> 다른 이유가 있는가?
<$IP_A, IP_B, P_A, P_B$>이것이 하나의 TCP Connection을 identify한다.
사용자($IP_A$, $P_A$)와 웹서버($IP_B$, $P_B$)가 있을 때, $P_A$가 바뀌어야 연결할 때마다 다른 Connection이 된다.
그래서 Connection을 만들 때마다 다른 $P_A$를 바꿔준다.



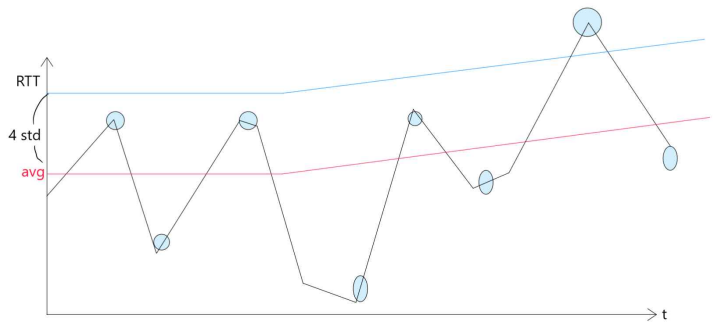
만일 같은 $P_A$, 같은 Connection을 사용한다면, 구분이 안간다.
보낸지 MSL만큼 지난 후에야 같은 Connetion을 사용가능하다.

TIME_WAIT 상태에는 동일한 $P_A$를 사용하지 못한다.

netstat에서 TCP 연결 상태를 확인할 수 있다.
-> Closed: FIN을 맞은 상태
-> ESTANLISH: 연결 성립
-> Timewait: TIME_WAIT 상태, 이 상태의 port number는 고르지 않는다.

http의 port 번호는 80이다.
OS는 최대한 다른 port 번호를 설정하려고 한다.

-TCP timeout-
재전송을 위한 Timer 설정? 시간 기준?
(도착했을 때의 시간-출발할 때의 시간)-> 평균, 표준편차를 계산
-> 여기에서의 표준편차는 루트가 없는 변형된 표준편차이다.

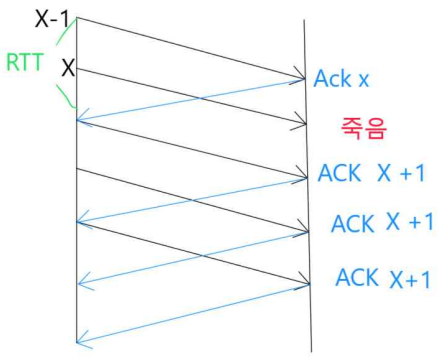


파란 선을 내리면 너무 지래짐작하고 재전송 해버린다.

ACK가 들어올 참인데, 재전송 해버리면 안 된다.
ACK가 죽었는데, 너무 오래 기다려도 안된다.
RTO = avg(RTT)+4*std(RTT) 이상으로 더 기다려야 할 패킷이 올 확률은 1/16에 불과하다.
(Chevyshef inequality이기 때문!)
패킷들이 올 때마다 running avg를 계산 가능

RTO가 1초보다 작으면 무조건 1초로 올린다.

-Fast Retransmit-




RTT + 3δ (δ는 엄청 작다.)

TCP는 OS가 허용하는 한 엄청 빠르게 쏜다.

-> 3번의 동일한 duplicated ACK를 받으면 재전송한다.
-> Fast Retransmit을 보낸다. (타이머에 비해 빨리 재전송 됨)
-> RTT + 3δ <<<<< avg(RTT) + 4*std(RTT) (δ <<<<< std(RTT))

그러나 이런 상황은 매우 예외적인 상황이다.
-> 앞뒤는 살고, 나는 죽는다.
-> 항상 가능한 것은 아니다.
(대부분 Router Conjection 때문에 죽는 건데 나만 죽고 앞뒤 사는 것은 매우 예외적이다.)
-> 기본적으로 Timer에 의존하는 것이 맞다.

-Delayed ACK algorithm-
TCP는 data전송은 엄청 열심히 하고, ACK는 엄청 늦게 전송하려고 한다.
-> TCP는 200ms이상 ACK를 뭉개지 않는다.
-> 왜 뭉개는가? 왜 일부러 늦게 전송하는가?
-> 30년 전에는 인터넷이 많이 느려서 ACK 하나 보내는 것이 아까웠다.
-> data는 없고, header만 있는 이것을 굳이 보내야 하겠나?

ACK를 보내야 하는데, 마침 원래 Sender쪽으로 가는 data Segment가 있으면, 거기에 ACK를 넣어서 보낸다.
-> 200ms를 기다려도 가는게 없으면 그냥 쌩으로 보내는 수 밖에 없다.

TCP 여러 개를 보내도, ACK가 하나가 올 수 있다.
TCP 2개 당 ACK하나가 오는 것을 쉽게 볼 수 있다.

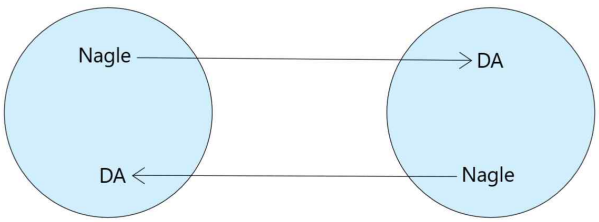
window update rule
: 흐름 제어, 자리가 났으니, 못 보내는 것이 있으면 이리로 보내라
ACK는 두 가지 정보를 가지고 있다.
1. ACK number: 어디까지 받았는지 알려준다.
2. Window size: 패킷이 아무것도 안 와도 Window size가 바뀌면 ACK를 보낼 수 있다.

왜 두 패킷마다 하나의 ACK가 오는가?
-> 어느 정도 Window size의 변화가 있어야 ACK를 보낼만 하지 않겠는가?
-> 그게 2*MSS이다.

예를 들어 매우 부지런한 응용이 있다고 가정하자.
그렇다면,
1. Delayed ACK Algorithm에 의해서 ACK가 뭉개지고,
2. window update에 의해서 window size가 2MSS이상이 되면 강제로 ACK가 나가게 된다.
그리고 1.의 Timer가 override되고, 2MSS만큼의 공간이 더 생겼다는 것을 알게 된다.

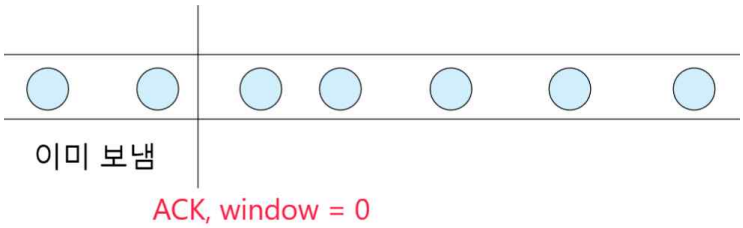
ACK가 두 개의 패킷에 대하여 1개가 온다 -> 부지런한 응용이다.
ACK가 여러 패킷에 대하여 1개가 온다 -> 게으른 응용이다.

-Nagle`s Algorithm-
: 응용으로부터 MSS이하인 데이터가 오면
-> 보내지 않고, 계속 모은다. ACK가 와야지 보낸다.
Nagle`s Algorithm은 끄고 켤 수 있다. Game처럼 즉각적인 반응이 중요한 것은 꺼야한다.

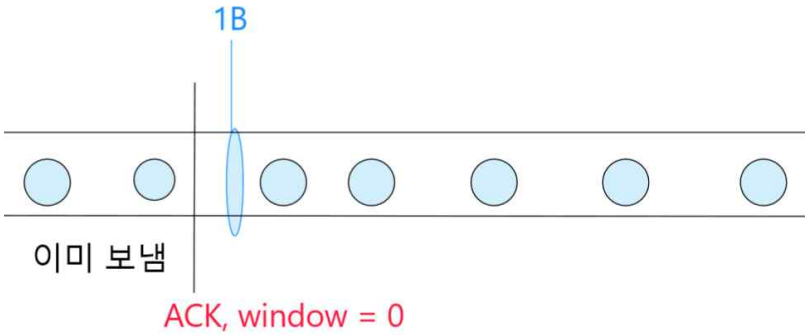



갔다 오는 길이 길면, Nagle은 효과적임
갔다 오는 길이 짧으면, Nagle은 그닥 효과적이지 않다.
(ACK가 빨리 와서 모을 시간이 거의 없다.)

-Zero window ACK and persist timer-
sender가 Zero window ACK를 받았다고 하자.




-> 어떻게 해결하나?
-> receiver의 응용이 segment를 퍼가서 2MSS 이상이 되어 ACK가 나가야 한다.
그러면 ACK number는 그대로, window size는 바뀌어서 ACK가 나간다.
그러나, window size가 변했다는 내용을 실은 ACK가 가다가 죽어버리면 어떻게하나?
-> Freeze. 아무것도 안 한다. 심지어 ACK는 재전송도 안한다.
-> 이를 해결하기 위해 예외를 준다.




window 이후의 data를 1B보낸다. (원래는 flow control 때문에 안됨)
만일 받으면, SEQ#가 1증가, 변경된 window size를 보냄 ()
-> ACK number, window size 변경
else, 아니면 다시 zero window를 ACK로 보냄.
-> 꽉 차있으면, ACK number, window size 그대로
window size 2MSS 이하이면, silly window syndrome 때문에
-> ACK number변경, window size=0인 ACK가 나감.