<인터넷 프로토콜>

작성자_2018320161_송대선
작성일_05_28

엄청 좋은 질문:
1. SEQ# = expected
다음에 받을 것을 ACK

2. SEQ# > expected
SACK, Cummulity ACK 뭐시기 한다.
일단 온 것은 다른 곳에 저장한다.

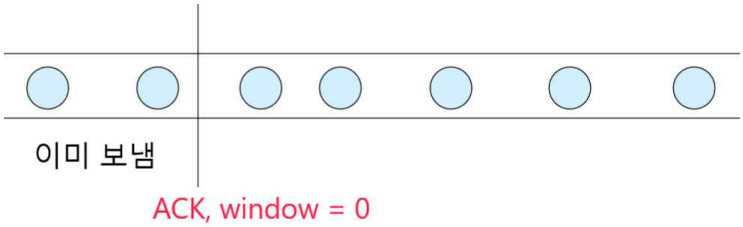
3. SEQ# < expected
가장 최근에 받은 segment에 대한 ACK를 보냈는데, 그 ACK가 죽은 것이다.
-> 일단 받은 segment는 버린다.
-> 그리고 delay없이 바로 지금까지 받은 것에 대한 ACK를 진행한다.

대형서버같은 친구들은 이것을 일부러 사용하기도 한다.
client가 이미 받은 segment를 보내본다.
-> 바로 ACK가 오면 살아 있는 것이다
-> else 같은 segment를 몇 번 다시 보내보고, 그래도 반응이 없으면 정리한다.
(half-open) 상태였던 것이다.

-Timers-
1. retransmission timeout
RTO = avg(RTT)+4*std(RTT)
(비슷한 기능으로 3개의 duplicated ACK가 오면 ACK를 보낸다.)

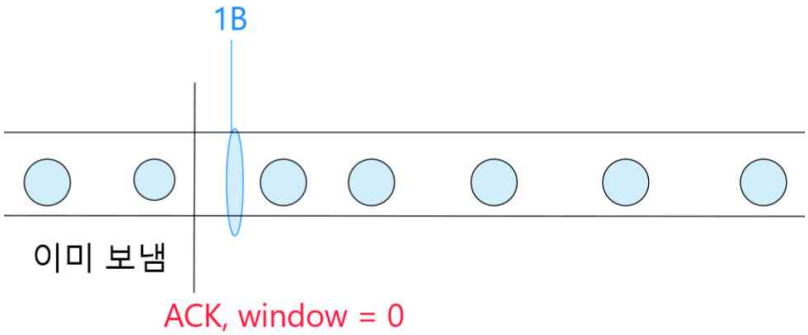
2. Delayed ACK algorithm
수신자가 매우 부지런하여 바로바로 데이터를 떠내가는 경향이 있다고 가정할 때,
2MSS만큼 꺼내면 ACK가 바로 나간다.

3. Persist timer (오늘 배울 것)
sender가 Zero window ACK를 받았다고 하자.



-> 어떻게 해결하나?
-> receiver의 응용이 segment를 퍼가서 2MSS 이상이 되어 ACK가 나가야 한다.

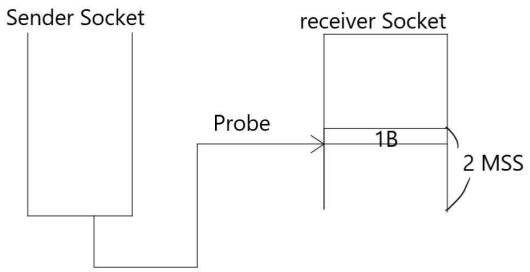
그러나, window size가 변했다는 내용을 실은 ACK가 가다가 죽어버리면 어떻게하나?
-> Freeze. 아무것도 안 한다. 심지어 ACK는 재전송도 안한다.
-> 이를 해결하기 위해 예외를 준다.



window 이후의 data를 1B보낸다. (원래는 flow control 때문에 안됨)
만일 받으면, SEQ#가 1증가, 변경된 window size를 보냄 ()
else, 아니면 다시 zero window를 ACK로 보냄.

-Silly Window Syndrome-
찔끔찔끔 보내고 읽는 일은 좋은 일이 아니다. Update가 너무 잦으면 오히려 귀찮다.
-> 그래서 오직 2MSS 이상의 빈공간이 없으면 window size를 0으로 표시해서 보낸다.
심지어 ACK해줘야 하는 상황에 진짜 Window size를 안알려주기도 한다. (0으로 표시)
-> 만일 그렇게 하지 않으면 Sender는 “Small” segment를 만들어 보낼 것이기 때문!
(그걸 안 만드는 일에 Receiver도 도와준다.)
-> 인터넷은 불신사회이기 때문에 상대방이 미친놈이라고 해도, 나는 제대로 해야 한다.
(보낼 때는 보수적으로, 받을 때는 진보적으로)
같은 차원으로, Sender에는 Nagle이 돌기도 한다.

사실



Persist timer가 돌고 있는 상황에, 사실 Receiver window에 정확히 2MSS만큼 남아있었는데, 1을 추가해버려서 2MSS-1이 되면 SEQ#를 1증가 시키고, window size는 0인 ACK를 보낸다.

다시 persist timer로 돌아가자.
persist timer가 돌고 있다는 것은 outstanding packet이 없다는 의미이다.
-> 재전송할 것이 없다 ->전부 ACK 받았다.
이때 persist timer는 그냥 RTO를 사용한다. (재전송 할 것이 없으므로!)
-> 두 개의 타이머가 따로 존재해야 할 이유가 없다.

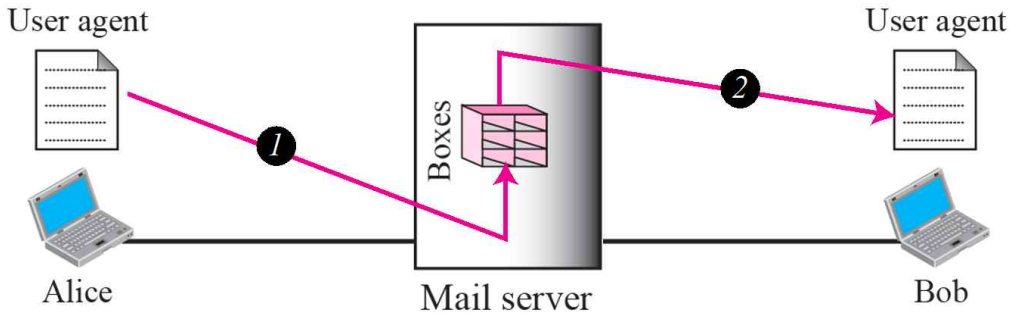
4. Keepalive timer
-> half open 상태를 확인한다.
-> server가 가끔 client를 찔러본다.
-> receiver가 이미 받은 SEQ#를 data없이 header만 보낸다.
-> 이걸 probe라고 한다.

Keepalive timer의 prove VS Persist timer의 probe
1. Keepalive timer은 data가 없다. VS Persist timer은 data가 1B만큼 있다.
2. Keepalive timer의 prove의 SEQ#은 지금 sender의 SEQ#보다 작다.
VS Persist timer의 prove의 SEQ#은 지금 sender의 SEQ#와 같다. (같은가? 질문.)

Persist timer의 probe는 정상적으로 delay된다. -> “==”에 해당됨.

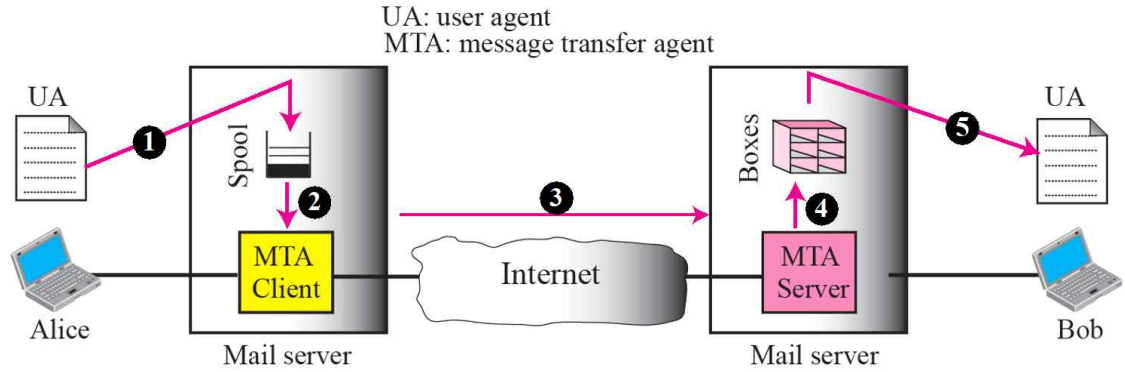
-SMTP- (Simple Mail Transfer Protocol)

Senario #1



-> 이 경우는 두 개의 UA만 있으면 된다.

Senario #2



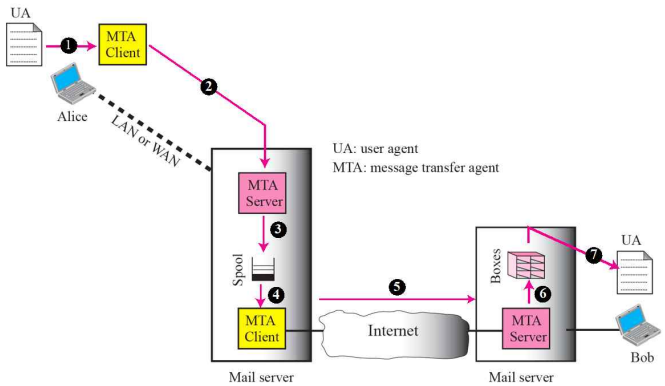
-> 이 경우는 두 개의 UA와 한 개의 MTA pair가 필요하다.

Spool은 일종의 queue이다.

Alice에서 Mail server 사이에도 라우터가 있고, 인터넷이 있다.

MTA Client(보내는 놈)-> MTA Server(받는 놈)

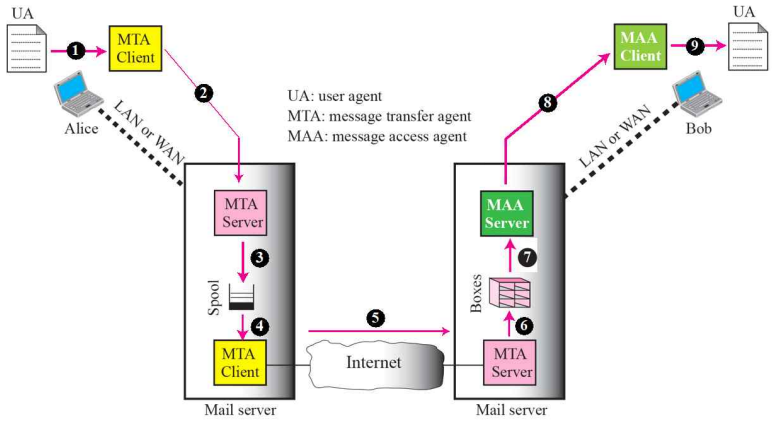
Senario #3



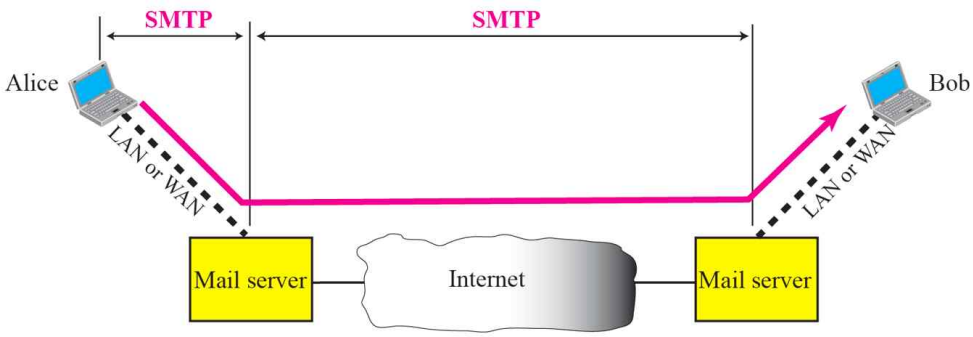
-> 이 경우는 두 개의 UA와 두 개의 MTA pair가 필요하다.

-> 한쪽에는 MTA server로, 다른 쪽에는 MTA client로 작동하는 Mail server가 필요하다.

Senario #4



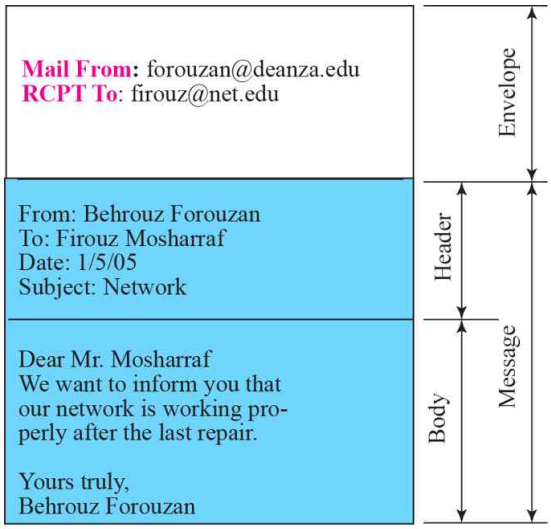
-> 이 경우는 두 개의 UA와 두 개의 MTA pair, 한 개의 MAA pair가 필요하다.
-> 제일 일반적인 형태이다.
-> MTA는 push만 할 줄 알고, pull을 몰라서 pull을 할 줄 아는 MAA이 필요하다.



응용: SMTP
transport: TCP (port=25)
network: IP

MTA Client가 맨 처음으로 하는 일은 @ 뒷부분을 MX를 이용하여 DNS서버에게 Mail server의 Domain Name을 물어본다. (그 response의 additional에 IP주소가 같이 들어있다.-> A로 다시 물어볼 필요가 없다.)
->그 Domain Name을 이용하자(IP 주소도 딱히 상관 없다.)
명령어 telnet mx.hamail.net 25
-> 메일 서버가 뜬다.
여기서 내가 암것도 안하고 가만히 있으면 time out으로 연결이 끊어진다.
->TCP Reset이다.

-email format-



-> 실제 편지와 유사한 구조를 가진다.

-> 이메일 주소에서 local part는 사실 mailbox의 이름을 의미한다.

명령어 telnet mx.hamail.net 25
HELO babo.com (3 hand shake)
MAIL FROM: hyogon@korea.ac.kr
-> 내 고객이 누구누야
RCPT TO: hyogon
-> 너희 고객 mailbox중에 hyogon이라는 이름을 가진 고객이 있니?

invalid이면-> 그런 고객 없어! 하고 연결을 끊어버림

Commands는 HELO, MAIL FROM, RCPT TO등이다.
Responses는 반응인데, "503 5, 5, 1 ~~~~"이런 식으로 온다.

200단위: 성공적으로 이루어짐
300단위: 중간에 거쳐가는 과정
400, 500단위: error
-> 사실 서버는 숫자들로만으로 통신가능
이 form이 전통적인 통신 폼이다.
STMP에서 HTTP가 만들어지고, HTTP에서 SIP이 만들어졌다.