03/30 – III. Transport 계층

window size
sender 입장에서는 window size를 지정해, 해당 사이즈만큼은 한 번에 쏟아부을 수 있다.

Pipelined Protocol의 두 가지 방식
1. go-Back-N 방식
윈도우 사이즈가 정해져 있고, 각각의 패킷을 한 번에 전송하는데
예를 들어 0번 패킷에 대한 ACK0이 오기 전에 타이머가 터지면 0번 이후, 윈도우 사이즈까지에 해당하는 패킷을 모두 재전송한다.
리시버는 버퍼도 없고 아무것도 없어서 받아야 하는 패킷만 주구장창 기다린다. 만약 받아야 하는 패킷이 1번이라면, 이 패킷이 오지 않고 다음 패킷이 올 때 그 패킷을 버리고 ACK0만을 계속 보낸다.

센더는 ACKO을 받으면 윈도우를 1칸 전진시킨다. 즉 ACK를 받은 칸까지만 윈도우를 전진시키고, 중첩시켜 패킷을 계속 보낼 수 있다.
예를 들어 window size가 4였다면 초기 패킷은 0-3번이 갔지만, ACK0을 받은 이후 window는 1-4가 된다.

패킷이 유실되면, 유실된 것 기준으로 N개만큼 돌아와서 Go-Back-N이다. 근데 리시버가 아무것도 안하니까 이것도... N번 패킷이 사라지면 그 다음 패킷도 죄다 재전송해서...

2. selective repeat방식
1에서 사라진 것들만 재전송한다. 그러려면 ACK N의 의미가 N번까지 다 받았다가 아니라, N번만 잘 받았다가 되어야함. 이때 버퍼에 받은 패킷을 넣어두었다가 순서를 맞추고 나서 ACK를 전송한다. 이때 ack는 cumulative하지 않고 selective하다. 또 수신 측도 window를 가진다.
네트워크에 부담은 덜한데... 얘도 문제가 있음. 만약 window-size가 3일 때 0,1,2를 잘 받아서 ACK(0), ACK(1), ACK(2)를 보냈는데, 이 ACK가 모두 유실되었다면 sender는 P(0, 1, 2)를 다시 보낼 것임. 근데 receiver는 duplicated를 3번 패킷인 줄 알고 받게 된다. 와우.. 그래서 시퀀스 넘버를 계속 돌려쓰면 안 되고 충분히 커야 한다. 그 범위는 2배 언저리 아닐까?

03/30 – III. TCP(진짜로)

APP – MESSAGE (편지지)
TCP – SEGMENT (편지봉투)
IP – PACKET (더 큰 편지봉투)
LINK – FRAME (서류봉투)
PHY

아무튼 상위의 DATA + HEADER는 하위 전송 단위의 DATA에 들어간다.

TCP SEGMENT STRUCTURE

SRC. PORT# DEST. PORT#
SEQ# – DATA의 가장 첫 번째 바이트 번호. 예를 들어 100바이트의 데이터가 전송되어야 하고 window size가 10이면, SEQ#은 0번, 10번, 20번, ...가 된다
ACK# – Go-Back-N처럼 cumulative하다. ACK10 == 10번 달라는 뜻. (Go-Back-N은 10번 받았다) 또, ACK#는 리시버가 어디까지 따라왔냐에 따라 붙는다. 예를 들어 10번부터 17번까지 갔다고 할 때 ACK#는 12번이 될 수 있다.
PSH –
CHECKSUM

ERTT=(1-a)ERTT + a(sampleRTT)
typically a=0.125

DRTT = (1-b)DRTT + b(sampleRTT – ERTT)
typically b=0.25

timeoutInterval = ERTT + 4DRTT

TCP: Overview
1. Point – to – point
receiver 하나, sender 하나. 소켓 한 쌍의 통신을 책임진다. 프로세스 하나에 여러 개의 소켓을 열 수 있기 때문에..
2. reliable, in-order byte stream
하나도 유실되지 않고, 순서대로 간다
3. pipelined
한꺼번에 쏟아붓는다
4. send & receive buffers (window)
window의 개념으로, 패킷을 주고받는 데 쓰는 각각의 버퍼가 양 쪽에 모두 존재한다. send buffer/receive buffer를 각각의 소켓이 가지고 있다는 의미. 한 쌍의 send-receive buffer는 사이즈가 같다.
5. full duplex data
receiver와 sender는 각각 역할이 구분되는 것이지만, 소켓 한 쌍은 sender이면서 receiver이다.
소켓 한 쌍이 sender이면서 receiver이기 때문에 데이터가 양방향으로 오고 간다.
6. connection-oriented
handshacking
7. flow-control
윈도우 크기만큼 붓긴 하는데, receiver가 받을 수 있는 만큼 보낸다.
8. congestion control
외부 네트워크가 받아들일 수 있을 만큼만 붓는다.

근데 양쪽 다 sender이면서 receiver면 어떻게 ack와 데이터를 보내는지? 사실 양쪽 다 데이터를 보내면서 같이 ACK를 보내면 좋은데, 그러지 말고 500ms 이후에 ACK를 보내주라고 권고함. Cumulative ACK이기 때문에 일일히 ACK하지 말고, 좀 기다렸다가 받은 것까지 한 번에 보내라는 의미.

segment가 유실되면 타이머로 그걸 판단한다고 했는데, 유실을 확신하면서 timeout value를 작게 가져가기 위해, 어떤 segment가 목적지에 가서 다시 돌아오기까지의 시간(Round trip time)을 측정한 후 이걸 timeout value로 지정해 보자. 문제는 이게 계속해서 변하는 값이어서... (세그먼트가 지나가는 경로가 다를 수 있고, 같더라도 Queueing delay에 따라서 변할 수 있음) estimated RTT를 도입해 보정함. 근데 그래도 이러면 진짜 너무 많이 나니까 여기다 마진(deviation)을 붙여서... 구함.

03/30 – III. TCP(진짜로)

APP – MESSAGE (편지지)
TCP – SEGMENT (편지봉투)
IP – PACKET (더 큰 편지봉투)
LINK – FRAME (서류봉투)
PHY

아무튼 상위의 DATA + HEADER는 하위 전송 단위의 DATA에 들어간다.

TCP SEGMENT STRUCTURE

SRC. PORT# DEST. PORT#
SEQ# – DATA의 가장 첫 번째 바이트 번호. 예를 들어 100바이트의 데이터가 전송되어야 하고 window size가 10이면, SEQ#은 0번, 10번, 20번, ...가 된다
ACK# – Go-Back-N처럼 cumulative하다. ACK10 == 10번 달라는 뜻. (Go-Back-N은 10번 받았다) 또, ACK#는 리시버가 어디까지 따라왔냐에 따라 붙는다. 예를 들어 10번부터 17번까지 갔다고 할 때 ACK#는 12번이 될 수 있다.
PSH –
CHECKSUM

ERTT=(1-a)ERTT + a(sampleRTT)
typically a=0.125

DRTT = (1-b)DRTT + b(sampleRTT – ERTT)
typically b=0.25

timeoutInterval = ERTT + 4DRTT

TCP Reliable Data Transfer Logic
1. IP의 unreliable service를 기반으로 rdt 구현
2. 파이프라인 방식
3. cumulated ACK 사용
4. 하나의 타이머 사용, 하지만 그 이후를 다 재전송하지 않음 (그 세그먼트만 재전송함)
4-1 seq 92에 대한 ACK가 유실된 경우 -> seq 92를 재전송
4-2 seq 92를 보내고, seq 100에 대한 데이터도 보냈는데 ACK92가 늦게 온 경우 -> seq092를 재전송. ACK100이 온다면 그 다음에는 sender는 100번까지 쭉 밀고 100번을 보낸다.
4-3 4-2에서 ACK92가 loss될 경우 ACK100이 loss되지 않은 경우 -> sender는 100번까지 쭉 밀고 100번을 보낸다.

4-2, 4-3같은 경우 때문에 데이터를 받고 나서 ACK를 즉시 할 필요가 없는것. TCP ACK generation에서는 그래서 500ms 대기탔다 주라고 하는것...

Fast retransmit
타이머가 터져야 segment의 유실을 판단할 수 있는데, 사실 안 터져도 판단할 수 있지않나? 만약 그 세그먼트가 안 오고 다음 세그먼트가 왔다면, 즉 ACK10이 연속으로 왔다면 10번이 없는것. 이 횟수를 4번으로 권고하고 있다