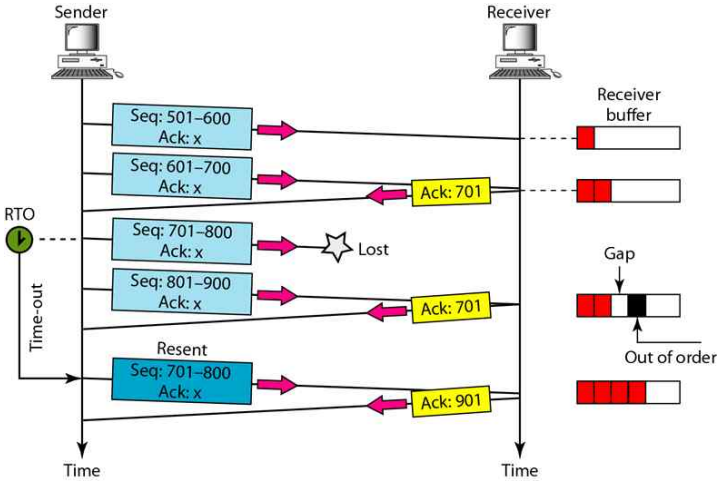
TCP Error control

Go-back N과 비슷한 점
1. 패킷 유실 확인을 타임아웃에 의존
2. ACK가 안온 최초의 패킷부터 시작해서 쭉 재전송. NAK사용안함. ACK가 cumulative(누적)한거는 동일한점.(ACK3을 보내면 3번을 기대 하고잇다. 1,2번은 잘받앗다 의미, 커버가능)
다른점
1. 순서가 뒤바뀌어서 들어온 segment에서 누락된 패킷이 들어올 때 까지 hold한다.(out-of-order segment)
2. sequence number가 0123(ID)을 대체한다.
3. Sender window와 receiver window가 다양한 사이즈. 적절한 사이즈로 조절하는게 tcp의 핵심

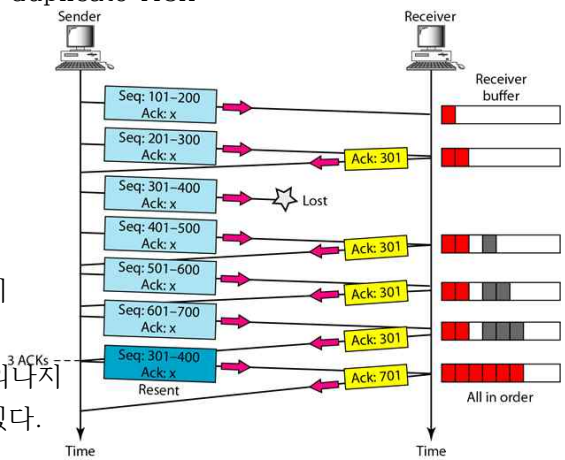


(out-of-order 상황)

RTT(Round trip time) : 세그먼트를 보내고 ACK가 도착하는데 걸리는 시간 : TCP의 경우 계산하기에 복잡하다.
이 Timeout period는 생각보다 길 수 있고 이거를 매번 기다리는것이 비효율적일수도 있다.
->이거를 개선하기 위한 방법(NAK사용없이..) : duplicate ACK

3-duplicated ACK
순서가 바뀌어서 온 Segment에 대해서
3번 연속으로 기대하지 않은 Segment가
도착하지 않으면 계속 ACK를 301로 보낸다.

효과: 타임아웃이 라운드 트립 타임보다 더 길게 설정되기 때문에 한 개씩 보내면 비효율적.
연속으로 세그먼트를 보내게 되면 시간차가 거의나지 않고 sender가 오류가 발생 했다고 상정 할수있다.



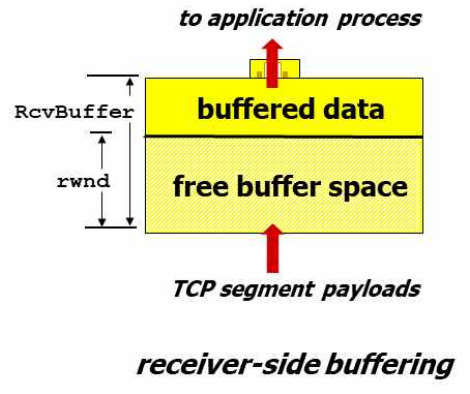
TCP Flow Control / Congestion Control

Flow control : sender가 receiver의 버퍼가 overflow 되는 것을 방지하기 위해 보내는 패킷 양을 조절하는 것. Reciever가 버벅거리면 window size를 작게 해주어야함.
Overflow : application layer가 버퍼의 데이터를 다 읽기도 전에 엄청 많은 데이터가 들어와서 결국 폐기한다.

rwnd : reciever가 size를 결정해야 한다. 이 사이즈로 정해진 Window size
Receiver가 seder에게 size 어캐 알려줄까 : 세그먼트에 보면 Window size가 잇는데 이게 sender에게 자신이 가용가능한 size를 알려주는 역할을 한다.
Clinent도, server도 sender이면서 receive이다. 따라서 둘 다 sender window와 receiver window가 잇다.

Flow control에서의 Window control : receiver가 자신이 가용한 버퍼의 크기를 알려주는 것 sender에게. 버퍼의 크기를 고려해서 sender window size 결정.

Flow control을 위한 Window size : TCP 헤더에 있는 Window size필드에 적어서 reciever가 sender에게 알려준다.
Reciever은 어떻게 자신의 가용 가능한 버퍼량을 계산할까? -> 운영체제가 해당 TCP 세션에 대해 할당해 놓은 가상의 버퍼를 상황에 따라서 결정.
Buffered data : 데이터가 차서 사용 불가한 부분. sender가 보내서 저장해 놨는데 application이 아직 읽지 않은 부분
Free buffer space : 사용 가능한 공간 -> rwnd로 set한다. 이것을 Tcp header에 적어서 보낸다.



Congestion : Queueing deley : 여러 개의 input port에서 들어온 데이터가 하나의 output port로 몰려 delay 발생 ->이런 상황 - congestion

Congestion Control : 혼잡 상황을 감지하고 혼잡 상황일 경우 Window 사이즈를 줄여준다.
Sender가 네트워크 혼잡 감지를 하면 조금만 보냄. 네트워크 사용자가 다같이 줄여준다.
이 경우에도 window size를 작게 해야한다.
Transmission rate= $\frac{W \times MSS}{RTT}$
Segment의 수 * Segment의 크기 / 시간(데이터 양) = Transmission rate

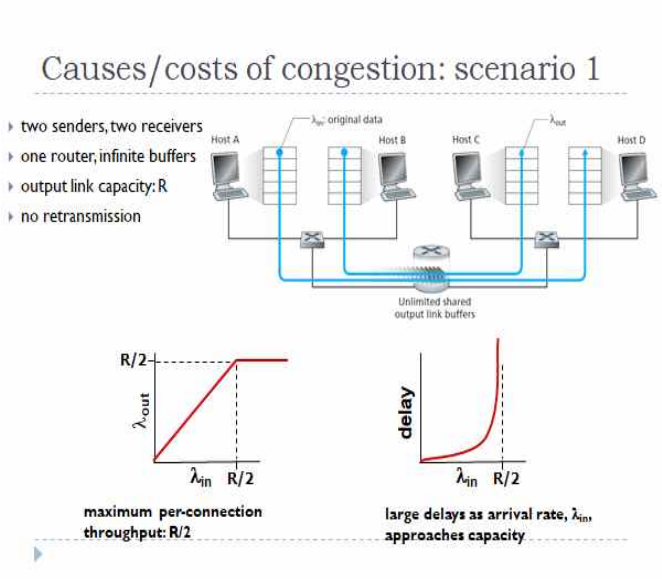
Congestion control : window size를 누가 정하냐?

Tcp의 Congestion 감지 방법 : Packet loss가 발생하면 Network가 혼잡 상태라고 유추
ACK가 안오면 혼잡이라고 가정한다. -> Window Size를 줄인다. cwnd까지.
cwnd : Congestion window size. 이것을 얼마로 할 것인가?
1000이니가 ack가 잘 안와 1000->100 줄엿으니 ack가 잘 와 그러면 조금씩 늘려 100->120->500->2000->4000 그랫더니 ack가 안와 그러면 또 확 줄여. packet이 loss될 때 줄인다. 소잃고 외양간 고치기. 안고치면 또 잃어버린다.

핵심 : Windows size는 min(cwnd,rwnd)로 결정.

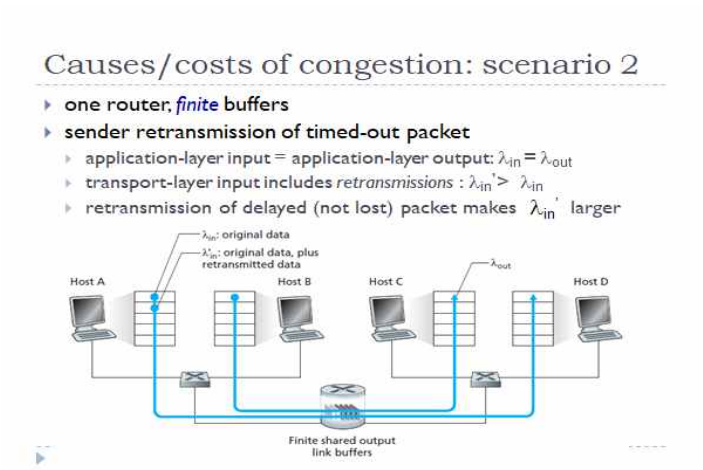
Congestion Control의 기본 원리

Congestion : 너무 많은 소스 들이 너무 많은 데이터를 네트워크가 처리하기 어려울 정도로 더 빨리 보내는 상황.
Short-term : 잠깐. 우연히 발생하는 congestion -> 이유)text 기반의 웹브라우징.
Internet의 트래픽이 잠깐 나타났다 사라지는 트래픽이기 때문(텍스트 기반) 버퍼를 충분히 확보함으로서 어느정도 해결 가능
Long-term : 데이터 congestion가 오랫동안 지속이 된다. -> 이 경우에는 sender가 transmission rate를 줄이는 것이 해결책.(window size를 줄인다) 이것 말고는 해결이 안 된다.
Congestion의 무서움 : 패킷loss, delay

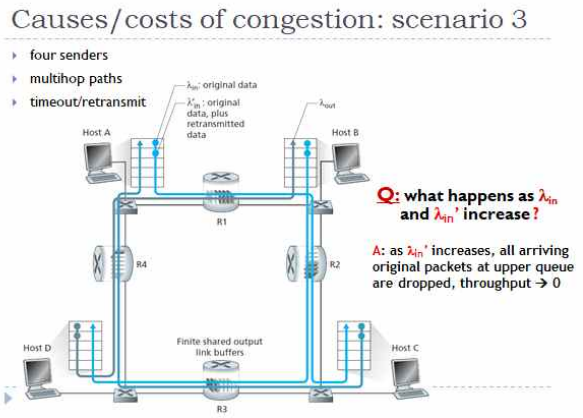


두개의 포트가 잇다. 왼쪽 1번 포트 오른쪽 2번 포트
1번 포트 에서 데이터가 들어옴 A와 B두개의 소스에서 들어온다. 2번 포트로 나가는데 두개의 TCP 세션이 하나는 C로가고 하나는 D로간다.
1)버퍼가 무한대인 상황. Congestion이 발생해도 lost packet이 없다.
1번과 2번 포트 모두 link capacity가 R이라고 하자. 이번에는 UDP를 사용한다고 가정해보자.
A와B의 transmission rate가 동일하다고 가정(ㅅin).
R/2까지는 문제없이 가고 R/2를 넘어가면 그 이상으로는 전송을 못 한다 왜냐하면 port2의 Link capacity가 R이기 때문에 넘어가면 segment가 무한대로 쌓이기 시작한다. 그래서 delay는 무한대가 된다.

그래서 해결방법은 A와B가 보내는 data양을 R/2이하로 조절하는 것. 그것이 TCP가 하는 역할.
평균 Transmission Rate가 R/2에 근접하면 delay는 기하급수적으로 커진다. R/2에 도달하면 무한대가 된다.(쌓인 데이터의 양==delay)
쌓인 data의 양으로 input rate를 결정한다.

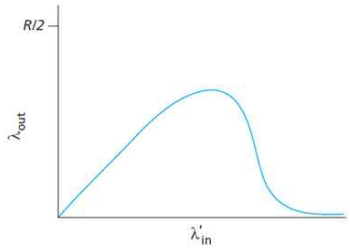


이 경우에는 버퍼가 유한하다 ==패킷 lost발생
Tcp 세션으로 restransmission을 한다.
Delay가 무한대로 늘어나진 않지만 loss가 발생하고 더 큰 data rate로 tcp가 보낸다.(restansmission때문.) 그래서 조절이 더욱 중요하다.



라우터 두개를 건너가는 상황.
R1이 버퍼가 꽉꽉 차면 패킷이 들어오고 D에도 영향 받아서 restransmission을 계속 요청. R4도 congestion . R3도 망가지고 다 망한다.
Congestion이 발생하면 하나가 망가지면 다 망가진다.
그래서 tcp는 매우 보수적인 방법으로 congestion 제어한다.

Throughput : sender에서 reciever까지 전달되는 data양을 time으로 나눈 것. 실제 전송률. Input data rate가 작을 때 는 같이 따라 올라가지만 어느 정도 되면 같이 망가지고 더 가면 0에 수렴하게 된다.(아무 데이터도 통과하지 못하는..)



Congestion Control 방식

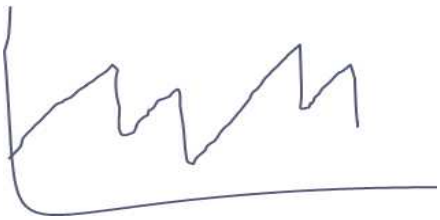
1.중간에 잇는 라우터 도움 없이 sender와 reciever의 handshaking만으로 congestion제어.

2. 간단하다는 장점. 네트워크의 의존성 X, 네트워크로부터의 정보는 없다. Sender와 reciver간의 정보교류만으로 제어. 네트워크의 도움이 필요없이 때문이다.

2번 방법. 네트워크의 중계노드(라우터) 같은 애들이 적극적으로 도와줌. 라우터가 꽉 찼다고 얘기 해준다. 나 꽉 찼어라는 field를 만들어두고 1bit짜리 1이면 찼어 0이면 ㄱㅊ아 방식 나지금 버벅 거리고 잇으니 이속도 이상으로 보내지마 라고도 signal 보내는 방식도잇음. 근데 이 2번방법은 좋은데 우리가 사용하는 인터넷에서는 사용하지 않는다. 왜냐하면 이미 tcp가 자리를 잡고 잇기 때문이야. 이미 자리잡은 기술을 퇴출시키기가 매우 어려워.

TCP Congestion Control 문제점 : Congestion만으로 packet loss가 발생하는 것이 아니라는 문제점. 무선 인터넷에서 옆 사람과의 충돌 등에 의해서도 발생하기 때문

A. Original Concept TCP



별일 없으면 (ACK 가오면)Window size가 additive하게 올라간다. 하나의 segment size만큼. Packet Loss가 발생하면 그다음에 window size를 반으로 줄인다. 그리고 loss가 발생할 때 까지 또 additive하게 증가한다. 이게 너무 보수적이라는 의견이 나왔다. 그래서..

개념 : Slow Start

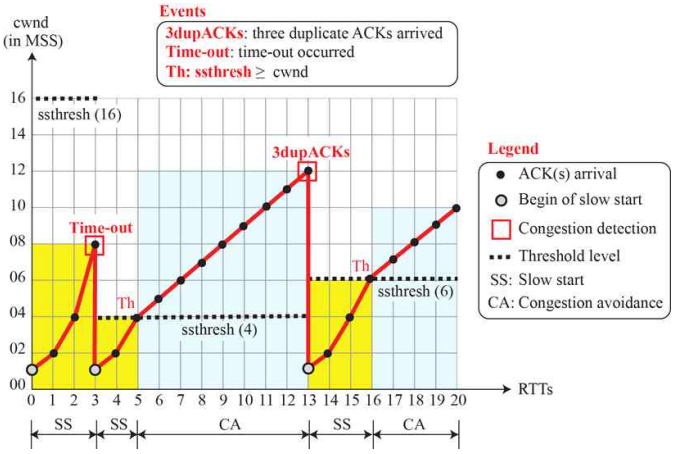
그 후에 여러 가지 제안 중 채택된 것. Additive increase가 너무 시간이 많이 걸리니 두 배씩 많이 보내자.(Window size를 두 배씩 늘리자) exponential하게 증가한다.

이 메커니즘을 Slow start라고 부른다. 이름과는 달리 전혀 일치하지 않는다.

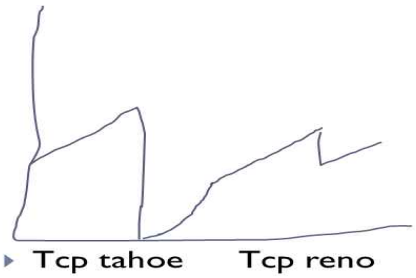
B. TCP Thao

1. Congestion 감지 시(timeout, 3-duplicate ACK) 현재 윈도우 값의 ½를 threshold로 저장한 후. 1로 줄인다. 그다음 Slow start (exponential increase)를 threshold까지.

3. 그리고 나서 addictive increase를 한다.



C. TCP Reno



Duplicate ACK 발생 : Tcp가 201번 받고 401번 501번 601번 받으면 3개의 duplicate ACK를 보낸다. 그러면 세개만 loss 됫다고 가정. Congestion이 그렇게 심각하지 않다 라고 생각 할수 있음. 이 경우에는 반으로 줄이고 addictive하게 늘이는 방법이 있음.
Timeout 발생 : Window size를 1로 줄임

Additive increase의 문제점이 무엇인가? 이 대신 생각 해 볼 수 잇는 기술이 뭐가 있을까??