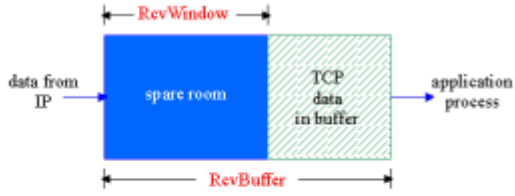
전송계층_3, 4

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP Flow Control

- receive side of TCP connection has a receive buffer:



- app process may be slow at reading from buffer

flow control
sender won't overflow receiver's buffer by transmitting too much, too fast

- speed-matching service: matching the send rate to the receiving app's drain rate

TCP Flow Control

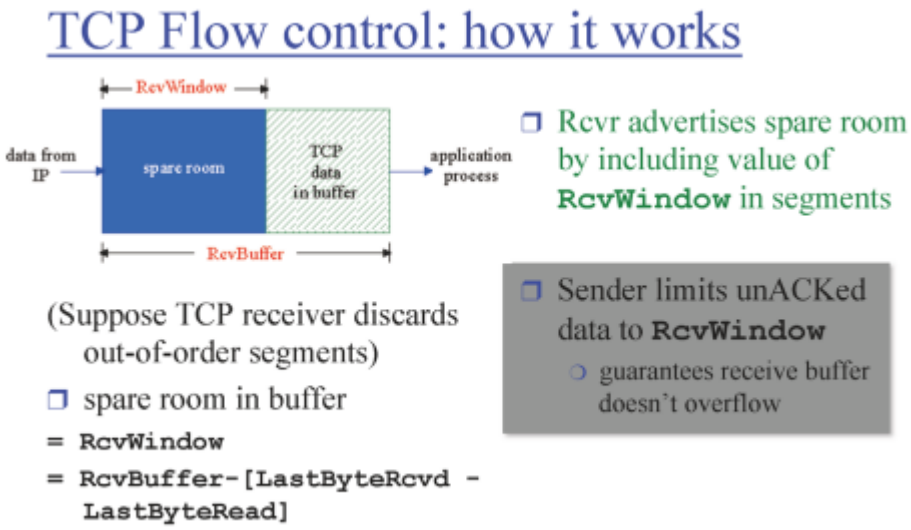
TCP(주 프로세스 사이에 데이터를 주고받는 것)

- 흐름 제어: 발신자는 너무 많이, 너무 빨리 전송하여 수신자의 버퍼를 오버플로우하지 않게함.(보내는 양 조절)

TCP segment의 header부분에 receiver buffer 태그에 담겨서 감

- TCP 연결의 수신측에 수신 버퍼가 있음.

- 앱 프로세스가 버퍼에서 읽는 속도가 느릴 수 있음
- 속도 일치 서비스:
  수신 앱의 전송 속도와 일치하는 전송 속도



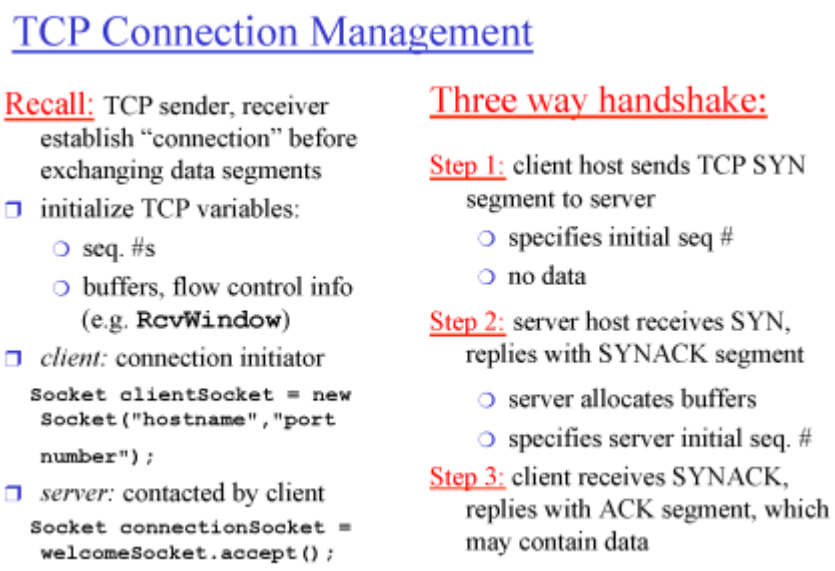
TOP Flow Control: how it works(작동원리)

(TCP 수신기가 순서가 잘못된 세그먼트를 버린다고 가정합니다.)

- 버퍼의 여유 공간

  = RcvWindow

  = RcvBuffer - [LastbyteRcvd - LastbyteRead]
- Rcvr은 세그먼트에 RcvWindow의 값을 포함하여 여유 공간을 알림.
- 발신자는 ACK되지 않은 데이터를 RcvWindow로 제한
  - 수신 버퍼가 오버플로되지 않음을 보장



TCP Connection Management(TCP 연결 관리)

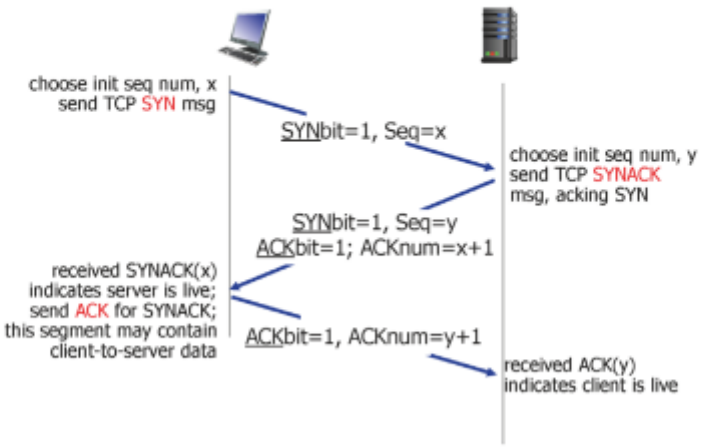
Recall : TCP 송신자, 수신자는 데이터 세그먼트를 교환하기 전에 "연결"을 설정

- TCP 변수 초기화
  - seq 넘버
  - 버퍼, 흐름 제어 정보
- 클라이언트 : 연결 개시자
- 서버: 클라이언트에서 연락

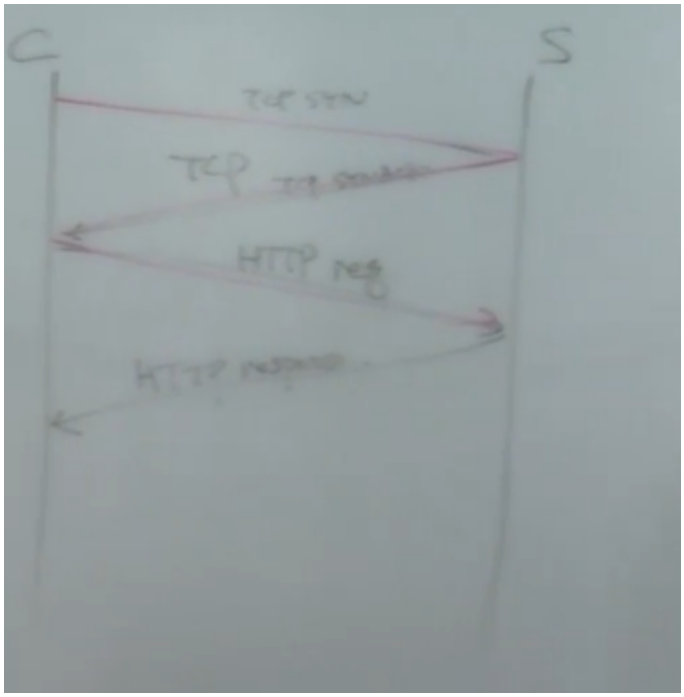
Three way handshake

- step1 : 클라이언트 호스트는 TCP SYN 세그먼트를 서버로 보냄.
  - 초기 시퀀스 번호를 지정
  - 데이터 없음
- step2: 서버 호스트는 SYN을 수신하고 SYNACK 세그먼트로 응답
  - 서버가 버퍼를 할당
  - 서버 초기 시퀀스 번호를 지정
- step3: 클라이언트는 SYNACK을 수신하고 데이터를 포함할 수 있는 ACK 세그먼트로 응답

TCP 3-way handshake



- 초기 시퀀스 번호를 선택하고, x TCP SYN 메시지 보내기
- 초기 시퀀스 번호를 선택하고, y TCP SYNACK 메시지를 보내고 SYN을 확인
- 수신된 SYNACK(x)는 서버가 활성 상태임을 나타냅니다. ; 이 세그먼트에는 클라이언트-서버 데이터가 포함될 수 있습니다.
- 수신된 ACK(y)는 클라이언트가 활성 상태임을 나타냅니다.



Closing TCP Connection

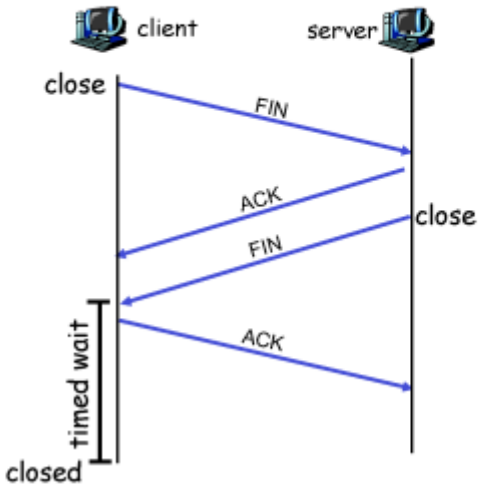
Closing a connection:

client closes socket:

```
clientSocket.close();
```

Step 1: client end system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.



Closing TCP Connection

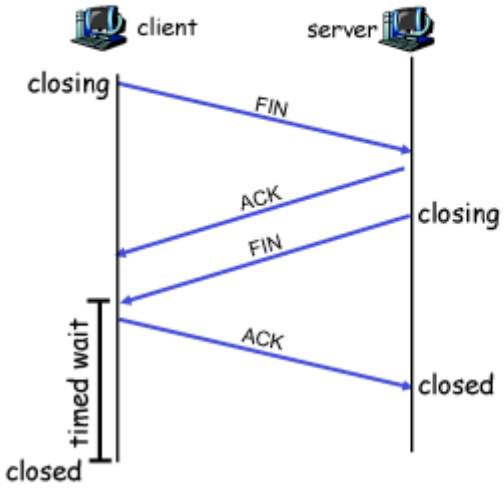
- step1 : 클라이언트 측 시스템은 TCP FIN 제어 세그먼트를 서버로 보냄
- step2 : 서버는 FIN을 수신하고 ACK로 응답. 연결을 닫고 FIN을 보냄.

TCP Connection Management (cont.)

Step 3: client receives FIN, replies with ACK.

- Enters "timed wait" - will respond with ACK to received FINs

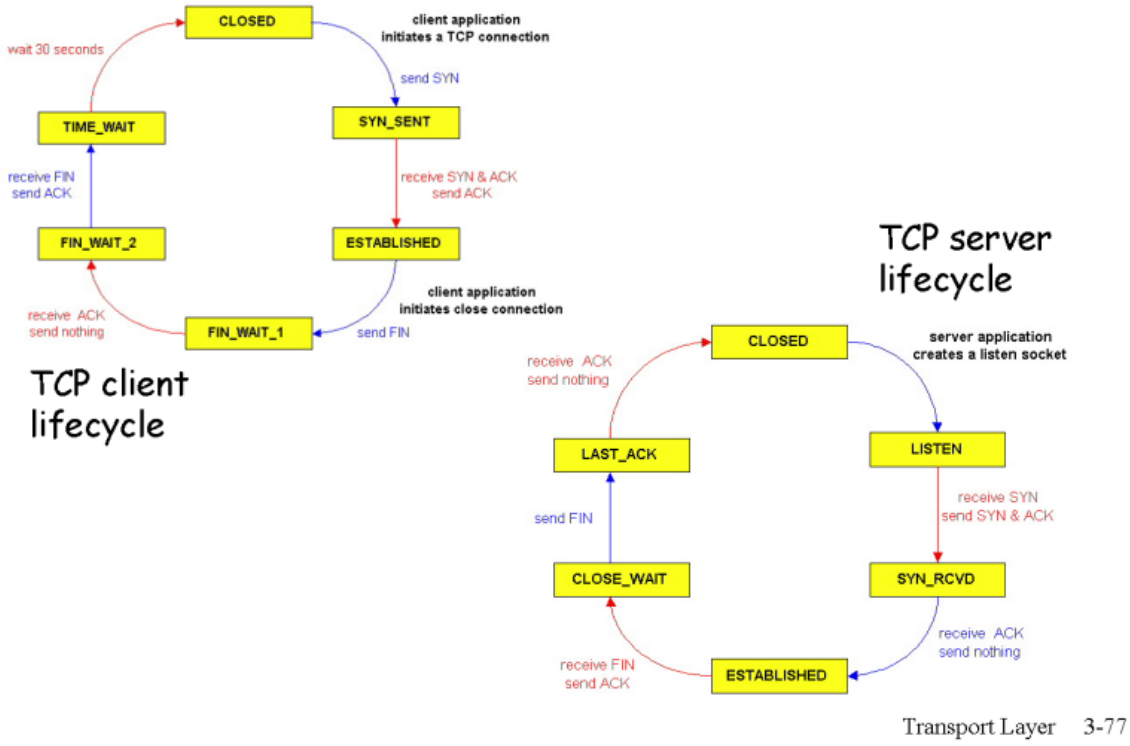
Step 4: server, receives ACK. Connection closed.



- step3 : 클라이언트는 FIN을 수신하고 ACK로 응답
  - "timed wait" 진입 - 수신된 FIN에 ACK로 응답

- step4 : 서버는 ACK를 받습니다. 연결 닫힘

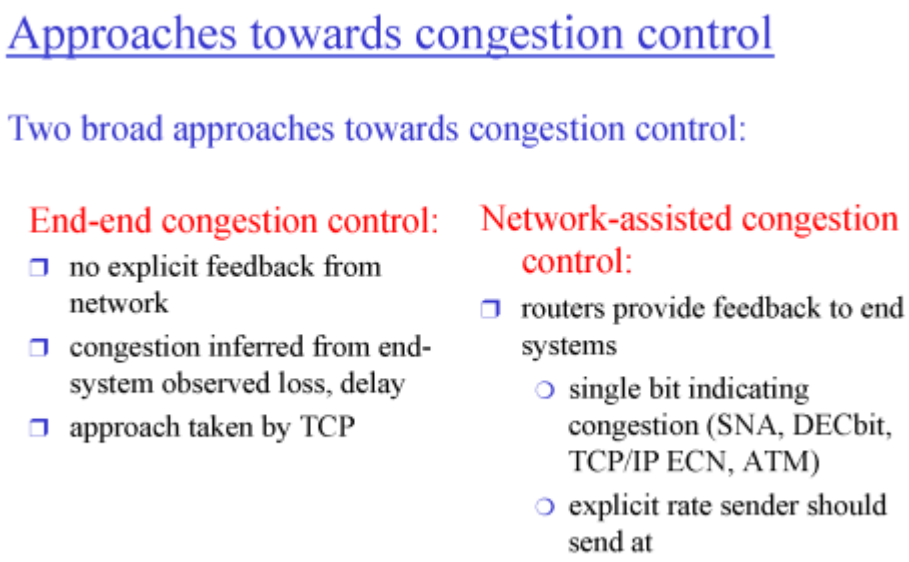
TCP Connection Management (cont)



Transport Layer 3-77

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control



Approaches towards congestion control(혼잡 제어에 대한 접근)

Two broad approaches towards congestion control(혼잡 제어에 대한 두 가지 광범위한 접근 방식)

End-end congestion control

- 네트워크에서 명시적인 피드백 없음
- end시스템에서 관찰된 손실, 지연에서 추론된 혼잡
- TCP의 접근 방식

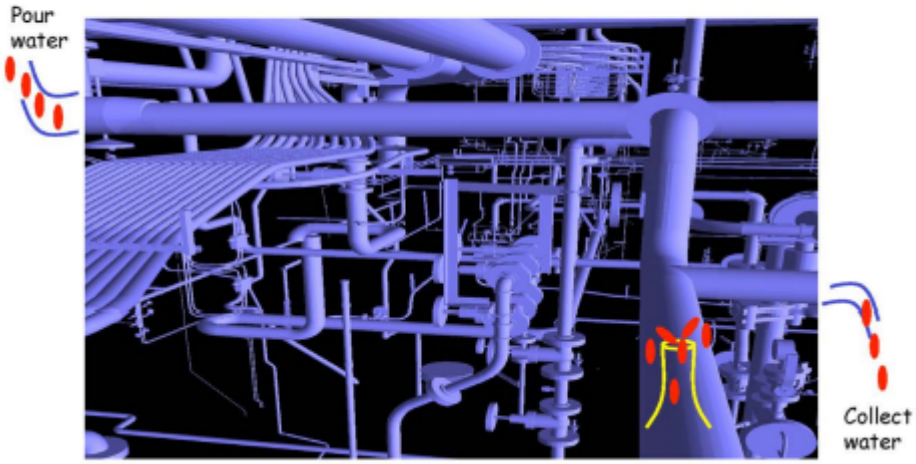
Network-assisted congestion control(네트워크지원 혼잡 제어)

- 라우터는 최종 시스템에 피드백을 제공
  - 혼잡을 나타내는 단일 비트
  - 명시적 요율 발신자는 다음으로 보내야 합니다.

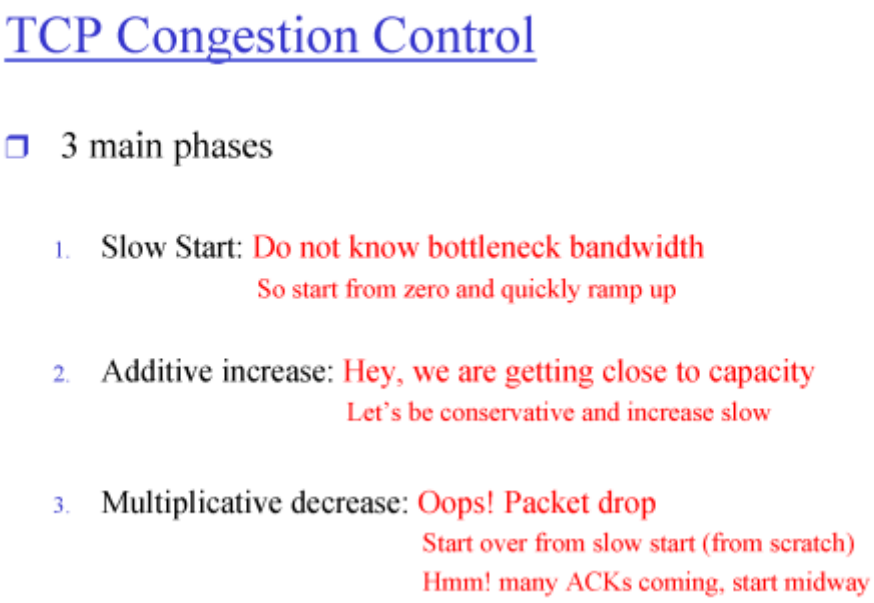
Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

The TCP Intuition



The TCP Intuition



TCP Congestion Control

- 3 주요 단계
  1. 느린 시작 : 병목 대역폭을 모른다

     따라서 0에서 시작하여 빠르게 증가(1, 2, 4, 8)
  2. 가산 증가 : 우리는 용량에 가까워지고 있다.

     보수적으로 천천히 늘리자(1, 2, 3, 4 …)
  3. 곱셈 감소 : Packet drop

     느린 시작에서 다시 시작(처음부터)

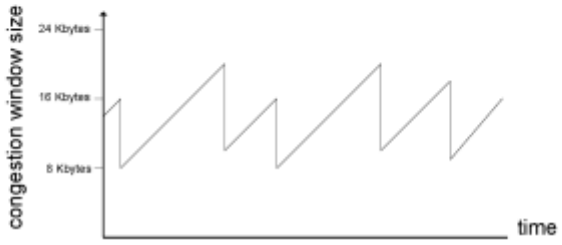
     많은 ACK가 오고 있으니, 중간에서 시작

- MSS(Maximum Segement Size) ⇒ 변화하는 단위
- RTT(RounTing Time) ⇒ 갔다 오는 시간

TCP congestion control: additive increase, multiplicative decrease

- *Approach:* increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **CongWin** by 1 MSS every RTT until loss detected
  - *multiplicative decrease:* cut **CongWin** in half after loss

Saw tooth behavior: probing for bandwidth



- x축 = time, y축 = window size

additive increase, multiplicative decrease(가산 증가, 곱셈 감소)

- 접근 : 손실이 발생할 때까지 전송 속도(창 크기) 증가, 사용 가능한 대역폭 탐색
  - additive increase:손실이 감지될 때까지 RTT마다 CongWin을 1MSS 증가
  - multiplicative decrease : CongWin 자른 후 절반으로 축소

TCP Congestion Control: details

- sender limits transmission:
  `LastByteSent-LastByteAcked ≤ CongWin`
- Roughly,

$$\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- **CongWin** is dynamic, function of perceived network congestion

How does sender perceive congestion?

- loss event = timeout *or* 3 duplicate acks
- TCP sender reduces rate (**CongWin**) after loss event

three mechanisms:

- AIMD
- slow start
- conservative after timeout events

- CongWin(Condition Windowsize) ⇒ 네트워크 상황에 따라 결정됨

details

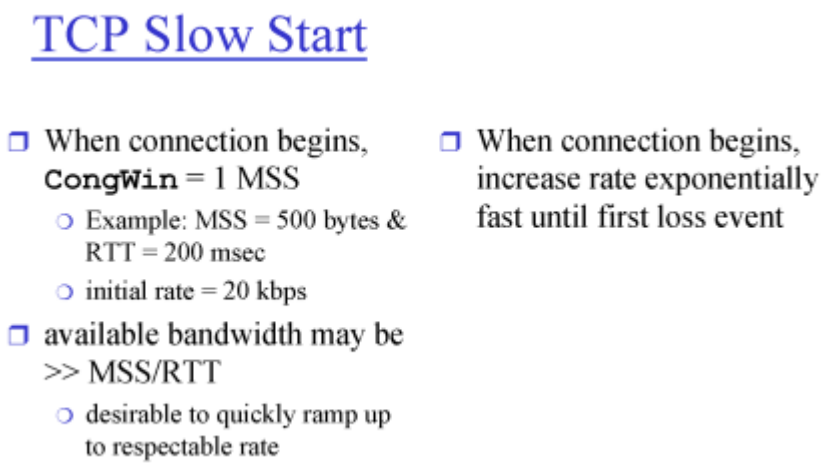
- 발신인이 전송을 제한
- CongWin은 동적이며 인지된 네트워크 혼잡의 기능
- ex) 네트워크가 많이 붐비면 CongWin은 작아지고, 한산하면 CongWin은 커지기때문에 전송속도가 빨라짐.

발신자는 혼잡을 어떻게 인식합니까?

- 손실 이벤트 = 시간 초과 또는 3개의 중복 ack
- TCP 발신자는 손실 이벤트 후 비율(CongWin)을 줄임

세 가지 메커니즘

- AIMD
- 느린 시작
- 시간 초과 이벤트 후 보수



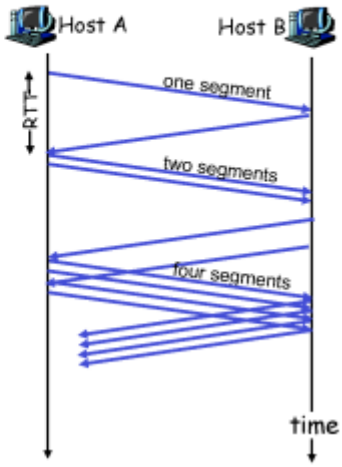
Slow Start

- 연결이 시작될 때, CongWin = 1 MSS

- 사용 가능한 대역폭은 >> MSS/RTT
- 연결이 시작되면 첫 번째 손실 이벤트가 발생할 때까지 속도를 기하급수적으로 빠르게 증가

TCP Slow Start (more)

- When connection begins, increase rate exponentially until first loss event:
  - double **CongWin** every RTT
  - done by incrementing **CongWin** for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast

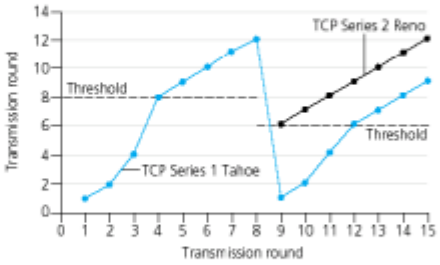


- 연결이 시작될 때, 첫 번째 손실 이벤트까지 비율을 기하급수적으로 증가
  - double CongWin every RTT
  - 수신된 모든 ACK에 대해 CongWin을 증가시켜 수행
- 요약: 초기 속도는 느리지만 기하급수적으로 빠르게 증가합니다.

Refinement

Q: When should the exponential increase switch to linear?

A: When **CongWin** gets to 1/2 of its value before timeout.



Implementation:

- Variable Threshold
- At loss event, Threshold is set to 1/2 of CongWin just before loss event

- y축 = condition window size / x축 = time

TCP Tahoe = 파란선

- ex) 기존의 Threshold가 8 이었는데, packet loss가 탐지되면 Threshold값을 packet loss가 탐지된 그 순간에 windowsize의 절반으로 바꿈

  ⇒ 위 예시에서는 탐지된 순간 값이 12였으니까, 다음 Threshold 값은 절반인 6으로 변경됨 ⇒ 그리고 다시 slow start 시작

TCP Reno = 검정선

- 패킷 유실을 어떻게 탐지? ⇒ timer
- 1. time out 현상이 발생했을 때 / 2. 3 dup ACK 를 받았을때
- 1의 네트워크 상황 ⇒ 그 패킷도 안 갔고, 그 이후의 패킷도 안감
- 2의 네트워크 상황 ⇒ 운나쁘게 그 하나의 패킷만 안감
- 2로 인해서 패킷 유실이 탐지됐을때(= 네트워크 혼잡의 조짐이 보였을때)

  ⇒ 잠깐 숨만 죽이자 == window size 절반으로 줄이기
- 1로 인해서 패킷 유실이 탐지됐을때(= 네트워크 혼잡의 조짐이 보였을때)

  ⇒ 아예 처음부터 완전히 밑바닥에서 시작

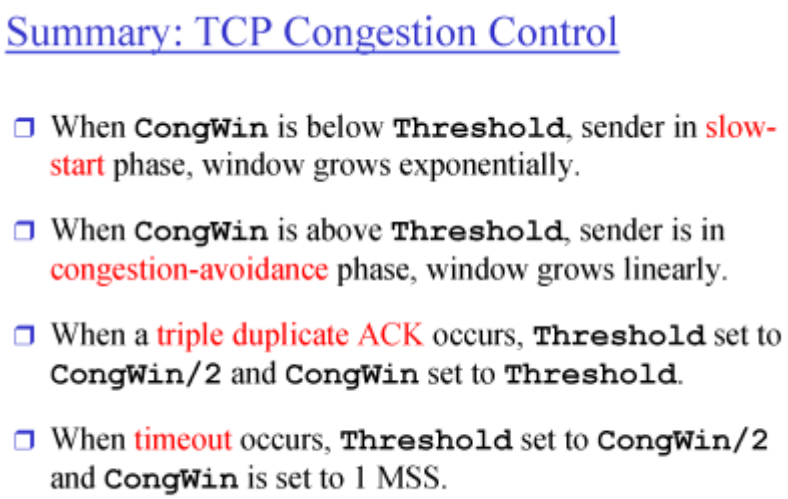
Refinement(정제)

Q : 지수 증가는 언제 선형으로 전환해야 합니까?

A : CongWin이 시간 초과 전에 값의 1/2에 도달하면

Implementation(구현)

- 가변 임계값
- 손실 이벤트에서 임계값은 손실 이벤트 직전 CongWin의 1/2로 설정됩니다.

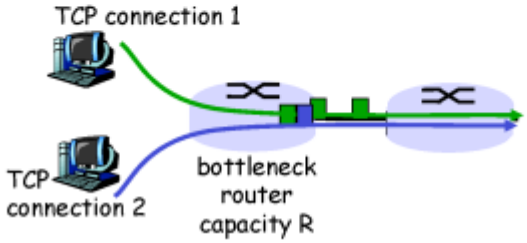


요약

- congwin이 임계값 미만일 때 느린 시작 단계의 발신자, 창이 기하급수적으로 커짐
- congwin이 임계값보다 높으면 발신자가 혼잡 회피 단계에 있고 창이 선형적으로 증가합니다.
- 3 duplicate ACK가 발생하면 임계값을 CongWin/2로 설정하고 CongWin을 임계값으로 설정
- 타임아웃이 발생하면 임계값은 CongWin/2로 설정되고 CongWin은 1 MSS로 설정

TCP Fairness

Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



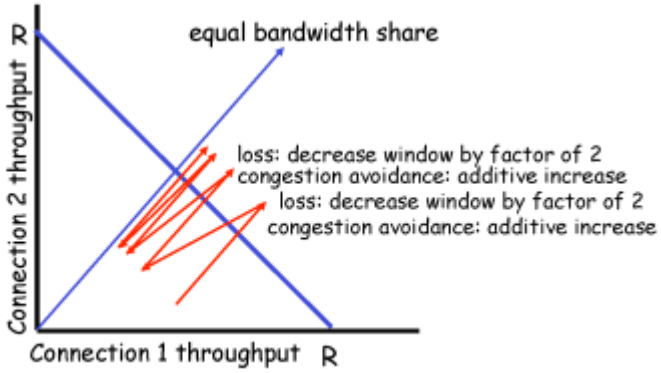
TCP Fairness(공정성)

- Fairness goal : K개의 TCP 세션이 대역폭 R의 동일한 병목 링크를 공유하는 경우 각각은 R/K의 평균 속도를 가져야 합니다.

Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally

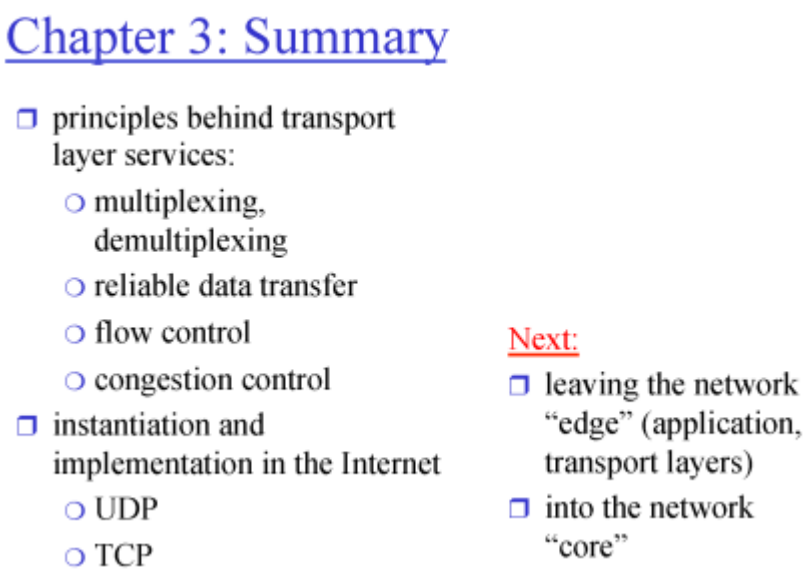


- x축 = 1번째 컴퓨터가 가질 수 있는 전송량 / y축 = 2번째 컴퓨터가 가질 수 있는 전송량

TCP가 공정한 이유는?

두 개의 경쟁 세션

- 가산 증가는 전체 증가에 따라 1의 기울기를 제공합니다.
- 곱셈 감소는 처리량을 비례적으로 감소



Chapter 3 요약

- 전송 계층 서비스의 원리
  - 다중화, 역다중화
  - 안정적인 데이터 전송
  - 흐름 제어
  - 혼잡 통제
- 인터넷에서 인스턴스화 및 구현
  - UDP
  - TCP