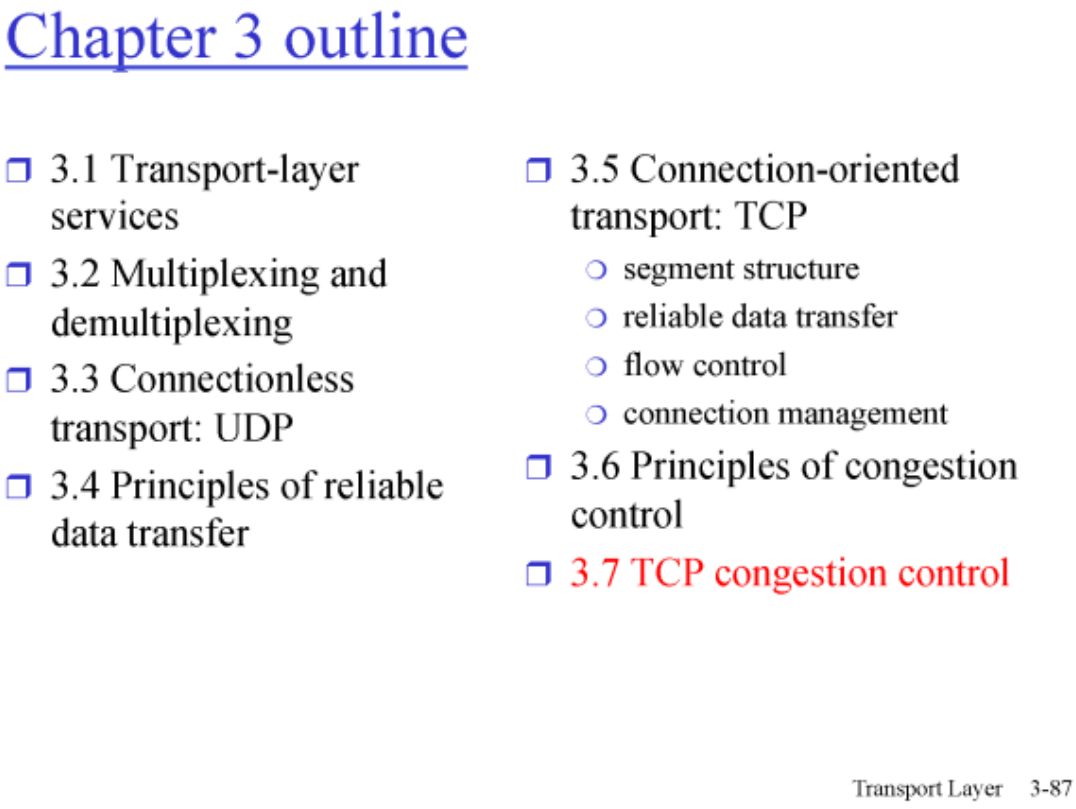
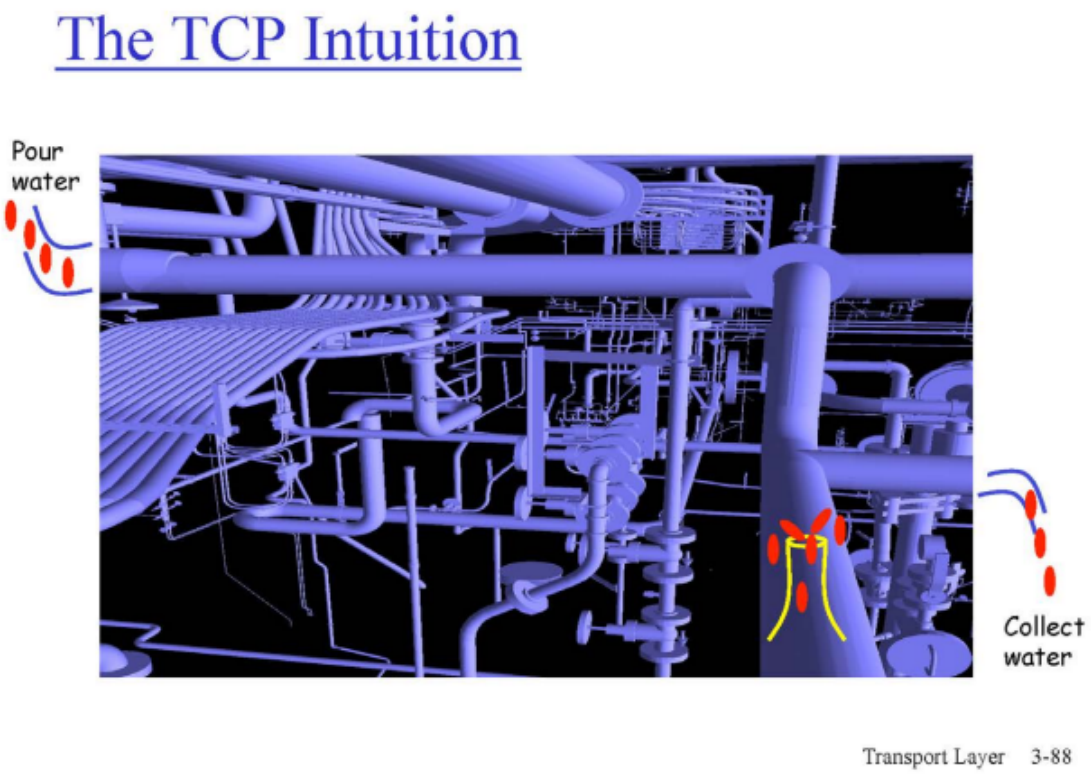
전송계층4

강의날짜	@2022/10/10
작성일시	@2022년 10월 10일 오후 11:54
편집일시	@2022년 10월 11일 오전 12:36
분야	알고리즘
공부유형	스터디 그룹
복습	☐
태그	



- 네트워크 상황이 안 좋아지면 속도를 낮추고, 네트워크 상황이 좋아지면 속도를 높임

- 이것은 본인을 위한 것도 있지만, 네트워크를 위한 것도 있음
- TCP라는 것은 패킷이 전송이 안되면 재전송을 하기 때문에 네트워크가 꽉 막히는 상황을 피하기 위해서 잘 조절을 한다
- 네트워크에서 직접적인 피드백이 없지만 저 멀리 있는 상대방 recv한테 피드백을 받아 유추함



- 보내는 TCP에서 받는 TCP
- 보내는 양 그대로 받는게 이상적임, 보내는 양이 많으면 더 이상적
- 보내는 양이 너무 많으면 파이프가 터짐
- 하지만 파이프 굵기를 알 수가 없고, 연결되면서 파이프 굵기가 다 달라서 알 수가 없음
- 각기 다른 굵기의 파이프를 따라가면서 가장 얇은 파이프보다 더 많이 보내면 병목현상에 의해 터짐
- 그 포인트를 못 찾음 아무도 report를 안해줘서 네트워크로부터 리포트가 아니라 양 끝단에 있는 애들끼리 정보 교환
- 한 방울, 두방울, 세방울 매우 조심스럽게 조금씩 부어봄

TCP Congestion Control

- 3 main phases
  1. Slow Start
  2. Additive increase
  3. Multiplicative decrease

- TCP 세가지 phases

1. Slow Start
   a. 처음에는 조금씩 천천히 작은 양 보내기
   b. 처음에 1, 2, 3, 4 보냈는데 감당할 수 있는 양이 100000이면 너무 겸손했다
   c. 따라서 점점 2배씩 증가하는 1, 2, 4, 8
2. Additive increase
   a. threshold 지점을 넘으면 조심해야함
   b. window size 늘려나가기
3. Multiplicative decrease
   a. packet loss를 탐지하면, window size를 절반으로 떨어뜨림
   b. 막힐때까지 하나씩 증가시키되 막히면 모두가 발을 확 빼야함 - 그래서 절반으로 줄임
   c. 다시 linear increase 함

TCP Congestion Control

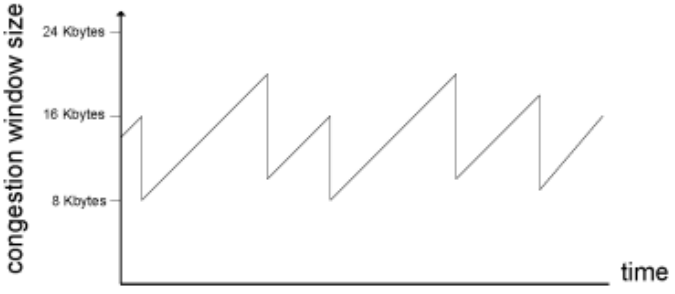
- 3 main phases

1. Slow Start: Do not know bottleneck bandwidth
   So start from zero and quickly ramp up
2. Additive increase: Hey, we are getting close to capacity
   Let's be conservative and increase slow
3. Multiplicative decrease: Oops! Packet drop
   Start over from slow start (from scratch)
   Hmm! many ACKs coming, start midway

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



- 계속해서 그 지점을 찾는 과정을 반복하게 됨

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

Transport Layer 3-92

- 전송속도는 RTT분의 Window 사이즈
- RTT라는 것은 변하는 값이긴 하지만, 실제로 변화가 심한것은 window size
- 전송속도는 결국에서는 congestion window size에 의해 결정되는데, 이는 곧 네트워크 상황에 의해 결정됨
- 많이 붐비면 congWin은 작아지고, 한산하면 congWin은 커짐
- 우리 모두의 행동이 각기 자신의 전송 속도를 결정하게됨

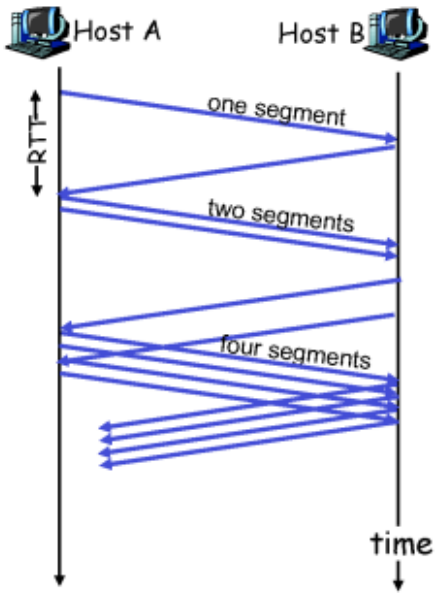
TCP Slow Start

- When connection begins, **CongWin** = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate

- When connection begins, increase rate exponentially fast until first loss event

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

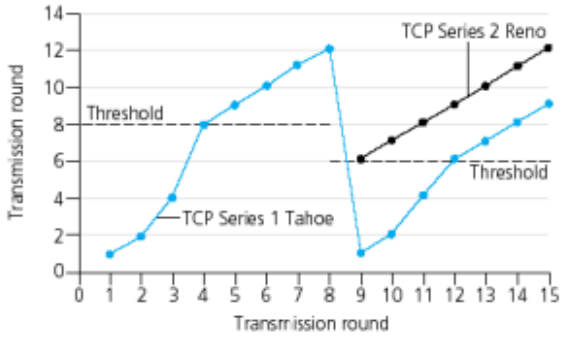


1. 아주 겸손하게 segment를 보낸다
2. 실제로는 slow하지 않고 빠르다

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

Transport Layer 3-95

- x축은 시간 y축은 congestion window size
- threshold를 넘는 순간부터 하나씩 증가
- Tahoe : 4까지 slow start 9초에 12 → 6으로 변화
- timeout → 패킷유실 / 3 dup ACK → 패킷유실 : 2가지
- timeout 네트워크 : 그 뒤로 아무것도 안 간다 큰일임
- 3 dup ACK : 다 잘받았는데 ACK10만 문제가 생김
  - timeout과 3 dup ACK는 다른 상황 다른 대응이 필요함
  - Reno : 3 dup ACK 는 아예 발 뺄 필요가 없다 → 절반만 뺀다
  - Tahoe : timeout이다 발을 아예 뺀다
- 처음에 threshold는 어떻게 잡냐?
  - 알 수 없다 구현하는 사람 맘대로 처음부터 어떤 값이 세팅되어있을 수도

- threshold값은 packet loss가 발생한 상황 window size의 절반 값으로 세팅한다
  threshold 이후의 값은 무조건 linear increase

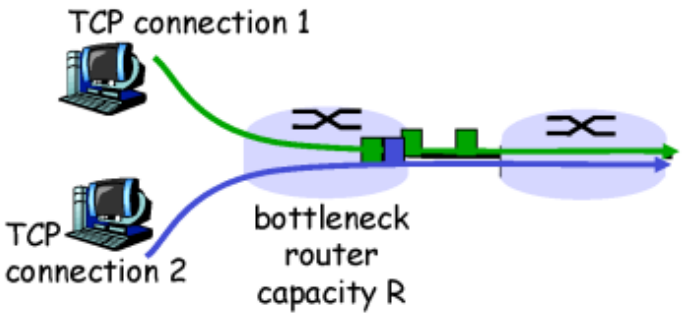
Summary: TCP Congestion Control

- When **CongWin** is below **Threshold**, sender in slow-start phase, window grows exponentially.
- When **CongWin** is above **Threshold**, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, **Threshold** set to **CongWin/2** and **CongWin** set to **Threshold**.
- When timeout occurs, **Threshold** set to **CongWin/2** and **CongWin** is set to 1 MSS.

Transport Layer 3-98

TCP Fairness

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



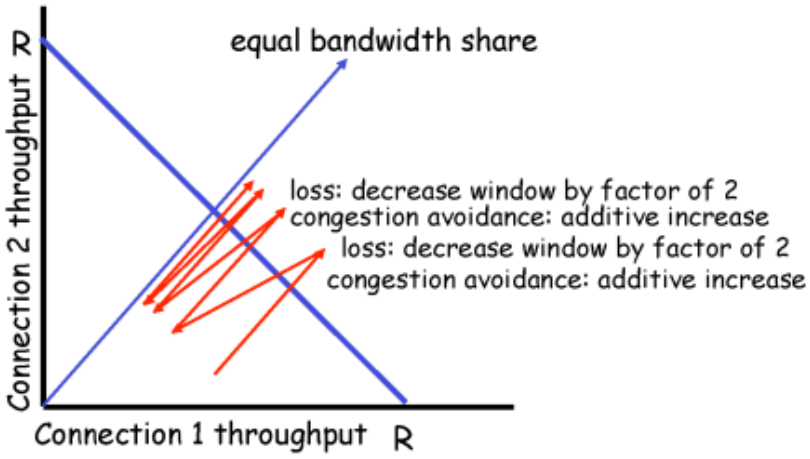
Transport Layer 3-102

- end to end : 둘 사이의 전송량 조절
- 네트워크 둘 만 쓰나? 모두가 동시에 사용 중
- 독립적으로 사용할 때 TCP가 Fair하게 됨

Why is TCP fair?

Two competing sessions:

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



Transport Layer 3-103

- x 축은 첫번째 컴퓨터 전송량
- y축은 두번째 컴퓨터 전송량
- R이라는 value를 서로 나눠쓰는 상황
  - 반복해서 점점 Fair한 지점을 찾게됨
  - 진짜로 Fair한가?
  - 맹점 : TCP connection 많이 연 사람이 더 많은 네트워크를 사용하게 됨
  - TCP 간의 Fair일 뿐이다

Chapter 3: Summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation and implementation in the Internet
  - UDP
  - TCP

Next:

- leaving the network "edge" (application, transport layers)
- into the network "core"