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

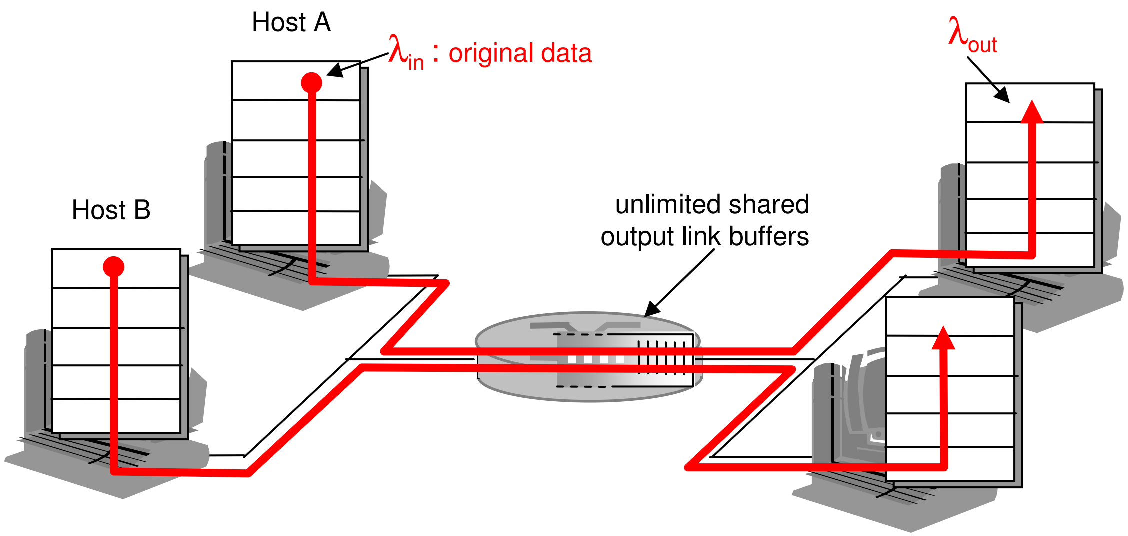
Principles of Congestion Control
壅塞控制的原理

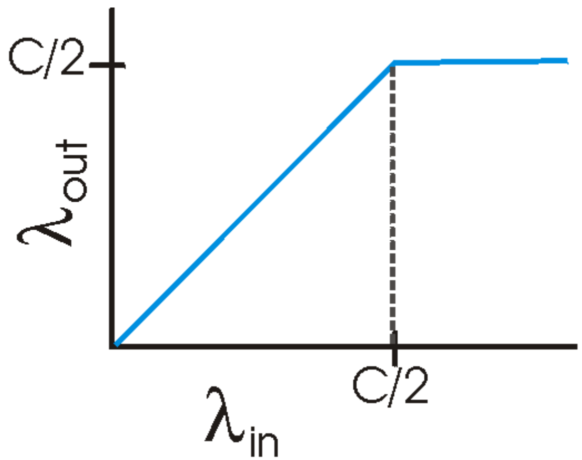
Congestion: 什麼是壅塞？

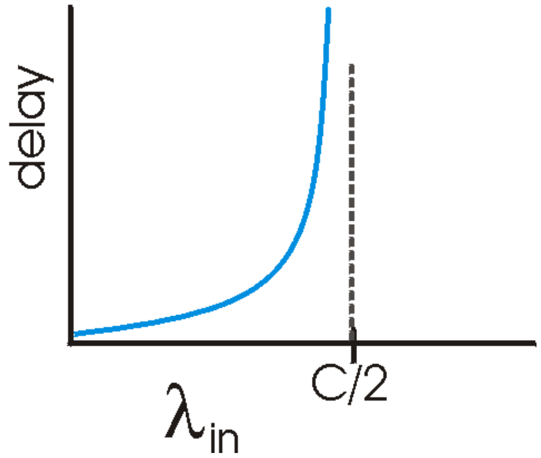
- informally: "too many sources sending too much data too fast for *network* to handle"
  網路無法負荷太多來源所送出的資料
- different from flow control! 和流量控制不同
- manifestations:會發生的問題
  - lost packets (buffer overflow at routers)
    封包遺失
  - long delays (queueing in router buffers)
    延遲過長
- a top-10 problem!

Causes/costs of congestion: scenario 1
兩個傳送端，一台有無限緩衝區的路由器

- two senders, two receivers
- one router, infinite buffers
- no retransmission
  沒有重傳





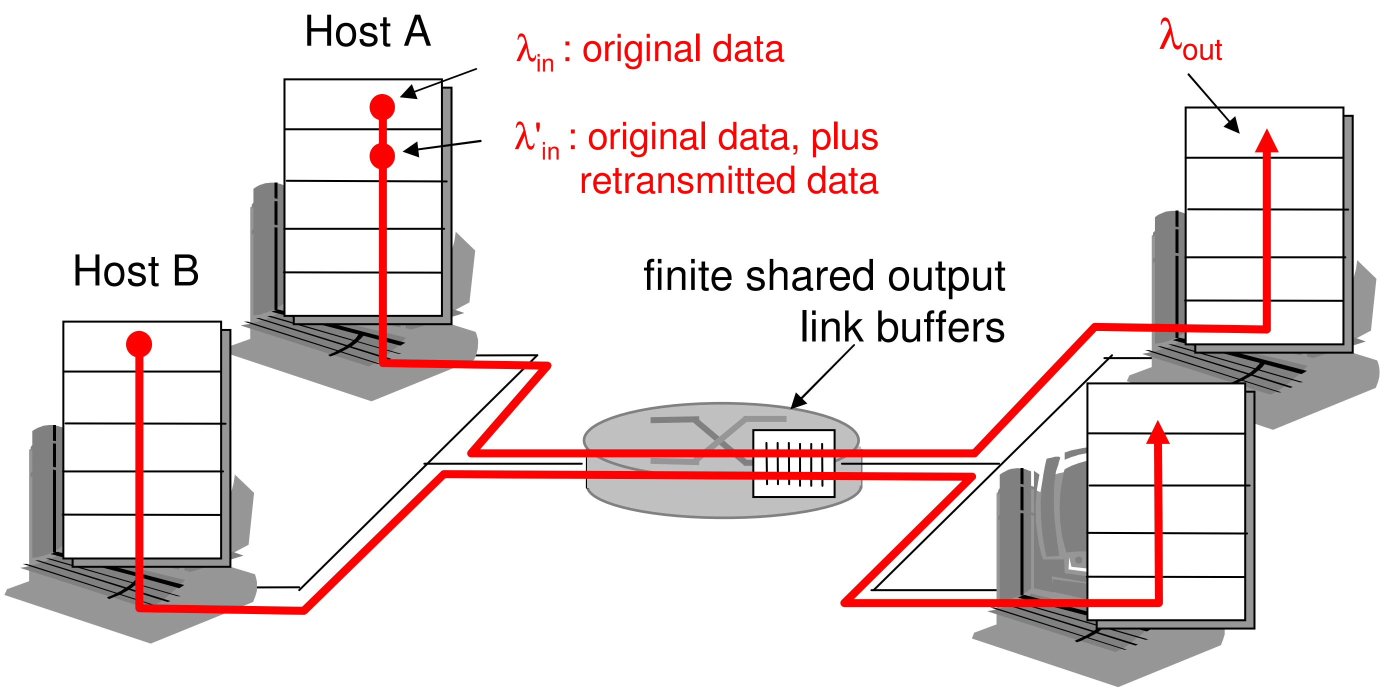


- large delays when congested
- maximum achievable throughput

Causes/costs of congestion: scenario 2

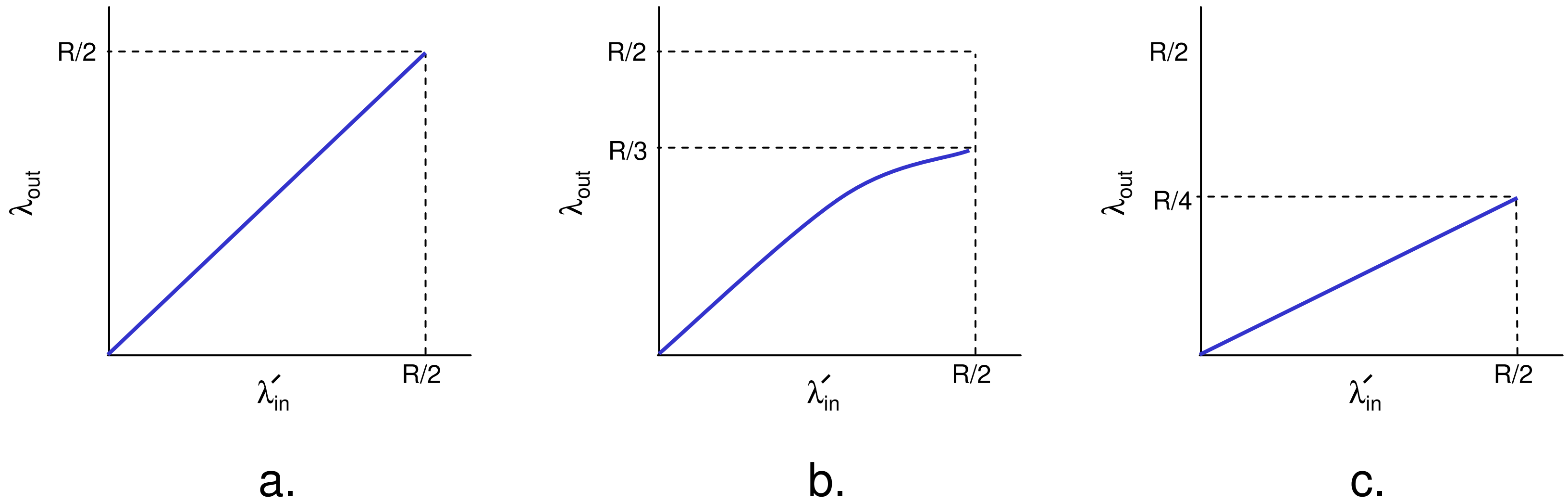
兩個傳送端，一台具有有限容量緩衝區的路由器

- one router, *finite* buffers
- sender retransmission of lost packet 封包遺失時重傳



Causes/costs of congestion: scenario 2

- always: $\lambda_{in} = \lambda_{out}$ (goodput)
- "perfect" retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



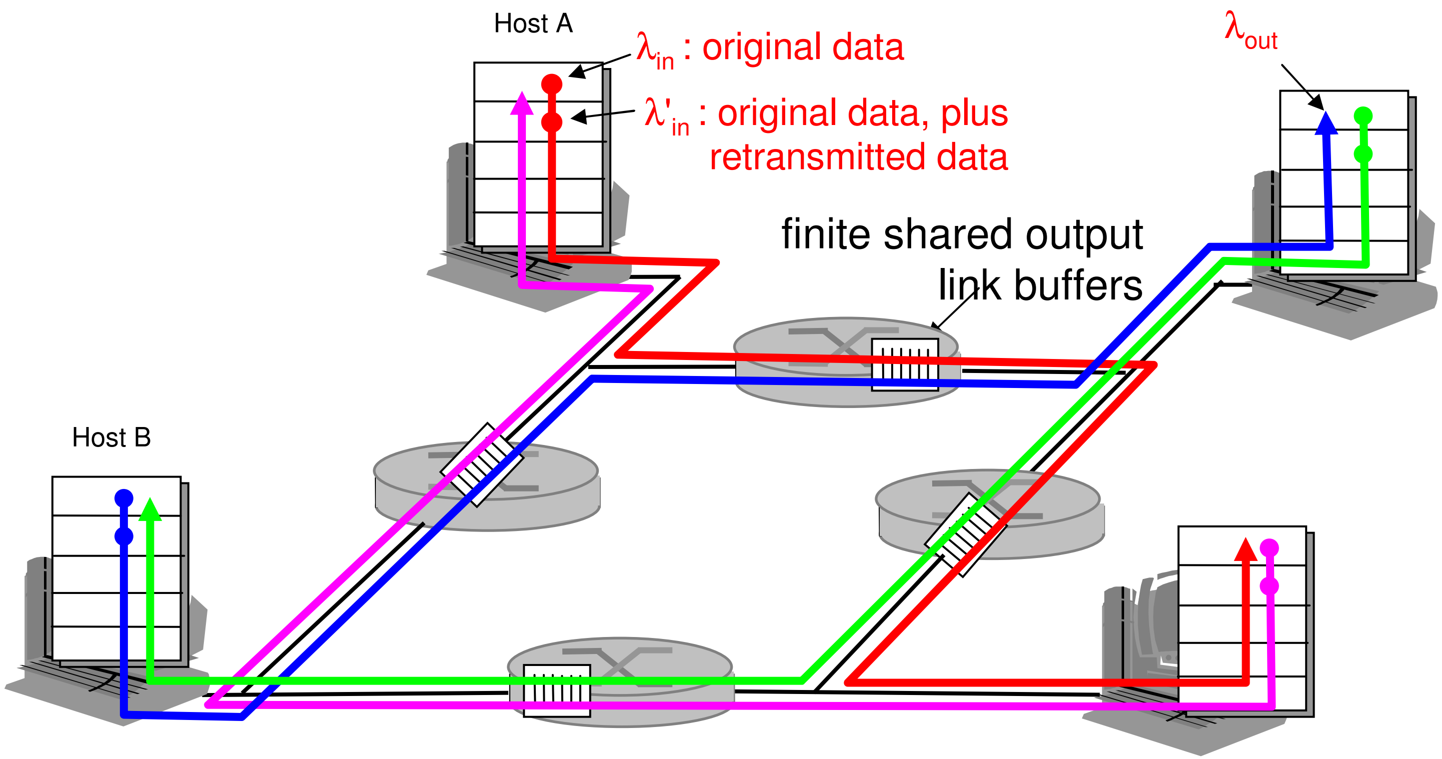
"costs" of congestion: 壅塞的代價

- more work (retrans) for given "goodput" 重傳
- unneeded retransmissions: link carries multiple copies of pkt 不必要的重傳

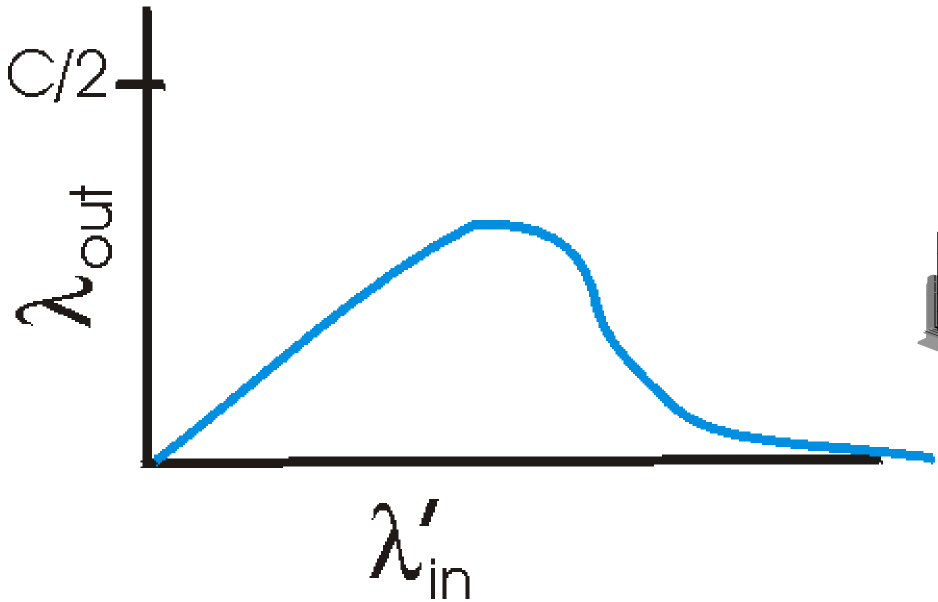
Causes/costs of congestion: scenario 3

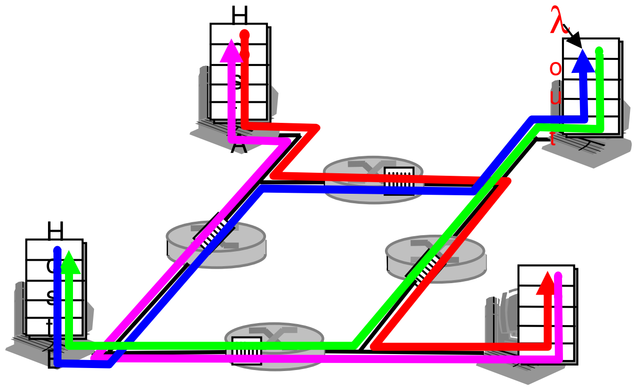
- four senders 四個傳送端
- multihop paths 多段路徑
- timeout/retransmit

Q: what happens as λ_{in} and λ'_{in} increase ?



Causes/costs of congestion: scenario 3





Another "cost" of congestion:

- when packet dropped, any "upstream transmission capacity used for that packet was wasted!
  浪費用已傳輸的容量

Approaches towards congestion control
壅塞控制的方法

Two broad approaches towards congestion control:

End-end congestion control: 點對點控制

- no explicit feedback from network 網路不會幫忙
- congestion inferred from end-system observed loss, delay 由端點發現壅塞
- approach taken by TCP TCP使用這種方法

Network-assisted congestion control:

網路協助

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM) 用1 bit來指出壅塞
  - explicit rate sender should send at

Case study: ATM ABR congestion control
範例：ATM ABR壅塞控制

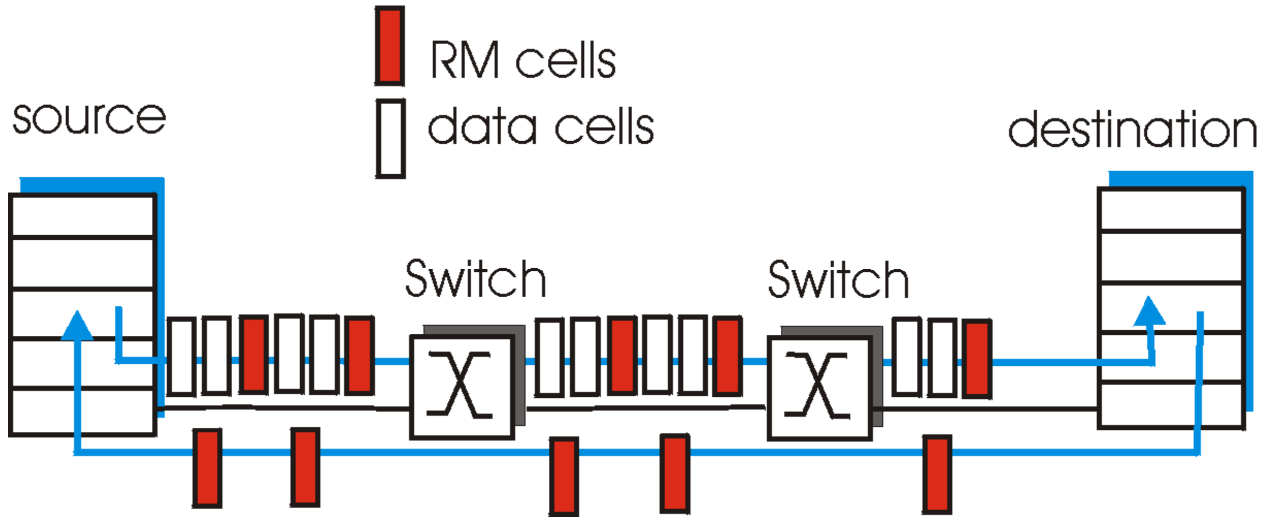
ABR: available bit rate:

- "elastic service"
- if sender's path "underloaded": 不壅塞的狀況
  - sender should use available bandwidth
- if sender's path congested: 壅塞的狀況
  - sender throttled to minimum guaranteed rate

RM (resource management) cells: 資源管理單元

- sent by sender, interspersed with data cells
- bits in RM cell set by switches ("*network-assisted*")
  - NI bit: no increase in rate (mild congestion)
  - CI bit: congestion indication
- RM cells returned to sender by receiver, with bits intact

Case study: ATM ABR congestion control



- two-byte ER (explicit rate) field in RM cell
  - congested switch may lower ER value in cell
  - sender' send rate thus maximum supportable rate on path
- EFCI bit in data cells: set to 1 in congested switch
  - if data cell preceding RM cell has EFCI set, sender sets CI bit in returned RM cell

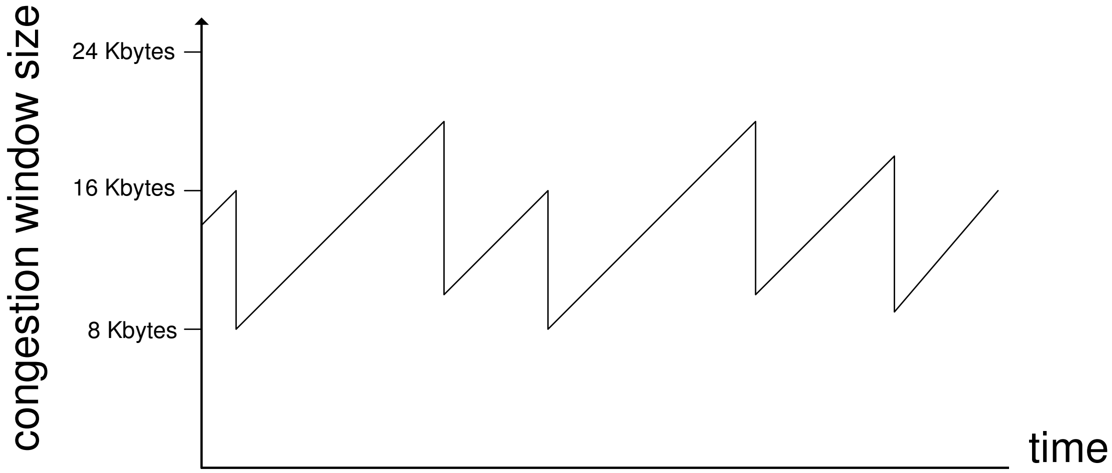
Chapter 3 outline

- 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 壅塞控制

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 累加遞增
  - *multiplicative decrease*: cut **CongWin** in half after loss 倍數遞減

Saw tooth behavior: probing for bandwidth



TCP Congestion Control: details
TCP 壅塞控制

- sender limits transmission:

  LastByteSent-LastByteAcked
  $\leq$ **CongWin**
- Roughly,

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

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

  壅塞窗格是會改變的，隨著網路的壅塞程度改變。

How does sender perceive congestion?
如何知道發生壅塞？

- loss event = timeout *or* 3 duplicate acks
  逾時或三次重覆的acks
- TCP sender reduces rate (**CongWin**) after loss event
  發現壅塞，減少傳送速度

three mechanisms:

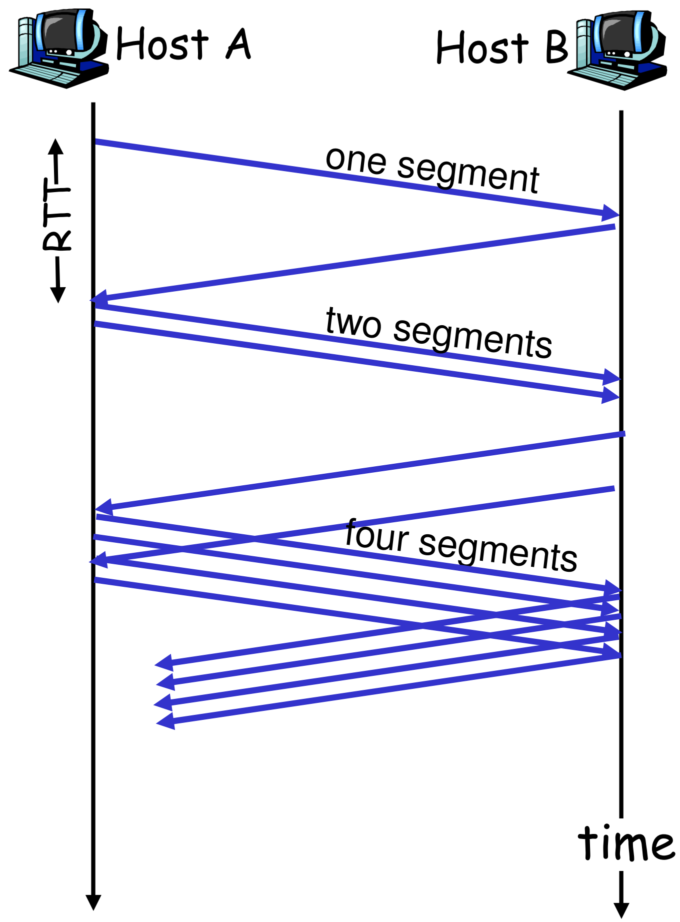
- AIMD
- slow start 低速啟動
- conservative after timeout events 回應逾時事件

TCP Slow Start 低速啟動

- When connection begins, **CongWin** = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 25 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
  速度呈指數增加，直到發生第一次封包遺失

TCP Slow Start (more) 低速啟動

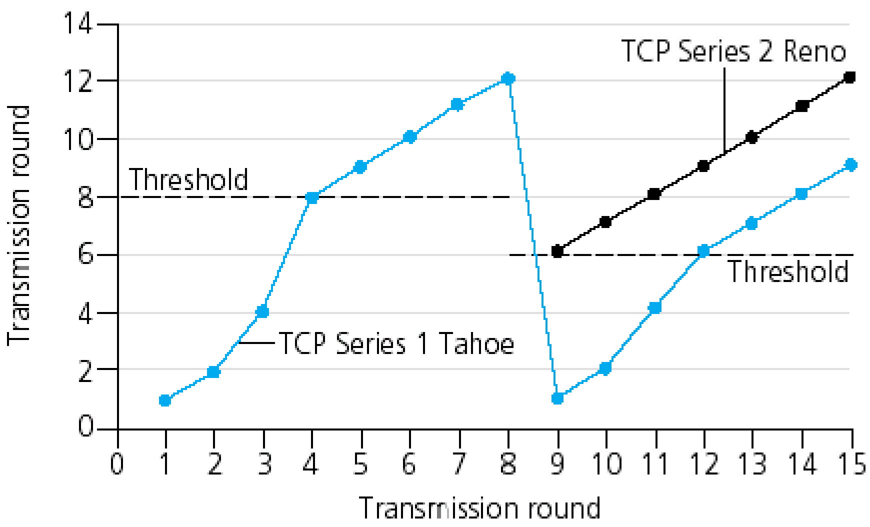
- When connection begins, increase rate exponentially until first loss event:
  - double CongWin every RTT 兩倍增加
  - done by incrementing CongWin for every ACK received
- <u>Summary:</u> initial rate is slow but ramps up exponentially fast



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

1~8, 9~12 slow start 低速啟動
4~8, 12~15 AIMD
8~9 Timeout (slow start agin)

Refinement: inferring loss

- After 3 dup ACKs: 重覆ack
  - `CongWin` is cut in half CongWin減半
  - window then grows linearly線性增加
- But after timeout event:
  - `CongWin` instead set to 1 MSS; 直接設成1
  - window then grows exponentially 低速啓動
  - to a threshold, then grows linearly 至原先一半時，線性增加

Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- timeout indicates a "more alarming" congestion scenario

Summary: TCP Congestion Control

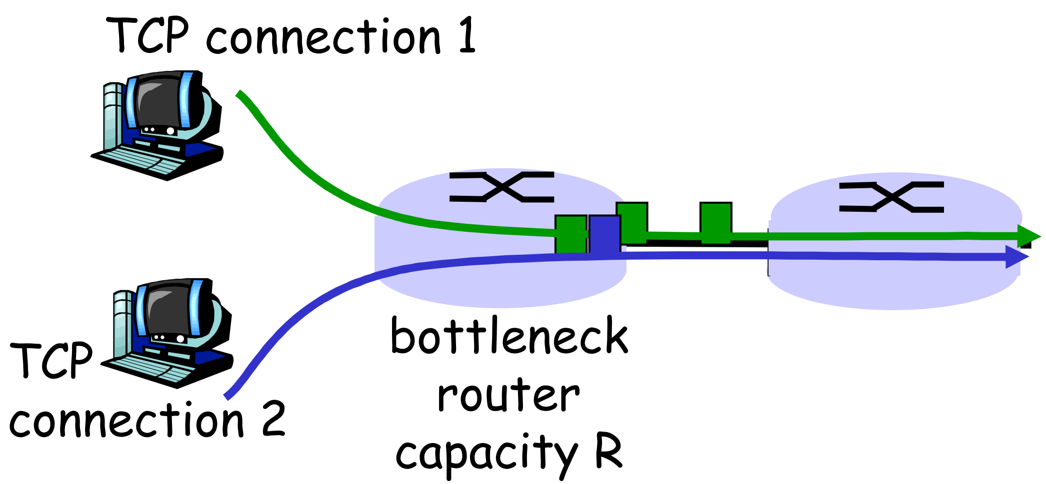
- When `CongWin` is below `Threshold`, sender in slow-start phase, window grows exponentially. 低速啟動階段
- When `CongWin` is above `Threshold`, sender is in congestion-avoidance phase, window grows linearly. 避免壅塞階段
- When a triple duplicate ACK occurs, `Threshold` set to `CongWin/2` and `CongWin` set to `Threshold`. 三個重覆ACK發生
- When timeout occurs, `Threshold` set to `CongWin/2` and `CongWin` is set to 1 MSS. 逾時發生

TCP sender congestion control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

TCP Fairness 公平性

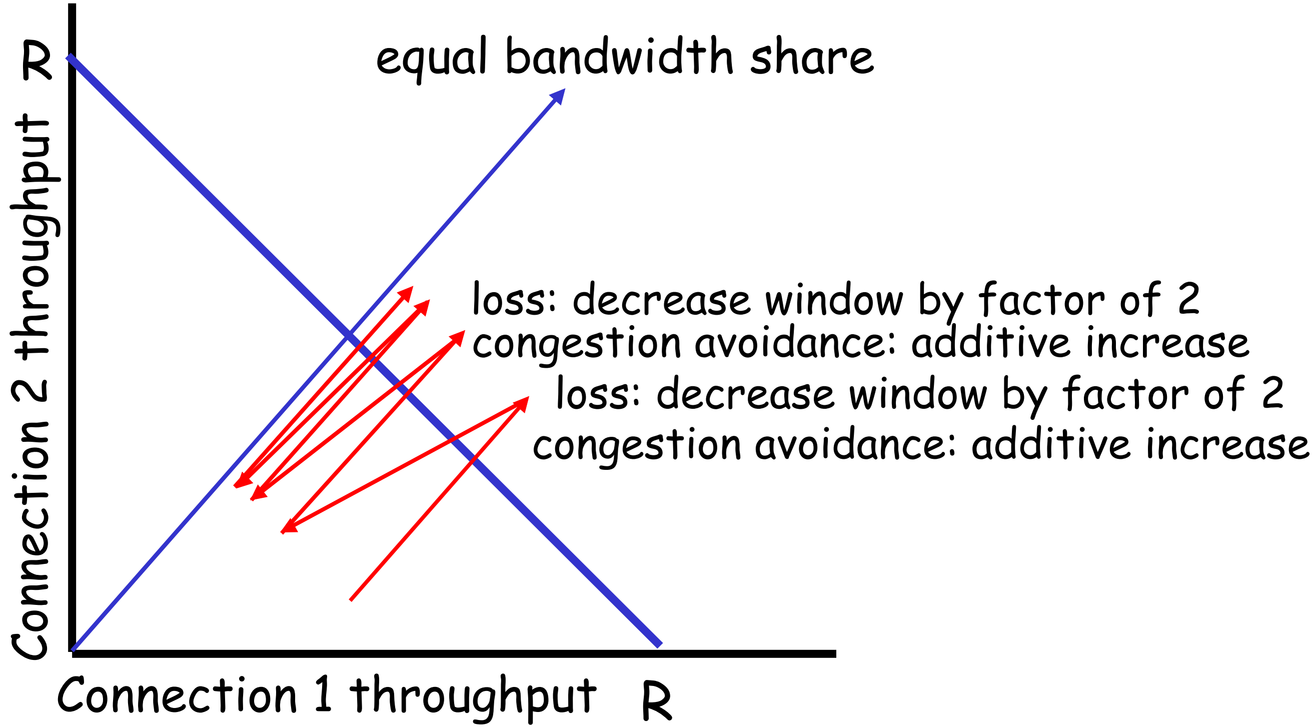
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K 平均使用頻寬



Why is TCP fair? TCP公平的原因

Two competing sessions:

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



Fairness (more)

Fairness and UDP

- Multimedia apps often do not use TCP
  - do not want rate throttled by congestion control 速率不振盪
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts. 同時啟動多筆連線
- Web browsers do this
- Example: link of rate R supporting 9 connections;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !

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"