运输层

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/

Outline

- TCP flow control
- TCP congestion control
- TCP congestion control wrap-up
- Router assisted congestion control

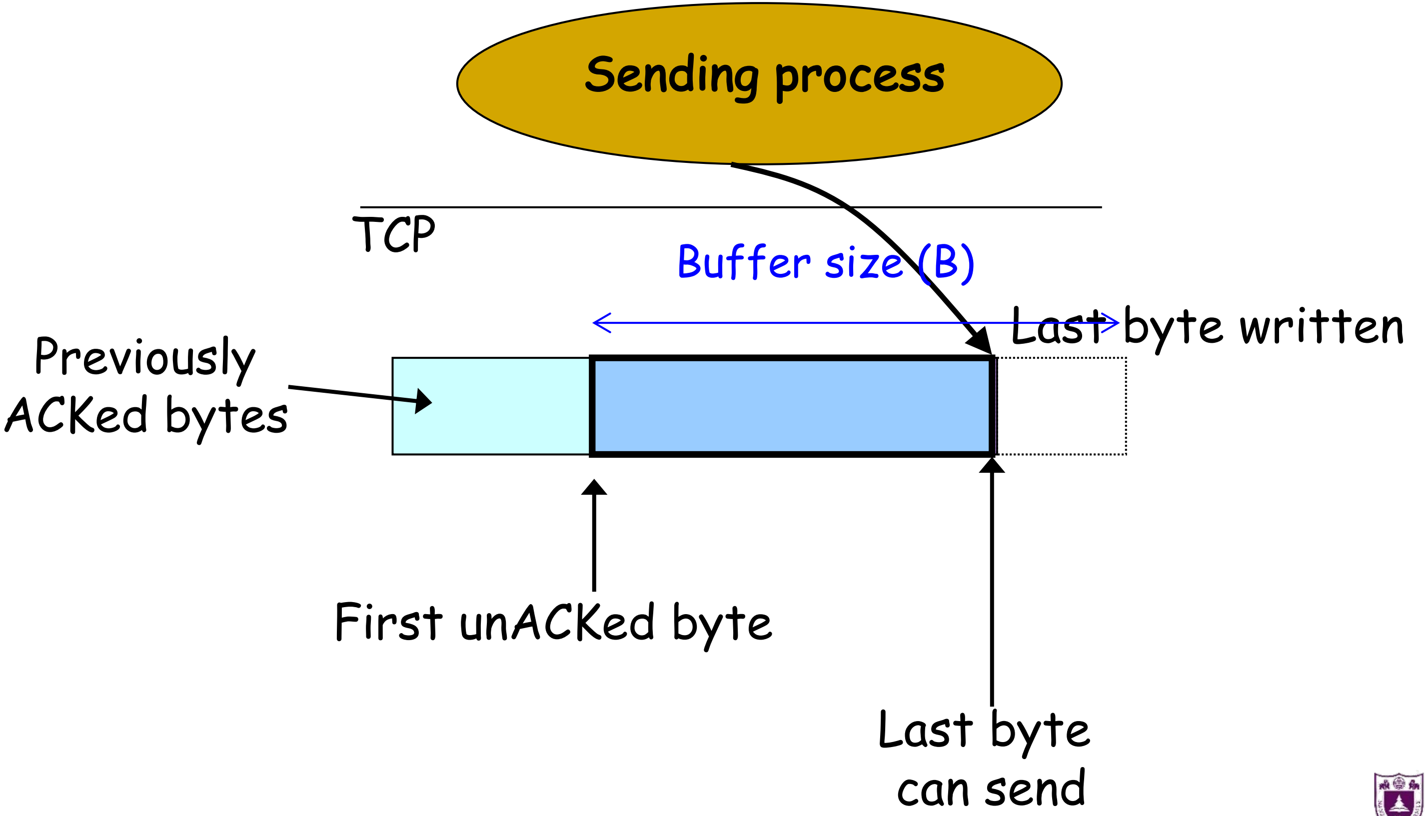
TCP header

<table>
<tr><td colspan="3">Source port</td><td>Destination port</td></tr>
<tr><td colspan="4">Sequence number</td></tr>
<tr><td colspan="4">Acknowledgment</td></tr>
<tr><td>HdrLen</td><td>0</td><td>Flags</td><td>Advertised window</td></tr>
<tr><td colspan="3">Checksum</td><td>Urgent pointer</td></tr>
<tr><td colspan="4">Options (variable)</td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

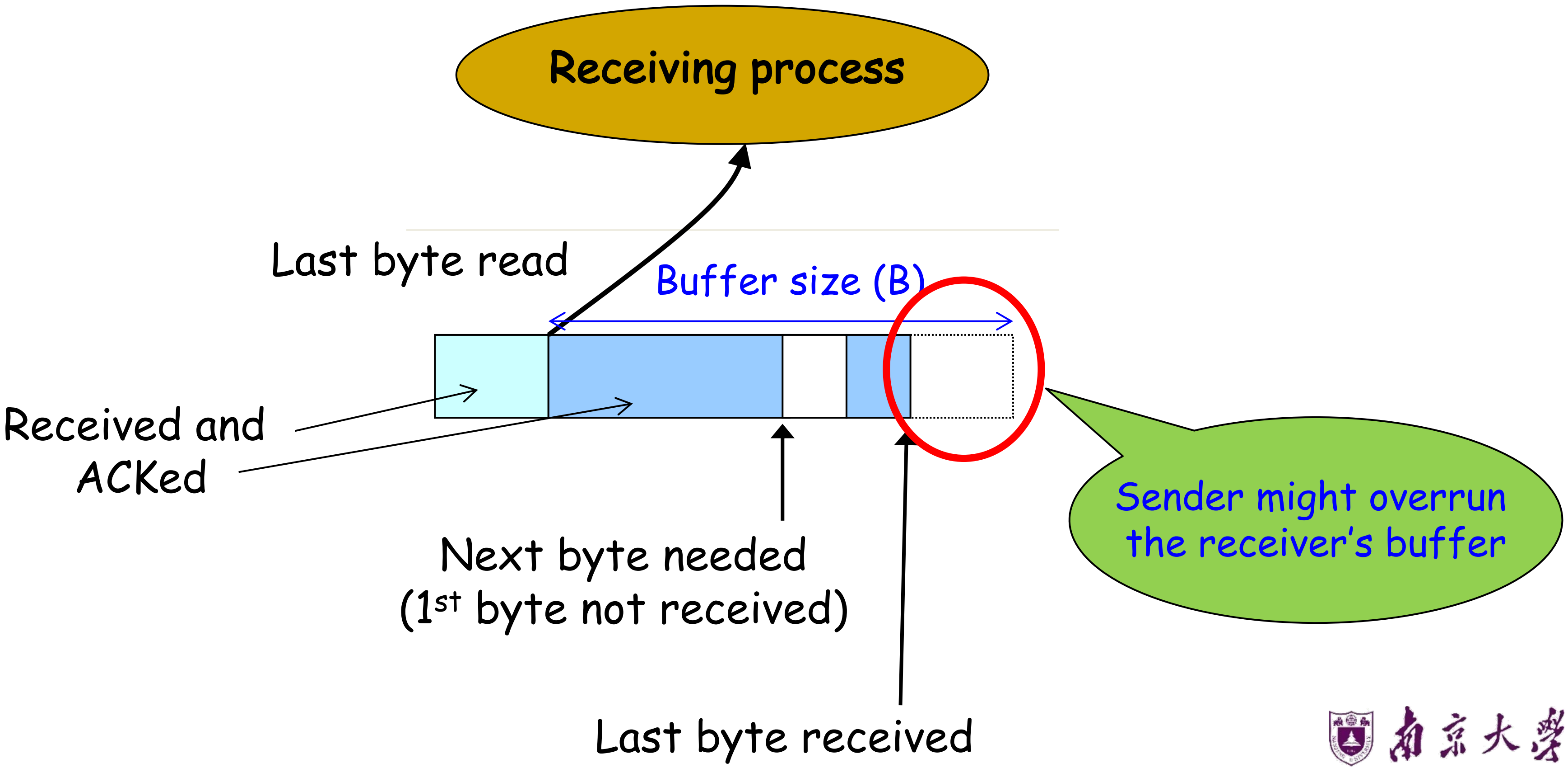
Recap: Sliding window

- Both sender and receiver maintain a window
- Left edge of window:
  - Sender: beginning of unacknowledged data
  - Receiver: beginning of expected data
    - ✓ First "hole" in received data
    - ✓ When sender gets ack, knows that receiver's window has moved
- Right edge: Left edge + constant
  - The constant is only limited by buffer size in the transport layer

Sliding window at sender



Sliding window at receiver



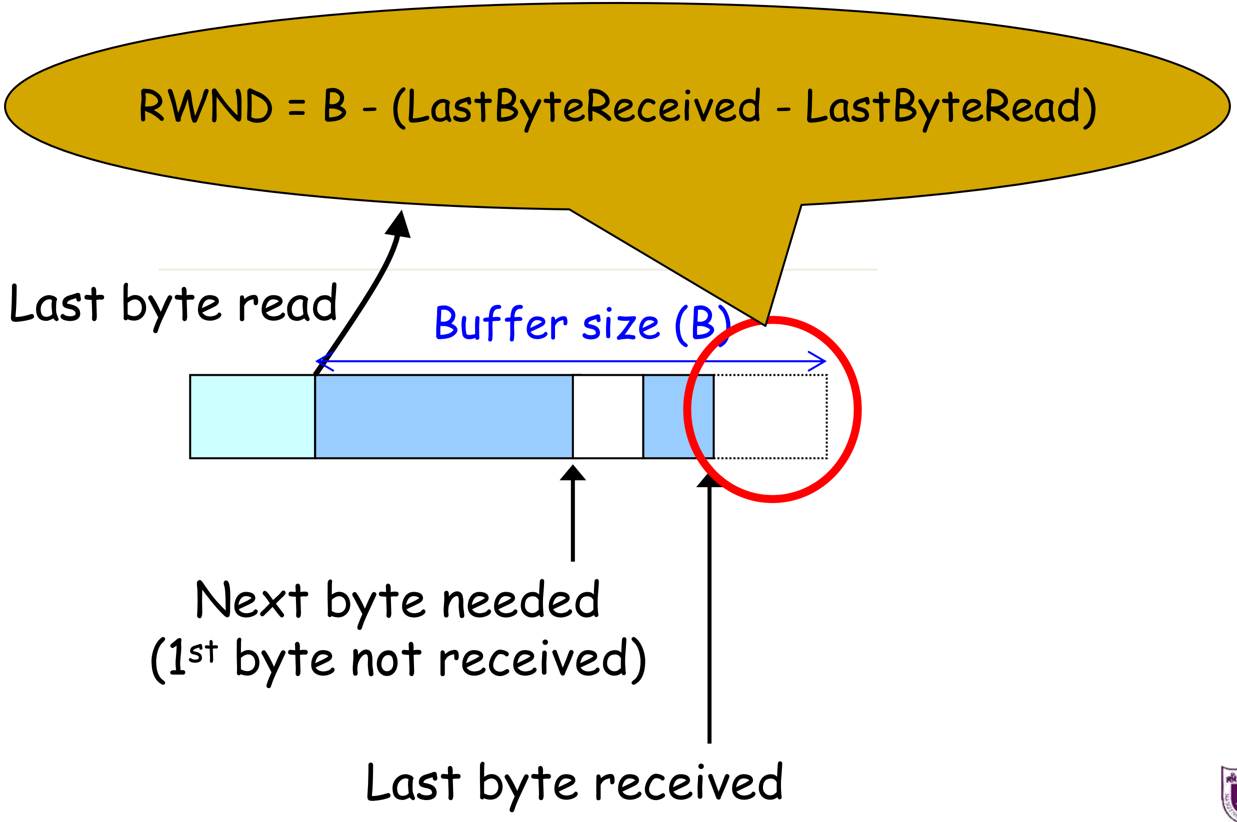
Fixed sliding window ?

- Fixed sliding window
  - Works well on reliable direct links
- Problem:
  - Failure to receive ACK is taken as flow control indication
  - The receiver can achieve flow control by stop sending ACK, but the sender can not distinguish between lost segment and flow control

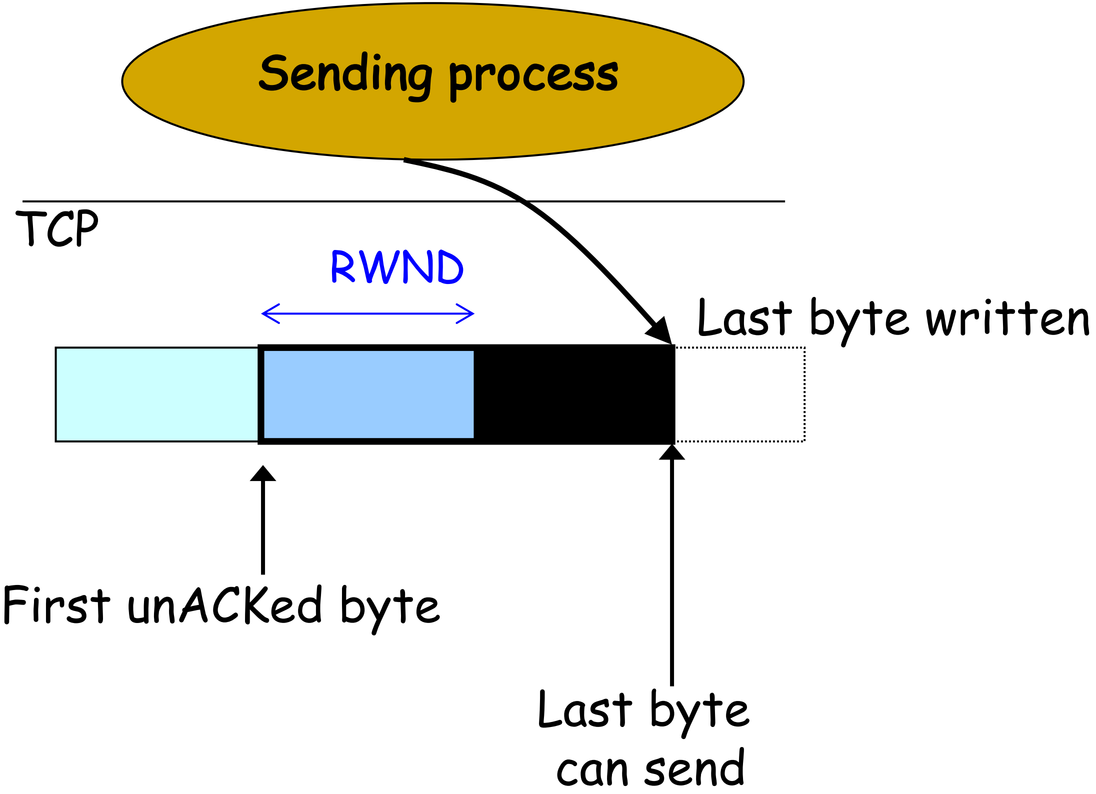
Solution: Credit Scheme

- Receiver advertises spare room (credits) using an "Advertised Window" (RWND) to prevent sender from overflowing its window
  - Receiver indicates value of RWND in ACKs
  - Sender ensures that the total number of bytes in flight <= RWND

Sliding window at receiver



Sliding window at sender



Sliding window with flow control

- Sender: window advances when new data ACK'd
- Receiver: window advances as receiving process consumes data
- Receiver advertises to the sender where the receiver window currently ends ("righthand edge")
  - Sender agrees not to exceed this amount
- UDP does not have flow control
  - Data can be lost due to buffer overflow

- Credit allocation deadlock
  - B sends A: segment with AN=i, W=0 closing rcv-window
  - B sends A: AN=i, W=j to reopen, but this maybe lost
  - Now A thinks window is closed, B thinks it is open and wait

- Handle
  - Use window timer
  - If timer expires without any receiving, send something
  - Could be re-transmission of previous segment

Outline

- TCP flow control
- TCP congestion control
- TCP congestion control wrap-up
- Router assisted congestion control

A story: Congestion collapse in 1980s

- In 1981, TCP was standardized and widely deployed.
- No congestion control is considered.

- In October of 1986, the Internet had the first of what became a series of 'congestion collapses'. During this period, the data throughput from LBL to UC Berkeley (sites separated by 400 yards and two IMP hops) dropped from 32 Kbps to 40 bps. (Van Jacobson, Congestion Avoidance and Control)

- This open a new area of congestion control study.

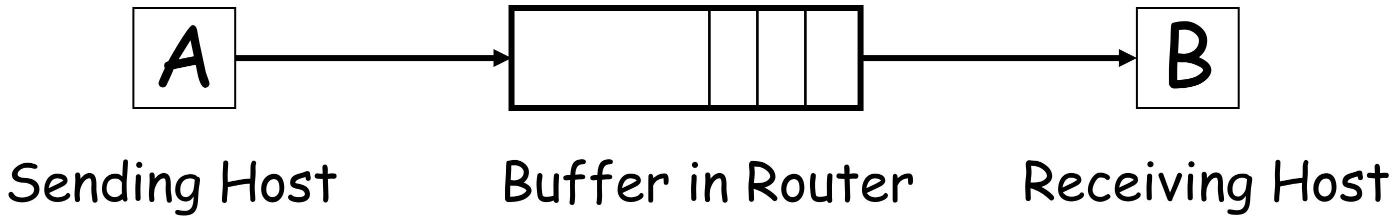
Key design considerations

- How do we know the network is congested?
  - Implicit and/or explicit signals from the network
- Who takes care of congestion?
  - End hosts (may receive some help from the network)
- How do we handle congestion?
  - Continuous adaptation

Three issues to consider

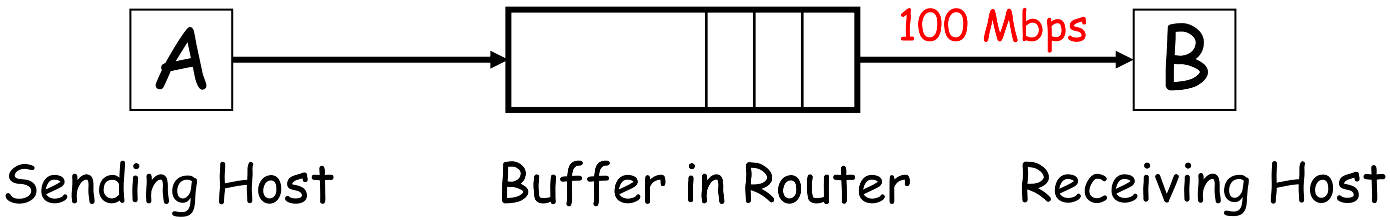
- Discovering the available (bottleneck) bandwidth
- Adjusting to variations in bandwidth
- Sharing bandwidth between flows

Abstract view



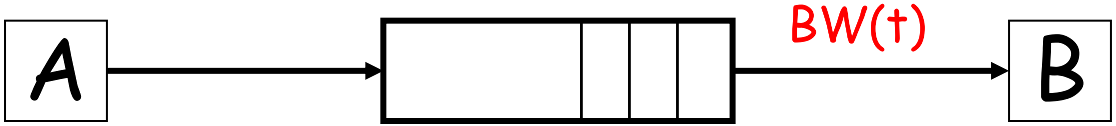
- Ignore internal structure of router and model it as a single queue for a particular input-output pair

Discovering available bandwidth



- Pick sending rate to match bottleneck bandwidth
  - Without any a priori knowledge
  - Could be gigabit link, could be a modem

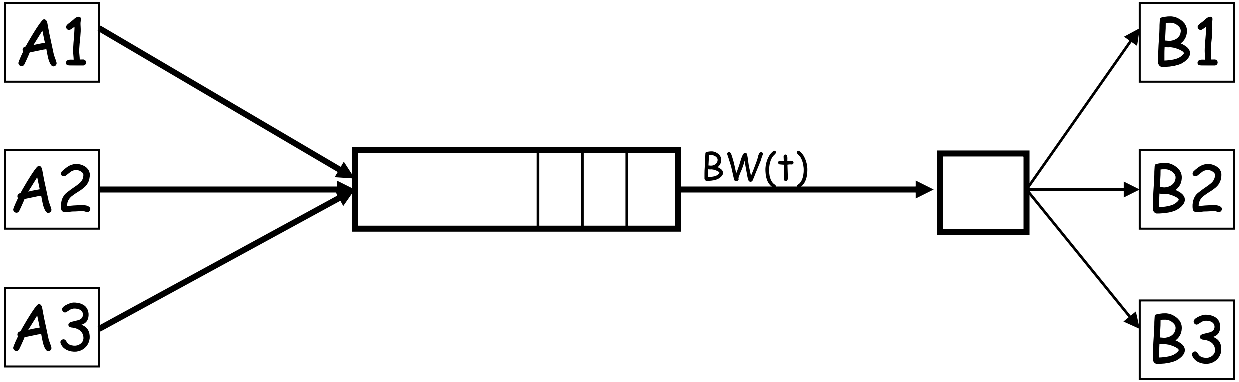
Adjusting to variations in bandwidth



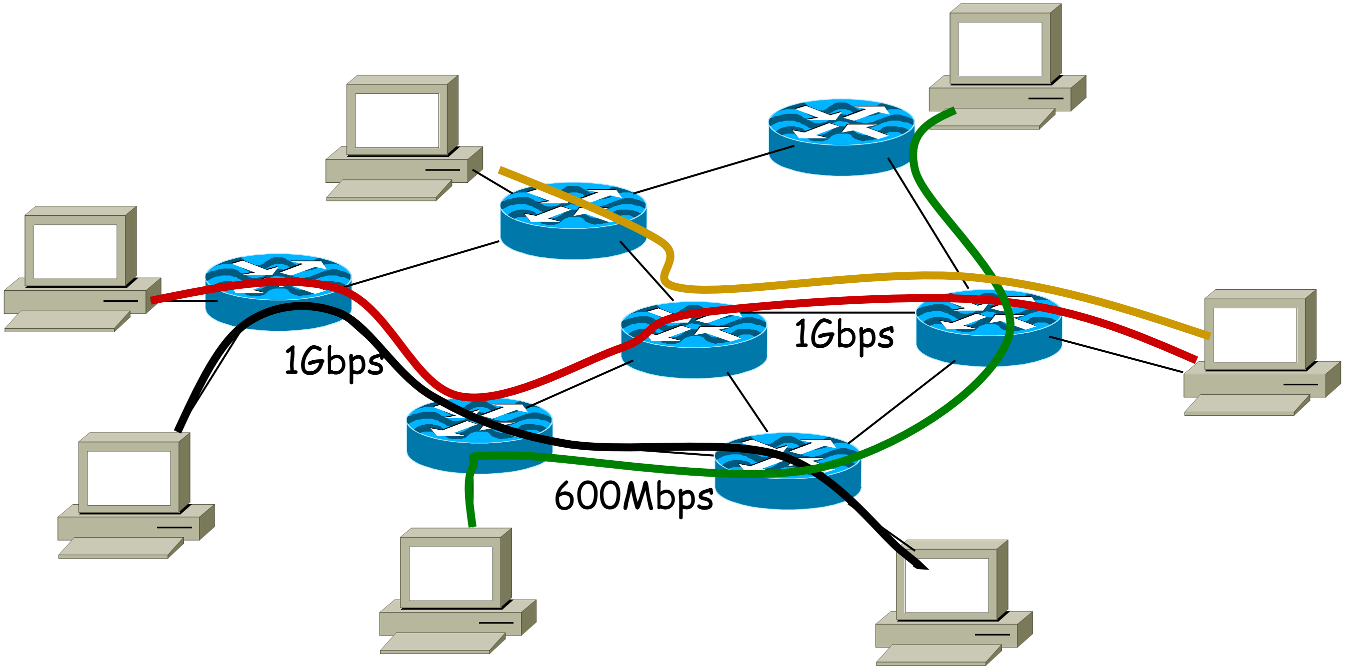
- Adjust rate to match instantaneous bandwidth
  - Assuming you have rough idea of bandwidth

Multiple flows and sharing bandwidth

- Two Issues:
  - Adjust total sending rate to match bandwidth
  - Allocation of bandwidth between flows



Reality



Congestion control is a resource allocation problem involving many flows, many links, and complicated global dynamics

Possible approaches

(0) Send without care

- Many packet drops

Possible approaches

(0) Send without care

(1) Reservations

- Pre-arrange bandwidth allocations
- Requires negotiation before sending packets
- Low utilization

Possible approaches

(0) Send without care

(1) Reservations

(2) Pricing

- Don't drop packets for the high-bidders
- Requires payment model

Possible approaches

(0) Send without care

(1) Reservations

(2) Pricing

(3) Dynamic Adjustment

- Hosts infer level of congestion; adjust
- Network reports congestion level to hosts; hosts adjust
- Combinations of the above
- Simple to implement but suboptimal, messy dynamics

Possible approaches

(0) Send without care

(1) Reservations

(2) Pricing

(3) Dynamic Adjustment

- Generality of dynamic adjustment has proven to be very powerful
  - Doesn't presume business model, traffic characteristics, application requirements
  - But does assume good citizenship!

Two basic questions

- How does the sender detect congestion?
- How does the sender adjust its sending rate?
  - To address three issues
    - ✓ Finding available bottleneck bandwidth
    - ✓ Adjusting to bandwidth variations
    - ✓ Sharing bandwidth

Detecting congestion

- Packet delays
  - Tricky: noisy signal (delay often varies considerably)
- Routers tell end hosts when they're congested
- Packet loss
  - Fail-safe signal that TCP already has to detect
  - Complication: non-congestive loss (e.g., checksum errors)

Not all losses are the same

- Duplicate ACKs: isolated loss
  - Still getting ACKs
- Timeout: much more serious
  - Not enough dupacks
  - Must have suffered several losses
- Will adjust rate differently for each case

Rate adjustment

- Basic structure
  - Upon receipt of ACK (of new data): increase rate
  - Upon detection of loss: decrease rate
- How we increase/decrease the rate depends on the phase of congestion control we're in:
  - Discovering available bottleneck bandwidth (Slow Start)
  - Adjusting to bandwidth variations (Congestion Avoidance: AIMD)

Bandwidth discovery with "Slow Start"

- Goal: estimate available bandwidth
  - Start slow (for safety)
  - Ramp up quickly (for efficiency)
- Consider
  - RTT = 100ms, MSS=1000bytes
  - Window size to fill 1Mbps of BW = 12.5 packets
  - Window size to fill 1Gbps = 12,500 packets
  - Either is possible!

Slow Start phase

- Sender starts at a slow rate, but increases exponentially until first loss
- Start with a small congestion window
  - Initially, CWND = 1
  - So, initial sending rate is MSS/RTT
- Double the CWND for each RTT with no loss

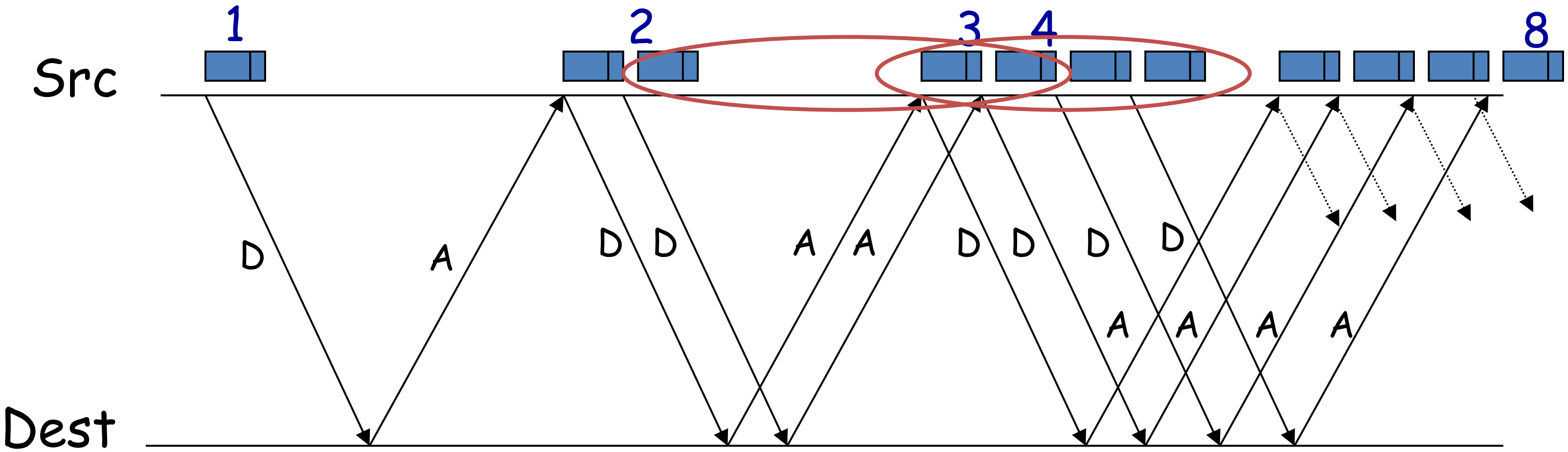
Slow Start in action

- For each RTT: double CWND
  - i.e., for each ACK, CWND += 1

Linear increase per ACK(CWND+1) → exponential increase per RTT (2*CWND)

Slow Start in action

- For each RTT: double CWND
  - i.e., for each ACK, CWND += 1



When does Slow Start stop?

- Slow Start gives an estimate of available bandwidth
  - At some point, there will be loss
- Introduce a "slow start threshold" (ssthresh)
  - Initialized to a large value
- If CWND > ssthresh, stop Slow Start

Adjusting to varying bandwidth

- CWND > ssthresh
  - Stop rapid growth and focus on maintenance
- Now, want to track variations in this available bandwidth, oscillating around its current value
  - Repeated probing (rate increase) and backoff (decrease)
- TCP uses: "Additive Increase Multiplicative Decrease" (AIMD)

AIMD

- Additive increase: when CWND> ssthresh
  - For each ACK, CWND = CWND+ 1/CWND
  - CWND is increased by one only if all segments in a CWND have been acknowledged
- Multiplicative decrease
- On 3 duplicate ACKs (packet loss event)
  - ssthresh = CWND/2
  - CWND= ssthresh
  - Enter Congestion Avoidance: cwnd increases by 1 (linearly instead of exponentially) after each RTT
- On timeout event
  - ssthresh = CWND/2
  - CWND = 1
  - Initiate Slow Start

Illustration of Window

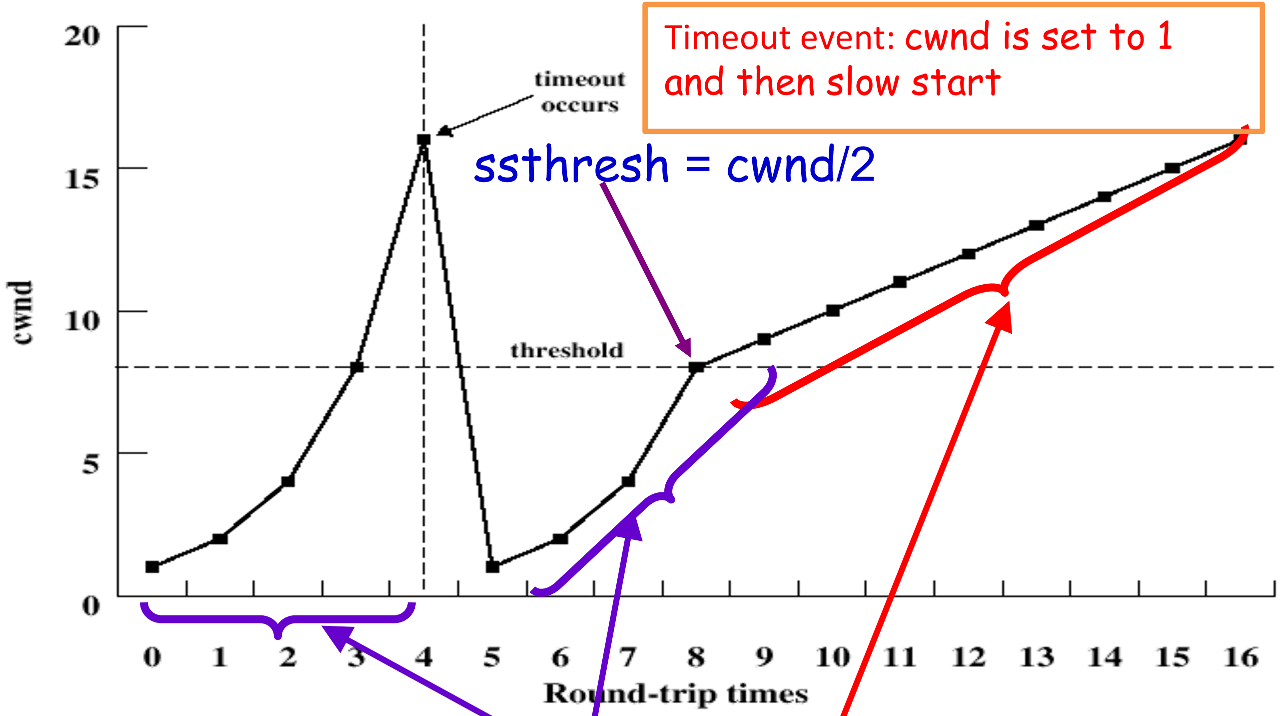
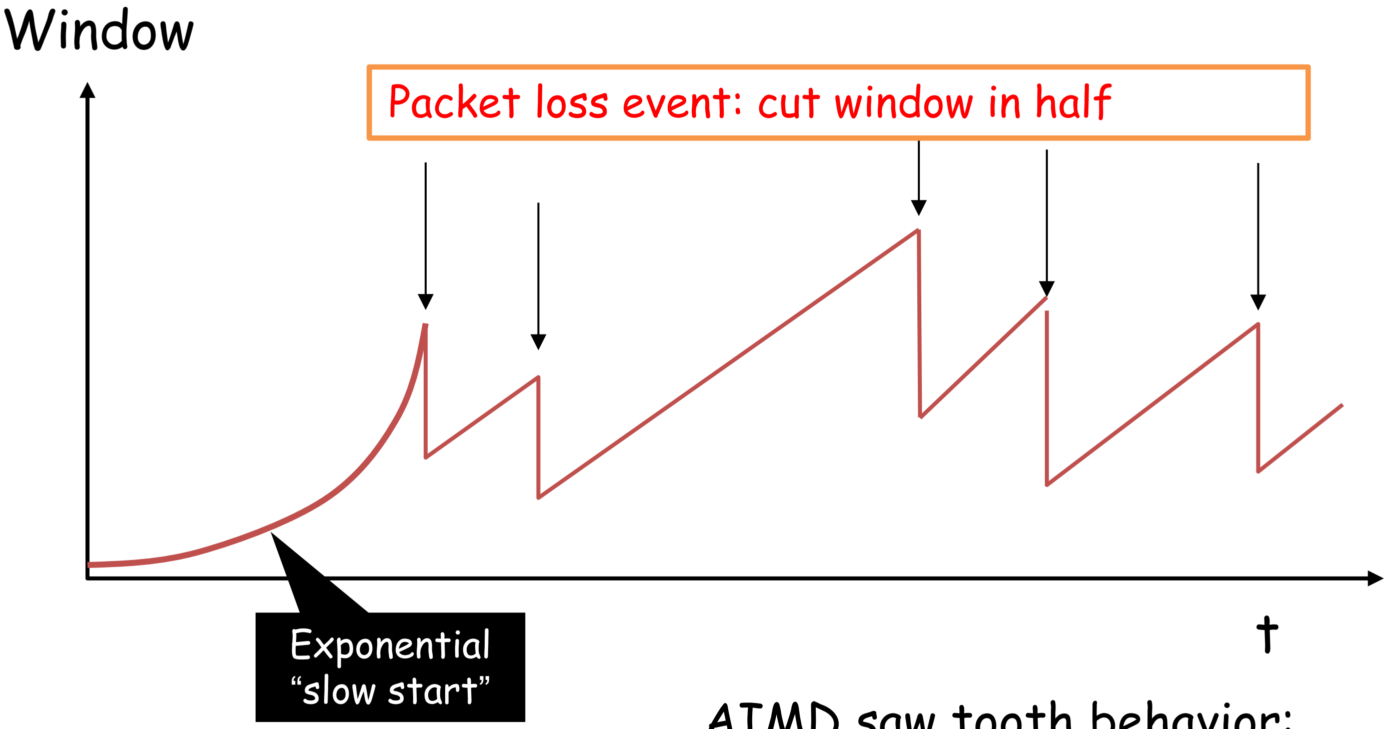


Figure 17.14 Illustration of Slow Start and Congestion Avoidance

Leads to the TCP "Sawtooth"



AIMD saw tooth behavior:
probing for bandwidth

Why AIMD?

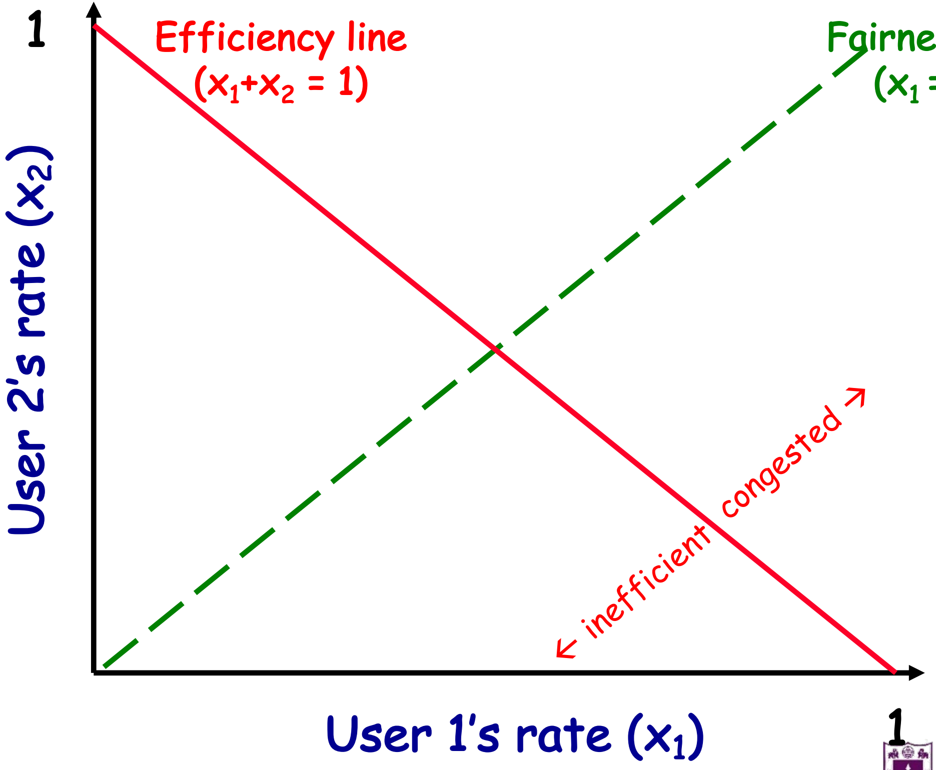
- Recall the three issues
  - Finding available bottleneck bandwidth
  - Adjusting to bandwidth variations
  - Sharing bandwidth
- Two goals for bandwidth sharing
  - Efficiency: High utilization of link bandwidth
  - Fairness: Each flow gets equal share

Why AIMD?

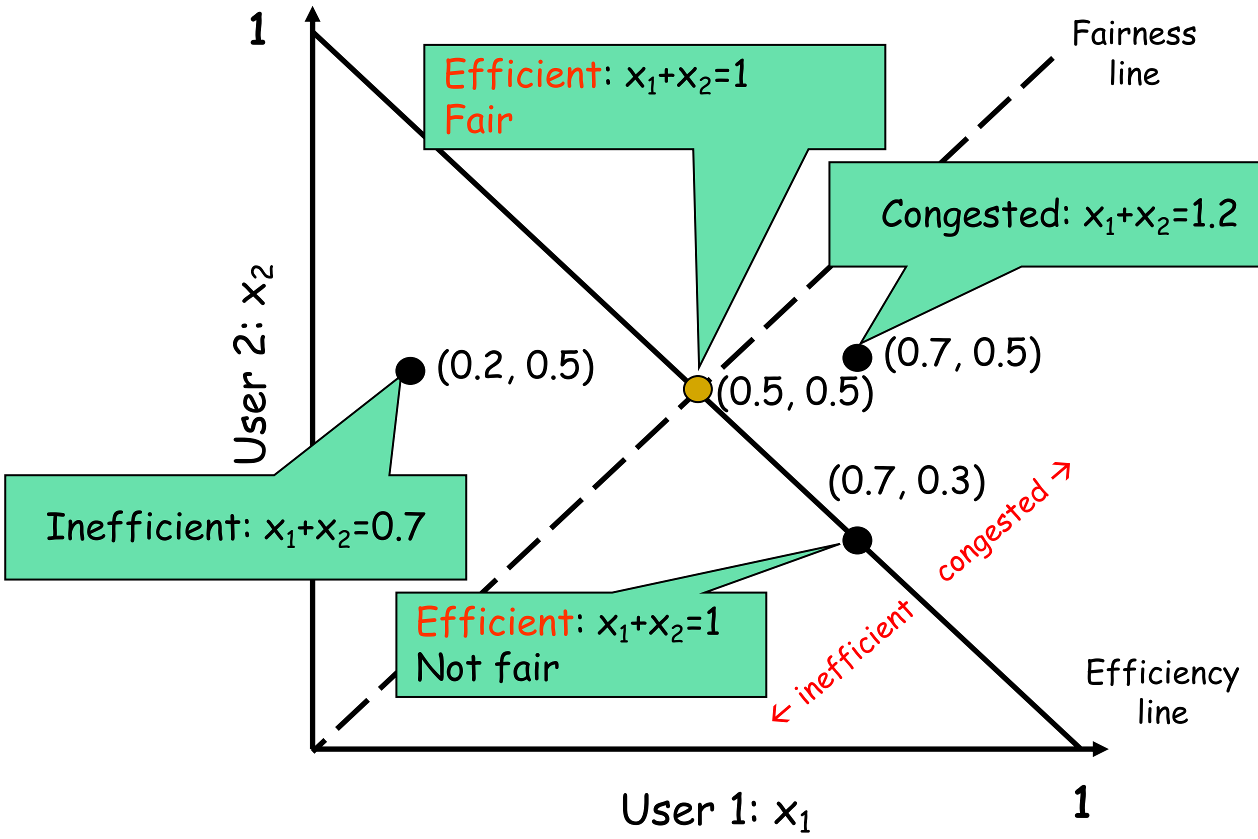
- Every RTT, we can do
  - Multiplicative increase or decrease: CWND→ a*CWND
  - Additive increase or decrease: CWND→ CWND + b
- Four alternatives:
  - AIAD: gentle increase, gentle decrease
  - AIMD: gentle increase, drastic decrease
  - MIAD: drastic increase, gentle decrease
  - MIMD: drastic increase and decrease

Simple model of congestion control

- Two users
  - ➢ rates x1 and x2
- Congestion when x1+x2 > 1
- Unused capacity when x1+x2 < 1
- Fair when x1 =x2

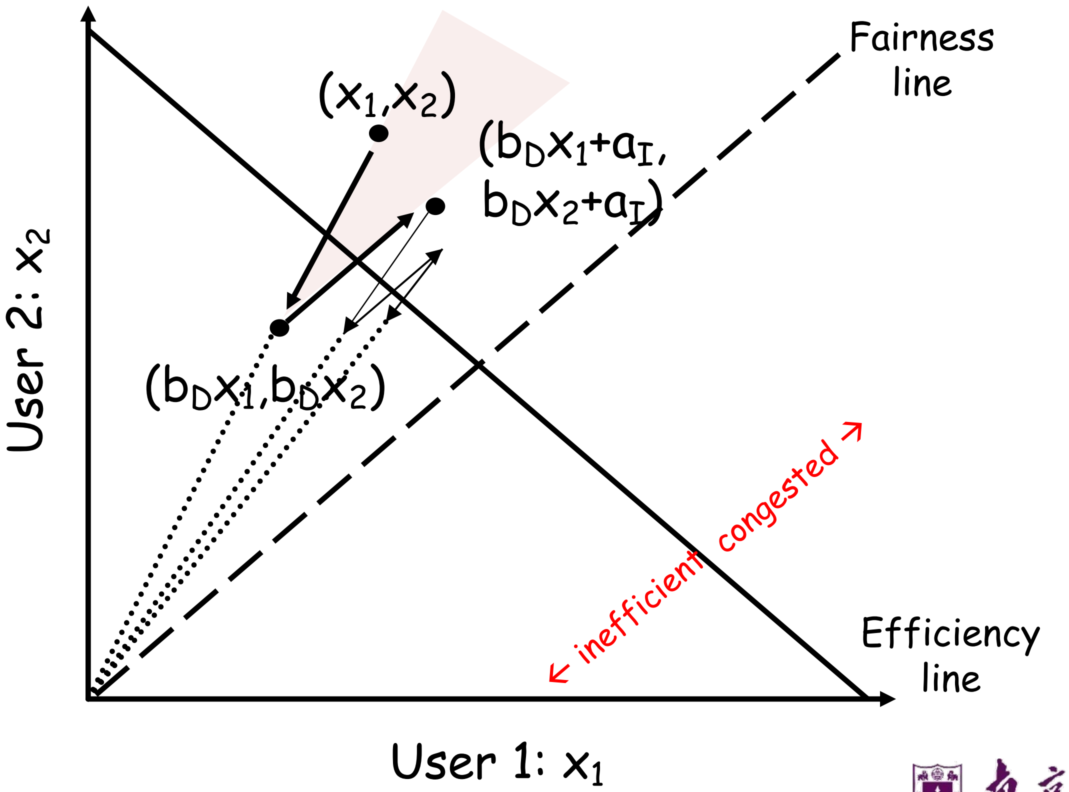


Example


1
Efficient: x1+x2=1
Fair
Fairness line
Congested: x1+x2=1.2
(0.2, 0.5)
(0.5, 0.5)
(0.7, 0.5)
User 2: x2
(0.7, 0.3)
congested →
Inefficient: x1+x2=0.7
Efficient: x1+x2=1
Not fair
← inefficient
Efficiency line
User 1: x1
1

AIMD

- Increase: $x+a_I$
- Decrease: $x*b_D$
- Converges to fairness



Outline

- TCP flow control
- TCP congestion control
- TCP congestion control wrap-up
- Router assisted congestion control

Fast recovery

- Idea: Grant the sender temporary "credit" for each dupACK so as to keep packets in flight
- If dupACKcount = 3
  - ssthresh = CWND/2
  - CWND = ssthresh + 3
- While in fast recovery
  - CWND = CWND + 1 for each additional dupACK
- Exit fast recovery after receiving new ACK
  - set CWND = ssthresh

Example

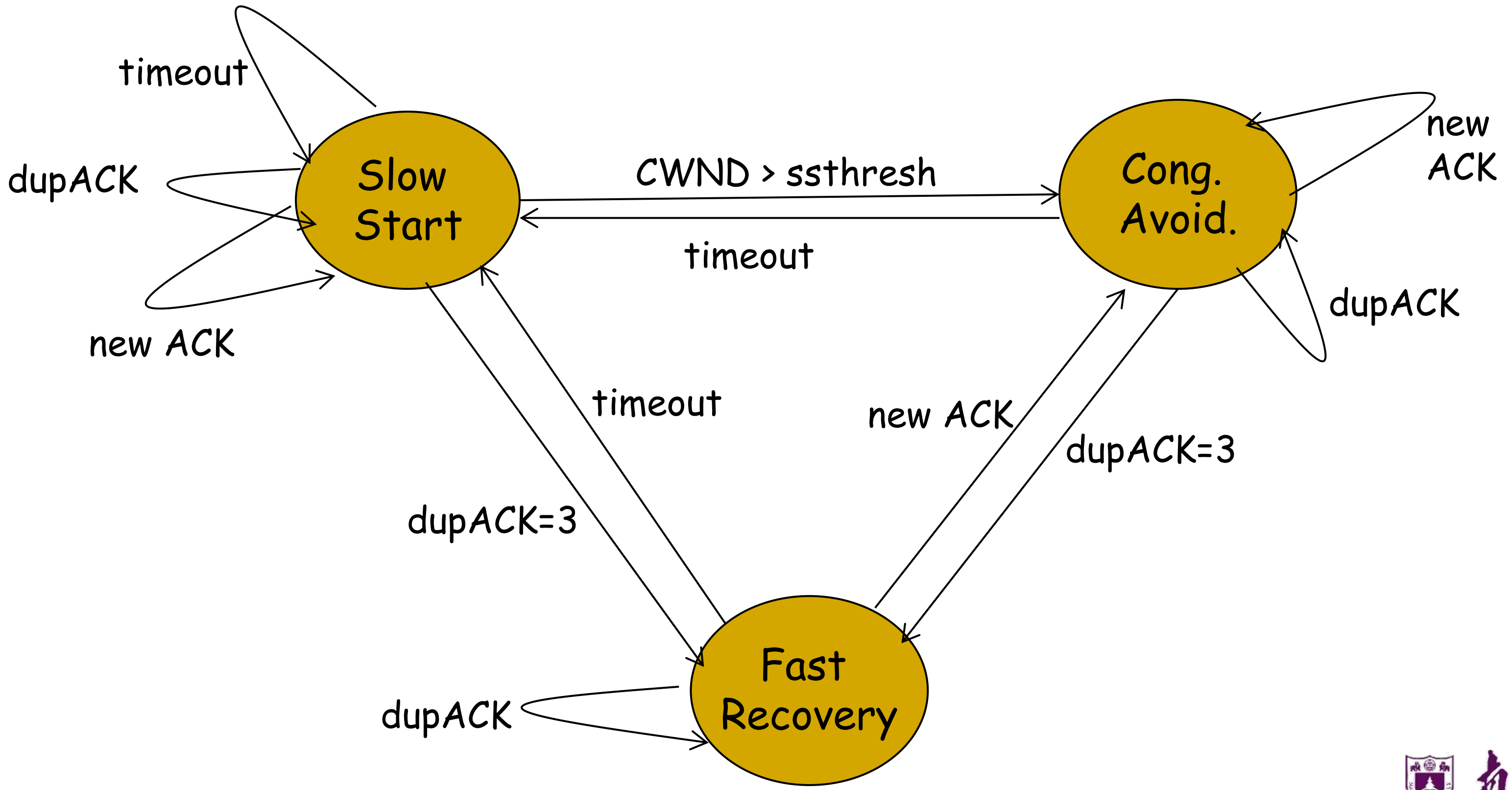
- Consider a TCP connection with:
  - CWND=10 packets
  - Last ACK was for packet # 101
    - ✓ i.e., receiver expecting next packet to have seq. no. 101
- 10 packets [101, 102, 103,…, 110] are in flight
  - Packet 101 is dropped

Timeline: [1~~0~~1, 102, ..., 110]

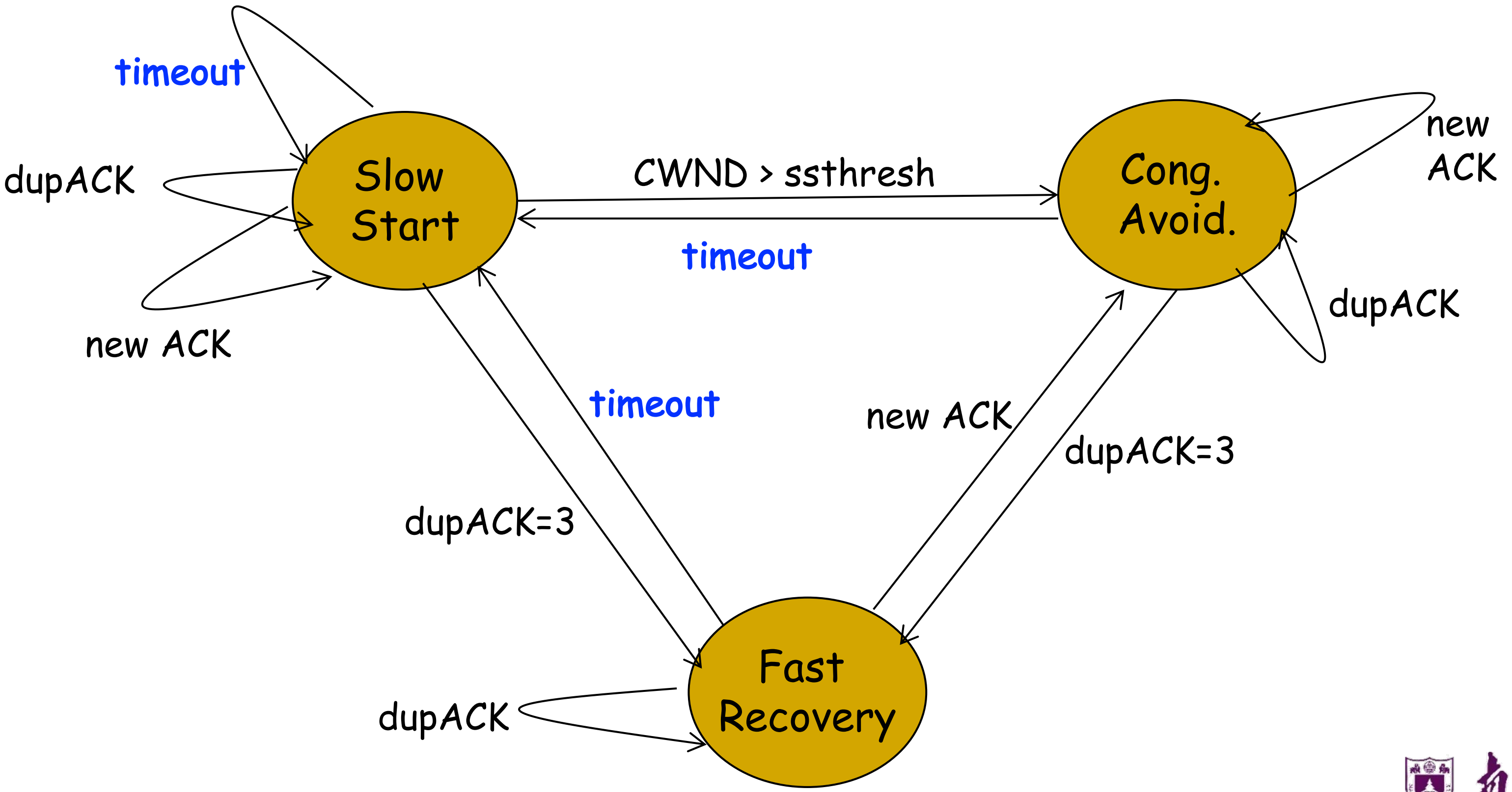
- ACK 101 (due to 102) cwnd=10 dup#1
- ACK 101 (due to 103) cwnd=10 dup#2
- ACK 101 (due to 104) cwnd=10 dup#3
- RETRANSMIT 101 ssthresh=5 cwnd= 8 (5+3)
- ACK 101 (due to 105) cwnd= 9 (no xmit)
- ACK 101 (due to 106) cwnd=10 (no xmit)
- ACK 101 (due to 107) cwnd=11 (xmit 111)
- ACK 101 (due to 108) cwnd=12 (xmit 112)
- ACK 101 (due to 109) cwnd=13 (xmit 113)
- ACK 101 (due to 110) cwnd=14 (xmit 114)
- ACK 111 (due to 101) cwnd = 5 (xmit 115) ← exiting fast recovery
- Packets 111-114 already in flight
- ACK 112 (due to 111) cwnd = 5 + 1/5 ← back in cong. avoidance

南京大學
NANJING UNIVERSITY

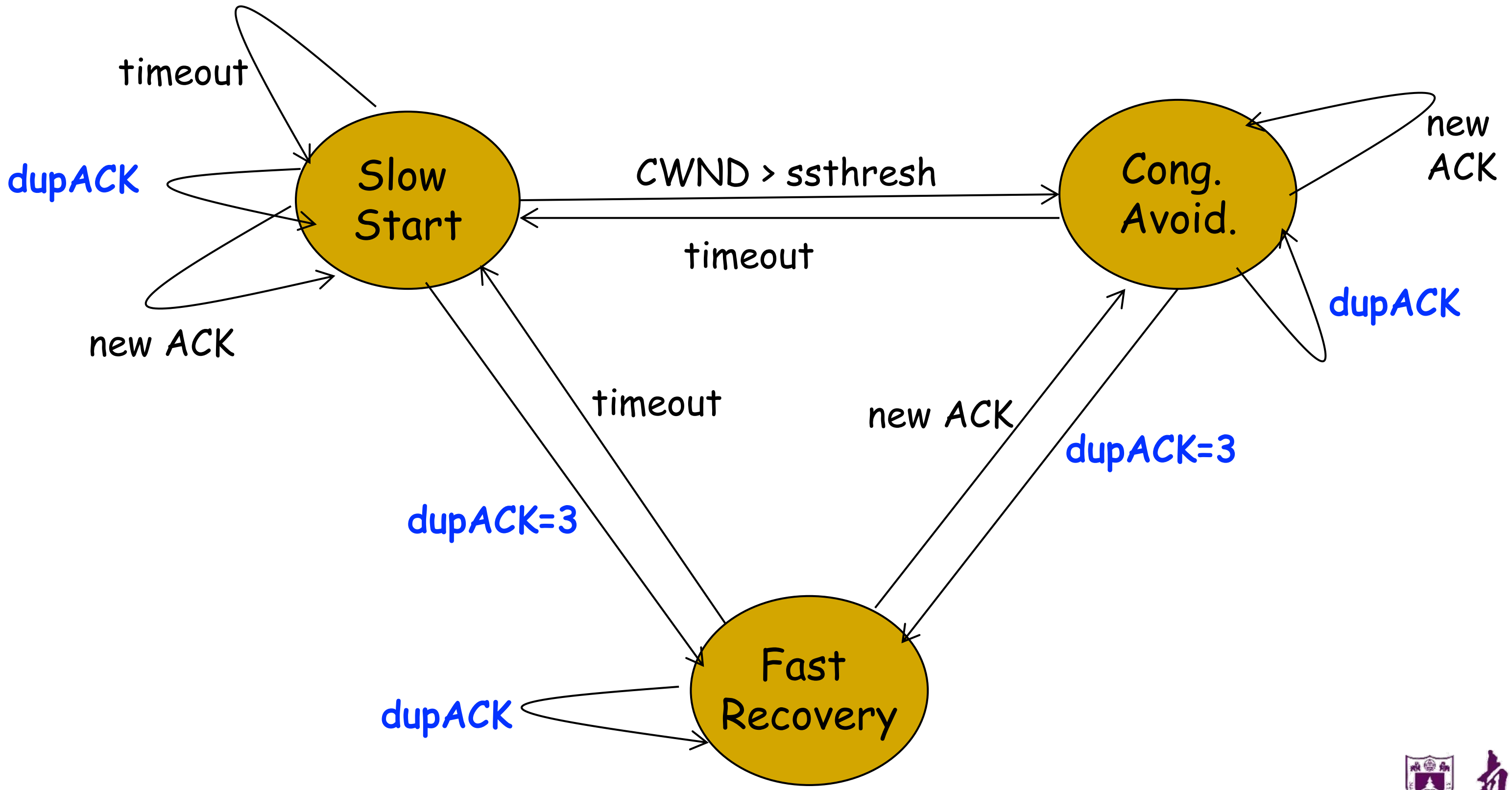
TCP state machine



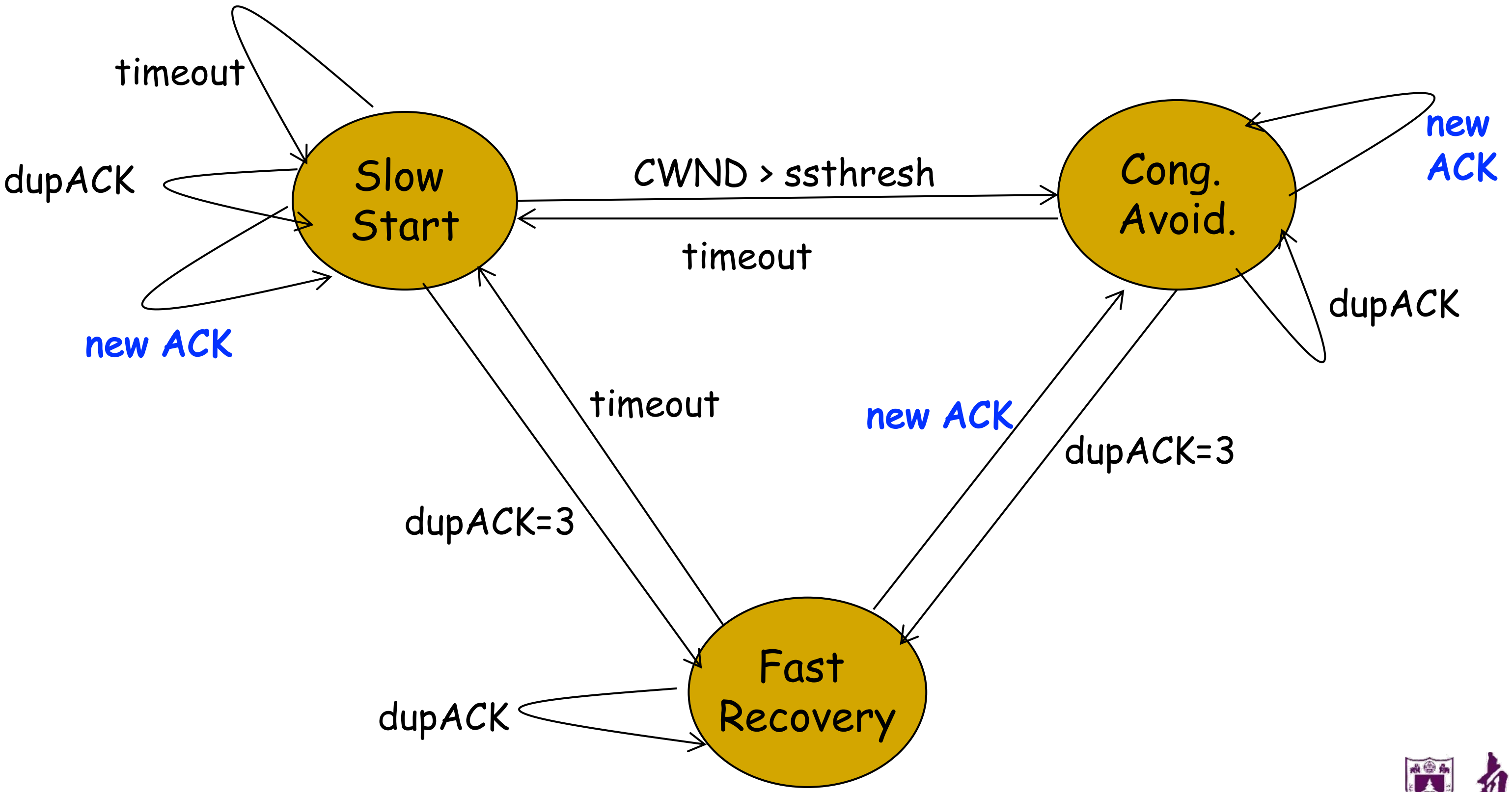
Timeouts → Slow Start



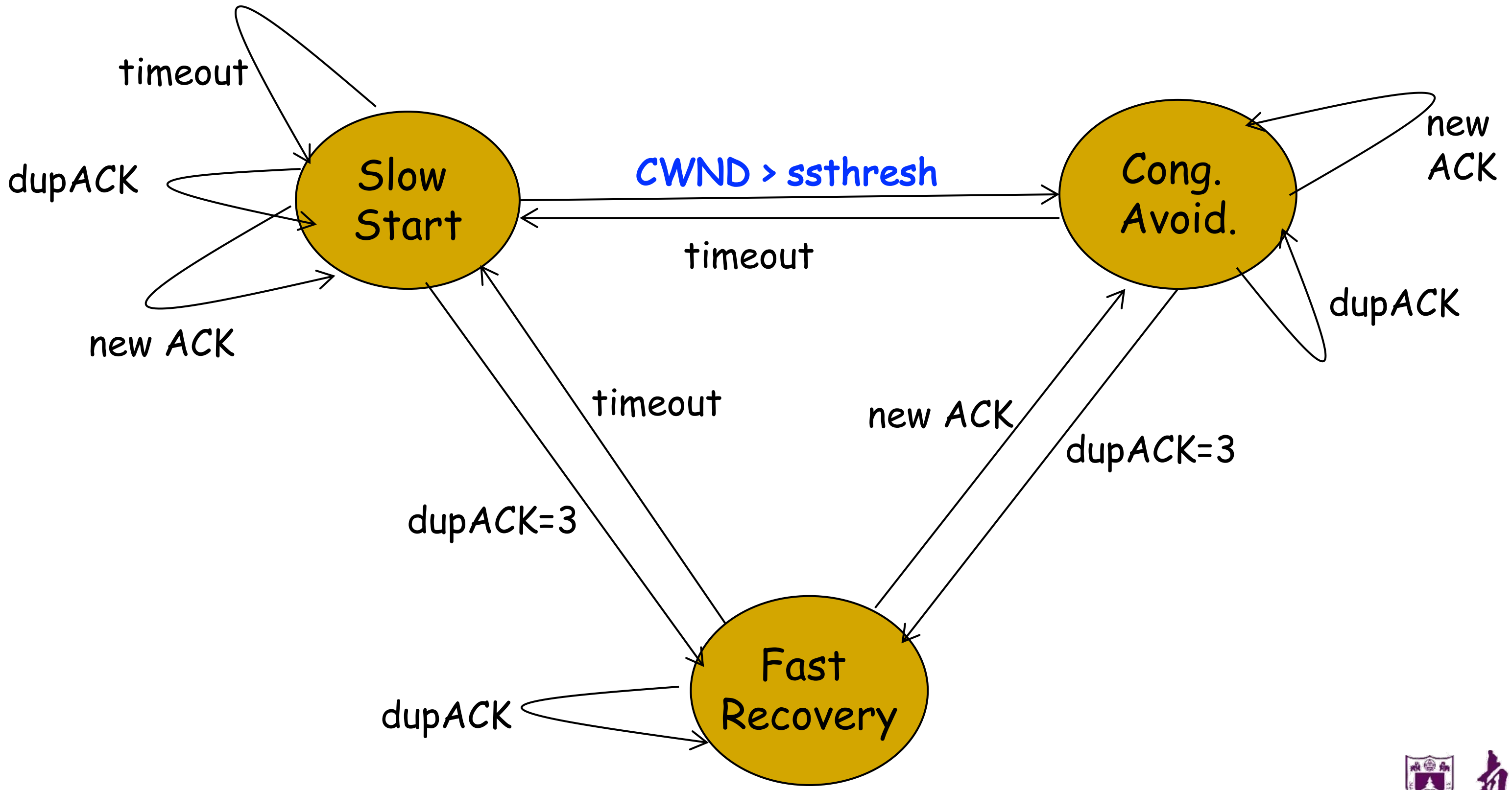
dupACKs → Fast Recovery



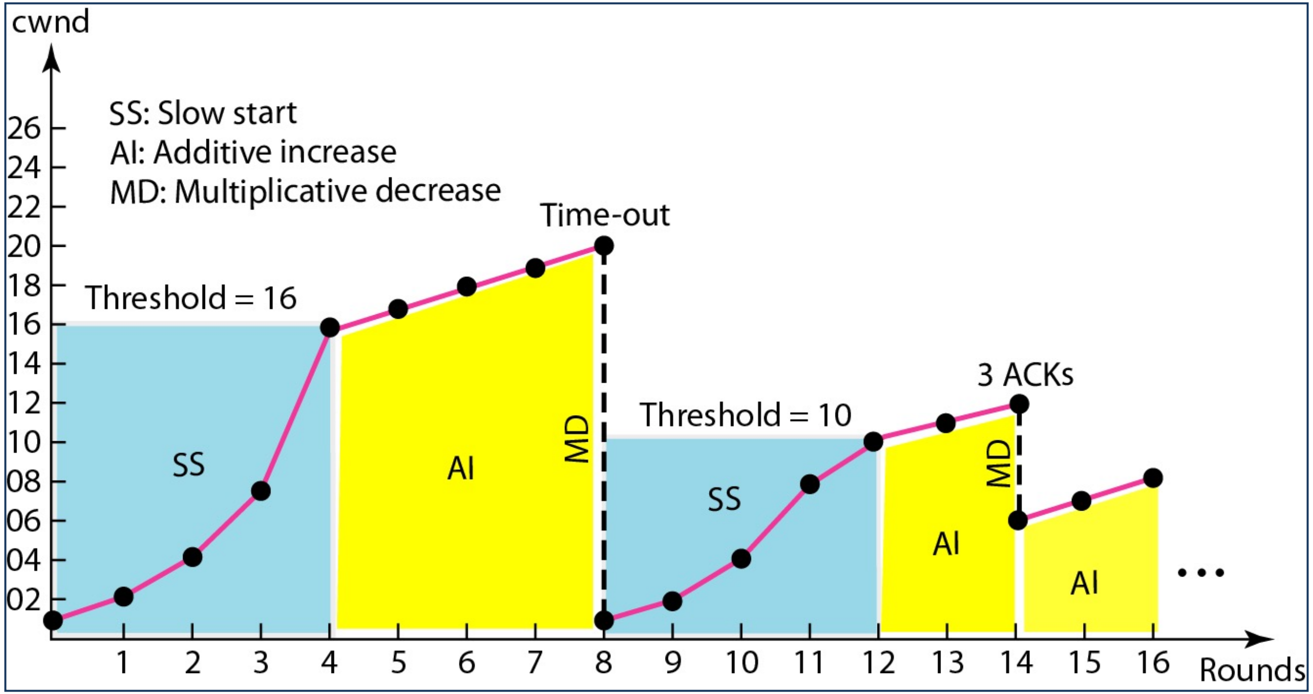
New ACK changes state ONLY from Fast Recovery



TCP state machine



Timeout and Dup-ack



TCP flavors

- TCP-Tahoe
  - CWND =1 on 3 dupACKs
- TCP-Reno
  - CWND =1 on timeout
  - CWND = CWND/2 on 3 dupACKs
- TCP-newReno
  - TCP-Reno + improved fast recovery
- TCP-SACK
  - Incorporates selective acknowledgements

Our default assumption

How can they coexist?

- All follow the same principle
  - Increase CWND on good news
  - Decrease CWND on bad news

Outline

- TCP flow control
- TCP congestion control
- TCP congestion control wrap-up
- Router assisted congestion control

Recap: TCP problems

- Misled by non-congestion losses
- Fills up queues leading to high delays
- Short flows complete before discovering available capacity
- AIMD impractical for high speed links
- Saw tooth discovery too choppy for some apps
- Unfair under heterogeneous RTTs
- Tight coupling with reliability mechanisms
- End hosts can cheat

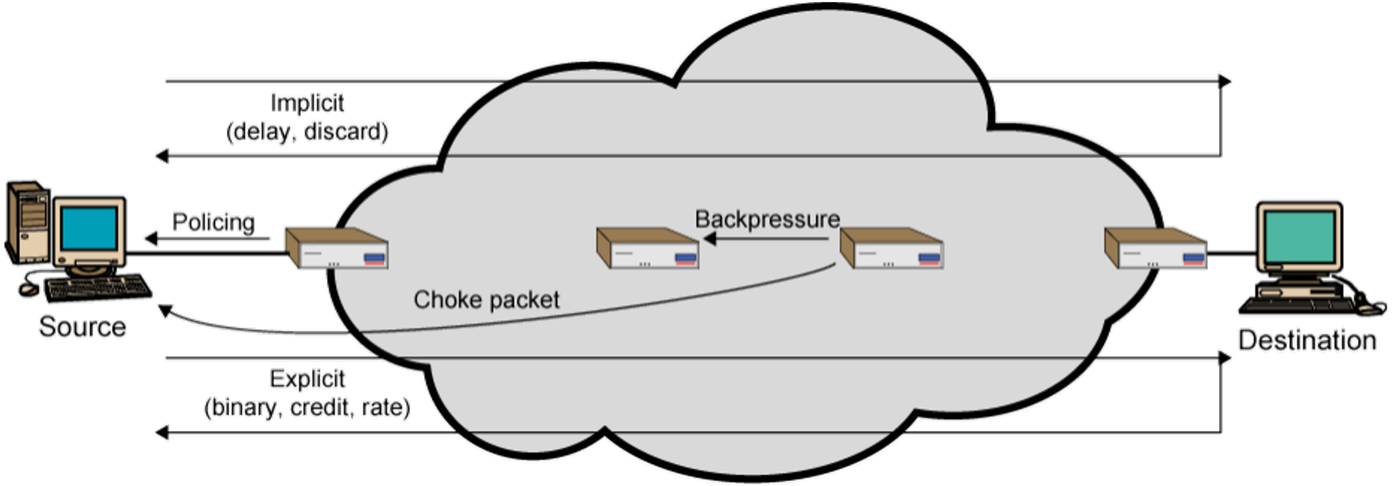
Routers tell endpoints if they're congested

Routers tell endpoints what rate to send at

Routers enforce fair sharing

Could fix many of these with some help from routers!

Mechanisms for Congestion Control



- Choke Packet
- Backpressure
- Warning bit
- Random early discard
- Fair Queuing (FQ)

- 抑制分组
- 反压
- 警告位
- 随机早期丢弃
- 公平队列

Warning Bit

Explicit Congestion Notification (ECN)

- Single bit in packet header; set by congested routers
  - If data packet has bit set, then ACK has ECN bit set
- Many options for when routers set the bit
  - Tradeoff between (link) utilization and (packet) delay
- Congestion semantics can be exactly like that of drop
  - i.e., end-host reacts as though it saw a drop

ECN

- Advantages:
  - ➢ Don't confuse corruption with congestion; recovery w/ rate adjustment
  - ➢ Can serve as an early indicator of congestion to avoid delays
  - ➢ Easy (easier) to incrementally deploy
    - ✓ Today: defined in RFC 3168 using ToS/DSCP bits in the IP header
    - ✓ Common in datacenters

- Use ECN as congestion markers
  - Whenever I get an ECN bit set, I have to pay $$

课程习题（作业）——截止日期：4月15日晚23:59

- **课本187-195页**：第P27、P32、P40、P48、P52、P54题
- 提交方式：https://selearning.nju.edu.cn/（教学支持系统）

- 命名：学号+姓名+第*章。
- 若提交遇到问题请及时发邮件或在下一次上课时反馈。

课程习题（作业）——截止日期：4月15日晚23:59

P27. 主机 A 和 B 经一条 TCP 连接通信，并且主机 B 已经收到了来自 A 的最长为 126 字节的所有字节。假定主机 A 随后向主机 B 发送两个紧接着的报文段。第一个和第二个报文段分别包含了 80 字节和 40 字节的数据。在第一个报文段中，序号是 127，源端口号是 302，目的地端口号是 80。无论何时主机 B 接收到来自主机 A 的报文段，它都会发送确认。

a. 在从主机 A 发往 B 的第二个报文段中，序号、源端口号和目的端口号各是什么？

b. 如果第一个报文段在第二个报文段之前到达，在第一个到达报文段的确认中，确认号、源端口号和目的端口号各是什么？

c. 如果第二个报文段在第一个报文段之前到达，在第一个到达报文段的确认中，确认号是什么？

d. 假定由 A 发送的两个报文段按序到达 B。第一个确认丢失了而第二个确认在第一个超时间隔之后到达。画出时序图，显示这些报文段和发送的所有其他报文段和确认。（假设没有其他分组丢失。）对于图上每个报文段，标出序号和数据的字节数量；对于你增加的每个应答，标出确认号。

P32. 考虑 TCP 估计 RTT 的过程。假设 $\alpha = 0.1$，令 SampleRTT_{T1} 设置为最新样本 RTT，令 SampleRTT_{T2} 设置为下一个最新样本 RTT，等等。

a. 对于一个给定的 TCP 连接，假定 4 个确认报文相继到达，带有 4 个对应的 RTT 值：SampleRTT_{T4}、SampleRTT_{T3}、SampleRTT_{T2} 和 SampleRTT_{T1}。根据这 4 个样本 RTT 表示 EstimatedRTT。

b. 将你得到的公式一般化到 n 个 RTT 样本的情况。

c. 对于在（b）中得到的公式，令 n 趋于无穷。试说明为什么这个平均过程被称为指数移动平均。

课程习题（作业）——截止日期：4月15日晚23:59

P40. 考虑图 3-61 假设 TCP Reno 是一个经历如上所示行为的协议，回答下列问题。在各种情况中，简要地论证你的回答。

a. 指出 TCP 慢启动运行时的时间间隔。

b. 指出 TCP 拥塞避免运行时的时间间隔。

c. 在第 16 个传输轮回之后，报文段的丢失是根据 3 个冗余 ACK 还是根据超时检测出来的？

d. 在第 22 个传输轮回之后，报文段的丢失是根据 3 个冗余 ACK 还是根据超时检测出来的？

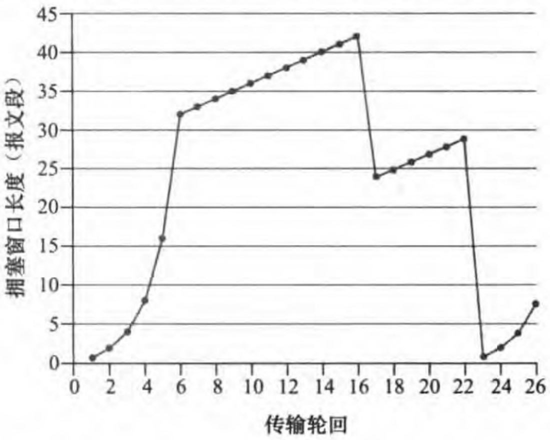


图 3-61 TCP 窗口长度作为时间的函数

e. 在第 1 个传输轮回里，ssthresh 的初始值设置为多少？

f. 在第 18 个传输轮回里，ssthresh 的值设置为多少？

g. 在第 24 个传输轮回里，ssthresh 的值设置为多少？

h. 在哪个传输轮回内发送第 70 个报文段？

i. 假定在第 26 个传输轮回后，通过收到 3 个冗余 ACK 检测出有分组丢失，拥塞的窗口长度和 ssthresh 的值应当是多少？

j. 假定使用 TCP Tahoe（而不是 TCP Reno），并假定在第 16 个传输轮回收到 3 个冗余 ACK。在第 19 个传输轮回，ssthresh 和拥塞窗口长度是什么？

k. 再次假设使用 TCP Tahoe，在第 22 个传输轮回有一个超时事件。从第 17 个传输轮回到第 22 个传输轮回（包括这两个传输轮回），一共发送了多少分组？

课程习题（作业）——截止日期：4月15日晚23:59

P48 考虑仅有一条单一的 TCP（Reno）连接使用一条 10Mbps 链路，且该链路没有缓存任何数据。假设这条链路是发送主机和接收主机之间的唯一拥塞链路。假定某 TCP 发送方向接收方有一个大文件要发送，而接收方的接收缓存比拥塞窗口要大得多。我们也做下列假设：每个 TCP 报文段长度为 1500 字节；该连接的双向传播时延是 150ms；并且该 TCP 连接总是处于拥塞避免阶段，即忽略了慢启动。

a. 这条 TCP 连接能够取得的最大窗口长度（以报文段计）是多少?

b. 这条 TCP 连接的平均窗口长度（以报文段计）和平均吞吐量（以 bps 计）是多少?

c. 这条 TCP 连接在从丢包恢复后，再次到达其最大窗口要经历多长时间?

课程习题（作业）——截止日期：4月15日晚23:59

P52 考虑一种简化的 TCP 的 AIMD 算法，其中拥塞窗口长度用报文段的数量来度量，而不是用字节度量。在加性增中，每个 RTT 拥塞窗口长度增加一个报文段。在乘性减中，拥塞窗口长度减小一半（如果结果不是一个整数，向下取整到最近的整数）。假设两条 TCP 连接 C1 和 C2，它们共享一条速率为每秒 30 个报文段的单一拥塞链路。假设 C1 和 C2 均处于拥塞避免阶段。连接 C1 的 RTT 是 50ms，连接 C2 的 RTT 是 100ms。假设当链路中的数据速率超过了链路的速率时，所有 TCP 连接经受数据报文段丢失。

a. 如果在时刻 t_0，C1 和 C2 具有 10 个报文段的拥塞窗口，在 1000ms 后它们的拥塞窗口为多长？

b. 经长时间运行，这两条连接将取得共享该拥塞链路的相同的带宽吗？

P54 考虑修改 TCP 的拥塞控制算法。不使用加性增，使用乘性增。无论何时某 TCP 收到一个合法的 ACK，就将其窗口长度增加一个小正数 a（$0 < a < 1$）。求出丢包率 L 和最大拥塞窗口 W 之间的函数关系。论证：对于这种修正的 TCP，无论 TCP 的平均吞吐量如何，一条 TCP 连接将其拥塞窗口长度从 $W/2$ 增加到 W，总是需要相同的时间。

助教信息

甘世维	博士	计算机科学与技术系	gswycf@qq.com
马诗源	博士	计算机科学与技术系	dg20330018@smail.nju.edu.cn
樊龙	博士	计算机科学与技术系	fanl@smail.nju.edu.cn
王学浩	硕士	计算机科学与技术系	522023330098@smail.nju.edu.cn
俞涛	硕士	计算机科学与技术系	502023330077@smail.nju.edu.cn

Q & A

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/