# Transport Congestion Control: TCP/Reno, Analysis

# Quiz

- Explain why TCP's AIMD increases fairness among network flows while maintaining efficiency.

# TCP Congestion Control

- Closed-loop, end-to-end, window-based congestion control
- Designed by Van Jacobson in late 1980s, based on the AIMD alg. of Dah-Ming Chu and Raj Jain
- Works well so far: the bandwidth of the Internet has increased by more than 200,000 times
- Many versions
  - TCP/Tahoe: this is a less optimized version
  - TCP/Reno: many OSs today implement Reno type congestion control
  - TCP/Vegas: not currently used

For more details: see TCP/IP illustrated; or read http://lxr.linux.no/source/net/ipv4/tcp_input.c for linux implementation

# TCP/Reno Congestion Detection

- Detect congestion in two cases and react differently:
  - 3 dup ACKs
  - timeout event

Philosophy:

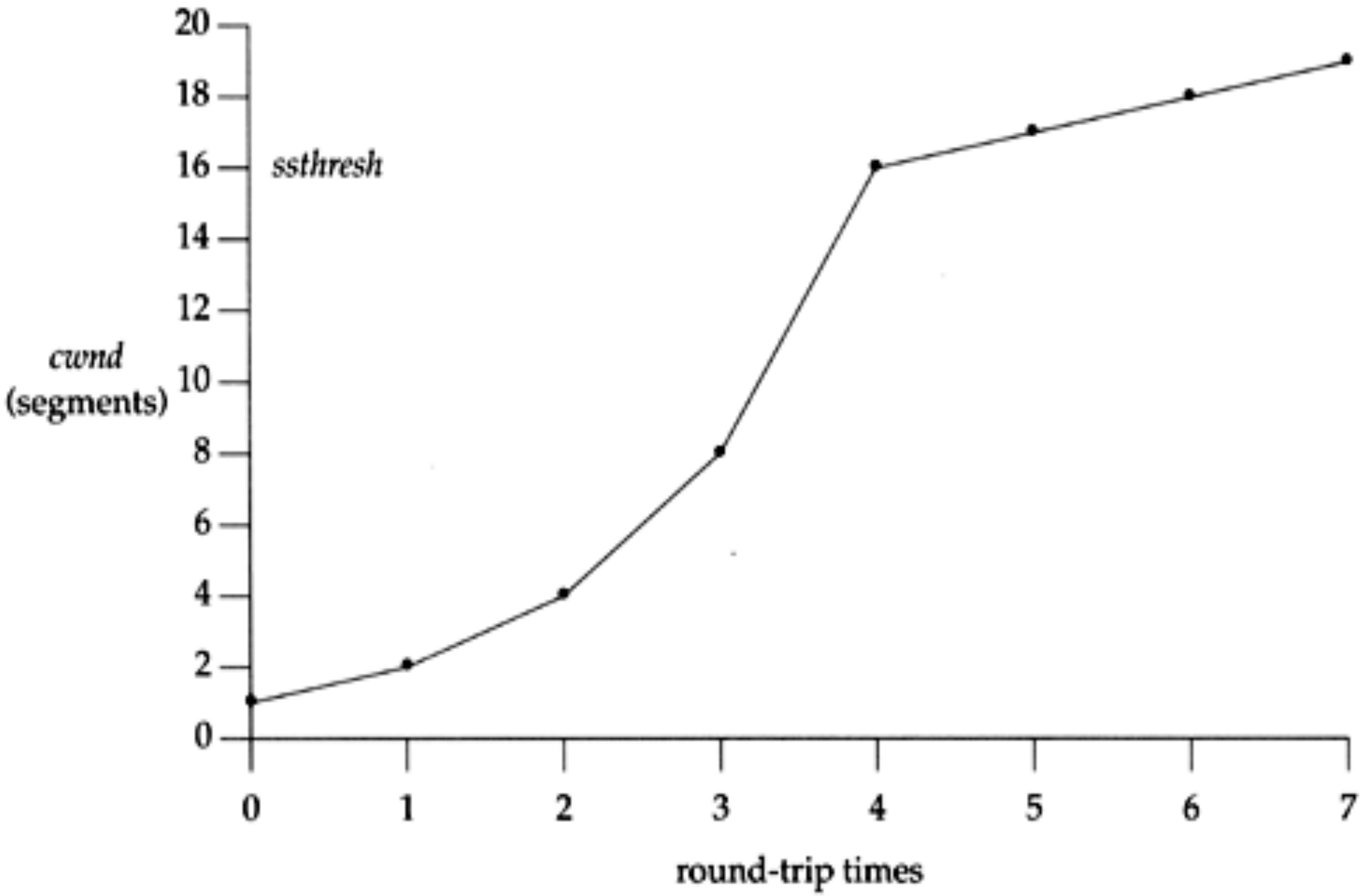
- 3 dup ACKs indicates network capable of delivering some segments
- timeout is "more alarming"

# Basic Structure

- Two "phases"
  - Slow-start: MI
  - Congestion avoidance: AIMD

- Important variables:
  - *cwnd*: congestion window size
  - *ssthresh*: threshold between the slow-start phase and the congestion avoidance phase

# Visualization of the Two Phases



# Slow Start: MI

- What is the goal?
  - Getting to equilibrium gradually but quickly

- Implements the MI algorithm
  - Double *cwnd* every RTT until network congested → get a rough estimate of the optimal of *cwnd*

# Slow-start

**Initially:**

cwnd = 1;

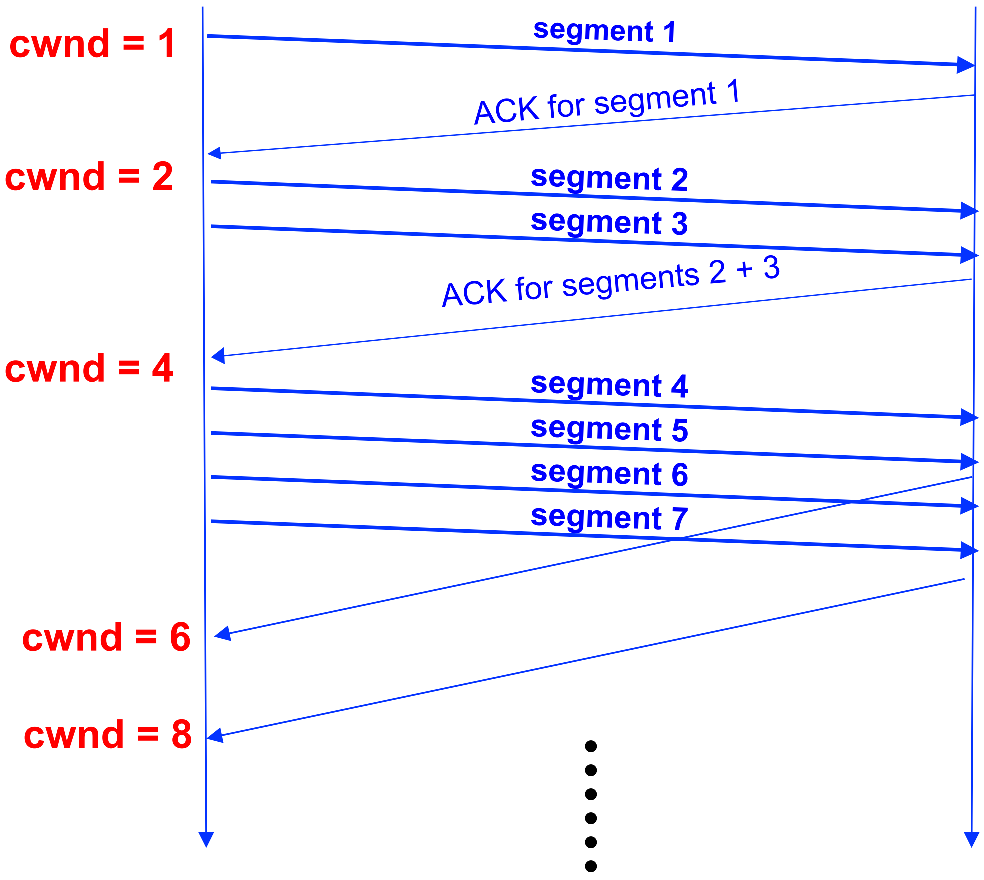
ssthresh = infinite (e.g., 64K);

**For each newly ACKed segment:**

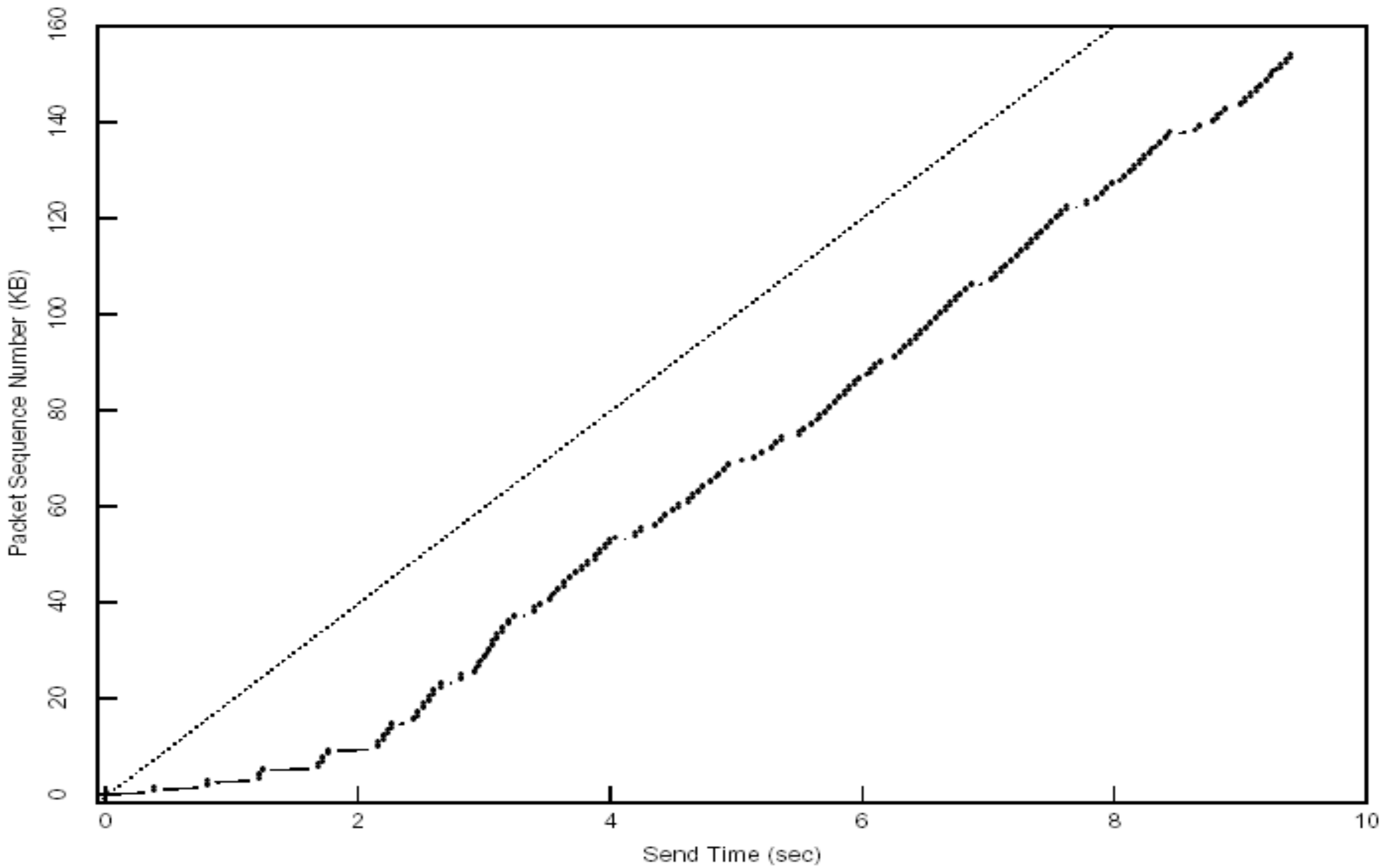
if (cwnd < ssthresh)

/* slow start*/

cwnd = cwnd+1;



# Startup Behavior with Slow-start




See [Jac89]

# TCP/Reno Congestion Avoidance

- Maintains equilibrium and reacts around equilibrium

- Implements the AIMD algorithm
  - Increases window by 1 per round-trip time (how?)
  - Cuts window size
    - To half when detecting congestion by 3 DUP
    - To 1 if timeout
    - If already timeout, *doubles* timeout

# TCP/Reno Congestion Avoidance

**Initially:**

cwnd = 1;

ssthresh = infinite (e.g., 64K);

**For each newly ACKed segment:**

if (cwnd < ssthresh)

/* slow start*/

cwnd = cwnd + 1;

else

/* congestion avoidance; cwnd increases (approx.) by 1 per RTT */

cwnd += 1/cwnd;

**Triple-duplicate ACKs:**

/* multiplicative decrease */

cwnd = ssthresh = cwnd/2;

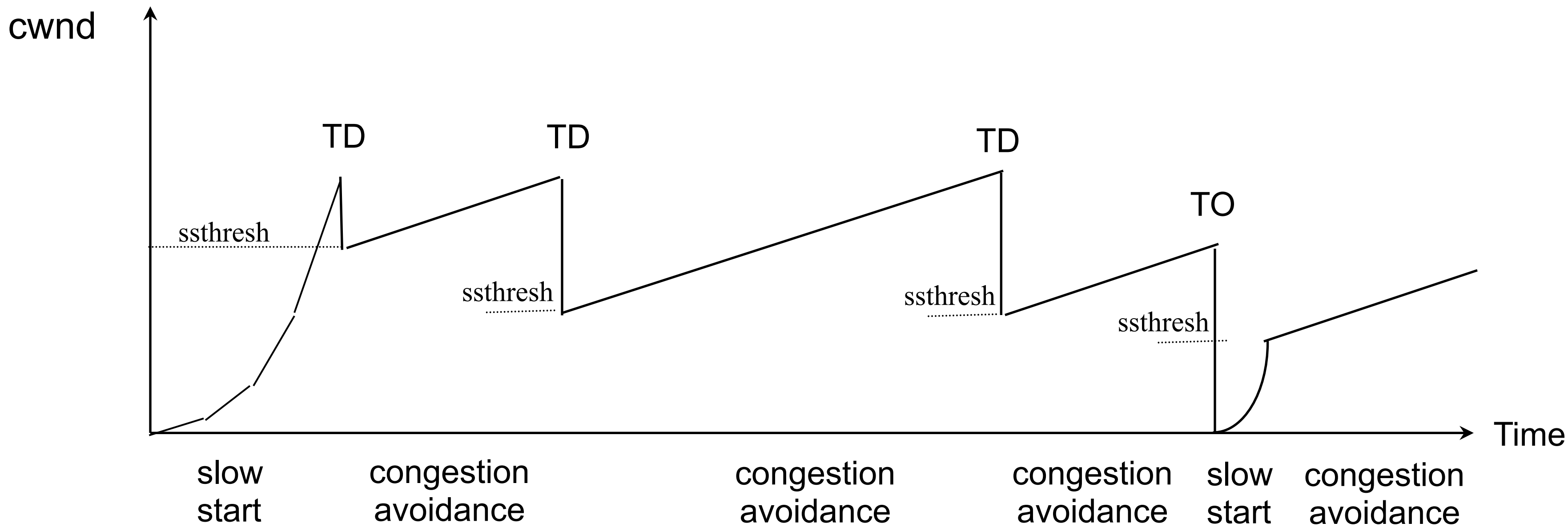
**Timeout:**

ssthresh = cwnd/2;

cwnd = 1;

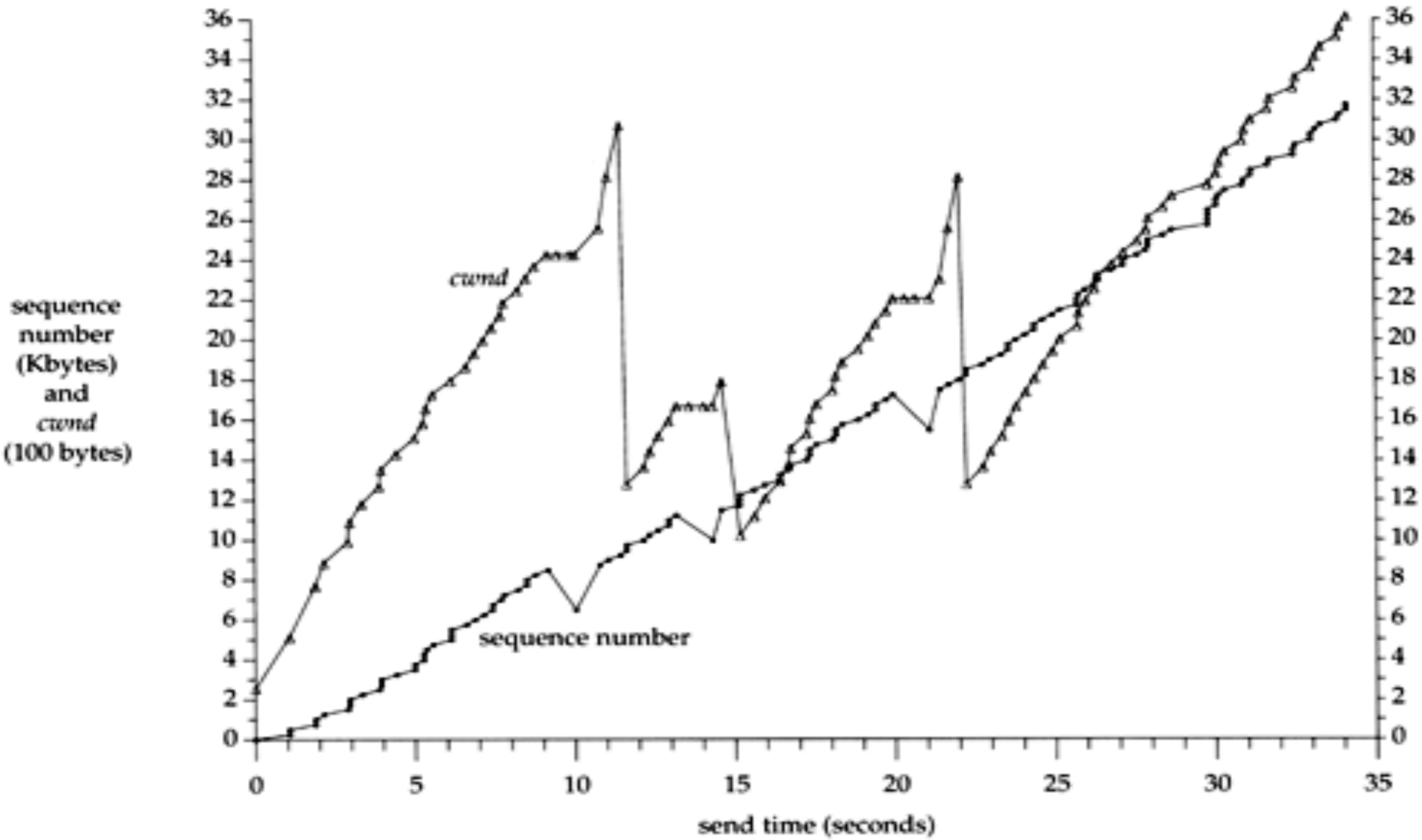
(if already timed out, double timeout value; this is called *exponential backoff*)

# TCP/Reno: Big Picture




TD: Triple duplicate acknowledgements
TO: Timeout

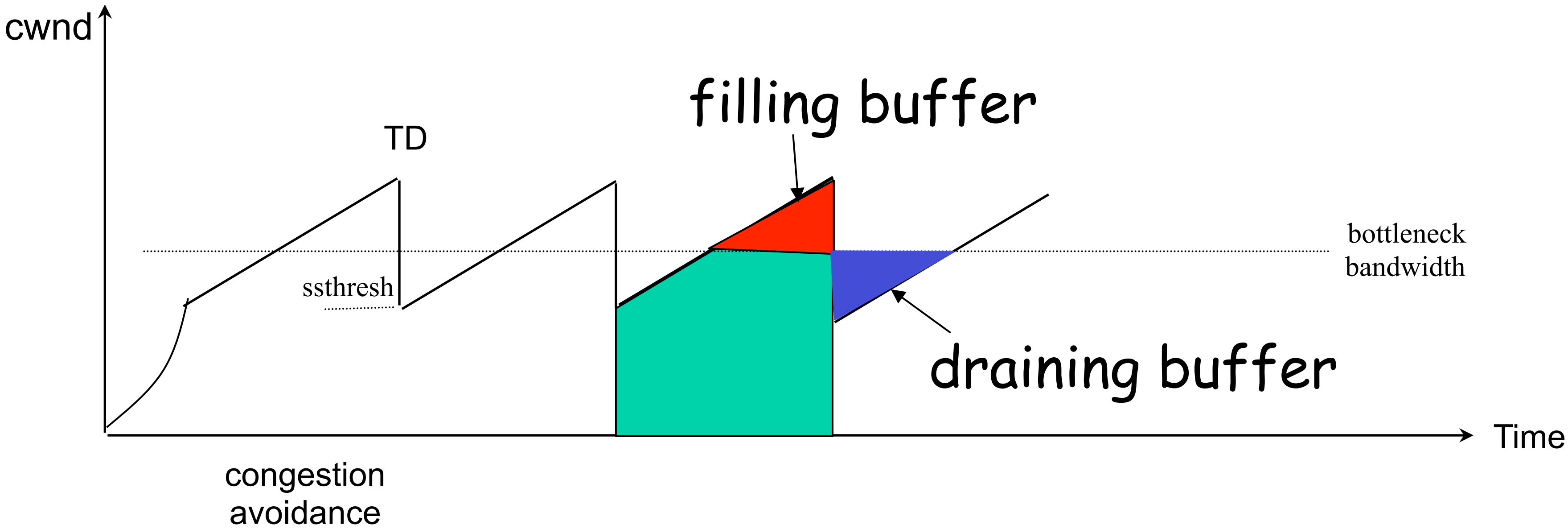
# A Session




Question: when cwnd is cut to half, why sending rate is not?

# TCP/Reno Queueing Dynamics

- Consider congestion avoidance only




There is a filling and draining of buffer process for each TCP flow.

# Outline

- Recap
- Linear congestion control law
- TCP/Reno
- *TCP/Reno throughput analysis*

# Objective

- To understand the throughput of TCP/Reno as a function of RTT (RTT), loss rate (p) and packet size
- We will derive the formula twice, using two setups using two different approaches

# TCP/Reno Throughput Modeling

- Given mean packet loss rate p, mean round-trip time RTT, packet size S
- Consider only the congestion avoidance mode (long flows such as large files)
- Assume no timeout
- Assume mean window size is $W_m$ segments, each with $S$ bytes sent in one $RTT$:

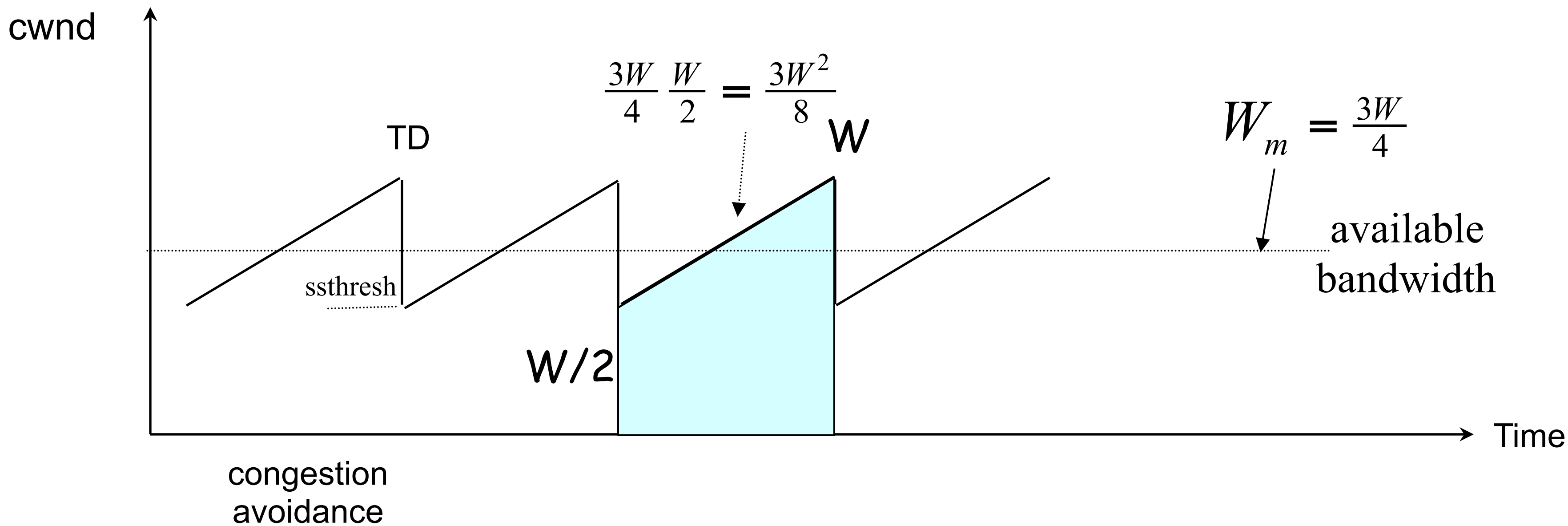
$$\boxed{\text{throughput} \approx \frac{W_m * S}{RTT} \text{ bytes/sec}}$$

# Outline

- Recap
- Linear congestion control law
- TCP/Reno
- TCP/Reno throughput analysis
  - Analysis 1: deterministic

# TCP/Reno Throughput Modeling: Relating W with Loss Rate p

- Consider congestion avoidance only




Assume one packet loss (loss event) per cycle

Total packets send per cycle = (W/2 + W)/2 * W/2 = $3W^2/8$

Thus p = $1/(3W^2/8) = 8/(3W^2)$

$$W = \frac{\sqrt{8/3}}{\sqrt{p}} = \frac{1.6}{\sqrt{p}} \Rightarrow throughput = \frac{S * W_M}{RTT} = \frac{S}{RTT}\frac{3}{4}\frac{1.6}{\sqrt{p}} = \frac{1.2S}{RTT\sqrt{p}}$$

# TCP Futures

- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size W = 83,333 in-flight segments
- Throughput in terms of loss rate:

$$\frac{1.22 \cdot MSS}{RTT\sqrt{Loss}}$$

- ➜ Loss = $2 \cdot 10^{-10}$ Wow
- New versions of TCP for high-speed needed!

# Outline

- Recap
- Linear congestion control law
- TCP/Reno
- TCP/Reno throughput analysis
  - Analysis 1: deterministic
  - Analysis 2: random loss

# TCP/Reno Throughput Modeling

$$\Delta W = \begin{cases} \frac{1}{W} & \text{if the packet is not lost} \\ -\frac{W}{2} & \text{if packet is lost} \end{cases}$$

$$\mathit{mean\ of}\ \Delta W = (1-p)\frac{1}{W} + p(-\frac{W}{2}) = 0$$

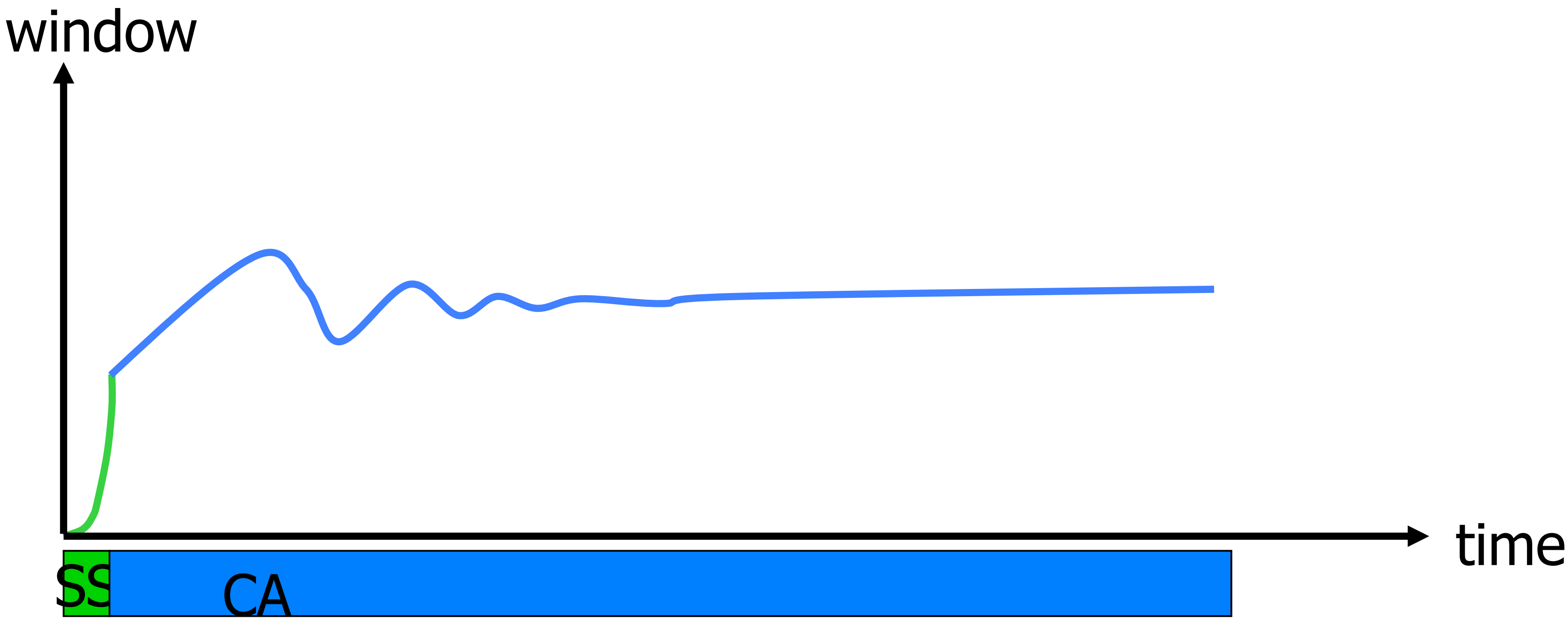
=> $$\mathit{mean\ of}\ W = \sqrt{\frac{2(1-p)}{p}} \approx \frac{1.4}{\sqrt{p}}, \text{ when } p \text{ is small}$$

=> $$\mathit{throughput} \approx \frac{1.4S}{RTT\sqrt{p}}, \text{ when } p \text{ is small}$$

# Outline

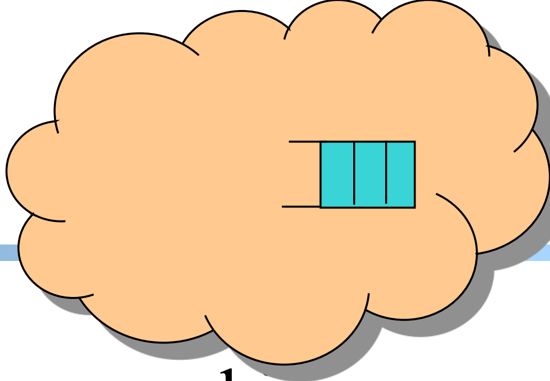
- Recap
- Linear congestion control law
- TCP/Reno
- TCP/Reno throughput analysis
  - Analysis 1: deterministic
  - Analysis 2: random loss
- TCP Vegas

# TCP/Vegas (Brakmo & Peterson 1994)

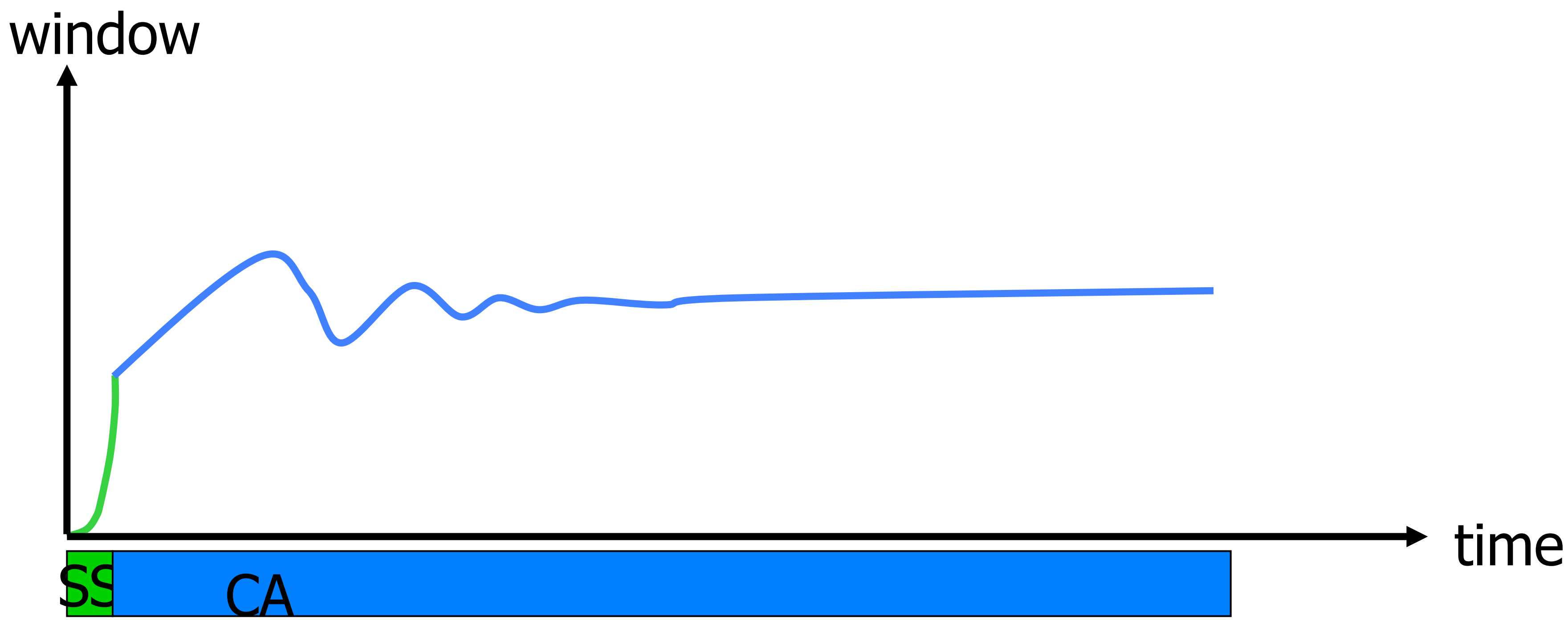



- Idea: try to detect congestion by delay before loss
- Objective: not to overflow the buffer; instead, try to maintain a *constant* number of packets in the bottleneck queue

# TCP/Vegas: Key Question

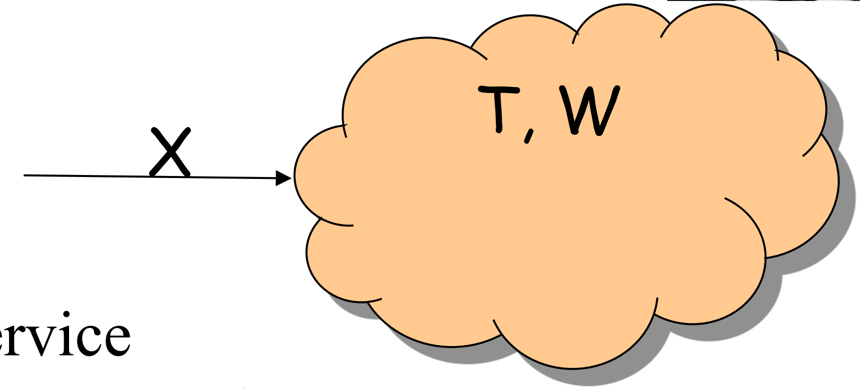


- How to estimate the number of packets queued in the bottleneck queue?



# Background: Little's Law (1961)

- For any system with no or (low) loss.
- Assume
  - Mean arrival rate $X$, mean service time $T$, and mean number of requests in the system $W$



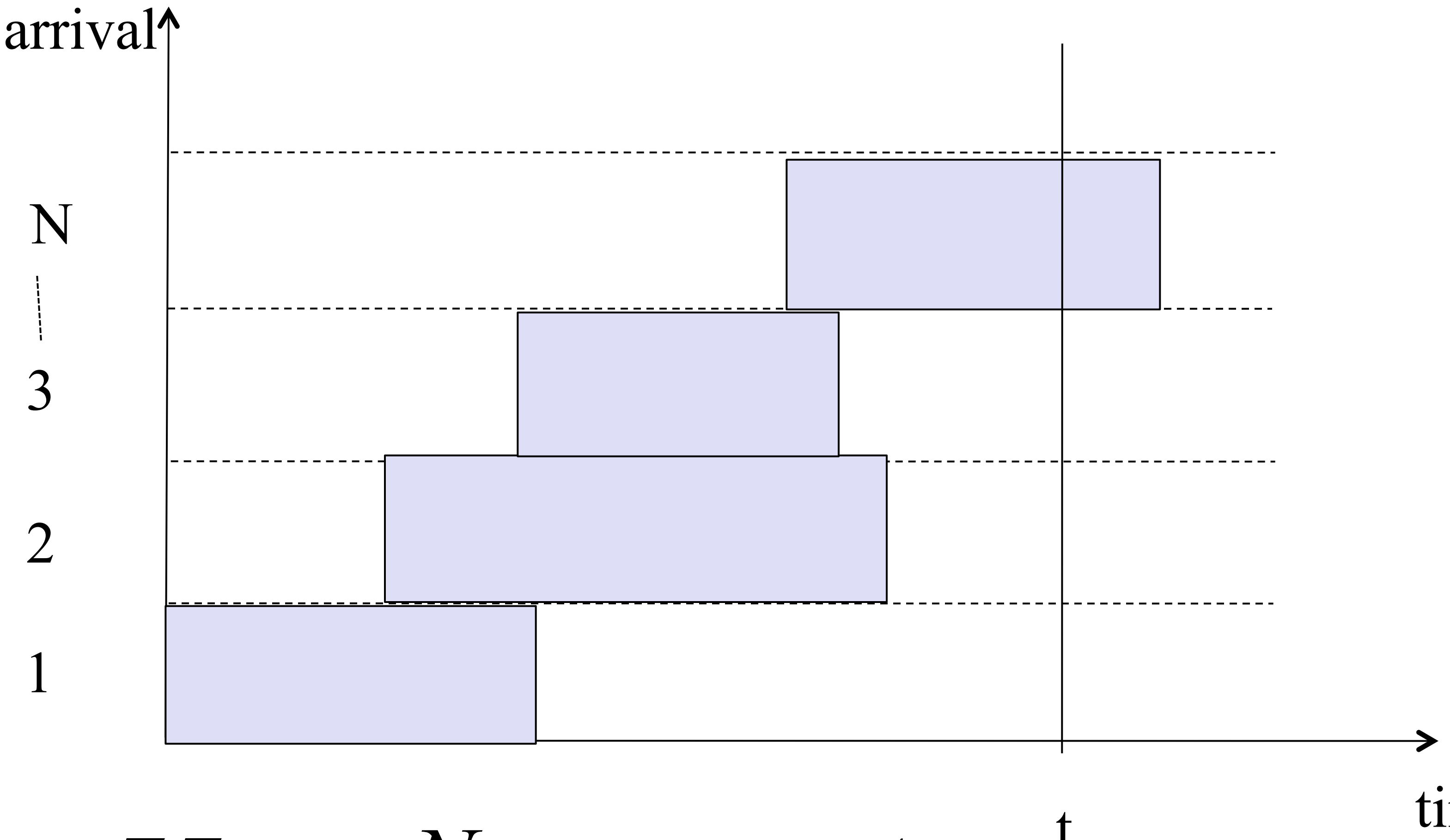

- Then relationship between $W$, $X$, and $T$:

$$W = XT$$

Example: SJTU admits 2500 students each year, and mean time a student stays is 4 years, how many students are enrolled?

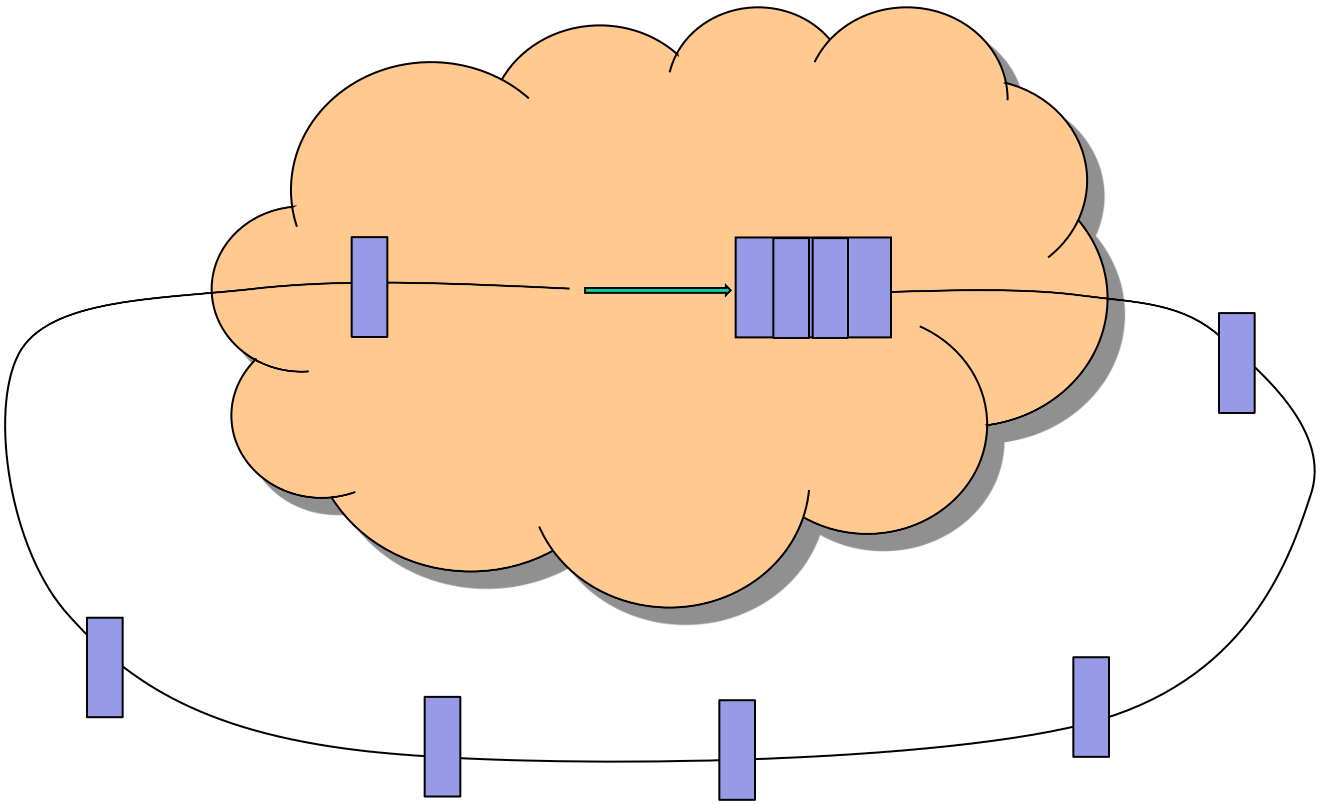
# Little's Law

$$W = XT$$

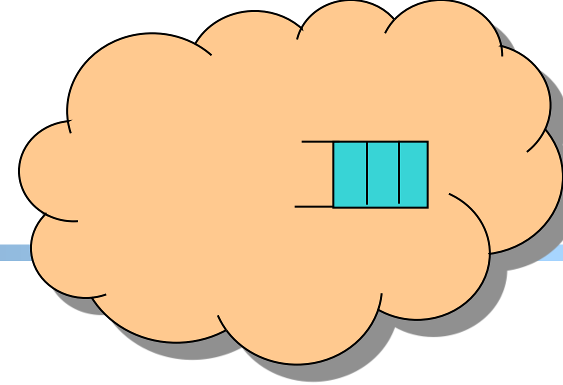



$$X = \frac{N}{t} \quad T = \frac{Area}{N} \quad W = \frac{Area}{t}$$

# Estimating Number of Packets in the Queue



# TCP/Vegas CA algorithm



$$T = T_{prop} + T_{queueing}$$

- Applying Little's Law:

  $$x_{vegas}\, T = x_{vegas}\, T_{prop} + x_{vegas}\, T_{queueing},$$

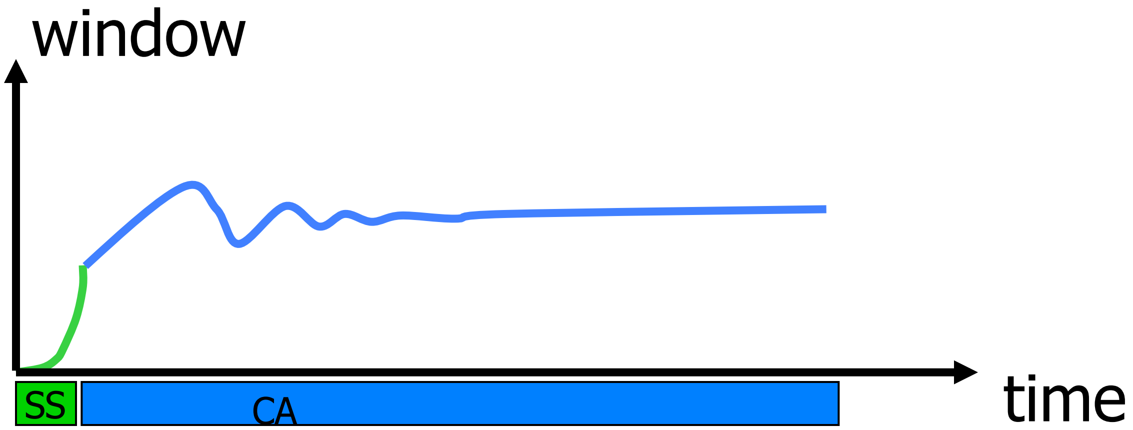
  where $x_{vegas} = W / T$ is the sending rate

- Then number of packets in the queue is

  $$x_{vegas}\, T_{queueing} = x_{vegas}\, T - x_{vegas}\, T_{prop}$$
  $$= W - W/T\, T_{prop}$$

# TCP/Vegas CA algorithm

maintain a *constant* number of packets in the bottleneck buffer




```
for every RTT
{    if W – W/RTT RTTmin < α  then W ++
     if W – W/RTT RTTmin > α  then W --
}
for every loss
         W := W/2
```

queue size

# TCP/Vegas Dynamics

$$\Delta \mathrm{w}_{\mathrm{RTT}} \approx -(w - xRTT_{\min} - \alpha)$$

$$\Delta \mathrm{w}_{\text{unit-time}} = -\left(\frac{w}{RTT} - \frac{x}{RTT} RTT_{\min} - \frac{\alpha}{RTT}\right) = \frac{x}{RTT} RTT_{\min} + \frac{\alpha}{RTT} - x$$

$$\Delta x = \frac{\Delta \mathrm{w}_{\text{unit-time}}}{RTT} = \frac{x}{RTT^2}\left(RTT_{\min} + \frac{\alpha}{x} - RTT\right)$$

# TCP/Reno vs. TCP/Vegas

| | TCP/Reno | TCP/Vegas |
|---|---|---|
| Congestion signal | loss rate p | queueying delay $T_{queueing}$ |
| Dynamics | $\Delta x = \frac{1}{RTT^2} - p\frac{1}{2}x^2$ | $\Delta x = \frac{x}{RTT^2}(RTT_{\min} + \frac{\alpha}{x} - RTT)$ |
| Equilibrium | $x_{reno} = \frac{\alpha_{reno}}{RTT\sqrt{p}}$ | $x_{vegas} = \frac{\alpha_{vegas}}{T_{queueing}}$ |

Discussion: Why and why not TCP/Vegas?