

---

# TCP Reno/Vegas

10/21/2009

# Admin.

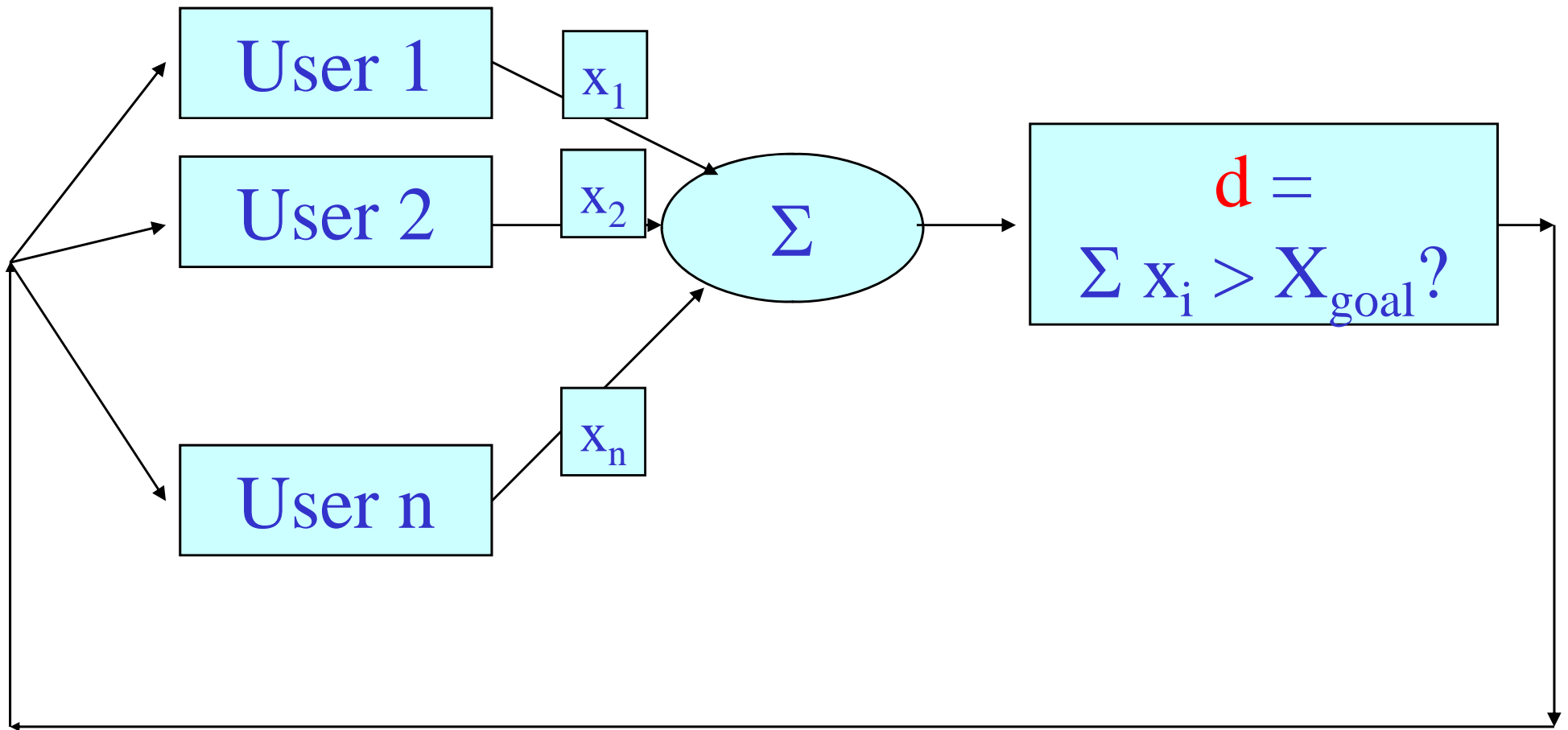
---

- Questions on programming assignment 2?

# Recap: The Desired Properties of a Congestion Control Scheme

- ❑ Efficiency: close to full utilization but low delay
  - fast convergence after disturbance
- ❑ Fairness (resource sharing)
- ❑ Distributedness (no central knowledge for scalability)

# Recap: A Simple Model



Flows observe congestion signal  $d$ , and locally take actions to adjust rates.

# Four Special Cases

	<u>A</u> dditive <u>D</u> ecrease	<u>M</u> ultiplicative <u>D</u> ecrease
<u>A</u> dditive <u>I</u> ncrease	AIAD ( $b_I=b_D=1$ )	AIMD ( $b_I=1, a_D=0$ )
<u>M</u> ultiplicative <u>I</u> ncrease	MIAD ( $a_I=0, b_I>1, b_D=1$ )	MIMD ( $a_I=a_D=0$ )

$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\ a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.} \end{cases}$$

# Another Look

---

- Consider the difference or ratio of the rates of two flows
  - AIAD
  
  - MIMD
  
  - MIAD
  
  - AIMD

## Mapping A(M)I-MD to Protocol

- What do we need to apply the A(M)I-MD algorithm to a sliding window protocol?

$$x_i(t+1) = \begin{cases} a_I + x_i(t) & \text{if } d(t) = \text{no cong.} \\ b_D x_i(t) & \text{if } d(t) = \text{cong.} \end{cases}$$

# Outline

---

- Recap

- *TCP /Reno*

# 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](http://lxr.linux.no/source/net/ipv4/tcp_input.c) for linux implementation

# TCP/Reno Congestion Detection

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

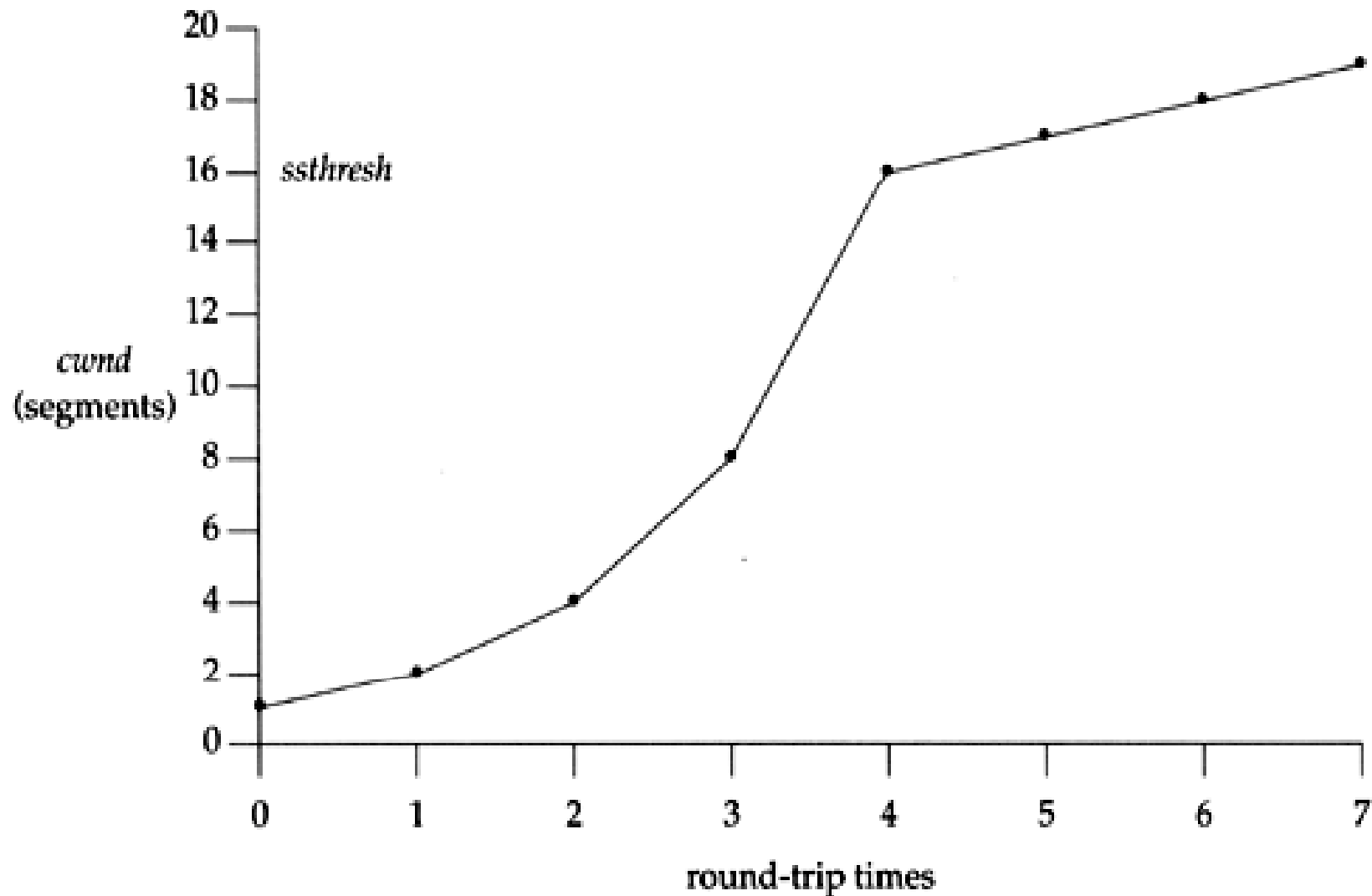
## Philosophy:

- 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;`

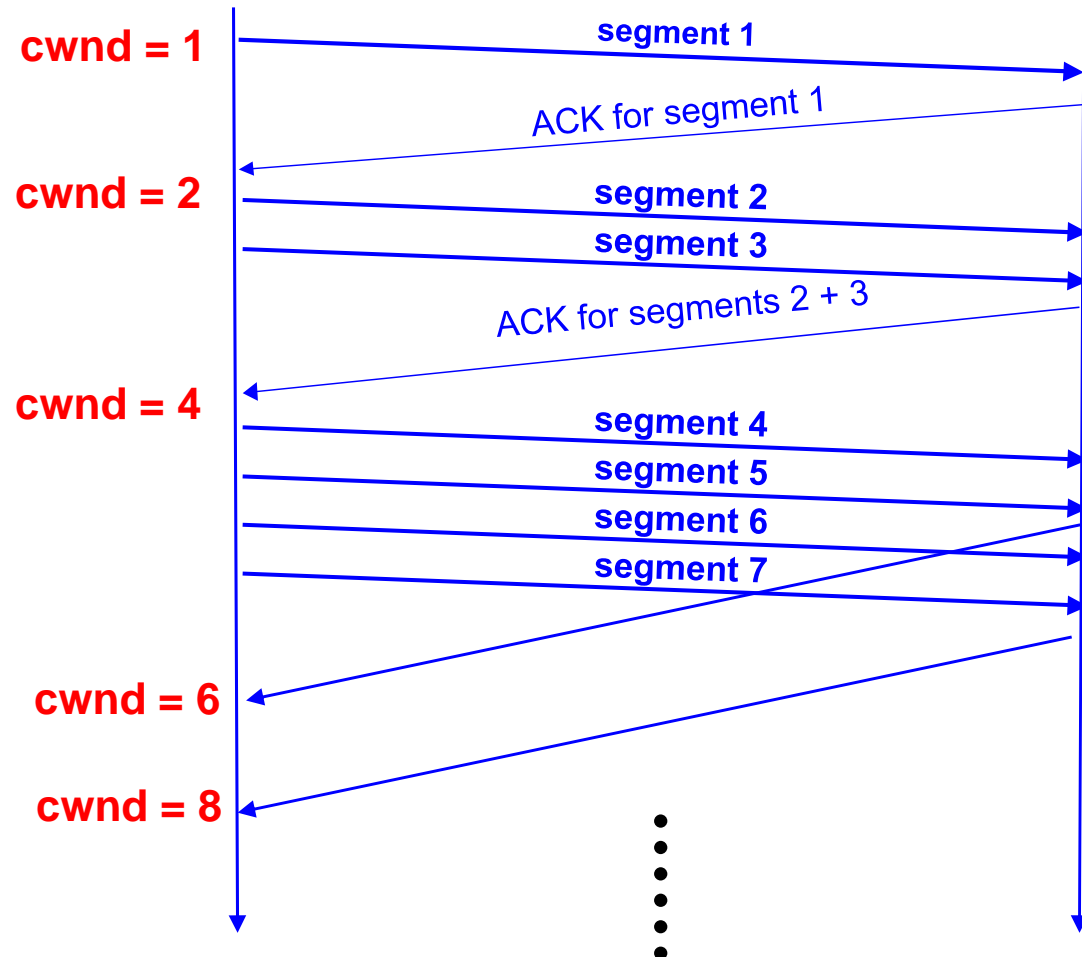
`ssthresh = infinite (e.g., 64K);`

**For each newly ACKed segment:**

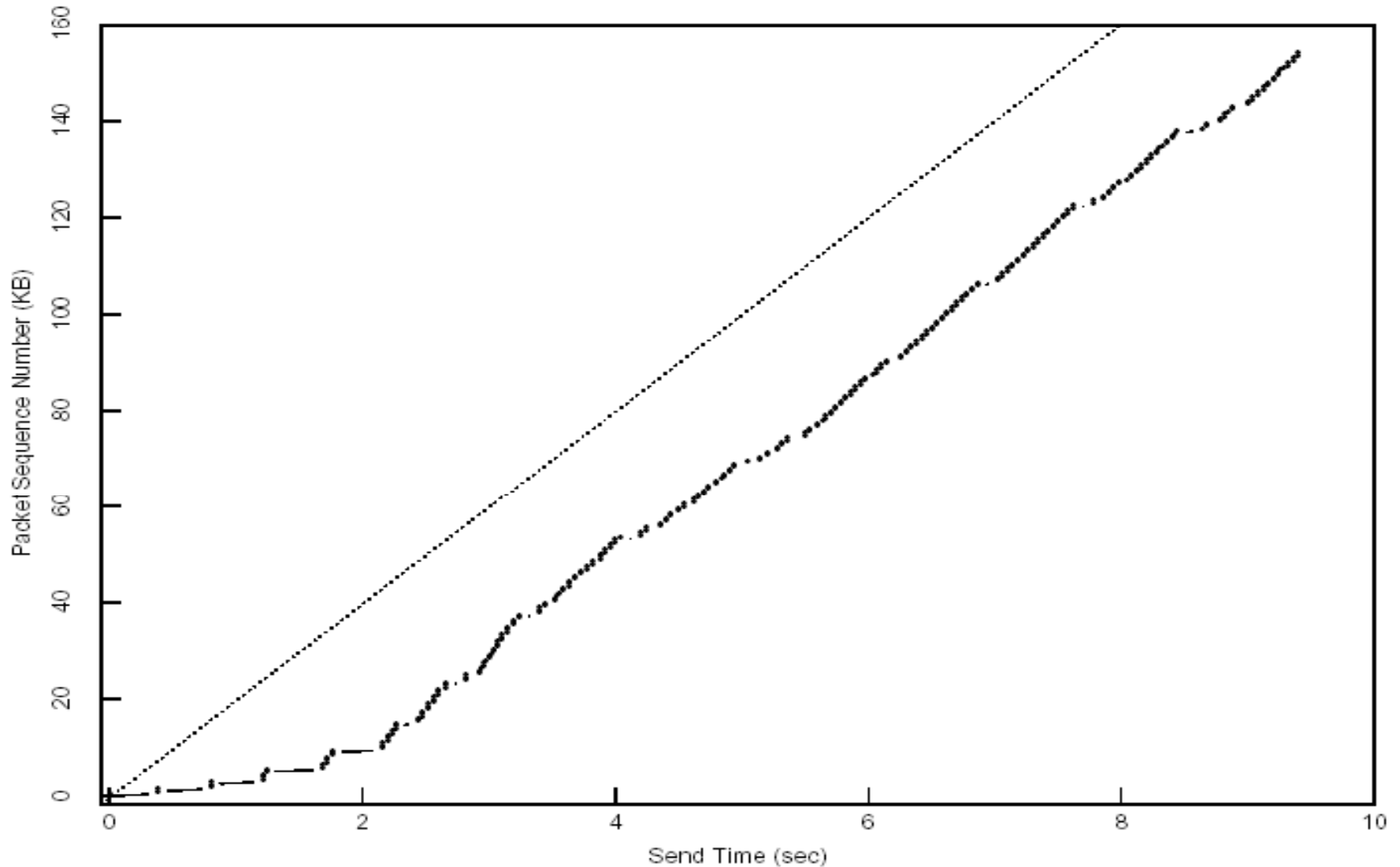
`if (cwnd < ssthresh)`

`/* slow start*/`

`cwnd = cwnd + 1;`



# Startup Behavior **with** Slow-start



# 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 3DUP
    - 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;`

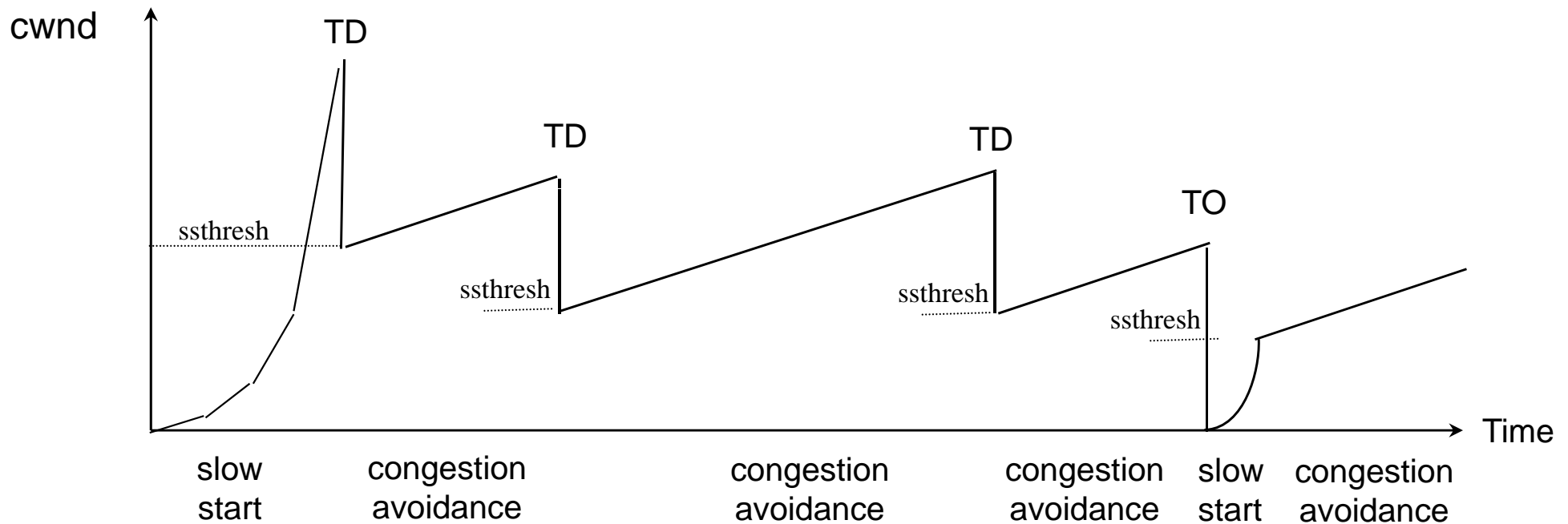
**Timeout:**

`ssthresh = cwnd/2;`

`cwnd = 1;`

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

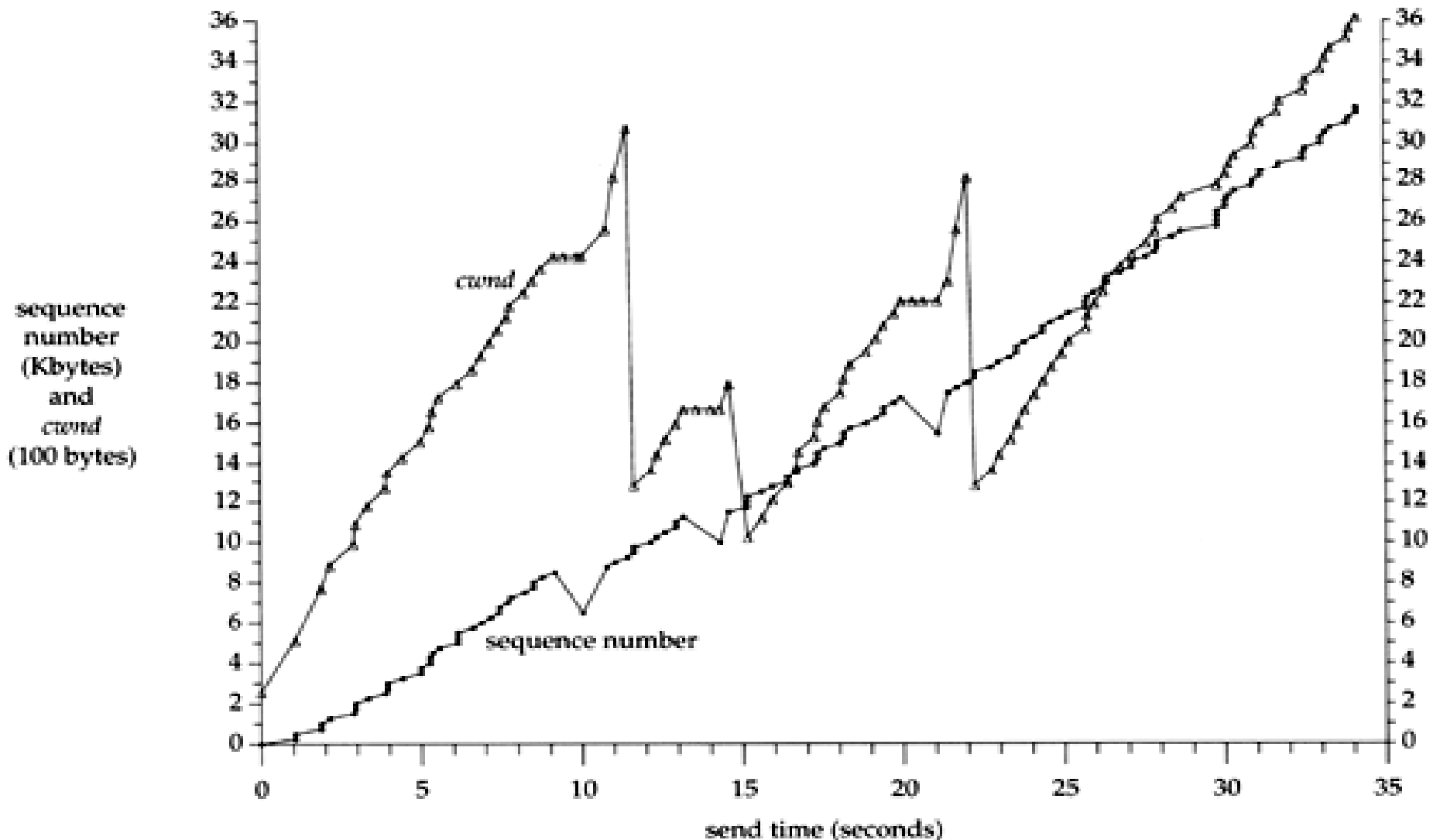
# TCP/Reno: Big Picture



TD: Triple duplicate acknowledgements

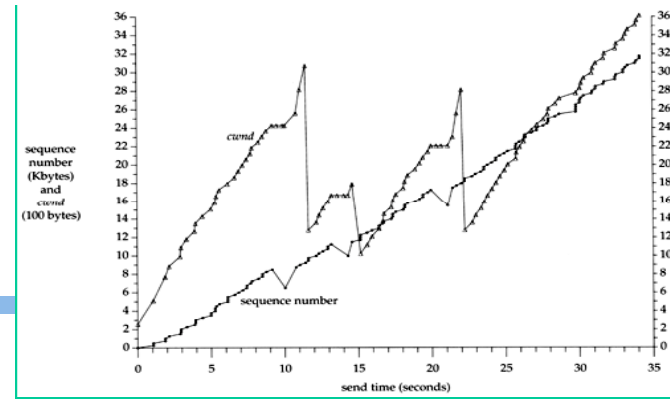
TO: Timeout

# A Session

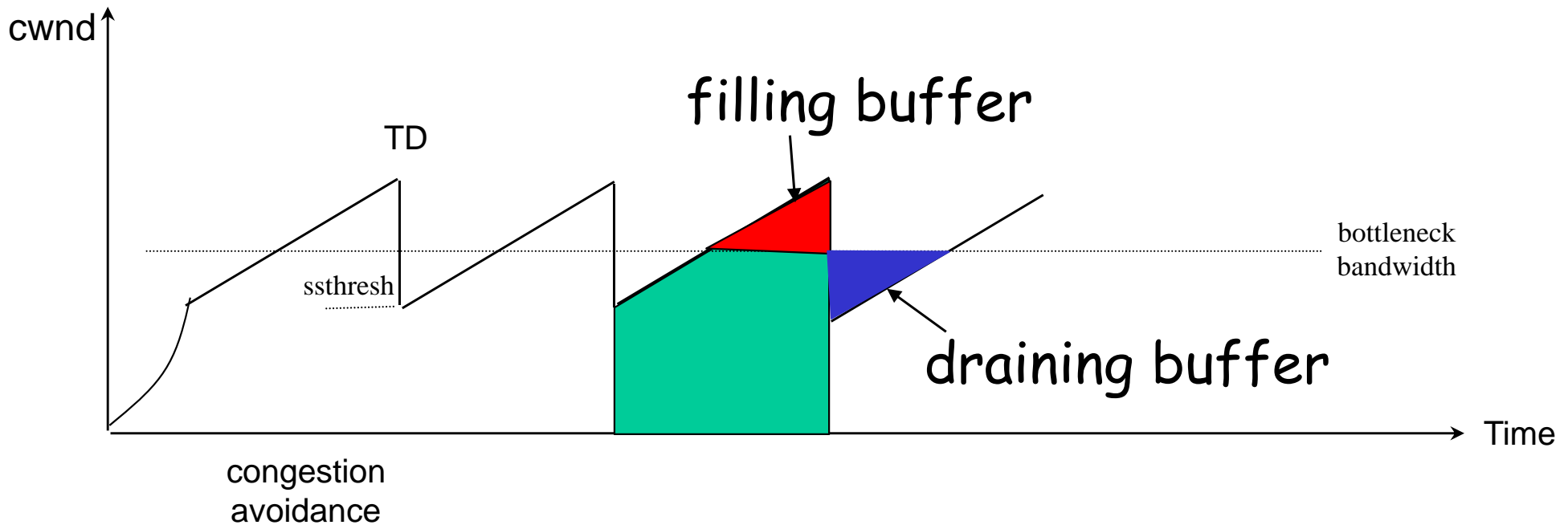


Question: when cwnd is cut to half, why sending rate is not?

# TCP/Reno Queueing Dynamics



□ Consider congestion avoidance only



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

# Outline

---

- Recap
- 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  $RTT$ :

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

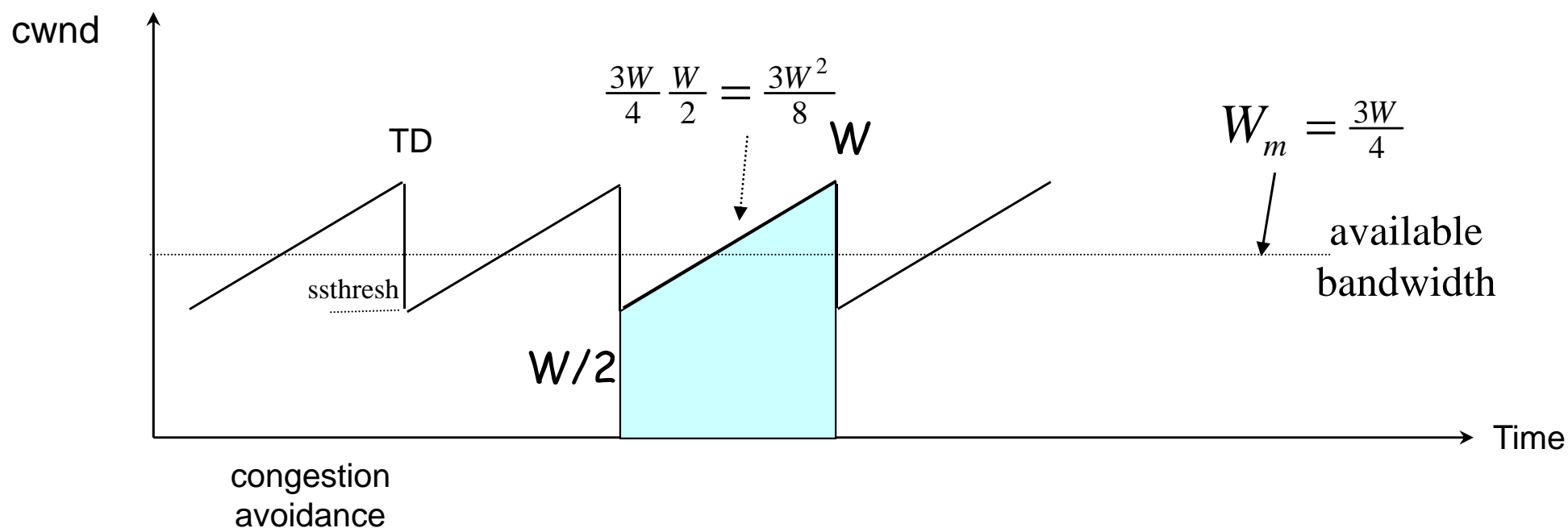
# Outline

---

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

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

# Outline

---

- Recap
- TCP/Reno
- TCP/Reno throughput analysis
  - analysis 1: deterministic
  - analysis 2: random loss

# TCP/Reno Throughput Modeling

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

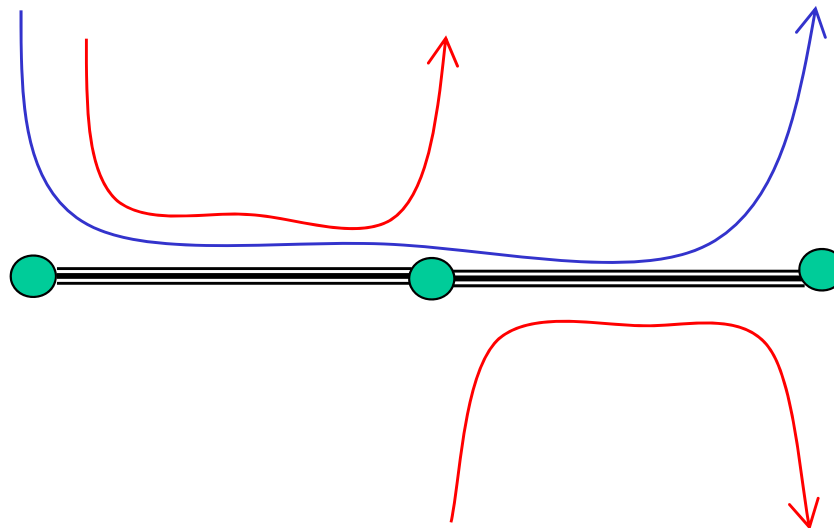
$$\text{mean of } \Delta W = (1-p)\frac{1}{W} + p(-\frac{W}{2}) = 0$$

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

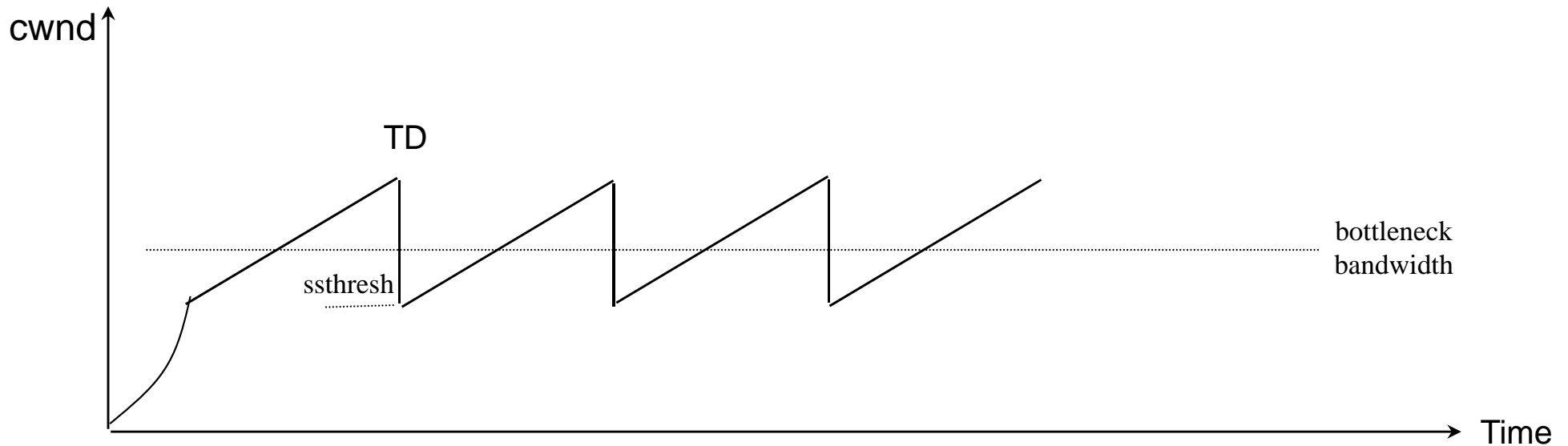
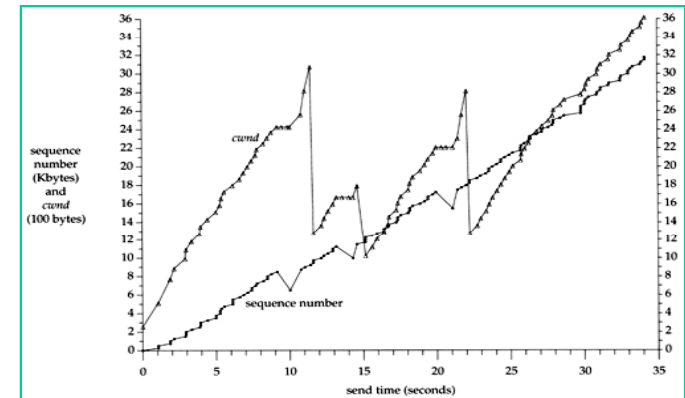
$$\Rightarrow \text{throughput} \approx \frac{1.4S}{RTT\sqrt{p}}, \text{ when } p \text{ is small}$$

# Summary: TCP/Reno Throughput Modeling

- They are all *approximate* modeling
  - the details and the exact numbers are not important
  - the objective is to help us understand TCP better



# Problem of TCP/Reno Behavior

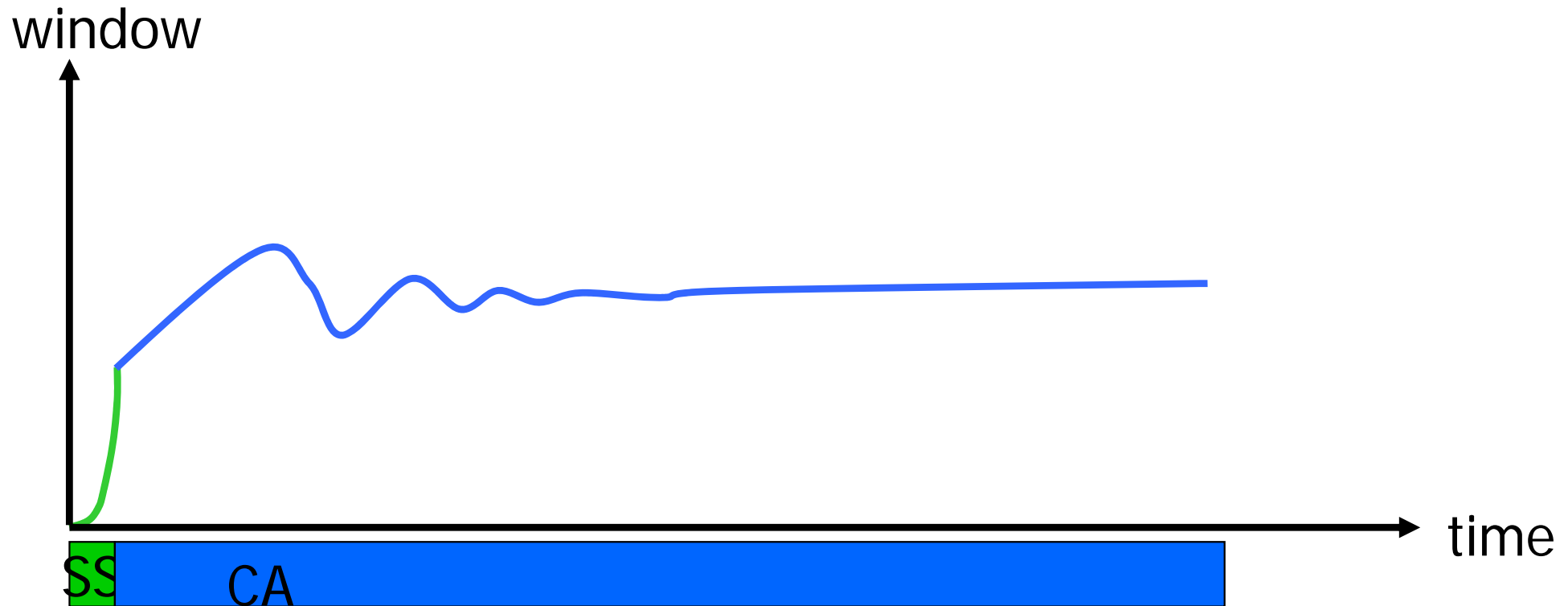


# Outline

---

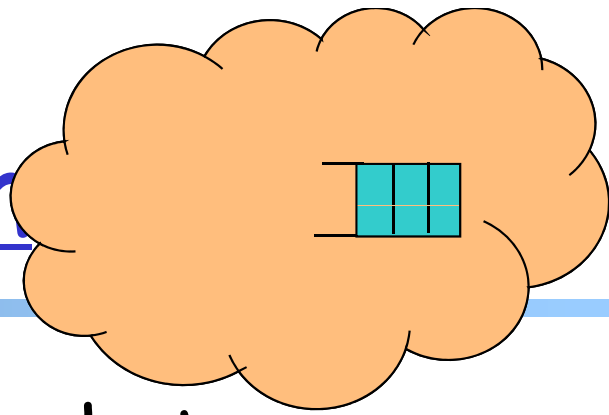
- Recap
- TCP/Reno
- TCP/Reno throughput analysis
- *TCP/Vegas*

# TCP/Vegas (Brakmo & Peterson 1994)

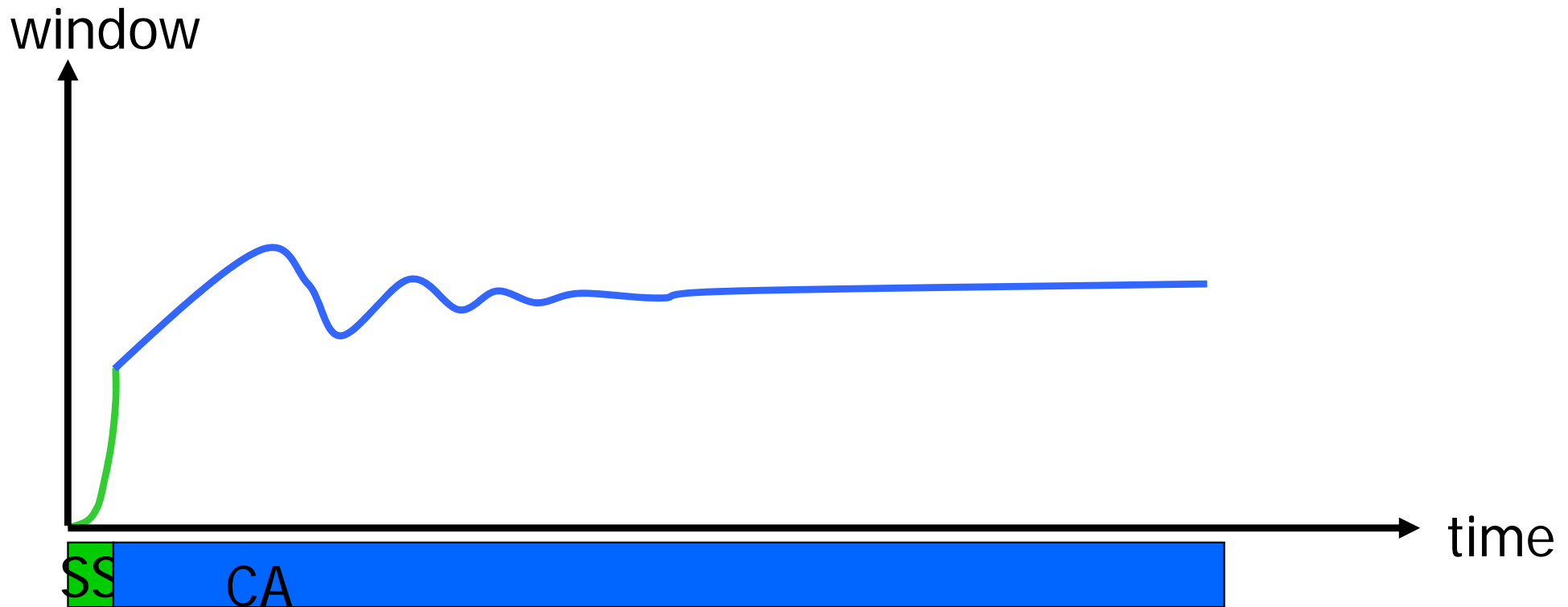


- Idea: try to detect congestion by delay before loss
- Objective: not to overflow the buffer; instead, try to maintain a *constant* number of packets in the bottleneck queue

# 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.

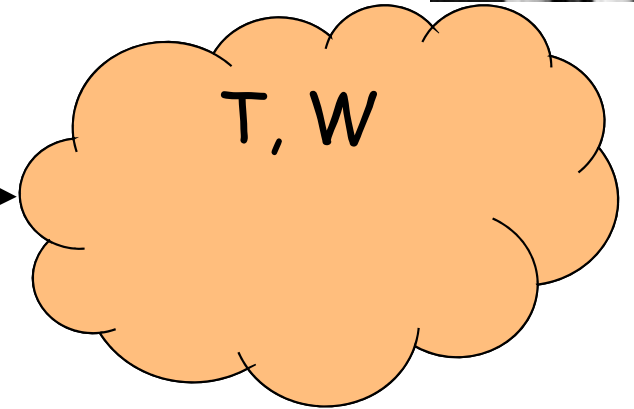
□ Assume

- mean arrival rate  $X$ , mean service time  $T$ , and mean number of requests in the system  $W$

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

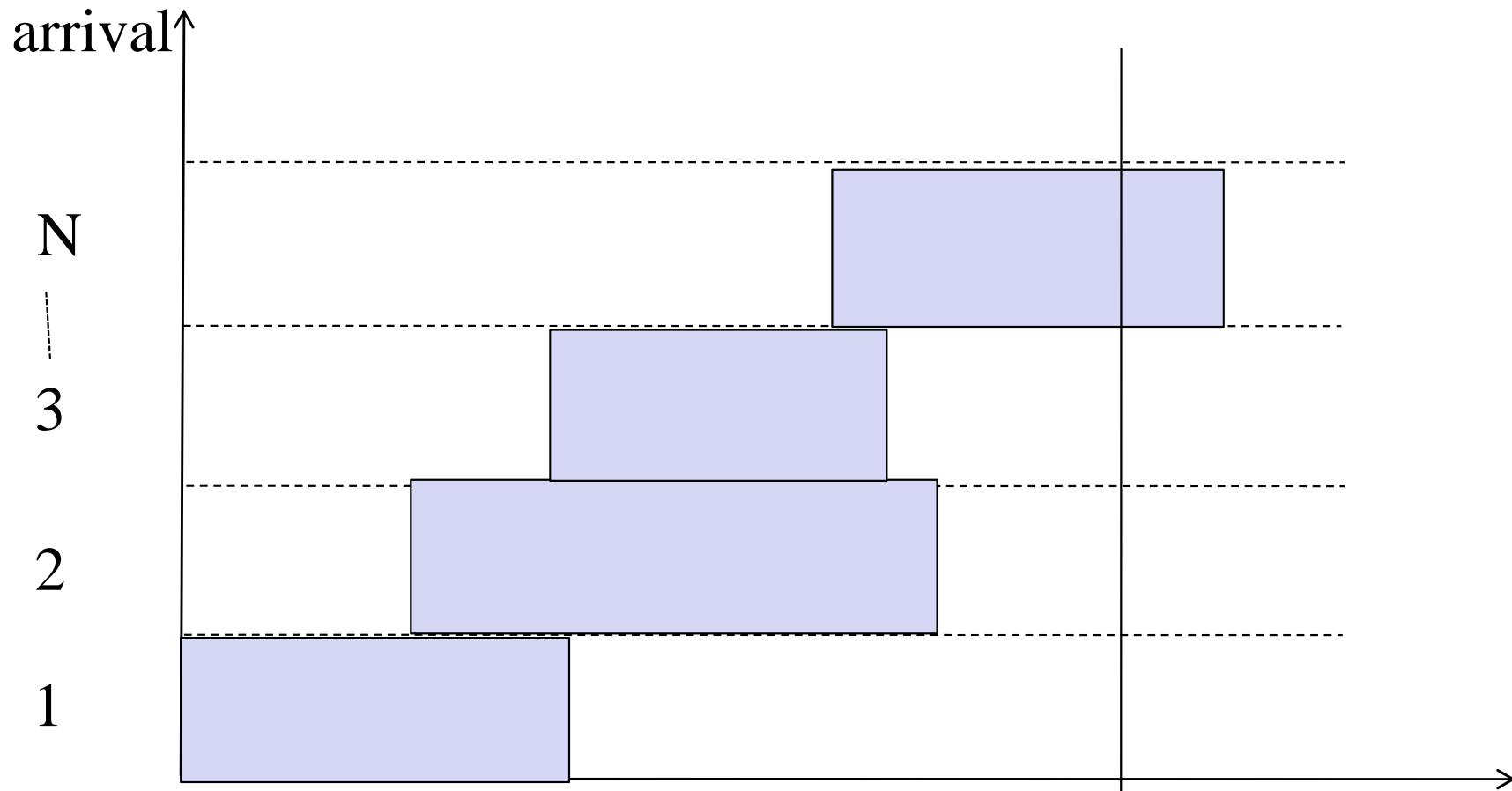
$$W = XT$$

Example: Yale College admits 1500 students each year, and mean time a student stays is 4 years, how many students are enrolled?



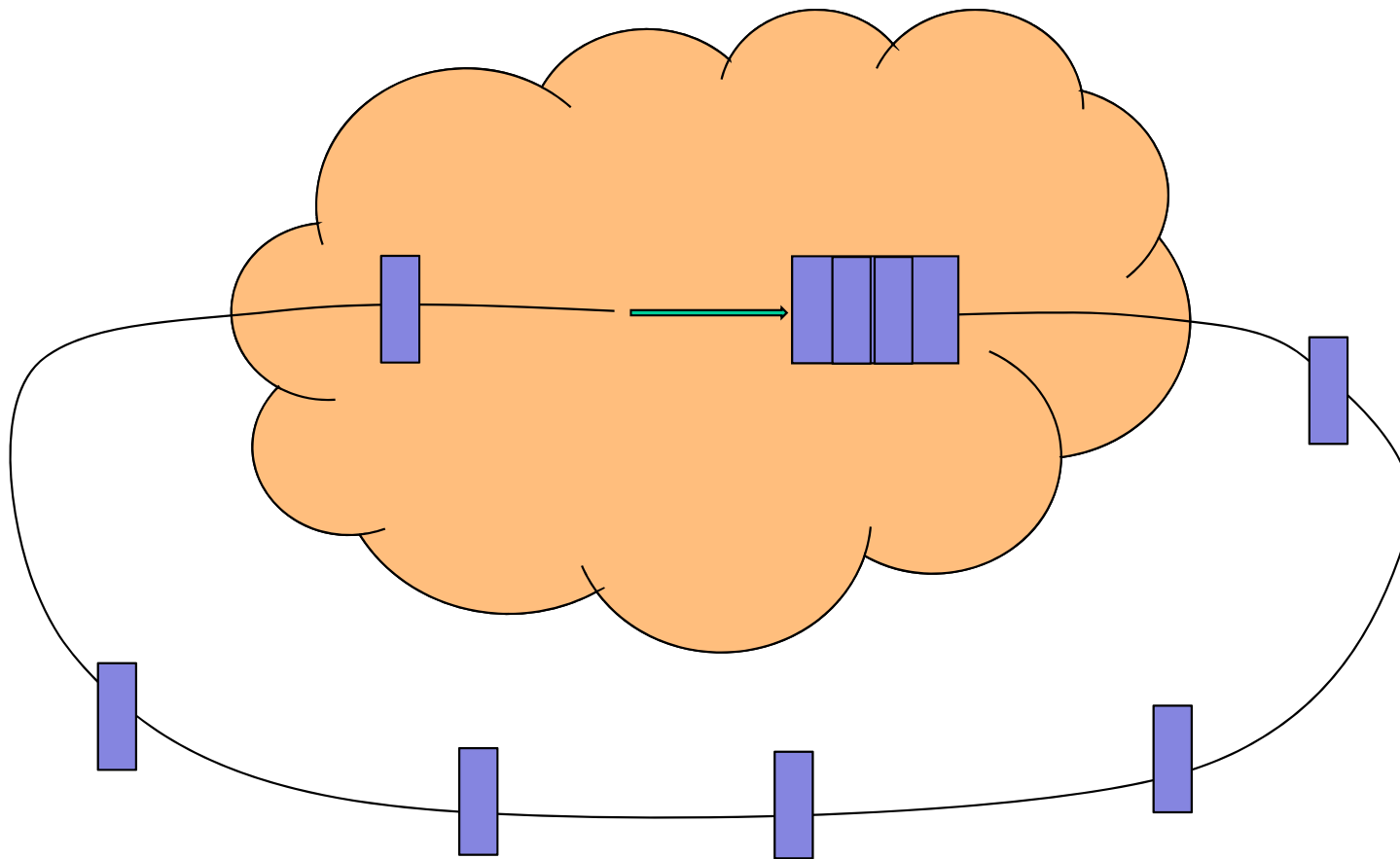
# Little's Law

$$W = XT$$

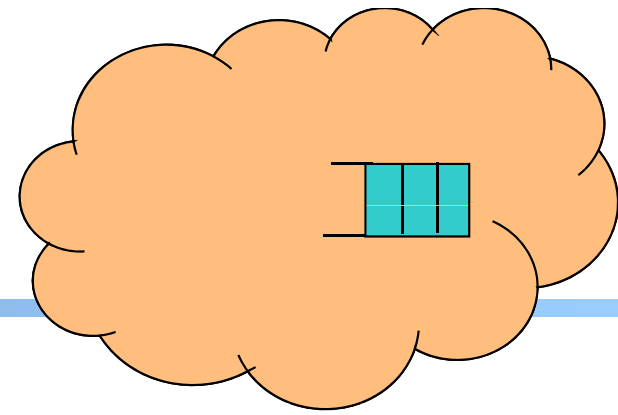


$$X = \frac{N}{t} \quad T = \frac{\text{Area}}{N} \quad W = \frac{\text{time Area}}{t}$$

# Estimating Number of Packets in the Queue



# TCP/Vegas CA algorithm



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

Applying Little's Law:

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

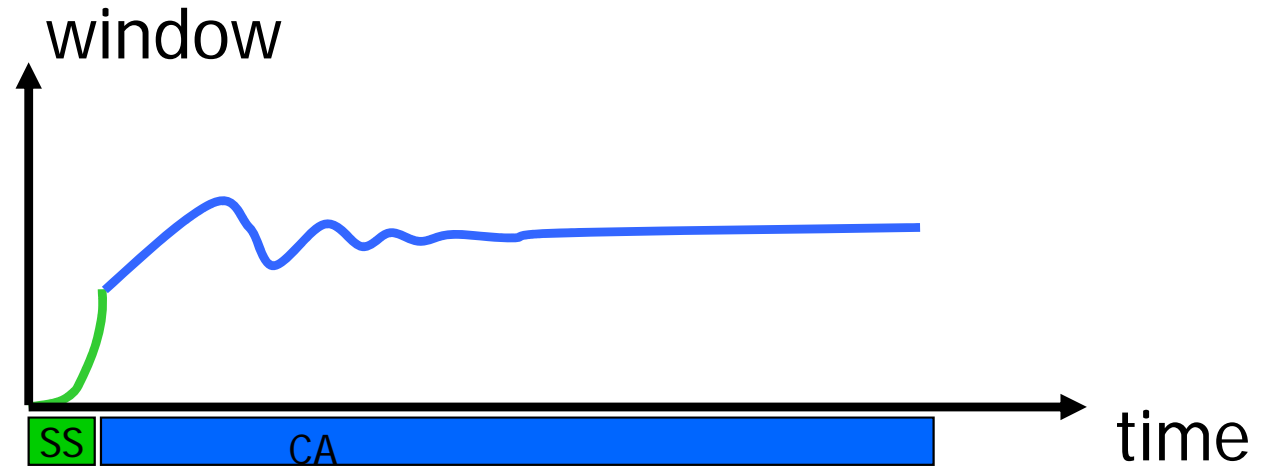
where  $x_{\text{vegas}} = W / T$  is the sending rate

Then number of packets in the queue is

$$\begin{aligned} x_{\text{vegas}} T_{\text{queueing}} &= x_{\text{vegas}} T - x_{\text{vegas}} T_{\text{prop}} \\ &= W - W/T T_{\text{prop}} \end{aligned}$$

# TCP/Vegas CA algorithm

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



```
for every RTT
{
  if  $W - W/RTT_{min} < \alpha$  then  $W++$ 
  if  $W - W/RTT_{min} > \alpha$  then  $W--$ 
}
for every loss
   $W := W/2$ 
```

queue size