

---

School of Computing Science  
Simon Fraser University

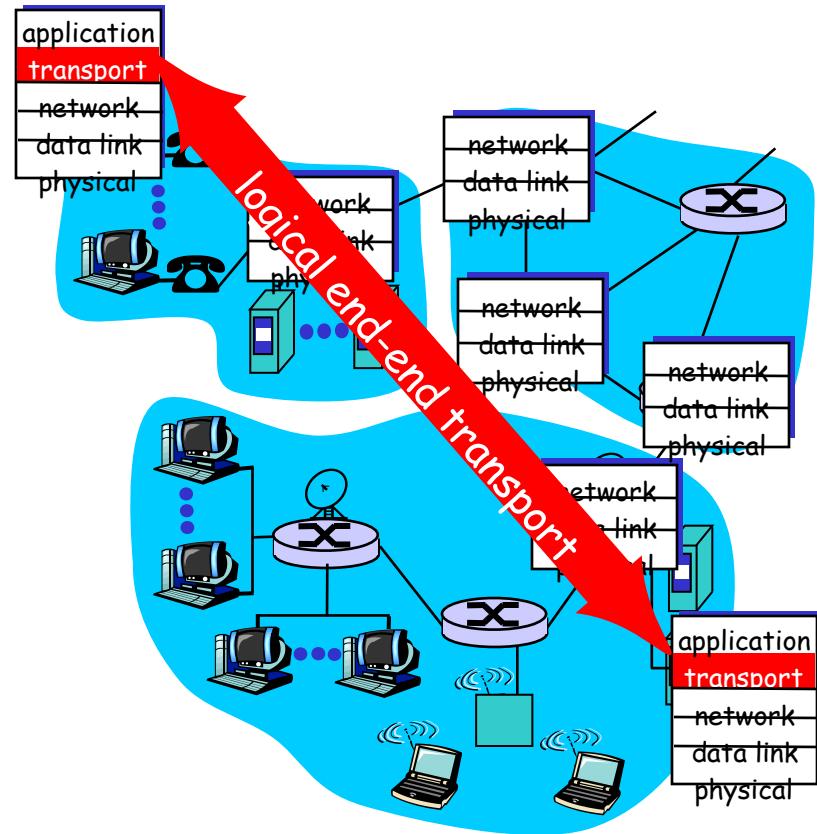
## CMPT 471: Networking II

Transport Layer & Resource Management

Instructor: Mohamed Hefeeda

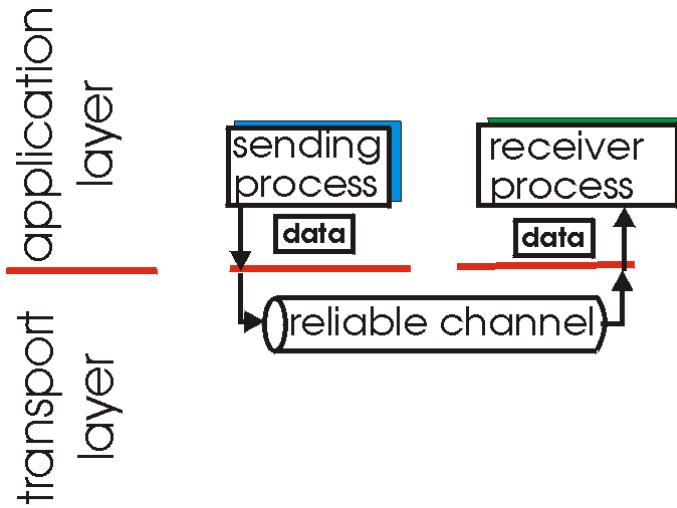
# Transport services and protocols

- ❑ provide **logical communication** between app processes running on different hosts
- ❑ transport protocols run in end systems
  - ❖ send side: breaks app messages into **segments**, passes to network layer
  - ❖ rcv side: reassembles segments into messages, passes to app layer
- ❑ more than one transport protocol available to apps
  - ❖ Internet: TCP and UDP



# Reliable data transfer

- ❑ important in application, transport, and link layers
- ❑ top-10 list of important networking topics!

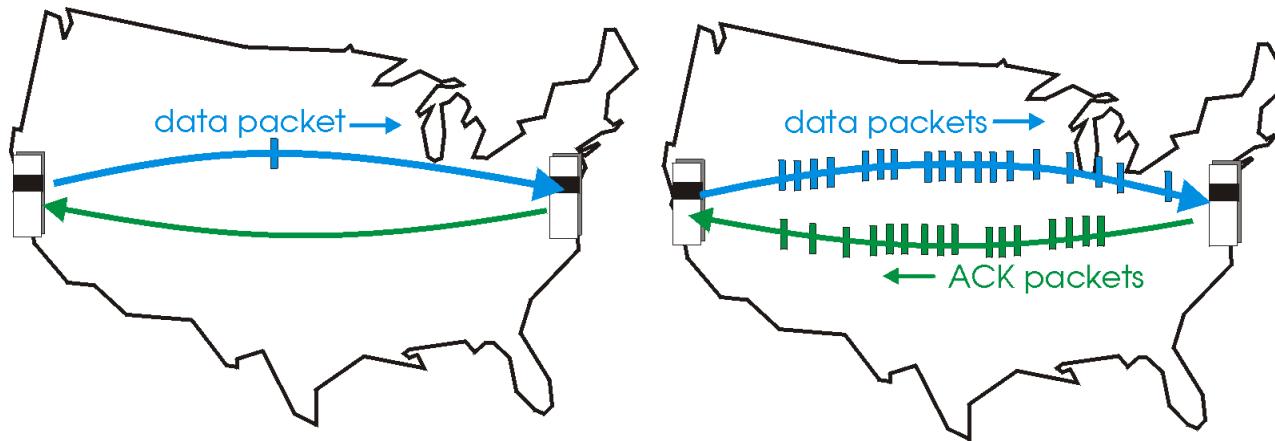


(a) provided service

# Pipelined (Sliding Window) Protocols

**Pipelining:** sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

- ❖ range of sequence numbers must be increased
- ❖ buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation

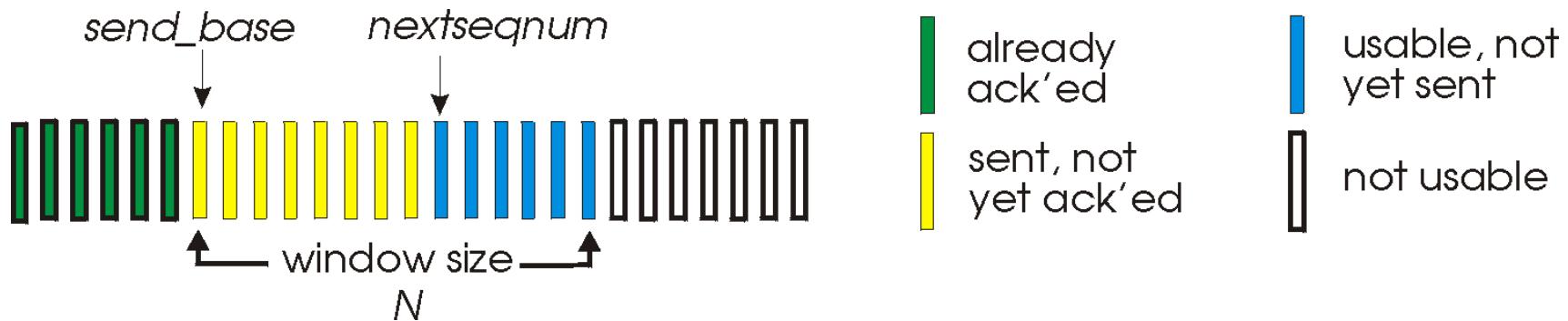
(b) a pipelined protocol in operation

- ❑ Two generic forms of pipelined protocols: **go-Back-N, selective repeat**

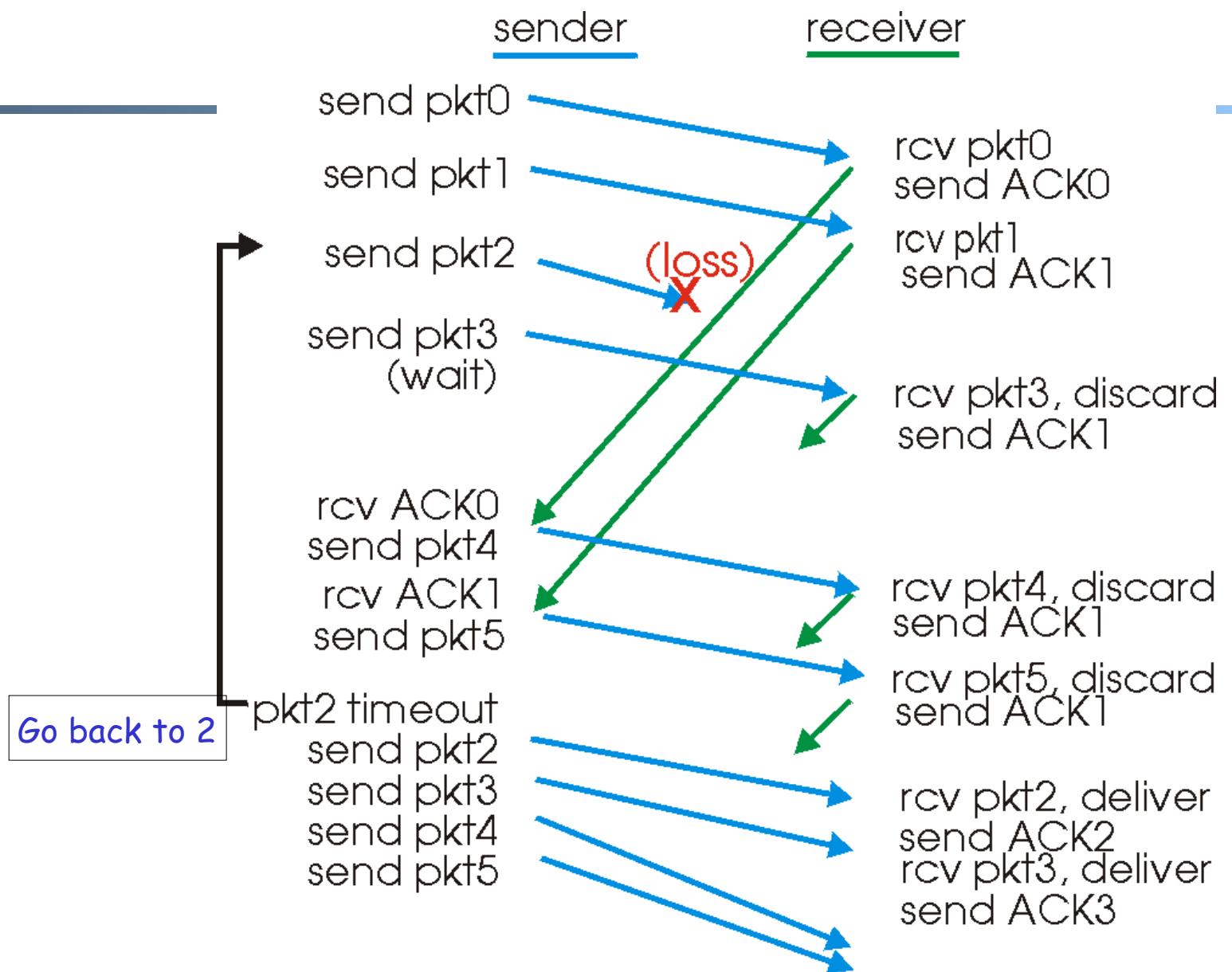
# Go-Back-N

## Sender:

- ❑ k-bit seq # in pkt header
- ❑ “window” of up to N consecutive unack’ed pkts allowed

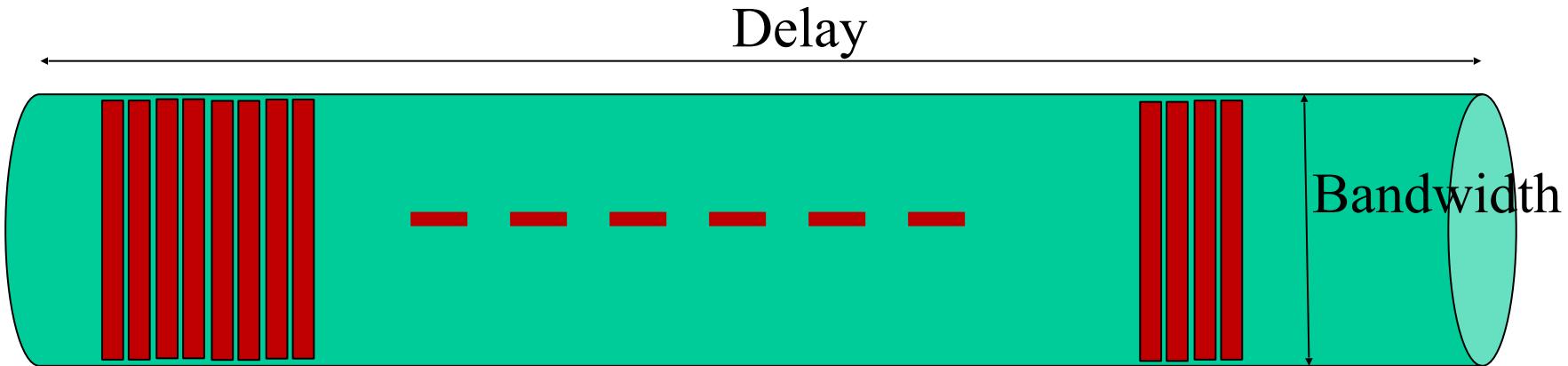


- ❑ ACK(n): ACKs all pkts up to and including seq # n -- **cumulative ACK**
  - ❖ may receive duplicate ACKs
- ❑ timer for each in-flight pkt
- ❑ timeout(n): retransmit pkt n and all higher seq # pkts in window
  - ❖ i.e., go back to n



Window size, N = 4

# Go-Back-N: Problems?

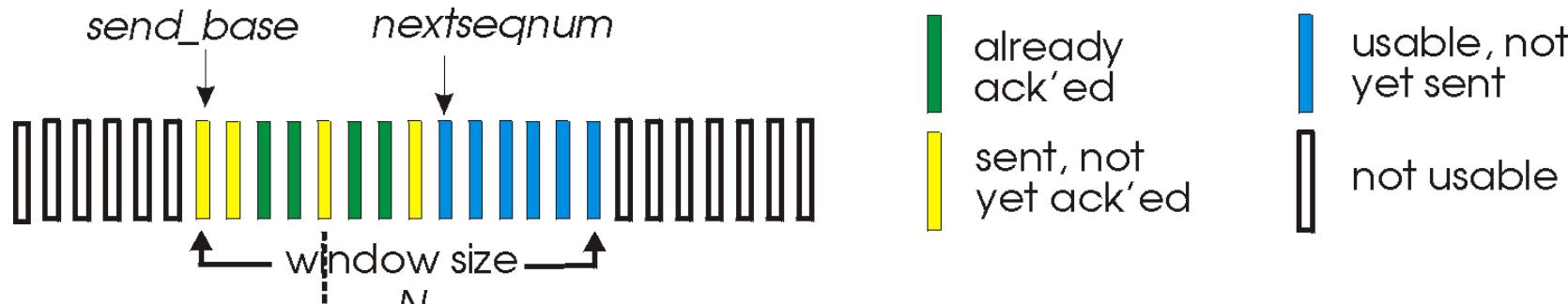


- ❑ consider high-speed links with long delays
  - ❖ (called large bandwidth-delay product pipes)
- ❑ GBN can fill that pipe by having large N →
- ❑ many unACKed pkts could be in the pipe
- ❑ A single lost pkt could cause re-transmission of huge number (up to N) of pkts → waste of bandwidth
  
- ❑ Solutions??

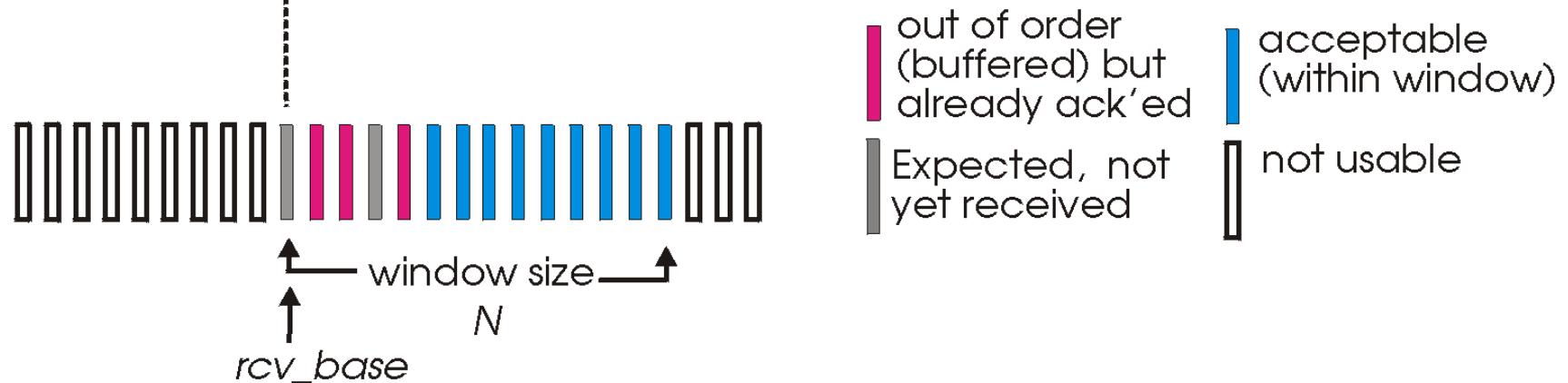
# Selective Repeat

- ❑ receiver **individually acknowledges correctly received pkts**
  - ❖ buffers pkts, as needed, for eventual in-order delivery to upper layer
- ❑ sender only resends pkts for which ACK not received
  - ❖ sender timer for each unACKed pkt
- ❑ sender window
  - ❖ N consecutive seq #'s
  - ❖ again limits seq #s of sent, unACKed pkts

# Selective repeat: sender, receiver windows



(a) sender view of sequence numbers

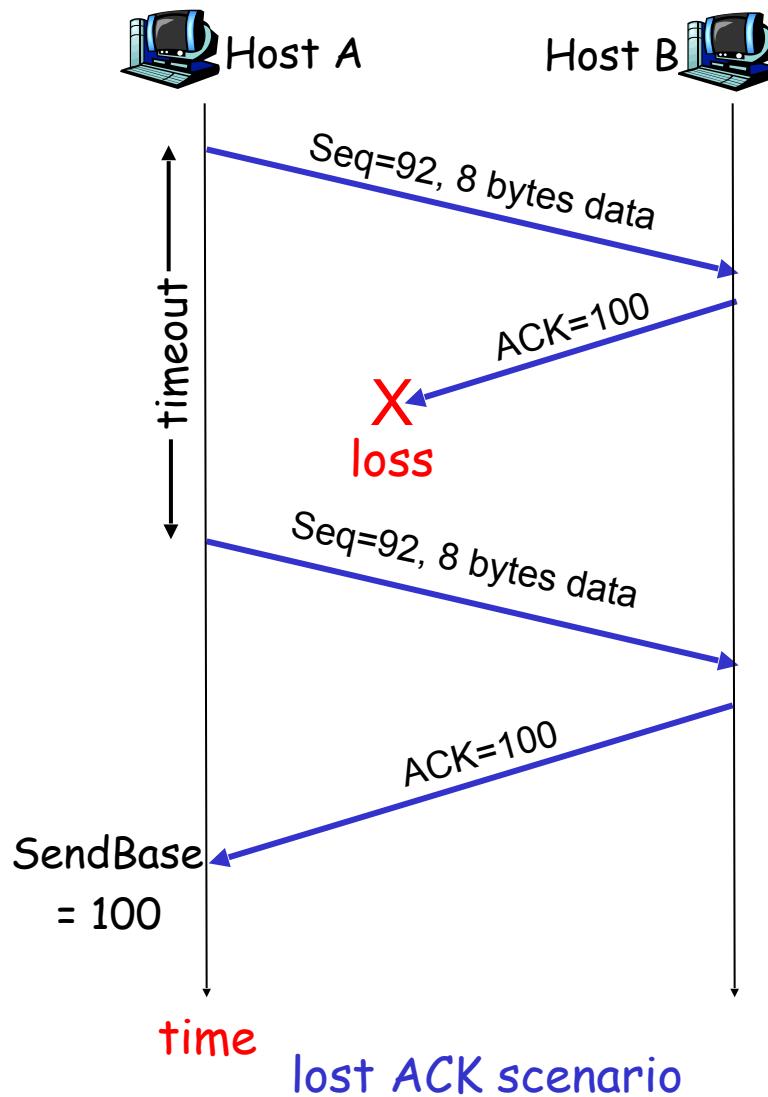


(b) receiver view of sequence numbers

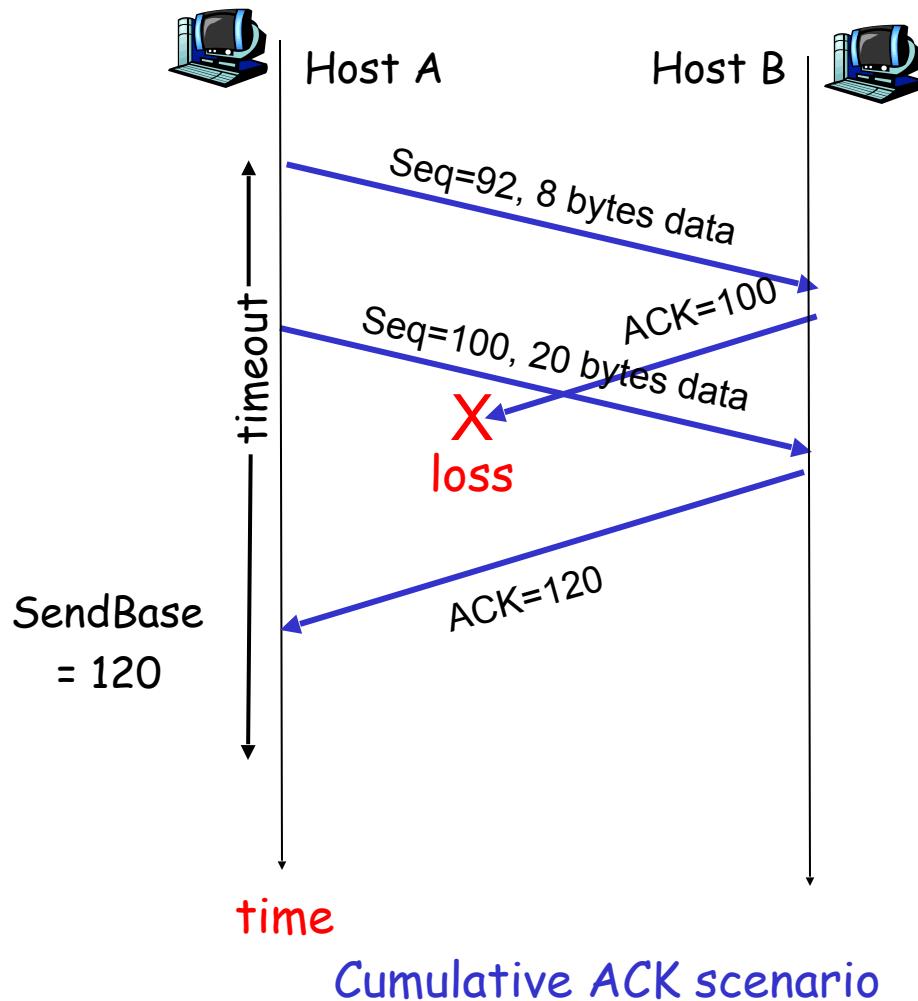
# TCP reliable data transfer

- ❑ TCP creates rdt service on top of IP's unreliable service
- ❑ Pipelined segments
- ❑ Cumulative acks
- ❑ TCP uses single retransmission timer
  - ❖ Why single timer?
- ❑ Retransmissions are triggered by:
  - ❖ timeout events
  - ❖ duplicate acks
- ❑ Initially consider simplified TCP sender:
  - ❖ ignore duplicate acks
  - ❖ ignore flow control, congestion control

# TCP: retransmission scenarios



# TCP retransmission scenarios (more)



# TCP Round Trip Time and Timeout

- ❑ If TCP timeout is
  - ❖ too short: premature timeout → unnecessary retransmissions
  - ❖ too long: slow reaction to segment loss

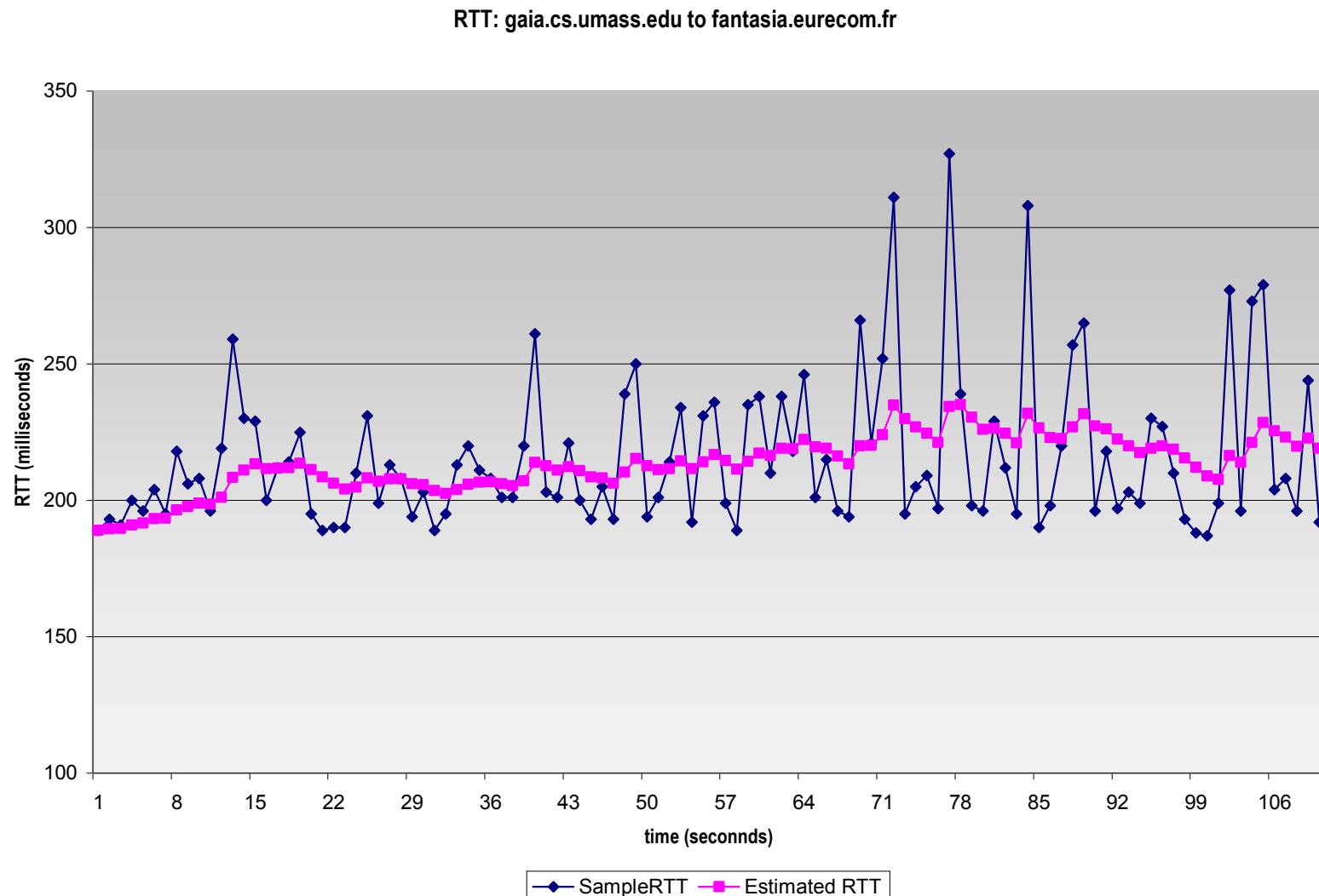
**Q: how to set TCP timeout value?**

- ❑ Based on Round Trip Time (RTT), but RTT varies with time!
- ❑ → We estimate current RTT

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- ❑ Exponential weighted moving average
- ❑ influence of past sample decreases exponentially fast
- ❑ typical value:  $\alpha = 0.125$  ( $\rightarrow$  efficient computation **why?**)

# Example RTT estimation:



# TCP Round Trip Time and Timeout

## Setting the timeout

- EstimatedRTT plus safety margin
  - ❖ large variation in EstimatedRTT → larger safety margin
- first estimate how much SampleRTT deviates from EstimatedRTT:

$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically,  $\beta = 0.25$ )

Then set timeout interval:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

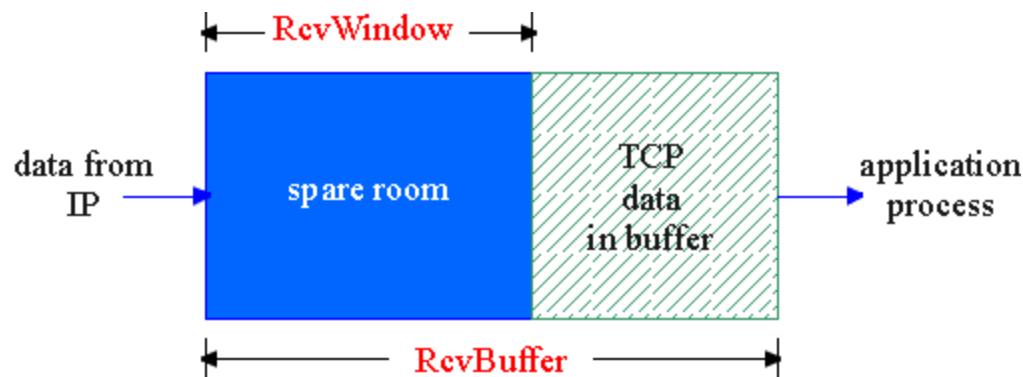
# Fast Retransmit

- Time-out period often relatively long:
  - ❖ long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - ❖ Sender often sends many segments back-to-back
  - ❖ If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - ❖ **fast retransmit:** resend segment before timer expires

# TCP Flow Control

- ❑ receive side of TCP connection has receive buffer:

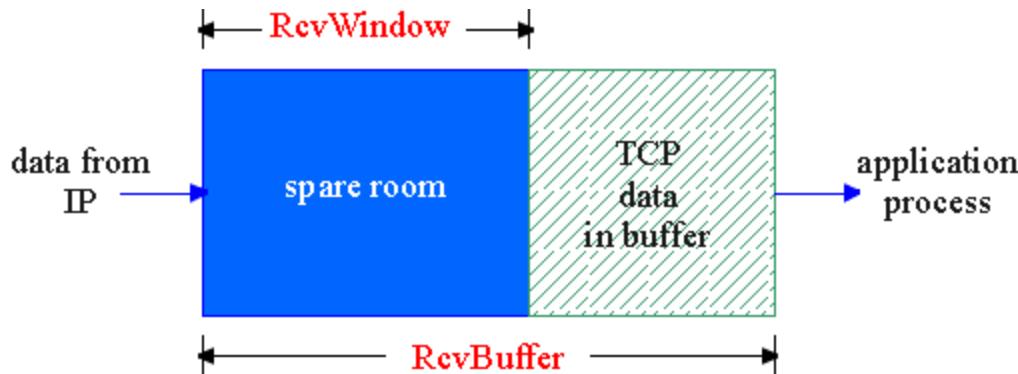
flow control  
sender won't overflow receiver's buffer by transmitting too much, too fast



- ❑ app process may be slow at reading from buffer

- ❑ speed-matching service: matching send rate to receiving app's drain rate
- ❑ Flow control is end to end

# TCP Flow control: how it works



(Suppose TCP receiver discards out-of-order segments)

- ❑ spare room in buffer
  - = RcvWindow
  - = RcvBuffer - [LastByteRcvd - LastByteRead]

- ❑ Rcvr advertises spare room by including value of RcvWindow in segments
- ❑ Sender limits unACKed data to RcvWindow
  - ❖ guarantees receive buffer doesn't overflow

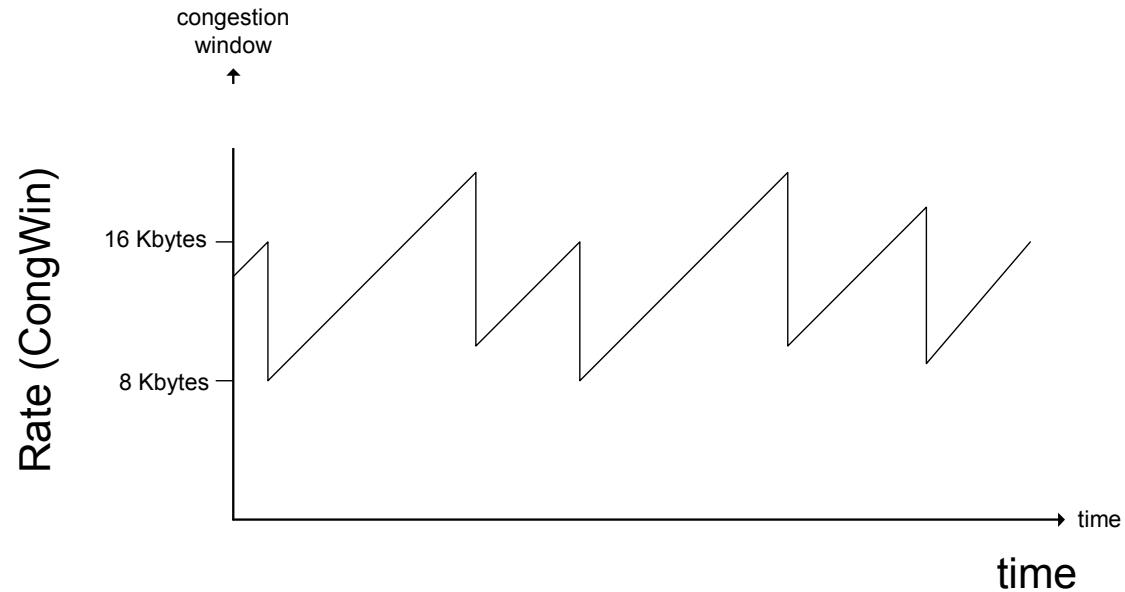
# Congestion Control

- ❑ Congestion: sources send too much data for network to handle
  - ❖ different from flow control, which is e2e
- ❑ Congestion results in ...
  - ❖ lost packets (buffer overflow at routers)
    - more work (retransmissions) for given “goodput”
    - waste of upstream links’ capacity
      - Pkt traversed several links, then dropped at congested router
  - ❖ long delays (queuing in router buffers)
    - poor performance (less responsive app)
    - premature (unneeded) retransmissions

# TCP congestion control: Approach

- **Approach:** probe for usable bandwidth in network
  - ❖ increase transmission rate until loss occurs then decrease
  - ❖ Additive increase, multiplicative decrease (AIMD)

Saw tooth  
behavior: probing  
for bandwidth



# TCP Congestion Control

- Sender keeps new variable, Congestion Window (**CongWin**), and limits unacked bytes to:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min \{\text{CongWin}, \text{RcvWin}\}$$

- ❖ For our discussion: assume RcvWin is large enough
- ❖ Above equation achieves both flow and congestion control

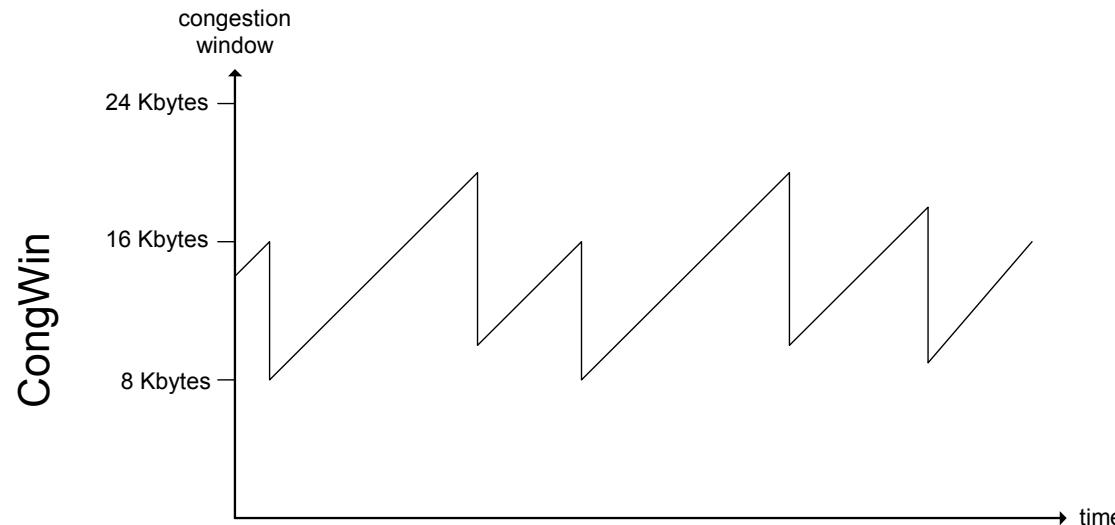
- Roughly, what is the sending rate as a function of CongWin?
  - ❖ Ignore loss and transmission delay
- Rate = CongWin/RTT (bytes/sec)
  - ❖ So, rate and CongWin are somewhat synonymous

# TCP Congestion Control

- ❑ Congestion occurs at routers (inside the network)
  - ❖ Routers do not provide any feedback to TCP
- ❑ How can TCP infer congestion?
  - ❖ From its symptoms: timeout or duplicate acks
  - ❖ Define loss event = timeout or 3 duplicate acks
  - ❖ TCP decreases its CongWin (rate) after a loss event
- ❑ TCP Congestion Control Algorithm: three components
  - ❖ AIMD: additive increase, multiplicative decrease
  - ❖ slow start
  - ❖ Reaction to timeout events

# AIMD

- additive increase: (congestion avoidance phase)
  - ❖ increase CongWin by 1 MSS every RTT until loss detected
  - ❖ TCP increases CongWin by:  $MSS \times (MSS/CongWin)$  for every ACK received
  - ❖ Ex.  $MSS = 1,460$  bytes and  $CongWin = 14,600$  bytes
  - ❖ With every ACK, CongWin is increased by 146 bytes
- multiplicative decrease:
  - ❖ cut CongWin in half after loss

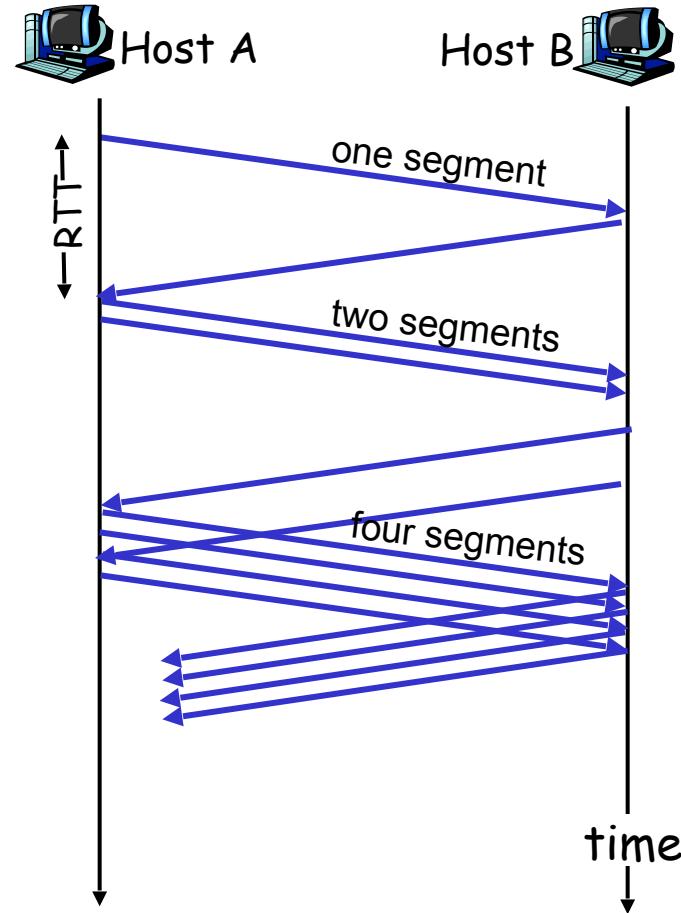


# TCP Slow Start

- When connection begins, CongWin = 1 MSS
  - ❖ Example: MSS = 500 bytes & RTT = 200 msec
  - ❖ initial rate = CongWin/RTT = 20 kbps
- available bandwidth may be >> MSS/RTT
  - ❖ desirable to quickly ramp up to respectable rate
- Slow start:
  - ❖ When connection begins, increase rate exponentially fast until first loss event. **How can we do that?**
  - ❖ double CongWin every RTT. **How?**
  - ❖ Increment CongWin by 1 MSS for every ACK received

# TCP Slow Start (cont'd)

- ❑ Increment CongWin by 1 MSS for every ACK
- ❑ Summary: initial rate is slow but ramps up exponentially fast



# Reaction to a Loss event

- ❑ TCP Tahoe (Old)
  - ❖ Threshold = CongWin / 2
  - ❖ Set CongWin = 1 MSS
  - ❖ Slow start till threshold
  - ❖ Then Additive Increase                    // congestion avoidance
  
- ❑ TCP Reno (most current TCP implementations)
  - ❖ If 3 dup acks                            // fast retransmit
    - Threshold = CongWin / 2
    - Set CongWin = Threshold + 3 MSS // fast recovery
    - Additive Increase
  - ❖ Else                                      // timeout
    - Same as TCP Tahoe

# Reaction to a Loss event (cont'd)

Congestion  
Window  
(seg)

● TCP Tahoe

● TCP Reno

- Why differentiate between 3 dup acks and timeout?
- 3 dup ACKs indicate network capable of delivering some segments
- timeout indicates a "more alarming" congestion scenario

# TCP Congestion Control: Summary

- ❑ Initially
  - ❖ Threshold is set to large value (65 Kbytes), has no effect
  - ❖ CongWin = 1 MSS
- ❑ Slow Start (SS): CongWin grows exponentially
  - ❖ till loss event occurs (timeout or 3 dup ACKs) or reaches Threshold
- ❑ Congestion Avoidance (CA): CongWin grows linearly
- ❑ 3 duplicate ACK occurs:
  - ❖  $\text{Threshold} = \text{CongWin}/2$ ;  $\text{CongWin} = \text{Threshold} + 3 \text{ MSS}$ ; CA
- ❑ Timeout occurs:
  - ❖  $\text{Threshold} = \text{CongWin}/2$ ;  $\text{CongWin} = 1 \text{ MSS}$ ; SS till Threshold

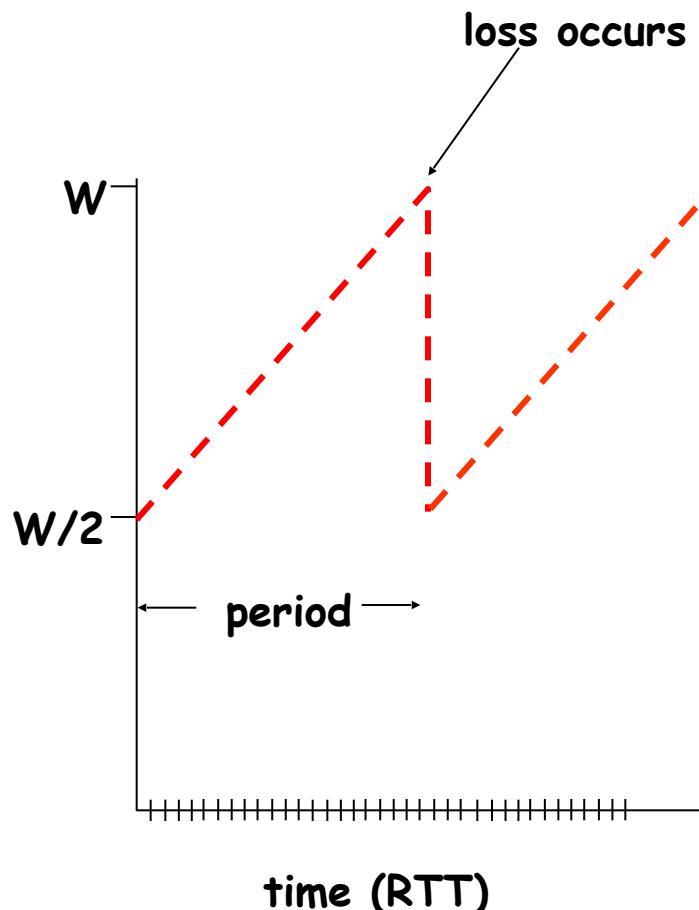
# TCP Throughput Analysis

- Understand the fundamental relationship between
  - ❖ Packet loss probability,
  - ❖ RTT, and
  - ❖ TCP performance (throughput)
- We present **simple** model, with several assumptions
  - ❖ Yet it provides quite useful **insights**

# TCP Throughput Analysis

- Any TCP model must capture
  - ❖ Window Dynamics (internal and deterministic)
    - Controlled internally by the TCP algorithm
    - Depends on the particular flavor of TCP
    - We assume TCP Reno (the most common)
  - ❖ Packet Loss Process (external and uncertain)
    - Models aggregate network conditions across all nodes on the TCP connection path
    - Typically modeled as Stochastic Process with probability  $p$  that a packet loss occurs
    - TCP responds by reducing the window size
- We usually analyze the steady state
  - ❖ Ignore the slow start phase (transient)
  - ❖ Although many connections finish within slow start, because they send only a few kilobytes

# Simple (Periodic) Model



- Packet losses occur with constant probability  $p$
- TCP window starts at  $W/2$  grows to  $W$ , then halves, repeat forever ... →
- TCP Throughput ranges between:
  - Min:  $(W/2) / \text{RTT}$ , and
  - Max:  $W / \text{RTT} \rightarrow$

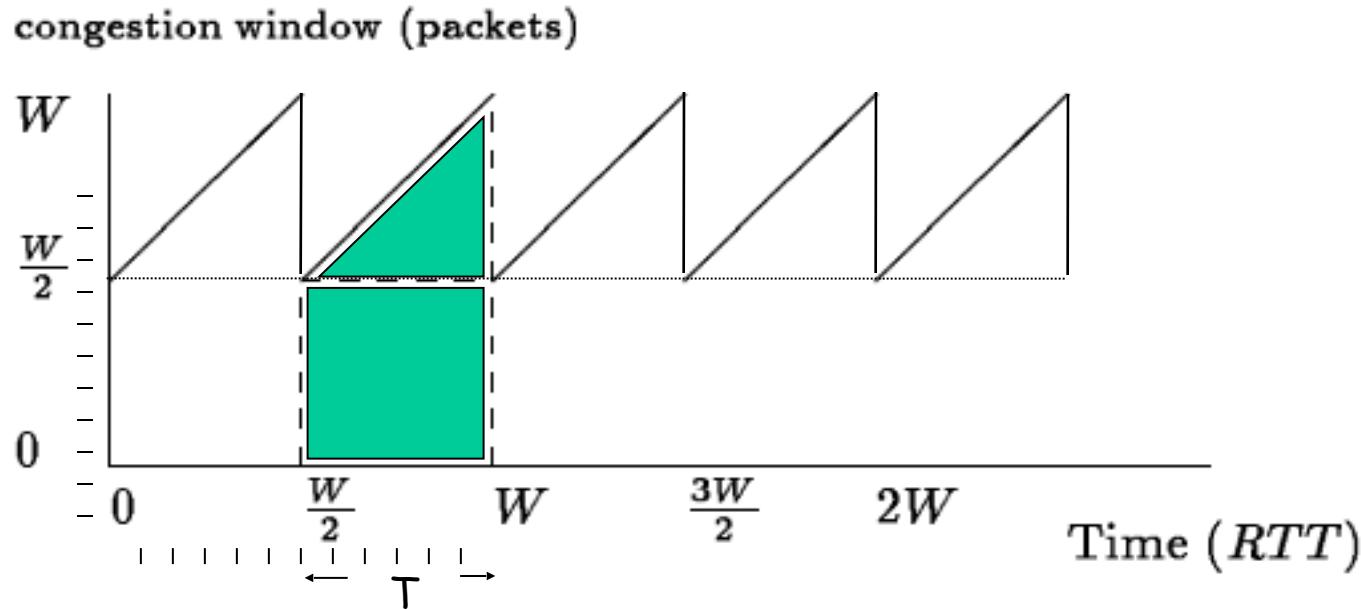
$$\text{Avg throughput} = (3/4 W) / \text{RTT} \quad (1)$$

# Simple (Periodic) Model

$$\text{Avg throughput} = (3/4 W) / \text{RTT} \quad (1)$$

- Now, we want to relate  $W$  (max window size) with the packet loss rate in the network  $p$ 
  - So that we have the TCP throughput as function of RTT and loss rate (both are external network parameters)

# Simple (Periodic) Model



Throughput  $X(t) = \text{green area (packets sent)} / T$

$$X(t) = \frac{\frac{W}{2} \times \frac{W}{2} + \frac{1}{2} \times \frac{W}{2} \times \frac{W}{2}}{T} = \frac{\frac{3}{8} W^2}{T} \quad (2)$$

# Simple (Periodic) Model

- On the other hand, we have average packet loss rate of  $p$ 
  - ❖ How many packets we send until we observe a loss?
  - ❖  $1/p$
- Throughput  $X(t) = \text{number of packets sent} / T \rightarrow$

$$X(t) = \frac{1/p}{T} \quad (3)$$

- Solve (2) and (3)  $\rightarrow$

$$W = \sqrt{\frac{8}{3p}}$$

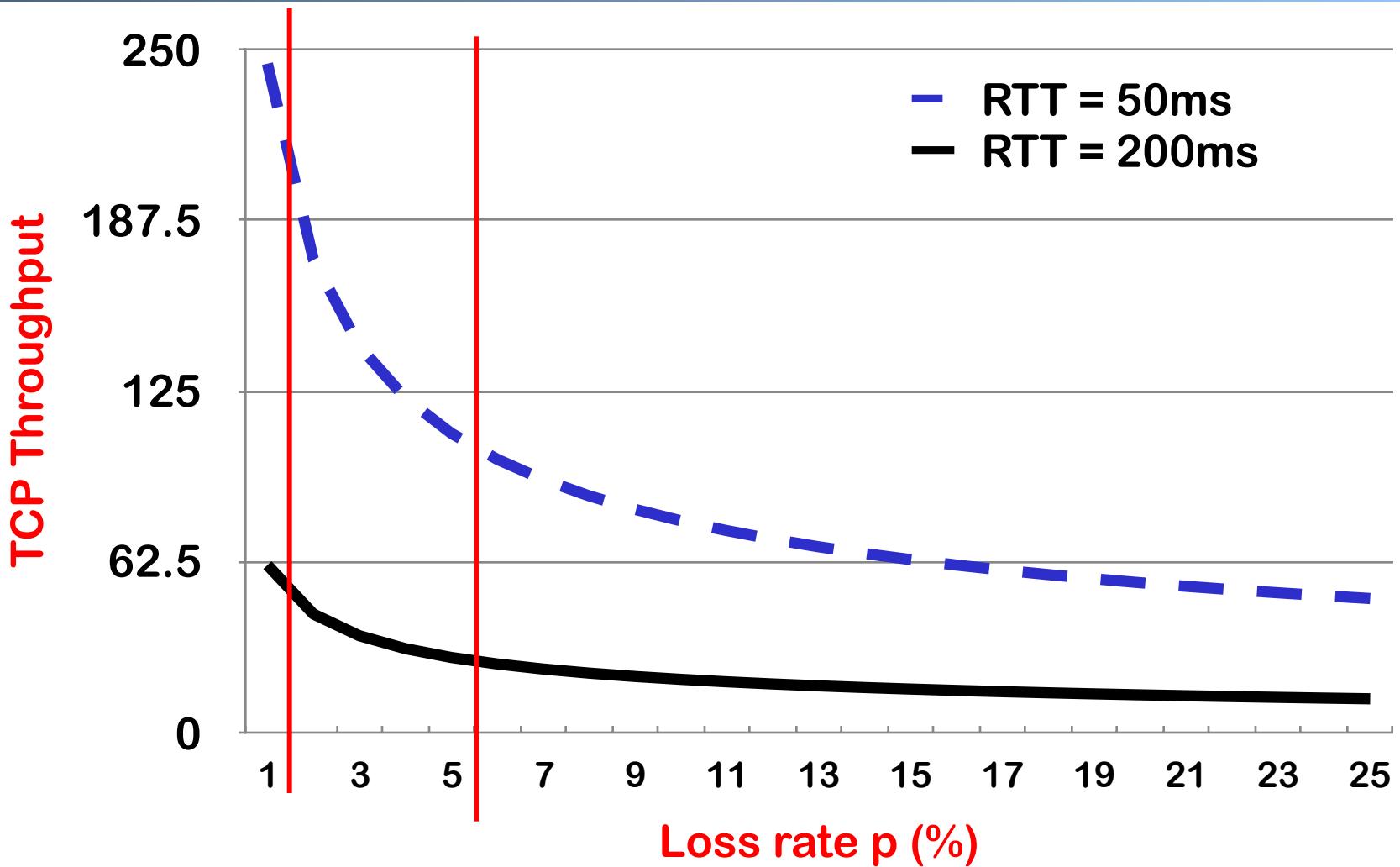
# Simple (Periodic) Model

- Substitute W in (1) →

$$X(p) = \frac{1}{RTT} \sqrt{\frac{3}{2p}}$$

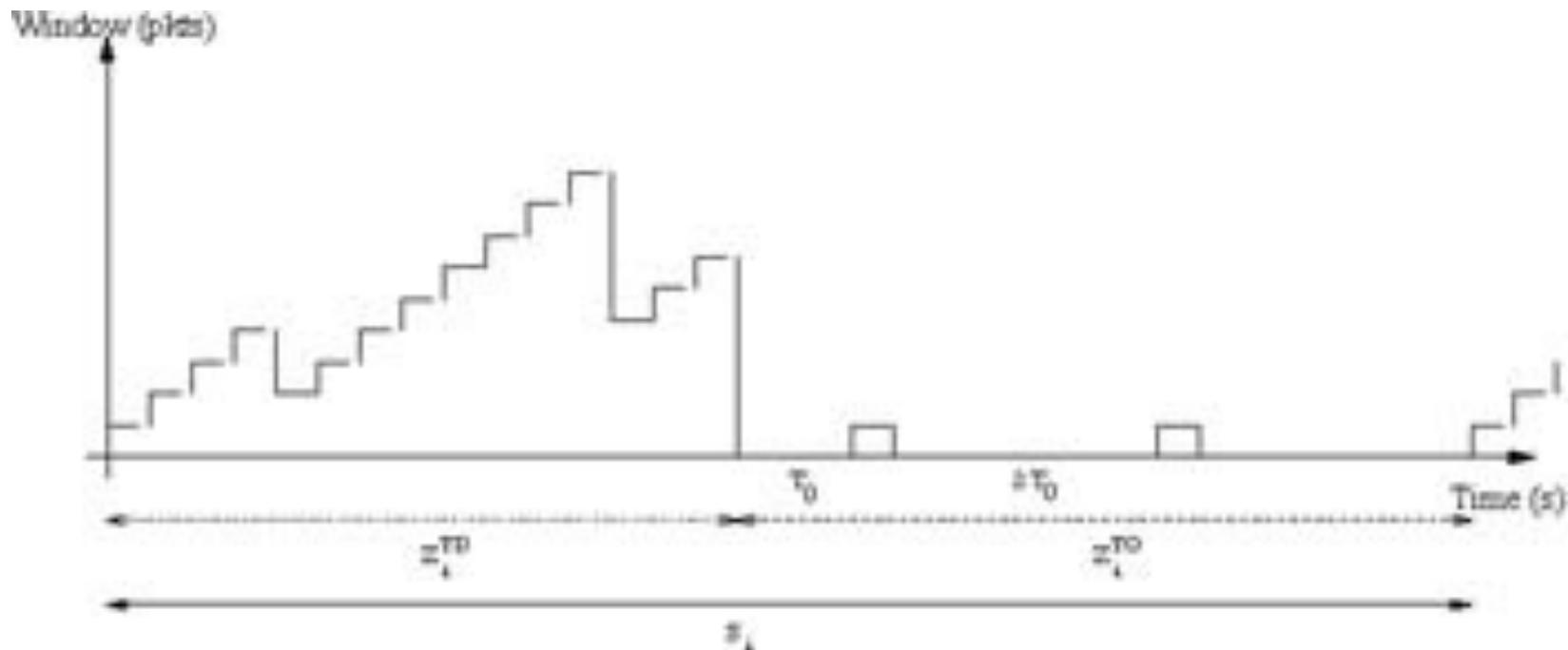
- Called **Inverse Square-Root-p Law**
- TCP throughput is inversely proportional to RTT and square root of packet loss probability p

# Impact of RTT & Loss Rate on TCP



## In More Realistic Models ...

- ❑ Packet loss probability is not constant and is bursty
- ❑ Consider effect of duplicate ACKs and Timeouts
- ❑ Consider receiver window limit



# Other models

---

# TCP Performance over Wireless Networks

---

## ❑ Performance of TCP suffers:

- ❖ First recall that TCP interprets loss as congestion
- ❖ But in wireless networks, packets can be lost because of bit-errors (usually high) and handoff (long delays)
- ❖ Thus, TCP may **un-necessarily** decrease its sending rate (congestion window)

## ❑ Solutions?

# Enhancing TCP Performance in Wireless Networks

## ❑ Local Recovery

- ❖ ARQ: local retransmission (e.g., in 802.11)
- ❖ FEC: for long-delay networks (e.g., cellular)

## ❑ Make TCP aware of the wireless link

- ❖ Distinguish loss due to congestion from others
- ❖ Decrease sending rate only in congestion

## ❑ Split TCP connections

- ❖ One from mobile to base station, and another from base station to the other end
- ❖ Over the wireless part, we can use either standard TCP (shorter delay now) or custom/new transport protocols, e.g. reliable UDP, TCP with selective repeat
- ❖ Used in cellular networks, significant improvement in TCP

# Summary

---

- ❑ Transport layer: logical channel between processes
- ❑ Main protocols: TCP and UDP
- ❑ TCP:
  - ❖ Reliable
  - ❖ Congestion control
  - ❖ Flow control
- ❑ Simple analytic model for TCP
  - ❖ Throughput inversely proportional with RTT and greatly affected by packet loss rate
  - ❖ TCP performance over wireless networks may suffer: performance mitigation

# Resource Management and QoS

---