CSCE 463/612
Networks and Distributed Processing
Fall 2024

Transport Layer V

Dmitri Loguinov
Texas A&M University

October 17, 2024

Chapter 3: Roadmap

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

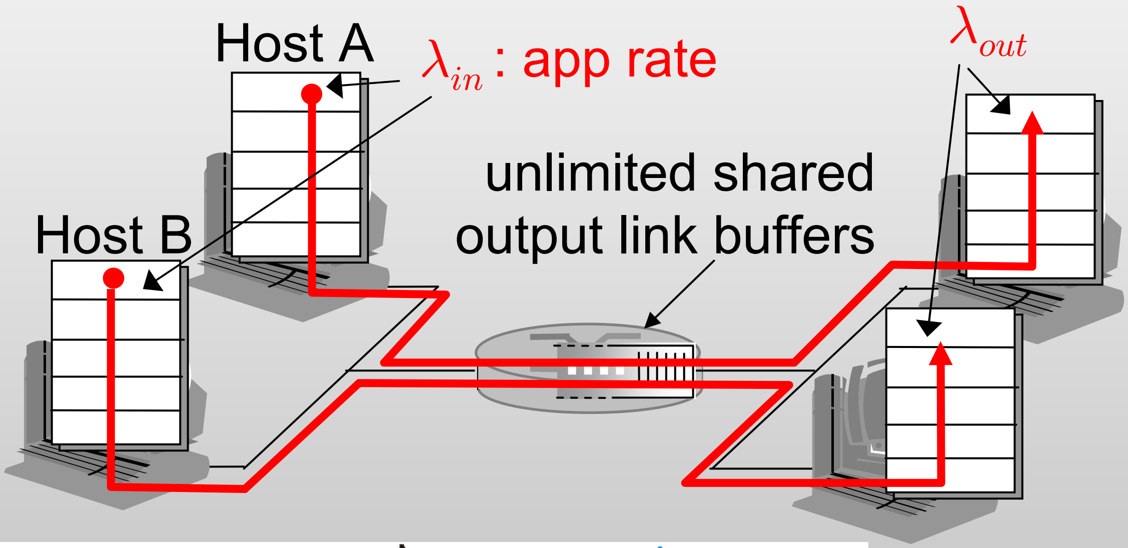
Principles of Congestion Control

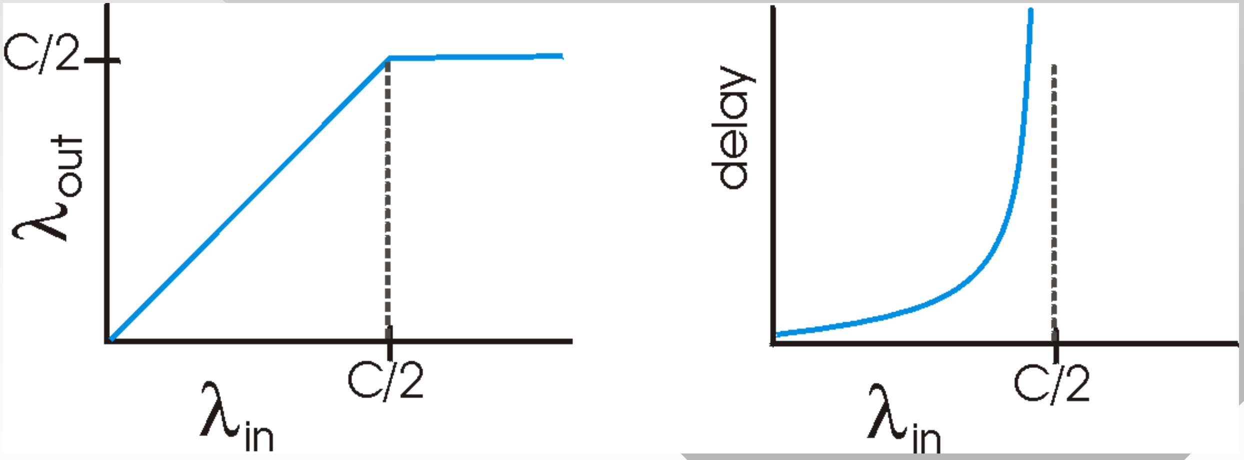
Congestion:

- Informally: “too many sources sending data too fast for the *network* to handle”
- Different from flow control!
- Manifestations:
  - Lost packets (buffer overflows)
  - Delays (queueing in routers)
- Important networking problem

Causes/Costs of Congestion: Scenario 1

- Two senders, two receivers
- One router of capacity C, infinite buffers, no loss
- No retransmission

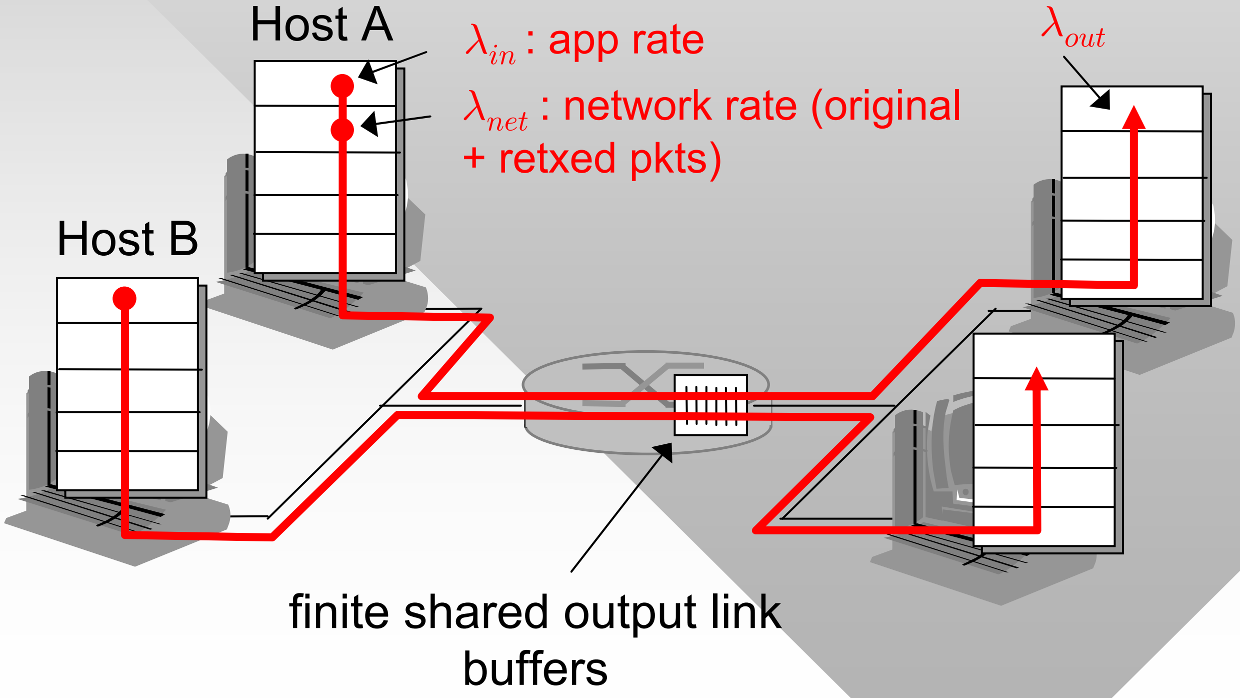




Cost 1: queuing delays in congested routers

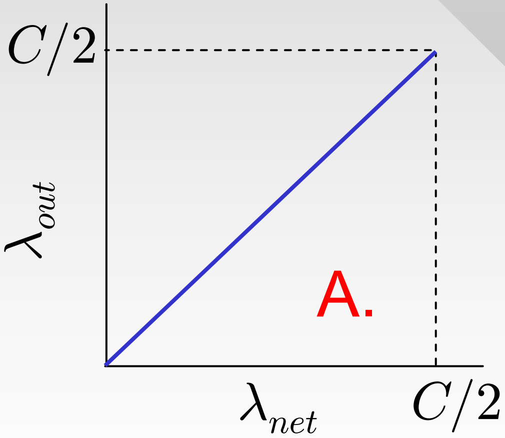
Causes/Costs of Congestion: Scenario 2

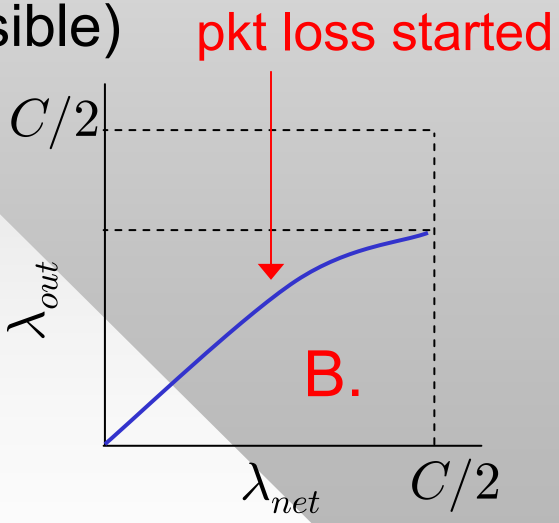
- One router, *finite* buffers (pkt loss is possible now)
- Sender retransmission of lost packet
- During congestion $2\lambda_{net} = 2(\lambda_{in} + \lambda_{retx}) = C$

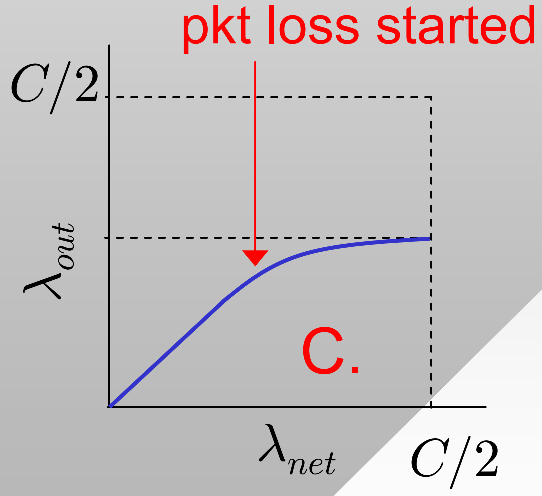


Causes/Costs of Congestion: Scenario 2

- We call λ_{out} *goodput* and λ_{net} throughput
    - Case A: pkts never lost while $\lambda_{net} < C/2$ (not realistic)
    - Case B: pkts are lost when λ_{net} is “sufficiently large,” but timeouts are perfectly accurate (not realistic either)
    - Case C: same as B, but timer is not perfect (duplicate packets are possible)





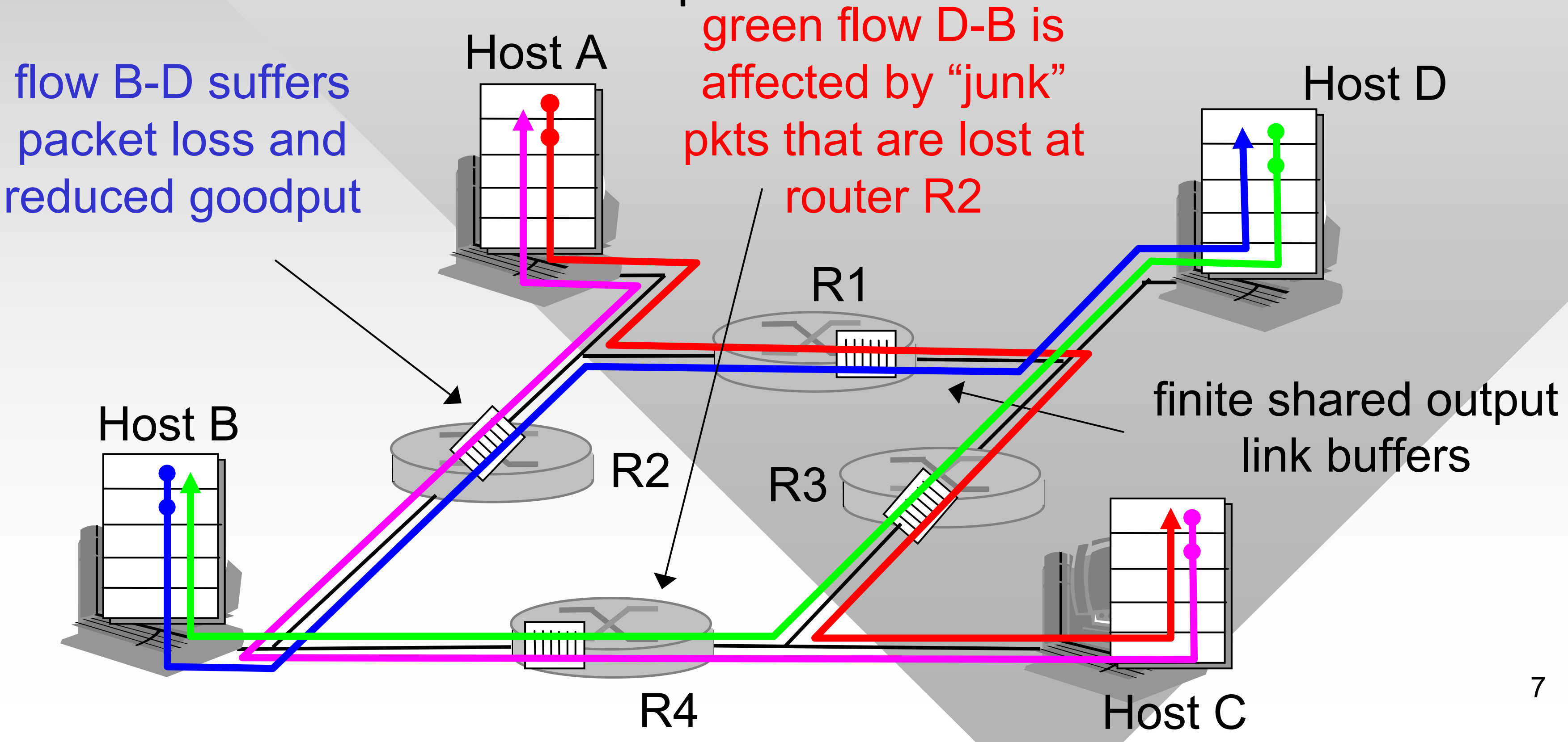


Cost 2: retransmission of lost packets and premature timeouts increase network load, reduce *flow’s own* goodput

Causes/Costs of Congestion: Scenario 3

- Multihop case
  - Timeout/retransmit
  - R2 = 50 Mbps, R1 = R3 = R4 = 100 Mbps
  - Flow C-A: sends 90 Mbps

Cost 3: congestion causes goodput reduction for *other* flows



Approaches Towards Congestion Control

Two broad approaches towards congestion control:

End-to-end:

- No explicit feedback from network
- Congestion *inferred* by end-systems from observed loss/delay
  - Approach taken by TCP (relies on loss)

Network-assisted:

- Routers provide feedback to end systems
  - Single bit indicating congestion (DECbit, TCP/IP ECN)
  - Two bits (ATM)
  - Explicit rate senders should send at (ATM)

ATM = Asynchronous Transfer Mode

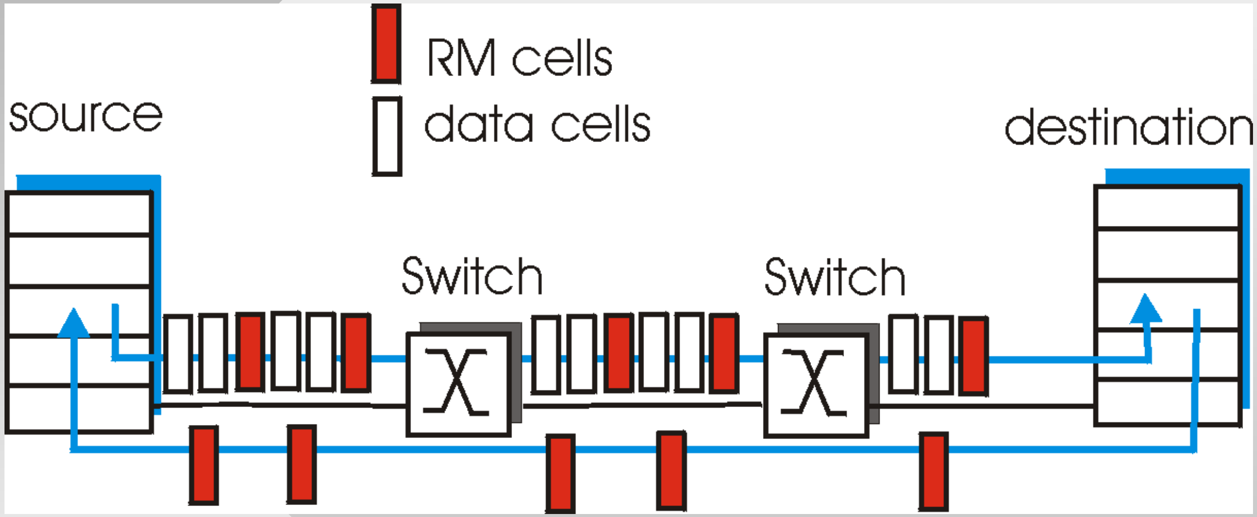
Case Study: ATM ABR Congestion Control

- For network-assisted protocols, the logic can be binary:
  - Path underloaded, increase rate
  - Path congested, reduce rate
- It can also be ternary
  - Increase, decrease, hold steady
  - ATM ABR (Available Bit Rate) profile

RM (resource management) packets (cells):

- Sent by sender, interspersed with data cells
- Bits in RM cell set by switches/routers
  - NI bit: no increase in rate (impending congestion)
  - CI bit: reduce rate (congestion in progress)
- RM cells returned to sender by receiver, with bits intact

Case Study: ATM ABR Congestion Control



- Additional approach is to use a two-byte ER (explicit rate) field in RM cell
  - Congested switch may lower ER value
  - Senders obtain the maximum supported rate on their path
- Issues with network-assisted congestion control?

Chapter 3: Roadmap

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 Congestion Control

- TCP congestion control has a variety of algorithms developed over the years
  - TCP Tahoe (1988), TCP Reno (1990), TCP SACK (1992)
  - TCP Vegas (1994), TCP New Reno (1996)
  - High-Speed TCP (2002), Scalable TCP (2002)
  - FAST TCP (2004), TCP Illinois (2006)
- Many others: H-TCP, CUBIC TCP, L-TCP, TCP Westwood, TCP Veno (Vegas + Reno), TCP Africa
- Linux: BIC TCP (2004), CUBIC TCP (2008)
- Vista and later: Compound TCP (2005)
  - Server 2019 switched to CUBIC
- Google: BBR (2016)

TCP Congestion Control

- End-to-end control (no network assistance)
- Sender limits transmission:
  `LastByteSent - LastByteAcked ≤ CongWin`
- `CongWin` is a function of perceived network congestion
- The *effective* window is the minimum of `CongWin`, flow-control window carried in the ACKs, and sender’s own buffer space

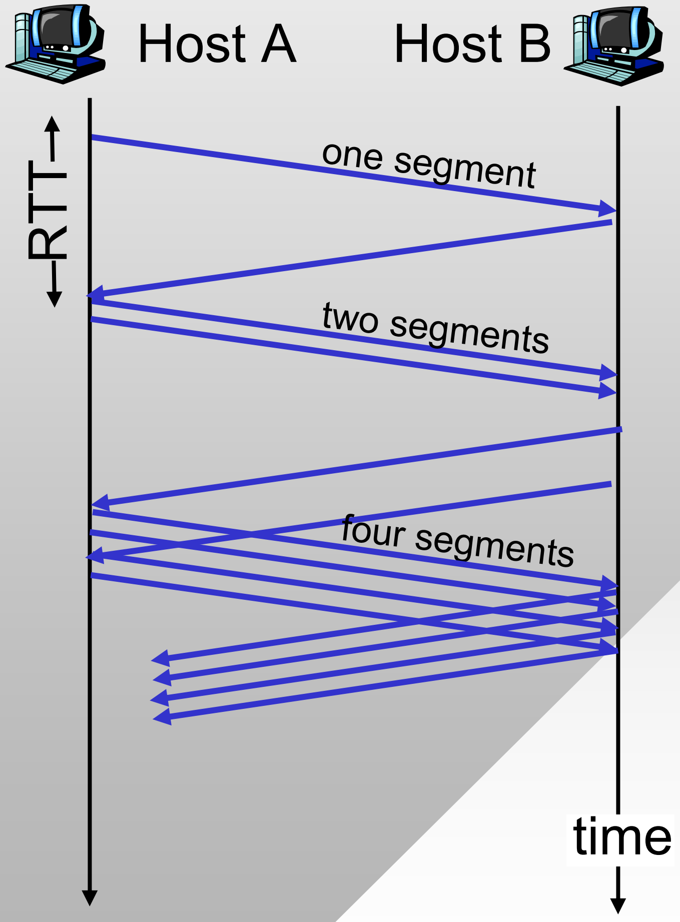
- How does sender perceive congestion?
  - Loss event = timeout or 3 duplicate acks
- TCP sender reduces rate (`CongWin`) after loss event
- Three mechanisms:
  - Slow start
  - Conservative after timeouts
  - AIMD (congestion avoidance)

TCP Slow Start

- When connection begins, `CongWin` $= 1$ MSS
  - Example: MSS = 500 bytes and RTT = 200 msec
  - Q: initial rate?
  - A: 20 Kbits/s
- Available bandwidth may be much larger than MSS/RTT
  - Desirable to quickly ramp up to a “respectable” rate
- Solution: Slow Start (SS)
  - When a connection begins, it increases rate exponentially fast until first loss or receiver window is reached
  - Term “slow” is used to distinguish this algorithm from earlier TCPs which directly jumped to some huge rate

TCP Slow Start (More)

- Let W be congestion window in pkts and $B =$ `CongWin` be the same in bytes ($B = MSS * W$)
- Slow start
  - Double `CongWin` every RTT
- Done by incrementing `CongWin` for every ACK received:
  - $W = W+1$ per ACK (or $B = B + MSS$)
- Summary: initial rate is slow but ramps up exponentially fast

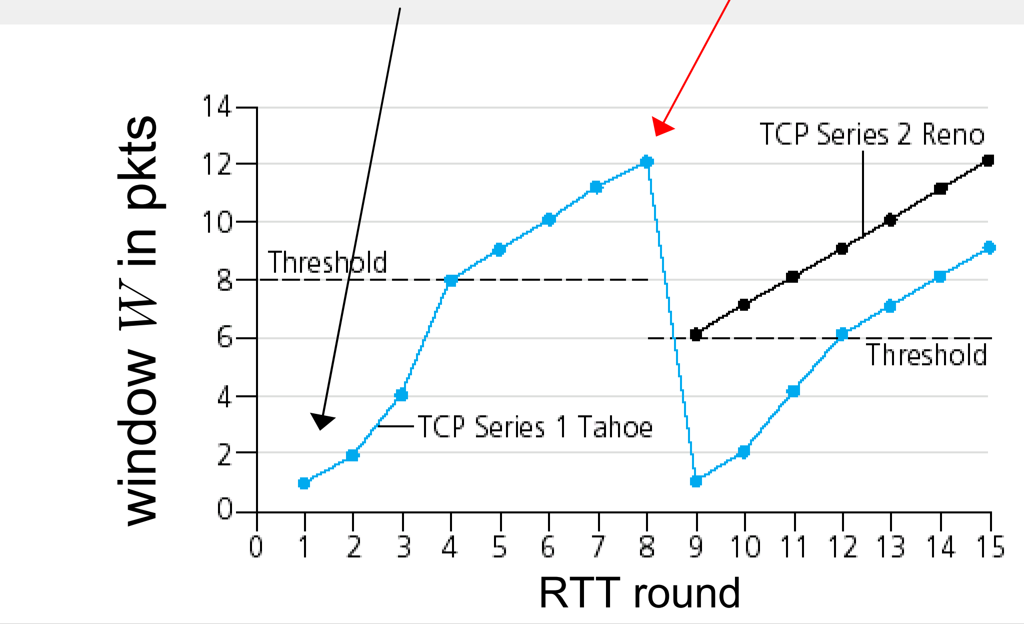


Congestion Avoidance

- TCP Tahoe loss (timeout or triple dup ACK):
  - `Threshold = CongWin/2`
  - `CongWin` is set to 1 MSS
  - Slow start until `threshold` is reached; then move to linear probing
- TCP Reno loss:
  - Timeout: same as Tahoe
  - 3 dup ACKs: `CongWin` is cut in half (method called fast recovery)

loss detected via triple dup ACK

previous timeout



Fast Recovery Philosophy:

Three dup ACKs indicate that network is capable of delivering subsequent segments

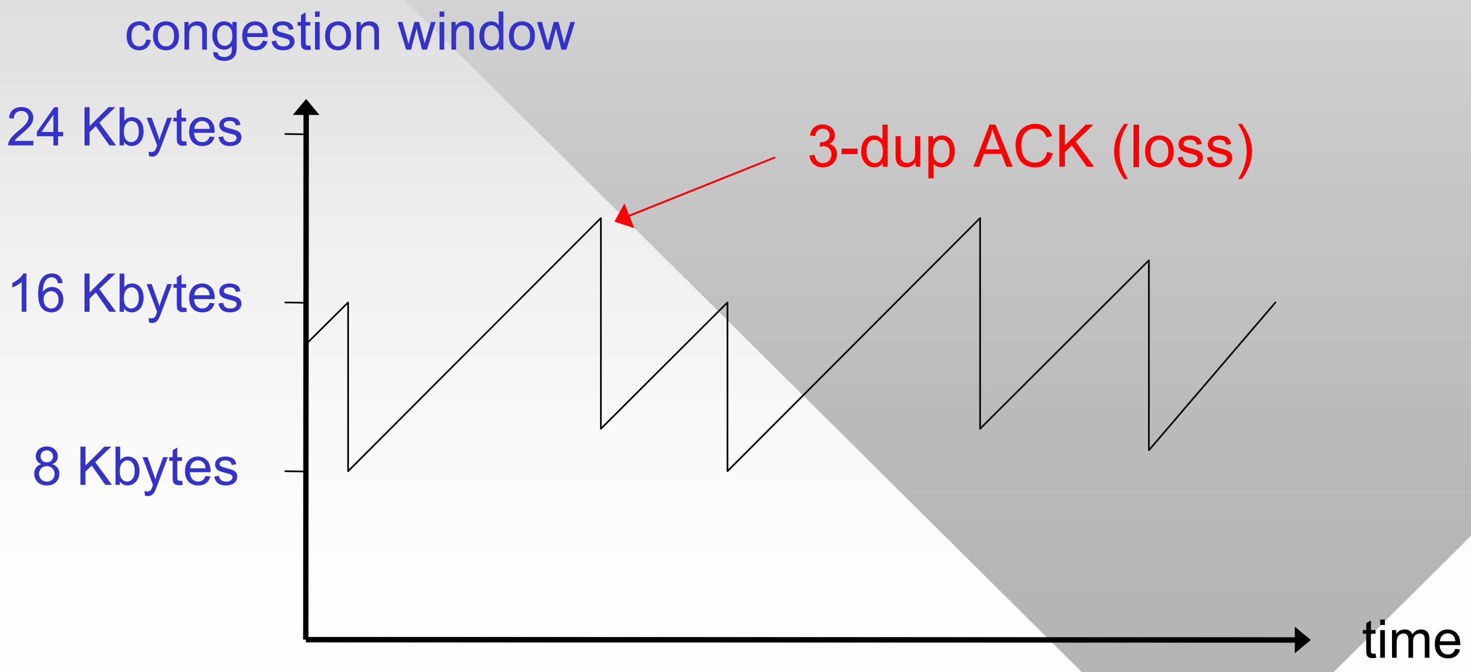
Timeout before 3-dup ACK is more alarming

TCP Reno AIMD (Additive Increase, Multiplicative Decrease)

Additive increase: increase `CongWin` by 1 MSS every RTT in the absence of loss events: *probing*

Multiplicative decrease: cut `CongWin` in half after fast retransmit (3-dup ACKs)

Peaks are different: # of flows or RTT changes



TCP Reno Equations

- To better understand TCP, we next examine its AIMD equations (congestion avoidance)
- General form (loss detected through 3-dup ACK):

$$W = \begin{cases} W + \frac{1}{W} & \text{per ACK} \\ W/2 & \text{per loss} \end{cases}$$

- Reasoning
  - For each window of size W, we get exactly W acknowledgments in one RTT (assuming no loss!)
  - This increases window size by roughly 1 packet per RTT
- In general, many other protocols also perform actions on packet arrival rather than timers

TCP Reno Equations

$$W = \begin{cases} W + \frac{1}{W} & \text{per ACK} \\ W/2 & \text{per loss} \end{cases}$$

- What is the equation in terms of $B = MSS * W$?

$$B = \begin{cases} B + \frac{MSS^2}{B} & \text{per ACK} \\ B/2 & \text{per loss} \end{cases}$$

- Equivalently, TCP increases B by MSS per RTT
- What is the rate of TCP given that its window size is B (or W)?
- Since TCP sends a full window of pkts per RTT, its ideal rate can be written as:

$$r = \frac{B}{RTT + L/R} \approx \frac{B}{RTT} = \frac{MSS * W}{RTT}$$

TCP Reno Sender Congestion Control

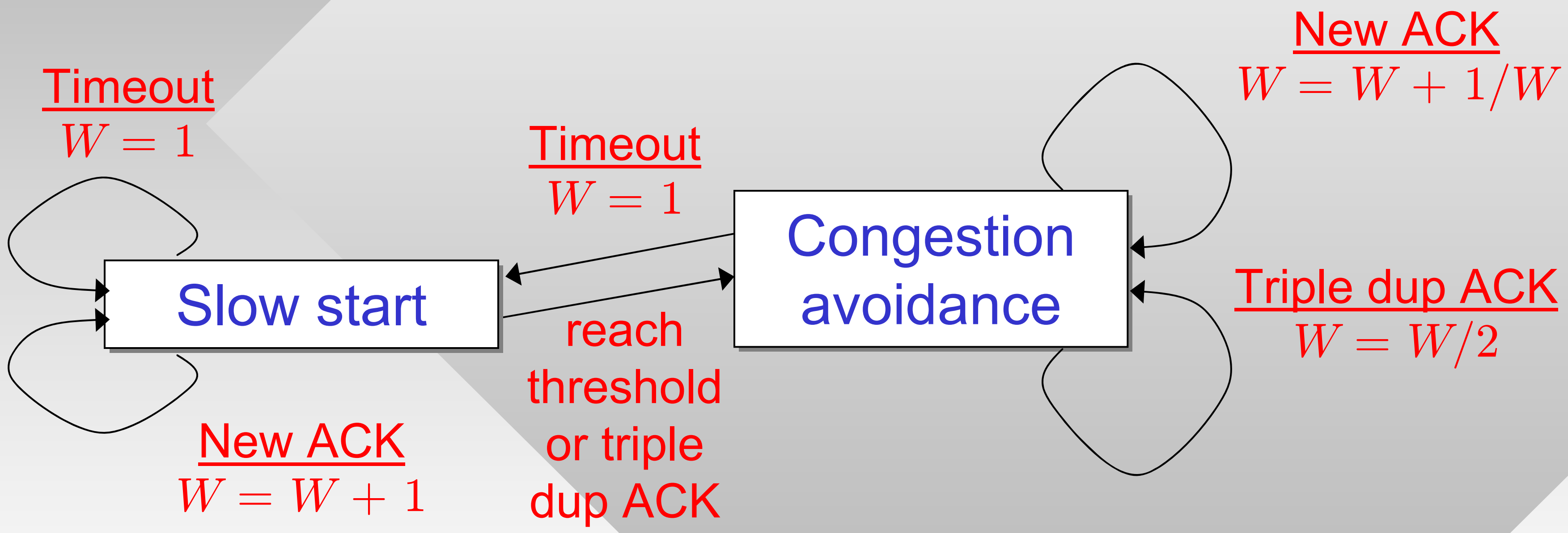
Event	State	TCP Sender Action	Commentary

ACK receipt for previously unacked data	Slow Start (SS)	CongWin += MSS,
		If (CongWin >= ssthresh) {
		Set state to “Congestion Avoidance”
		}
			Results in a doubling of CongWin every RTT

ACK receipt for previously unacked data	Congestion Avoidance (CA)	CongWin += MSS^2 / CongWin	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
Loss event detected by triple duplicate ACK	SS or CA	ssthresh = max(CongWin/2, MSS) CongWin = ssthresh Set state to “Congestion Avoidance”	Fast recovery, implementing multiplicative decrease
Timeout	SS or CA	ssthresh = max(CongWin/2, MSS) CongWin = MSS Set state to “Slow Start”	Enter slow start
Duplicate ACK	SS or CA	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

TCP Reno Congestion Control

- Summary:



Slow start

Congestion avoidance

Timeout
$W = 1$

New ACK
$W = W + 1$

Timeout
$W = 1$

reach threshold or triple dup ACK

New ACK
$W = W + 1/W$

Triple dup ACK
$W = W/2$