COMP 3331/9331: Computer Networks and Applications

Week 6

Congestion Control (Transport Layer)

Reading Guide: Chapter 3, Sections: 3.6-3.7

Transport Layer: Outline

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

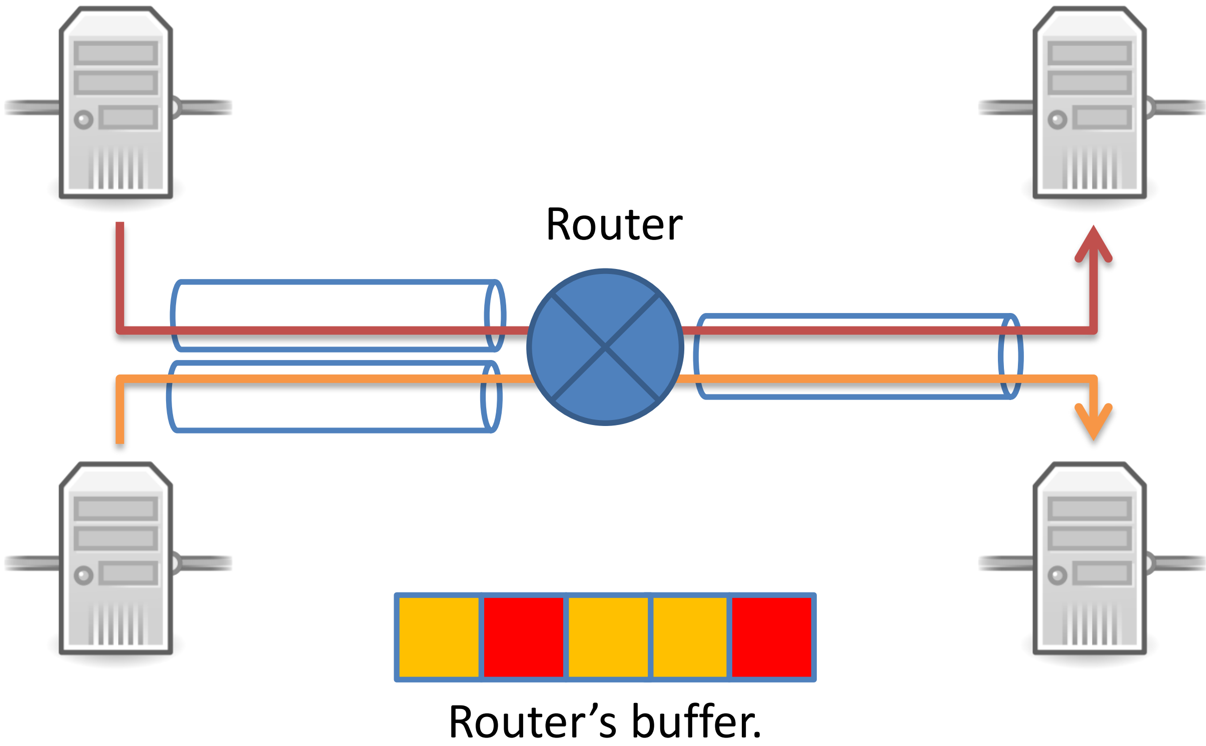
3.7 TCP congestion control

Principles of congestion control

congestion:

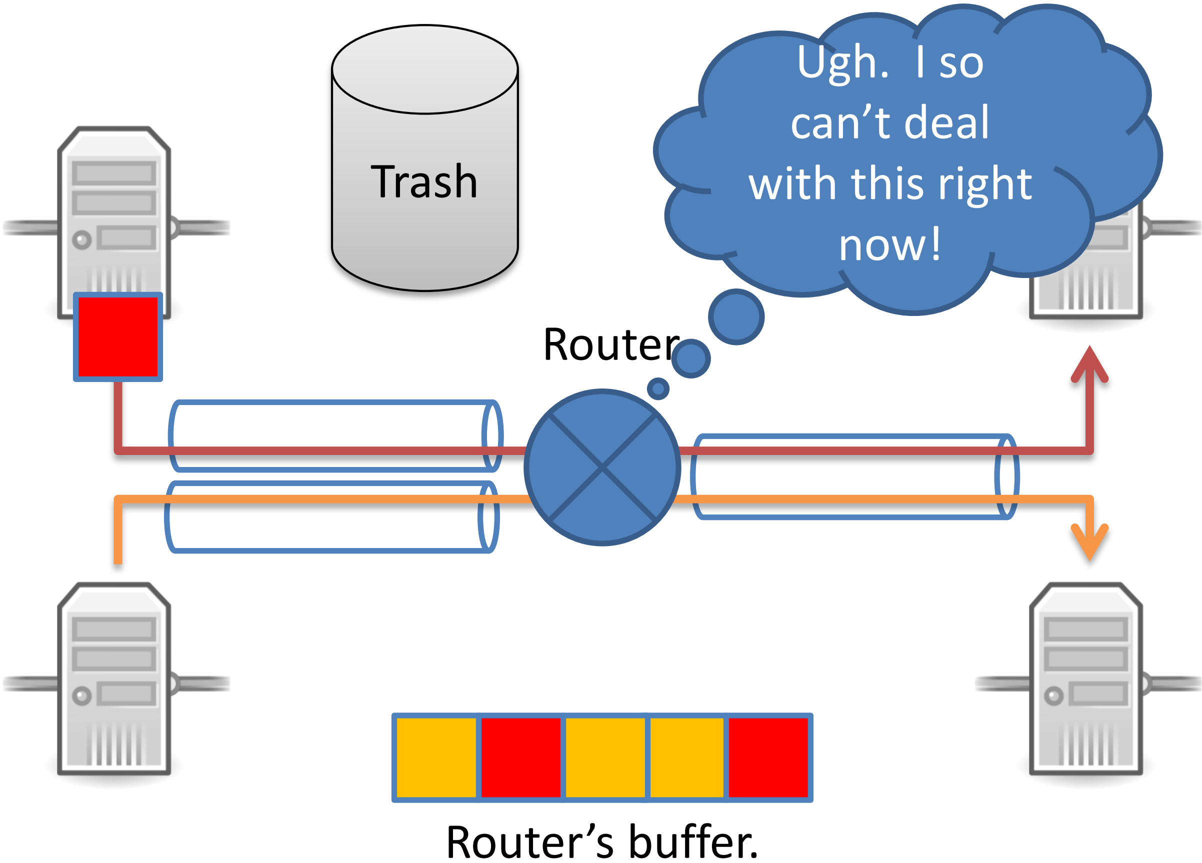
- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

Congestion



Incoming rate is faster than outgoing link can support.

Congestion



Incoming rate is faster than outgoing link can support.

Quiz: What's the worst that can happen?

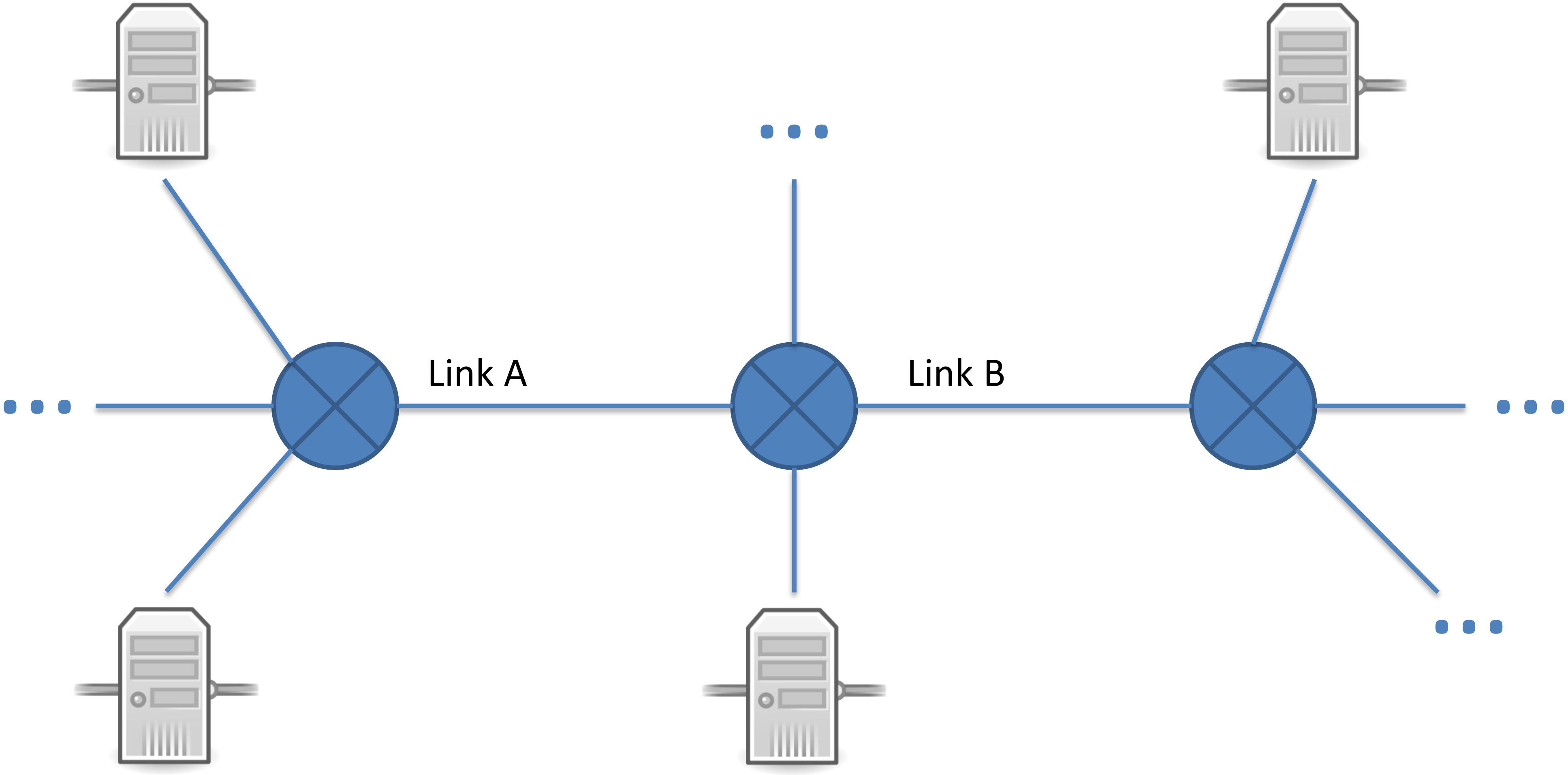
A: This is no problem. Senders just keep transmitting, and it'll all work out.

B: There will be retransmissions, but the network will still perform without much trouble.

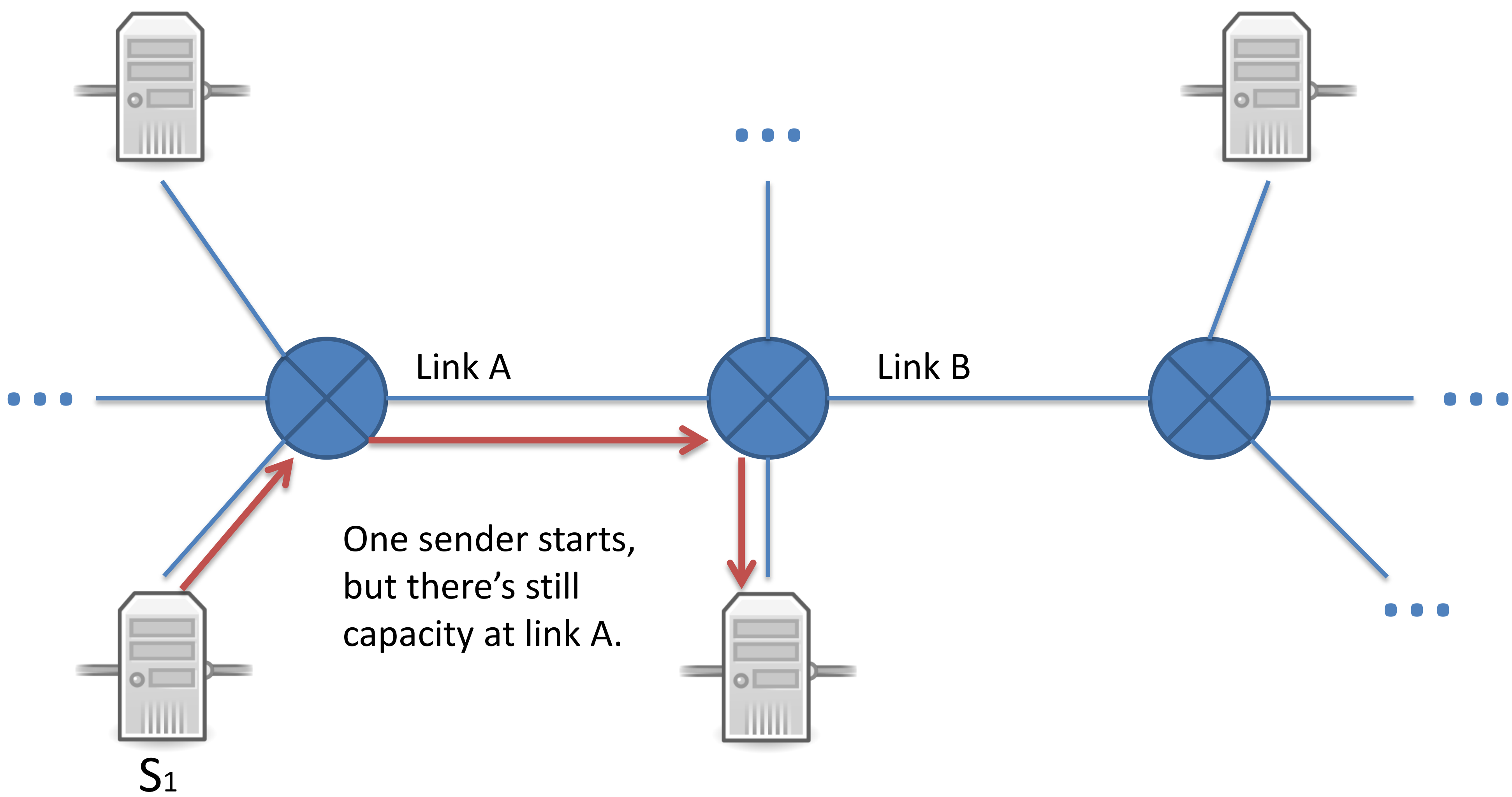
C: Retransmissions will become very frequent, causing a serious loss of efficiency

D: The network will become completely unusable

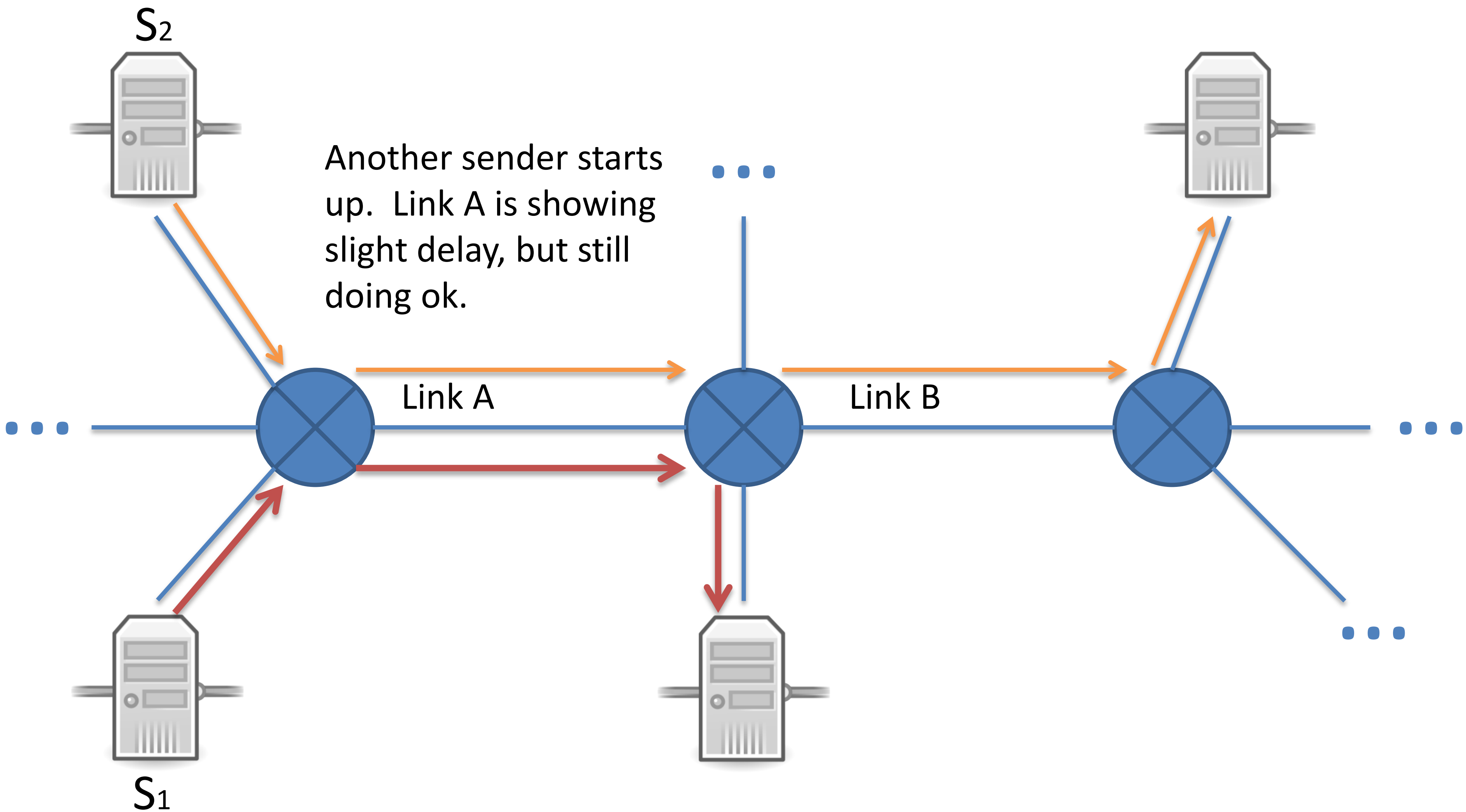
Congestion Collapse



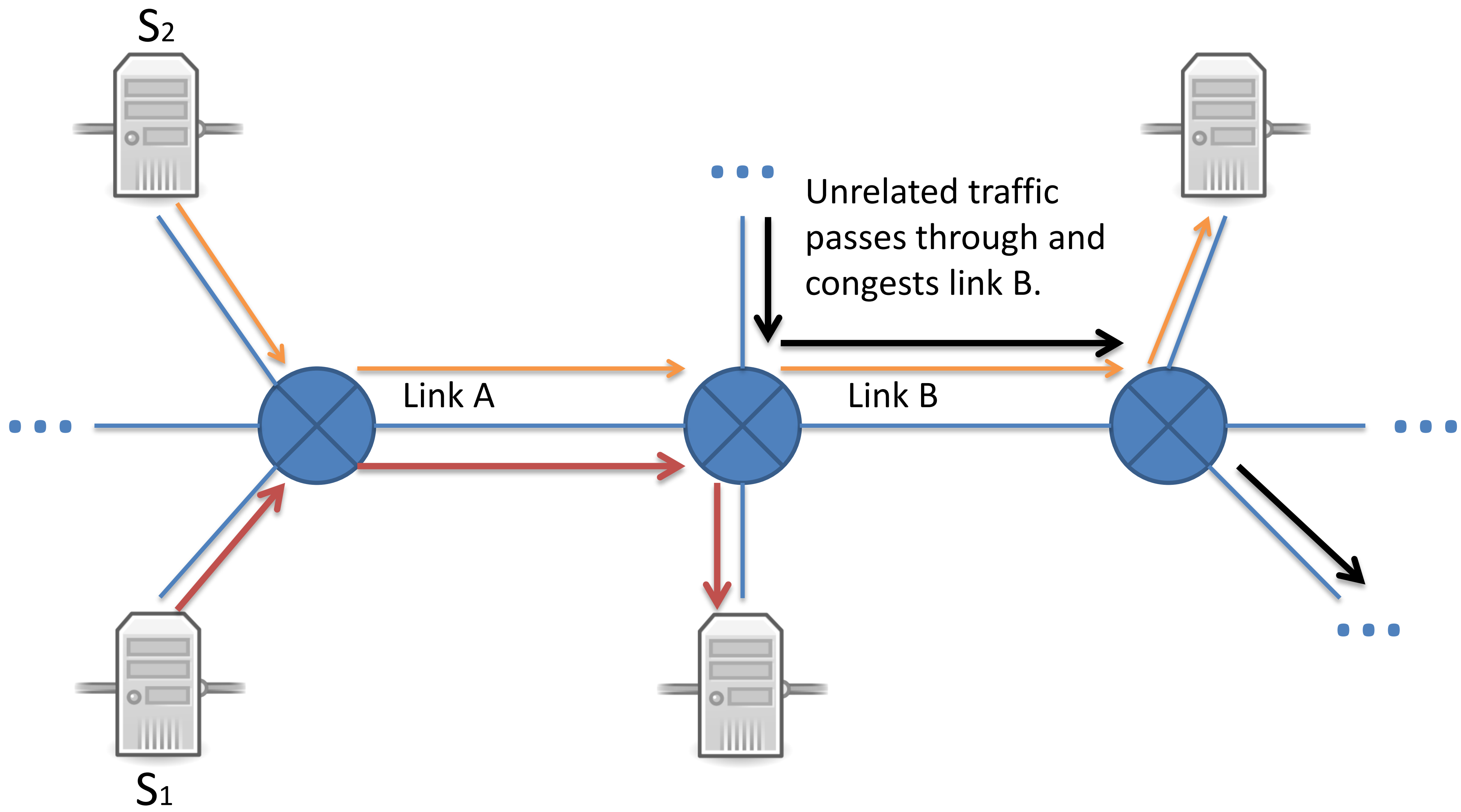
Congestion Collapse



Congestion Collapse

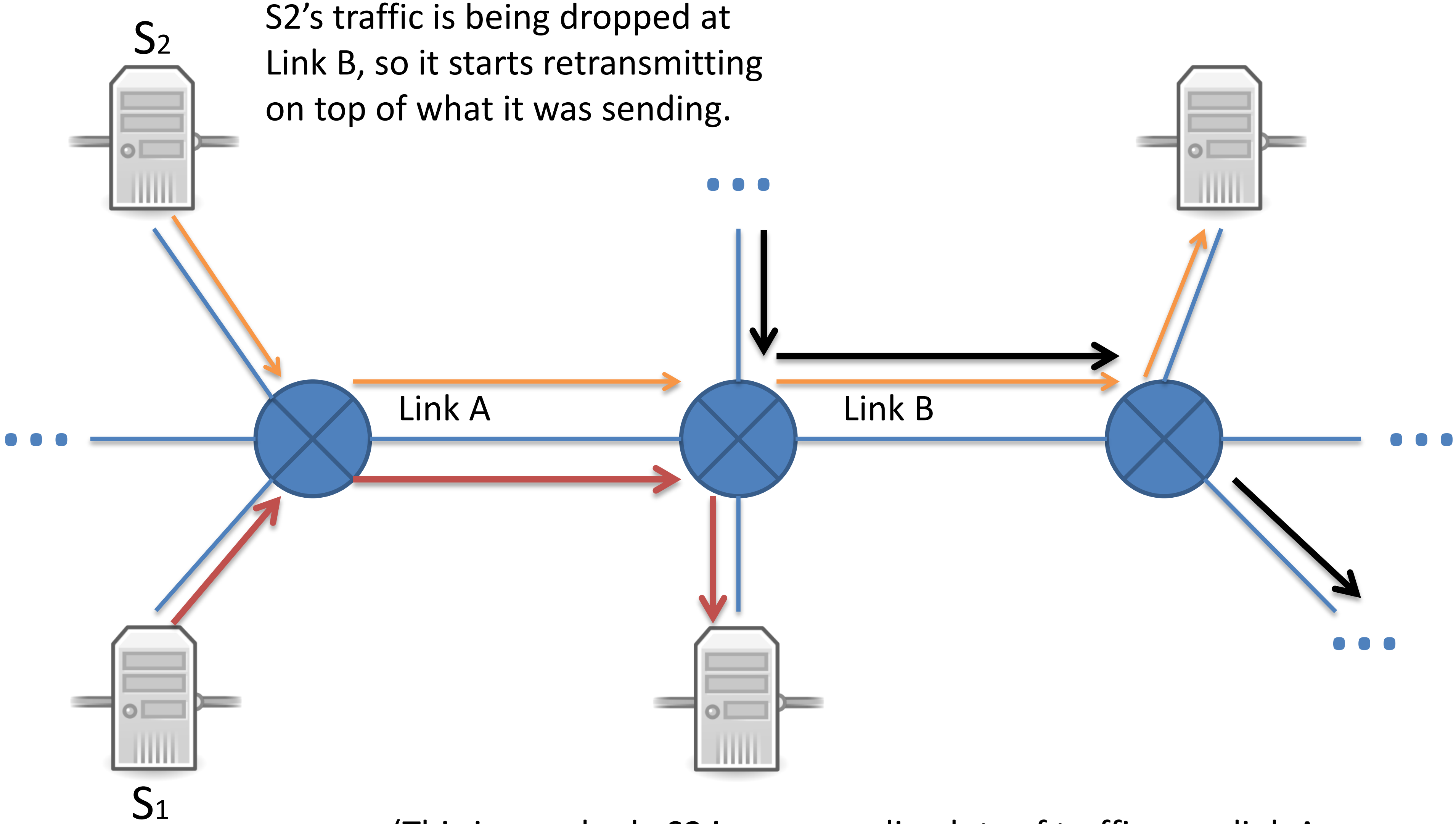


Congestion Collapse



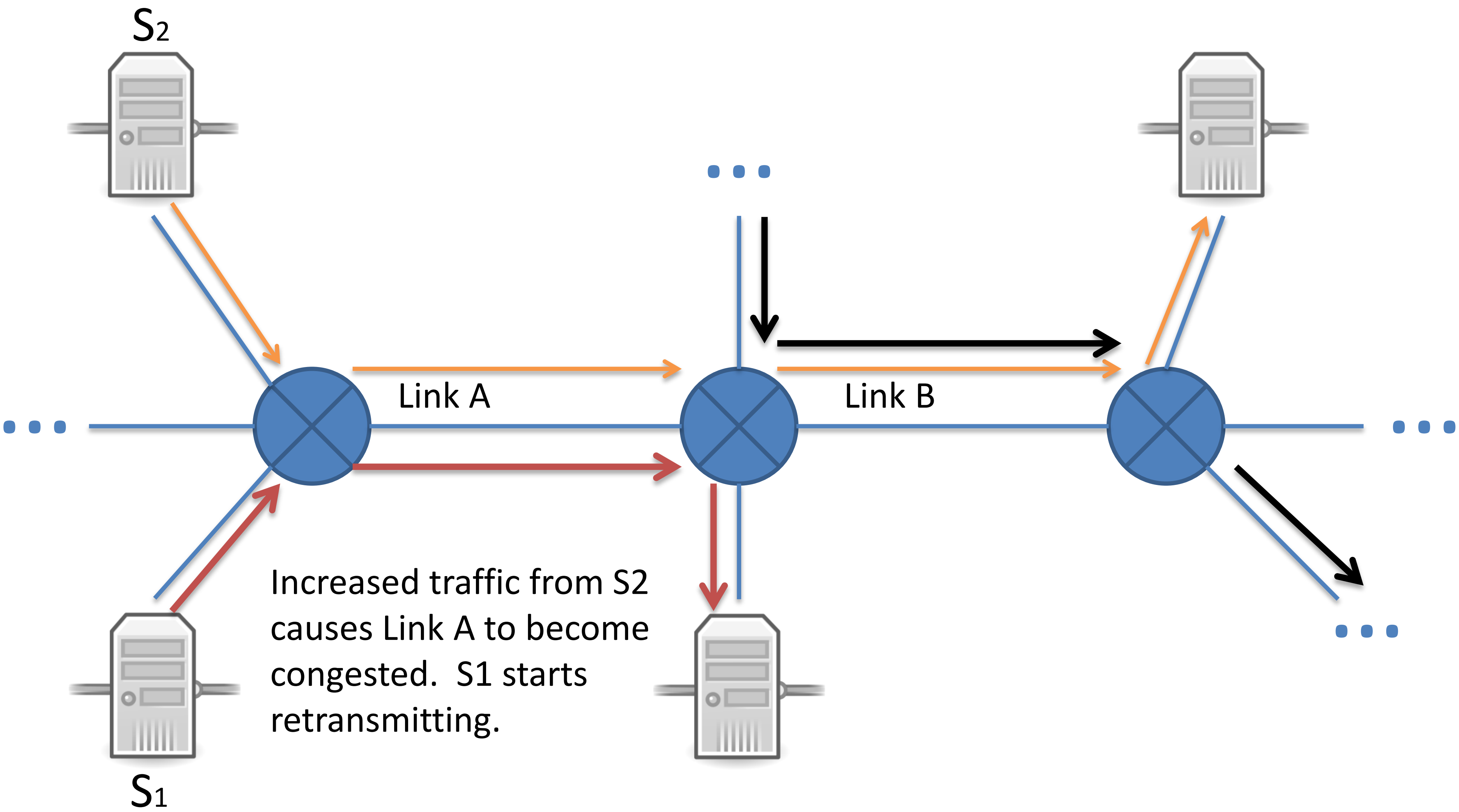
Congestion Collapse

S2's traffic is being dropped at Link B, so it starts retransmitting on top of what it was sending.



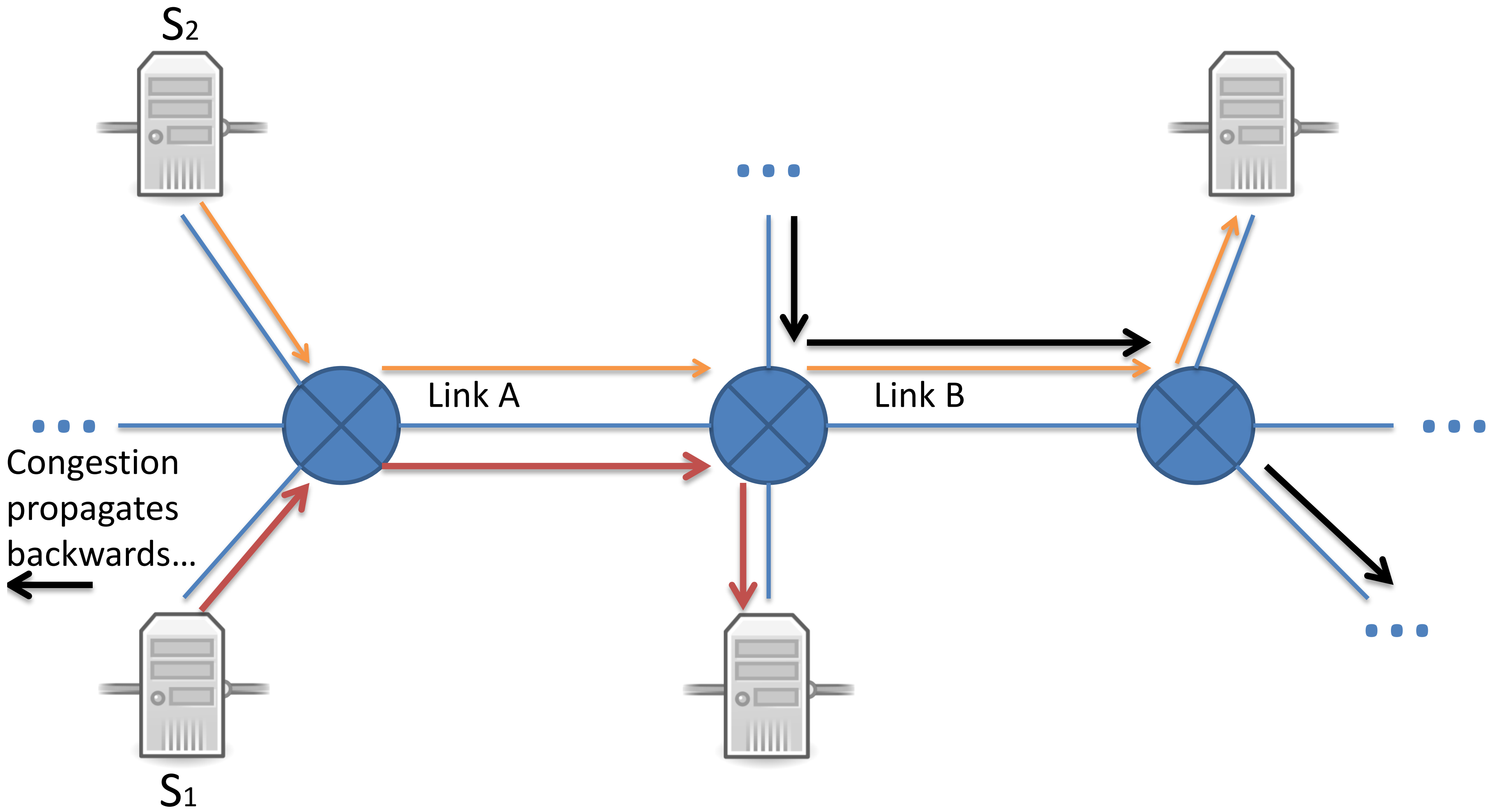
(This is very bad. S2 is now sending lots of traffic over link A that has no hope of crossing link B.)

Congestion Collapse



Increased traffic from S2 causes Link A to become congested. S1 starts retransmitting.

Congestion Collapse



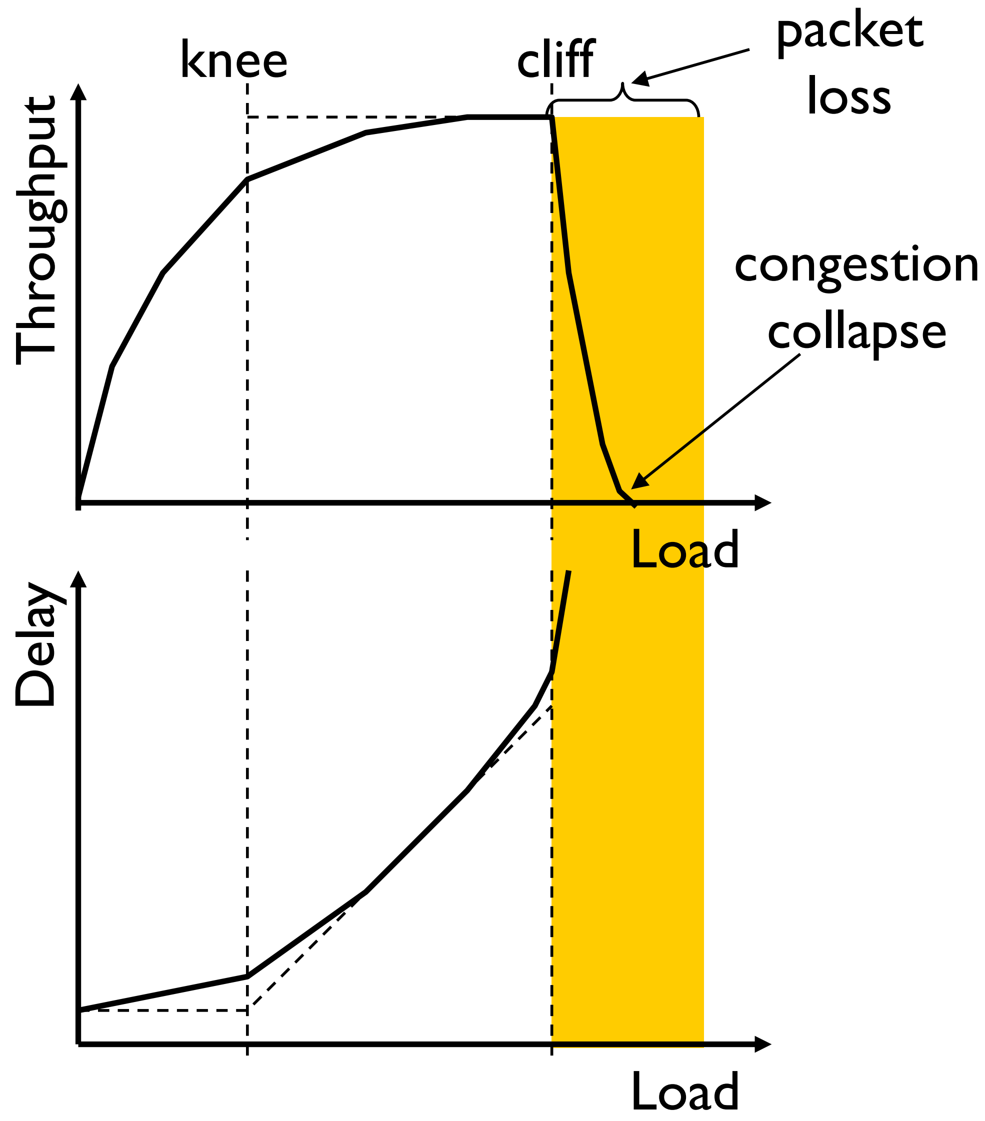
Without congestion control

congestion:

- Increases delivery latency
- Increases loss rate
- Increases retransmissions, many unnecessary
- Wastes capacity of traffic that is never delivered
- Increases congestion, cycle continues …

Cost of Congestion

- Knee – point after which
  - Throughput increases slowly
  - Delay increases fast
- Cliff – point after which
  - Throughput starts to drop to zero (congestion collapse)
  - Delay approaches infinity



Congestion Collapse

This happened to the Internet (then NSFnet) in 1986

- Rate dropped from a blazing 32 Kbps to 40bps
- This happened on and off for *two years*
- In 1988, Van Jacobson published "Congestion Avoidance and Control"
- The fix: senders voluntarily limit sending rate

Approaches towards congestion control

two broad approaches towards congestion control:

end-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

network-assisted congestion control:

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate for sender to send at

Transport Layer: Outline

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

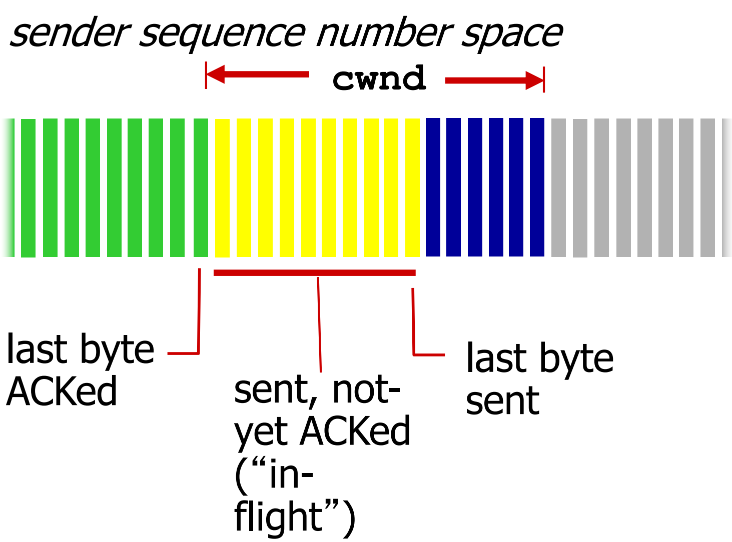
3.7 TCP congestion control

TCP's Approach in a Nutshell

- TCP connection has window
  - Controls number of packets in flight

- *TCP sending rate:*
  - *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$



- Vary window size to control sending rate

All These Windows…

- Congestion Window: CWND
  - How many bytes can be sent without overflowing routers
  - Computed by the sender using congestion control algorithm

- Flow control window: AdvertisedWindow (RWND)
  - How many bytes can be sent without overflowing receiver's buffers
  - Determined by the receiver and reported to the sender

- Sender-side window = minimum{CWND, RWND}
  - Assume for this lecture that RWND >> CWND

CWND

- This lecture will talk about CWND in units of MSS
  - (Recall MSS: Maximum Segment Size, the amount of payload data in a TCP packet)
  - This is only for pedagogical purposes

- Keep in mind that real implementations maintain CWND in bytes

Two Basic Questions

- How does the sender detect congestion?

- How does the sender adjust its sending rate?

Quiz: What is a “congestion event”

A: A segment loss (but how can the sender be sure of this?)

B: Increased delays

C: Receiving duplicate acknowledgement (s)

D: A retransmission timeout firing

D: Some subset of A, B, C & D (what is the subset?)

Quiz: How should we set CWND?

A: We should keep raising it until a “congestion event” then back off slightly until we notice no more events.

B: We should raise it until a “congestion event”, then go back to 1 and start raising it again

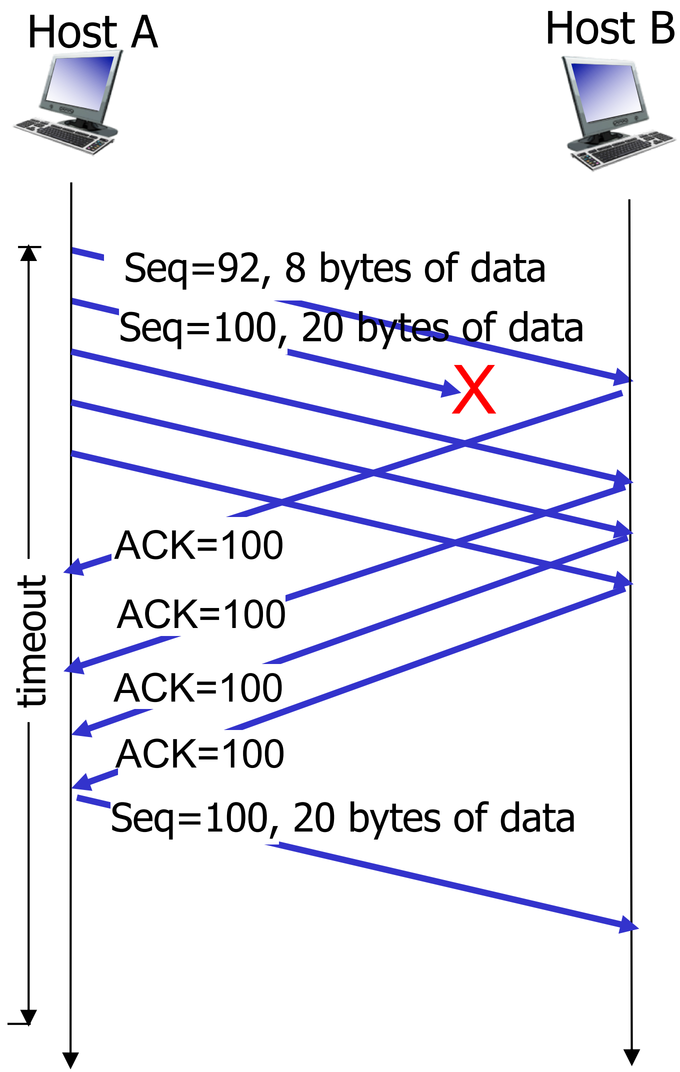
C: We should raise it until a “congestion event” then go back to median value and start raising it again.

D: We should sent as fast as possible at all times.

Not All Losses the Same

- Duplicate ACKs: isolated loss
  - dup ACKs indicate network capable of delivering some segments
- Timeout: much more serious
  - Not enough dup ACKs
  - Must have suffered several losses
- Will adjust rate differently for each case

RECAP: TCP fast retransmit



fast retransmit after sender receipt of triple duplicate ACK

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 *vs.*
  - Adjusting to bandwidth variations

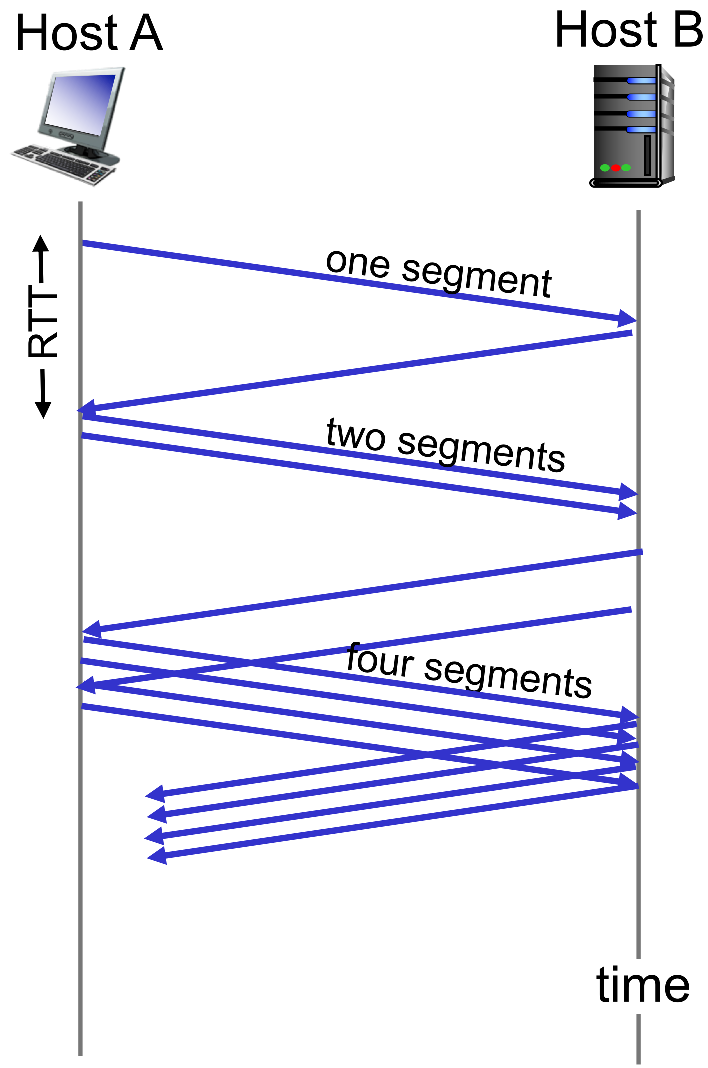
Bandwidth Discovery with Slow Start (SS)

- Goal: estimate available bandwidth
  - start slow (for safety)
  - but 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!

TCP Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - Simpler implementation achieved by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow but ramps up exponentially fast



Adjusting to Varying Bandwidth

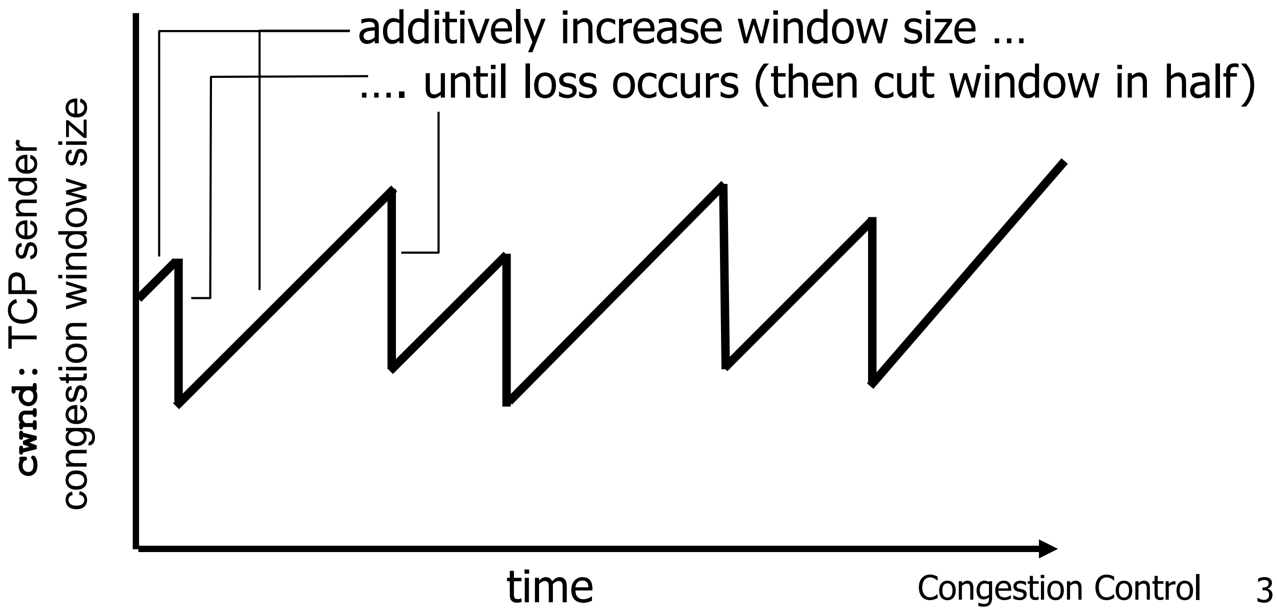
- Slow start gave an estimate of available bandwidth
- Now, want to track variations in this available bandwidth, oscillating around its current value
  - Repeated probing (rate increase) and backoff (rate decrease)
  - Known as Congestion Avoidance (CA)
- TCP uses: "Additive Increase Multiplicative Decrease" (AIMD)
  - We'll see why shortly…

AIMD

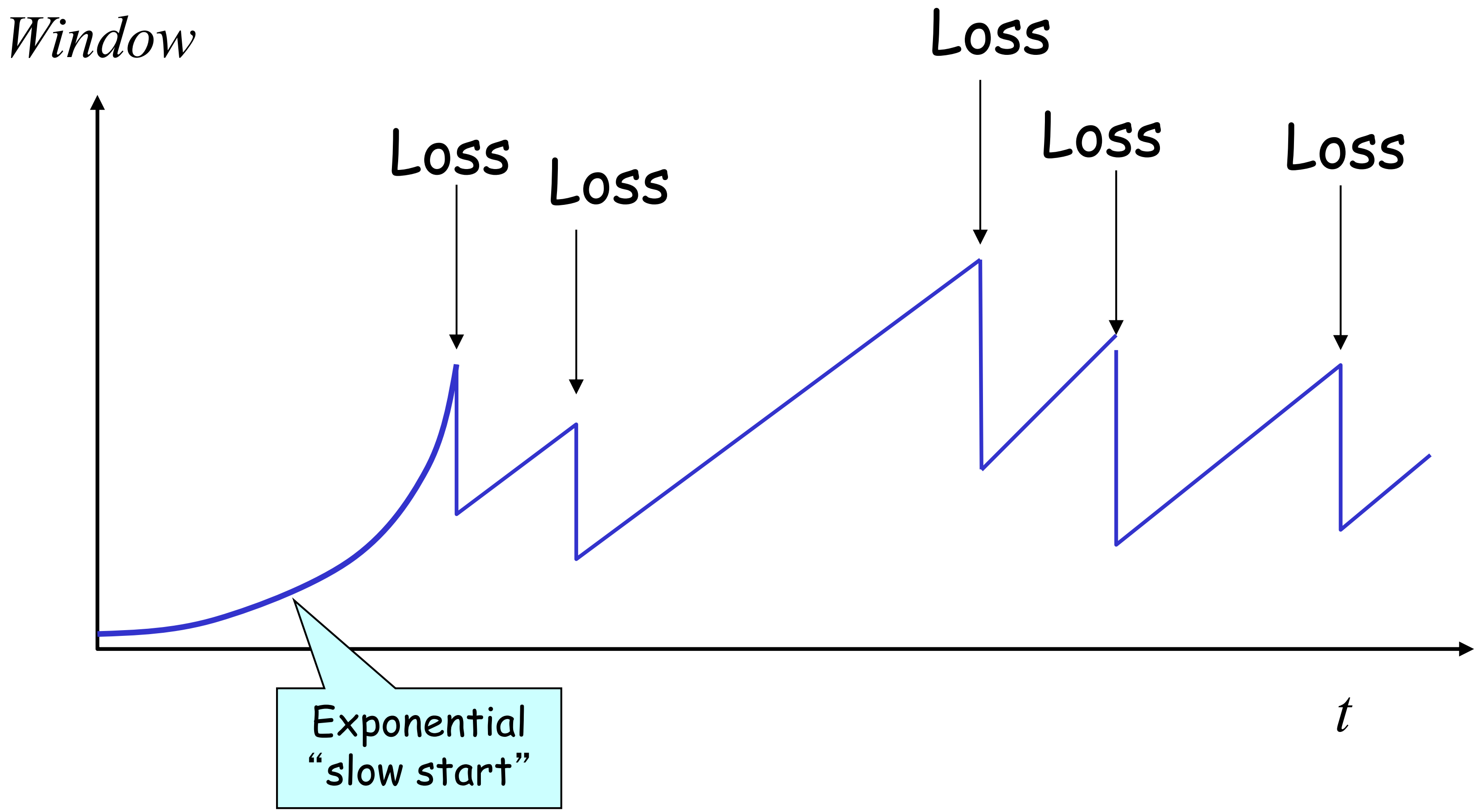
- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **`cwnd`** by 1 MSS every RTT until loss detected
  - For each successful RTT, **`cwnd = cwnd +1`**
  - Simple implementation: for each ACK, **`cwnd = cwnd + 1/cwnd`**

  - *multiplicative decrease:* cut **`cwnd`** in half after loss

AIMD saw tooth behavior: probing for bandwidth



Leads to the TCP "Sawtooth"



Slow-Start vs. AIMD

- When does a sender stop Slow-Start and start Additive Increase?
- Introduce a "slow start threshold" (ssthresh)
  - Initialized to a large value
  - On timeout/loss, ssthresh = CWND/2
- When CWND = ssthresh, sender switches from slow-start to AIMD-style increase

Implementation

- State at sender
  - CWND (initialized to a small constant)
  - ssthresh (initialized to a large constant, so initial slow start can learn network condition fast)
  - [Also dupACKcount and timer, as before]

- Events
  - ACK (new data)
  - dupACK (duplicate ACK for old data)
  - Timeout

Event: ACK (new data)

- If CWND < ssthresh
  - CWND += 1

> - *CWND packets per RTT*
> - *Hence after one RTT with no drops:*
>
>   *CWND = 2xCWND*

Event: ACK (new data)

- If CWND < ssthresh
  - CWND += 1

Slow start phase

- Else
  - CWND = CWND + 1/CWND

"Congestion Avoidance" phase (additive increase)

- *CWND packets per RTT*
- *Hence after one RTT with no drops:*
  CWND = CWND + 1

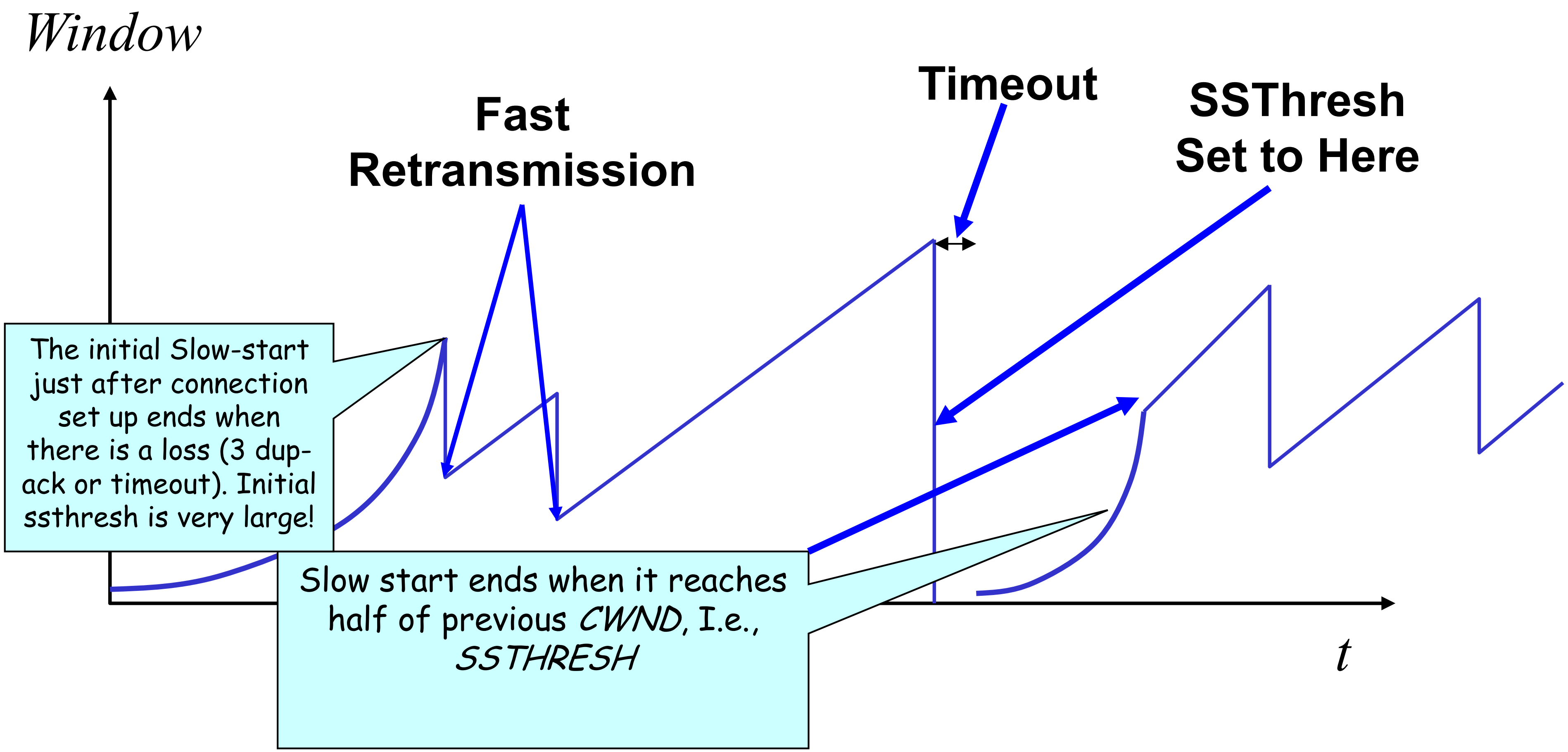
Event: dupACK

- dupACKcount ++
- If dupACKcount = 3 /* fast retransmit */
  - ssthresh = CWND/2
  - CWND = CWND/2

Event: TimeOut

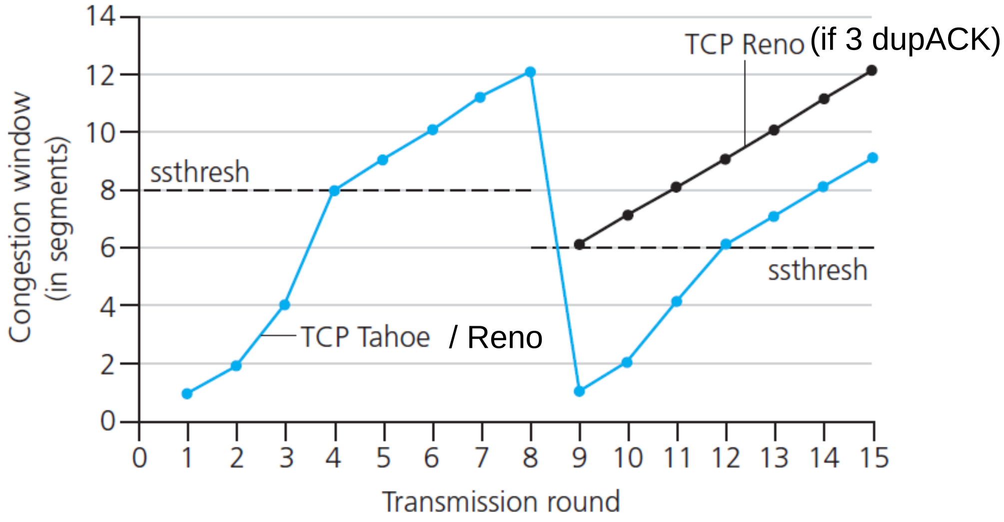
- On Timeout
  - ssthresh ← CWND/2
  - CWND ← 1

Example



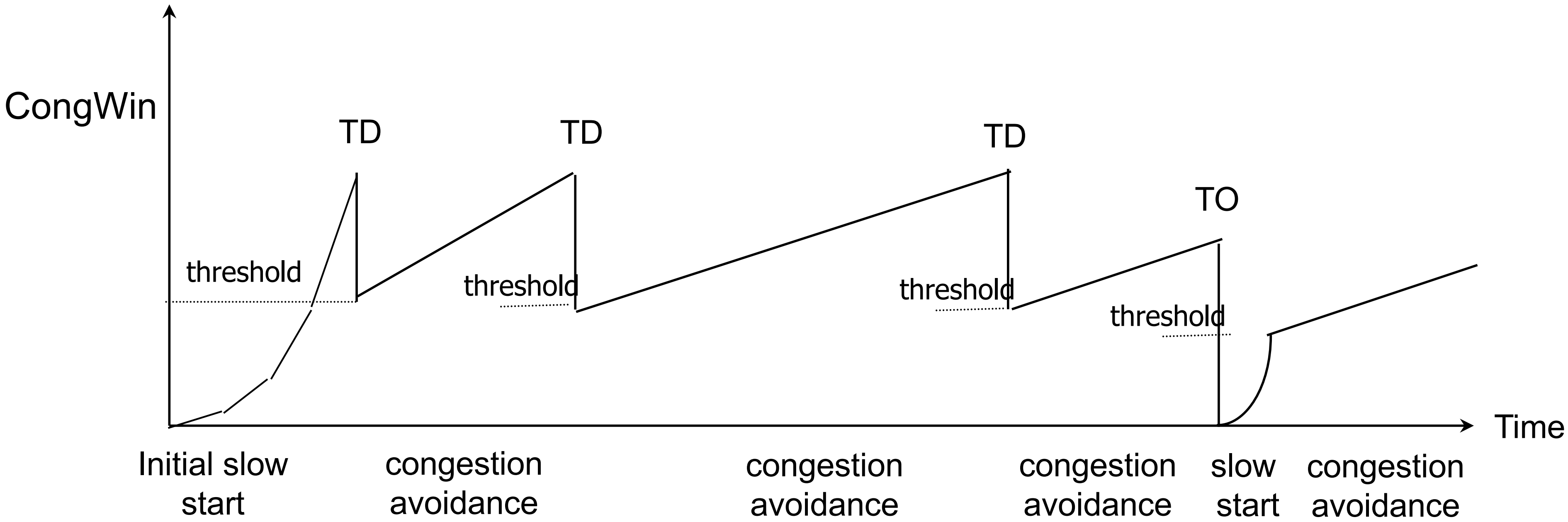
Slow-start restart: Go back to CWND = 1 MSS, but take advantage of knowing the previous value of CWND

TCP Flavours



- TCP-Reno (Assumed Default in this course)
  - cwnd =1 on timeout
  - cwnd = cwnd/2 on triple dup ACK
- TCP-Tahoe (Old/original version)
  - cwnd =1 on both triple dup ACK & timeout
- Figure 3.52, page 304 of 7th Ed. textbook assumes a special TCP Reno that implements Fast Recovery, which is out of scope in this course.

TCP/Reno (Default in this course): Big Picture



TD: Triple duplicate acknowledgements
TO: Timeout

Transport Layer: Summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP

next:

- leaving the network "edge" (application, transport layers)
- into the network "core"