15-441 15-641 Computer Networking

Congestion Control

Peter Steenkiste

Fall 2015

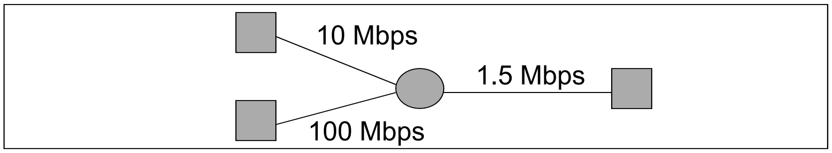
www.cs.cmu.edu/~prs/15-441-F15

Outline

- Congestion control fundamentals
  - Challenges
  - Basic mechanisms
- TCP congestion control
- TCP slow start

2

Congestion



- Many sources “share*” resources inside network
- Problem: demand can exceed capacity of the network
  - Sources are unaware of current state of resource
  - Sources are unaware of each other
- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Long delays (queuing in router buffers)
- Challenge:

  How do we coordinate all nodes in the Internet?

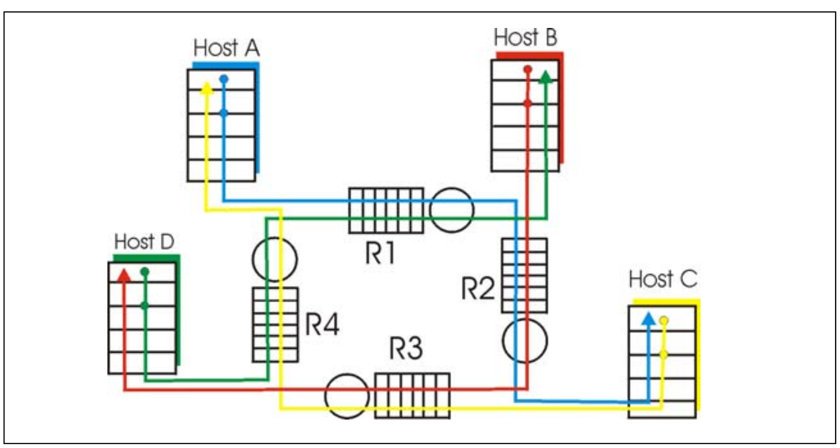
* Share → Compete for?

3

Causes & Costs of Congestion

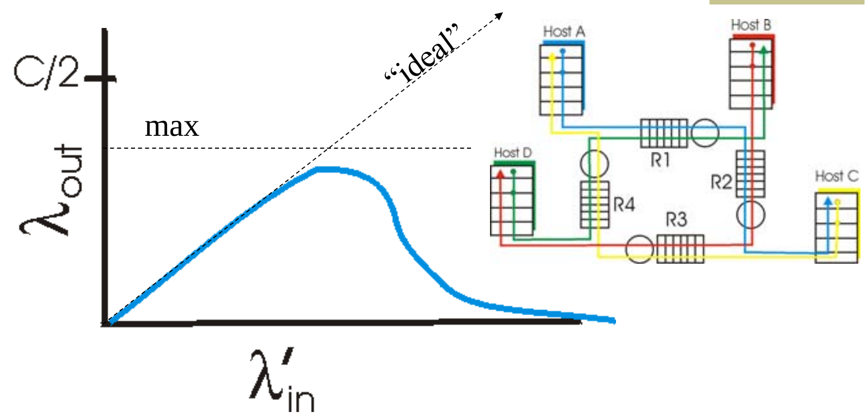
- Four senders – multihop paths
- Timeout/retransmit

Q: What happens as rate increases?



4

Causes & Costs of Congestion



- When packet dropped, any "upstream transmission capacity used for that packet was wasted!

5

Congestion Collapse

- Definition: *Increase in network load results in decrease of useful work done*
- Many possible causes
  - Spurious retransmissions of packets still in flight
    - Classical congestion collapse
    - How can this happen with packet conservation
    - Solution: better timers and TCP congestion control
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Solution: congestion control for ALL traffic

6

Plan for Today

- So far we considered two networks
  - Network 1: 1 router, 3 links
  - Network 2: 4 routers, 8 links
- Next step: how do we deal with congestion in the Internet
  - Millions of routers
  - Even more links
  - 100s of millions of sources

Outline

- Congestion control fundamentals
  - Challenges
  - Basic mechanisms
- TCP congestion control
- TCP slow start

Congestion Control Goals

- A mechanism that:
  - Uses network resources efficiently
  - Prevents or avoids collapse
  - Preserves fair network resource allocation
- Congestion collapse is not just a theory
  - Has been frequently observed in many networks

9

Two Approaches Towards Congestion Control

End-to-end congestion control:

- No explicit feedback from network
- End-systems infer congestion status from observed loss, delay, …
- Approach taken by TCP
- Problem: making it work
  - Avoid significant packet loss
  - Maintain high utilization

Network-assisted congestion control:

- Routers provide feedback to end systems
  - Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - Explicit rate sender should send at (ATM)
- Problem: makes routers more complicated
  - Per-flow state → poor scalability
  - Can sometimes be avoided

10

Congestion Control with Binary Feedback (TCP)

- Very simple mechanisms in network
  - FIFO scheduling with shared buffer pool
  - Feedback through packet drops (or binary feedback)
- TCP interprets packet drops as signs of congestion and sender slows down
  - This is an assumption: packet drops are not a sign of congestion in all networks, e.g., wireless networks
- Sender periodically probes the network to check whether more bandwidth has become available
- Key questions: how much to reduce (after a drop) and increase (when probing) rate

Objectives

- Simple router behavior
- Distributedness
- Efficiency: $X = \Sigma x_i(t)$
- Fairness: $(\Sigma x_i)^2/n(\Sigma x_i^2)$
  - What are the important properties of this function?
- Convergence: control system must be stable

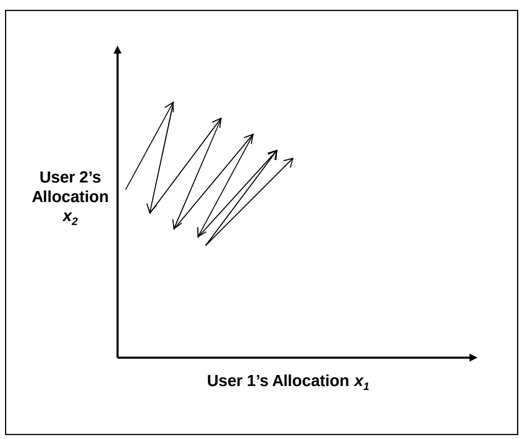
Linear Control

- Many different possibilities for reaction to congestion and probing
  - Examine simple linear controls
    - Window(t + 1) = a + b Window(t)
    - Different a_i/b_i for increase and a_d/b_d for decrease
- Supports various reaction to signals
  - Increase/decrease additively
  - Increased/decrease multiplicatively
  - Which of the four combinations is optimal?

13

Phase Plots

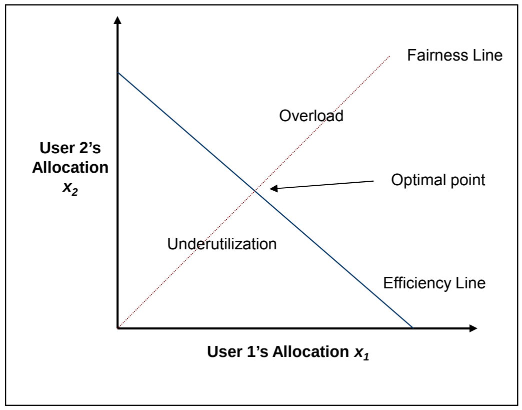
- Simple way to visualize behavior of competing connections over time
- Sequence of steps with 2 synchronized senders



14

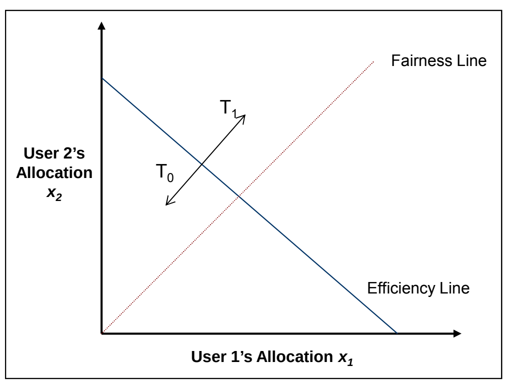
Phase Plots

- What are desirable properties?
- What if flows are not equal?



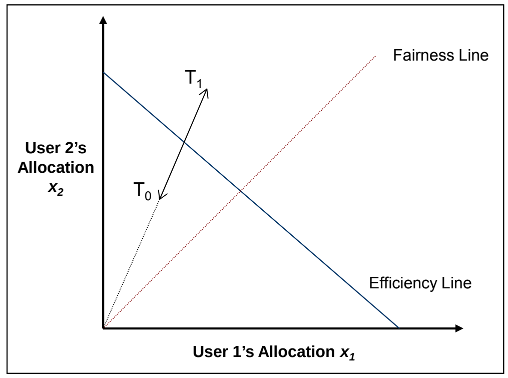
Additive Increase/Decrease

- Both X_1 and X_2 increase/ decrease by the same amount over time
- Additive increase improves fairness and additive decrease reduces fairness

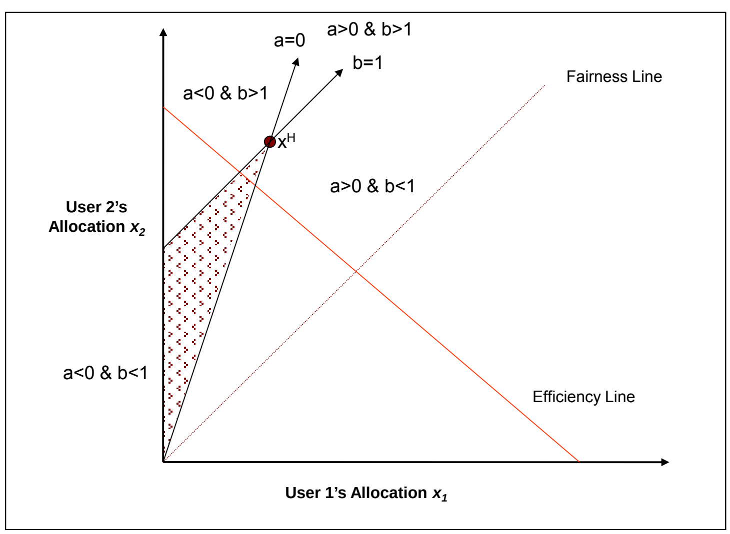


Muliplicative Increase/Decrease

- Both X_1 and X_2 increase by the same factor over time
  - Extension along line through origin
- Constant fairness

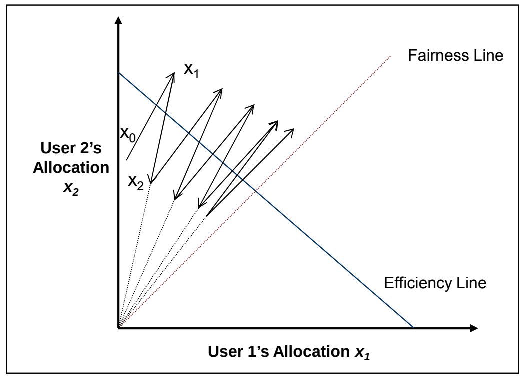


Achieving Fairness AND Efficiency



What is the Right Choice?

- Constraints limit us to AIMD
  - Can have multiplicative term in increase (MAIMD)
  - AIMD moves towards optimal point

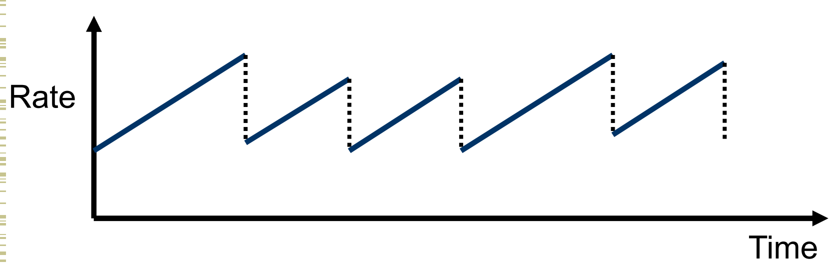


Outline

- Congestion control fundamentals
- TCP congestion control
  - Implementing AIMD
  - Packet pacing
  - Fast recovery
- TCP slow start

TCP Congestion Control: Implicit Feedback and AIMD

- Distributed, fair and efficient
- Packet loss is seen as sign of congestion and results in a multiplicative rate decrease: factor of 2
- TCP periodically probes for available bandwidth by increasing its rate: by one packet per RTT



Implementation Issue

- Operating system timers are very coarse – how to pace packets out smoothly?
- Implemented using a congestion window that limits how much data can be in the network.
  - Similar to using a flow control window to avoid flooding receiver
  - TCP also keeps track of how much data is in transit
- Data can only be sent when the amount of outstanding data is less than the congestion window.
  - The amount of outstanding data is increased on a “send” and decreased on “ack”
  - (last sent – last acked) < congestion window
- Window limited by both congestion and buffering
  - Sender’s maximum window = Min (advertised window, cwnd)

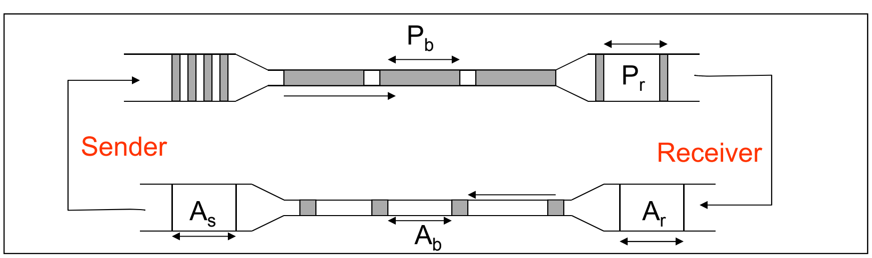
Packet Conservation

- At equilibrium, inject packet into network only when one is removed
  - Controlled by sliding window, not rate
  - But still need to avoid sending burst of packets → would overflow links
    - Need to carefully pace out packets
    - Helps provide stability
- Need to eliminate spurious retransmissions
  - Accurate RTO estimation
  - Better loss recovery techniques (e.g. fast retransmit)

23

TCP Packet Pacing

- Congestion window helps to “pace” the transmission of data packets
- In steady state, a packet is sent when an ack is received
  - Data transmission remains smooth, once it is smooth
  - Self-clocking behavior



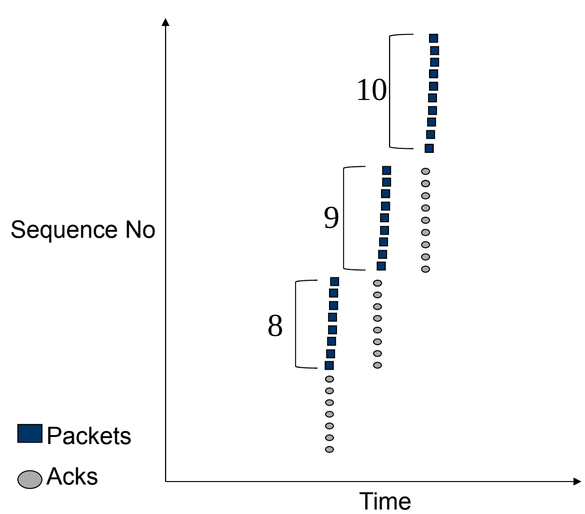
24

Congestion Avoidance

- If loss occurs when cwnd = W
  - Network can handle 0.5W ~ W segments
  - Set cwnd to 0.5W (multiplicative decrease)
- Upon receiving ACK
  - Increase cwnd by (1 packet)/cwnd
    - What is 1 packet? → 1 MSS worth of bytes
    - After cwnd packets have passed by → approximately increase of 1 MSS
- Implements AIMD

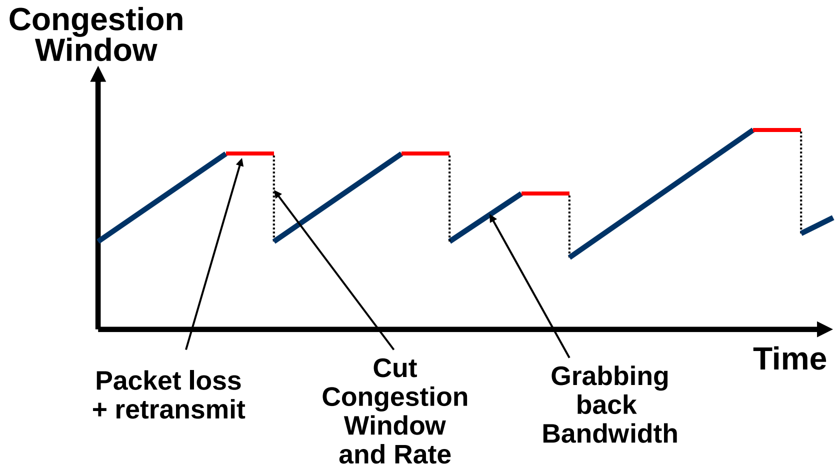
25

Congestion Avoidance Sequence Plot Pacing and “AI”

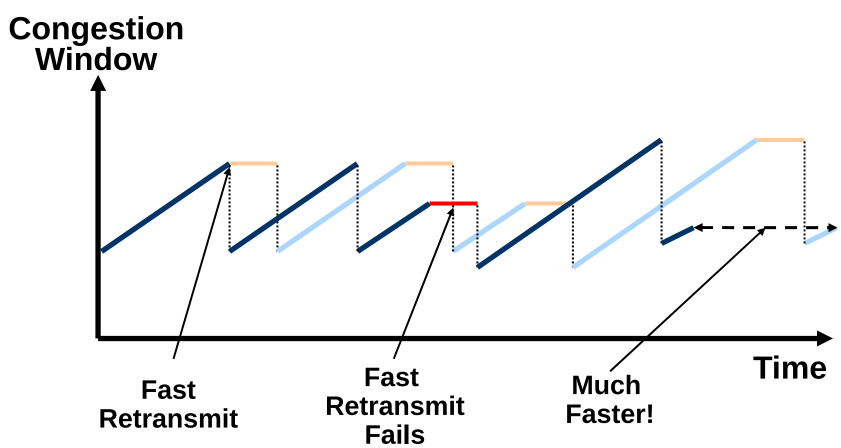


26

Congestion Avoidance Behavior



Remember Fast Retransmit?

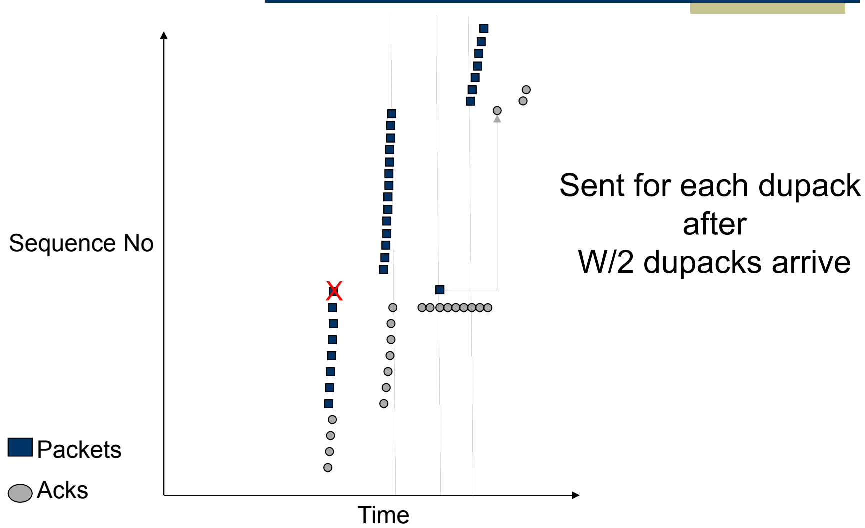


Fast Recovery

- With fast retransmit, TCP can often avoid timeout, but loss signals congestion → cut window in half
- Challenge: how do we maintain ack clocking?
- Observation: each duplicate ack notifies sender that a single packet has cleared the network
- When < **new** cwnd packets are outstanding
  - Allow new packets out with each new duplicate acknowledgement
- Behavior
  - Sender is idle for some time – waiting for ½ cwnd worth of dupacks
  - Transmits at original rate after wait with ack clocking

29

Fast Recovery



30

Outline

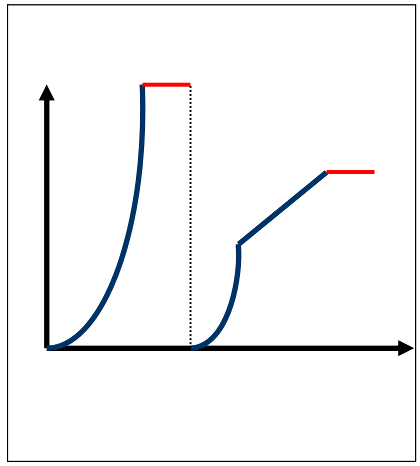
- TCP connection setup/data transfer
- TCP congestion avoidance
- TCP slow start

Reaching Steady State

- Doing AIMD is fine in steady state but how do we get started …
- How does TCP know what is a good initial rate to start with?
  - Should work both for a CDPD (10s of Kbps or less) and for supercomputer links (10 Gbps and growing)
  - Need quick initial phase to help TCP get up to speed
- Also, after a timeout, the “pipe has drained”
  - cwnd = 0.5 * cwnd
  - How do we restart ACK clocking?

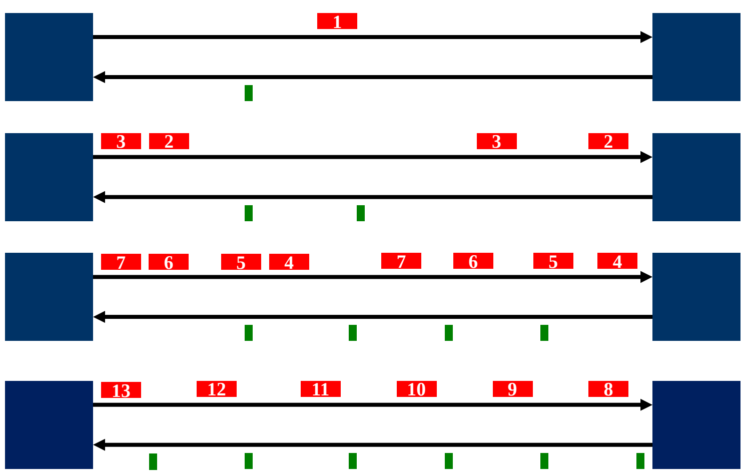
Slow Start Packet Pacing

- How do we get this clocking behavior to start?
  - Initialize cwnd = 1
  - Upon receipt of every ack, cwnd = cwnd + 1
  - Packet loss means you are going too fast
    - Hopefully Fast Retransmit works!
- Allows TCP to quickly find a good window size
  - Exponential increase!
  - Reaches W in RTT * $\log_2(W)$
  - Also starts packet pacing
- How is this slow?



33

Starting of Packet Pacing



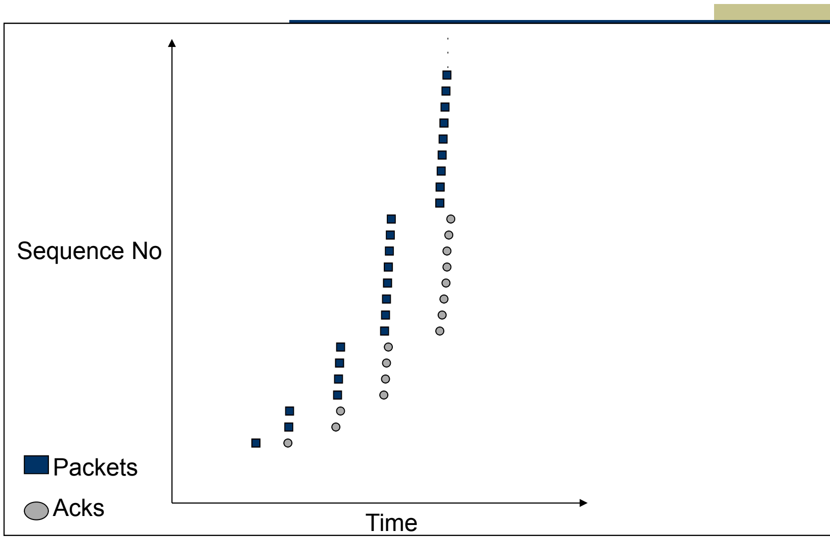
CC window increases by 1 packet/ACK

Queuing separates packet pair

This repeats each RTT

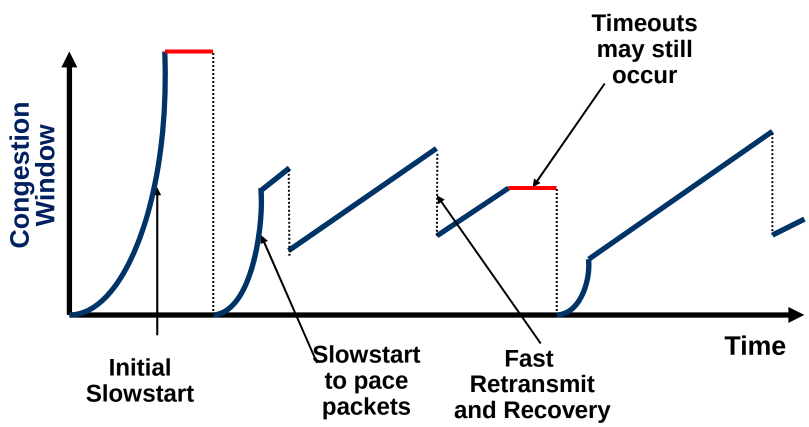
Until pipe is full

Slow Start Sequence Plot



35

TCP Sawtooth Behavior



36

Important Lessons

- TCP state diagram → setup/teardown
- TCP timeout calculation → how is RTT estimated
- Modern TCP loss recovery
  - Why are timeouts bad?
  - How to avoid them? → e.g. fast retransmit

57