Recap: Bandwidth allocation

- Want efficiency and fairness
  - The two can conflict
- Strict fairness not always the goal
- Max-min fairness is one ideal

- Today
  - Achieving bandwidth allocation in the Internet

Why is Bandwidth Allocation hard?

- Number of senders and their offered load changes
- Senders may be limited in other ways
  - Other parts of network or by applications
- Network is distributed; no single party has an overall picture of its state

Bandwidth Allocation Solution Context

In networks without admission control (e.g., Internet) Transport and Network layers must work together

- Network layer sees congestion
  - Only it can provide direct feedback
- Transport layer causes congestion
  - Only it can reduce load

Bandwidth Allocation Solution Overview

- Senders adapt concurrently based on their own view of the network
- Design this adaptation so the network usage as a whole is efficient and fair
  - In practice, efficiency is more important than fairness
- Adaptation is continuous since offered loads continue to change over time

Bandwidth Allocation Models

- Open loop versus closed loop
  - Open: reserve bandwidth before use
  - Closed: use feedback to adjust rates
- Host versus Network support
  - Who is sets/enforces allocations?
- Window versus Rate based
  - How is allocation expressed?

TCP is a closed loop, host-driven, and window-based

Bandwidth Allocation Models (2)

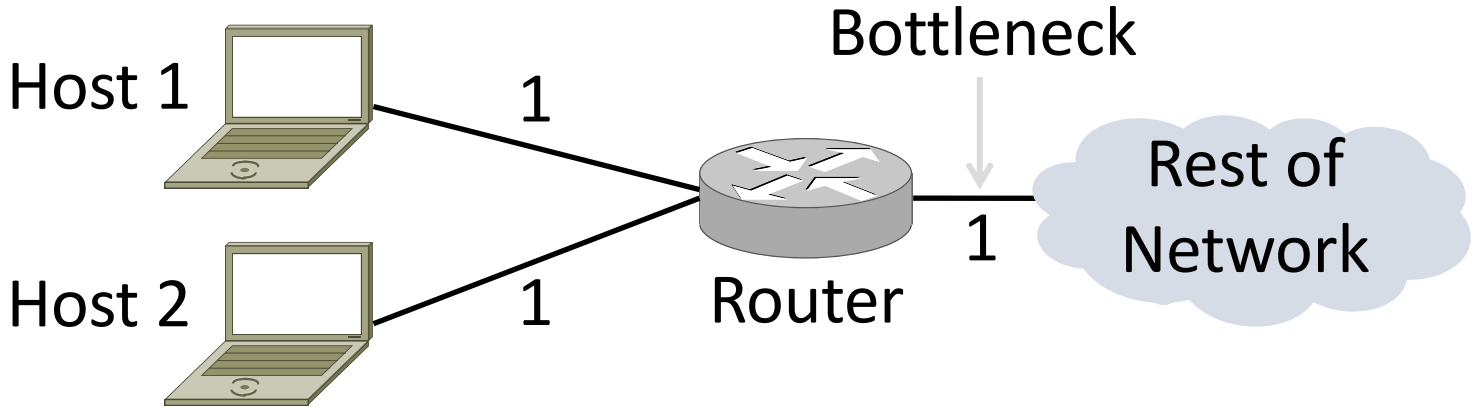
- We'll study closed-loop, host-driven, and window-based too
- Network layer returns feedback on current allocation to senders
  - At least tells if there is congestion
- Transport layer adjusts sender's behavior via window in response
  - How senders adapt is a control law

Additive Increase Multiplicative Decrease

- AIMD is a control law hosts can use to reach a good allocation
  - Hosts additively increase rate while network not congested
  - Hosts multiplicatively decrease rate when congested
  - Used by TCP
- Let's explore the AIMD game …

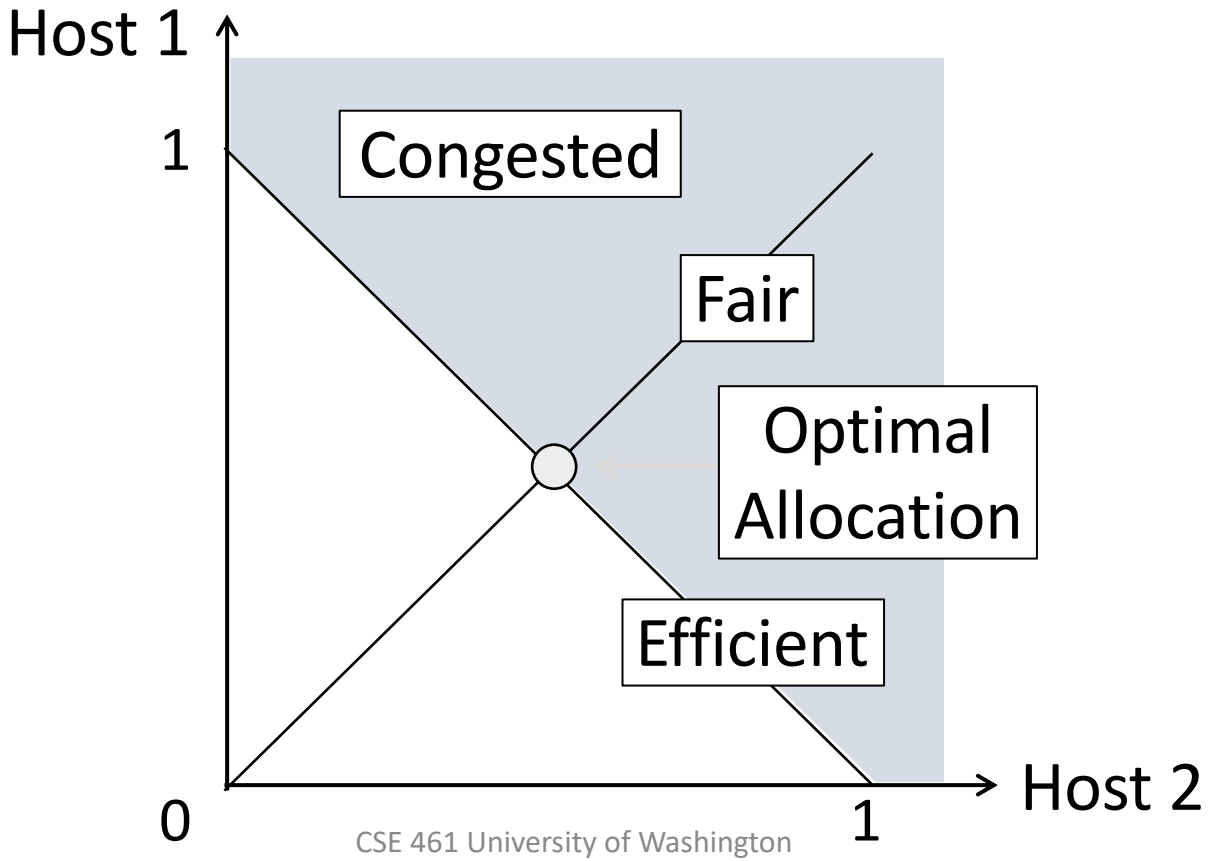
AIMD Game

- Hosts 1 and 2 share a bottleneck
  - But do not talk to each other directly
- Router provides binary feedback
  - Tells hosts if network is congested



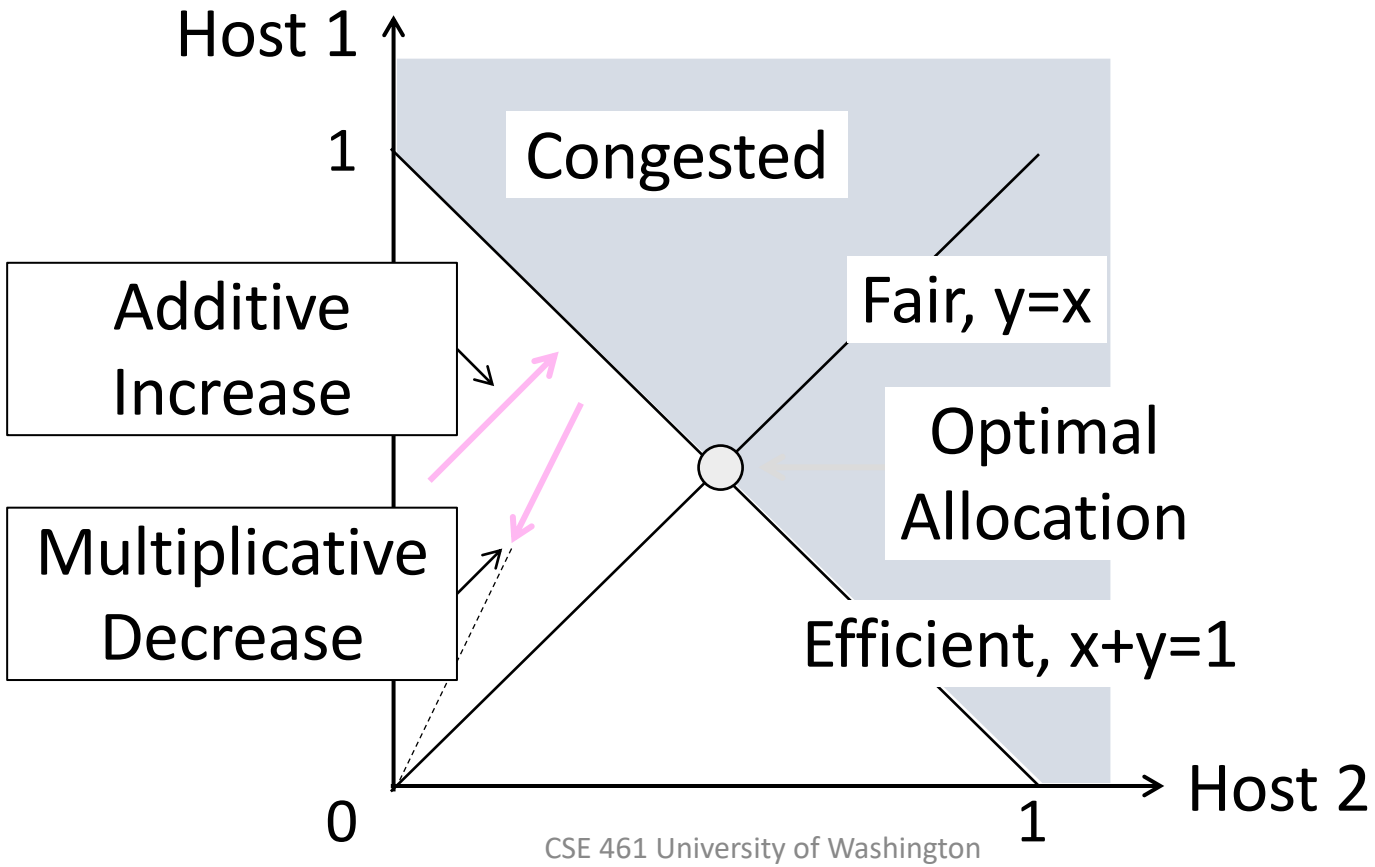
AIMD Game (2)

- Each point is a possible allocation



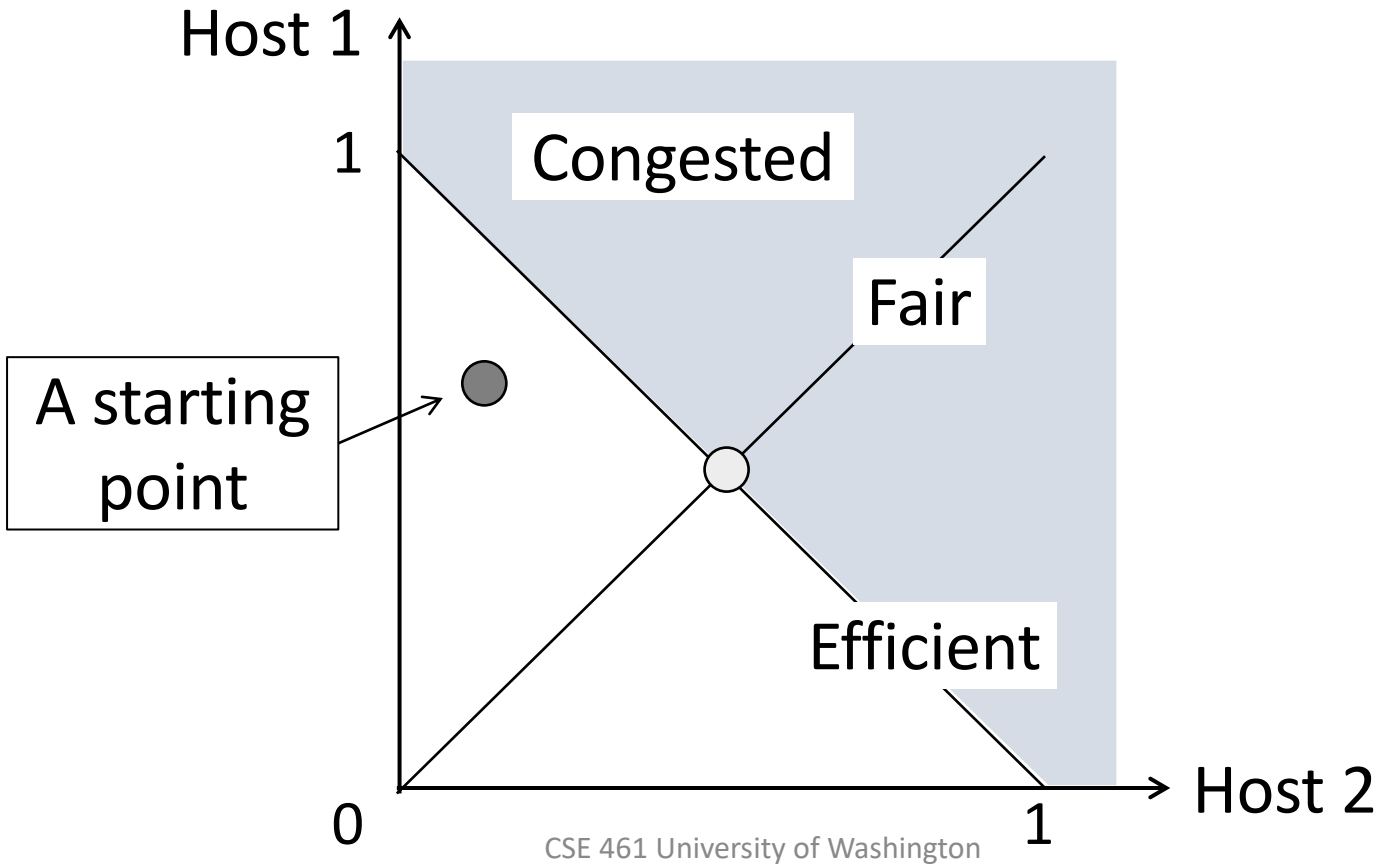
AIMD Game (3)

- AI and MD move the allocation



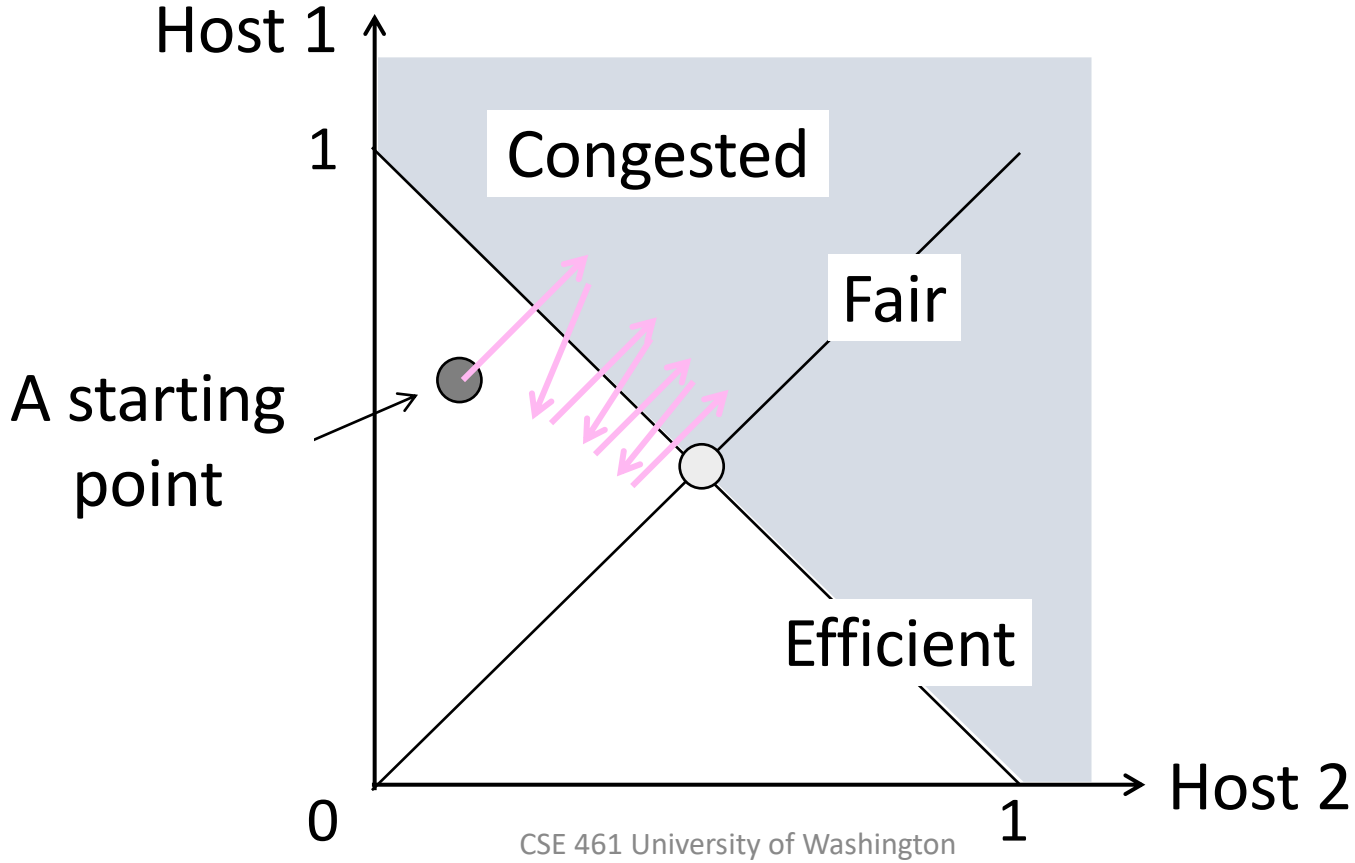
AIMD Game (4)

- Play the game!



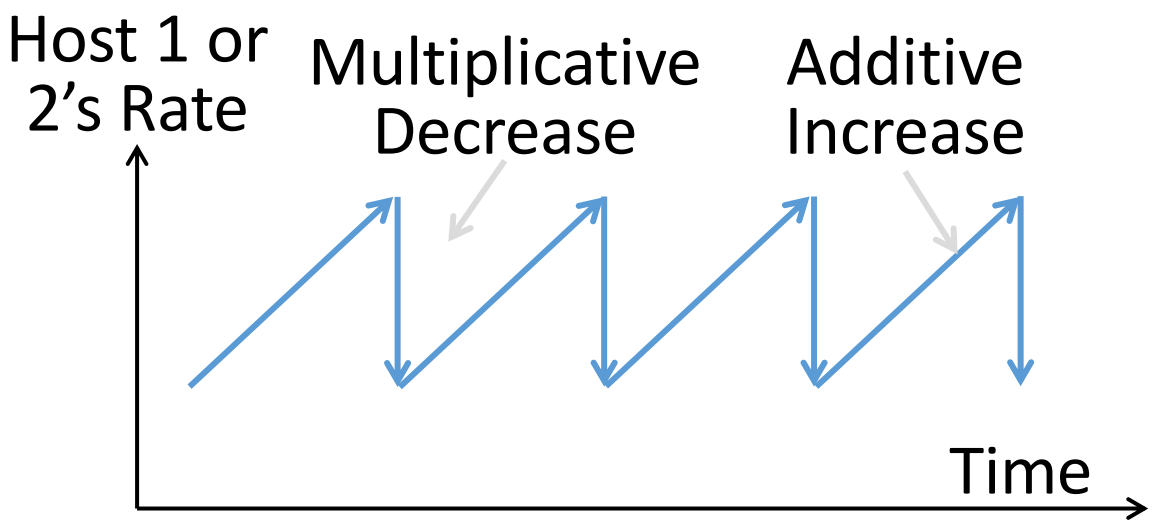
AIMD Game (5)

- Always converge to good allocation!



AIMD Sawtooth

- Produces a "sawtooth" pattern over time for rate of each host
  - This is the TCP sawtooth (later)



AIMD Properties

- Converges to an allocation that is efficient and fair when hosts run it
  - Holds for more general topologies
- Other increase/decrease control laws do not! (Try MIAD, MIMD, MIAD)
- Requires only binary feedback from the network

Feedback Signals

- Several possible signals, with different pros/cons
  - We'll look at classic TCP that uses packet loss as a signal

Signal	Example Protocol	Pros / Cons
Packet loss	TCP NewReno Cubic TCP (Linux)	Hard to get wrong Hear about congestion late Other events can cause loss
Packet delay	BBR (Google)	Hear about congestion early Need to infer congestion
Router indication	TCPs with Explicit Congestion Notification	Hear about congestion early Require router support

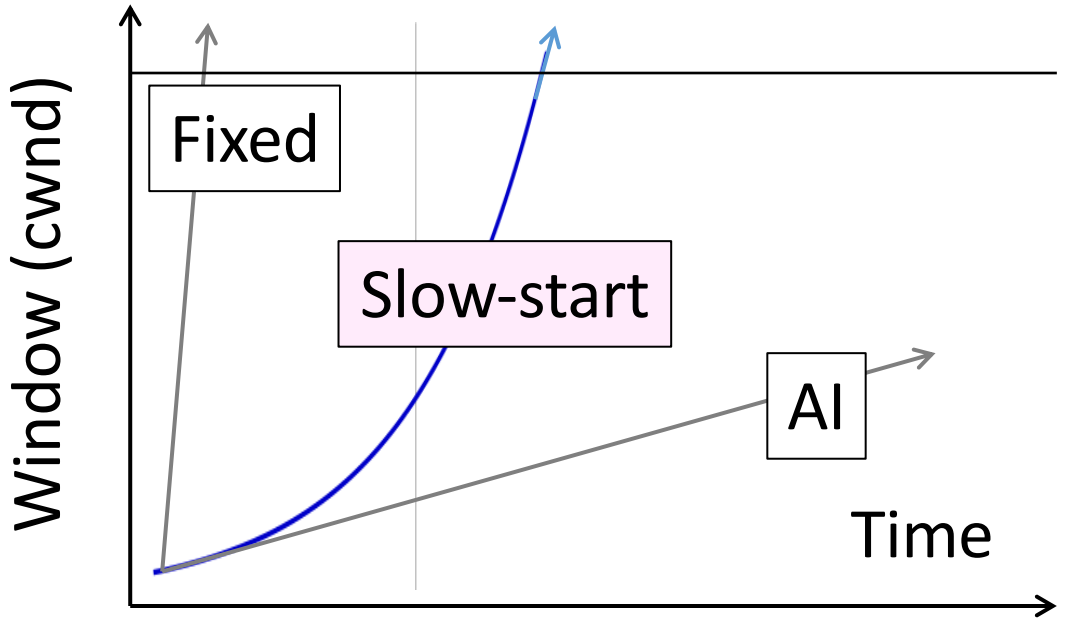
Slow Start (TCP Additive Increase)

TCP “Slow Start” Problem

- We want to quickly near the right rate, $cwnd_{IDEAL}$, but it varies greatly
  - Fixed sliding window doesn’t adapt and is rough on the network (loss!)
  - Additive Increase with small bursts adapts cwnd gently, but might take a long time to become efficient

Slow-Start Solution

- Start by doubling cwnd every RTT
  - Exponential growth (1, 2, 4, 8, 16, …)
  - Start slow, quickly reach large values

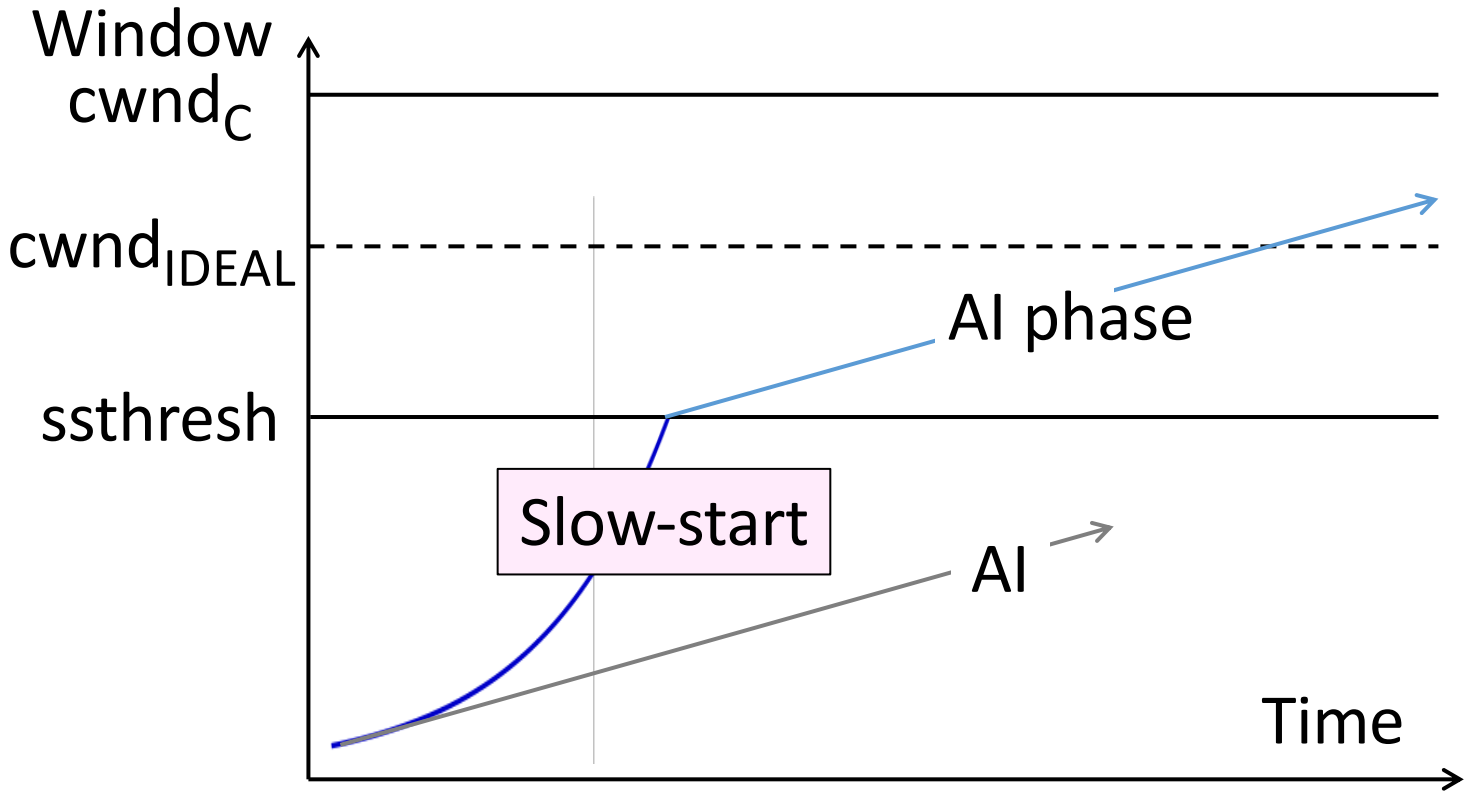


Slow-Start Solution (2)

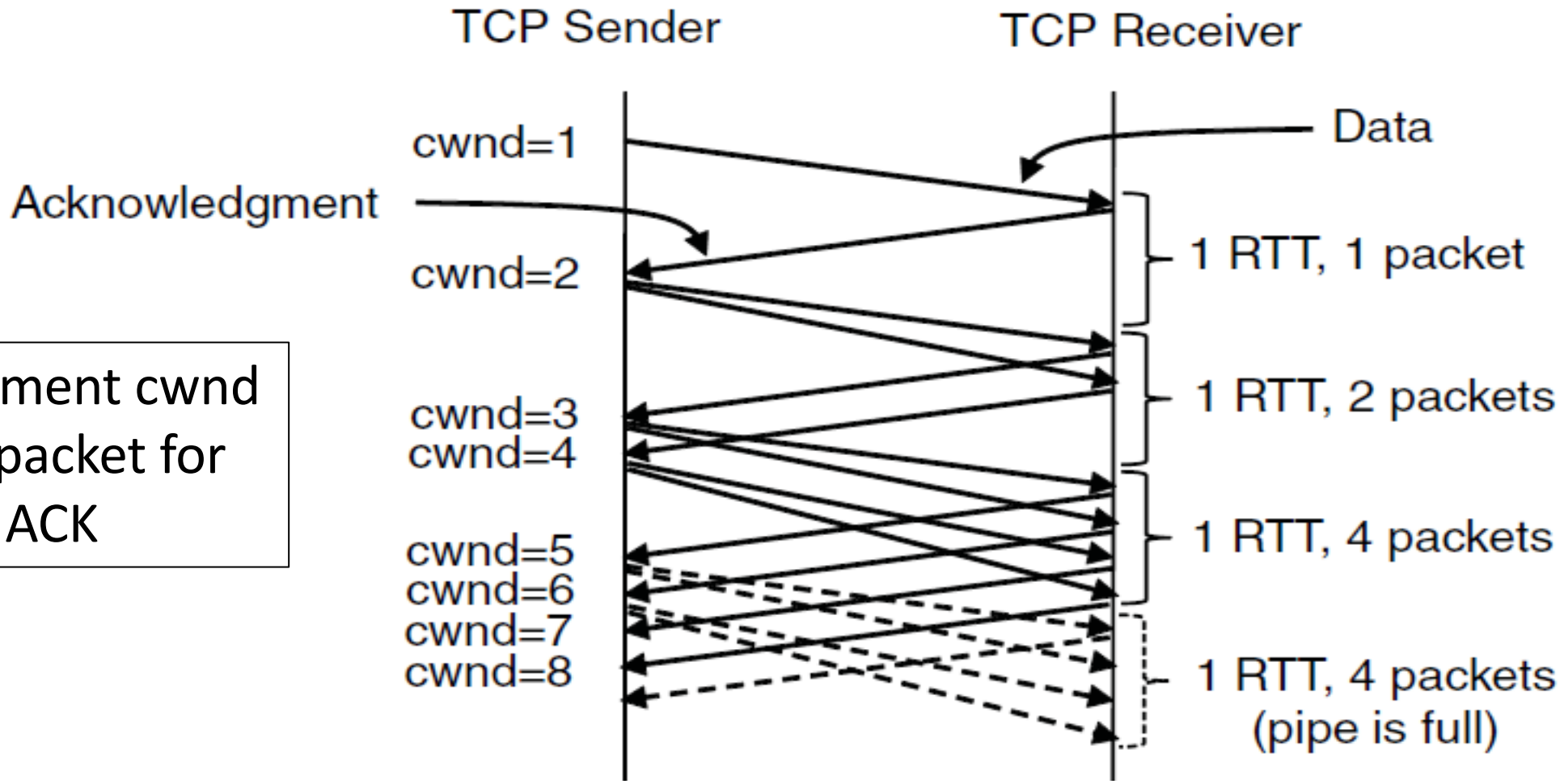
- Eventually packet loss will occur when the network is congested
  - Loss timeout tells us cwnd is too large
  - Next time, switch to AI beforehand
  - Slowly adapt cwnd near right value
- In terms of cwnd:
  - Expect loss for $cwnd_C \approx 2BD+queue$
  - Use ssthresh = $cwnd_C/2$ to switch to AI

Slow-Start Solution (3)

- Combined behavior, **after first time**
  - Most time spent near right value

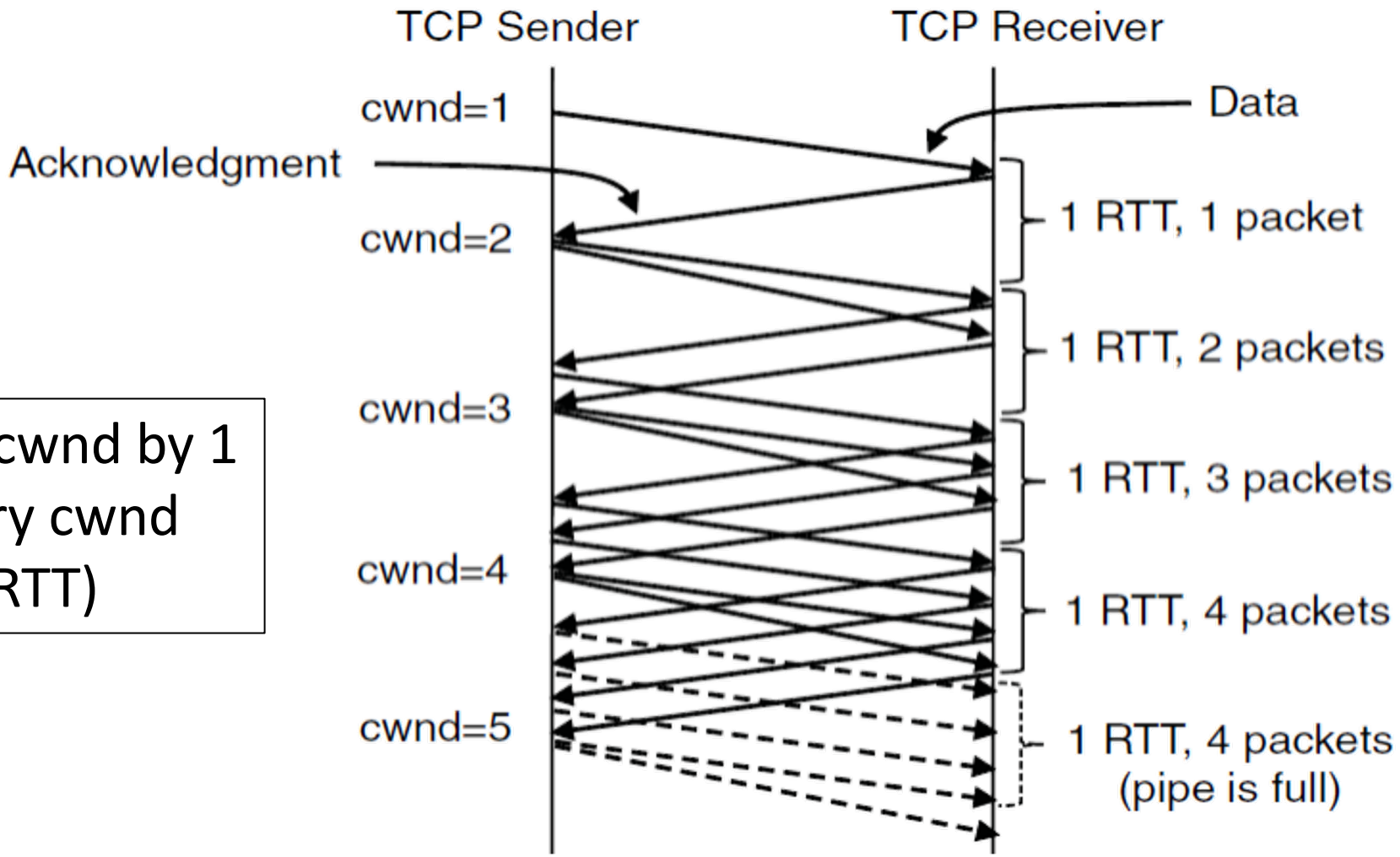


Slow-Start (Doubling) Timeline



Increment cwnd by 1 packet for each ACK

Additive Increase Timeline



Increment cwnd by 1 packet every cwnd ACKs (or 1 RTT)

TCP Tahoe (Implementation)

- Initial slow-start (doubling) phase
  - Start with cwnd = 1 (or small value)
  - cwnd += 1 packet per ACK
- Later Additive Increase phase
  - cwnd += 1/cwnd packets per ACK
  - Roughly adds 1 packet per RTT
- Switching threshold (initially infinity)
  - Switch to AI when cwnd > ssthresh
  - Set ssthresh = cwnd/2 after loss
  - Begin with slow-start after timeout