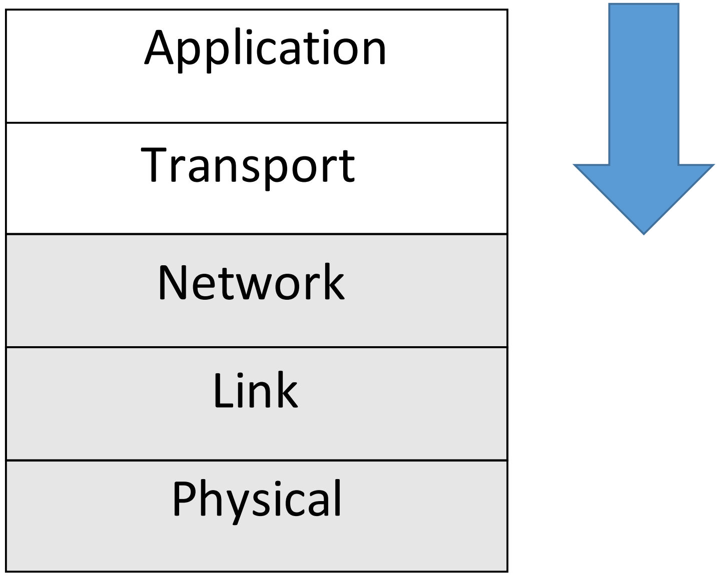
Transport Layer
(Congestion Control)

Where we are in the Course

- Still at the Transport Layer
  - Some of the functionality discussed spills into Network layer



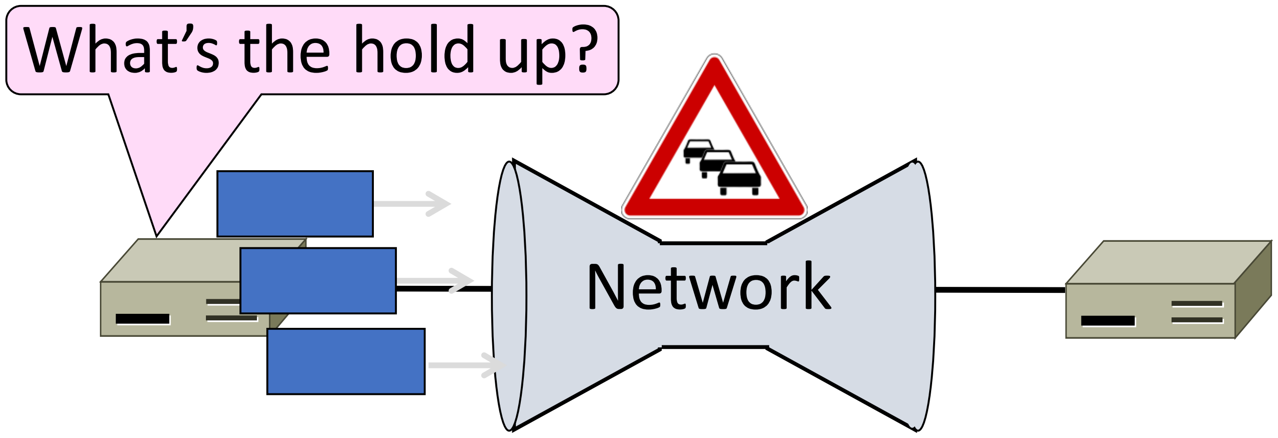
TCP to date:

- We can set up a connection (connection establishment)
- Tear down a connection (connection release)
- Keep the sending and receiving buffers from overflowing (flow control)

What's missing?

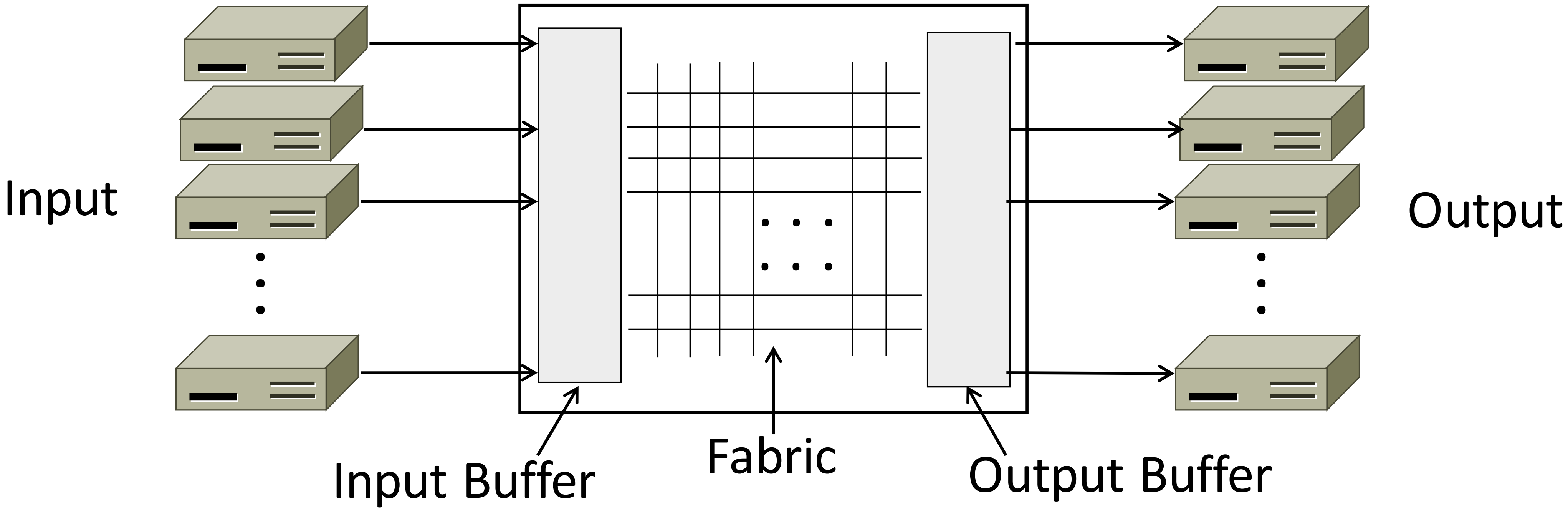
Network Congestion

- A "traffic jam" in the network
  - Later we will learn how to control it



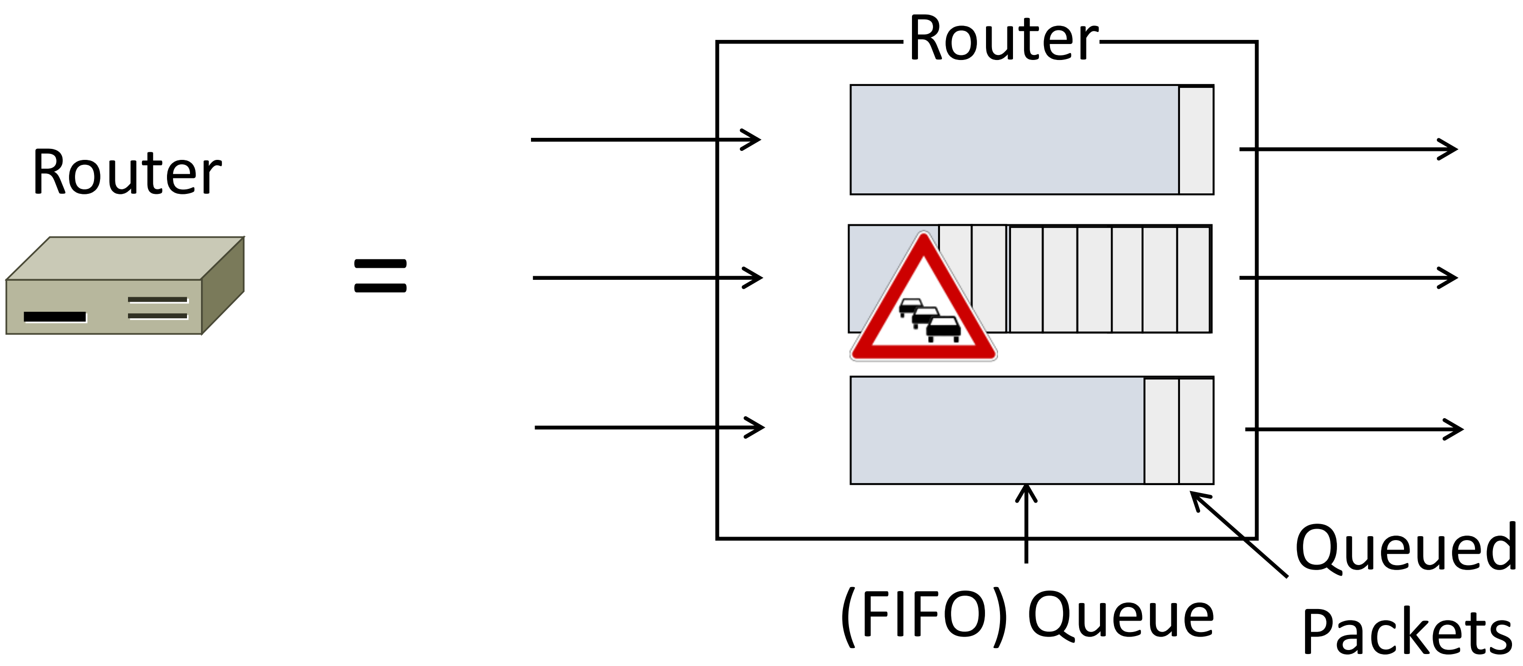
Nature of Congestion

- Routers/switches have internal buffering



Nature of Congestion

- Simplified view of per port output queues
  - Typically FIFO (First In First Out), discard when full

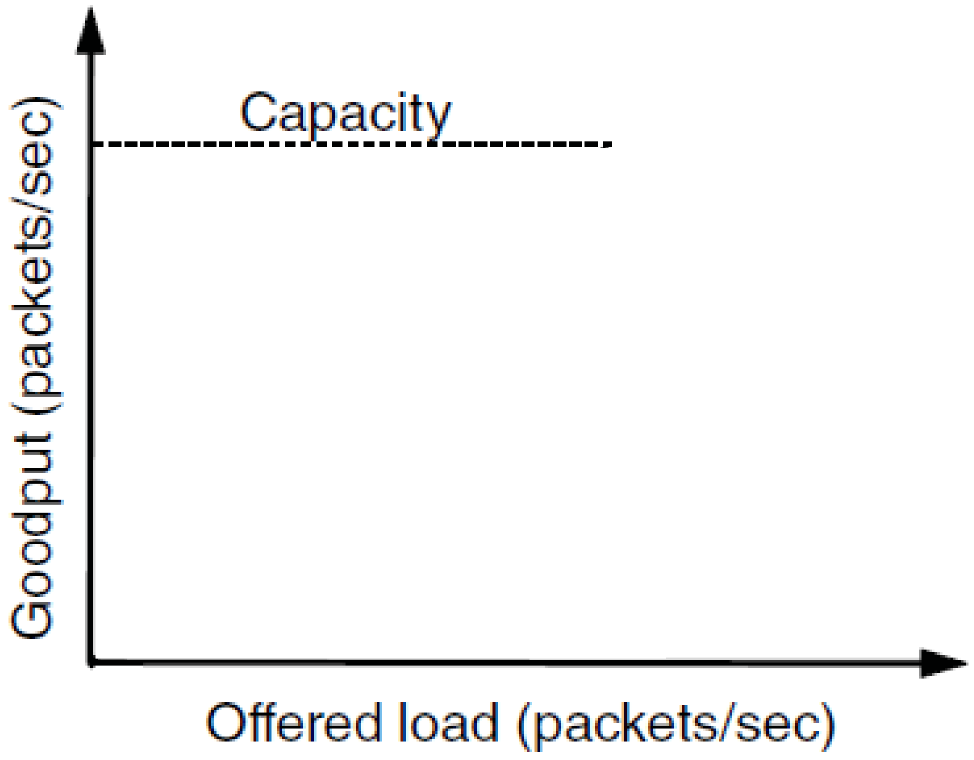


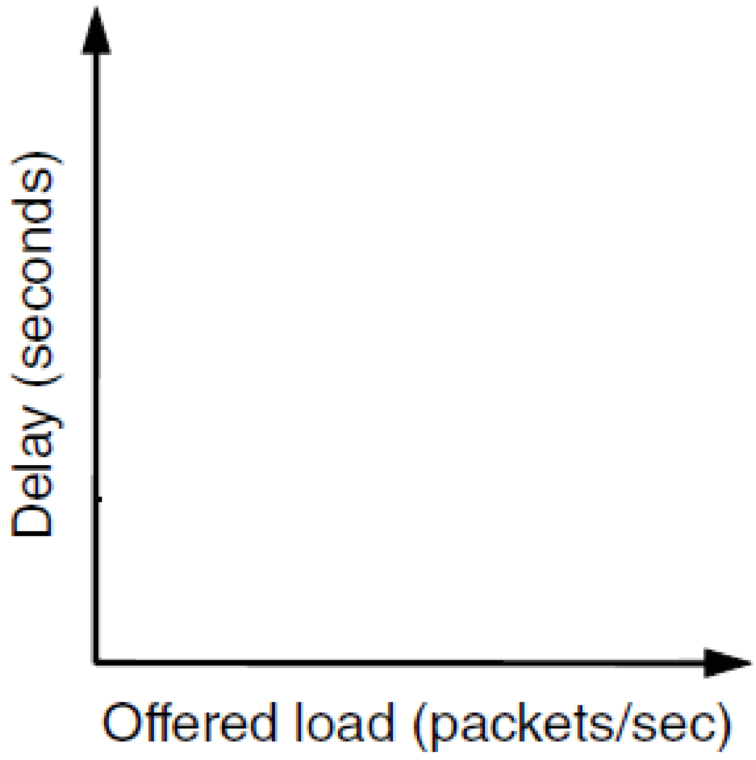
Nature of Congestion

- Queues help by absorbing bursts when input > output rate
- But if input > output rate for long enough, queue will overflow
  - This is congestion
- Congestion is a function of the traffic patterns and topology
  - can occur even if every link has the same capacity

Effects of Congestion - Measures

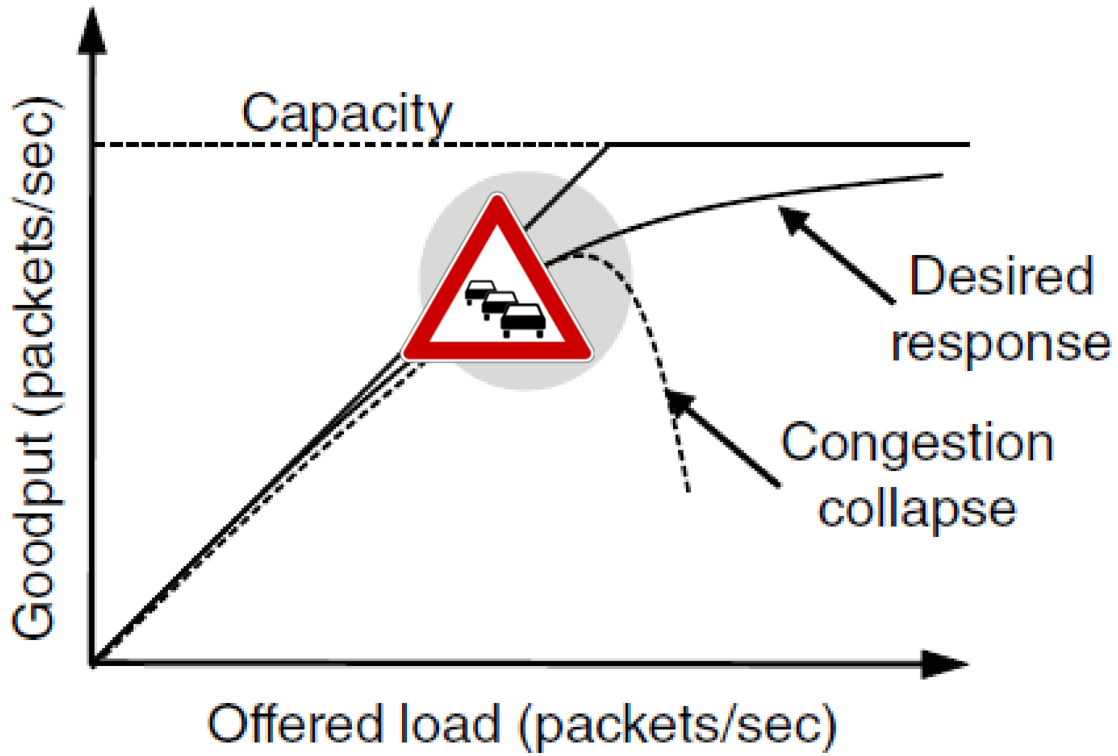
- What happens to performance as we increase load?

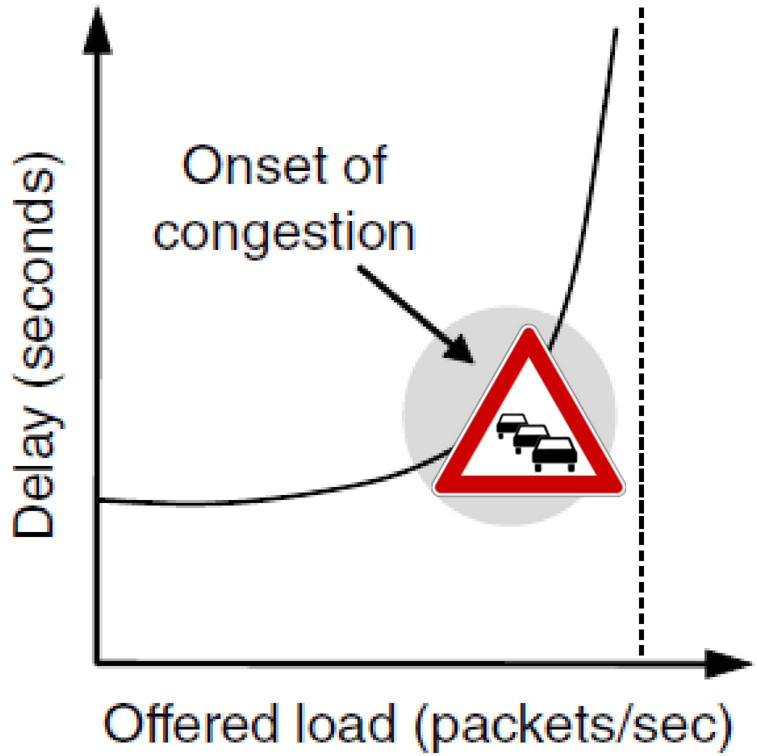




Effects of Congestion

- What happens to performance as we increase load?





Effects of Congestion

- As offered load rises, congestion occurs as queues begin to fill:
  - Delay and loss rise sharply with more load
  - Throughput falls below load (due to loss)
  - Goodput may fall below throughput (due to retransmissions)
- None of the above is good!
  - Want network to operate just before congestion

Bandwidth Allocation: Goals

- Important task for network is to allocate its capacity to senders
  - Good allocation is both efficient and fair
- Efficient means goodput is very near the hardware transmission rate(s)
- Fair means something about senders getting a reasonable share the network

Bandwidth Allocation

- Why is it hard? (Just split equally!)
  - Number of senders and their offered loads change
  - Senders may lack capacity in different parts of network
  - Network is distributed; no single party has an overall picture of its state

Bandwidth Allocation

- Solution framework:
  - Senders adapt concurrently based on their own view of the network
  - Design this adaption so the network usage as a whole is efficient and fair
  - Adaption is continuous since offered loads continue to change over time

Fair Allocations

Fair Allocation

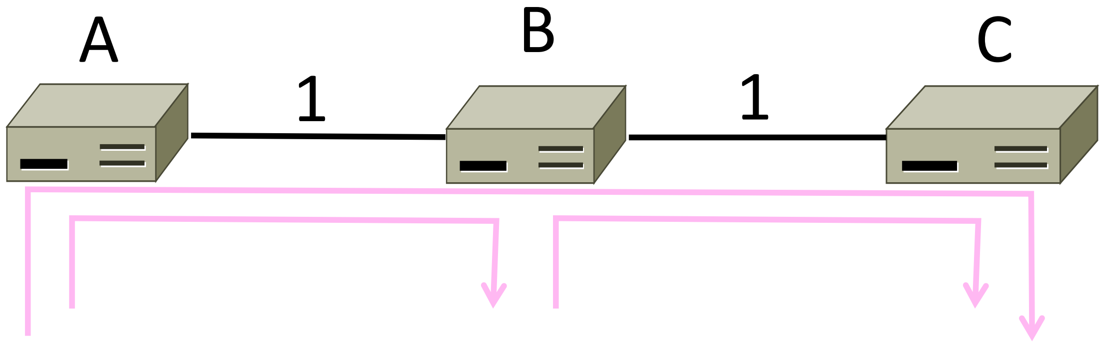
- What's a "fair" bandwidth allocation?
  - One answer: the max-min fair allocation

Recall

- We want a good bandwidth allocation to be both fair and efficient
  - Now we define what fair means
- Caveat: in practice, efficiency is more important than fairness

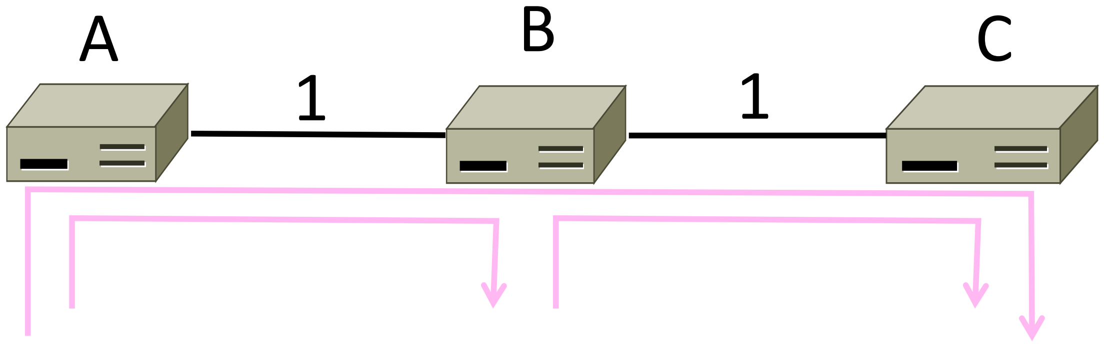
Efficiency vs. Fairness

- Cannot always have both!
  - Example network with traffic:
    - A→B, B→C and A→C
  - How much traffic can we carry?



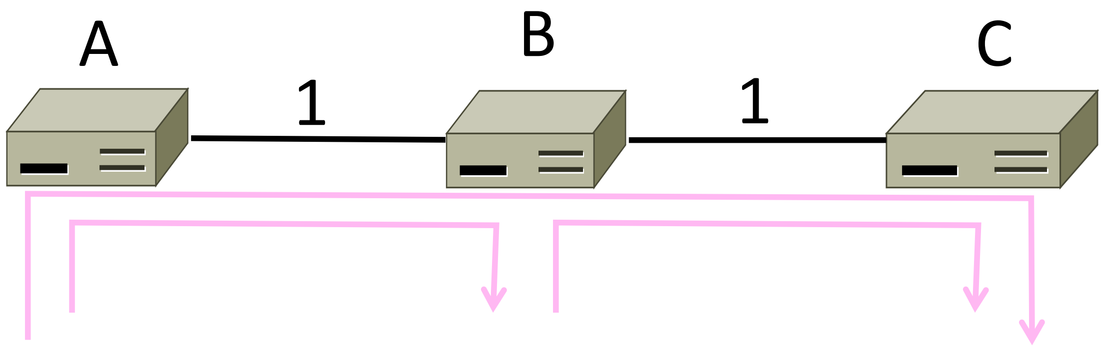
Efficiency vs. Fairness

- If we care about fairness:
  - Give equal bandwidth to each flow
  - A→B: ½ unit, B→C: ½, and A→C, ½
  - Total traffic carried is 1 ½ units



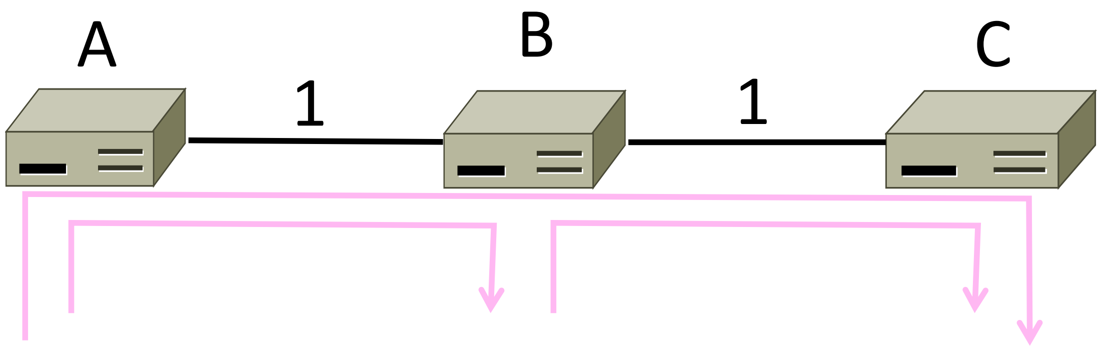
Efficiency vs. Fairness

- If we care about efficiency:
  - Maximize total traffic in network
  - A→B: 1 unit, B→C: 1, and A→C, 0
  - Total traffic rises to 2 units!



The Unclear Notion of Fairness

- Want to be fair over what?
  - Sending hosts?
  - Receiving hosts?
  - Flows (TCP connections)?
  - Applications?
  - Users?



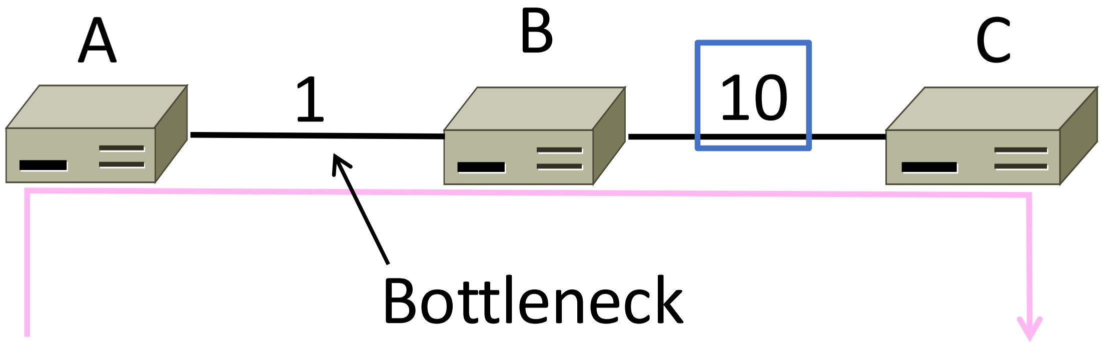
- Only one of these is “easy to implement”: flows

The Unclear Notion of Fairness

- We look for fairness among flows because we have a way to manipulate individual flows
  - TCP implementations
- Everything else is awkward – no one easily has the information required
- So, we're looking for "equal per flow"

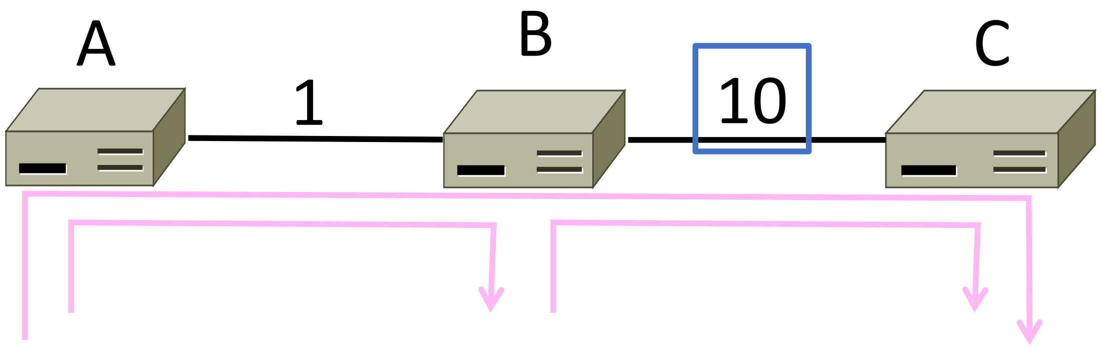
"Equal per Flow" and Bottlenecks

- Bottleneck for a flow of traffic is the link that limits its bandwidth
  - Where congestion occurs for the flow
  - For A→C, link A–B is the bottleneck
    - A can't use more than 10% of B-C, no matter its offered load



"Equal per Flow" and Bottlenecks

- Flows may have different bottlenecks
  - For A→C, link A–B is the bottleneck
  - For B→C, link B–C is the bottleneck
  - Can no longer divide links equally …



Max-Min Fairness

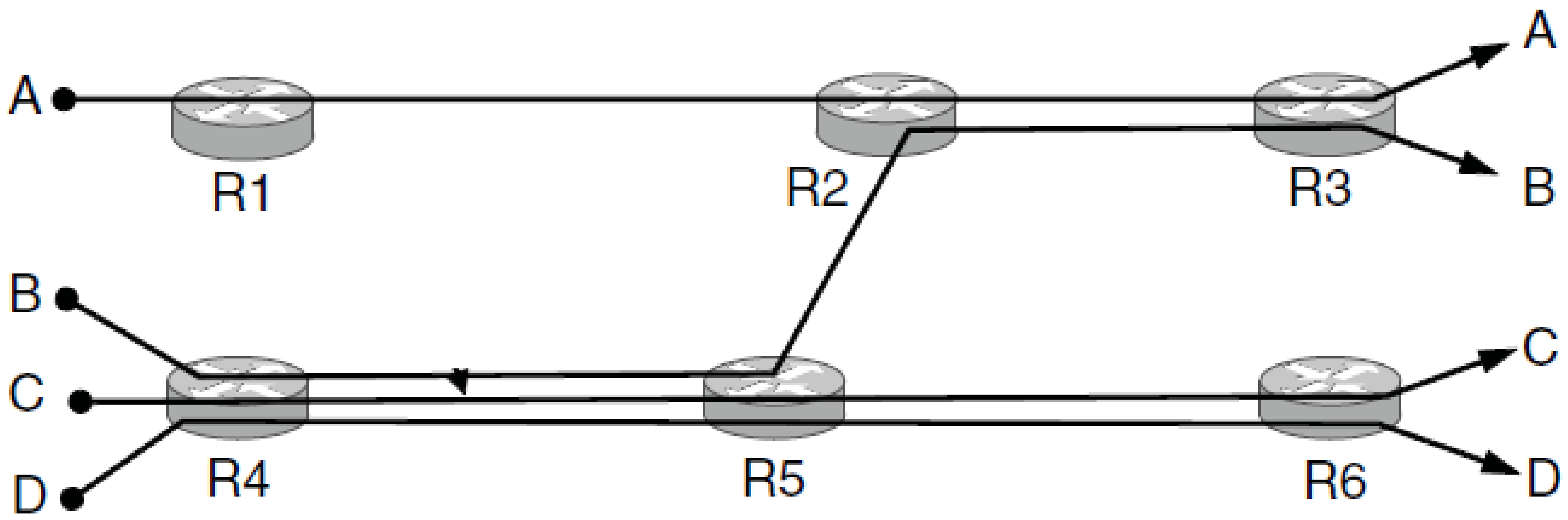
- Intuitively, flows bottlenecked on a link get an equal share of that link
- Max-min fair allocation is one that:
  - Increasing the rate of one flow will decrease the rate of a smaller flow
  - This “maximizes the minimum” flow

Max-Min Fairness (2)

- To find it given a network, imagine “pouring water into the network”
  1. Start with all flows at rate 0
  2. Increase the flows until there is a new bottleneck in the network
  3. Hold fixed the rate of the flows that are bottlenecked
  4. Go to step 2 for any remaining flows

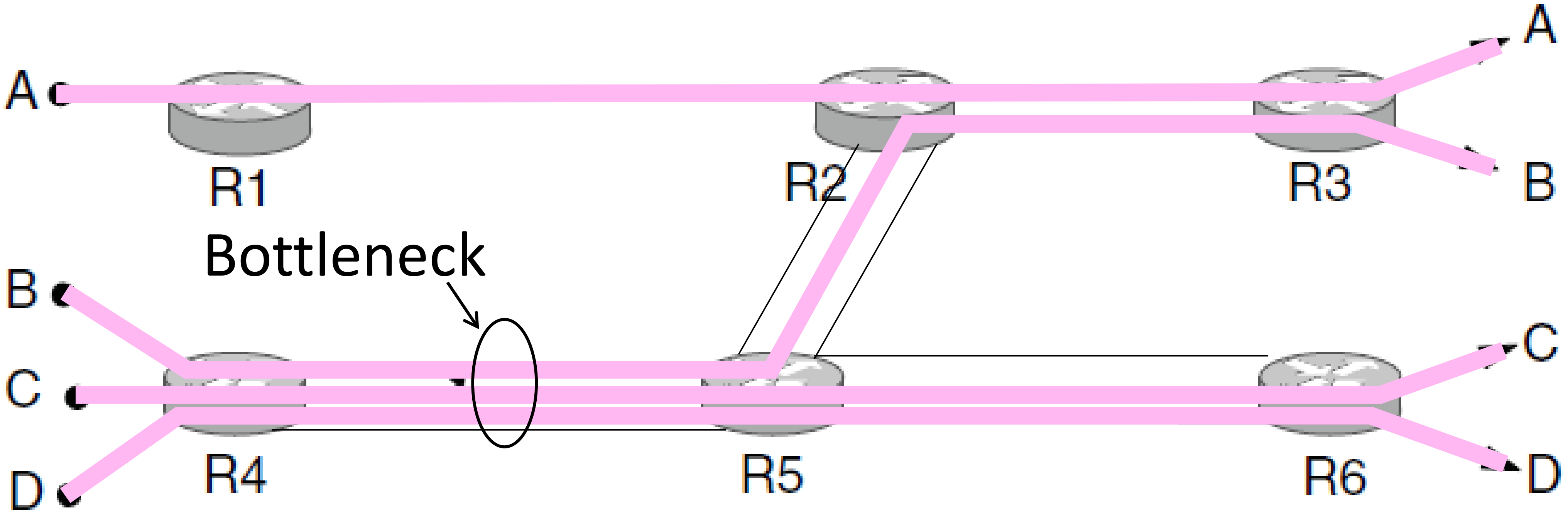
Max-Min Example

- Example: network with 4 flows, links equal bandwidth



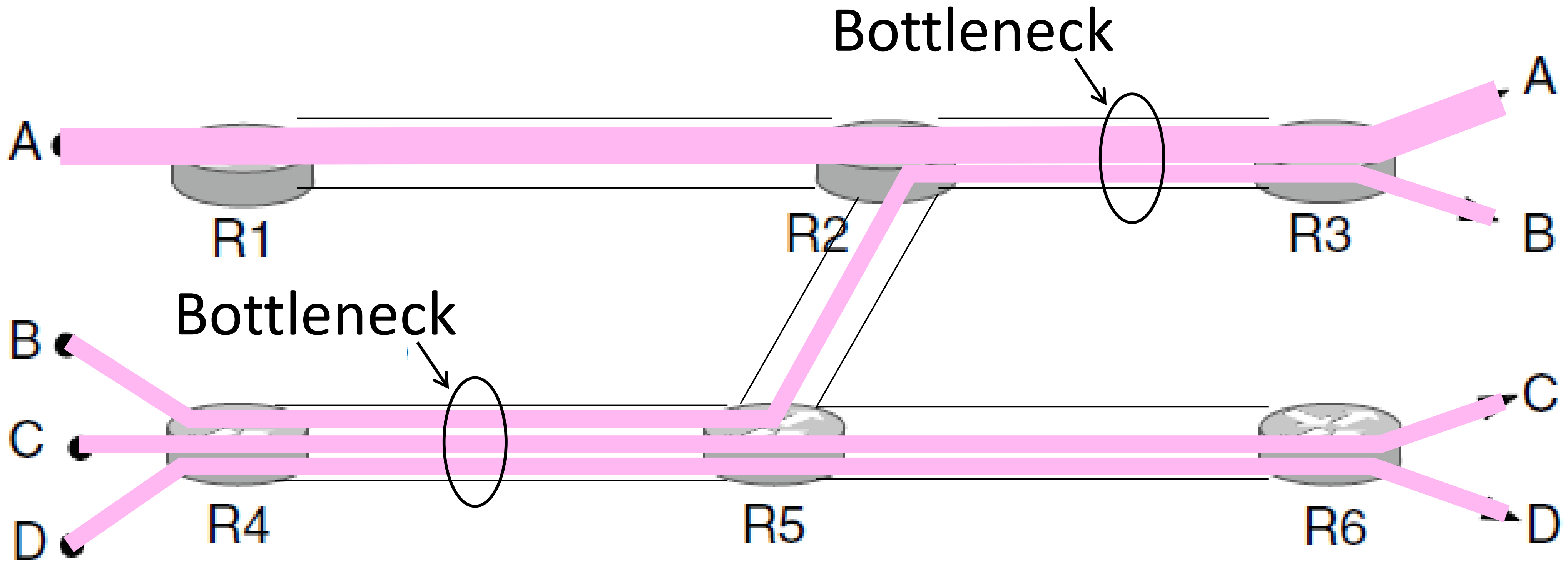
Max-Min Example

- When rate=1/3, flows B, C, and D bottleneck R4—R5
  - Fix B, C, and D, continue to increase A



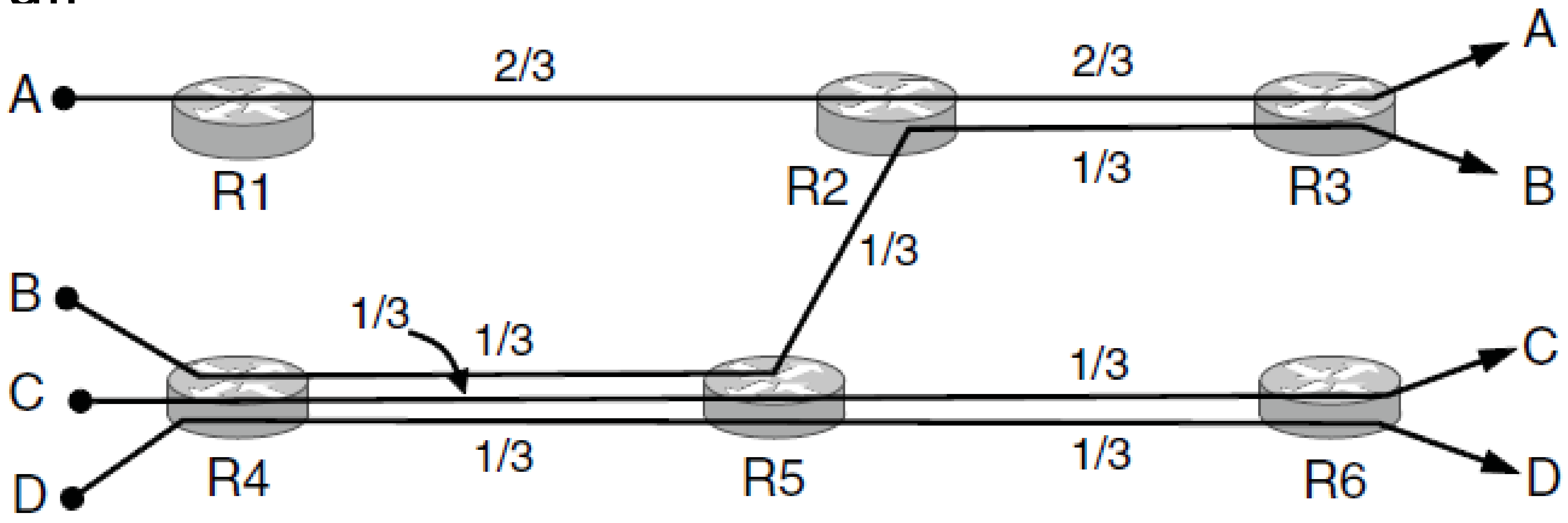
Max-Min Example

- When rate=2/3, flow A bottlenecks R2—R3. Done.



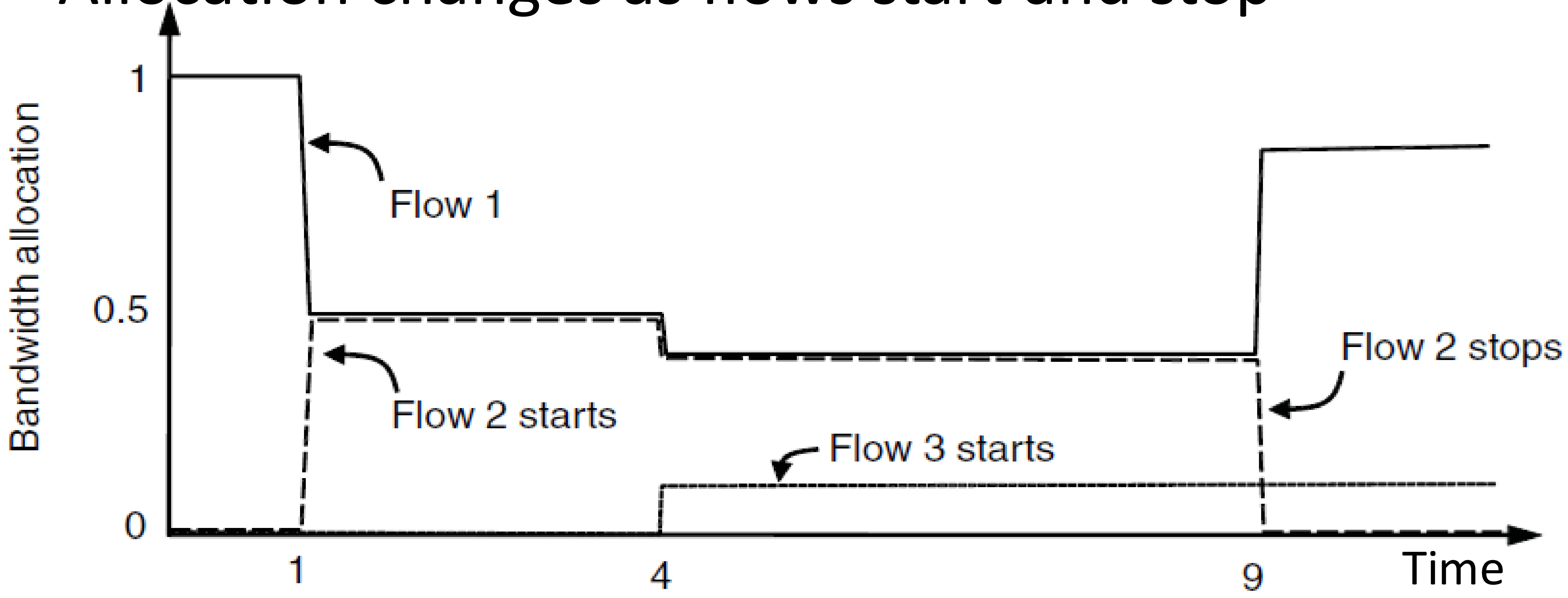
Max-Min Example

- End with A=2/3, B, C, D=1/3, and R2—R3, R4—R5 full

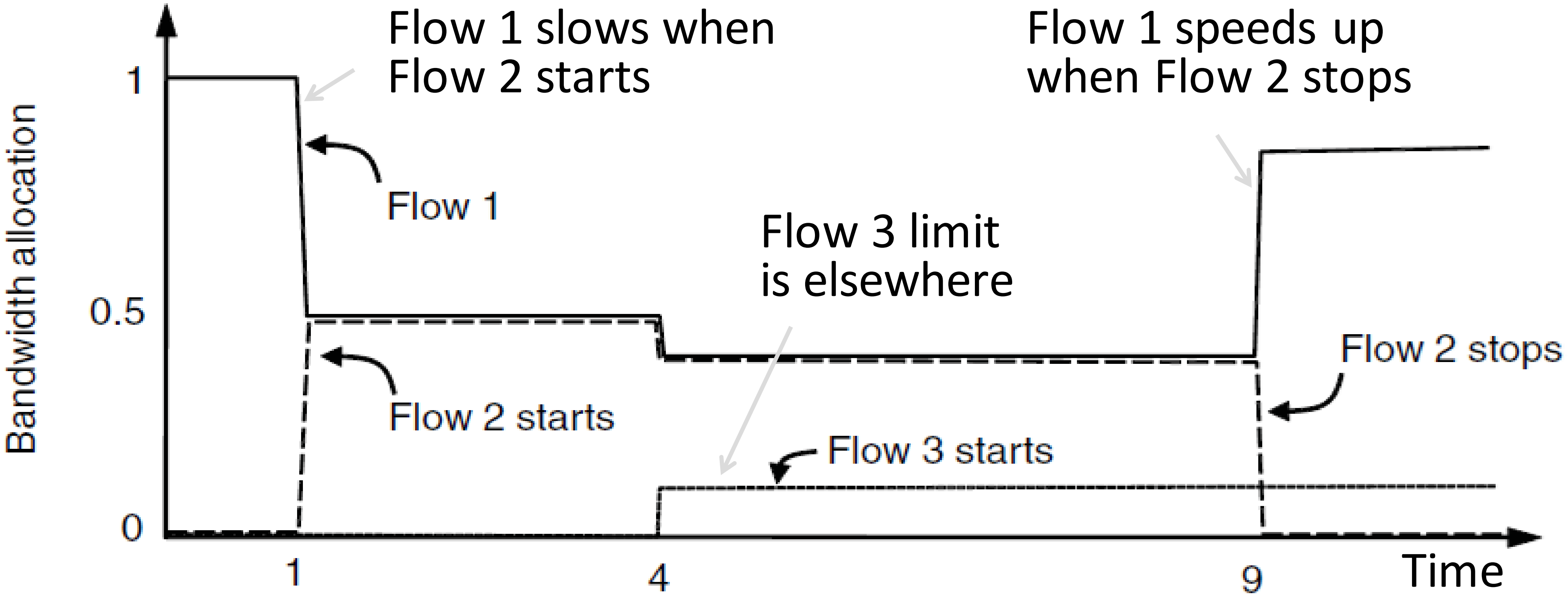


Adapting over Time

- Allocation changes as flows start and stop



Adapting over Time (2)



Bandwidth Allocation

Recall

- Want to allocate capacity to senders
  - Somehow get feedback about congestion at routers
  - Transport layer adjusts offered load
  - A good allocation is efficient and fair
- How should we perform the allocation?
  - Several different possibilities …

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? (Hosts? Routers?)
- Window versus Rate based
  - How is allocation expressed?

Bandwidth Allocation Models

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

- Infer information about congestion by observing characteristics of send-ACK loop
  - E.g., did segment seem to get to receiver? how long did it take?
- TCP implementation adjusts sender's behavior via window in response
  - How senders adapt is a <u>control law</u>

Additive Increase Multiplicative Decrease (AIMD)

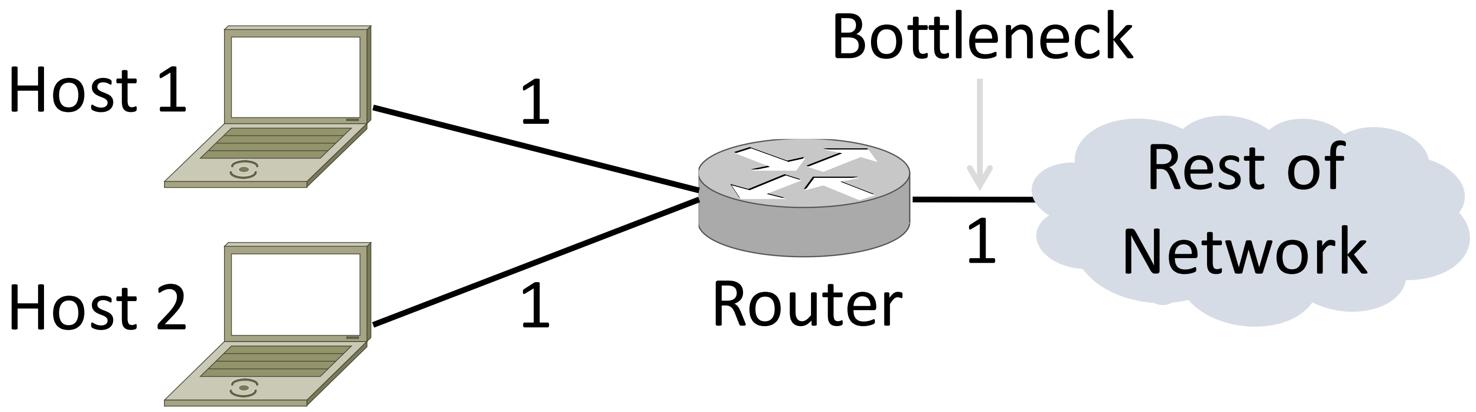
- AIMD is a control law hosts can use to reach a fair allocation (under idealized conditions, at least)

- AIMD:
  - Hosts additively increase rate while network not congested
  - Hosts multiplicatively decrease rate when congested
  - Used by TCP

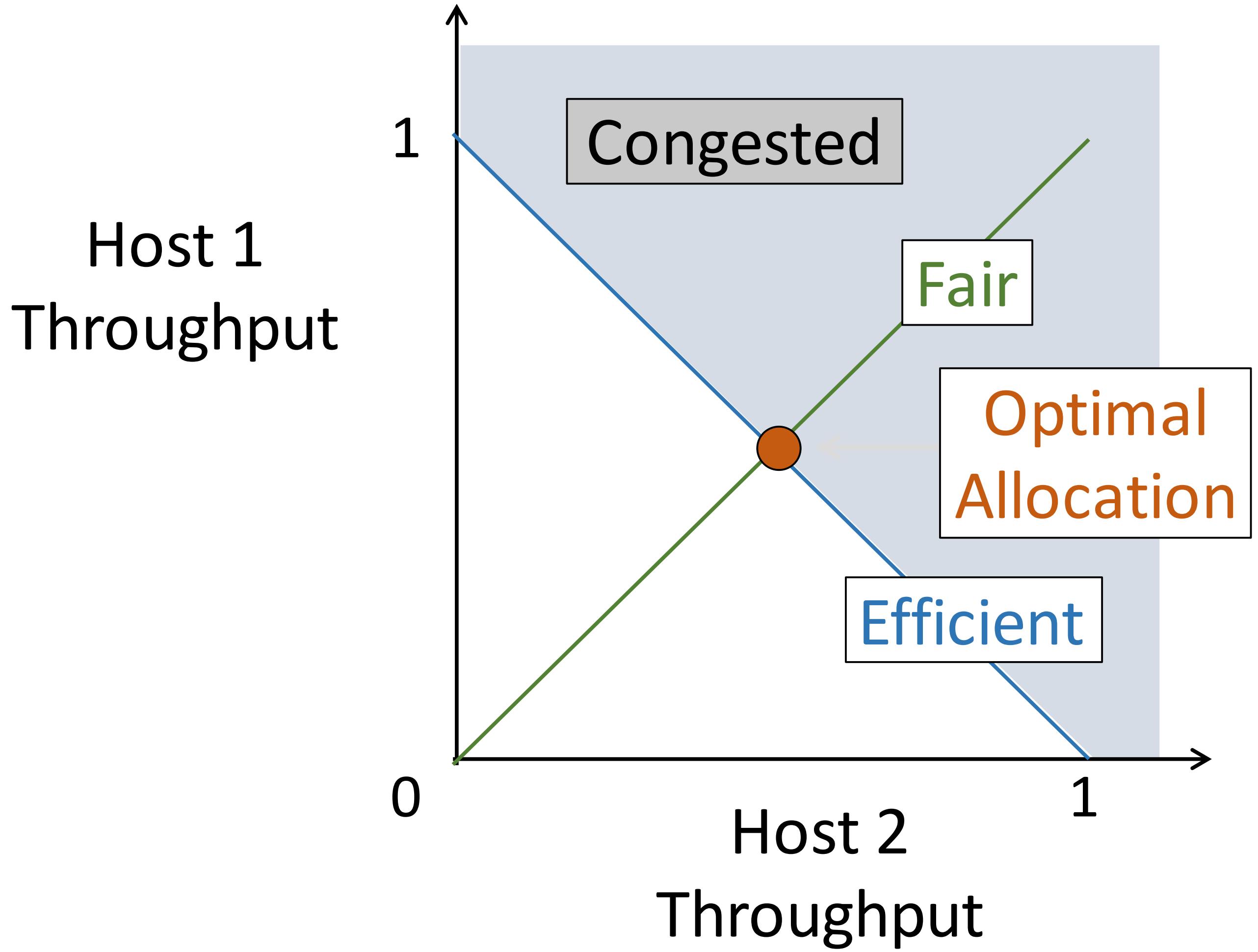
- Let's explore the AIMD game …

AIMD Game

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

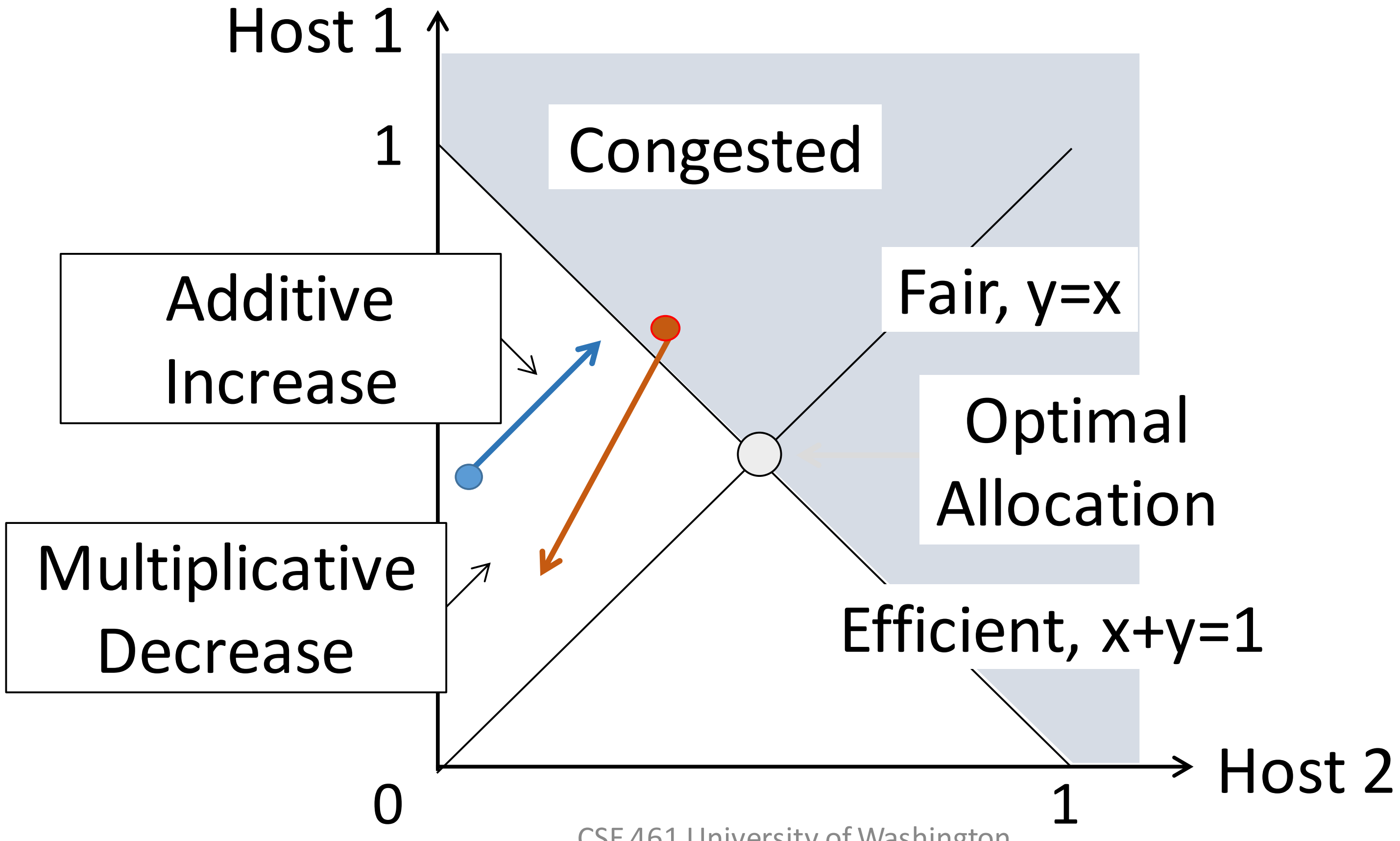


AIMD Game


Host 1
Throughput
1
Congested
Fair
Optimal
Allocation
Efficient
0
1
Host 2
Throughput

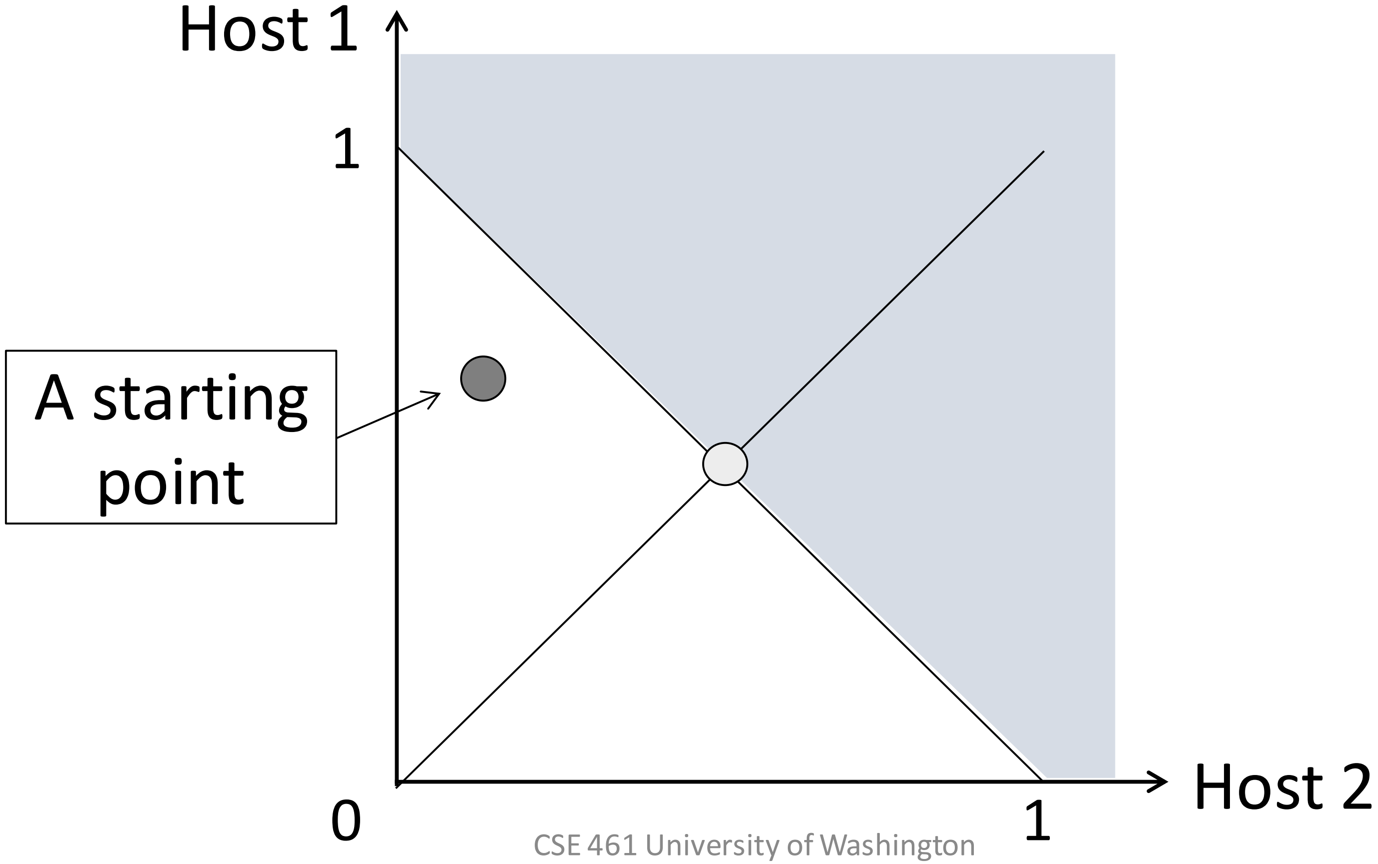
AIMD Game (3)

- AI and MD move the allocation



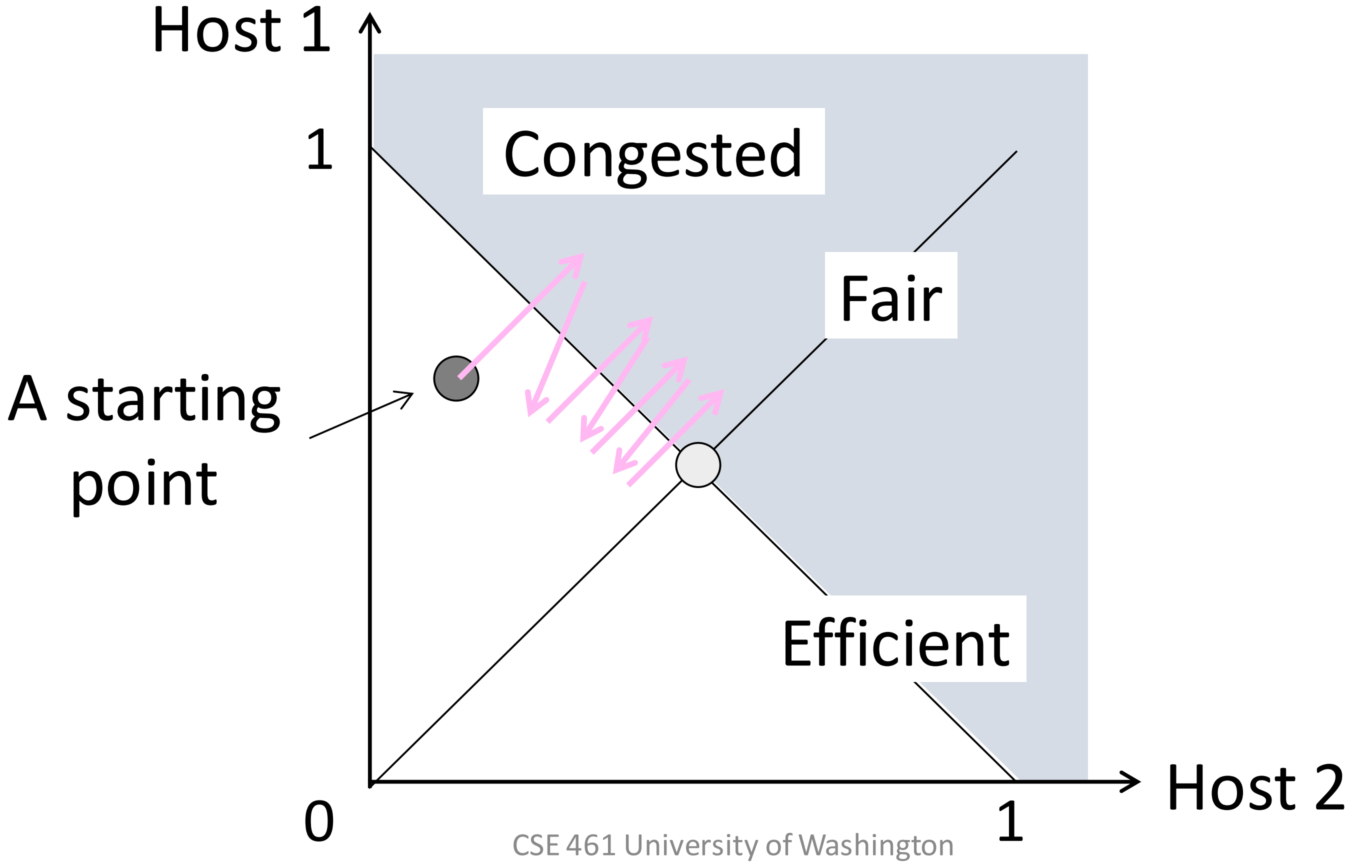
AIMD Game (4)

- Play the game!



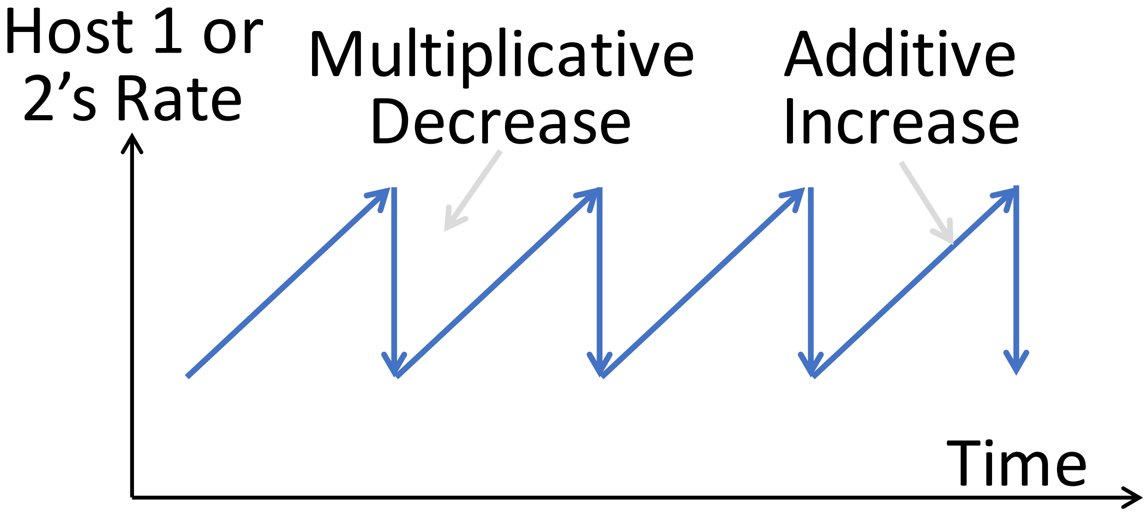
AIMD Game (5)

- Always converge to good allocation!



AIMD Sawtooth

- AIMD produces a “sawtooth” pattern over time for the 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
  - Why?
  - Holds for more general topologies
- Other increase/decrease control laws do not! (Try MIAD, MIMD, MIAD)
- Requires only simple (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
Packet delay	Compound TCP (Windows)	Hear about congestion early Need to infer congestion
Router indication	TCPs with Explicit Congestion Notification	Hear about congestion early Require router support