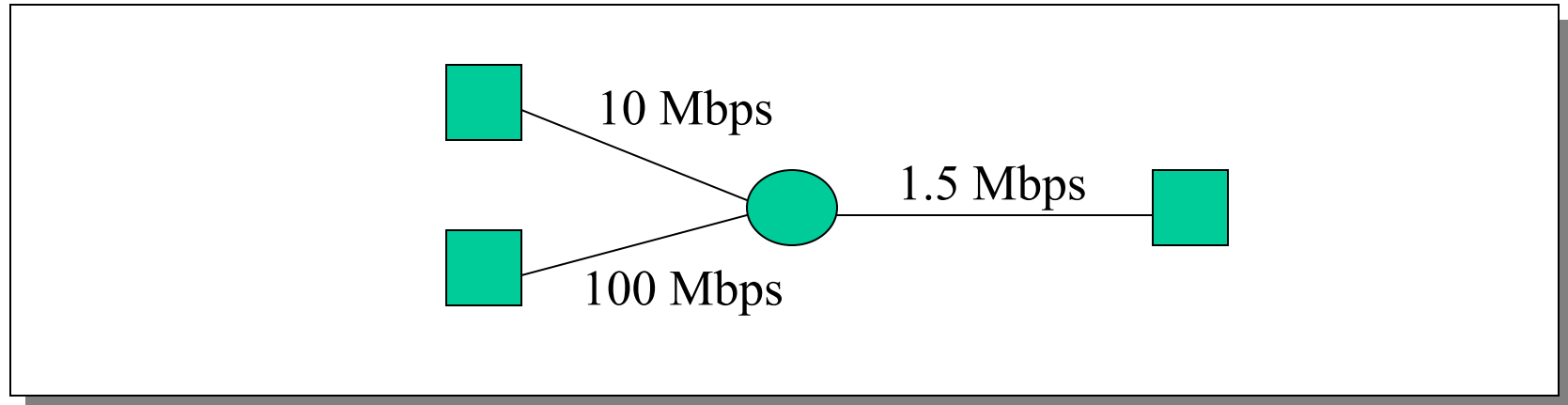


# Principles of congestion control

## 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)
- ❑ Another top-10 problem!

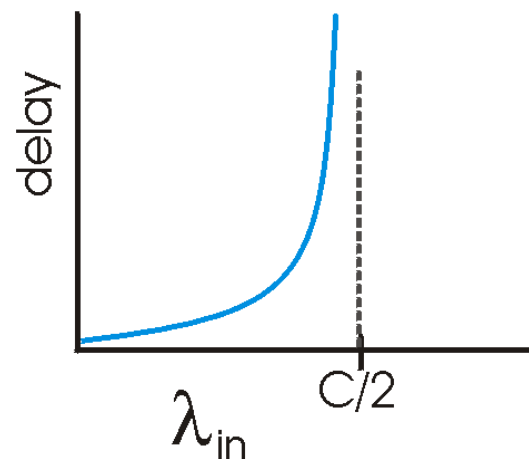
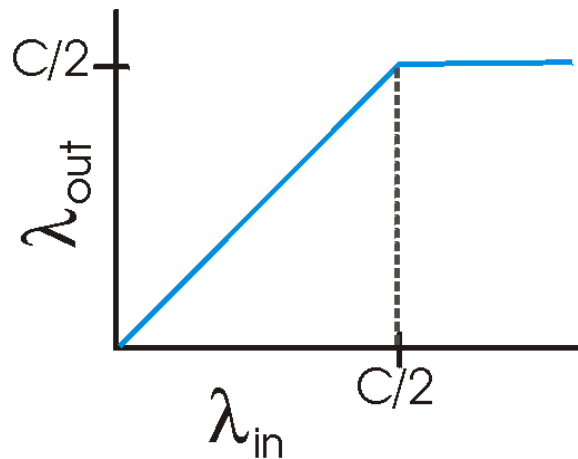
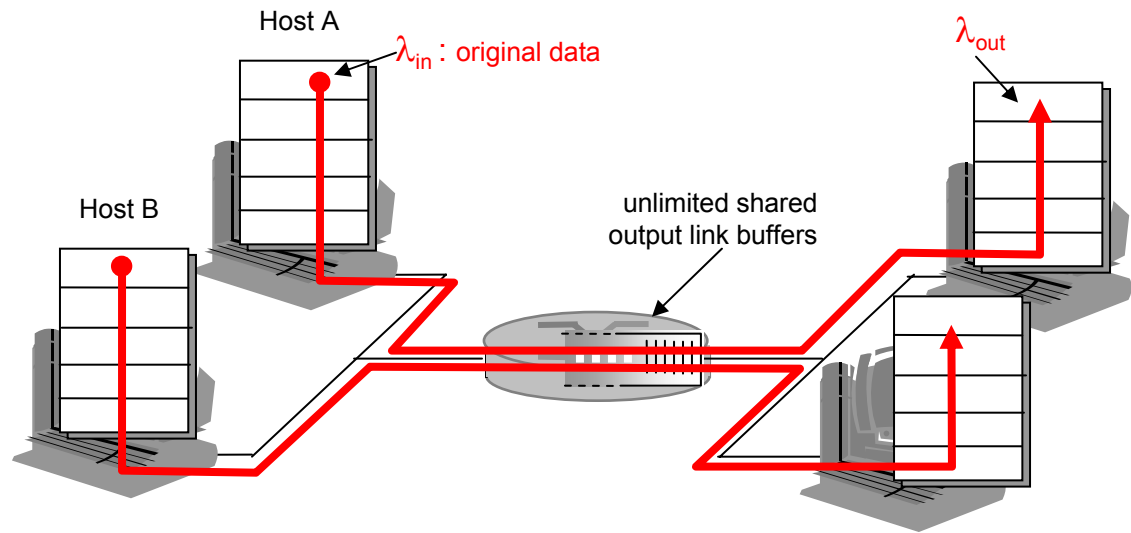
# Congestion



- ❑ Different sources compete for resources inside network
- ❑ Why is it a problem?
  - Sources are unaware of current state of resource
  - Sources are unaware of each other
  - In many situations will result in  $< 1.5$  Mbps of throughput (congestion collapse)

# Causes/costs of congestion: Scenario 1

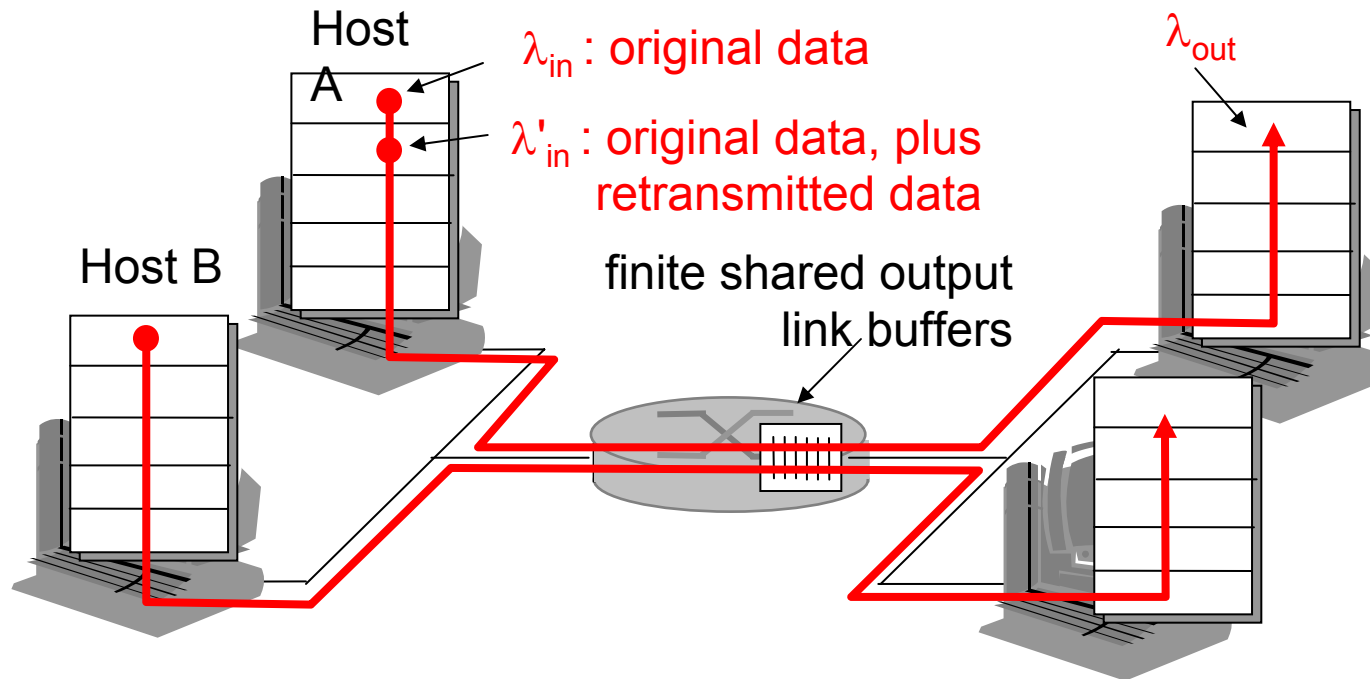
- ❑ Two senders, two receivers
- ❑ One router, infinite buffers
- ❑ No retransmission



- ❑ Maximum achievable throughput
- ❑ Large delays when congested

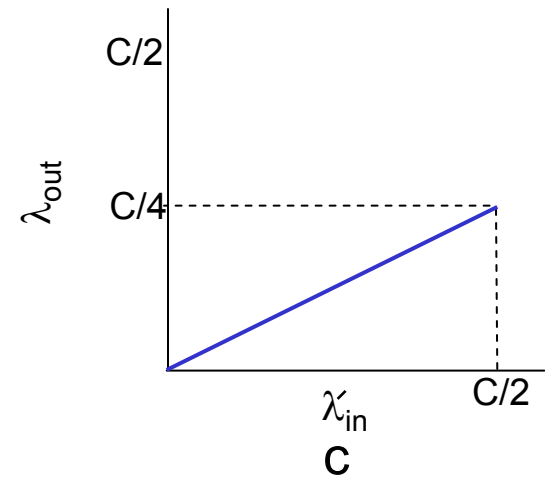
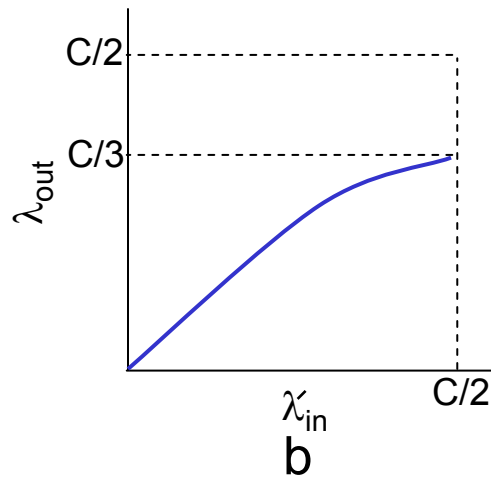
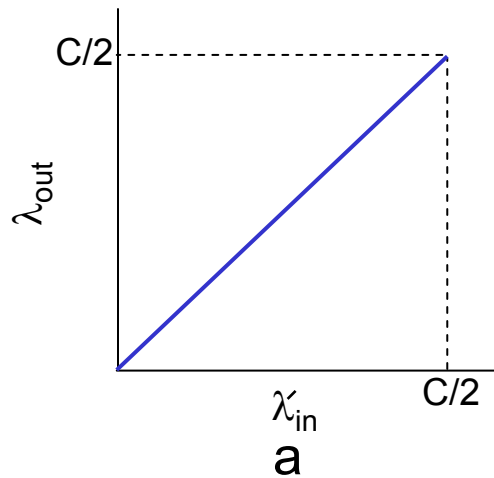
# Causes/costs of congestion: Scenario 2

- ❑ One router, *finite* buffers
- ❑ Sender retransmission of lost packet



# Causes/costs of congestion: Scenario 2

- Always:  $\lambda_{in} = \lambda_{out}$  (goodput)
- "Perfect" retransmission only when loss:  $\lambda'_{in} > \lambda_{out}$
- Retransmission of delayed (not lost) packet makes  $\lambda'_{in}$  larger (than perfect case) for same  $\lambda_{out}$



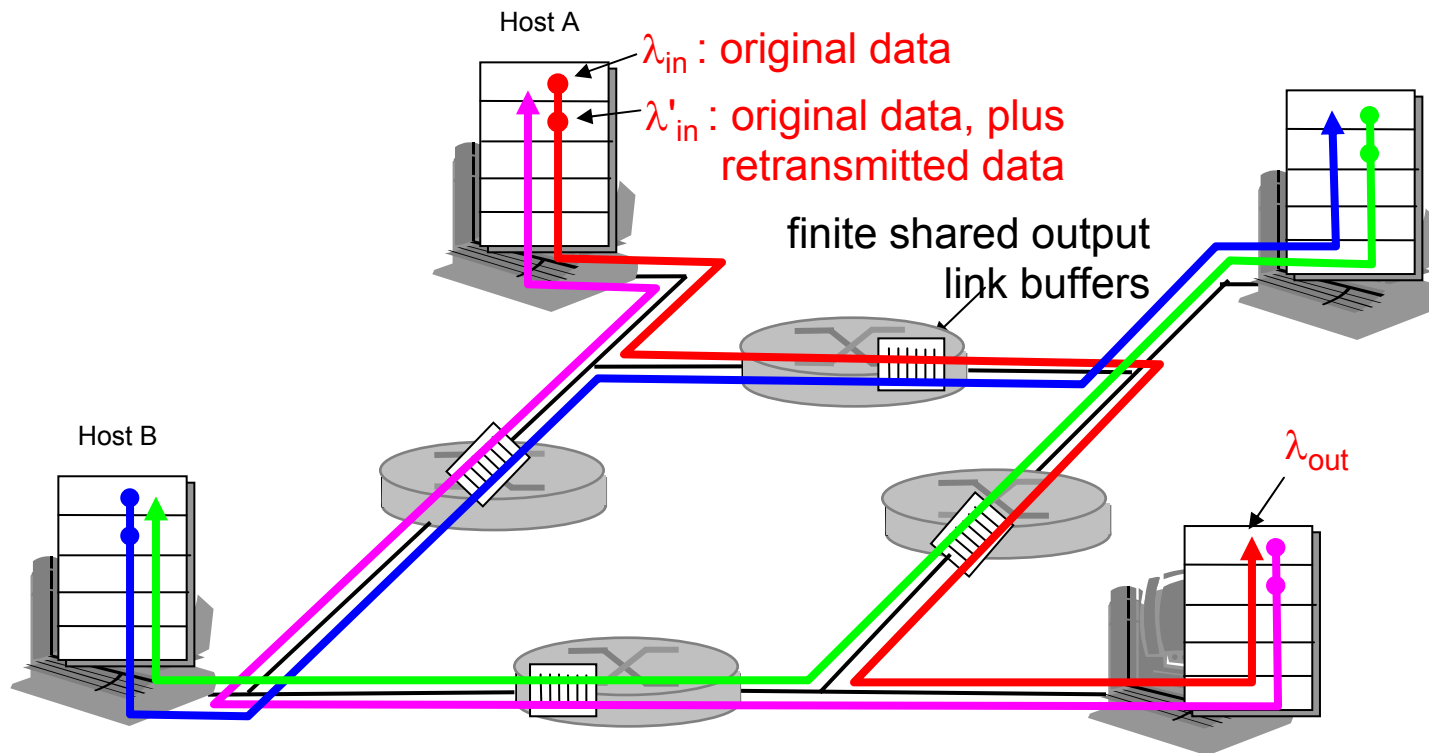
## "Costs" of congestion:

- More work (retrans) for given "goodput"
- Unneeded retransmissions: link carries multiple copies of pkt

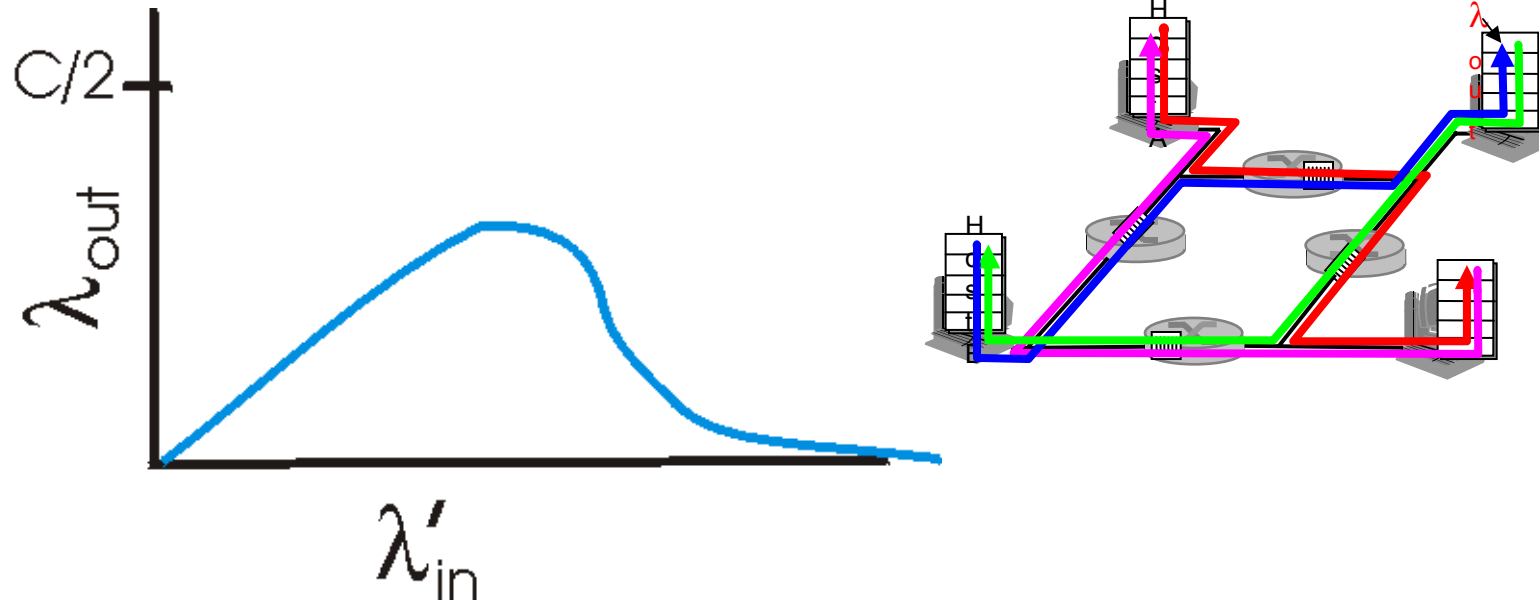
# Causes/costs of congestion: Scenario 3

- ❑ Four senders
- ❑ Multihop paths
- ❑ Timeout/retransmit

Q: What happens as  $\lambda_{in}$  and  $\lambda'_{in}$  increase ?



# Causes/costs of congestion: Scenario 3



Another “cost” of congestion:

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

# 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



# Other congestion collapse causes

- ❑ Fragments
  - Mismatch of transmission and retransmission units
  - Solutions
    - Make network drop all fragments of a packet
    - Do path MTU discovery
- ❑ Control traffic
  - Large percentage of traffic is for control
    - Headers, routing messages, DNS, etc.
- ❑ Stale or unwanted packets
  - Packets that are delayed on long queues
  - “Push” data that is never used

# Where to prevent collapse?

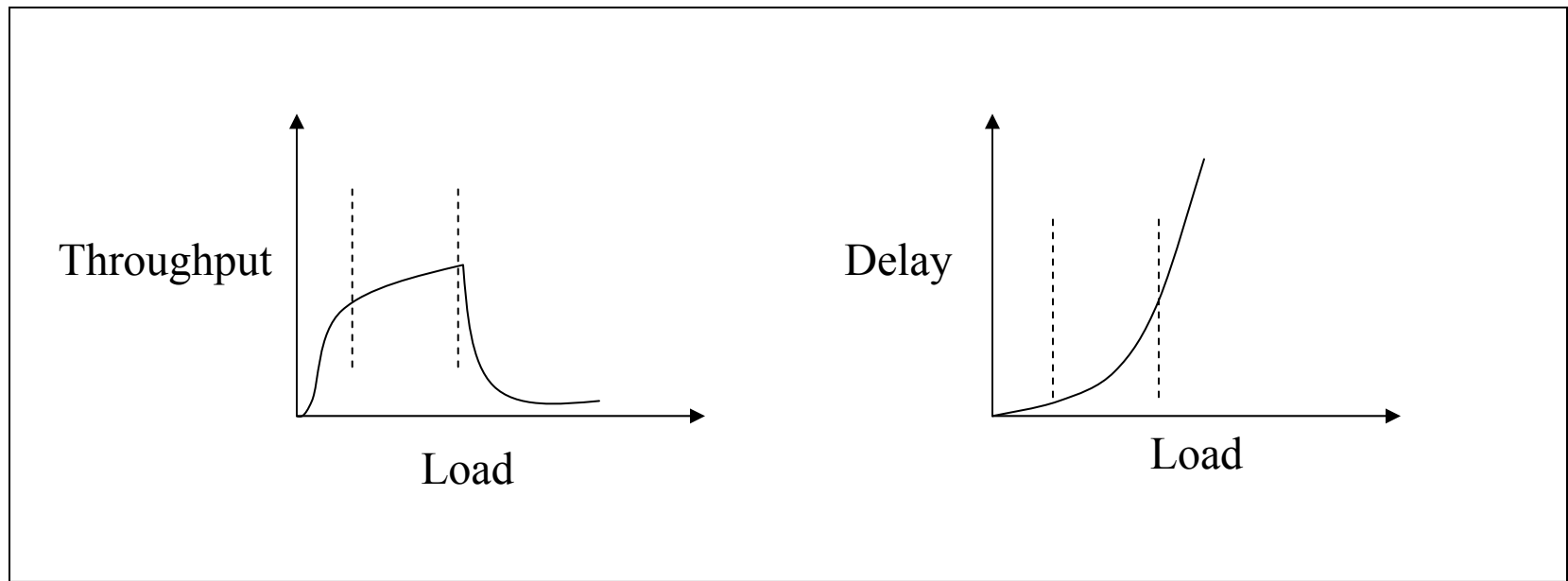
- ❑ Can end hosts prevent problem?
  - Yes, but must trust end hosts to do right thing
  - E.g., sending host must adjust amount of data it puts in the network based on detected congestion
- ❑ Can routers prevent collapse?
  - No, not all forms of collapse
  - Doesn't mean they can't help
  - Sending accurate congestion signals
  - Isolating well-behaved from ill-behaved sources

# Congestion control and avoidance

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

# Congestion control vs. avoidance

- ❑ Avoidance keeps the system performing at the knee
- ❑ Control kicks in once the system has reached a congested state



# 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
  - Choke packet from router to sender
  - Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - Explicit rate sender should send at

# Case study: ATM ABR congestion control

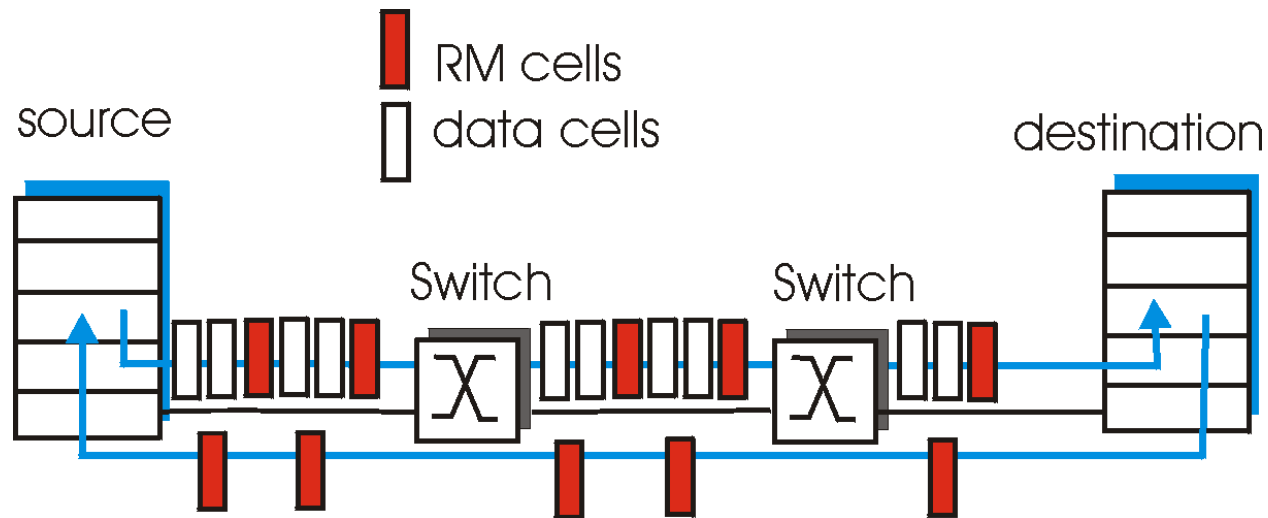
## ABR: available bit rate:

- ❑ “Elastic service”
- ❑ If sender’s path “underloaded”:
  - Sender should use available bandwidth
- ❑ If sender’s path congested:
  - Sender throttled to minimum guaranteed rate

## RM (resource management) cells:

- ❑ Sent by sender, interspersed with data cells
- ❑ Bits in RM cell set by switches (“*network-assisted*”)
  - **NI bit**: no increase in rate (mild congestion)
  - **CI bit**: congestion indication
- ❑ RM cells returned to sender by receiver, with bits intact

# Case study: ATM ABR congestion control



- ❑ Two-byte ER (explicit rate) field in RM cell
  - Congested switch may lower ER value in cell
  - Sender's send rate thus minimum supportable rate on path
- ❑ EFCI bit in data cells: set to 1 in congested switch
  - If data cell preceding RM cell has EFCI set, sender sets CI bit in returned RM cell

# End-to-end congestion control - objectives

- ❑ Simple router behavior
- ❑ Distributedness
- ❑ Efficiency:  $X_{knee} = \sum x_i(t)$
- ❑ Fairness:  $(\sum x_i)^2 / n(\sum x_i^2)$
- ❑ Power: (throughput <sup>$\alpha$</sup> /delay)
- ❑ Convergence: control system must be stable



# Basic control model

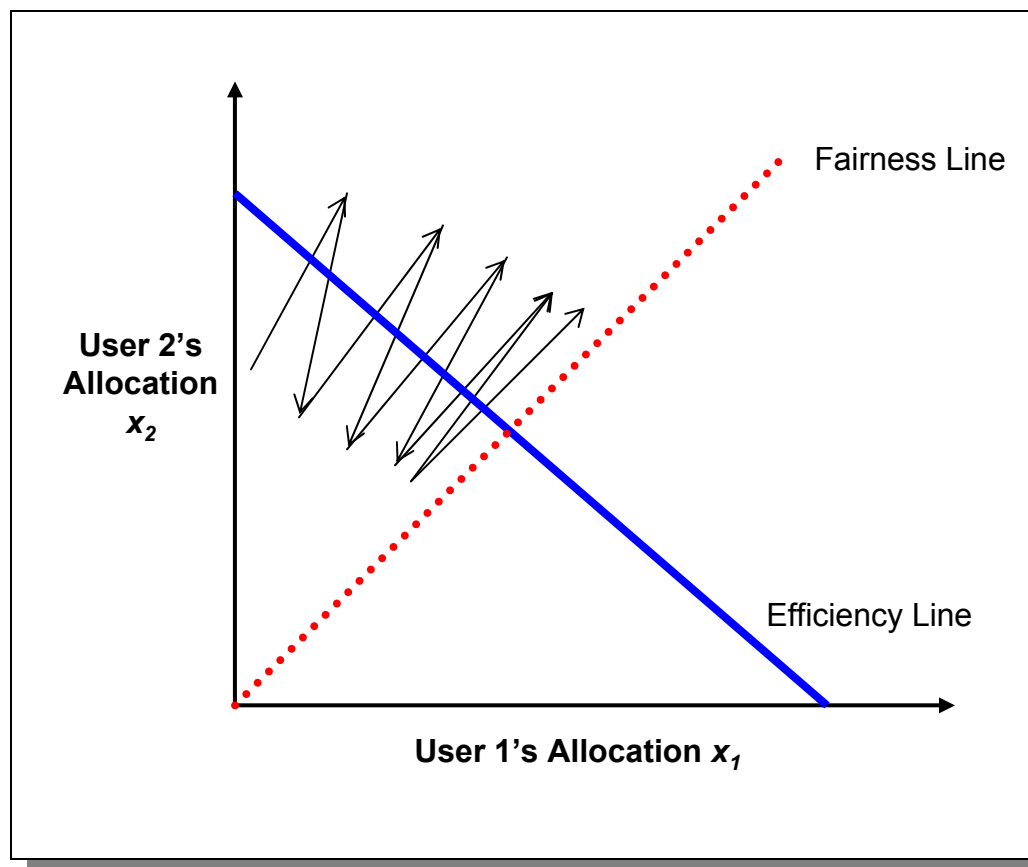
- ❑ Let's assume window-based control
- ❑ Reduce window when congestion is perceived
  - How is congestion signaled?
    - Either mark or drop packets
  - When is a router congested?
    - Drop tail queues – when queue is full
    - Average queue length – at some threshold
- ❑ Increase window otherwise
  - Probe for available bandwidth – how?

# Linear control

- Many different possibilities for reaction to congestion and probing
  - Examine simple linear controls
  - $\text{Window}(t + 1) = a + b \text{Window}(t)$
  - 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?

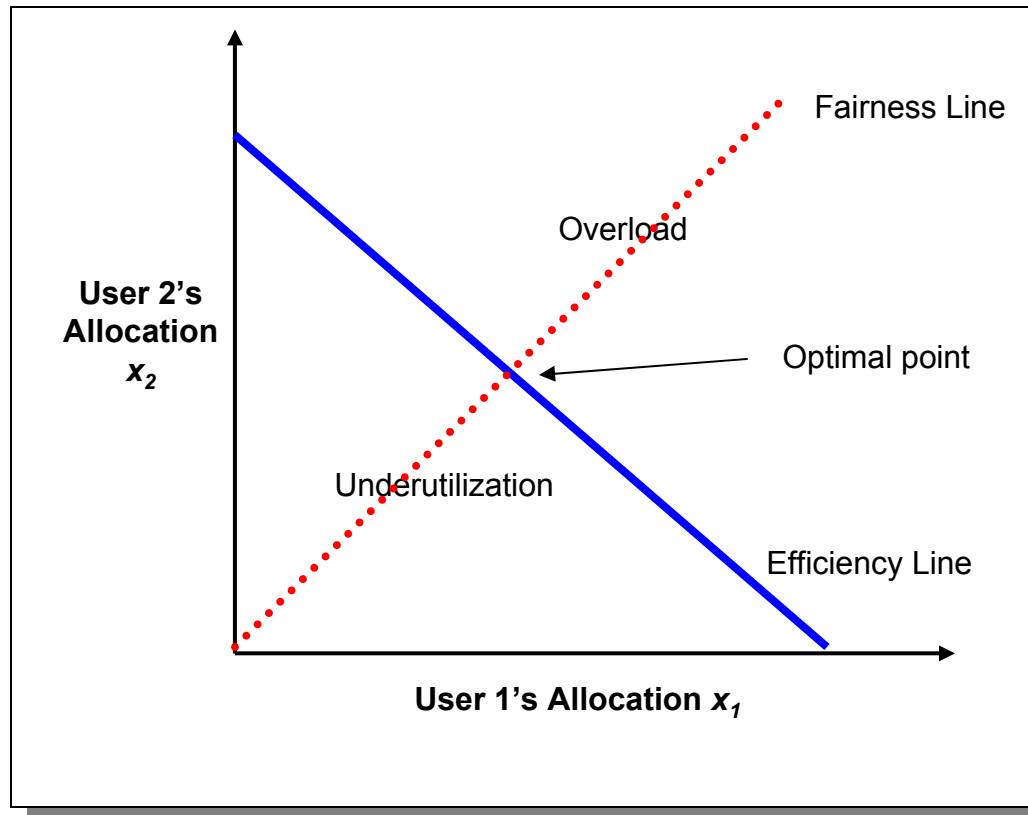
# Phase plots

- Simple way to visualize behavior of competing connections over time



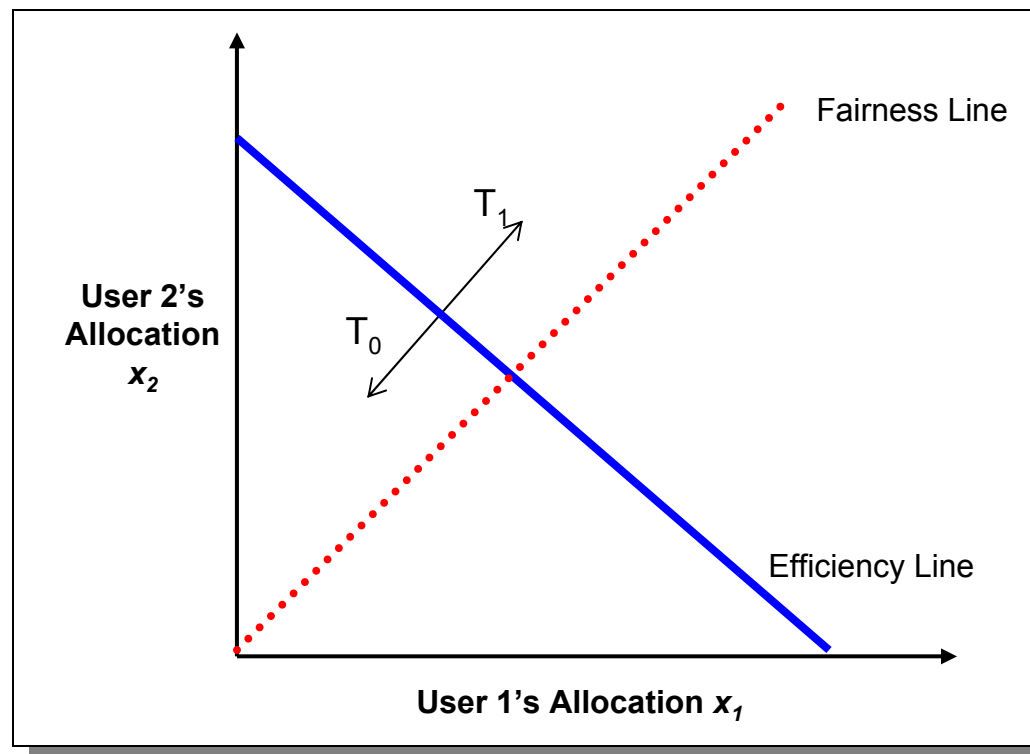
# Phase plots

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



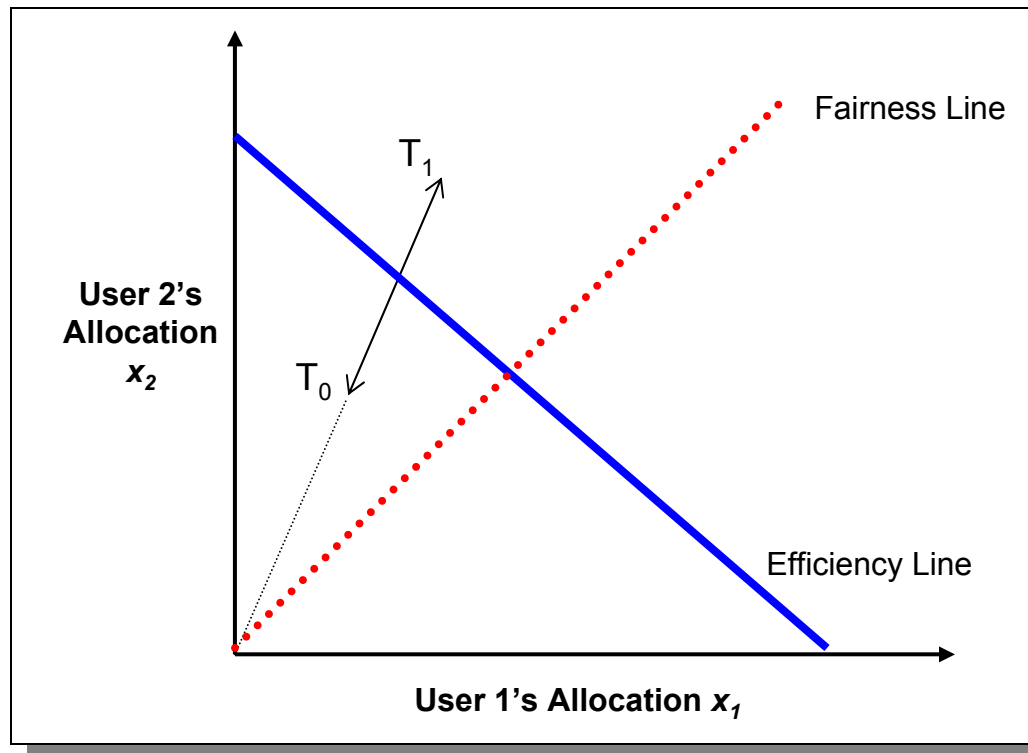
# Additive increase/decrease

- $x_1$  and  $x_2$  in-/decrease by same amount over time



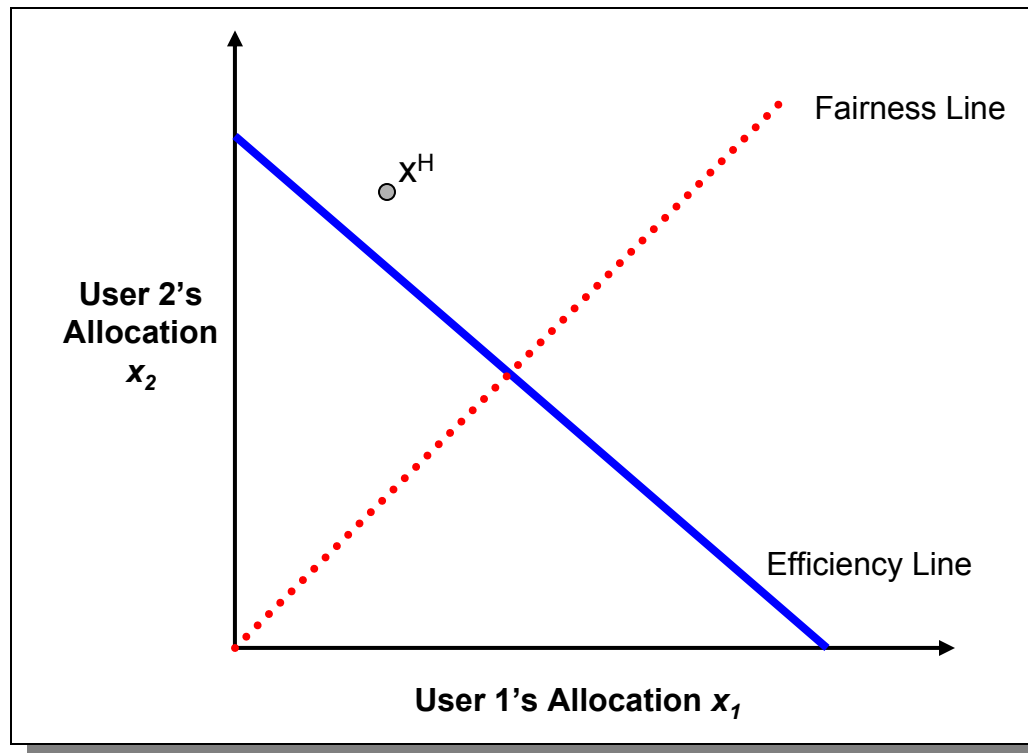
# Multiplicative increase/decrease

- $X_1$  and  $X_2$  in-/decrease by the same factor
  - Extension from origin

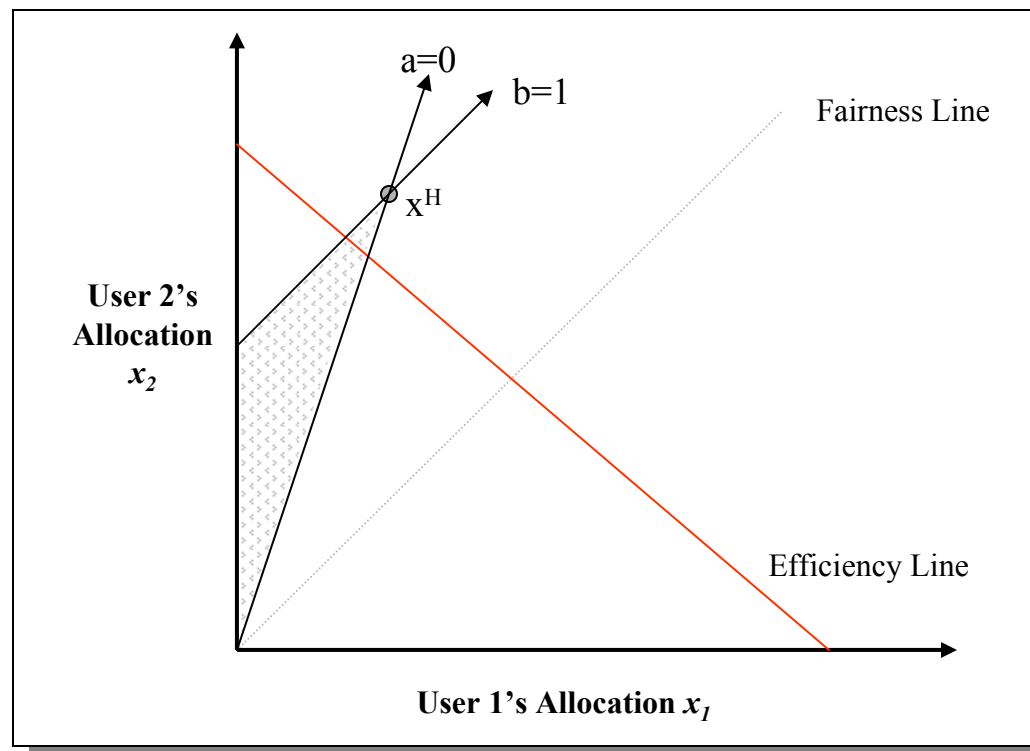


# Convergence to efficiency

- Want to converge quickly to intersection of fairness and efficiency lines

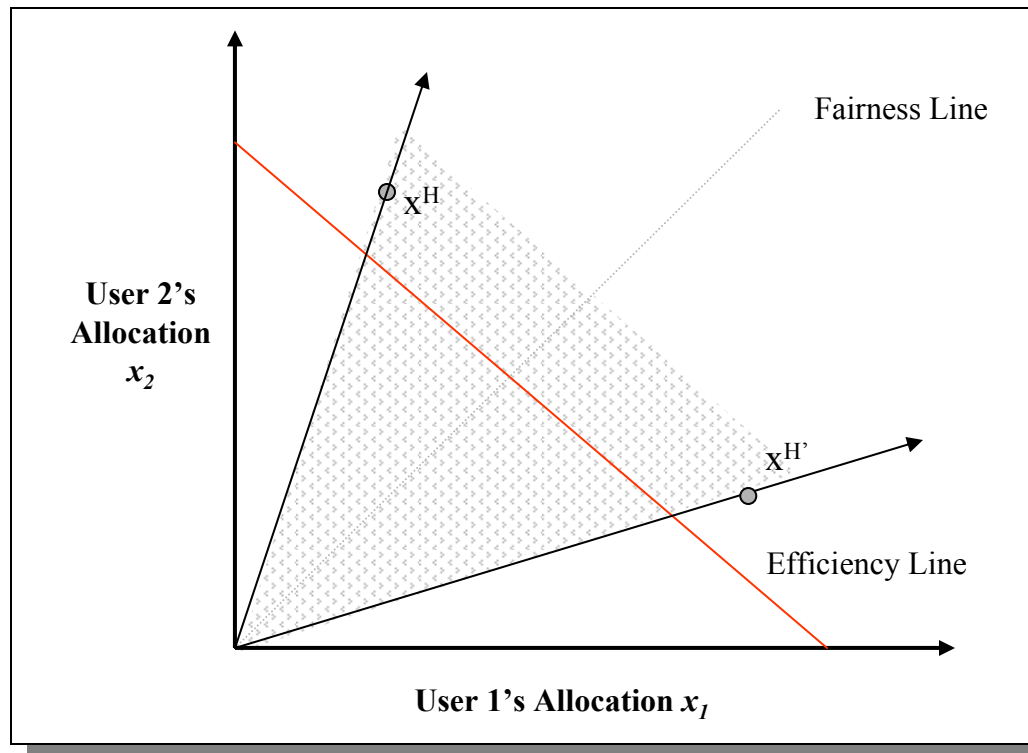


# Distributed convergence to efficiency

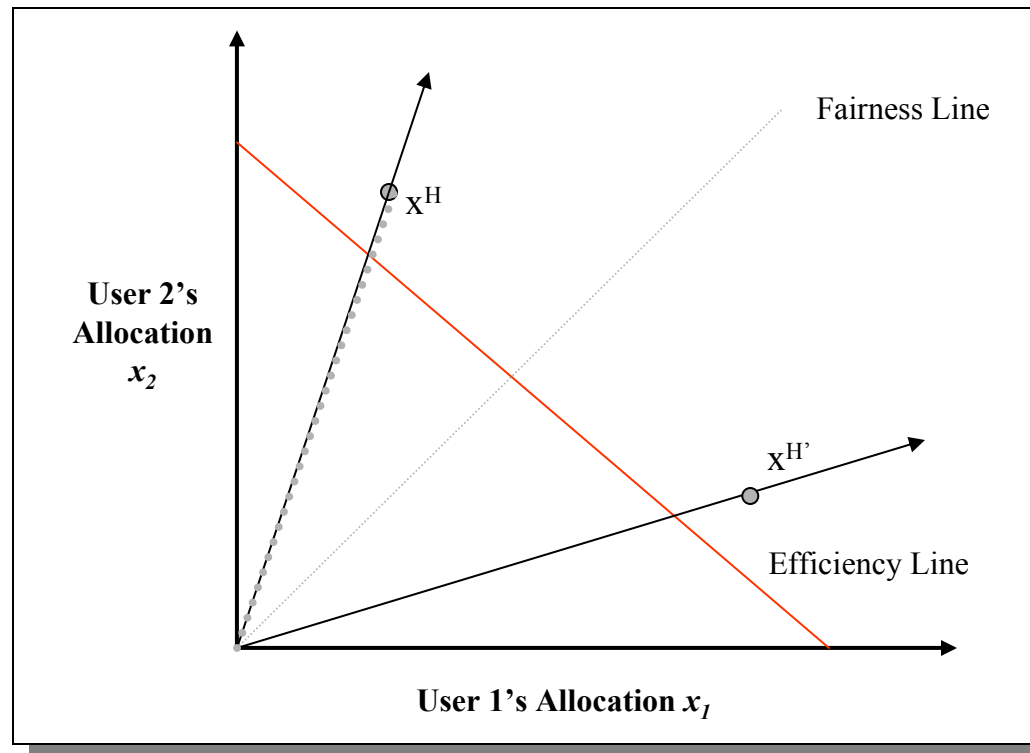




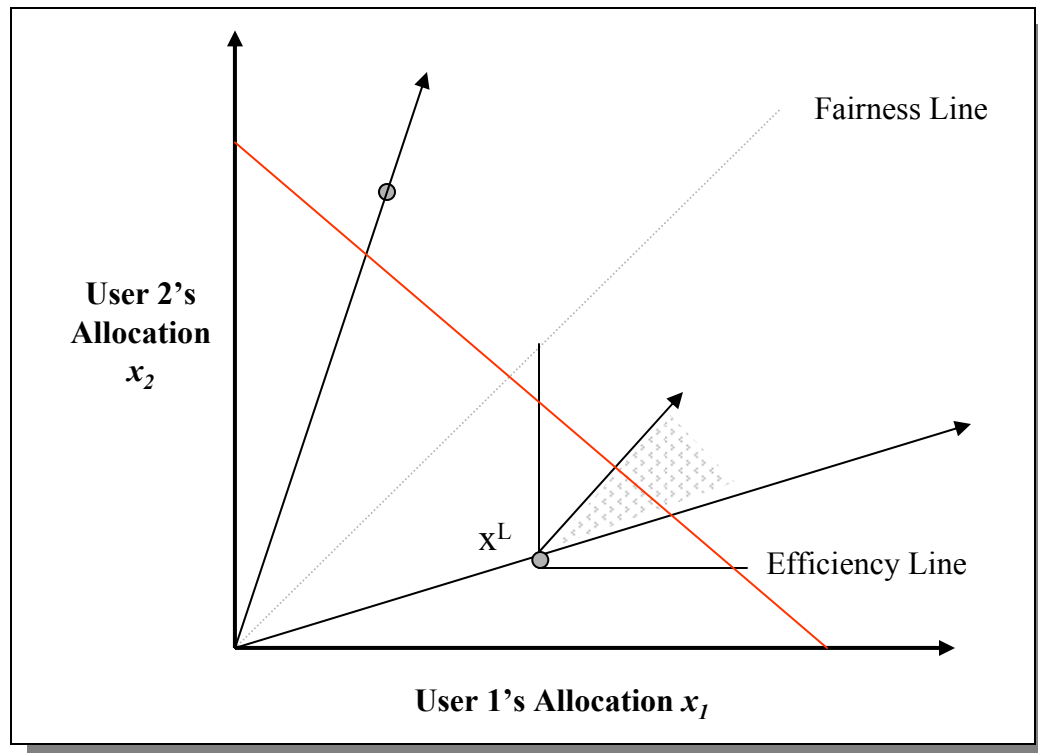
# Convergence to fairness



# Convergence to efficiency & fairness

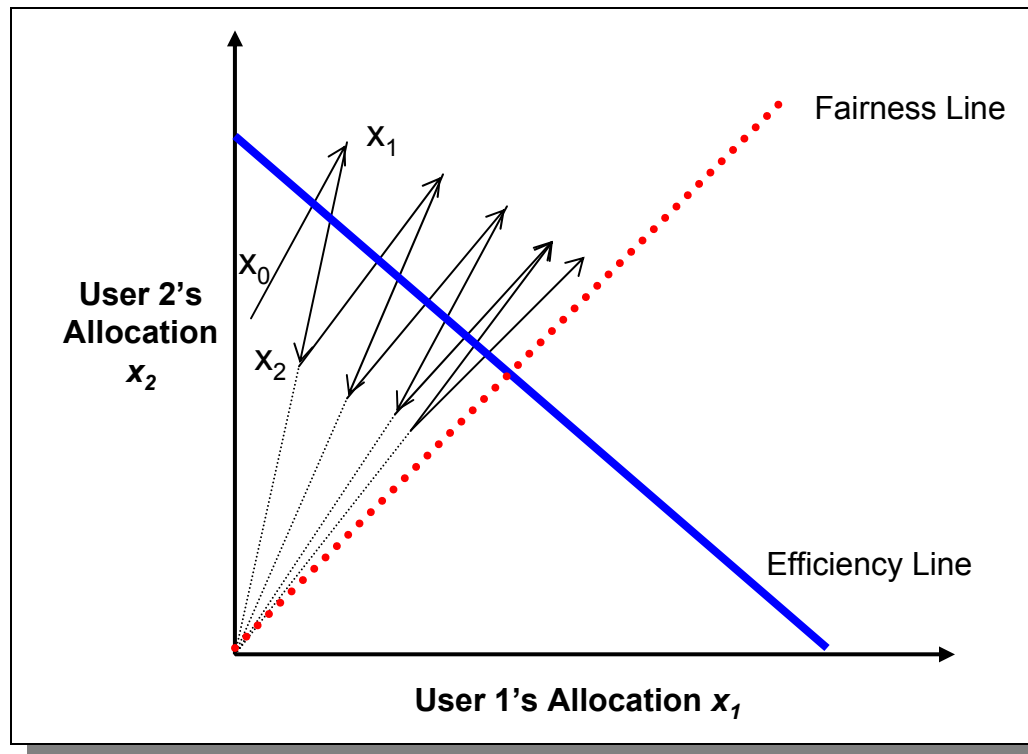


# Increase



# What is the right choice?

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



# TCP congestion control

- ❑ Motivated by ARPANET congestion collapse
- ❑ Underlying design principle: packet conservation
  - At equilibrium, inject packet into network only when one is removed
  - Basis for stability of physical systems
- ❑ Why was this not working?
  - Connection doesn't reach equilibrium
  - Spurious retransmissions
  - Resource limitations prevent equilibrium

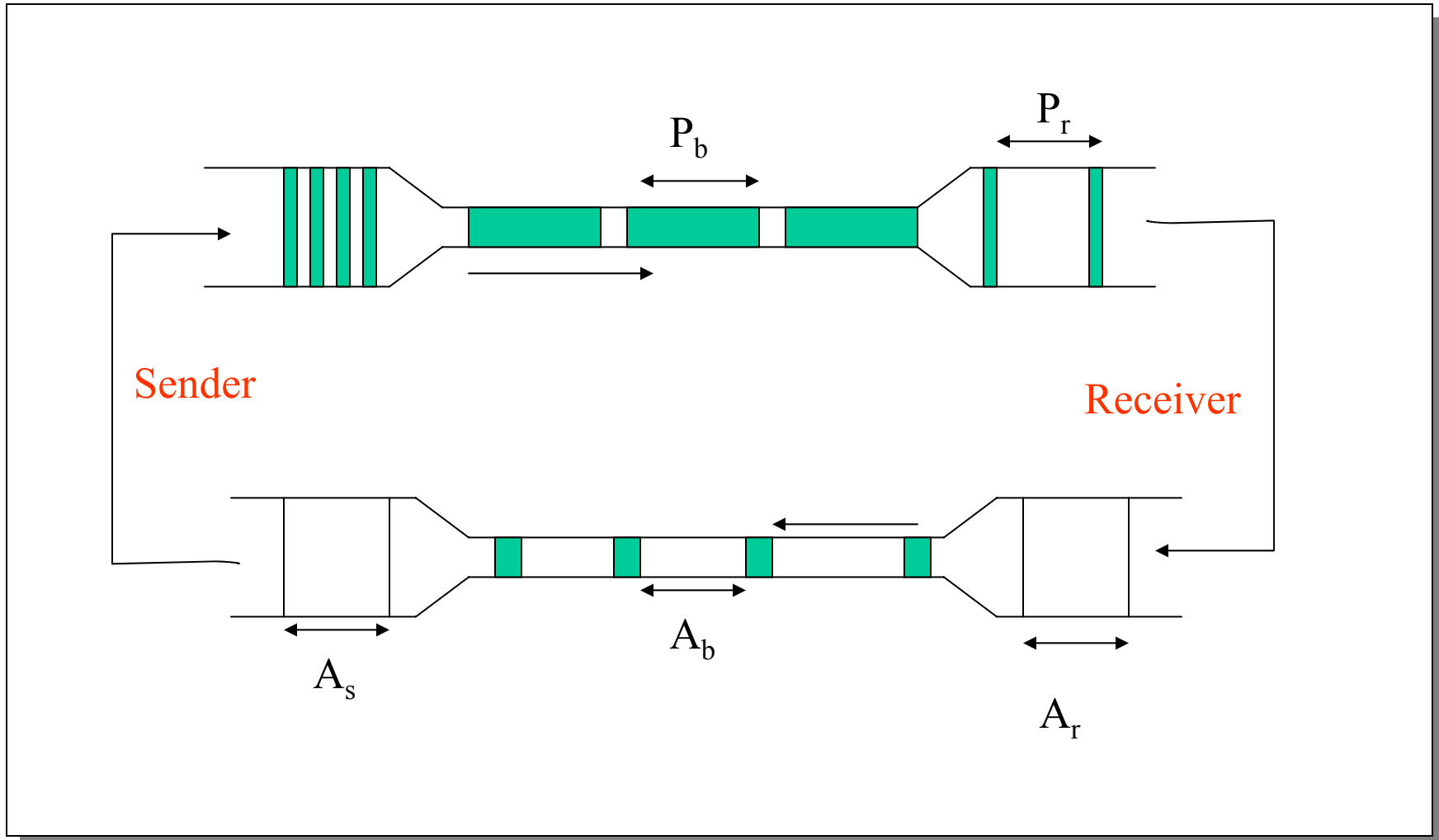
# TCP congestion control - solutions

- ❑ Reaching equilibrium
  - Slow start
- ❑ Eliminates spurious retransmissions
  - Accurate RTO estimation
  - Fast retransmit
- ❑ Adapting to resource availability
  - Congestion avoidance

# TCP congestion control basics

- ❑ Keep a congestion window, cwnd
  - Denotes how much network is able to absorb
- ❑ Sender's maximum window:
  - Min (advertised receiver window, cwnd)
- ❑ Sender's actual window:
  - Max window - unacknowledged segments
- ❑ If we have large actual window, should we send data in one shot?
  - No, use acks to clock sending new data

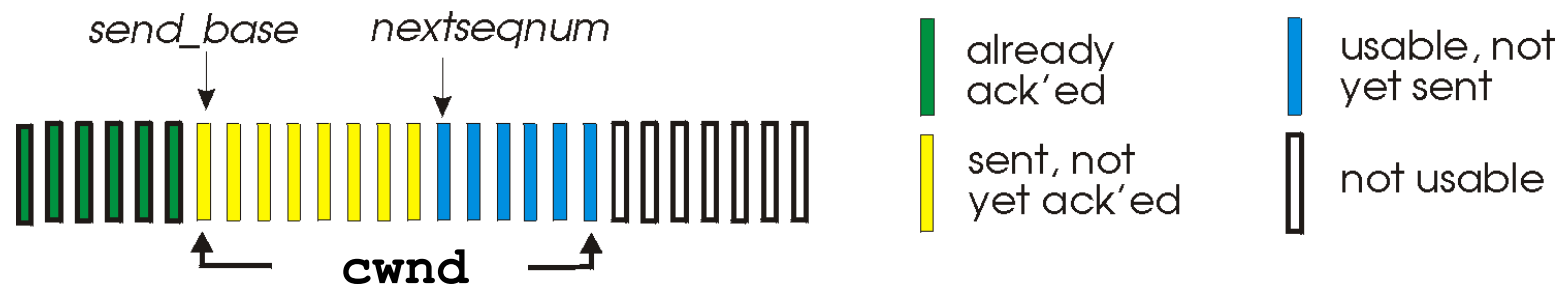
# Self-clocking





# TCP congestion control

- ❑ End-end control (no network assistance)
- ❑ TCP throughput limited by rcvr window (flow control)
- ❑ Transmission rate limited by congestion window size, **cwnd**, over segments:



- ❑  $w$  segments, each with MSS bytes sent in one RTT:

$$\text{throughput} = \frac{w * \text{MSS}}{\text{RTT}} \text{ Bytes/sec}$$

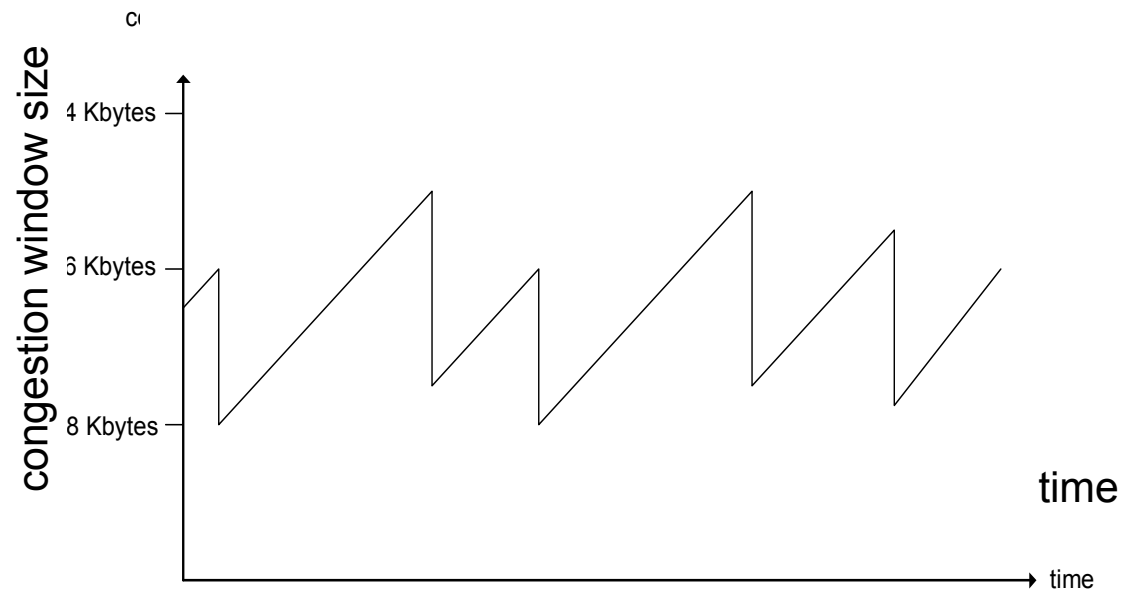
# TCP congestion control:

- “**Probing**” for usable bandwidth:
  - **Ideally**: transmit as fast as possible (**cwnd** as large as possible) without loss
  - *Increase cwnd* until loss (congestion)
  - Loss: *decrease cwnd*, then begin probing (increasing) again
- Two “phases”
  - **Slow start**
  - **Congestion avoidance**
- Important variables:
  - **Cwnd** (**congwin**)
  - **Threshold**: defines threshold between two slow start phase, congestion control phase

# TCP congestion control: additive increase, multiplicative decrease

- *Approach:* Increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *Additive increase:* Increase **cwnd** by 1 MSS every RTT until loss detected
  - *Multiplicative decrease:* Cut **cwnd** in half after loss

Saw tooth behavior: probing for bandwidth



# TCP congestion control: Details

- Sender limits transmission:

$$\text{LastByteSent} - \text{LastByteAked} \leq \text{cwnd}$$

- Roughly,

$$\text{rate} = \frac{\text{cwnd}}{\text{RTT}} \text{ Bytes/sec}$$

- **Cwnd** is dynamic, function of perceived network congestion

## How does sender perceive congestion?

- Loss event = timeout *or* 3 duplicate acks
- TCP sender reduces rate (**cwnd**) after loss event

## Three mechanisms:

- AIMD
- Slow start
- Conservative after timeout events