Last Lecture: TCP

1. *Multiplexing and Demultiplexing*
2. *Byte-stream service*
   - Stream of bytes sent and received, not stream of packets
3. *Reliable data transfer*
   - A combination of go-back-N and selective repeat, and performance tuning heuristics
4. *Connection management* ✔
   - Connection establishment and tear down
5. *Flow control* ✔
   - Prevent sender from overflowing receiver
6. *Congestion control* (later)

This Lecture: TCP

1. *Multiplexing and Demultiplexing*
2. *Byte-stream service*
   - Stream of bytes sent and received, not stream of packets
3. *Reliable data transfer*
   - A combination of go-back-N and selective repeat, and performance tuning heuristics
4. *Connection management*
   - Connection establishment and tear down
5. *Flow control*
   - Prevent sender from overflowing receiver
6. *Congestion control* ✔
   - General principles & How TCP does it

What is Congestion Control?

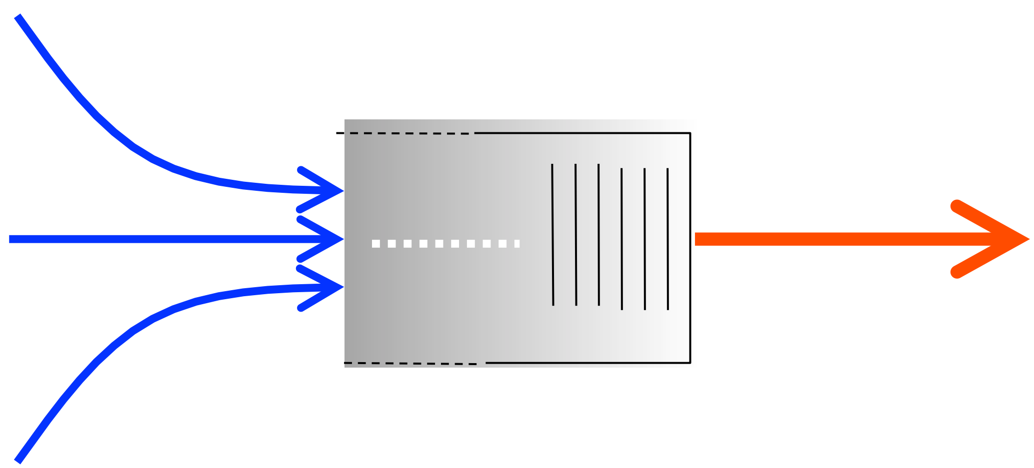
- *Flow control*:
  - Keep *one fast sender* from overflowing a slow receiver
- *Congestion control:*
  - Keep *a set of senders* from overloading *the network* (or, more precisely, some routers in the network)

Congestion is Unavoidable

Especially After A Soccer Match Win

Network Congestion Is Unavoidable

- Two packets arrive at the same time
  - The router can only transmit one
  - … and either *buffer* or drop the other
- If many packets arrive in short period of time
  - The router cannot keep up with arriving traffic
  - … and the buffer may eventually *overflow*



What Happens When Congestion Occurs?

If nothing is done to alleviate the problem, then

- The router(s) has to *drop* packets
  - More packets are lost
  - Retransmissions inject more packets into the network
  - *… more packets are lost*
- Packets not dropped wait in *long queue*
  - End-to-end delay gets large
  - Senders time out, retransmit more packets
  - *… queues become full, more packets dropped/lost*

This is a top-10 problem in Computer Networking

- Reliable data transfer is another

Ways to Deal With Congestion

- *Reservations*, like in circuit switching
  - Pre-allocate bandwidths along paths
  - Requires negotiation before sending packets

- *Pricing*
  - Don't drop packets for the high-bidders
  - Requires a payment model

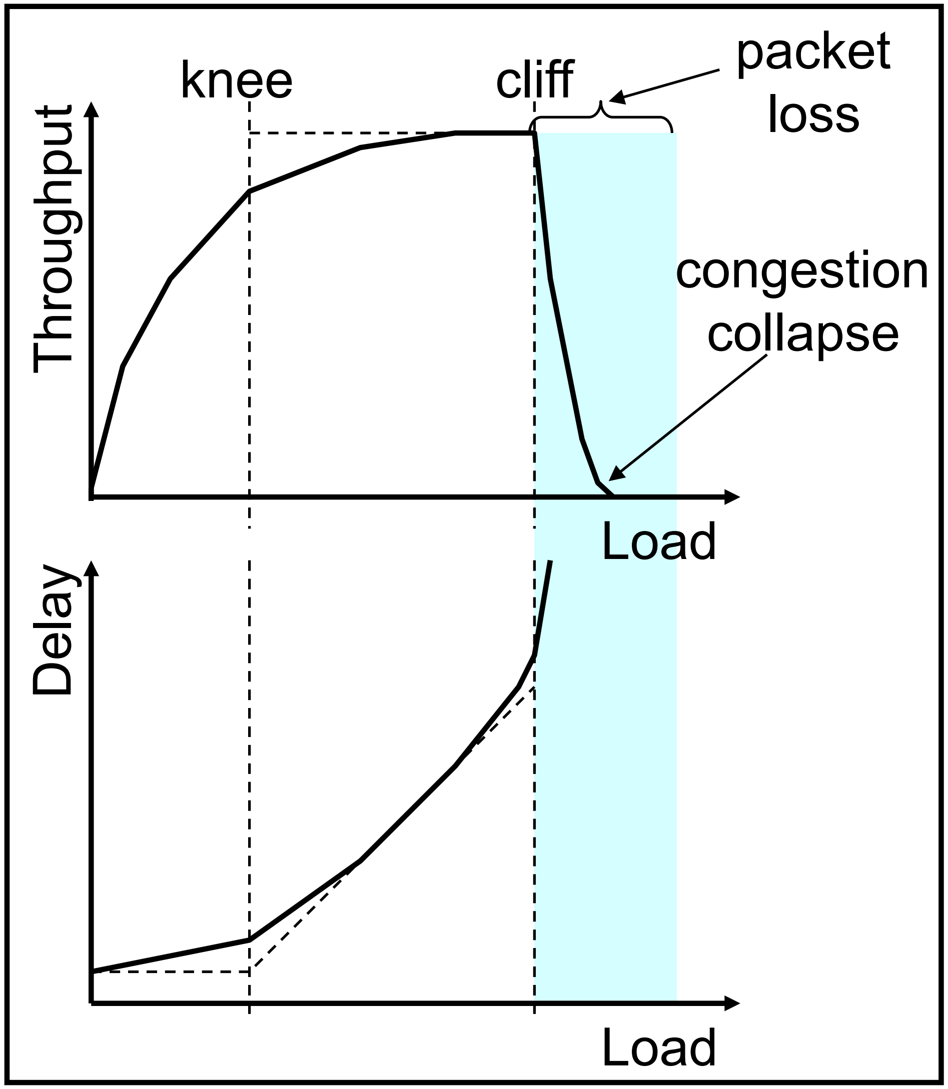
- *Dynamic adjustment* (TCP)
  - Every sender infers the level of congestion
  - And adapts its sending rate, for the greater good

Dynamic Adjustment Is Difficult, Intuitively

- Knee – point after which
  - throughput increases very slowly
  - delay increases fast
- Cliff – point after which
  - throughput starts to decrease very fast to zero
  - delay approaches infinity

- Congestion Avoidance
  - Try to stay left of knee
- Congestion Control
  - Try to stay left of cliff

Problem: knee and cliff are not fixed from the point of view of a single TCP connection



Congestion Control Is Difficult, Technically

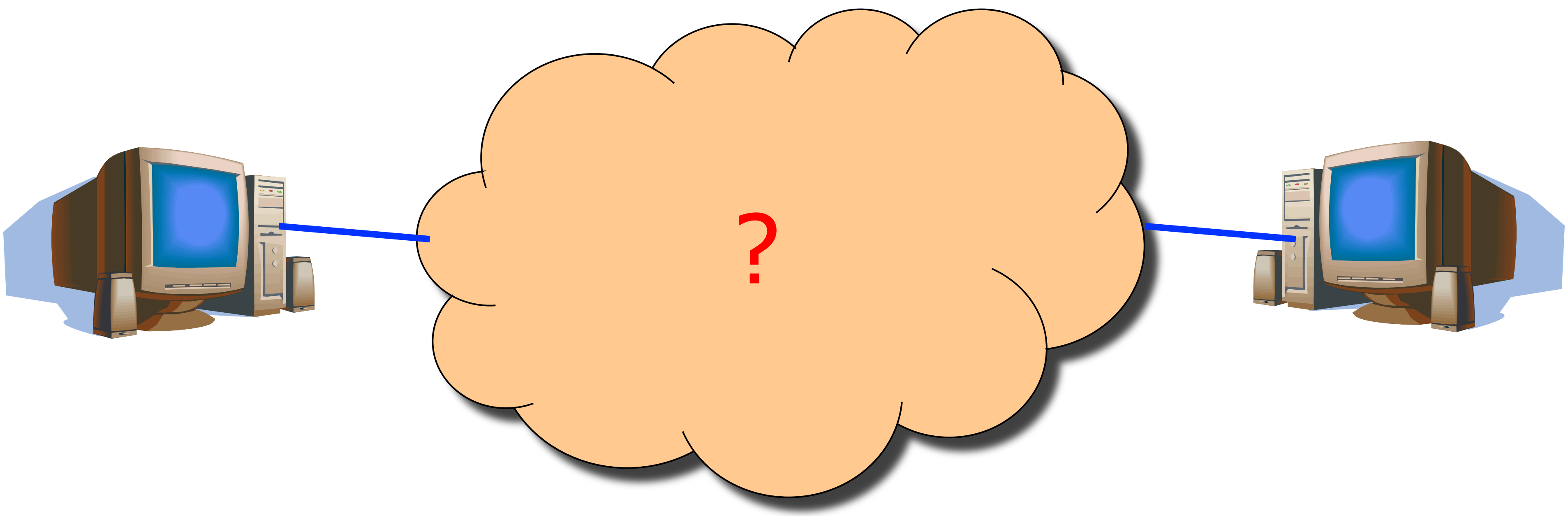
1. How does a sender know that there's congestion?
   - Network tells sender: "I'm congested"
   - Sender detects by itself
2. How to adapt when congestion occur?
   - The congestion "point" is definitely moving
   - The adaptation strategy "moves" that point too!!!
   - How should a new TCP connection behave?
3. What is "*the*" performance objective?
   - Maximize *my* throughput?
   - Maximize the *total* throughput?
   - Maximize fairness? (Extremely tricky to define)

1. How Sender Knows About Congestion?

- *Explicitly*, network says so
  - Called *network-assisted congestion control*
  - Examples of routers' feedback:
    - Data bit(s) indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
    - Explicit rate sender should send at
- *Implicitly*, sender *infers* it
  - Called *end-to-end congestion control*
  - Sender observes traffic, make intelligence inference
  - No feedback from network
  - Basically the approach taken by TCP
- Some combination of the two

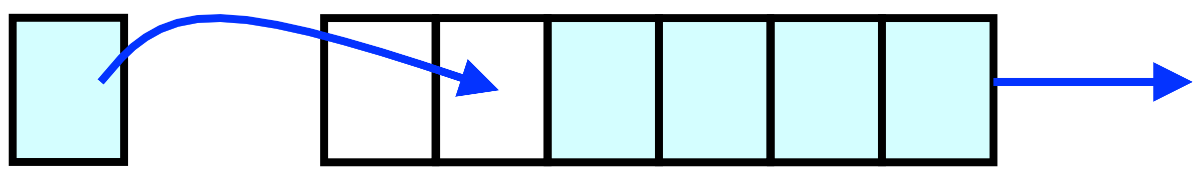
How Can TCP Infer Congestion?

- What does end hosts see?
- What can end hosts change to alleviate the problem?

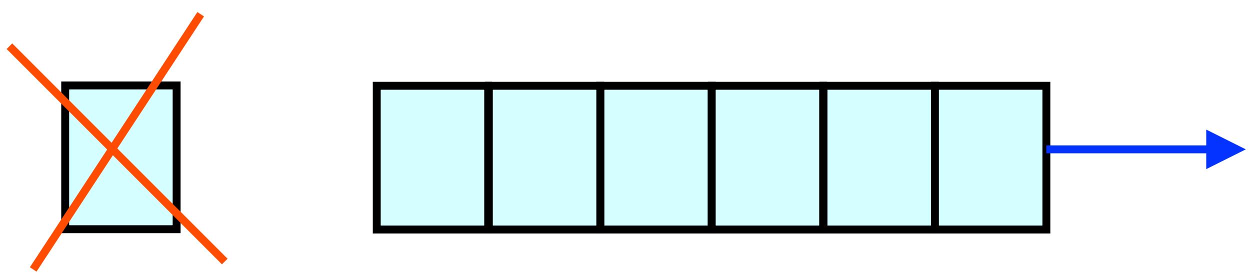


Where Congestion Happens: Links

- Packets queued waiting to get on links
- Access to the bandwidth: FIFO queue
  - Packets (typically) transmitted in the order they arrive



- Access to the buffer space: drop-tail queuing
  - If the queue is full, drop the incoming packet



How Congestion Looks to End Hosts?

- Packets experience “long” *delays*
- Packets are *lost* (dropped at routers)

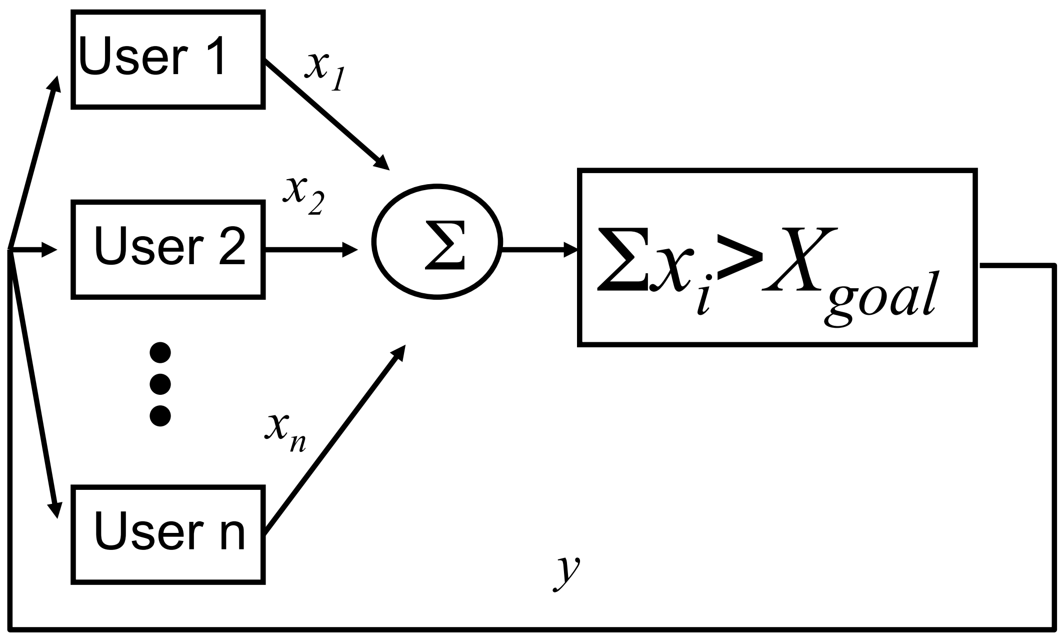
- How does TCP detect delays and losses?
  - *Delays*: RTT estimation, timeouts
  - *Losses*: timeouts, duplicate acknowledgements

2. How To Adapt When Congestion Occurs?

- Intuitively,
  - *Slow down* when congestion detected
  - *Speed up* when there's room to breath
  - Try to avoid too much oscillation, making network condition *unstable*
  - Try to be "*fair*" to other users of the network too!

- Main questions:
  - How slow should slowing down be?
  - How fast should speeding up be?
  - Need a good *control system model* to answer questions

Control System Model [Chiu & Jain 1989]



- Simple, yet powerful model
- Explicit binary signal of congestion

Possible Choices

$$x_i(t+1) = \begin{Bmatrix} a_I + b_I x_i(t) & increase \\ a_D + b_D x_i(t) & decrease \end{Bmatrix}$$

- Increase = “speeding up”, decrease = “slowing down”
- Multiplicative = “fast”, additive = “slow”

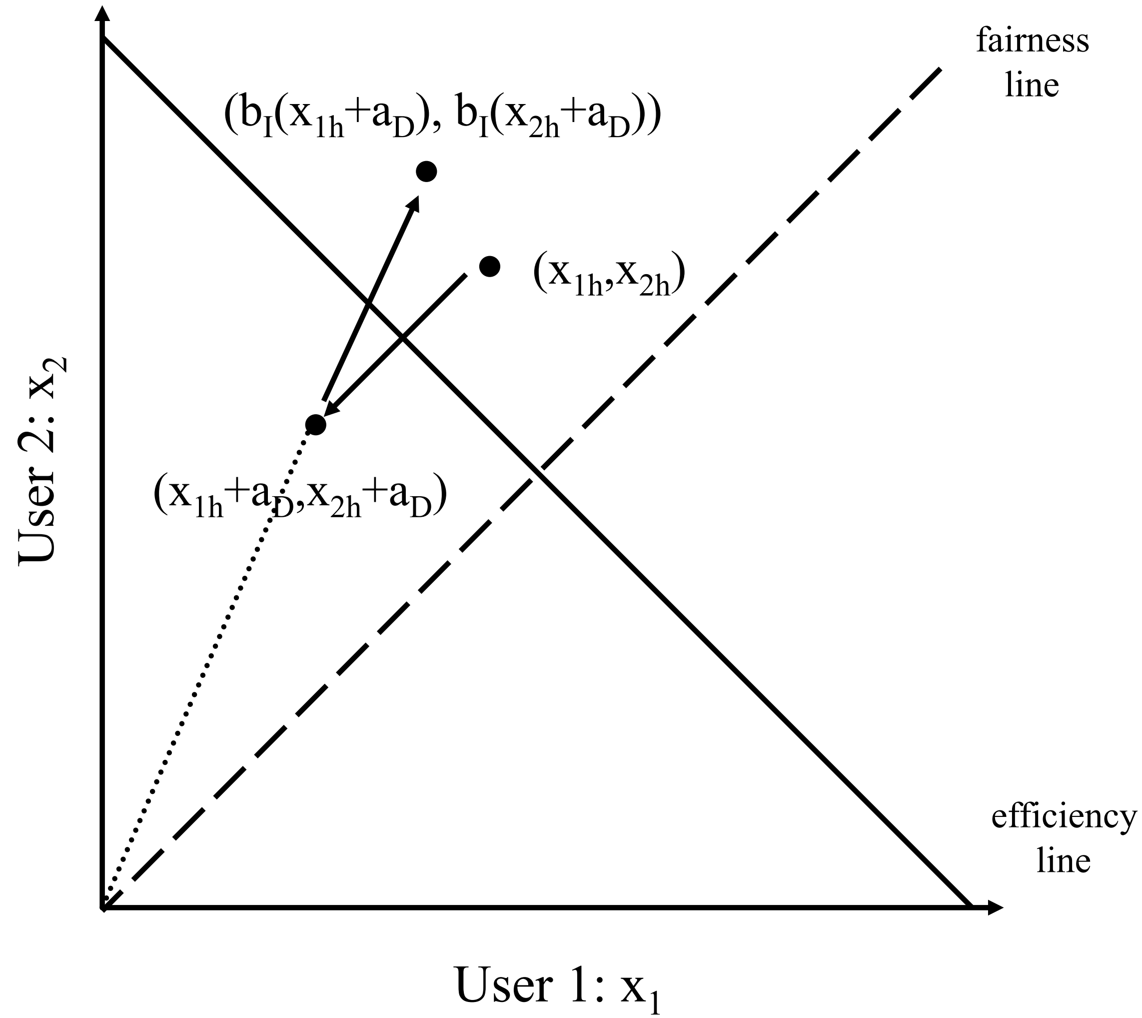
- Multiplicative increase, additive decrease
  - $a_I=0$, $b_I>1$, $a_D<0$, $b_D=1$
- Additive increase, additive decrease
  - $a_I>0$, $b_I=1$, $a_D<0$, $b_D=1$
- Multiplicative increase, multiplicative decrease
  - $a_I=0$, $b_I>1$, $a_D=0$, $0<b_D<1$
- Additive increase, multiplicative decrease
  - $a_I>0$, $b_I=1$, $a_D=0$, $0<b_D<1$
- **Question**: which one?

Multiplicative Increase, Additive Decrease

- Fixed point at

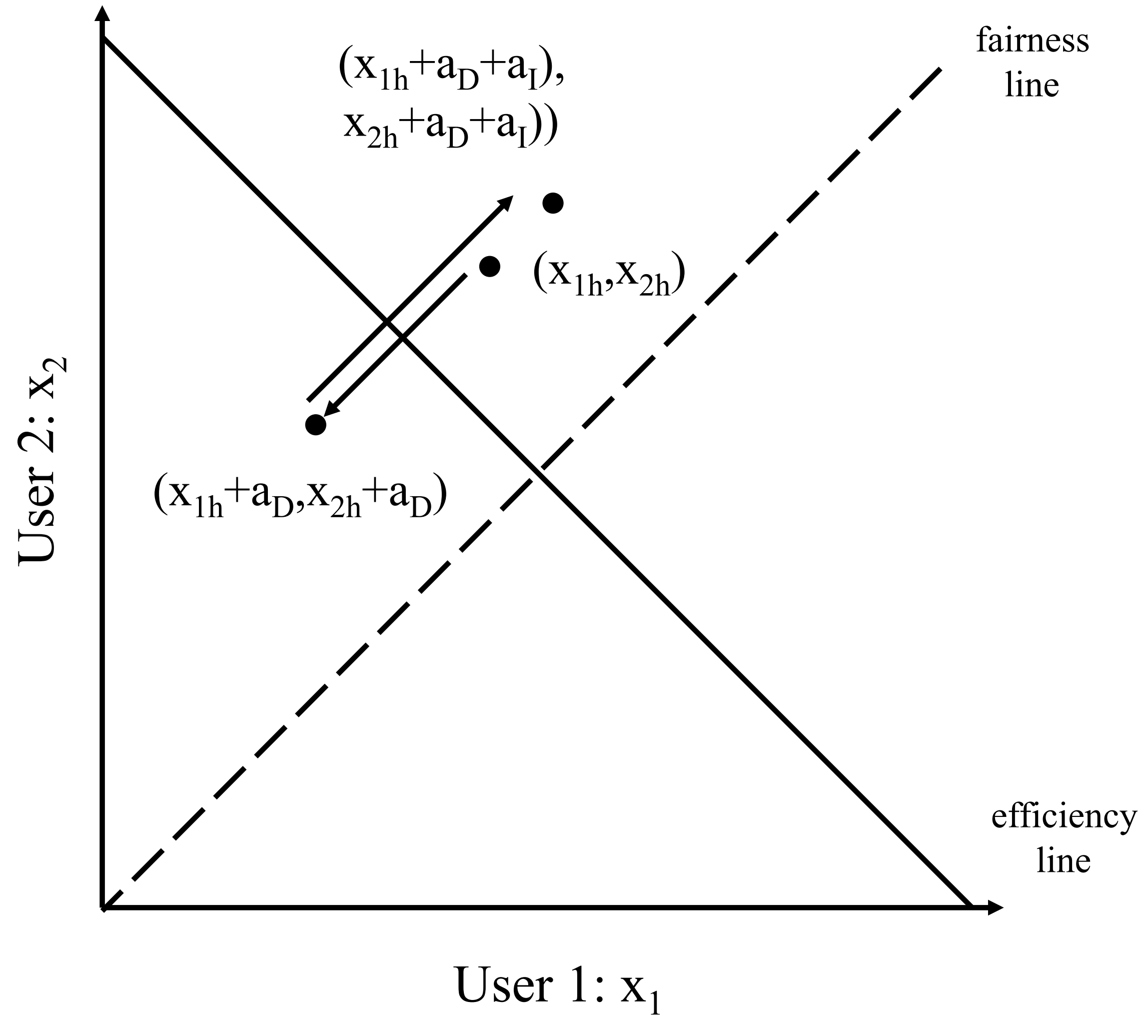
$$x_{1h} = x_{2h} = \frac{b_I a_D}{1 - b_I}$$

Fixed point is unstable!



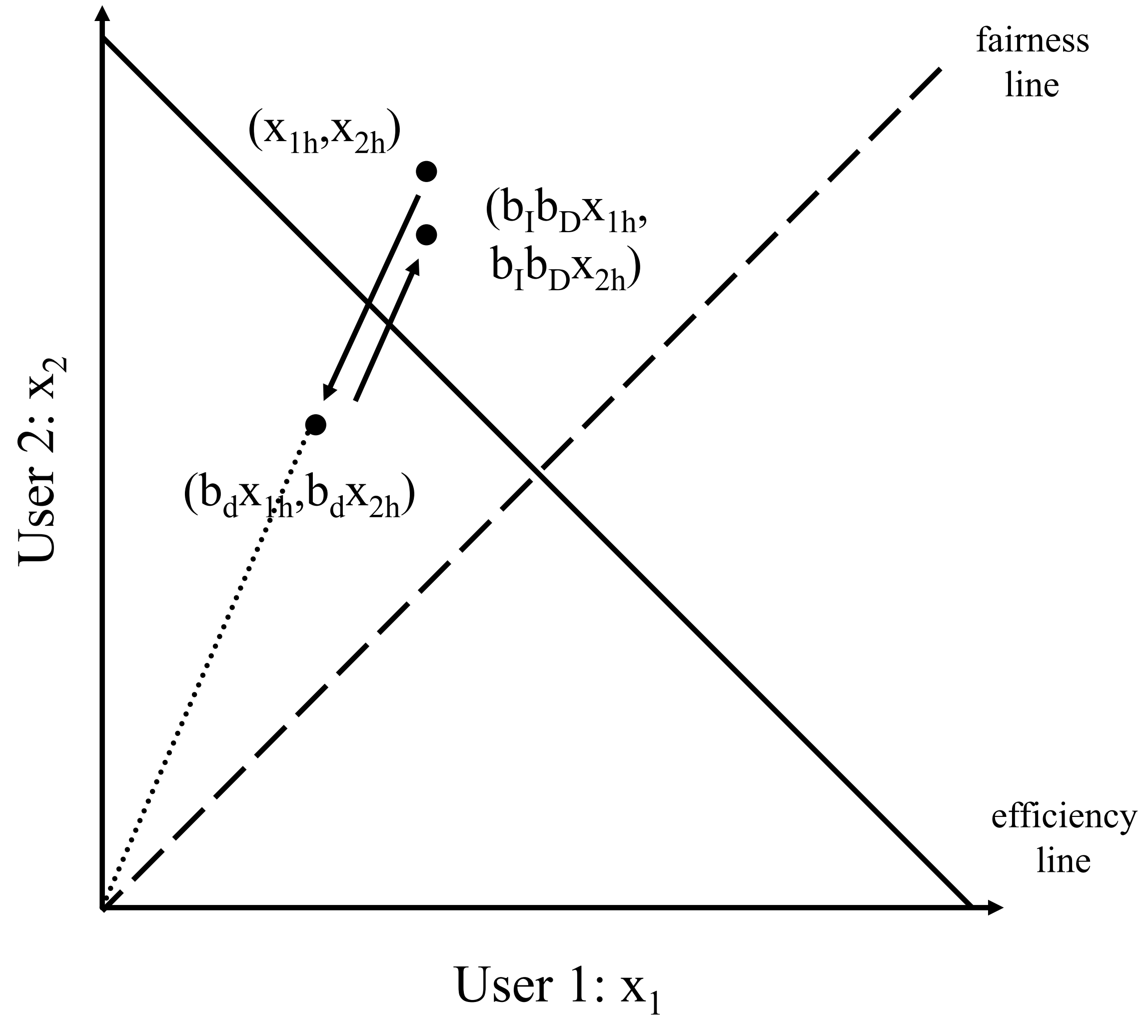
Additive Increase, Additive Decrease

- Reaches stable cycle, but does not converge to fairness



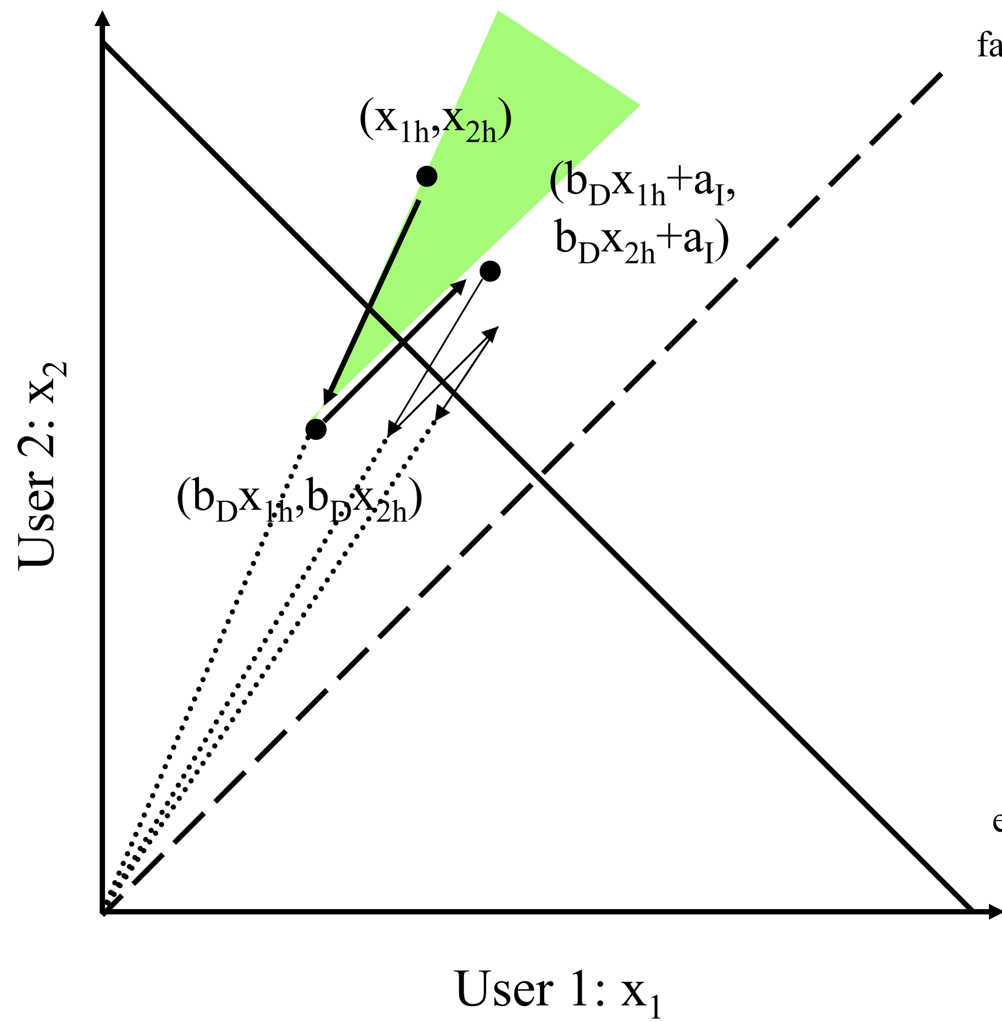
Multiplicative Increase, Multiplicative Decrease

- Converges to stable cycle, but is not fair



Additive Increase, Multiplicative Decrease

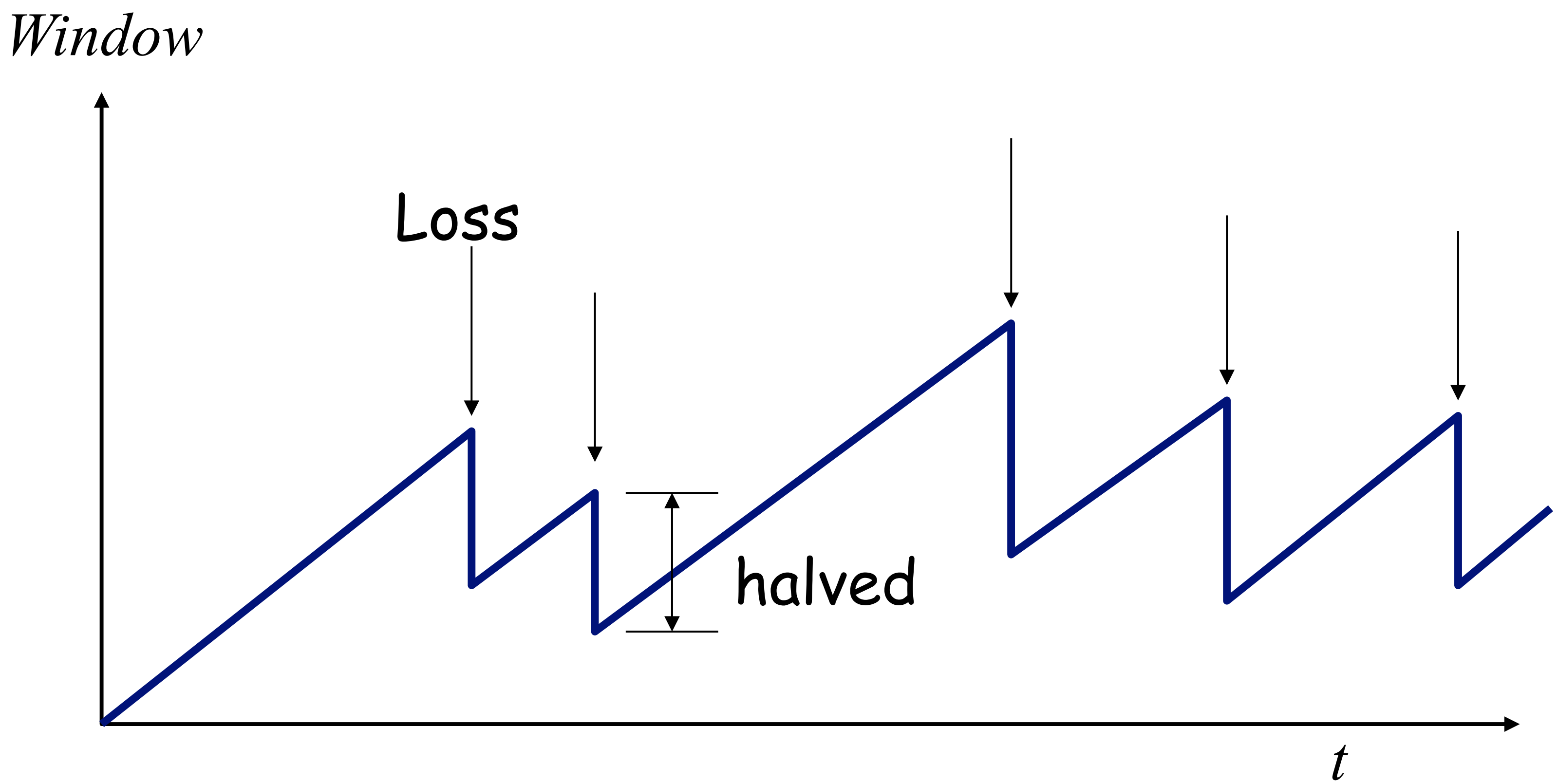
- Converges to *stable and fair* cycle



Note About Network Modeling

- Critical to understanding complex systems
  - [CJ89] model relevant after 20 years, 10^6x increase of bandwidth, 1000x increase in number of users
- Criteria for good models
  - Two conflicting goals: reality and simplicity
  - Realistic, complex model → too hard to understand, too limited in applicability
  - Unrealistic, simple model → can be misleading
- *AIMD seems right*, just have to implement it

AIMD Leads To TCP's "Saw-Tooth"

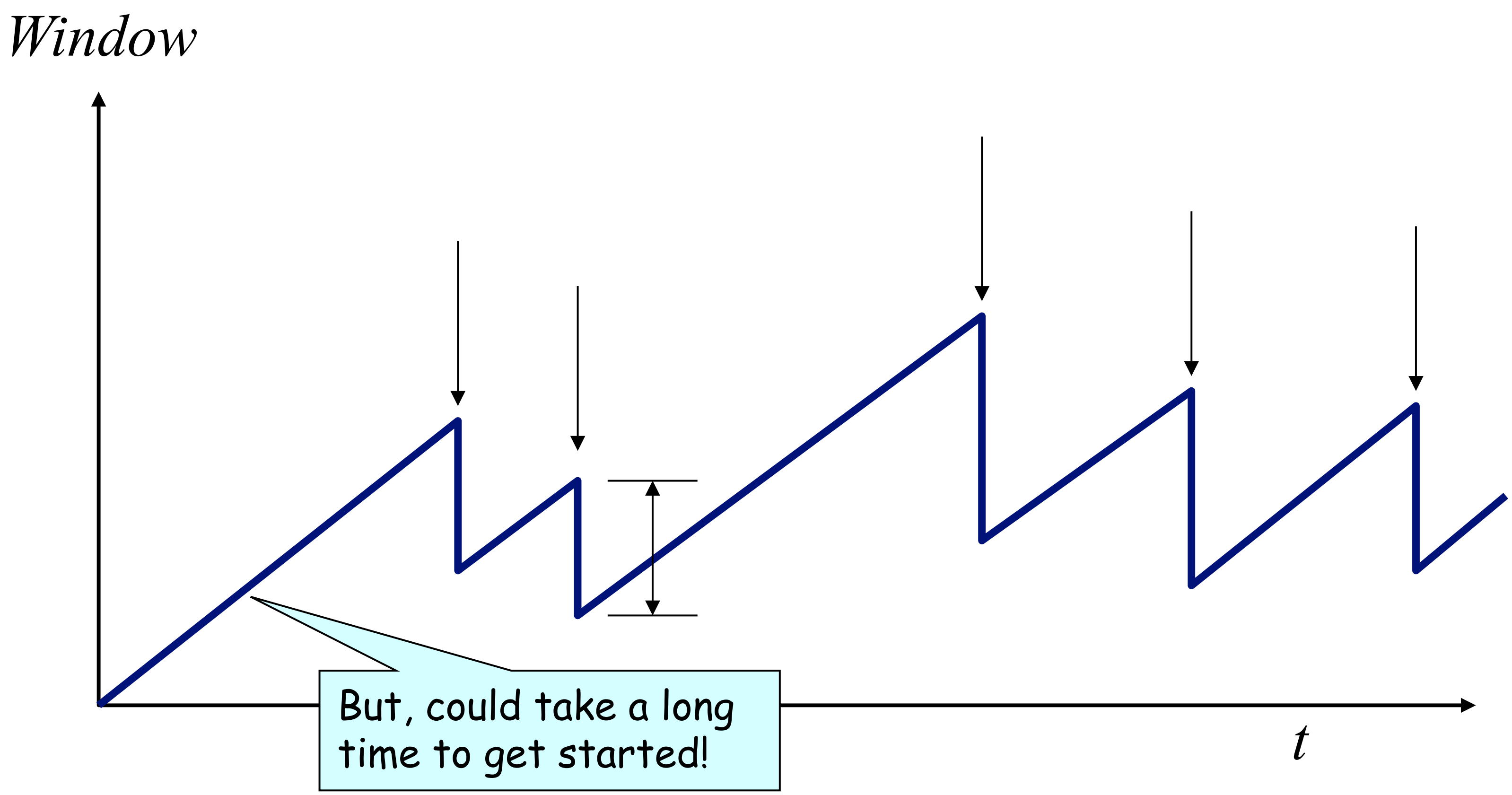


TCP's Congestion Control

- *CWind*: (an additional variable)
  - Similar to FWind, but used for congestion control
  - The actual window size is *min*(CWind, FWind)
    - But let's assume FWind is really large for now
- *ssthresh*:
  - Rough estimate of knee point
- Two main mechanisms:
  - AIMD
    - Slowly increase CWind in good times (additive increase)
    - Cut CWind in half in bad times (multiplicative decrease)
  - Slow start
    - Quickly get to ssthresh before additive increase

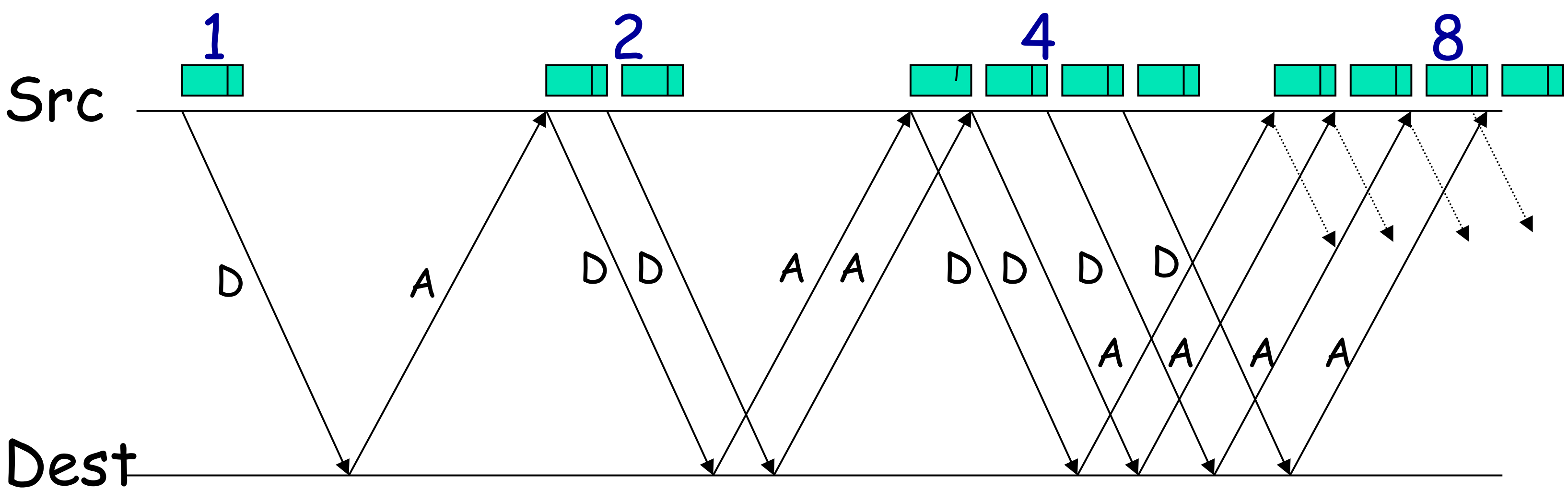
Slow Start Phase

- Applied when a new flow starts
- Or an existing flow "re-starts"

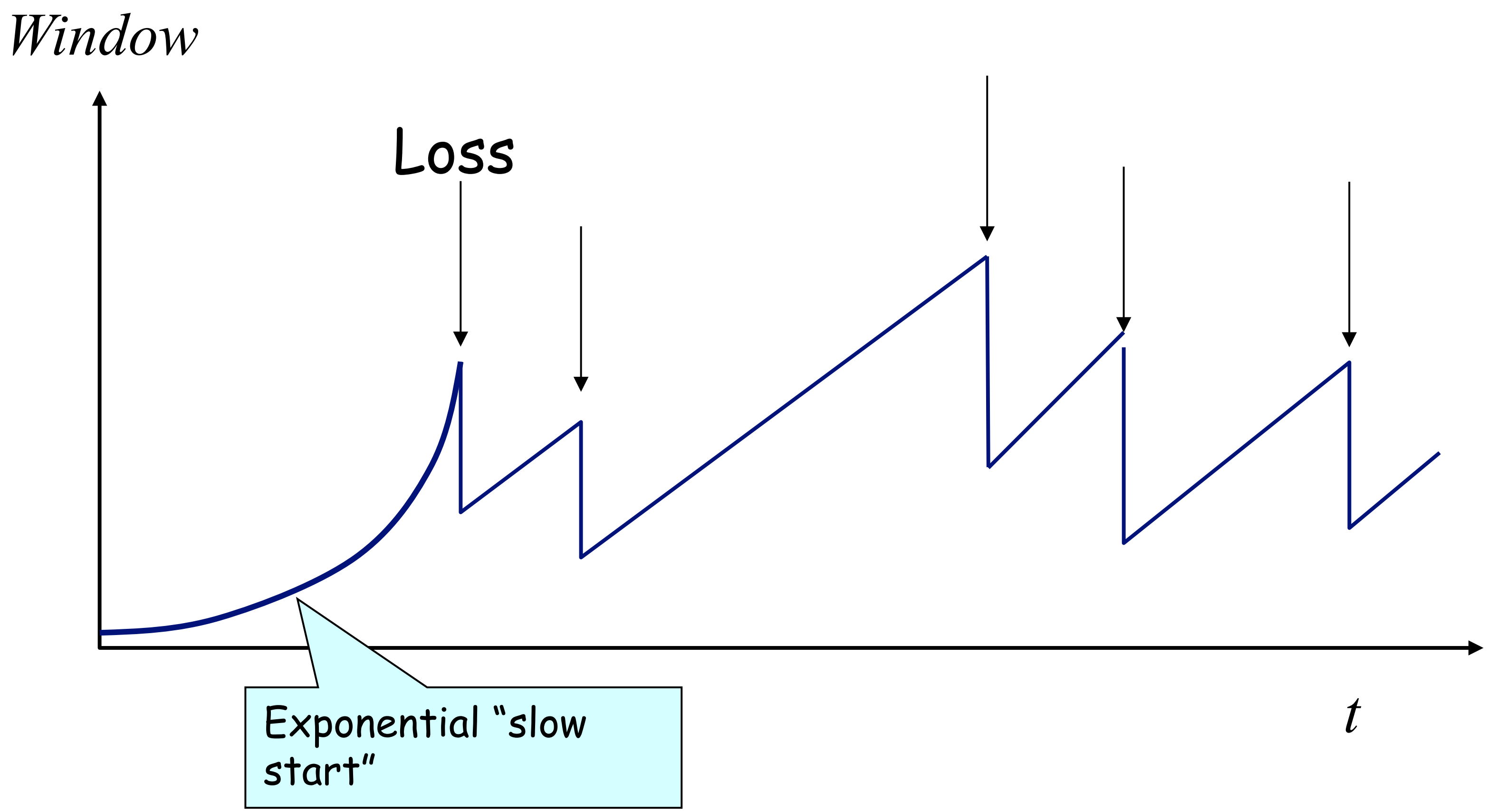


How Slow Start Works

- Initially set CWind = 1 MSS
- Increase CWind by 1 MSS for each ACK received
- *Slow start is exponentially fast!*
  - CWind is *doubled* for each RTT

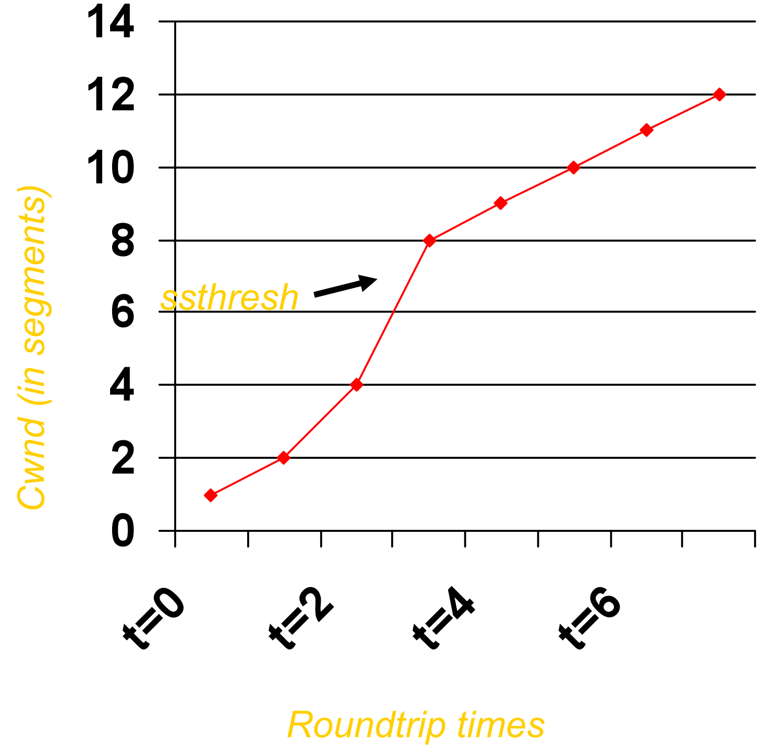


Slide Start and The TCP Saw-Tooth



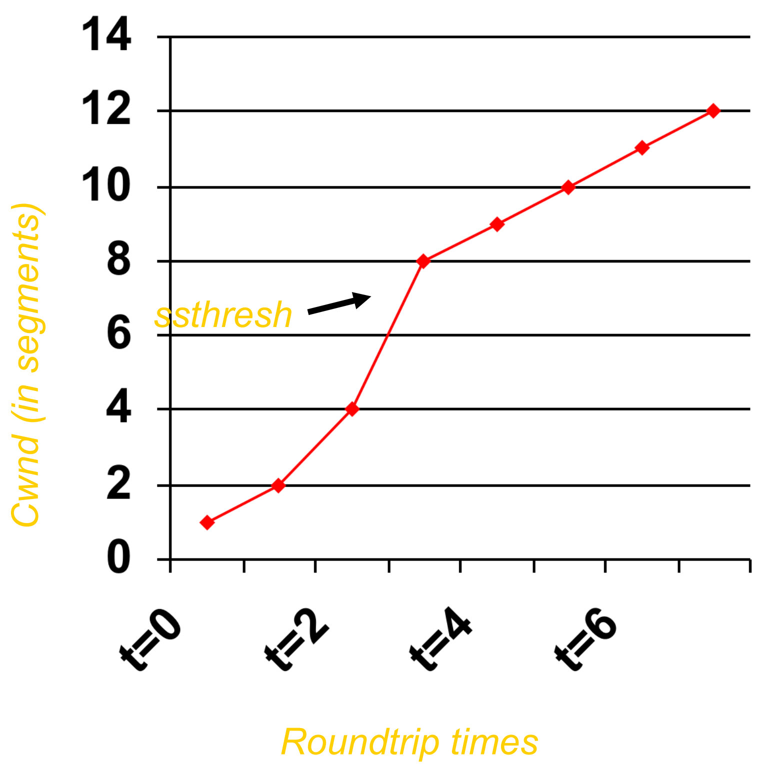
Next Comes Congestion Avoidance Phase

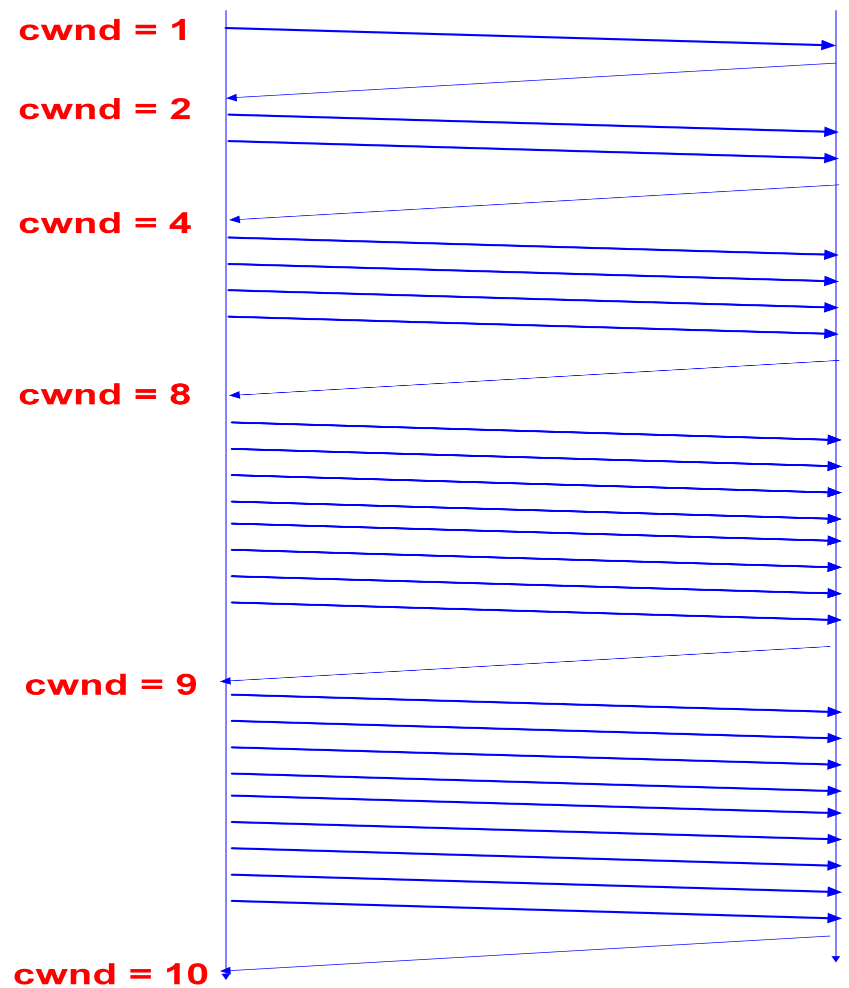
- Slow start until *CWind* reaches *ssthresh*
  - Initially, ssthresh = ∞
- Now begins the *additive increase* step
  - Increase CWind by 1 MSS *per* RTT (instead of doubling it)
  - Technically, for every newly received ACK
    CWind = CWind + MSS * (MSS/CWind)



Slow Start/Congestion Avoidance Example

- Assume that *ssthresh = 8*



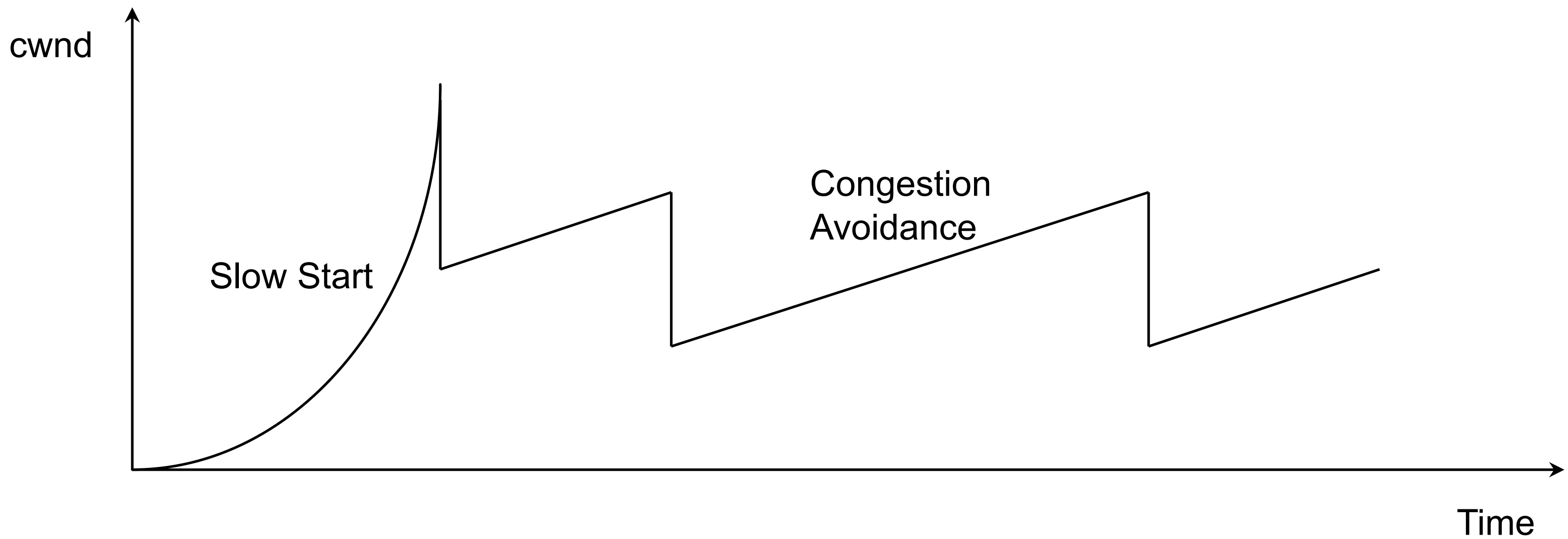


When Congestion Occurs

Sender detects by either timeout or duplicate ACKs

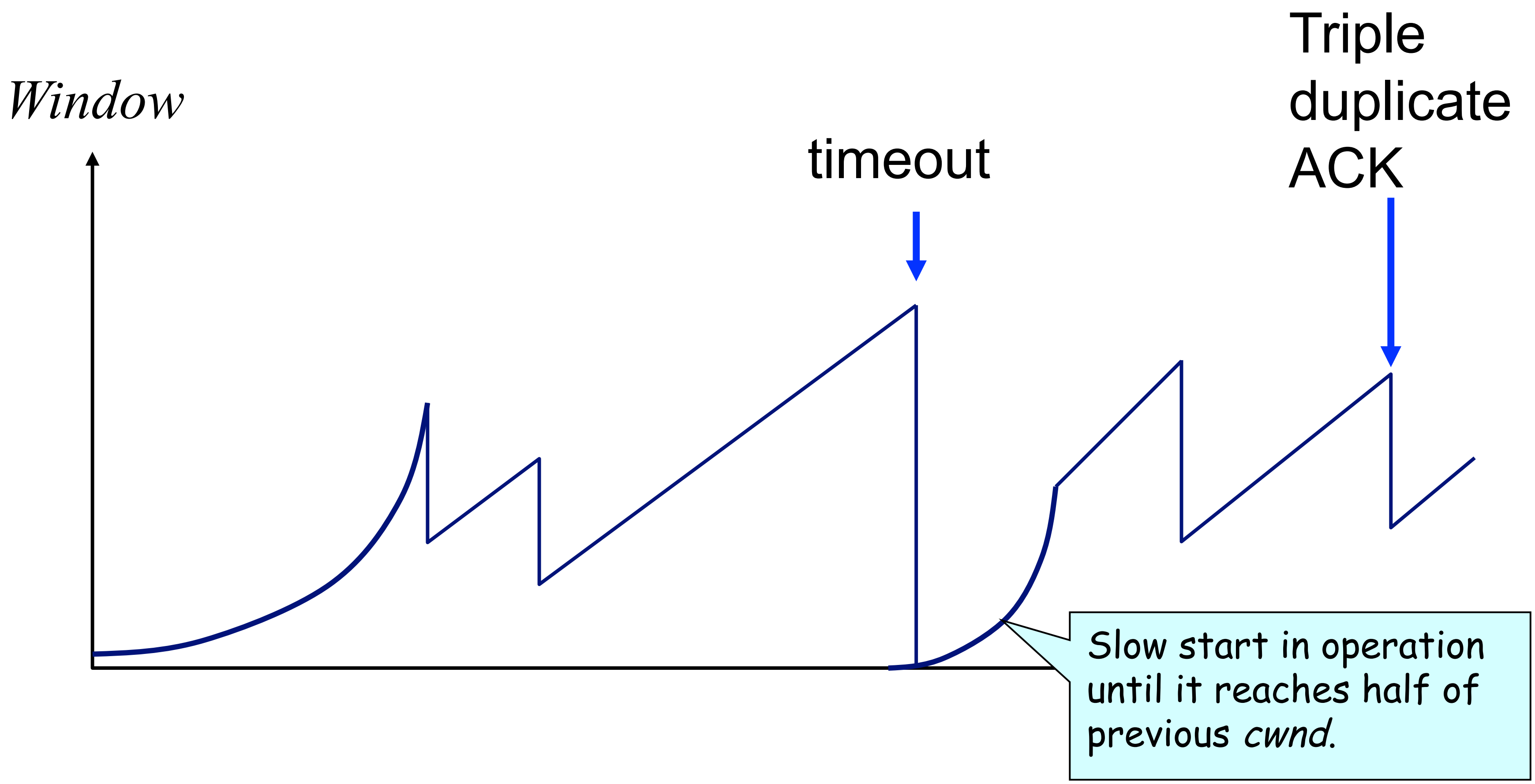
- *Timeout*
  - Packet n is lost and detected via a timeout
  - E.g., because all packets in flight were lost
  - Network condition very very bad
  - ssthresh = CWind/2; CWind = 1; begin slow start
- *Triple duplicate ACK*
  - Packet n is lost, but packets n+1, n+2, etc. arrive
  - Receiver sends duplicate acknowledgments
  - Network condition bad, but not too bad
  - Ssthresh = CWind/2; CWind = ssthresh; begin congestion avoidance (this is called "*fast recovery*", TCP Reno & later)

Fast Retransmit and Fast Recovery



- They often go together (TCP Reno and later)
  - Retransmit right away after 3 duplicated acks
  - Then start the fast recovery phase
- Hope: at steady state, *CWind* oscillates around the optimal window size.

The Saw-Tooth Again



Summary of TCP Congestion Control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to “Congestion Avoidance”	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to “Congestion Avoidance”	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to “Slow Start”	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

TCP Throughput

- What's the average throughput of TCP as a function of window size and RTT?
  - Ignore slow start
- Let W be the window size when loss occurs.
- When window is W, throughput is W/RTT
- Just after loss, window drops to W/2, throughput to W/2RTT.
- Average throughout: .75 W/RTT

Future of TCP: "Long-Fat Pipes" Problem

- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size W = 83,333 in-flight segments
- Throughput in terms of loss rate (homework 2):

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜ $L = 2 \cdot 10^{-10}$ *Wow*
- Need new versions of TCP for high-speed

Many Other Extensions to TCP

- Selective acknowledgements: *TCP SACK*
- Explicit congestion notification: *ECN*
- Delay-based congestion avoidance: *TCP Vegas*
- Discriminating between congestion losses and other losses: *cross-layer signaling and guesses*
- Randomized drops (*RED*) and other router-based mechanisms