Data Networks
UdS and IMPRS-CS

Lecture 15: Congestion Control II

Critical Features of TCP

- Increase rate until packet loss
  - What's the problem?
- Use loss as indication of congestion
  - What's the problem?
- AIMD mechanism oscillates around proper rate
  - What's the problem?
- Relies on AIMD behavior of end hosts
  - What's the problem?
- Slow start to probe for initial rate
  - What's the problem?

Some Answers

- Increase rate until packet loss
  - Drives network into congestion
  - High queuing delay, inefficient
- Use loss as indication of congestion
  - Cannot distinguish congestion from packet corruption
- AIMD mechanism oscillates around proper rate
  - Rate is not smooth
    - Bad for streaming applications (e.g. video)
  - Inefficient utilization
- Relies on AIMD behavior of end hosts for fairness
  - People can cheat (not use AIMD)
  - People can open many parallel connections
- Slow start to probe for initial rate
  - Bad for short lived flows (e.g. most Web transfers, a lot of Internet traffic is web transfer)

Why Bad for Short Lived Flows?

- Typical Web transfer ~ 10 KB
- That translates into ~ 10 packets
- That is a Web transfer is typically finished before slow-start is finished probing for bandwidth

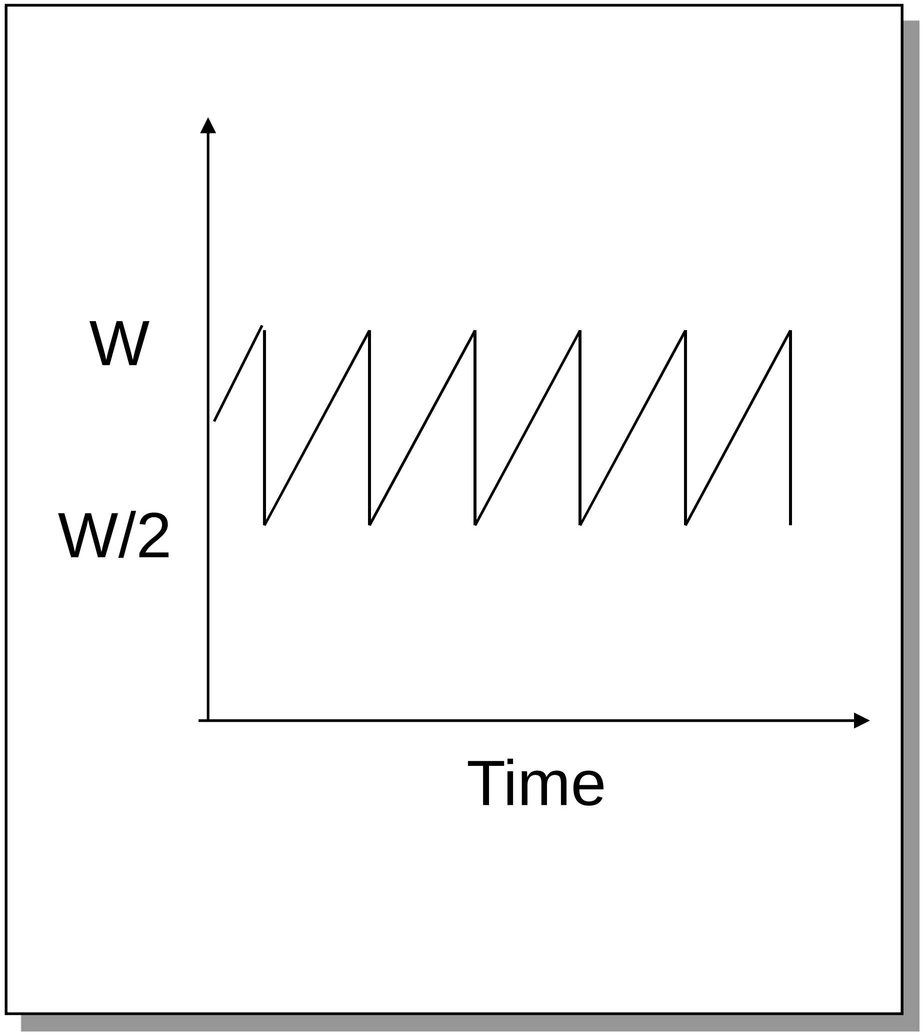
- Moreover, a small number of packet loss among 10 packets can be blow up the overall transfer time by a large amount
  - Potentially timeout, retransmit, etc
  - Transfer time is small, so any delay is very significant

Many Experimental Ideas Out There

- We’ll discuss a few
- Smoothing transmission rate
  - Equation-based congestion control
- Router assisted mechanisms:
  - Random Early Detection (RED)
  - Explicit Congestion Notification (ECN)
    - Idea similar to DECbit scheme in Peterson & Davie

Smoothing Transmission Rate

- TCP has saw tooth behavior, not smooth
- If we can calculate the average rate, then we can just transmit smoothly at the average rate



TCP Model

- Derive an expression for the steady state throughput as a function of
  - RTT
  - Loss probability

- Assumptions
  - Each packet dropped with *iid* probability p

- Methodology: analyze “average” cycle in steady state
  - How many packets are transmitted per cycle?
  - What is the duration of a cycle?

TCP Model

$$\text{throughput } \underset{\substack{\uparrow \\ \boxed{\text{loss probability (drop rate)}}}}{T(p)} = \frac{1/p}{RTT \cdot \frac{1}{2}\sqrt{\frac{8}{3p}}} = \frac{1}{RTT\sqrt{\frac{2}{3}p}}$$

•Note role of RTT. Is it “fair”?

•A “macroscopic” model

•Achieving this throughput is referred to as “TCP Friendly”

Equation-Based CC

- Idea:
  - Forget complicated increase/decrease algorithms
  - Use this equation T(p) directly!
- Approach:
  - measure drop rate (don’t need ACKs for this)
  - send drop rate p to source
  - source sends at rate T(p)
- Good for streaming audio/video that can’t tolerate the high variability of TCP’s sending rate

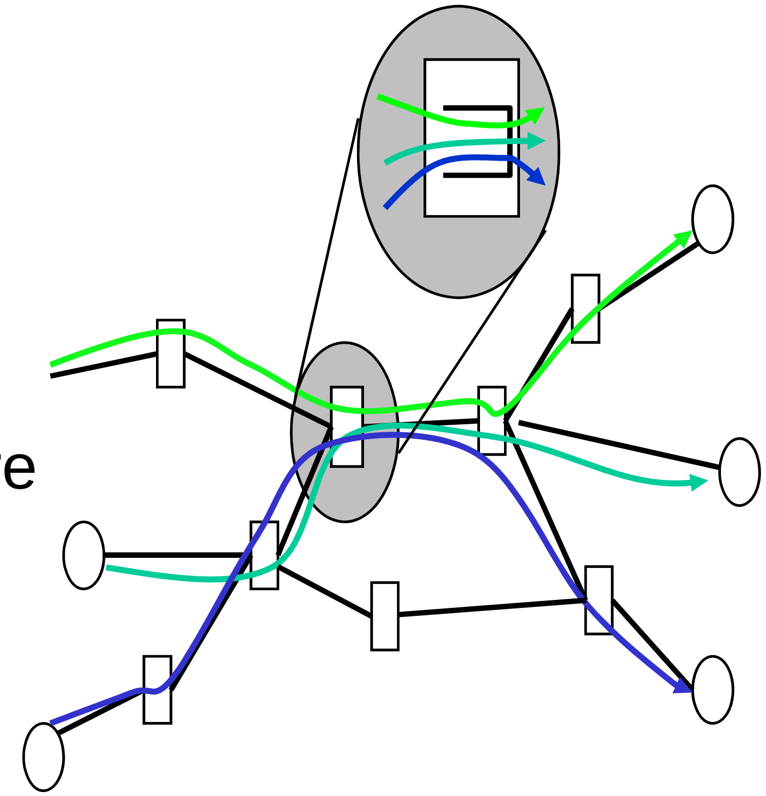
Question!

- Why use the TCP equation?
- Why not use any equation for T(p)?

What can routers do to help?

Traditional Role of Router

- Routers are in middle of action
- Main job is routing and forwarding
- But traditional routers are very passive in terms of congestion control
  - FIFO
  - Drop-tail



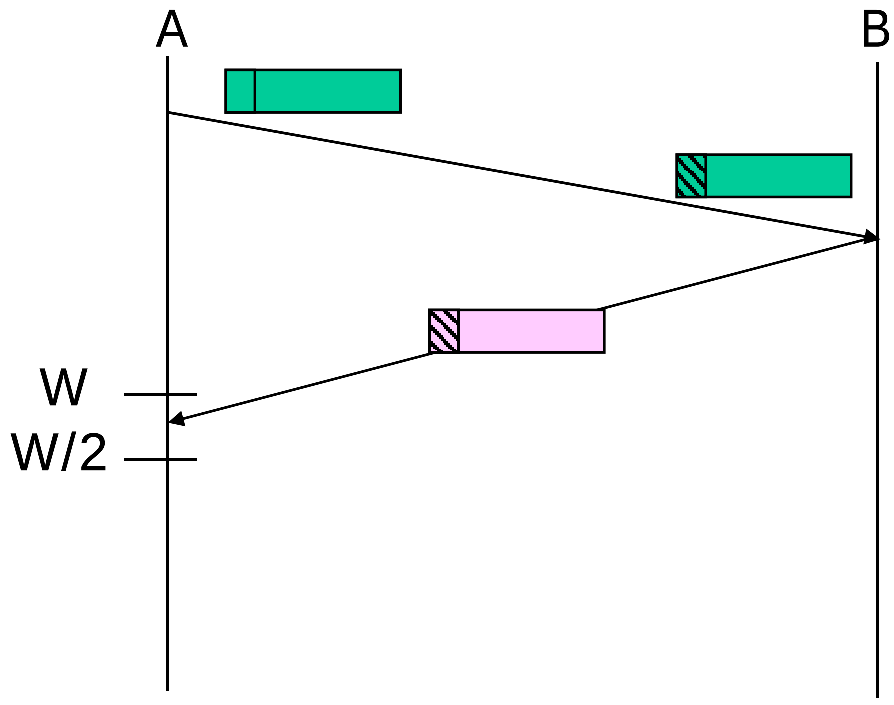
Explicit Congestion Notification

- Rather than drop packets to signal congestion, router can send an explicit signal

- Explicit congestion notification (ECN):
  - instead of optionally dropping packet, router sets a bit in the packet header
  - If data packet has bit set, then ACK has ECN bit set

- Backward compatibility:
  - bit in header indicates if host implements ECN
  - note that not all routers need to implement ECN

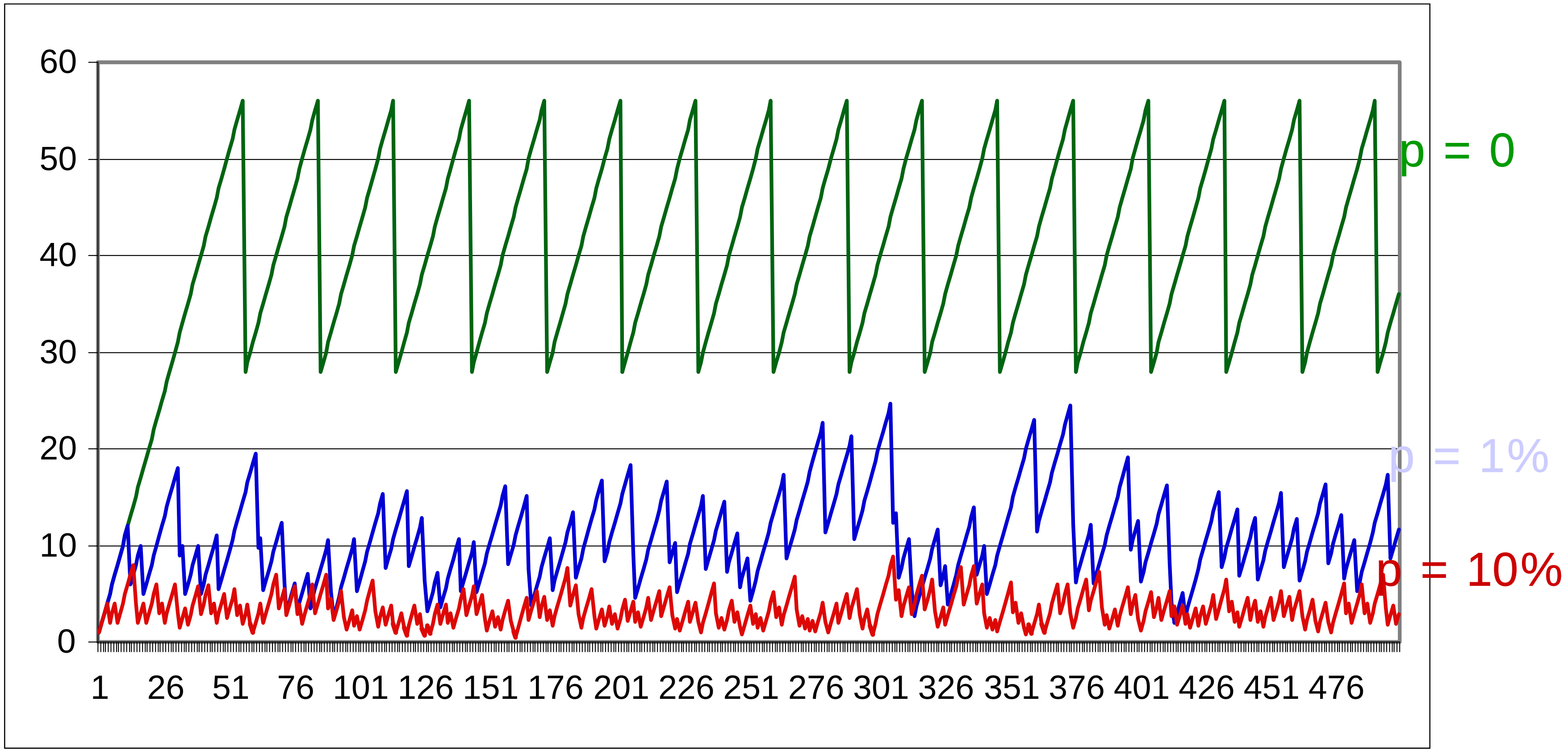
Picture



Lossy Links

- TCP assumes that all losses are due to congestion
- What happens when the link is lossy due to packet corruption (e.g. wireless)?
- Recall that Tput ~ 1/sqrt(p) where p is loss prob.
  - This applies even for non-congestion losses

Example

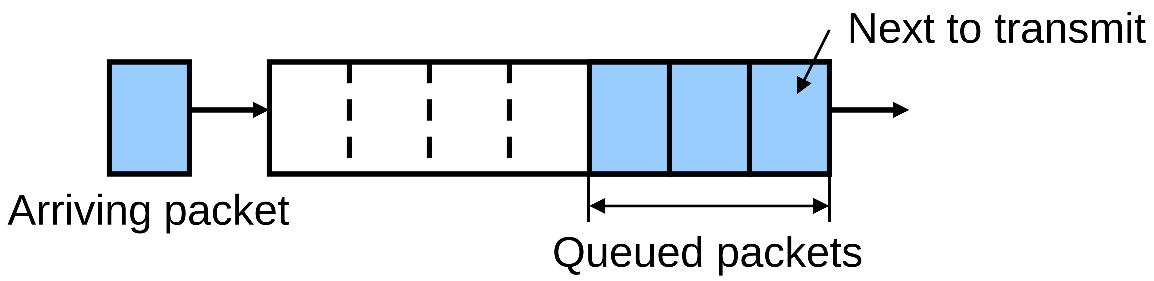

60
50
40
30
20
10
0
1
26
51
76
101
126
151
176
201
226
251
276
301
326
351
376
401
426
451
476
p = 0
p = 1%
p = 10%

ECN Advantages

- No need for retransmitting optionally dropped packets
- No confusion between congestion losses and corruption losses
- RED with ECN works much better than RED alone for short lived flows (e.g. Web transfers)

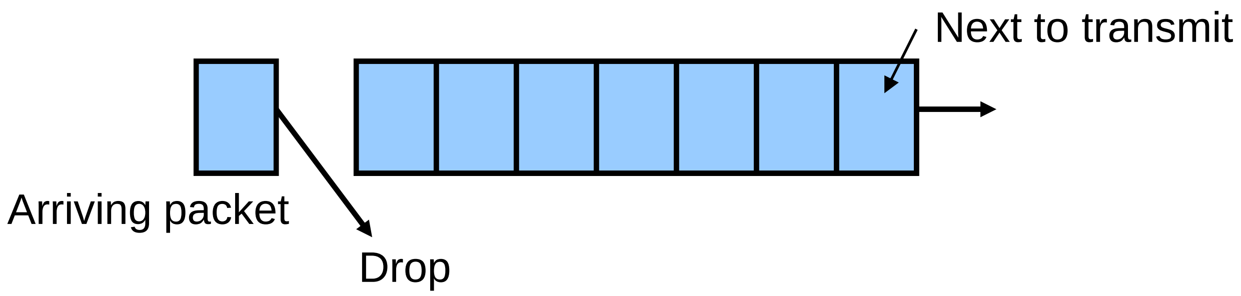
FIFO: First-In First-Out

- Maintain a queue to store all packets
- Send packet at the head of the queue



Tail-drop Buffer Management

- Drop packets only when buffer is full
- Drop arriving packet



Ways Routers Can Help Congestion Control

- Packet scheduling: non-FIFO scheduling
  - Weighted Fair Queuing (discussed before)
  - Needs classification, per flow queuing, and scheduling
  - Can guarantee fairness
  - Quite complex
- Packet dropping:
  - not drop-tail
  - not only when buffer is full
- Congestion signaling

Question!

- Why not use infinite buffers?
  - no packet drops!

Buffer Size

- Small buffers:
  - often drop packets due to bursts
  - but have small delays
- Large buffers:
  - reduce number of packet drops (due to bursts)
  - but increase delays
- Can we have the best of both worlds?

use Delay-Bandwidth-Product

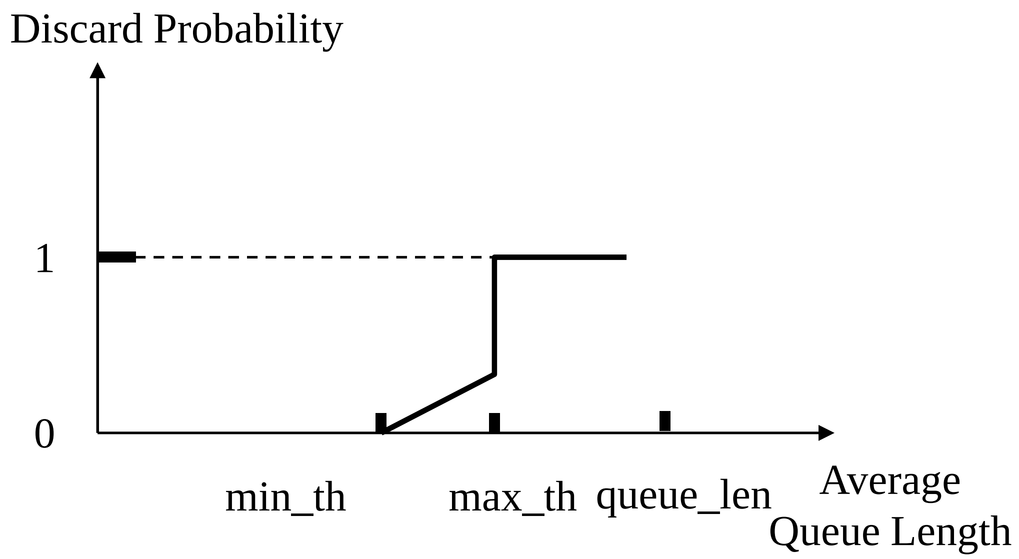
Random Early Detection (RED)

- Basic premise:
  - router should signal congestion when the queue first starts building up (by dropping a packet)
  - but router should give flows time to reduce their sending rates before dropping more packets
  - Note: when RED is coupled with ECN, the router can simply mark a packet instead of dropping it

- Therefore, packet drops (or ECN) should be:
  - early: don’t wait for queue to overflow
  - random: don’t drop (or mark) all packets in burst, but space drops (markings) out

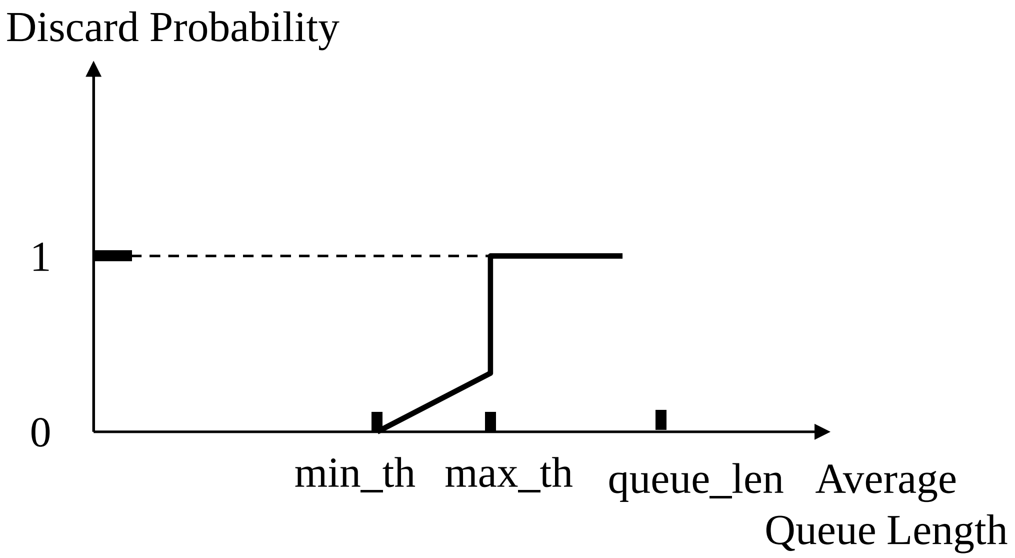
RED

- FIFO scheduling
- Buffer management:
  - Probabilistically discard (or ECN mark) packets
  - Probability is computed as a function of average queue length (why average?)



RED (cont'd)

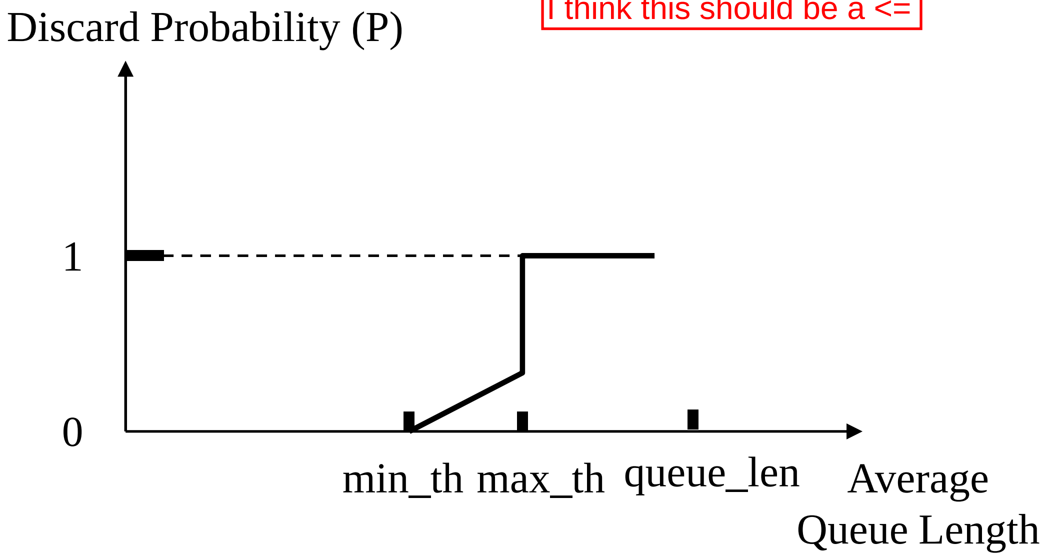
- min_th – minimum threshold
- max_th – maximum threshold
- avg_len – average queue length
  - avg_len = (1-w)*avg_len + w*sample_len



RED (cont'd)

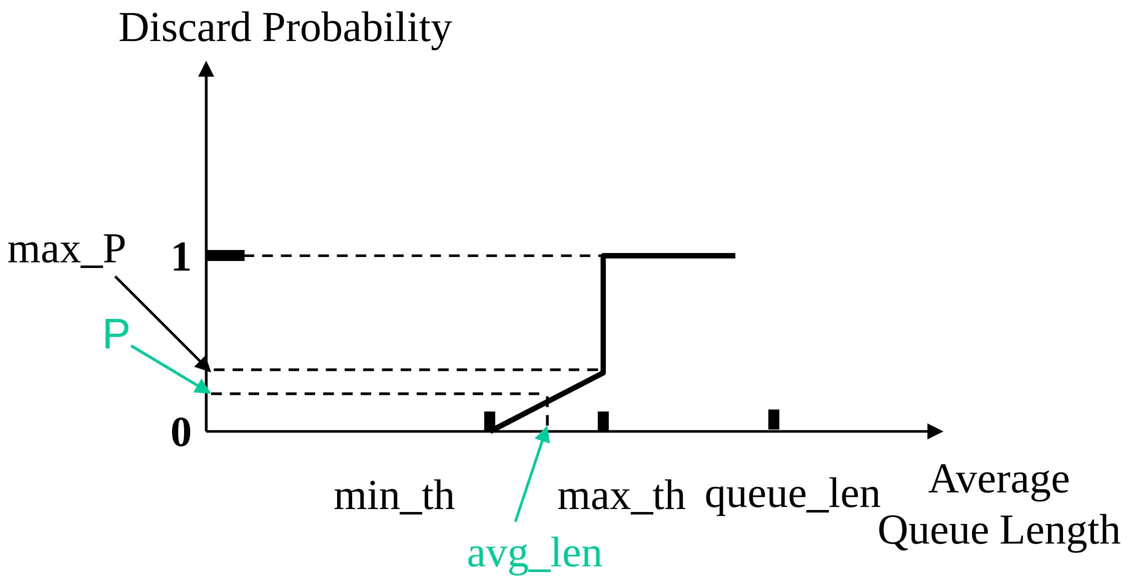
- If (avg_len < min_th) → enqueue packet
- If (avg_len > max_th) → drop (or ECN mark) packet
- If (avg_len >= min_th and avg_len < max_th) → enqueue (or do not mark) packet with probability P

I think this should be a <=

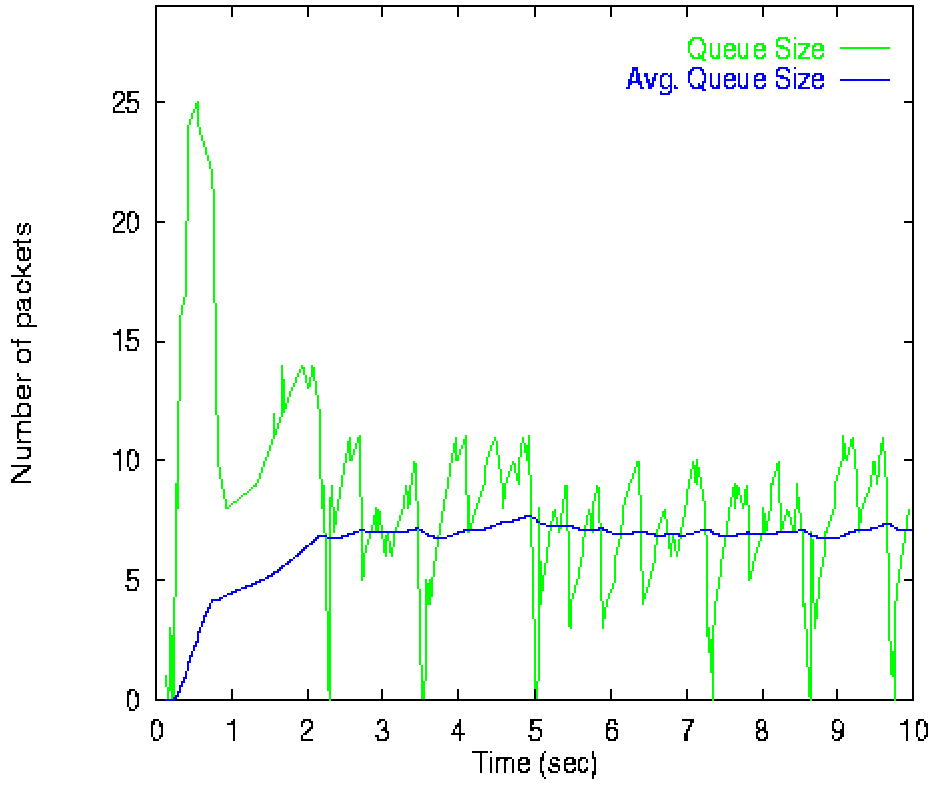


RED (cont'd)

- P = max_P*(avg_len – min_th)/(max_th – min_th)
- Improvements to spread the drops (or ECN markings) (see textbook)



Average vs Instantaneous Queue



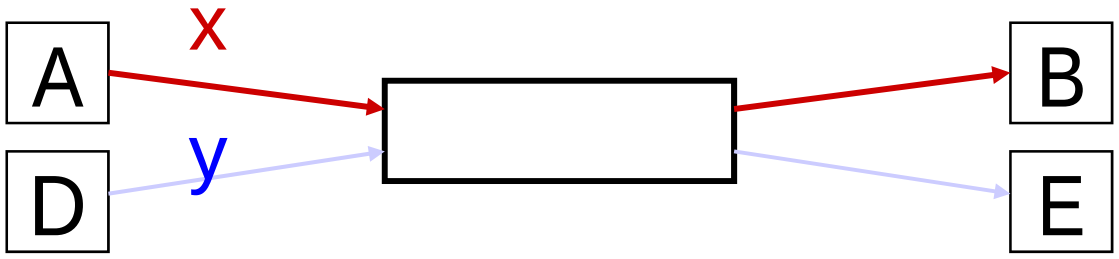
RED Summary

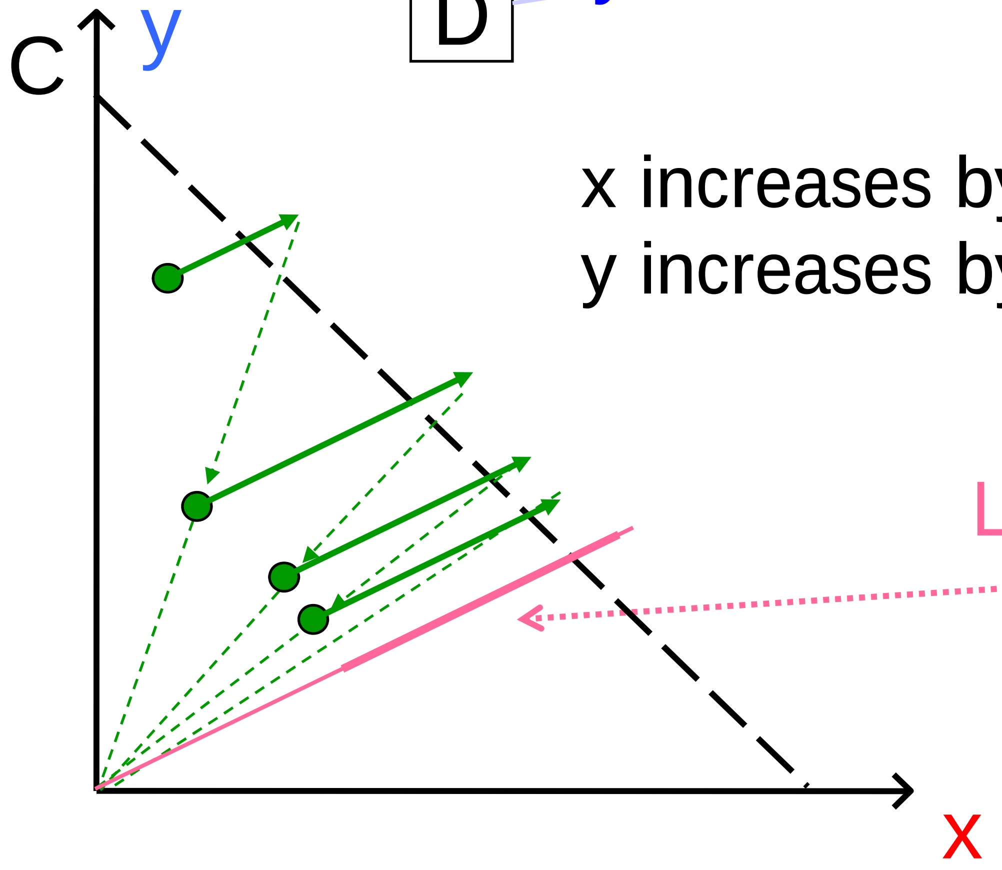
- Basic idea is sound, but does not always work well
  - Basically, dropping packets, early or late is a bad thing
  - So must couple with ECN to mark packets instead of dropping packets
- High network utilization with low delays when flows are long lived
- Average queue length small, but capable of absorbing large bursts
- Many refinements to basic algorithm make it more adaptive (requires less tuning)
- Turns out RED does not work well for short lived flows like Web traffic (which is a big share of traffic on Internet)
  - Dropping packets in an already short lived flow is devastating
  - ECN must be used to make it work well

Cheating

- Many ways to cheat, some ideas:
  - increasing cwnd faster than 1 per RTT
  - using large initial cwnd
  - Opening many connections

Increasing cwnd Faster

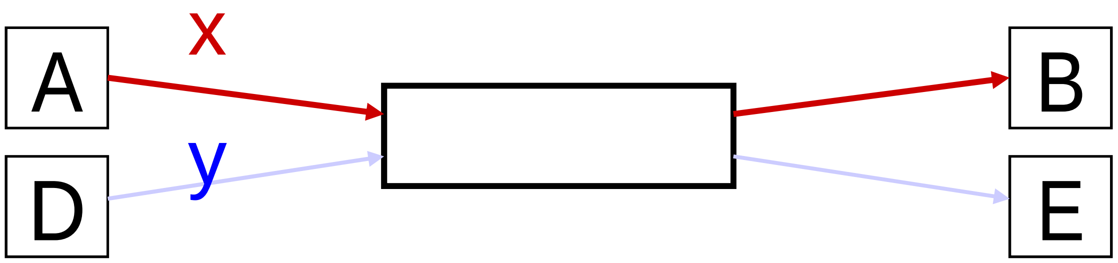


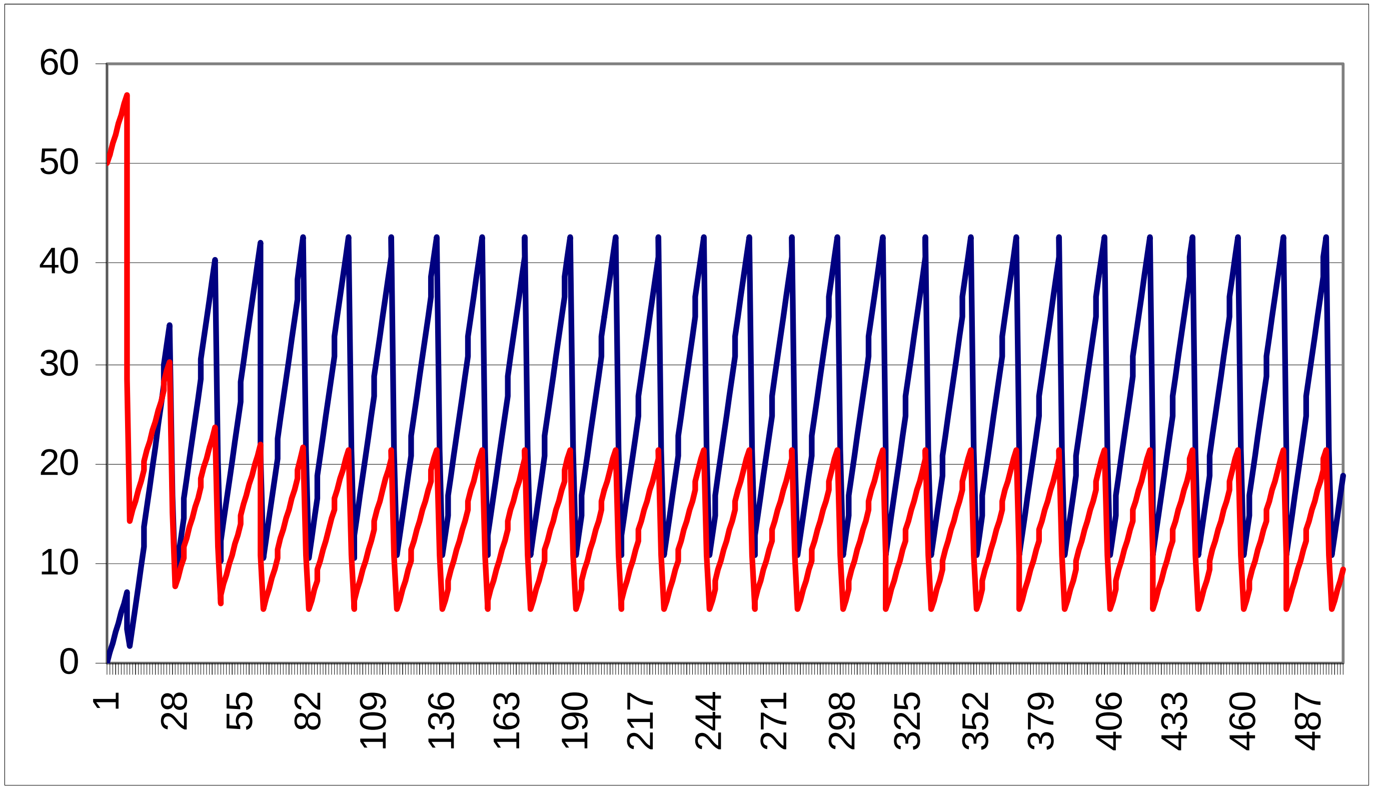


x increases by 2 per RTT
y increases by 1 per RTT

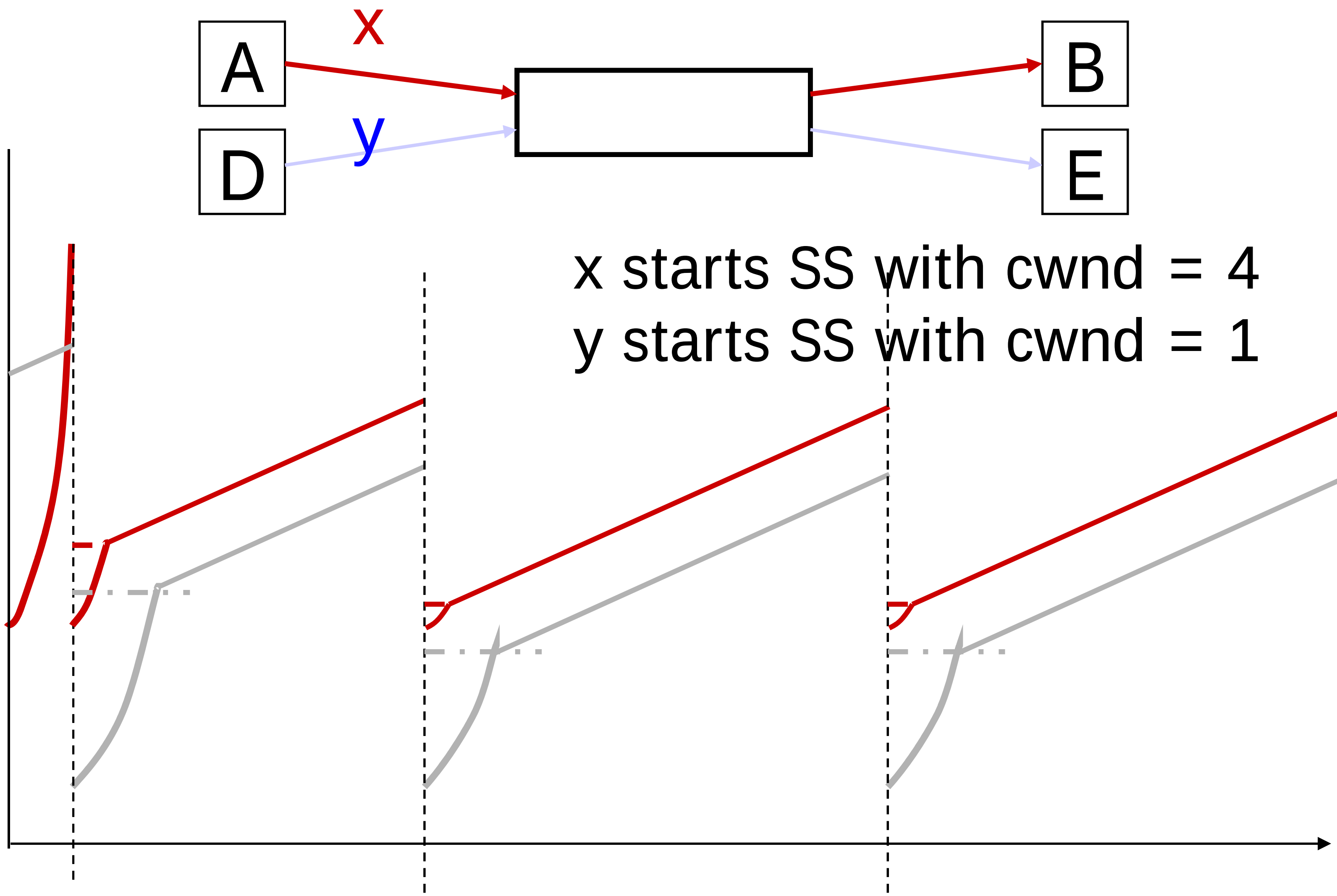
Limit rates:
x = 2y

Increasing cwnd Faster





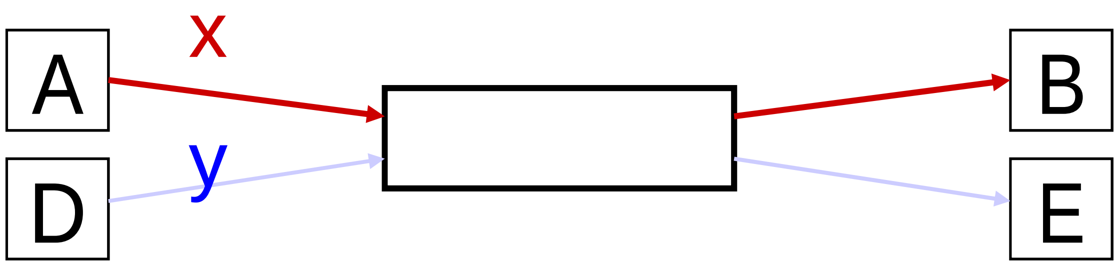
Larger Initial cwnd



x starts SS with cwnd = 4

y starts SS with cwnd = 1

Open Many Connections

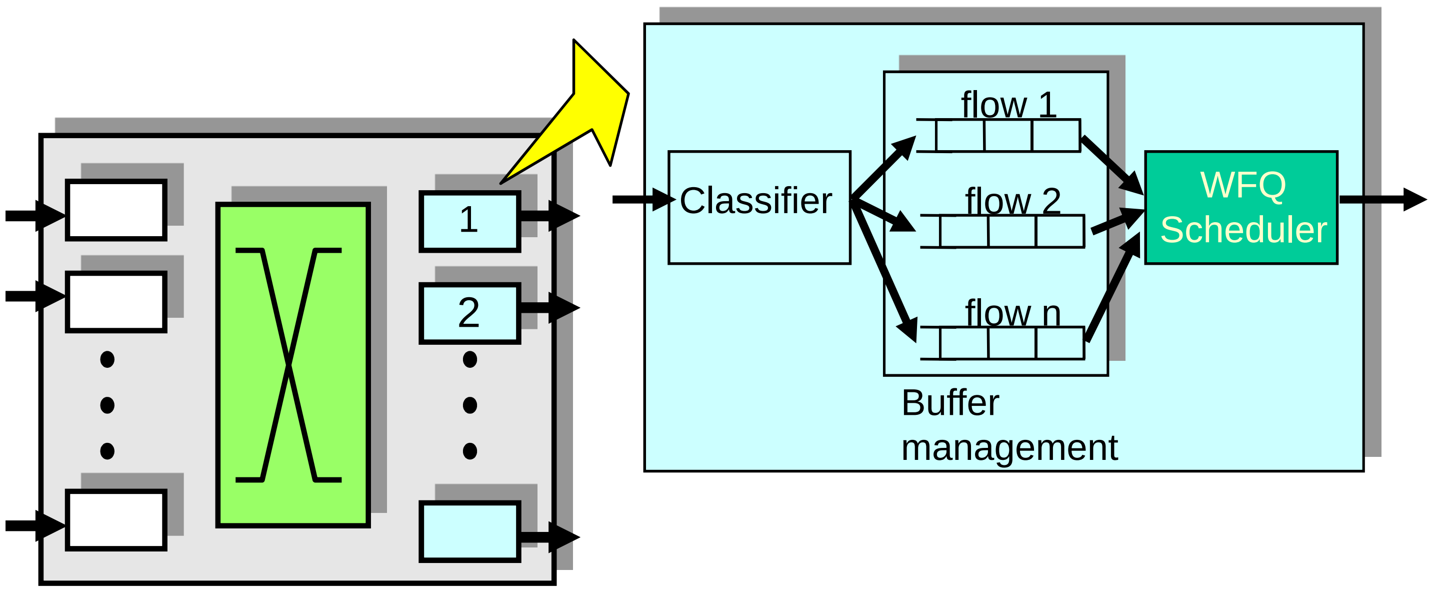


Assume

- A starts 10 connections to B
- D starts 1 connection to E
- Each connection gets about the same throughput

Then A gets 10 times more throughput than D

Generally, Need Stronger Router Mechanisms to Enforce Fairness (e.g. WFQ)



Definition of fairness is murky with parallel connections