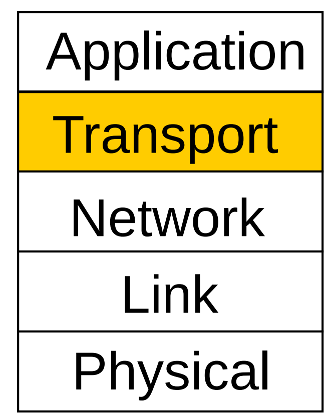
CSEP561 – Congestion Control

David Wetherall
djw@cs.washington.edu

Congestion Control

- Focus:
  - How to share bandwidth between senders

- Congestion & Fairness
- Bandwidth allocation
- TCP congestion control
- RED/ECN



Bandwidth Allocation

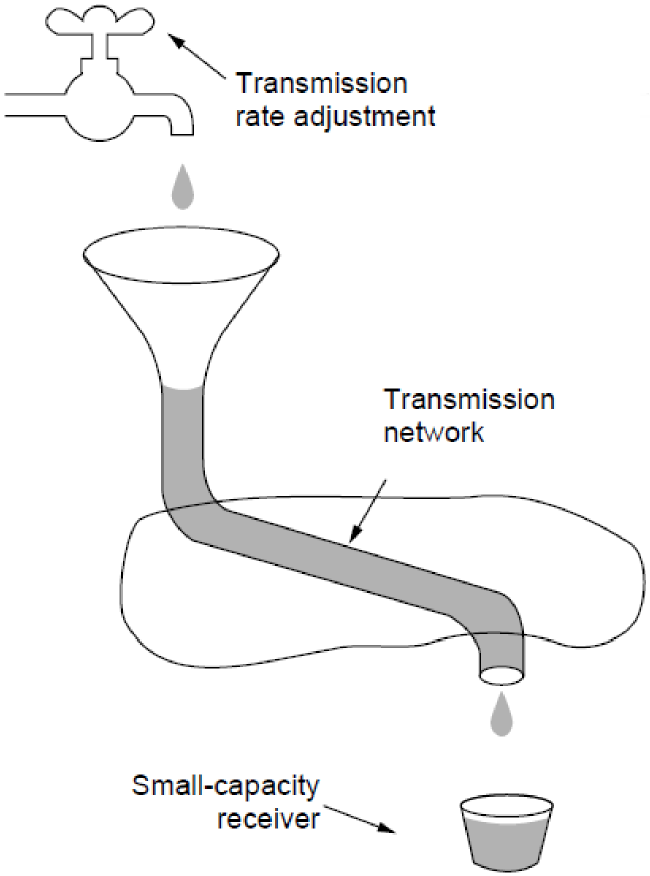
- How fast should the Web server send packets?
- Two big issues to solve!

- Congestion
  - sending too fast will cause packets to be lost in the network
- Fairness
  - different users should get their fair share of the bandwidth

- Often treated together (e.g. TCP) but needn't be

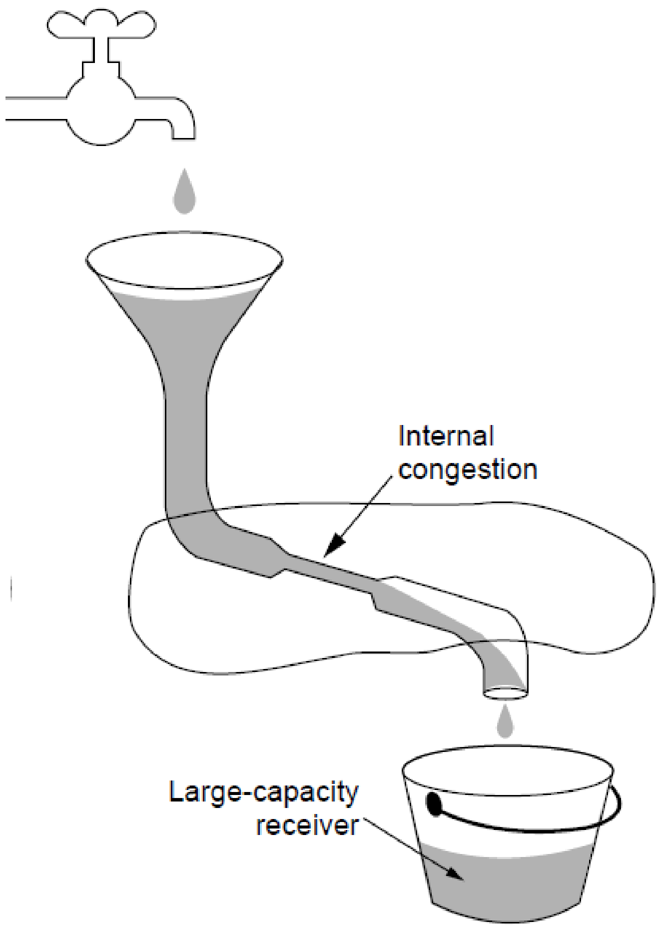
Flow Control

- Limit is the receiver
- No network congestion

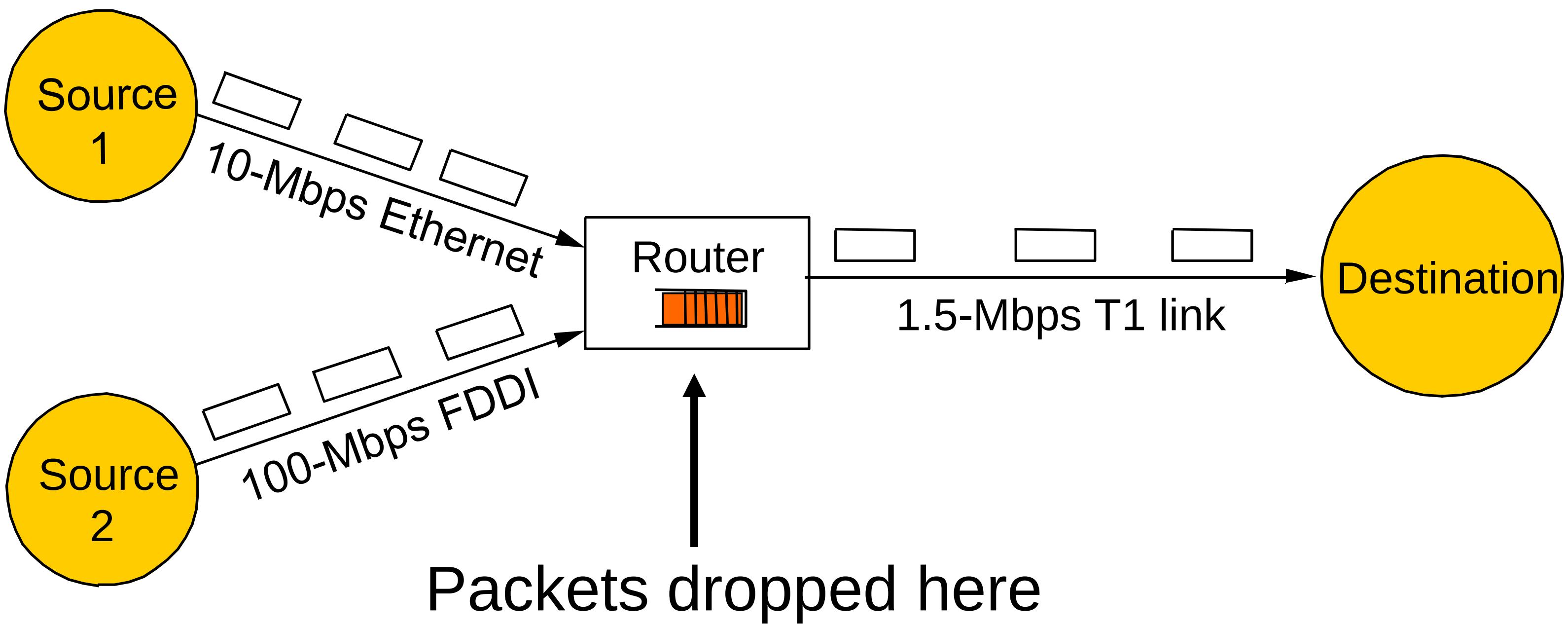


Network Congestion

- Now network is the limit …
- Sender needs to slow down in either of these cases



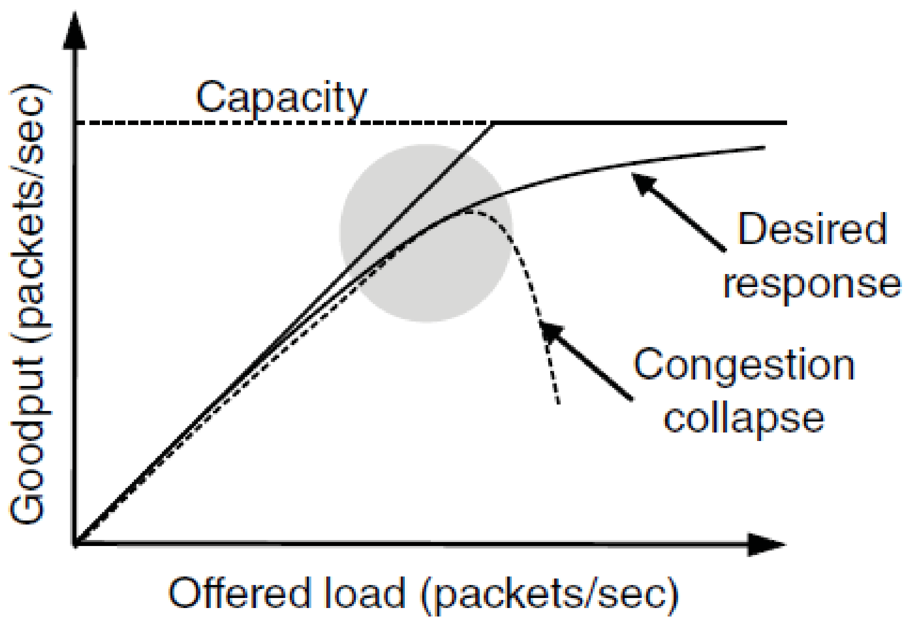
Congestion

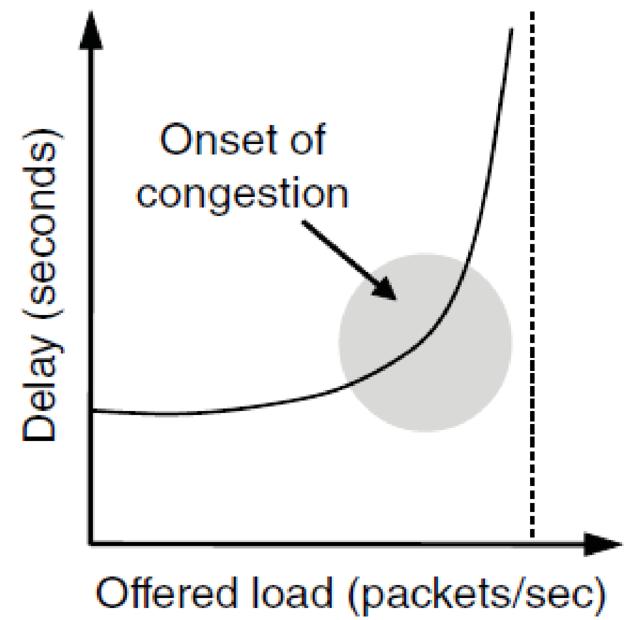


- Buffer intended to absorb bursts when input rate > output
- But if sending rate is persistently > drain rate, queue builds
- Dropped packets represent wasted work; goodput < throughput

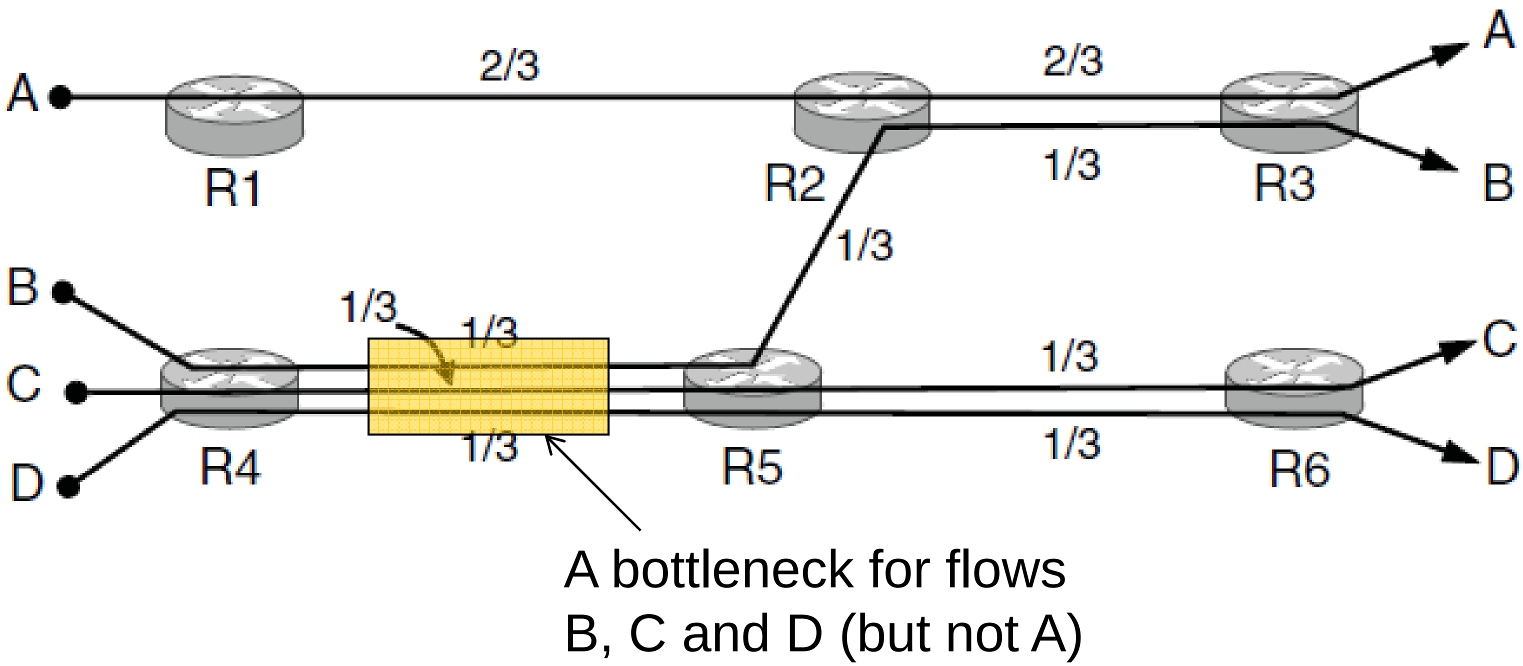
Effects of Congestion

- Want to operate with high throughput and low delay
  - Congestion can lead to collapse if protocols have problems



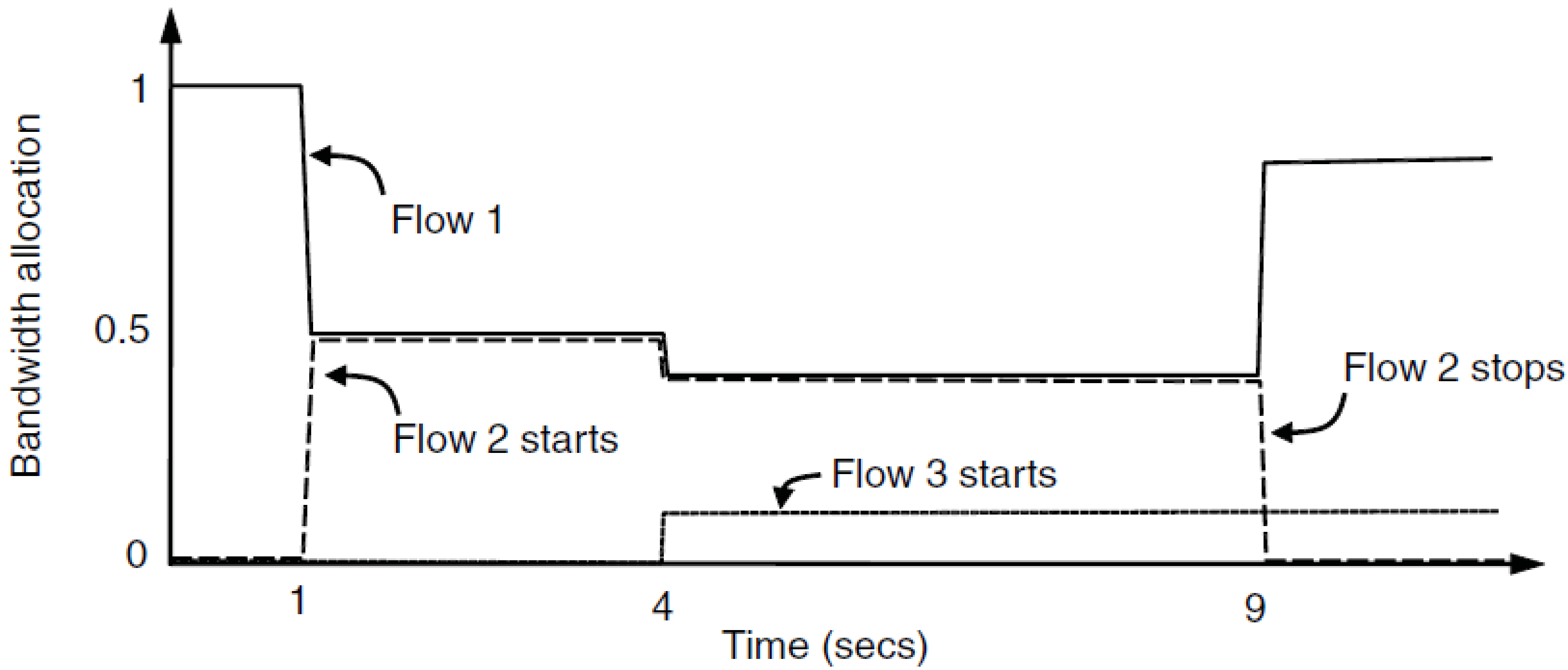


Max-Min Fairness



- Each flow from source to destination gets an equal share of their bottleneck link … depends on paths and other traffic
  - And flows take unclaimed excess bandwidth

Fair allocation changes over time



Bandwidth Allocation Control Loop

- Traffic is bursty
- Congestion is experienced at routers (Network layer)
- Traffic is controlled at sources (Transport/Network layer)

- The two need to talk to each other!
  - Sources sending more slowly is the only relief
  - Sources sending more quickly is the only way to use the capacity

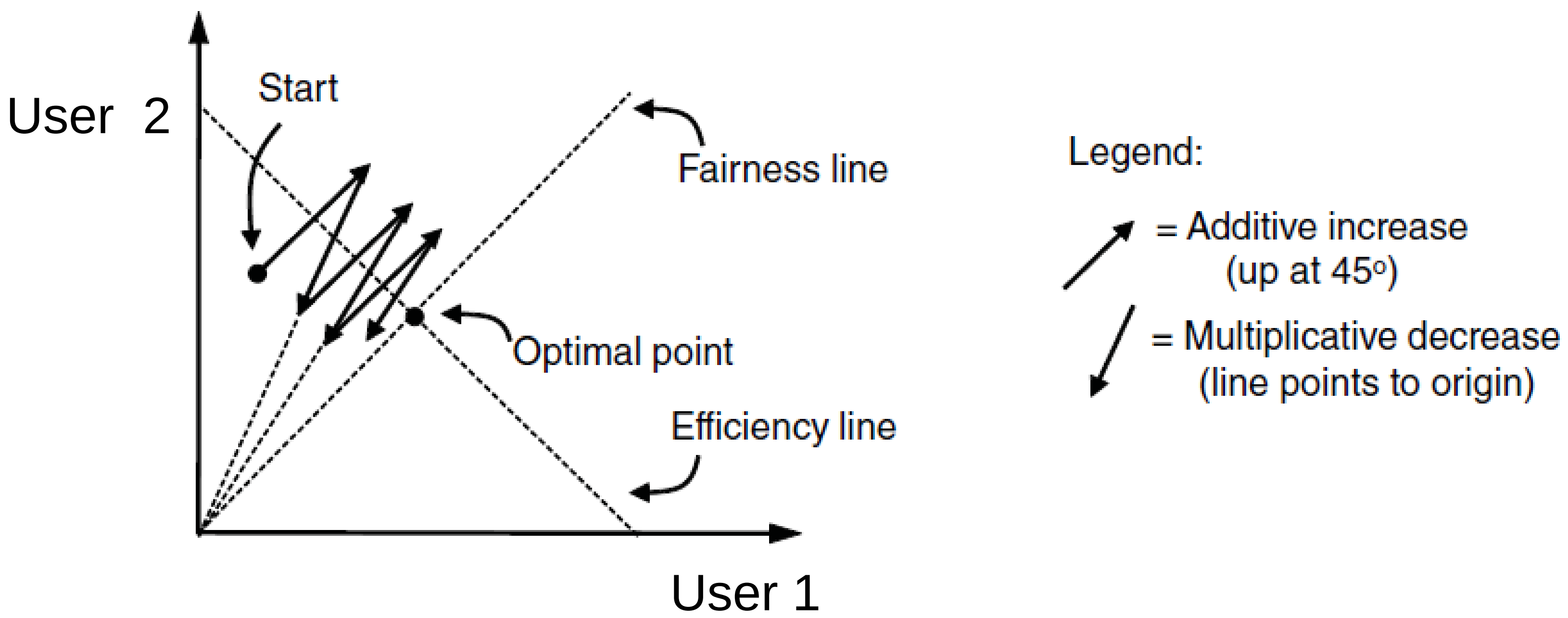
Control Loop Designs

- Open versus Closed loop
  - Open: reserve allowed traffic with network; avoid congestion
  - Closed: use network feedback to adjust sending rate
- Host-based versus Network support
  - Who is responsible for adjusting/enforcing allocations?
- Window versus Rate based
  - How is allocation expressed? Window and rate are related.

- Internet depends on TCP for bandwidth allocation
  - TCP is a host-driven, window-based, closed-loop mechanism

AIMD Control Law (Chiu & Jain, 1989)

- AIMD with binary signals finds the optimal point



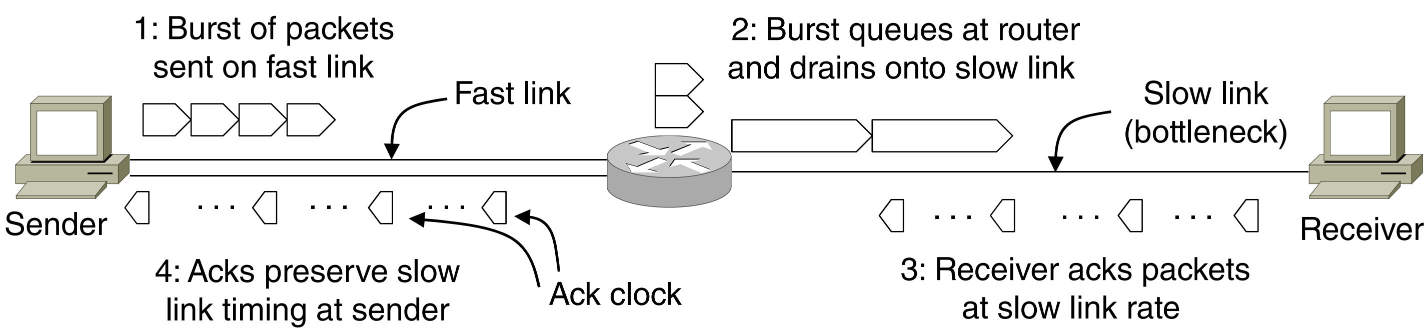
Control Loop Feedback Signals

- Many possible signals:
  - Hosts can observe E2E packet loss (e.g., TCP)
  - Hosts can observe E2E packet delay (e.g., Vegas, FAST)
  - Router can tell source of congestion (e.g., RED/ECN)
  - Router can tell source its allocation (e.g, XCP)
- Each has pros / cons and design implications

TCP Before Congestion Control

- Just use a fixed size sliding window!
  - Will under-utilize the network or cause unnecessary loss
- Congestion control dynamically varies the size of the window to match sending and available bandwidth
  - Sliding window uses minimum of cwnd, the congestion window, and the advertised flow control window
  - Assumes packet loss signals congestion
- The big question: how do we vary the window size?
  - TCP uses various heuristics to adjust cwnd

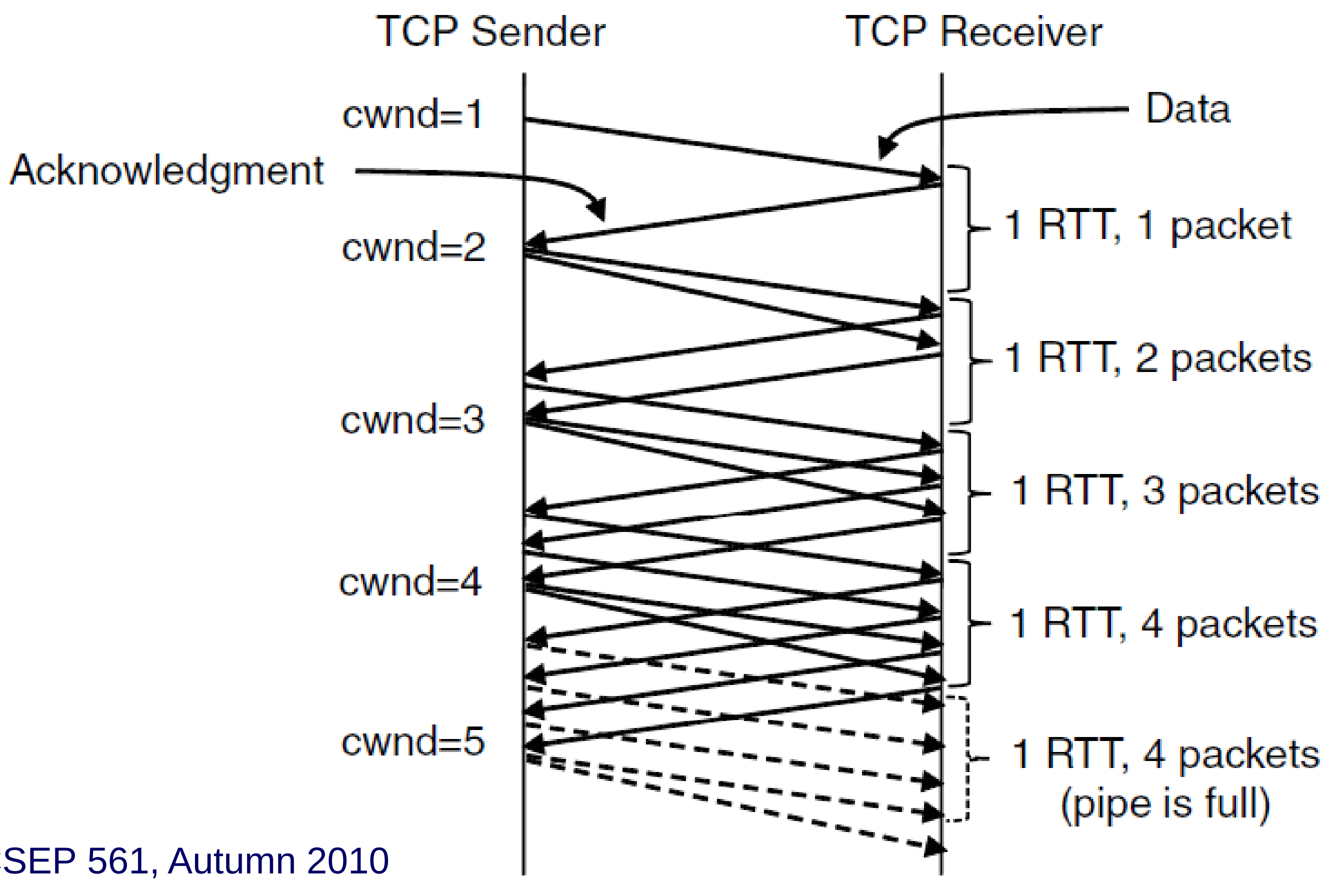
TCP is "Self-Clocking"



- Neat observation: acks pace transmissions at approximately the botteneck rate
- So "ack clock" with sliding window spreads packets out
- And just by sending packets we can discern the "right" sending rate (called the packet-pair technique)

AIMD

- (This is the additive increase part for one sender)



TCP AIMD cwnd rules

- Increase slowly while we believe there is bandwidth
  - Cwnd += 1 packet / RTT
  - Commonly approx. is cwnd += 1/cwnd per packet
  - Additive increase per RTT

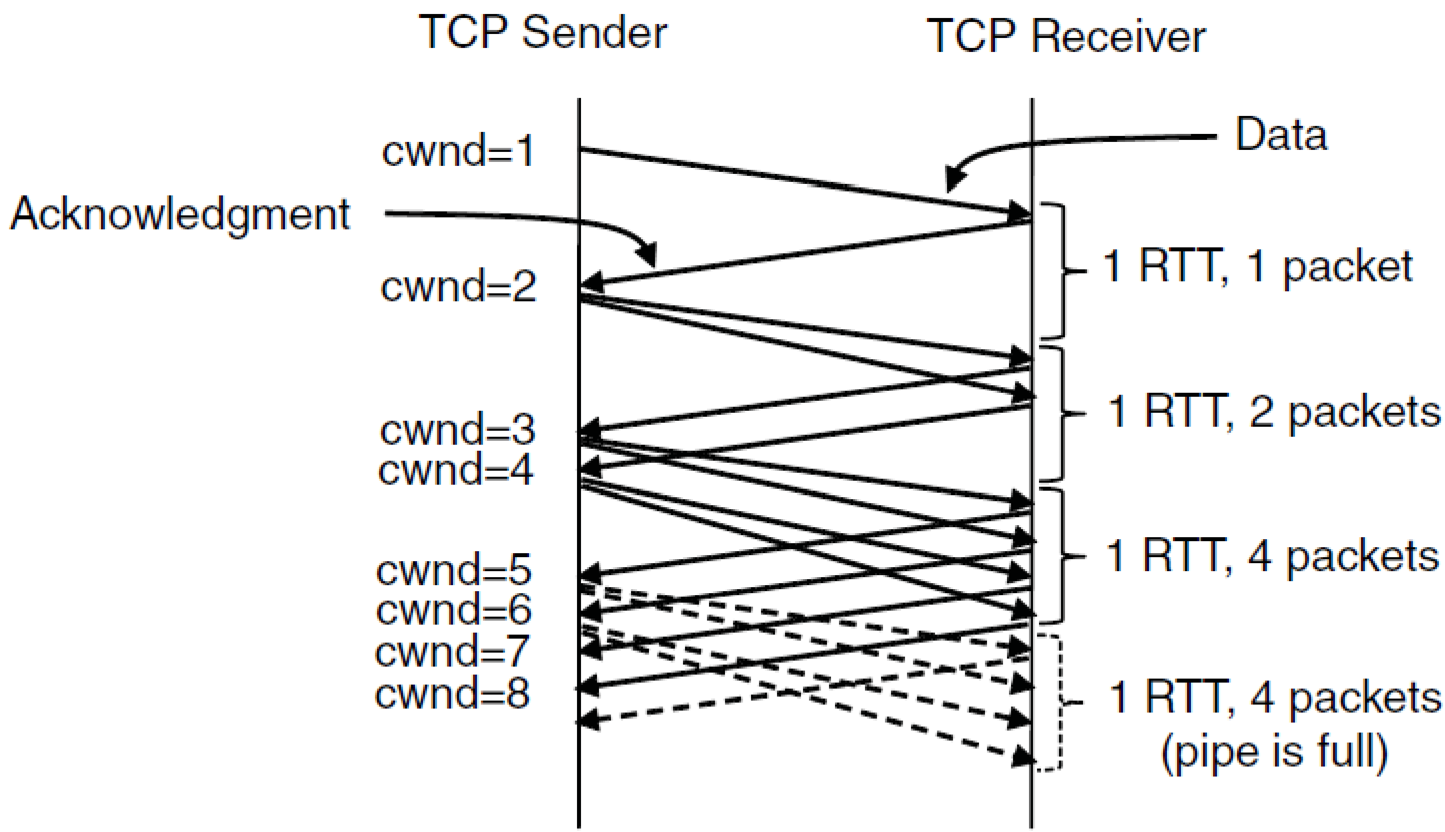
- Decrease quickly when there is loss (went too far!)
  - Cwnd /= 2
  - Multiplicative decrease

TCP “Slow Start”

- But it can take AIMD a long time to get to a good cwnd

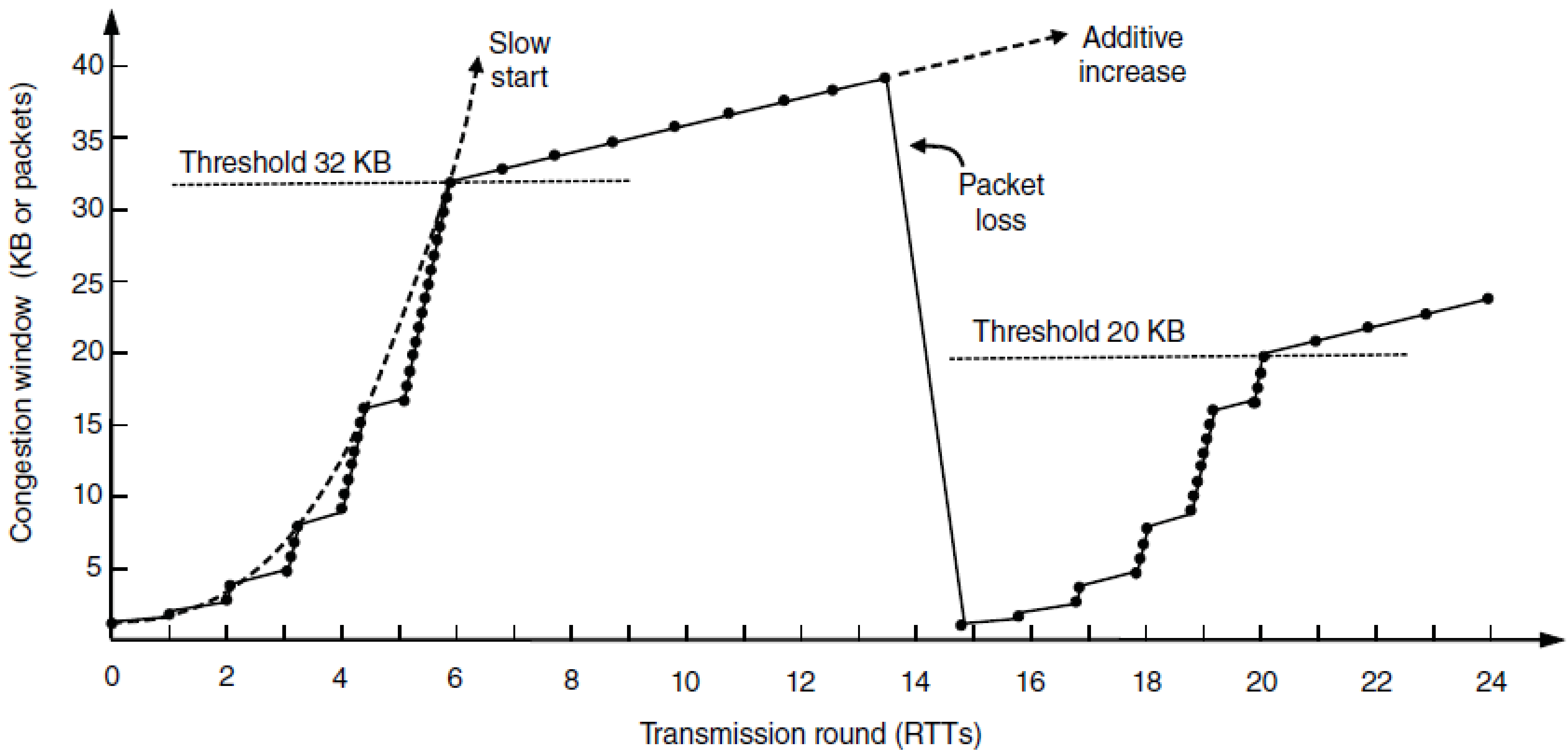
- Use a different strategy to get close
  - Double cwnd every RTT
  - Cwnd *= 2 / RTT
  - Commonly done as cwnd +=1 / packet received

TCP slow-start cwnd rules



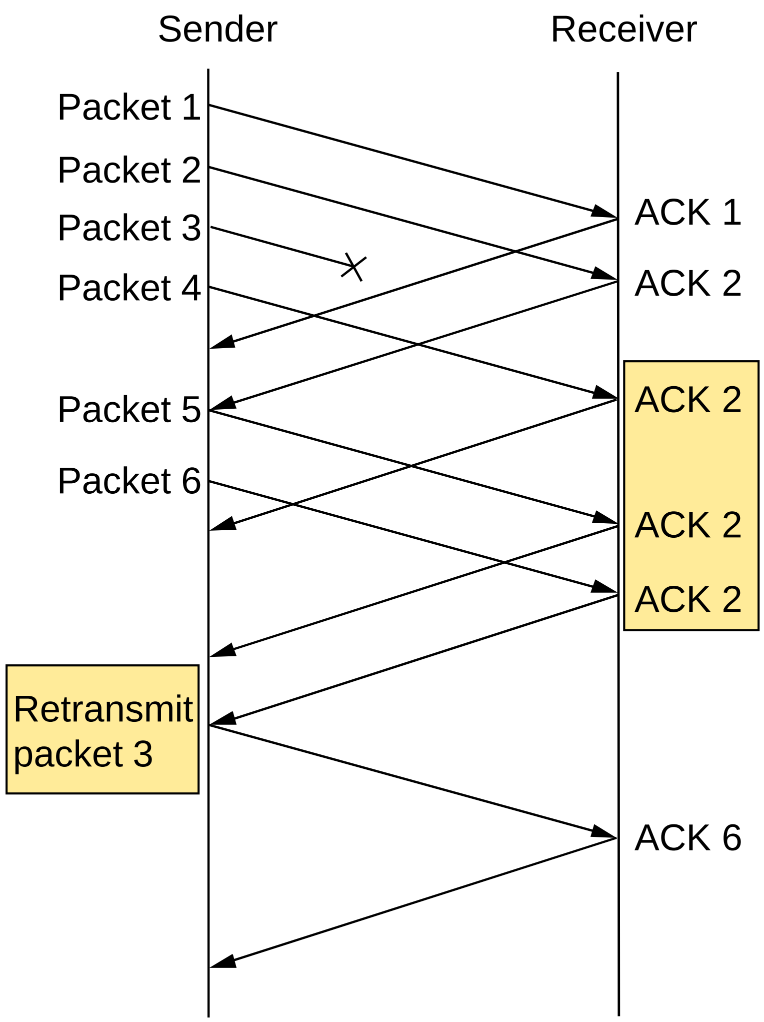
Combining Slow-Start and AI(MD)

- Switch to AI at a threshold; but why restart after loss?



Fast Retransmit

- No need to wait until a timeout to infer loss
- TCP uses cumulative acks, so duplicate acks start arriving after a packet is lost
  - 3 duplicate acks is enough
- Lets us halve cwnd and retransmit the lost packet quickly



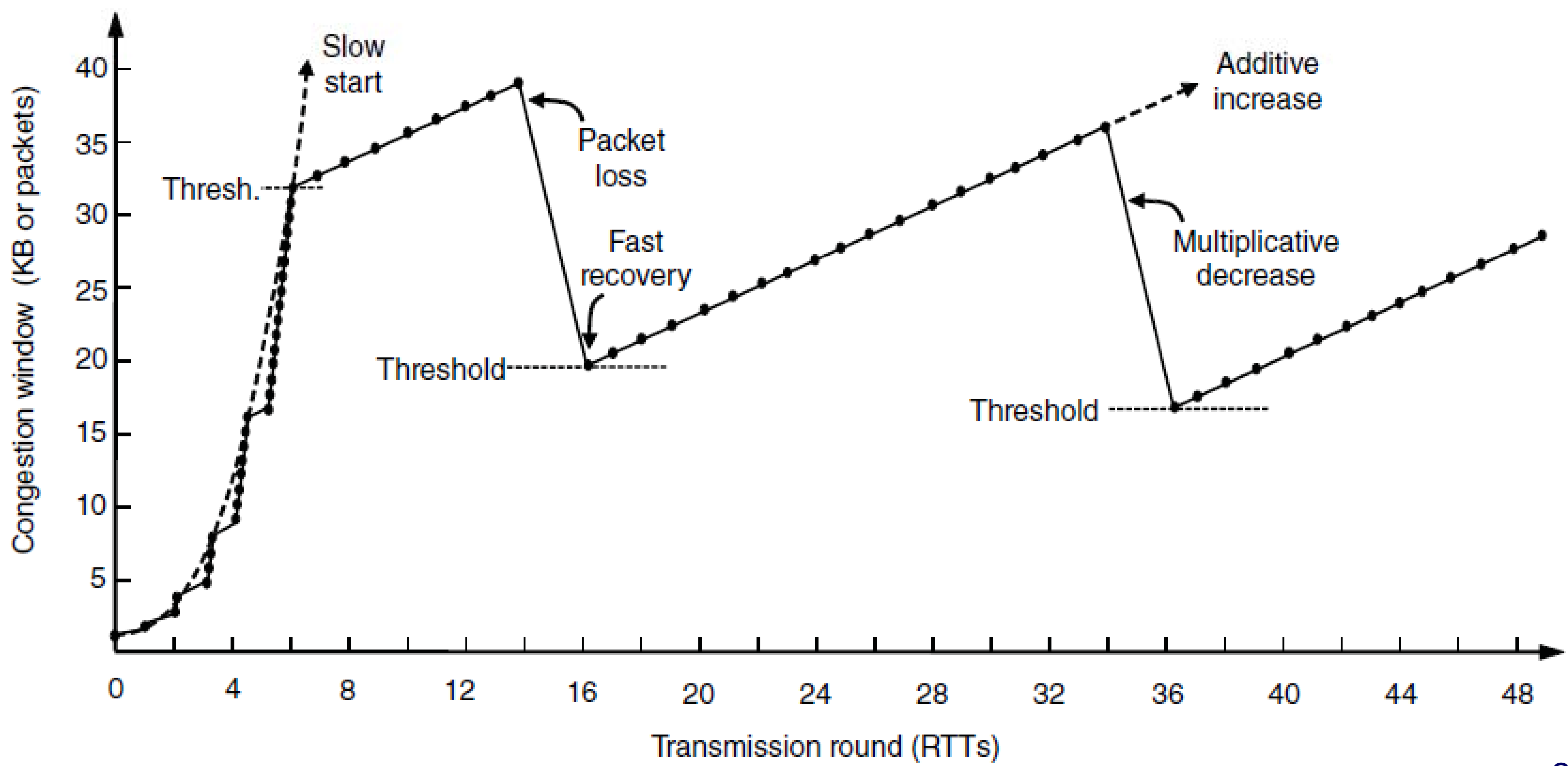
Fast Recovery

- After Fast Retransmit, further duplicate acks represent new packets that have left the network
  - Use them to grow cwnd and clock out new packets

- End result: Can achieve AIMD when there are single packet losses. Only slow start the first time.

TCP with Fast Retransmit/Recovery

- Creates the classic “TCP sawtooth” pattern



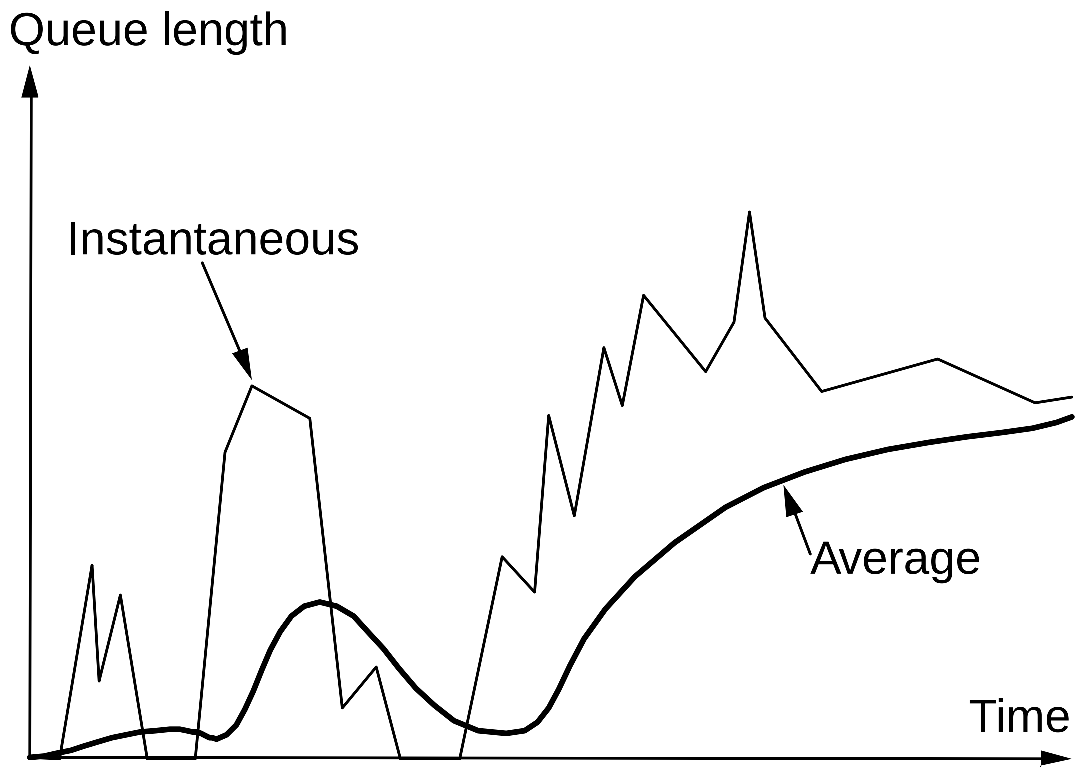
Avoidance versus Control

- Congestion control
  - Recover from congestion that is already degrading performance
- Congestion avoidance
  - Avoid congestion by slowing down at the onset

- Latter benefits from router support

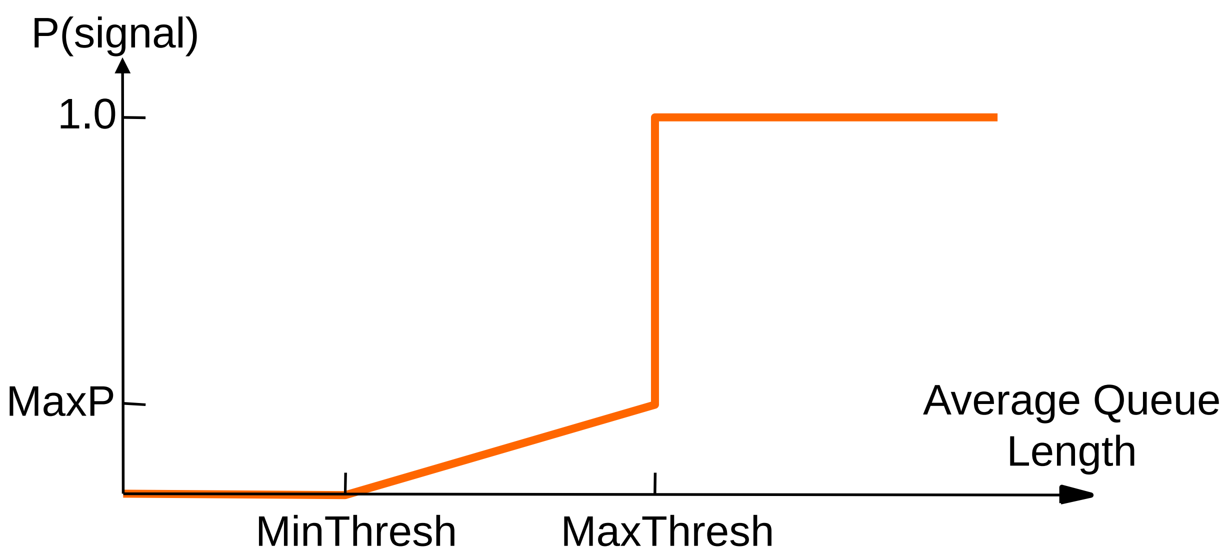
Detecting the onset of congestion

- Sustained overload causes queue to build and overflow
- Router can watch for an increase in the average delay



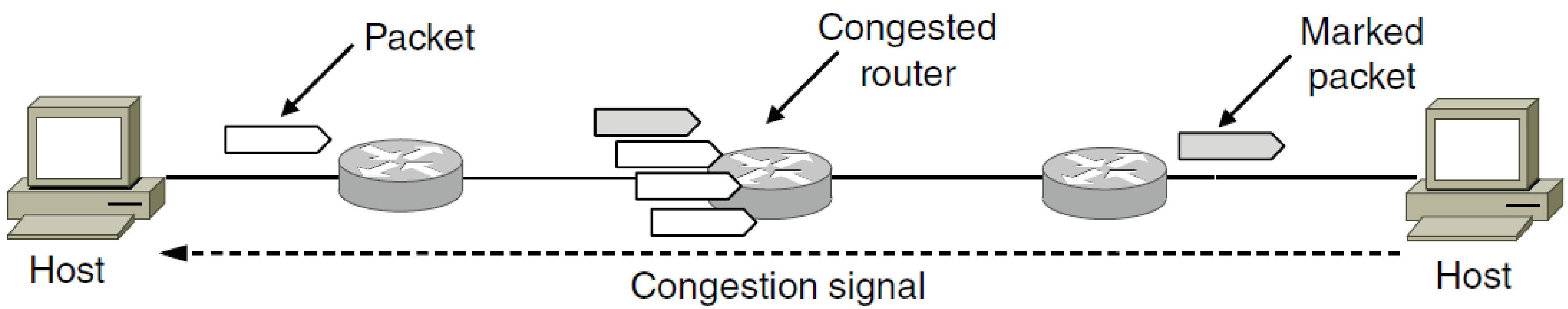
Random Early Detection (RED) routers

- Router sends “early” signal to source when avg. queue builds



- Probabilistically choose packet to signal; fast flows get more

RED signaling



- Preferred (future) method:
  - Set Explicit Congestion Notification bits in the IP packet header
  - Destination returns this signal to the source with reverse traffic
  - Reliable signal without extra packets at a time of congestion

More on RED signaling

- Deprecated (present) method
  - Drop the packet; that is what pre-RED routers do anyway
  - Source will get the hint
  - Paradox is that early loss can improve performance!
  - This is why RED tries to give each source only one signal
- In practice, RED is not widely used
  - Depends on tuning to work well
  - No strong incentive for early adopters