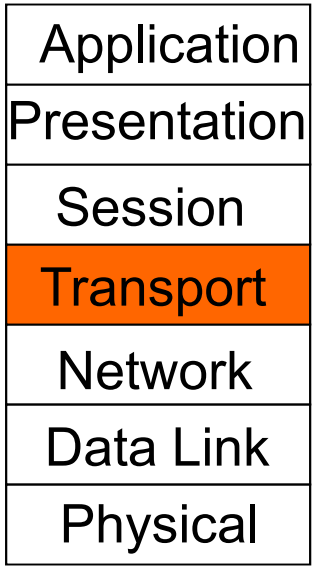
Bandwidth Allocation & TCP

- The Transport Layer

- Focus
  - How do we share bandwidth?

- Topics
  - Congestion control & fairness
  - TCP Additive Increase/Multiplicative Decrease
  - TCP Slow Start
  - TCP Fast Recovery



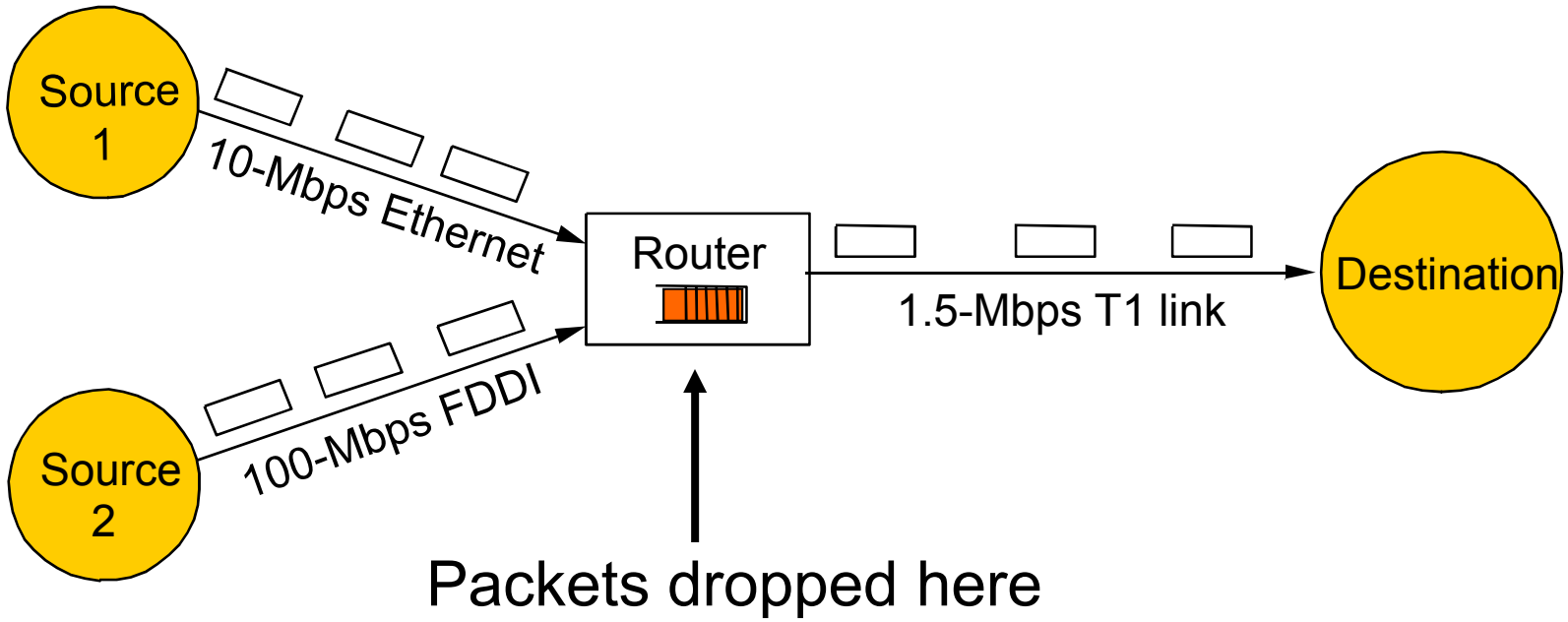
Bandwidth Allocation

- 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

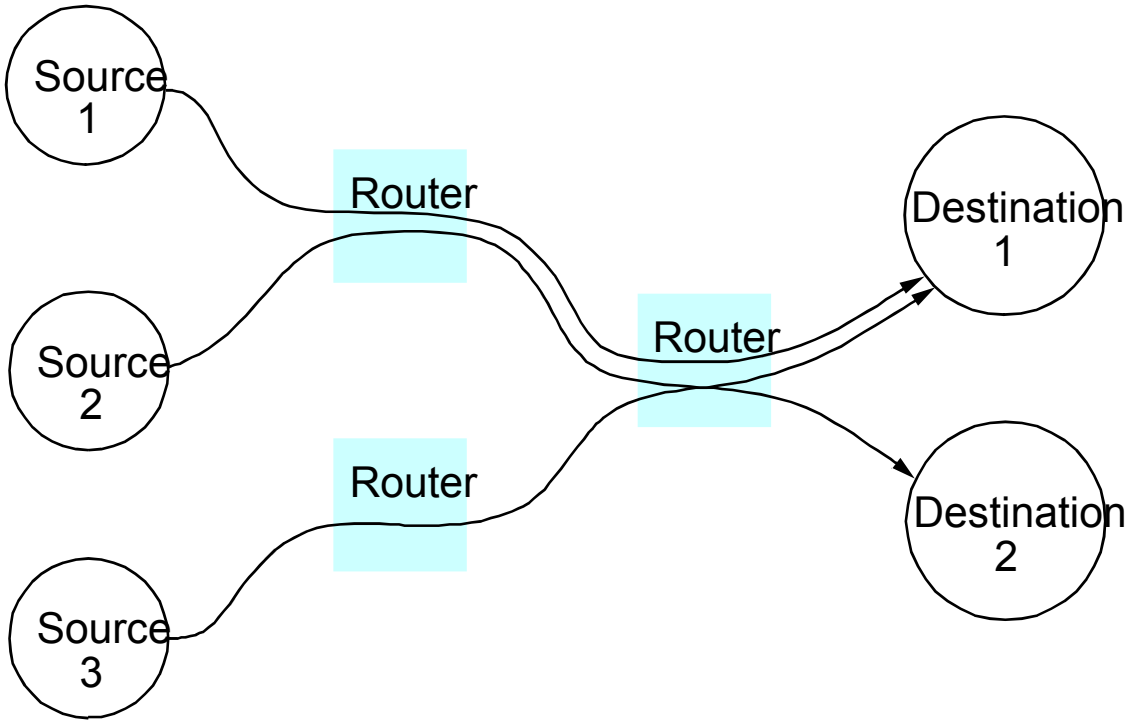
- Often treated together (e.g. TCP) but needn't be

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

Fairness



- Each flow from a source to a destination should get an equal share of the bottleneck link … depends on paths and other traffic

Bandwidth Allocation Approaches

- 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

Design Choices

- TCP/Internet provides “best-effort” service
  - Implicit network feedback, host controls via window.
  - No strong notions of fairness

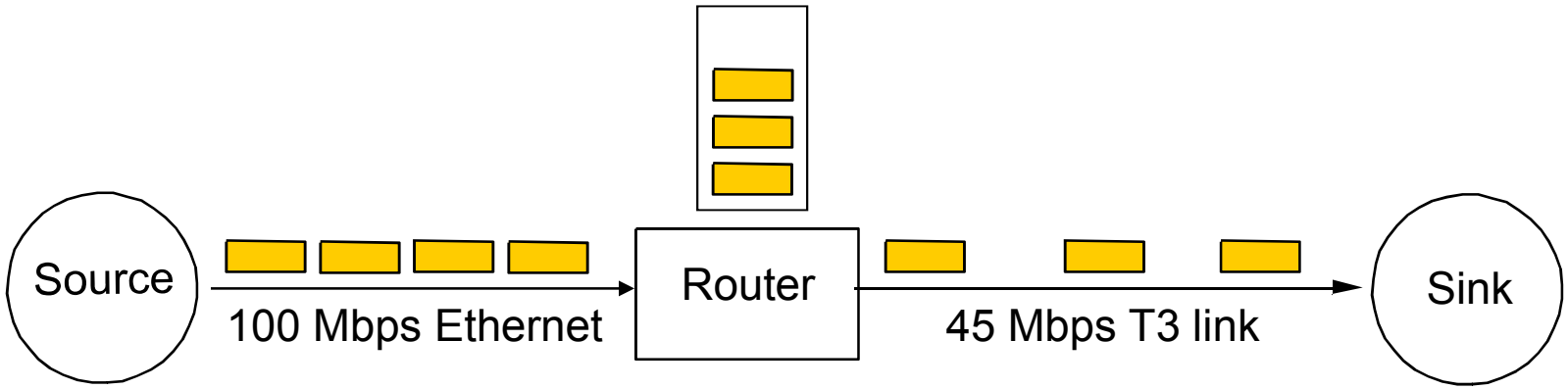
- A network in which there are QOS (quality of service) guarantees
  - Rate-based reservations natural choice for some apps
  - But reservations are need a good characterization of traffic
  - Network involvement typically needed to provide a guarantee

- Former tends to be simpler to build, latter offers greater service to applications but is more complex.

TCP Before Congestion Control

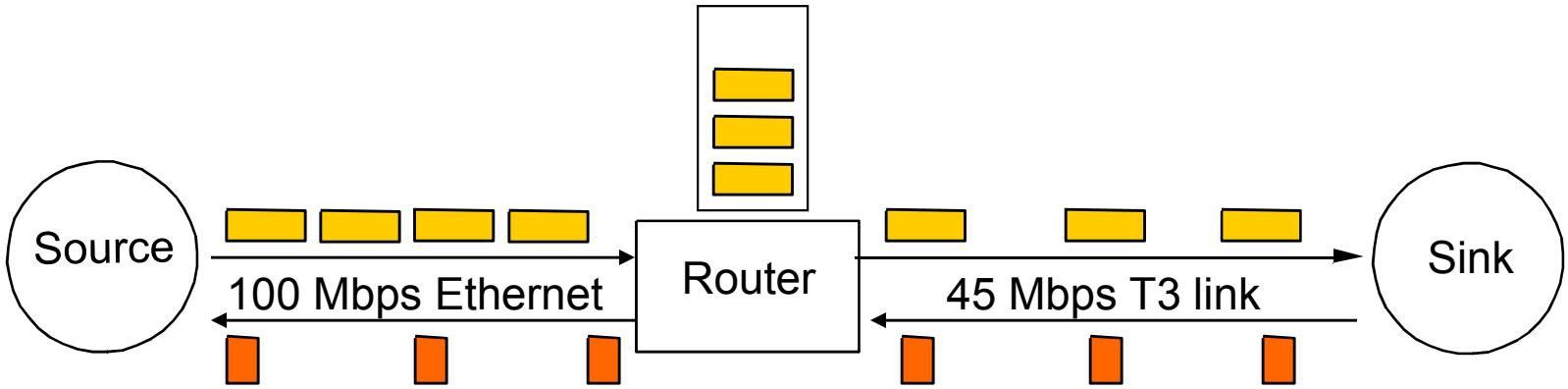
- Just use a fixed size sliding window!
  - Will under-utilize the network or cause unnecessary loss
- Congestion control dynamically varies the size of the window to match sending and available bandwidth
  - Sliding window uses minimum of cwnd, the congestion window, and the advertised flow control window
- The big question: how do we decide what size the window should be?

TCP Probes the Network



- Each source independently probes the network to determine how much bandwidth is available
  - Changes over time, since everyone does this
- Assume that packet loss implies congestion
  - Since errors are rare; also, requires no support from routers

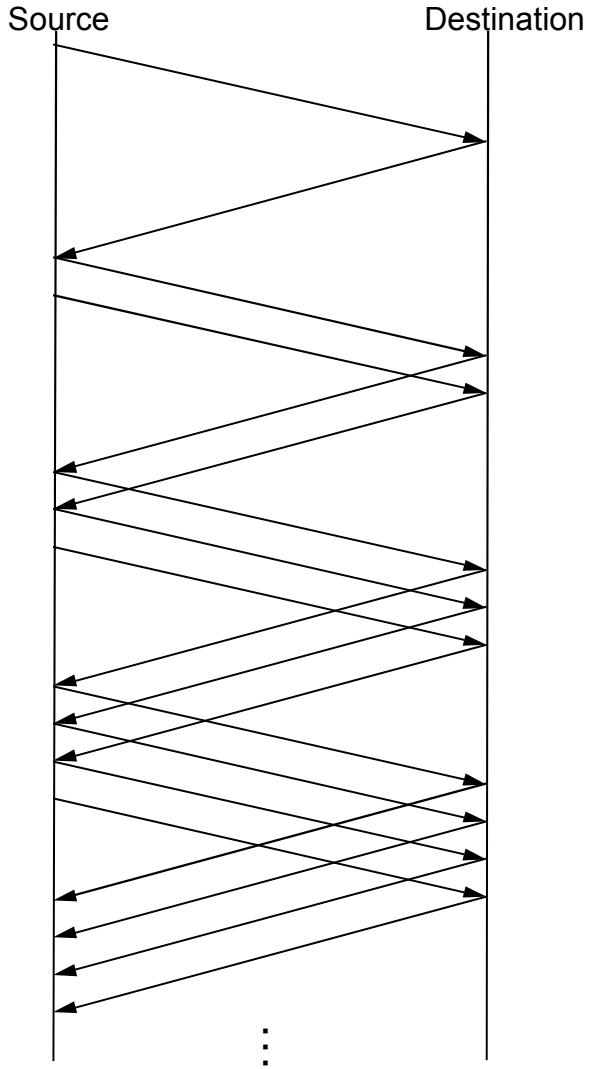
TCP is “Self-Clocking”



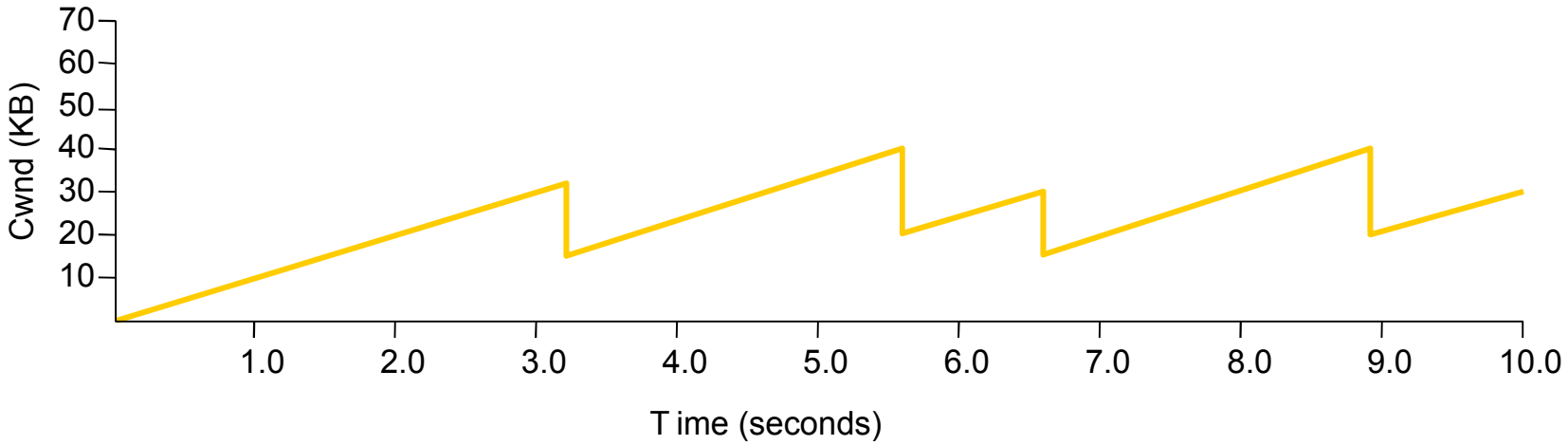
- Neat observation: acks pace transmissions at approximately the botteneck rate
- So just be sending packets we can discern the “right” sending rate (called the packet-pair technique)

AIMD (Additive Increase/Multiplicative Decrease)

- How to adjust probe rate?
- Increase slowly while we believe there is bandwidth
  - Additive increase per RTT
  - Cwnd += 1 packet / RTT
- Decrease quickly when there is loss (went too far!)
  - Multiplicative decrease
  - Cwnd /= 2



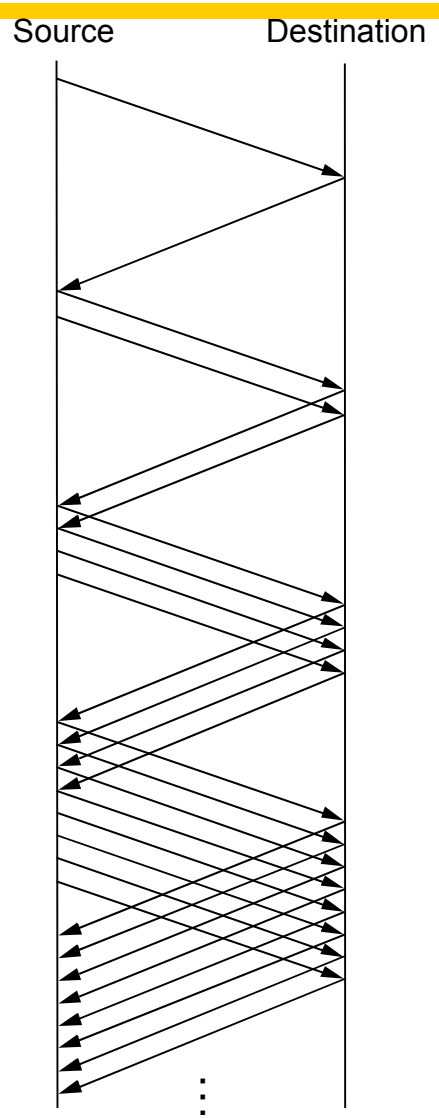
TCP Sawtooth Pattern



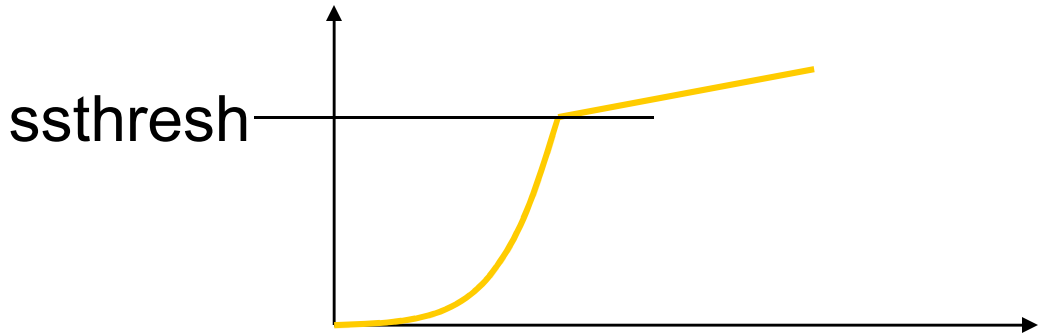
“Slow Start”

- Q: What is the ideal value of cwnd? How long will AIMD take to get there?

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

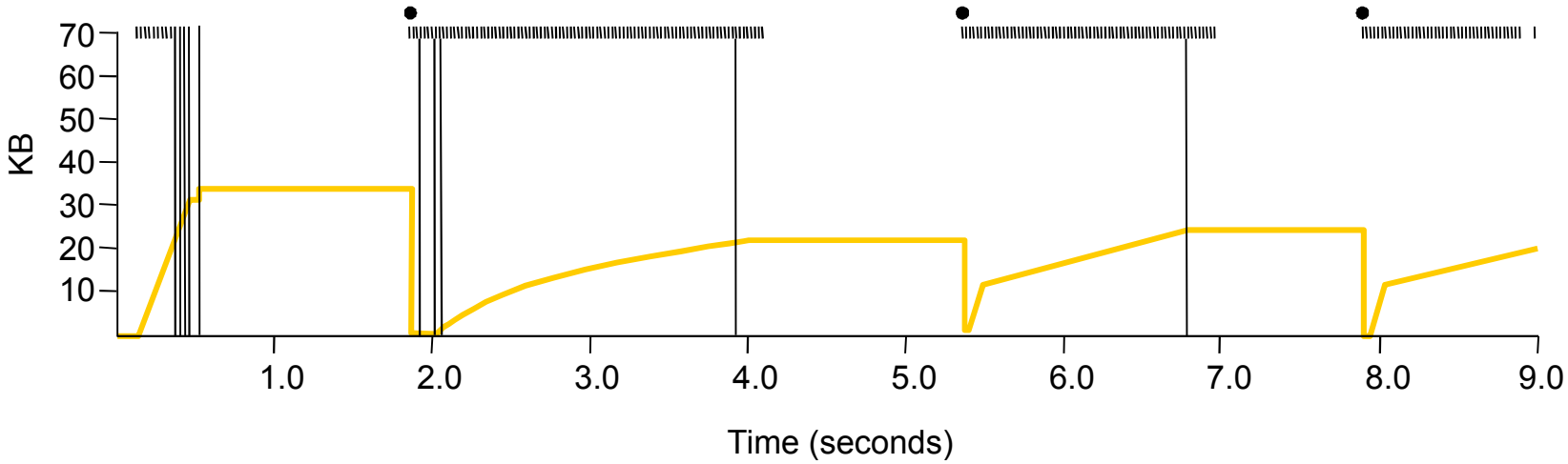


Combining Slow Start and AIMD



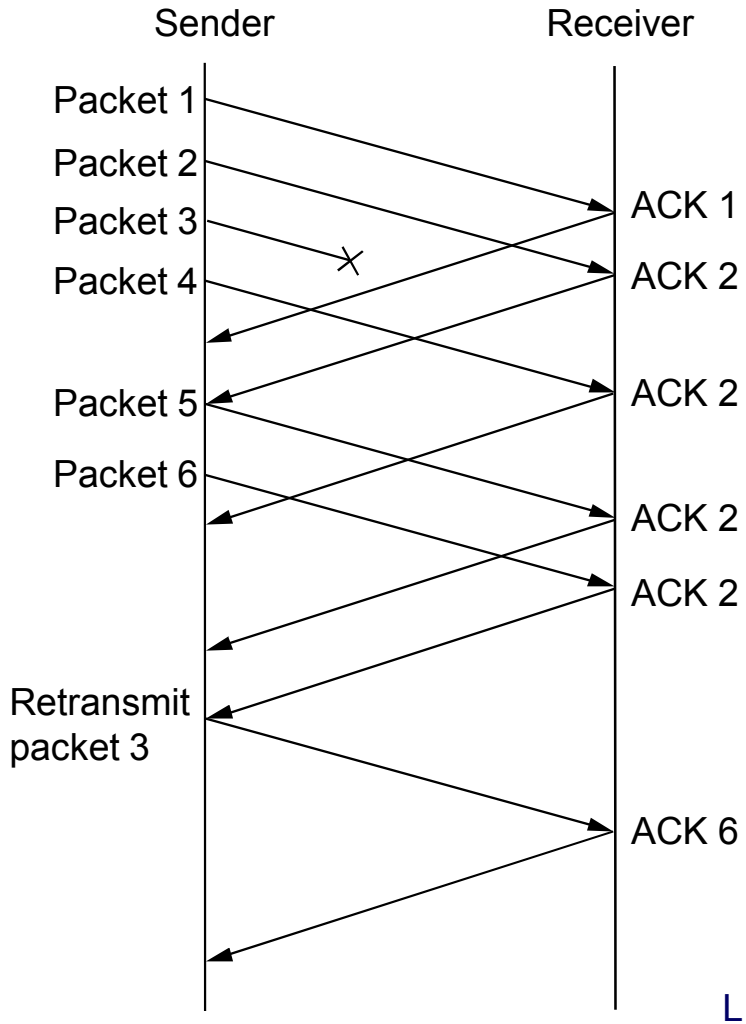
- Slow start is used whenever the connection is not running with packets: initially, and after timeouts
- But we don't want to overshoot our ideal cwnd, so remember the last cwnd that worked with no loss
  - Ssthresh = cwnd after cwnd /= 2 on loss
  - Switch to AIMD once cwnd passes ssthresh

Example (Slow Start +AIMD)

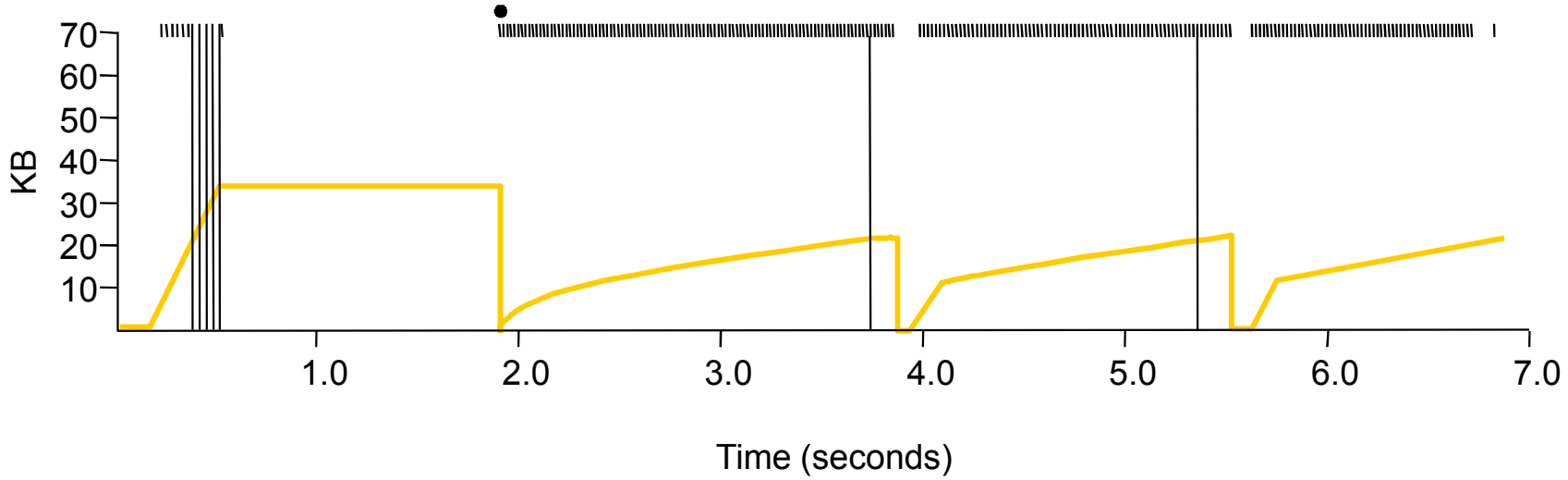


Fast Retransmit

- TCP uses cumulative acks, so duplicate acks start arriving after a packet is lost.
- We can use this fact to infer which packet was lost, instead of waiting for a timeout.
- 3 duplicate acks are used in practice



Example (with Fast Retransmit)



Fast Recovery

- After Fast Retransmit, use further duplicate acks to grow cwnd and clock out new packets, since these acks represent packets that have left the network.

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

Example (with Fast Recovery)