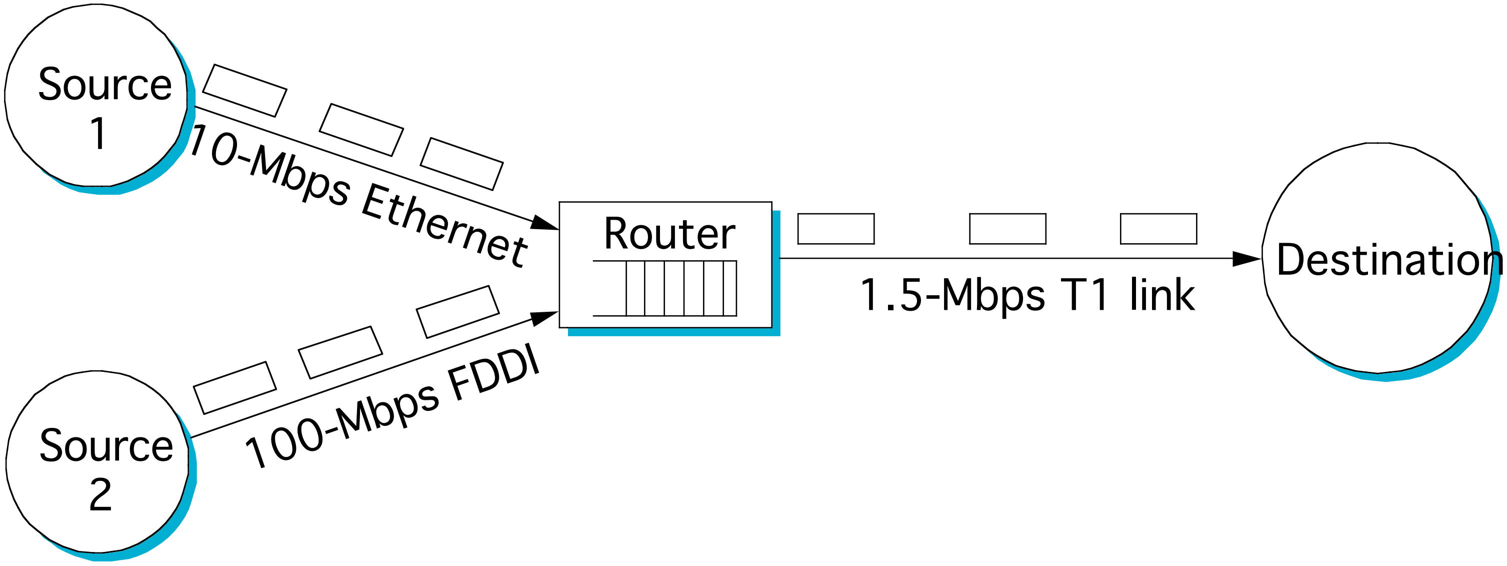
So far,

- On the networking side, we looked at mechanisms to links hosts using direct linked networks and then forming a network of these networks. We introduced Internet protocols as a way to name and access the nodes
- Now we are focusing on TCP as a mechanism to implement reliable traffic that can operate on a heterogeneous network and be friendly to other traffic
  - We’ve seen flow control, initial connection negotiation (ISN, SYN/FIN,)
  - Next we look at congestion control

Congestion



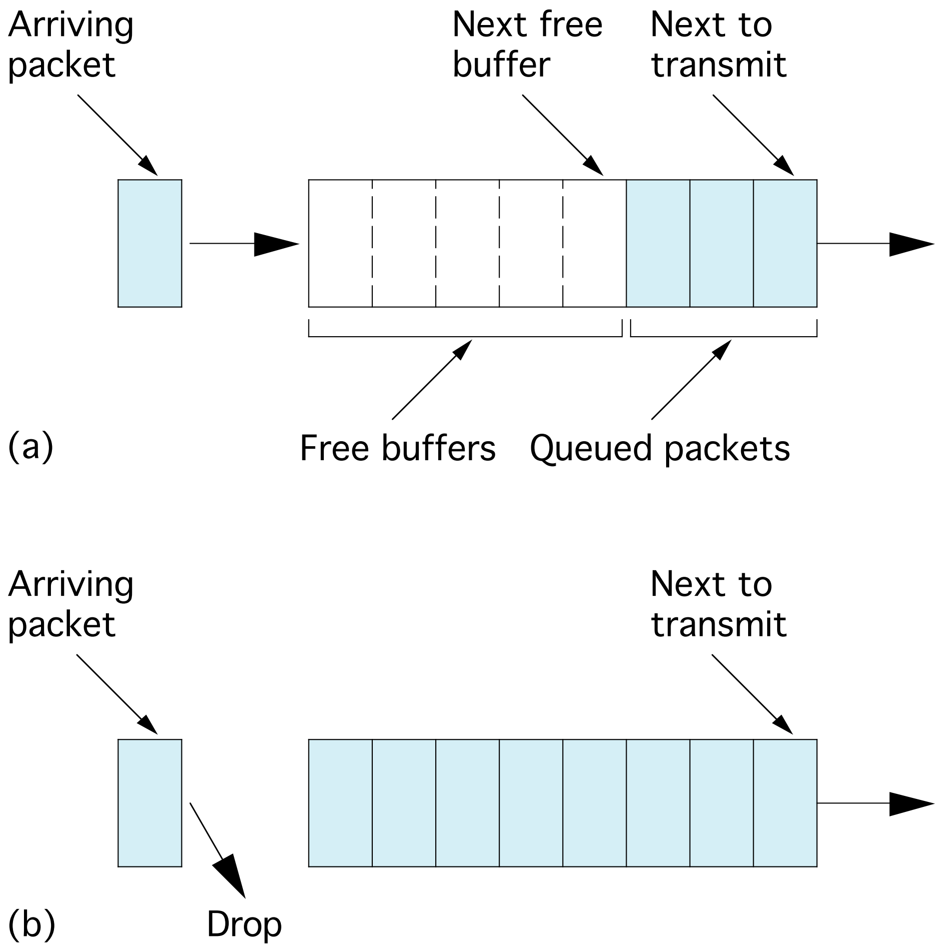
- If both sources send full windows, we may get congestion collapse
- Other forms of congestion collapse:
  - Retransmissions of large packets after loss of a single fragment
  - Non-feedback controlled sources

Resource allocation mechanisms

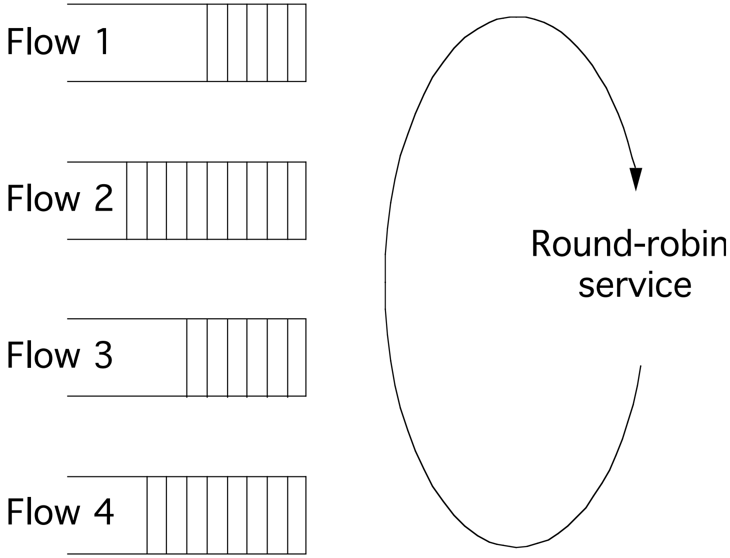
- Router Centric vs Host-centric
  - Whether routers are required to deal with congestion by themselves or whether end hosts monitor the network and respond to congestion.
  - Both require some help from the other component
- Reservation based or feedback based
  - Reservation: end host asks for certain resources
  - Feedback based: End host reacts to feedback from system, explcit or implicit
- Window based or rate based
  - TCP like: buffer size specifies amount of traffic to expect (can be bursty)
  - Constrain rate at which data is sent

Queuing disciplines

- FIFO + tail drop:



- Fair queuing: Fairness per flow



6.3 TCP Congestion Control

- Idea
  - assumes best-effort network (FIFO or FQ routers) each source determines network capacity for itself
  - uses implicit feedback
  - ACKs pace transmission (*self-clocking*)
- Challenge
  - determining the available capacity in the first place
  - adjusting to changes in the available capacity

TCP Congestion Control

- A collection of interrelated mechanisms:
  - Slow start
  - Congestion avoidance
  - Accurate retransmission timeout estimation
  - Fast retransmit
  - Fast recovery

Congestion Control

- Underlying design principle: packet conservation
    - At equilibrium, inject packet into network only when one is removed
    - Basis for stability of physical systems
- 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

TCP Congestion Control Basics

- Keep a congestion window, cwnd
    - Denotes how much network is able to absorb
- Sender's maximum window:
    - Min (advertised window, cwnd)
- Sender's actual window:
    - Max window - unacknowledged segments

Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available capacity
- New state variable per connection: **`CongestionWindow`**
  - limits how much data source has in transit

```
          MaxWin = MIN(CongestionWindow,
AdvertisedWindow)
          EffWin = MaxWin - (LastByteSent -
LastByteAcked)
```

- Idea:
  - increase **`CongestionWindow`** when congestion goes down
  - decrease **`CongestionWindow`** when congestion goes up

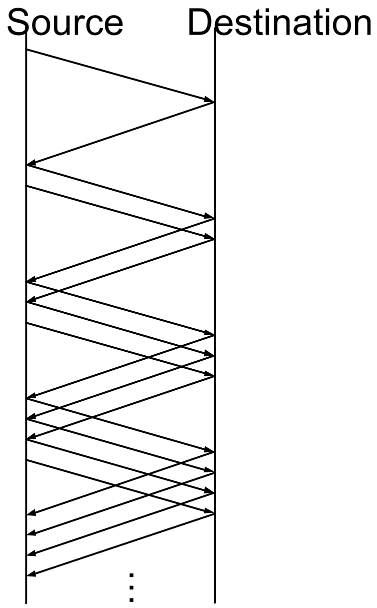
AIMD (cont)

- Question: how does the source determine whether or not the network is congested?

- Answer: a timeout occurs
    - timeout signals that a packet was lost
    - packets are seldom lost due to transmission error
    - lost packet implies congestion

AIMD (cont)

- Algorithm
  - increment **`CongestionWindow`** by one packet per RTT (*linear increase*)
  - divide **`CongestionWindow`** by two whenever a timeout occurs (*multiplicative decrease*)

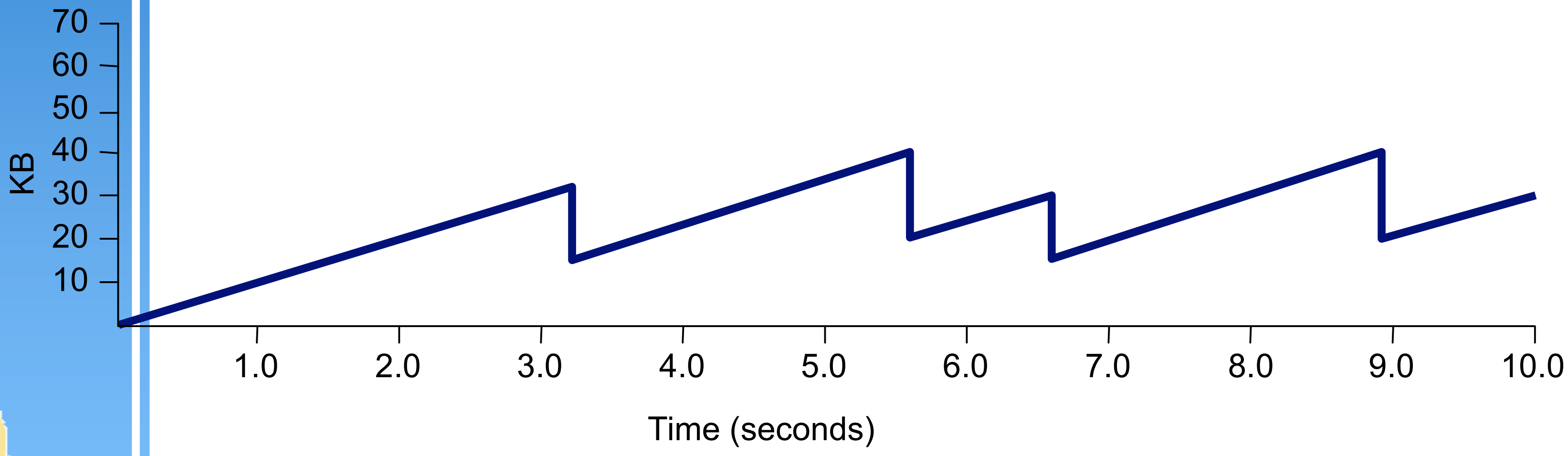


- In practice: increment a little for each ACK

```
Increment = (MSS * MSS)/CongestionWindow
CongestionWindow += Increment
```

AIMD (cont)

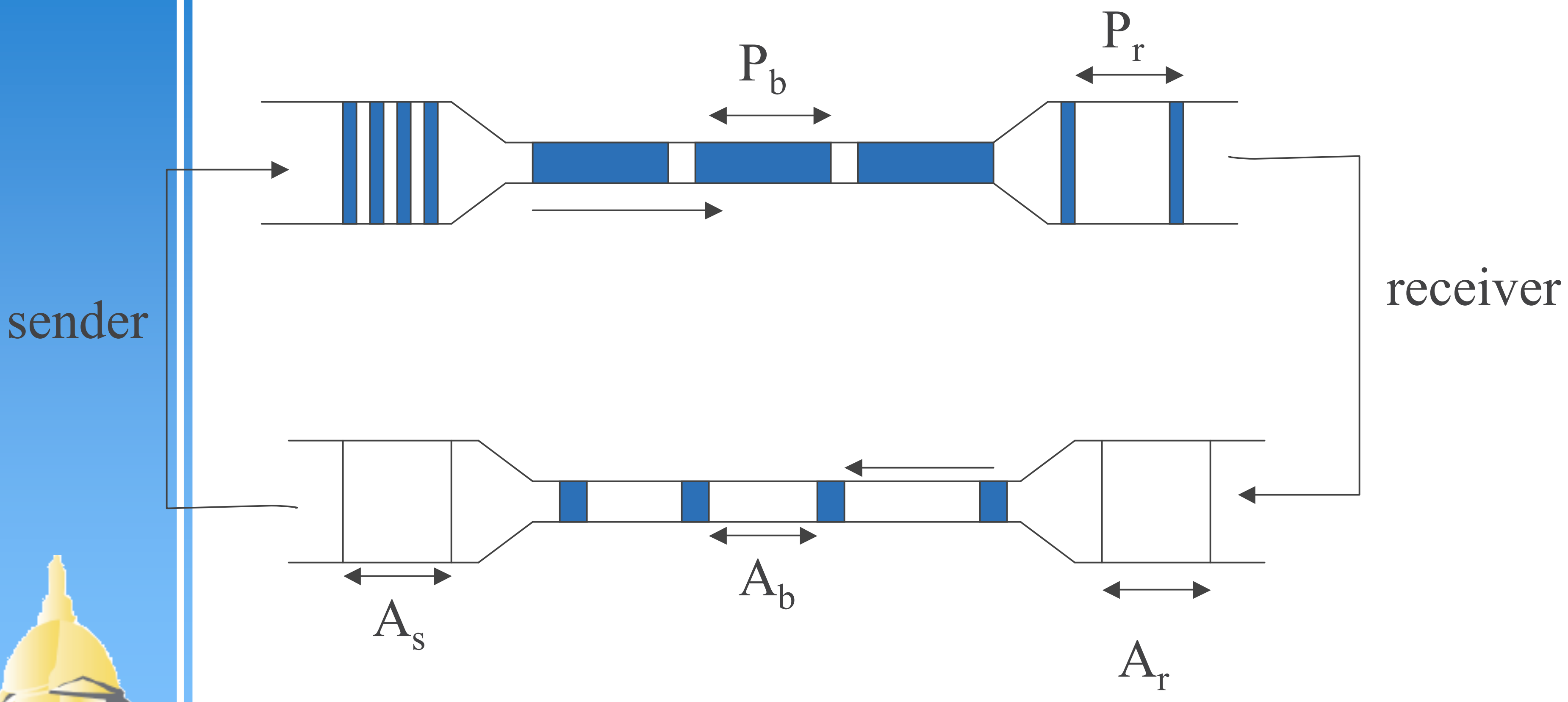
- Trace: sawtooth behavior for cwnd vs time



Self-clocking

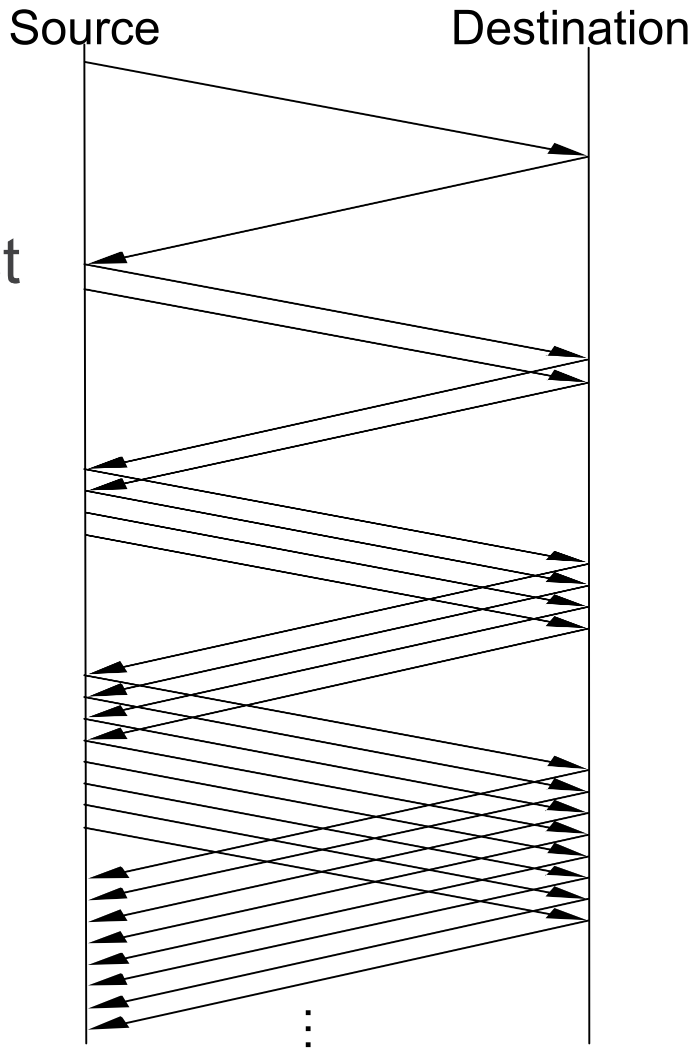
- If we have large actual window, should we send data in one shot?
  - No, use acks to clock sending new data

..Self-clocking

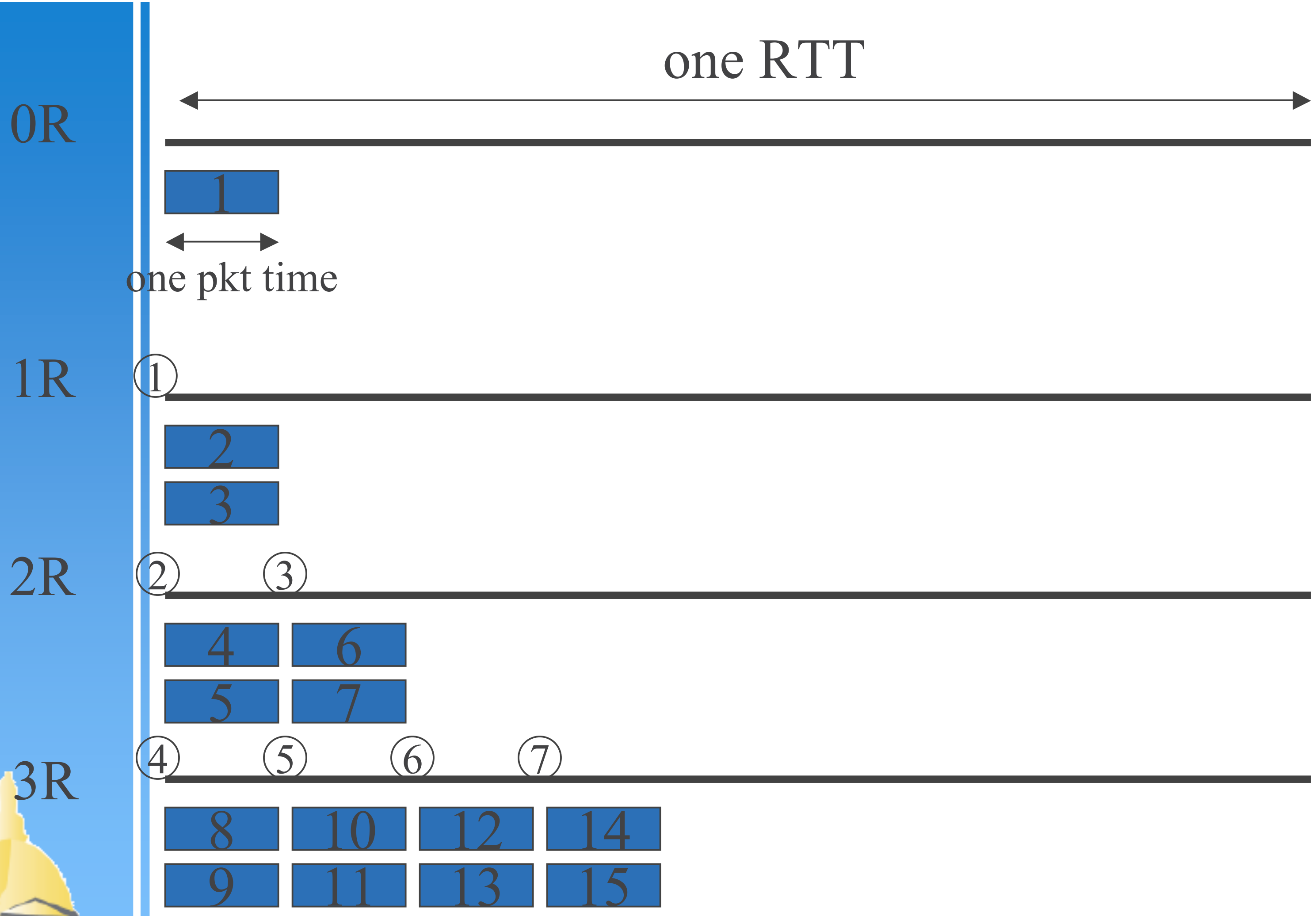


Slow Start

- AIMD is too slow to ramp up TCP performance
- Objective: determine the available capacity in the first
- Idea:
  - begin with CongestionWindow = 1 packet
  - double CongestionWindow each RTT (increment by 1 packet for each ACK)



Slow Start Example



one RTT

0R

1

one pkt time

1R ①

2

3

2R ② ③

4 6

5 7

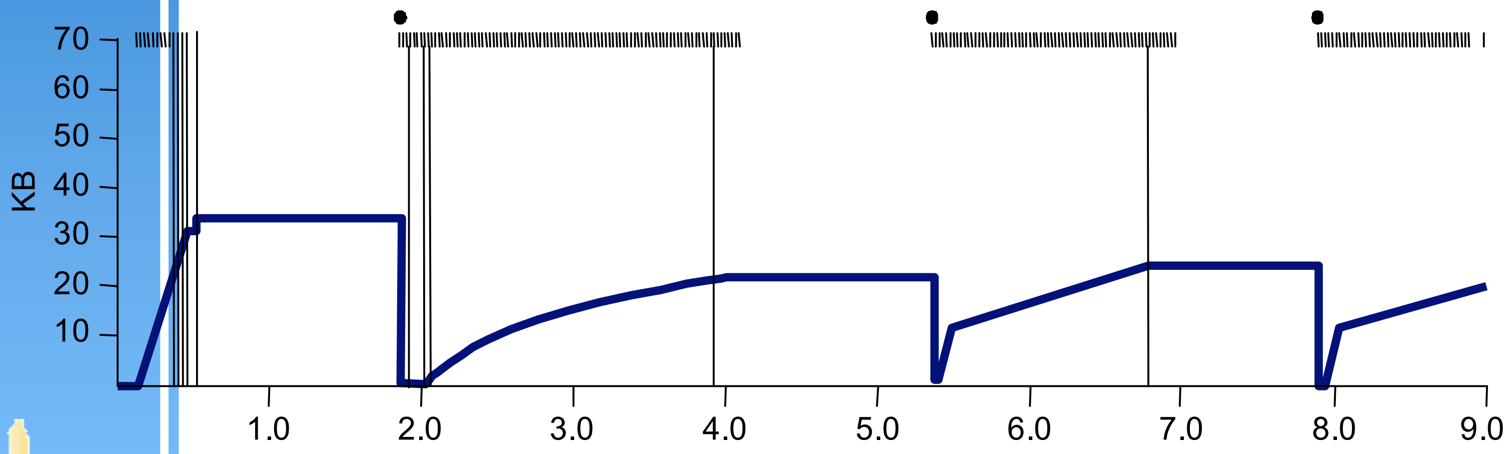
3R ④ ⑤ ⑥ ⑦

8 10 12 14

9 11 13 15

Slow Start (cont)

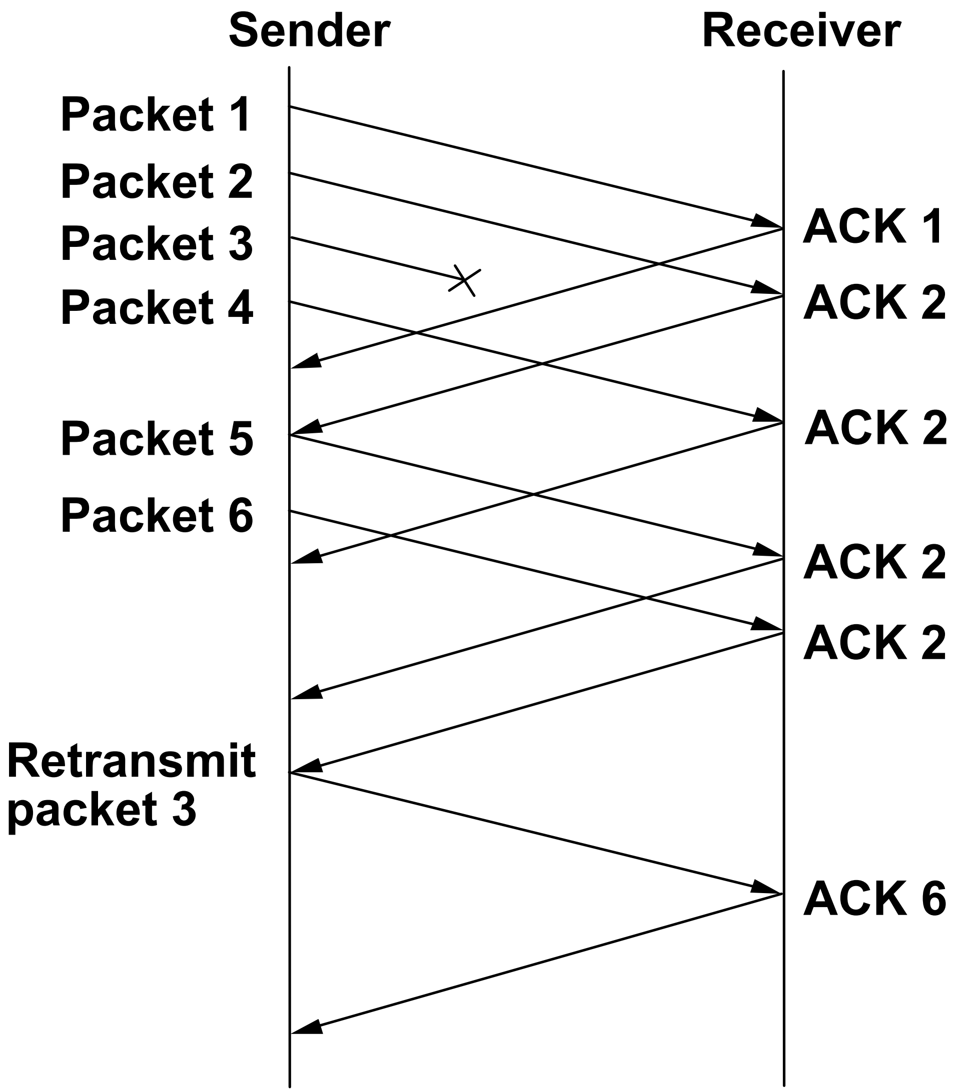
- Exponential growth, but slower than all at once
- Used…
  - when first starting connection
  - when connection goes dead waiting for timeout
- Trace



- Problem: lose up to half a CongestionWindow's worth of data

Fast Retransmit

- Problem: coarse-grain TCP timeouts lead to idle periods
- Fast retransmit: use duplicate ACKs to trigger retransmission



Sender

Receiver

Packet 1

Packet 2

Packet 3

Packet 4

Packet 5

Packet 6

Retransmit packet 3

ACK 1

ACK 2

ACK 2

ACK 2

ACK 2

ACK 6

Fast Retransmit

- If we get 3 duplicate acks for segment N
  - Retransmit segment N
  - Set ssthresh to 0.5*cwnd
  - Set cwnd to ssthresh + 3
- For every subsequent duplicate ack
  - Increase cwnd by 1 segment
- When new ack received
  - Reset cwnd to ssthresh (resume congestion avoidance)

Congestion Avoidance

- TCP needs to create congestion to find the point where congestion occurs

- Coarse grained timeout as loss indicator
- If loss occurs when cwnd = W
  - Network can absorb 0.5W ~ W segments
  - Set cwnd to 0.5W (multiplicative decrease)
  - Needed to avoid exponential queue buildup
- Upon receiving ACK
  - Increase cwnd by 1/cwnd (additive increase)
  - Multiplicative increase -> non-convergence

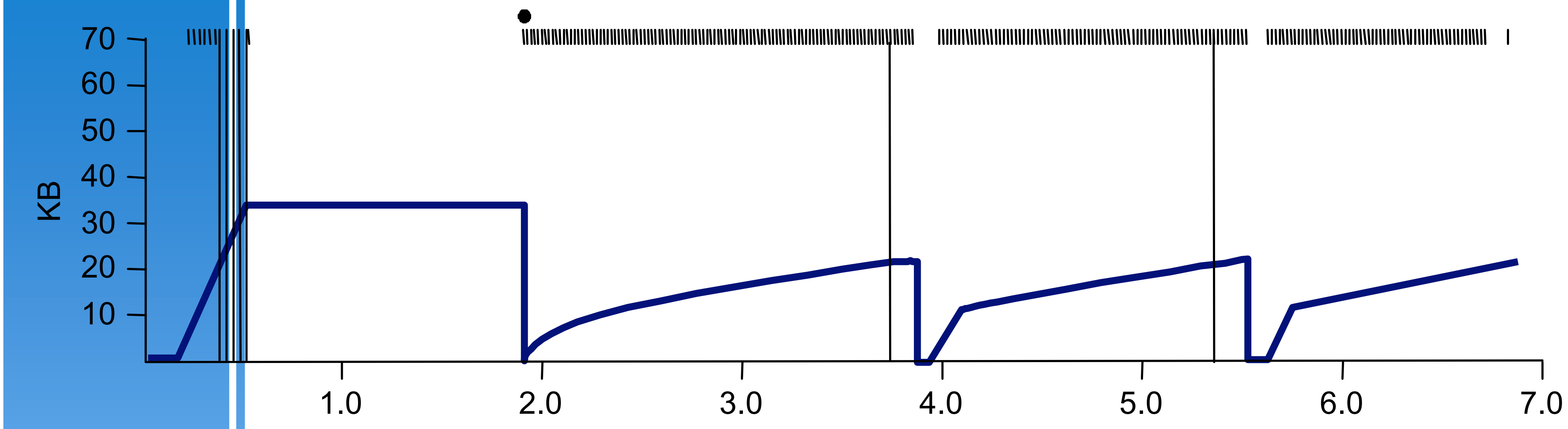
Slow Start and Congestion Avoidance

- If packet is lost we lose our self clocking as well
  - Need to implement slow-start and congestion avoidance together
- When timeout occurs set ssthresh to 0.5w
  - If cwnd < ssthresh, use slow start
  - Else use congestion avoidance

Fast Recovery

- In congestion avoidance mode, if duplicate acks are received, reduce cwnd to half
- If n successive duplicate acks are received, we know that receiver got n segments after lost segment:
  - Advance cwnd by that number

Results



- Fast recovery
  - skip the slow start phase
  - go directly to half the last successful **`CongestionWindow (ssthresh)`**

Impact of Timeouts

- Timeouts can cause sender to
  - Slow start
  - Retransmit a possibly large portion of the window
- Bad for lossy high bandwidth-delay paths
- Can leverage duplicate acks to:
  - Retransmit fewer segments (fast retransmit)
  - Advance cwnd more aggressively (fast recovery)