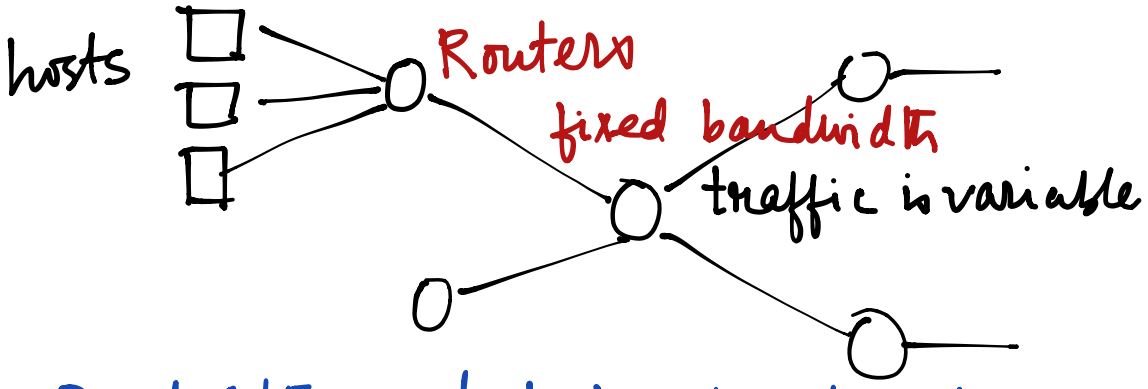
TCP Congestion Control

Issues: Efficiency and Fairness.



Bandwidth needs to be allocated dynamically to avoid congestion.

Bandwidth Allocation Model

recall

1. NW layer provides feedback
2. TRA layer adjusts the traffic [changes the window size]
3. Efficiency & Fairness

TCP: closed loop – uses network feedback to adjust the rate

host based – network layer devices

just pass the congestion information, doesn't adjust rates.

window based – the TCP sliding window size is adjusted through the feedback.

Additive Increase and Multiplicative Decrease

AIMD

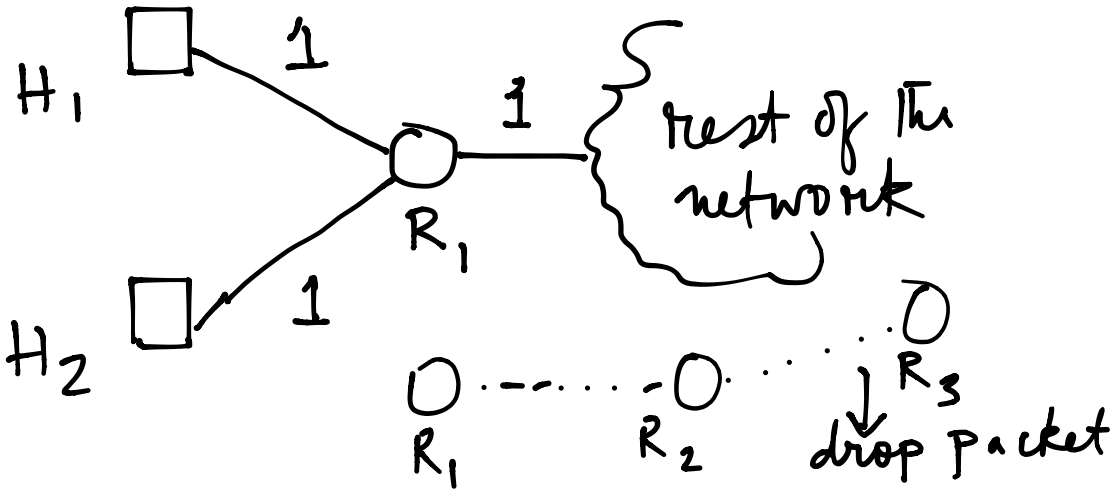
$$W_{t+1} = \begin{cases} W_t + a, & \text{if no congestion} \\ W_t \times b, & \text{otherwise} \end{cases}$$

W_t : window size at t

a, b are typically decided beforehand

$a = 1$ packet , $b = 1/2$ typical

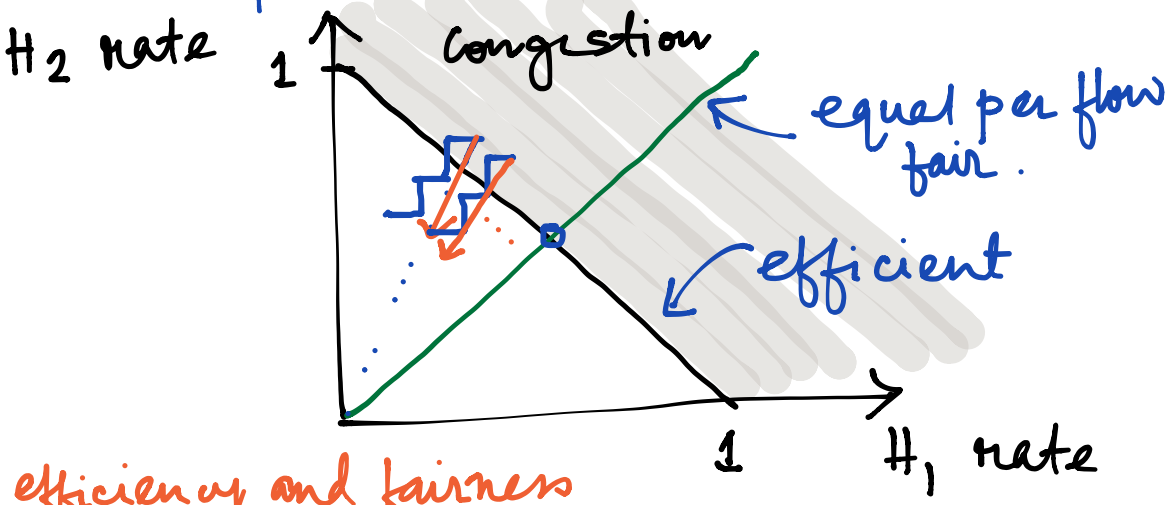
Ex:



Congestion could be at any other router but it will have a back-pressure till the host where the trouble is.

In the example, both hosts start with some Window which can cause congestion.

AIMD protocol



efficiency and fairness notions are local.

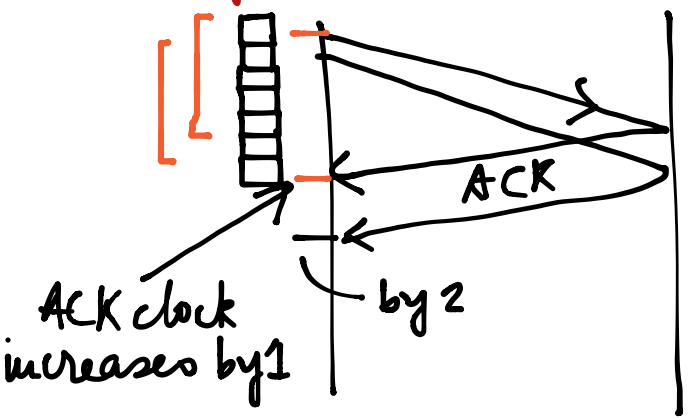
Note: Even if hosts are using dissimilar windows in the beginning, the protocol forces them to be such that the rate becomes equal - even when one uses the whole BW.

AIMD endnotes

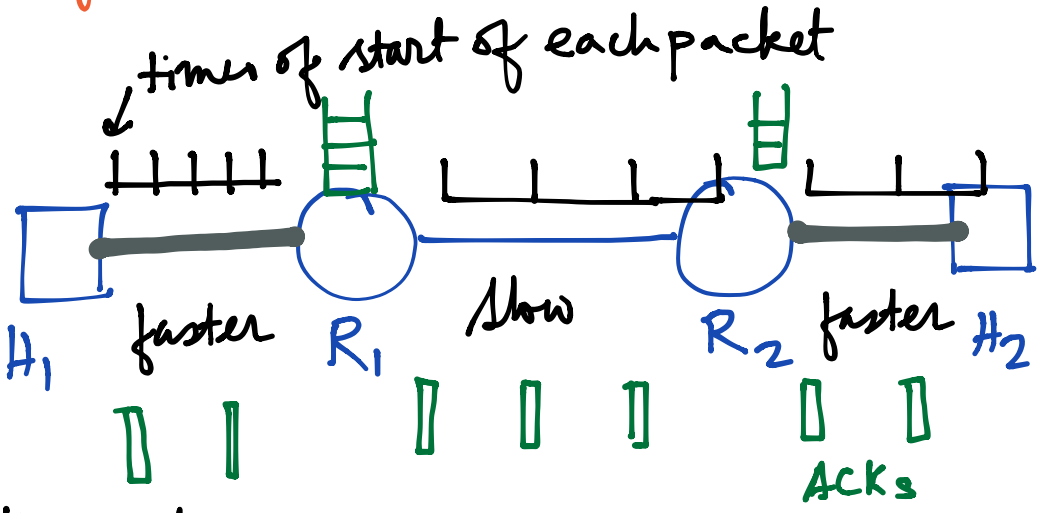
1. decentralized
2. convergence is guaranteed. [not the case for MIAD, MIMD, AIAD]
3. Needs simple binary feedback from the network

Feedback signal	Pros/cons	Example
packet loss	Easy to detect slow convergence	TCP NewReno Cubic TCP
packet delay	Early detection computation-heavy and may be inaccurate	Compound TCP
router feedback	Early detection require router to send info.	TCPs with explicit Congestion notification

Sliding Window ACK clocking



Why ACK clocking is useful?



the window will slide at this rate – automatically adjusting the rate of transmission.

- network smooths out the traffic
- ACK clocking ensures no routers are having buffer overflow.

TCP uses a sliding window because of ACK clocking

Sliding window controls how many segments in a window [the size is variable]

- called congestion window or cwnd
- Rate = $\frac{\text{cwnd}}{\text{RTT}}$

More details of TCP

How does TCP implement AIMD?

Stage 1: SLOW START → AI

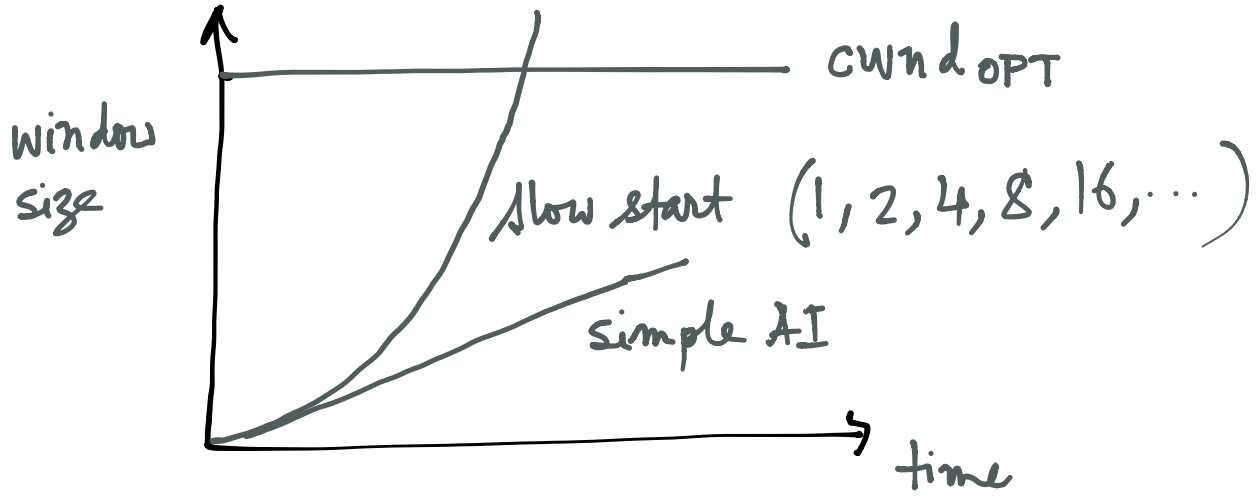
Recap
- Sender uses cwnd
- Sender uses packet loss as an indication of congestion
- TCP handles wide variety of rates and RTTs (need to dynamically adjust)

Problem: there is a $cwnd_{OPT}$, but how should TCP know it?

(1) Fixed window size – could be large or small

(2) AI from 1 could be too slow to reach the optimal

Slow Start (counterintuitive name) does it with exponential growth till a threshold called slow start threshold ssthres



The window size is doubled every time

but if the exponentiation is continued it will congest the network hugely since the increase of window size is large

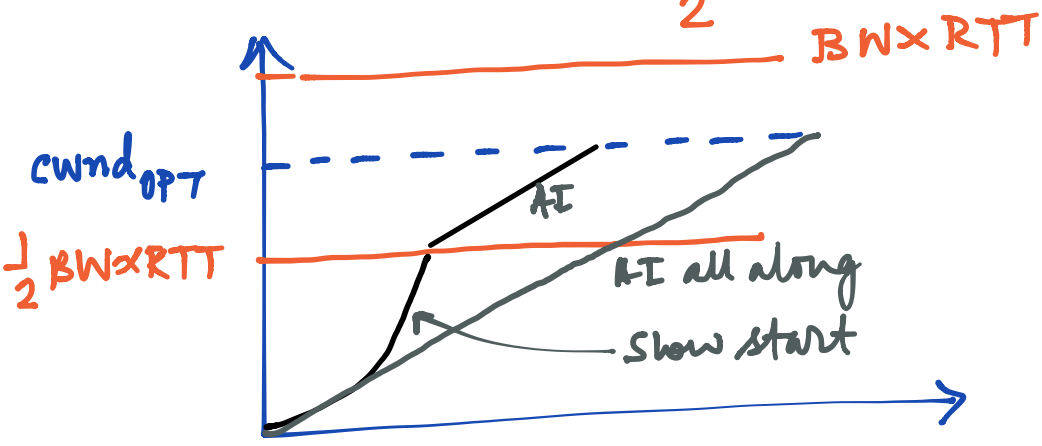
Switches to AI in advance

RTT is usually calculated between the packet and ACK

BW $\propto$ RTT $\approx$ reasonable proxy for the max window size

Threshold heuristic

$$ssthres = \frac{BW \times RTT}{2}$$



Example: TCP Tahoe

- Start with cwnd = 1

When cwnd < ssthres

- cwnd += 1 for every ACK.

  [When all ACKs for a window comes, the cwnd doubles]

when cwnd >= ssthres

- cwnd += $\frac{1}{\text{cwnd}}$ packets per ACK

  [increase one packet for the whole window]

  AI step

ssthres is also dynamic

if packet loss occurs

ssthres = $\frac{\text{cwnd}}{2}$

begin slow start after timeout with cwnd = 1 } not an MD step

---

Why not MD instead?

- loss of packet is detected via timeouts
  — which is a pessimistic estimate
- ACK clock synchronization is lost.

Therefore, no way to get back to the rate where the congestion occured.

If we want to retain the ACK clock sync and do an MD, then we need to detect loss before a timeout (avoidance rather a recovery – TCP Reno)

- Slow start is a reasonable compromise.

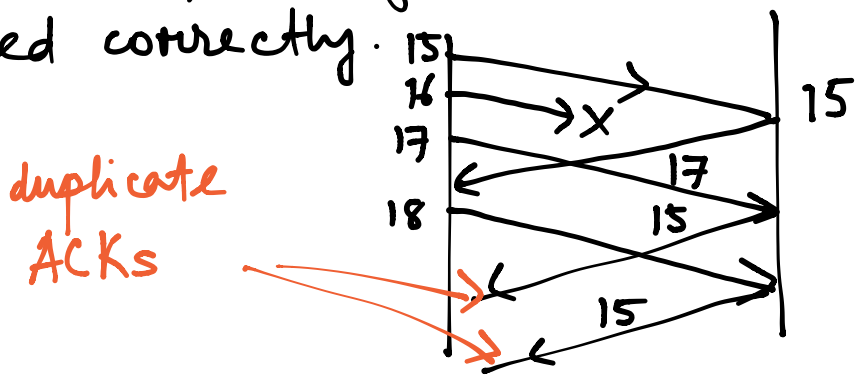
Stage 2: FAST RETRANSMIT & FAST RECOVERY (MD part)

TCP infers the loss from ACKs

uses cumulative ACKing

– sends the highest in-order seq. #

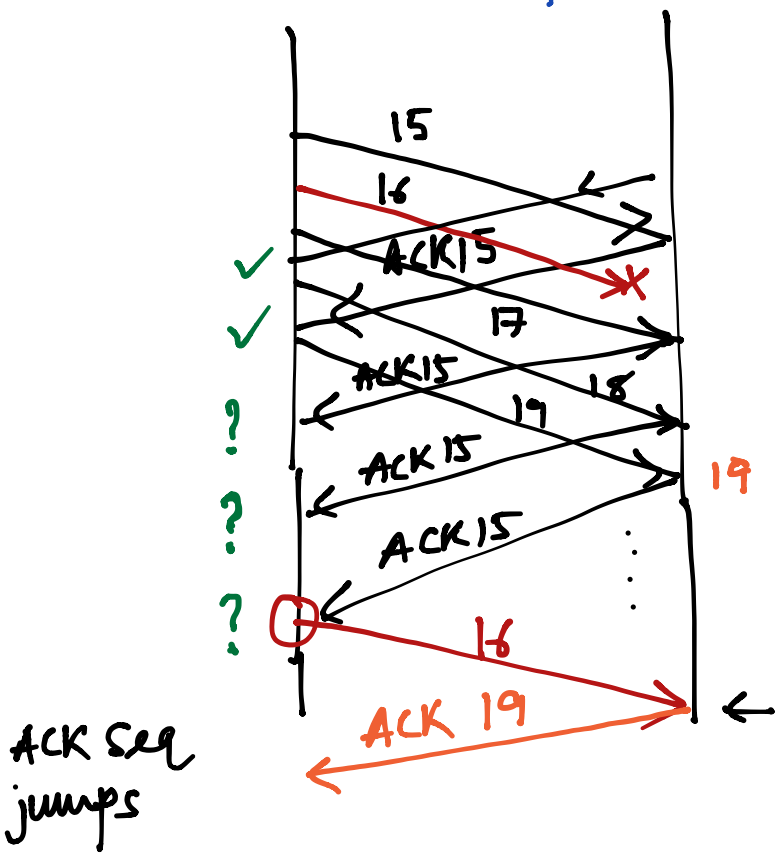
e.g., if packets till 15 was received correctly, it will send that ACK until it is received, even if 17, 18, ... are received correctly.



Duplicate ACKs hints that some data didn't reach. – i.e., the next packet

Fast Retransmit

- Treat 3 duplicate ACKs as the loss of the next packet.
- Retransmit that packet. (16 in our example)



ACK seq jumps

← retransmission fills the hole of 16.

Fast retransmit repairs single segment loss (can improvise this for multi segment loss)

- Still quiet time at sender/receiver when the retransmission is taking place.
- MD pending.

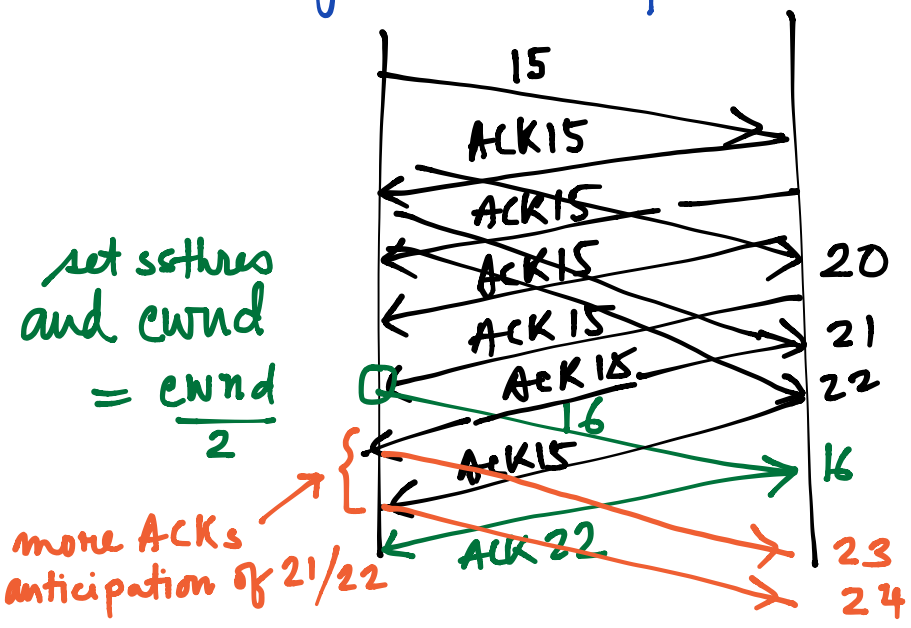
Observation: the duplicate ACKs say

(1) the next packet didn't arrive

(2) some later packet arrived. – i.e. no queuing overflow

Fast Recovery

- First fast retransmit, then MD.
- pretend duplicate ACKs are expected ACKs for the next packets.



With fast retransmit, fast recovery repairs a single segment loss quickly and keeps the ACK clock running.

While the 3rd duplicate ACK comes, it reduces the window by ½, not reset to 1.

Example: TCP Reno uses slow start, fast retransmit, fast recovery.

Endnotes:

- Tahoe, Reno repairs one loss per RTT
- For multiple losses other ACK heuristics are used
  - NewReno, e.g.
  - Selective ACK (SACK) another approach [range of bytes received]

Explicit Congestion Notification

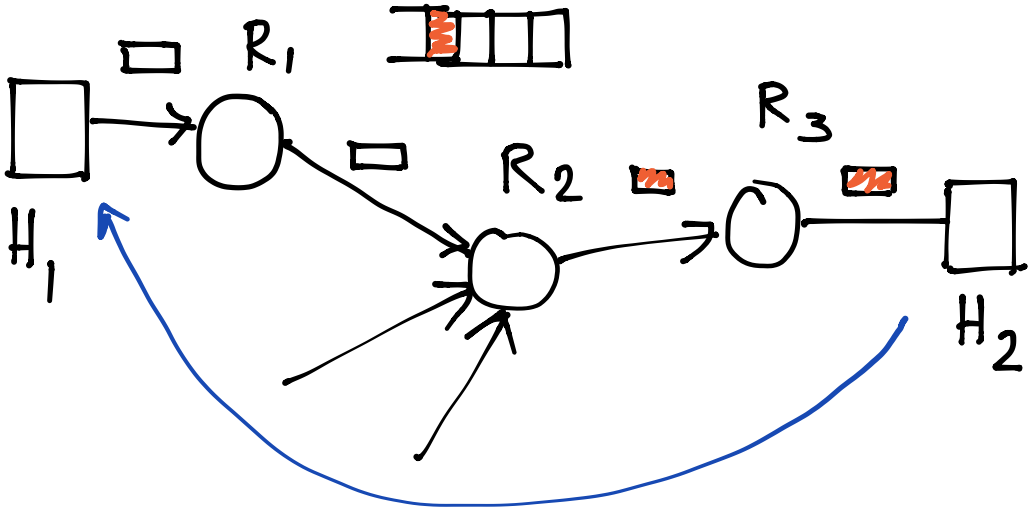
Routers explicitly notifying hosts about congestion.

- congestion control – driving the network towards congestion and then recovering from it.
- Congestion avoidance - shape the traffic in such a way that congestion doesn't occur.

Strategy of ECN

- Router detects congestion through its queue length
- when congested it marks every packet through a flag in the IP header (ECN)

EX.



receiving host sends the congestion notification when it sends ACK

- H_1 gets this and slows down.

Advantages

- congestion detected early, no retransmission needed
- no extra packet needs to be sent

disadvantage

- routers need to be upgraded