

H1(source#)-> H3 (destination#) -> flow 1

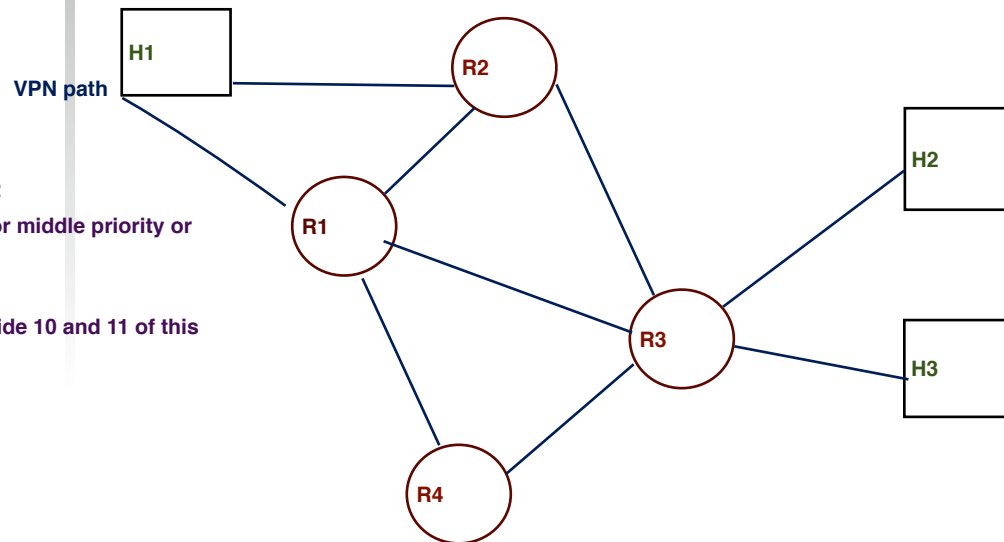
H1(source#)-> H2 (destination#) -> flow 2

classifier will define priority (high priority or middle priority or lowest priority)

P.S\* see more in notes with pictures of slide 10 and 11 of this chapter

## Chapter 6

# Resource Allocation and Congestion Control



# Problem

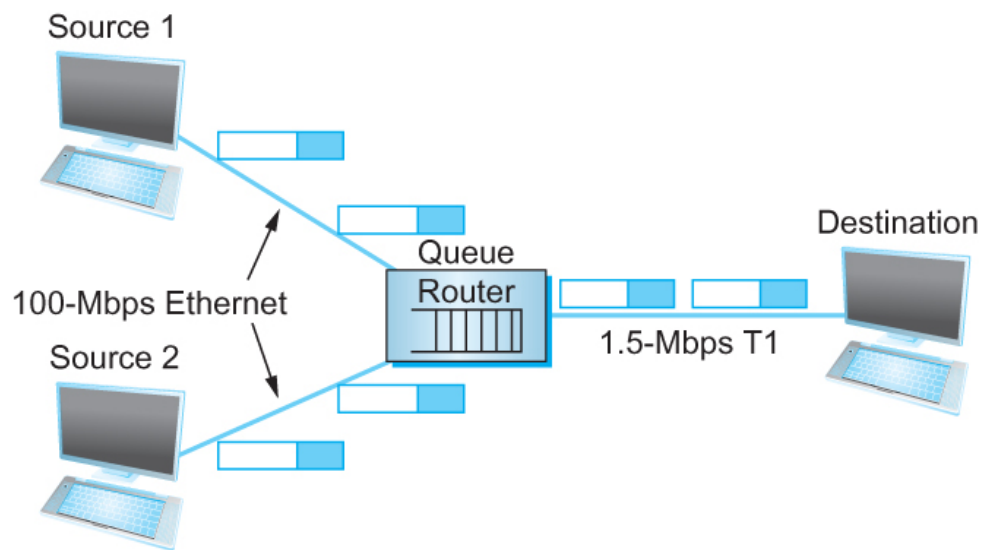
- We understand how data can be transferred among processes across heterogeneous networks
- Problem
  - How to *effectively* and *fairly* allocate resources among a collection of competing users?

# Chapter Outline

- Issues in Resource Allocation (6.1)
- Queuing Disciplines (6.2)
- TCP Congestion Control (6.3)
- Congestion Avoidance Mechanism (6.4)
- Quality of Service (6.5)

# Congestion Control and Resource Allocation

- Resources
  - Bandwidth of the links
  - Buffers at the routers and switches
- Packets compete at a router for the use of a link
- Each competing packet is placed in a queue waiting for its turn to be transmitted over the link.



# Congestion Control and Resource Allocation

- When too many packets are contending for the same link/buffer
  - The queue overflows
  - Packets get dropped
    - Network is congested!
- Network should provide a congestion control mechanism to deal with the a situation



# Congestion Control and Resource Allocation

- Congestion control and Resource Allocation
  - Two sides of the same coin
- If the network takes active role in allocating resources
  - The congestion may be avoided all together
  - So no need for congestion control
    - Meaning the congestion occurred so let's recover from it.

**Any network resource management strategy that has, as its goal, the alleviation or avoidance of congestion. A congestion-control mechanism may be implemented on the routers (switches) inside the network, by the hosts at the edges of the network, or by a combination of both.**

# ~~Congestion Control and Resource Allocation~~

- ~~Allocating resources with any precision is difficult~~
  1. ~~Dynamic nature of the network~~
  2. ~~Resources are distributed throughout the network~~
- ~~We can always start with letting the sources send as much data as they want~~
  1. ~~Let the congestion occur~~
  2. ~~Detect it~~
  3. ~~Then recover from it~~
- ~~Easier approach but it can be disruptive~~  
(why?)

# Congestion Control and Resource Allocation

- Congestion control and resource allocations involve both hosts and network elements  
**such as router**
- In network elements
  - Various queuing disciplines can be used to control the order in which packets get transmitted and which packets get dropped
- At the end hosts
  - The congestion control mechanism paces how fast sources are allowed to send packets



# Issues in Resource Allocation

## ■ ~~Network Model~~

### ■ ~~Packet Switched Network~~

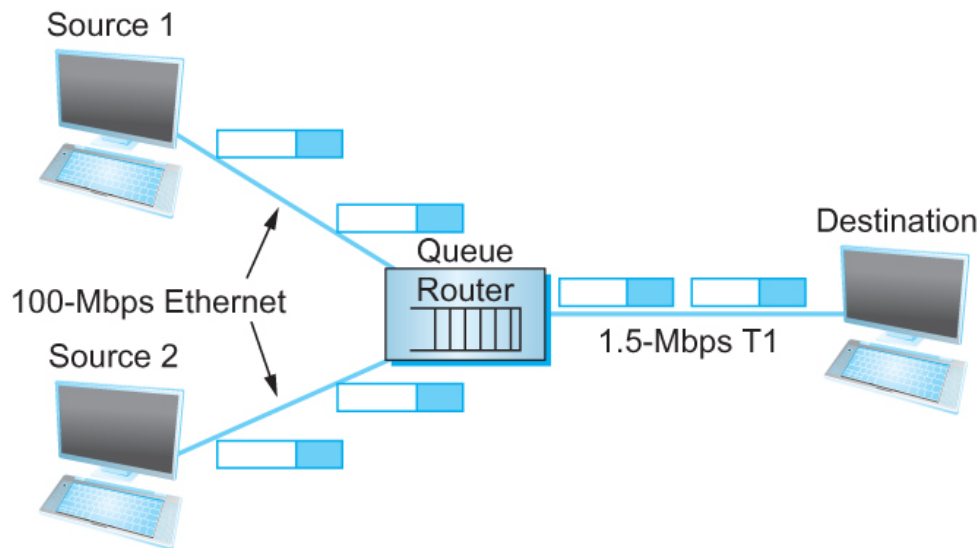
- ~~Multiple links and switches (or routers).~~

- ~~In such an environment,~~

- ~~Source may have enough capacity on the immediate outgoing link to send a packet.~~
- But, further ahead, its packets may encounter a bottleneck.
  - ~~A link that is being used by many different traffic sources~~


# Issues in Resource Allocation

- Network Model
  - Packet Switched Network



A potential bottleneck router.

## ■ Classifications

1. Router-centric versus Host-centric
2. Reservation-based versus Feedback-based 
3. Window-based versus Rate-based

skipped to here slide no#14

## 1. Router-centric versus Host-centric

- In a router-centric design, each router decides when packets are forwarded/dropped
  - Advantages: can be more aggressive, has a complete picture of network traffic
  - Disadvantages: expensive, algorithms difficult to implement in hardware
- In a host-centric design, the end hosts observe the network conditions and adjust their behavior accordingly.
  - Advantages: cheap, scalable
  - Disadvantage: requires cooperation
- Not mutually exclusive.



## 2. Reservation-based versus Feedback-based

- In a reservation-based system, some entity (e.g., the end host) asks the network for a certain amount of capacity to be allocated for a flow along the way.
- In a feedback-based approach, the end hosts begin sending data without first reserving any capacity and then adjust their sending rate according to the feedback they receive.
  - **Explicit** (send explicit message to the source by the router to slow down)
  - **Implicit** (drop packets to implicitly let the source know of a network congestion)

# Resource Allocation Mechanisms

3. **Window-based** versus Rate-based
  - **Window advertisement** is used within the network to reserve buffer space.
  - Control sender's behavior using a rate
    - How many bit per second the receiver or network is able to absorb.
    - **Single-rate** versus **Multi-rate**
      - Meaningful when considering multicast
      - Single-rate sends data to each client at the same rate
      - Multi-rate sends data to each client at whatever rate is best for that client

# Evaluation Criteria

## ■ Effective Resource Allocation

- A good starting point for evaluating the effectiveness of a resource allocation scheme is to consider the two principal metrics of networking:

- Throughput
- Delay. use RTT to calculate delay

Power = throughput/ Delay

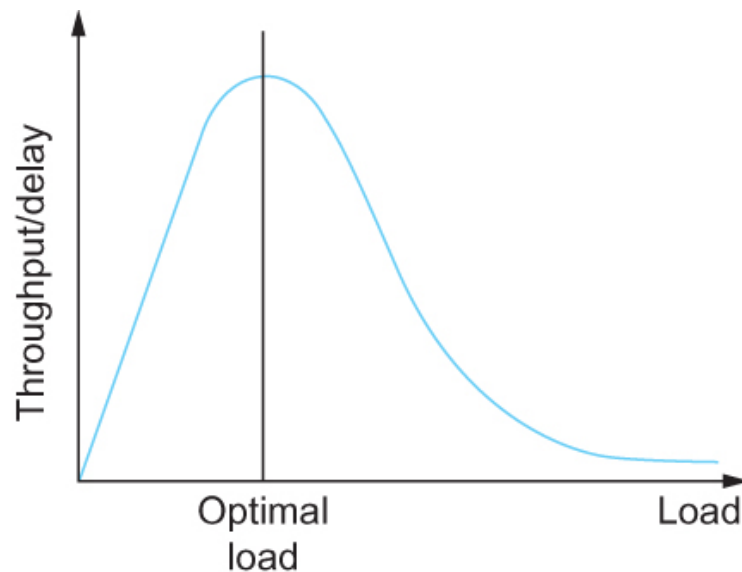
# Evaluation Criteria

- Effective Resource Allocation
  - Some network designers use the ratio of throughput to delay as a metric for evaluating the effectiveness of a resource allocation scheme.
  - $\text{Power} = \text{Throughput} / \text{Delay}$



# Evaluation Criteria

## ■ Effective Resource Allocation



Ratio of throughput to delay as a function of load

# Evaluation Criteria

## ■ Fair Resource Allocation

- We must also consider the issue of fairness.
  - So what exactly constitutes fair resource allocation.
- A reservation-based resource allocation scheme, for example, provides an explicit way to create controlled unfairness.
- We might use reservations to enable a video stream to receive 1 Mbps across some link while a file transfer receives only 10 Kbps over the same link.

**skipped**

# Evaluation Criteria

## ■ Fair Resource Allocation

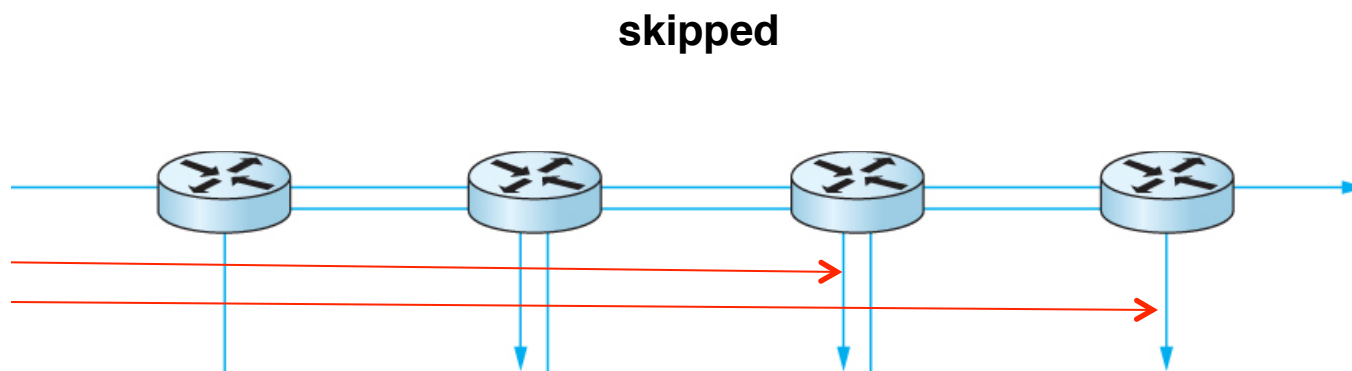
- When several flows share a particular link, we would like for each flow to receive an equal share of the bandwidth.
  - This definition presumes that a *fair share of bandwidth means an equal share of bandwidth*.
- But even in the absence of reservations, equal shares may not equate to fair shares.
  - Should we also consider the length of the paths being compared?

**skipped**

# Evaluation Criteria

## ■ Fair Resource Allocation

- Should we also consider the length of the paths being compared?
- What is fair when one four-hop flow is competing with three-hop flows?



One four-hop flow competing with three one-hop flows

# Evaluation Criteria

## ■ Fair Resource Allocation

- Assuming that fair implies equal and that all paths are of equal length, networking researcher Raj Jain proposed a metric that can be used to quantify the fairness of a congestion-control mechanism.

$$f(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2}$$

skipped

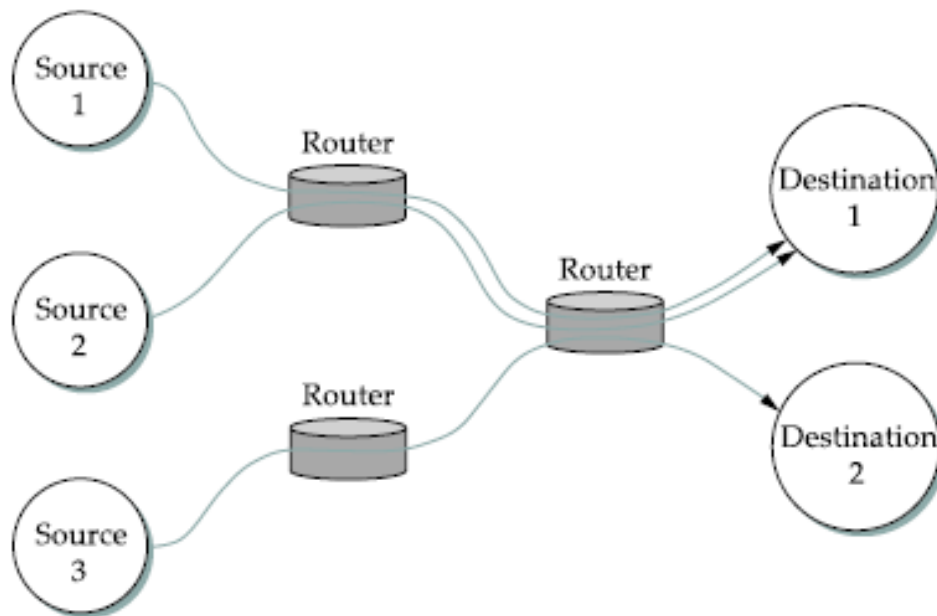
For  $x_1 = x_2 = \dots = x_n = 1$

$f(x_1, x_2, \dots, x_n) = n^2 / (n \cdot n) = 1$

# Connectionless Flows

- The network is essentially connectionless, with any connection-oriented service implemented in the transport protocol that is running on the end hosts.
  - Our classification of networks as being either connectionless or connection-oriented is a bit too restrictive.
  - The assumption that all datagrams are completely independent in a connectionless network is too strong.  
**wrong**
  - Usually a stream of datagrams between a particular pair of hosts (connection-oriented) flows through a particular set of routers (connectionless).

# Connectionless Flows



Multiple flows passing through a set of routers


# Connectionless Flows

- Flows can be defined at different granularities:
  - A flow can be host-to-host (flow)
  - A flow can be process-to-process (channel)
- In process-to-process, a flow is essentially the same as a channel
- Flow is visible to the routers inside the network, whereas a channel is an end-to-end (process-to-process) abstraction.

Flow is controlled by the destination, congestion is controlled by the source



# Connectionless Flows

- Makes sense to maintain some soft state information for each flow
  - Used to make resource allocation decisions on multiple flows passing through the router
- The difference between soft state and “hard” state is that soft state need not always be explicitly created and removed by signaling. 

# Connectionless Flows

- Soft state represents a middle ground
- In general, the correct operation of the network does not depend on soft state being present
- But when a packet happens to belong to a flow then the router is better able to handle the packet.

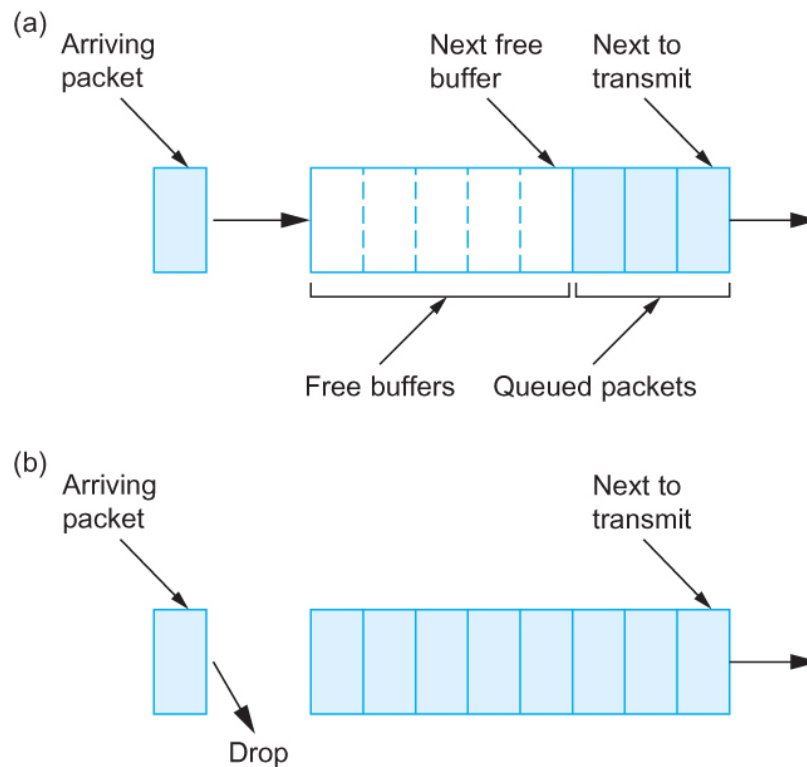
# Queuing Disciplines

- Each router must implement some queuing discipline, Why?
  - why need buffer: bcoz we have fast incoming link and slower outgoing link, it's there for incoming packets not to dropped
  - Packets entering from multiple inputs all wanting to get out of same port
  - Faster (higher BW) router input link but slower (lower BW) output link
  - To absorb short-term bursts
- There are two common queuing algorithms:
  - First-in-first-out      **Queuing Disciplines-1**
    - Simple FIFO
    - FIFO with multiple priority queues
  - Fair queuing      **Queuing Disciplines-2**

# Queuing Disciplines

- The idea of FIFO queuing, also called first-come-first-served (FCFS) queuing, is simple:
  - The first packet that arrives at a router is the first packet to be transmitted
    - FIFO scheduling discipline
  - If input > output, very soon the router queue gets full
  - If a packet arrives and queue is full, the router discards that packet regardless of its flow or priority
    - Tail drop policy
  - Note that tail drop and FIFO are two separable ideas.
    - FIFO is a *scheduling discipline*
    - Tail drop is a *drop policy*

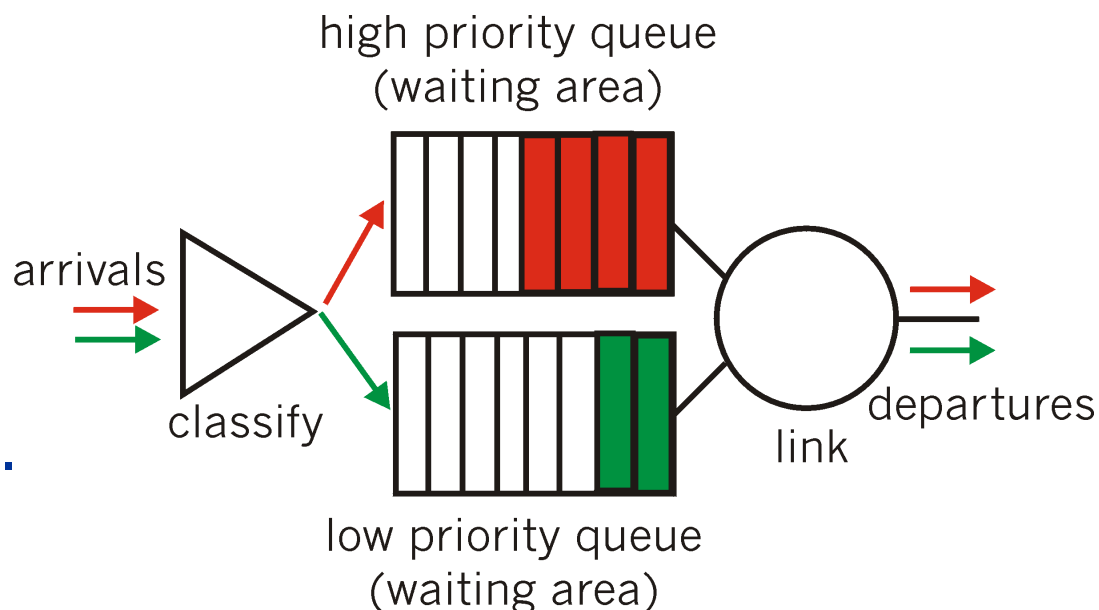
# Queuing Disciplines



(a) FIFO queuing discipline; (b) tail drop policy

# Queuing Disciplines

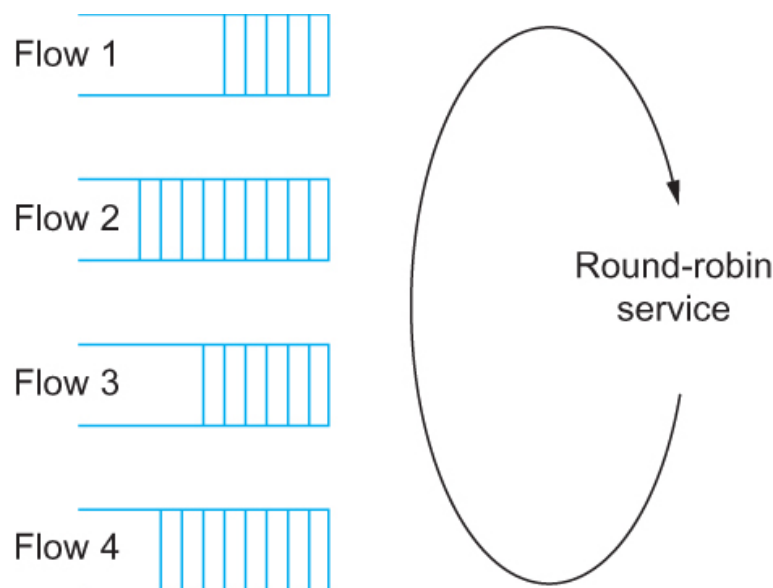
- Priority queuing.
- Multiple FIFO queues, one for each priority class.
- All packets of the highest-priority queue are transmitted first before moving on to the next priority queue.
- Within each priority, packets are still managed in a FIFO manner.



# Queuing Disciplines

## ■ Fair Queuing

- Maintain a separate queue for each flow currently being handled by the router.
- Services these queues in a sort of round-robin.



Round-robin service of four flows at a router

# Queuing Disciplines

## ■ Fair Queuing

- To truly allocate the bandwidth of the outgoing link in a fair manner, it is necessary to take packet length<sup>size</sup> into consideration.
  - For example, if a router is managing two flows, one with 1000-byte packets and the other with 500-byte packets, then a simple round-robin servicing of packets from each flow's queue will give the first flow two thirds of the link's bandwidth and the second flow only one-third of its bandwidth.



# Queuing Disciplines

## ■ Fair Queuing

- What we really want is bit-by-bit round-robin.
- Clearly, it is not feasible to interleave the bits from different packets.
- The FQ mechanism therefore simulates this behavior by first determining when a given packet would finish being transmitted if it were being sent using bit by bit round robin ( $F_i$ ), and then using this finishing time to sequence the packets for transmission.

# Queuing Disciplines

## ■ Fair Queuing

- To understand the algorithm for approximating bit-by-bit round robin, consider the behavior of a single flow
- For this flow, let
  - $P_i$ : denote the length of packet  $i$
  - $S_i$ : time when the router starts to transmit packet  $i$
  - $F_i$ : time when router finishes transmitting packet  $i$
  - Clearly,  $F_i = S_i + P_i$

# Queuing Disciplines

## ■ Fair Queuing

- When do we start transmitting packet  $i$ ?
  - Depends on whether packet  $i$  arrived before or after the router finishes transmitting packet  $i-1$  for the flow
- Let  $\Lambda_i$  denote the time that packet  $i$  arrives at the router
- Then  $S_i = \max(F_{i-1}, \Lambda_i)$
- $F_i = \max(F_{i-1}, \Lambda_i) + P_i$

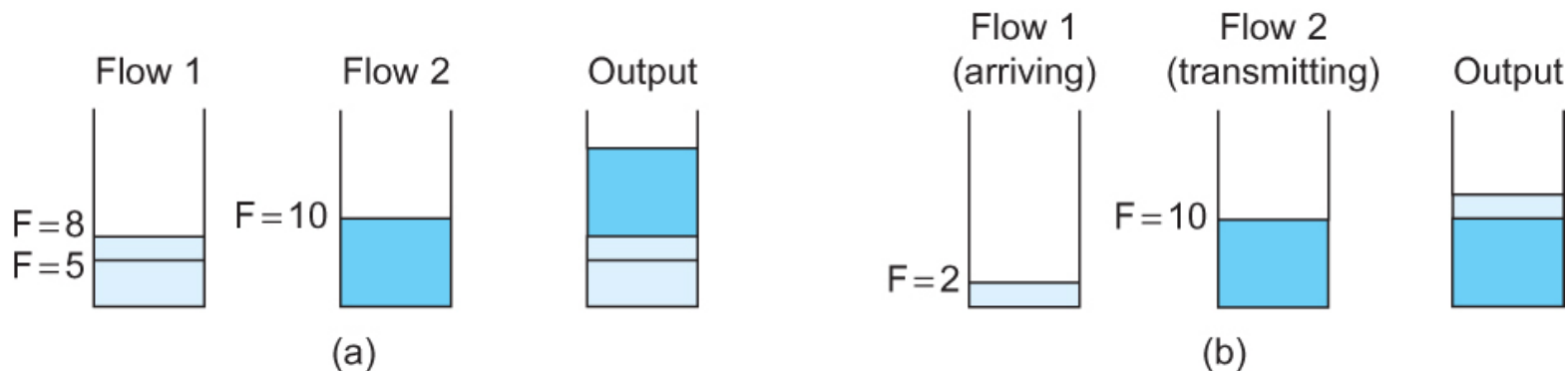
# Queuing Disciplines

## ■ Fair Queuing

- ~~Now for every flow, we calculate  $F_i$  for each packet that arrives using our formula~~
- ~~We then treat all the  $F_i$  as timestamps~~
  - ~~Higher  $F_i$  indicates later arriving packet~~
- ~~Next packet to transmit is always the packet that has the lowest timestamp (arrived earlier)~~
  - ~~The packet that should finish transmission before all others~~

# Queuing Disciplines

## ■ Fair Queuing

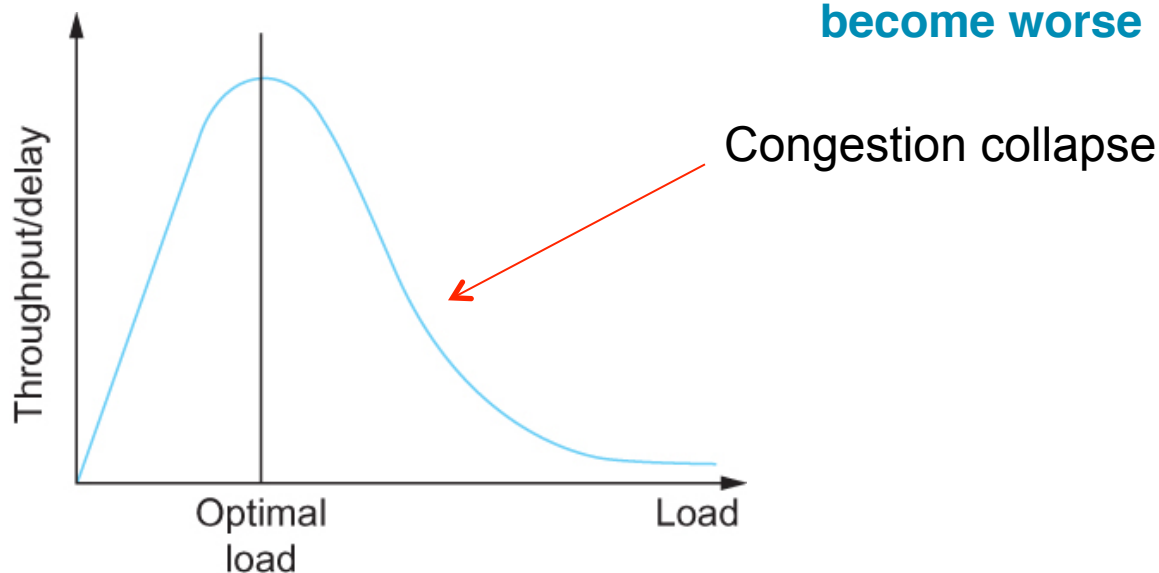


Example of fair queuing in action: (a) packets with earlier finishing times are sent first; (b) sending of a packet already in progress is completed

# TCP Congestion Control

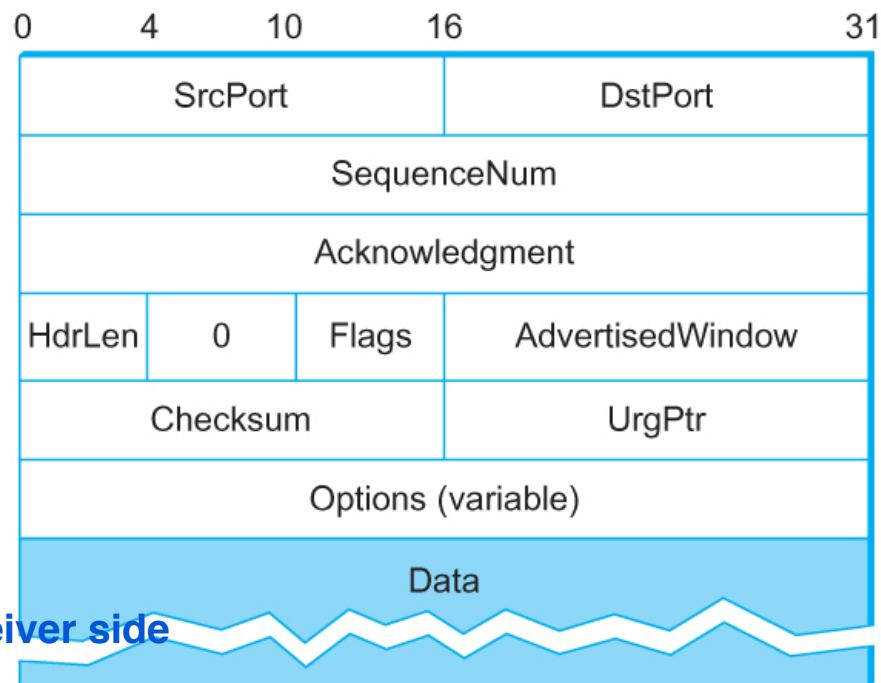
- In the late 1980s the Internet suffered a congestion collapse
- Wikipedia:
  - “In data networking and queueing theory, **network congestion** occurs when a link or node is carrying so much data that its quality of service deteriorates.”

become worse



# TCP Congestion Control

- Hosts would send their packets into the Internet as fast as the advertised window would allow
  - Congestion would occur at some router (causing packets to be dropped)
  - The hosts would time out and retransmit their packets, resulting in even more congestion.



congestion window is from host side

advertised window from destination/receiver side

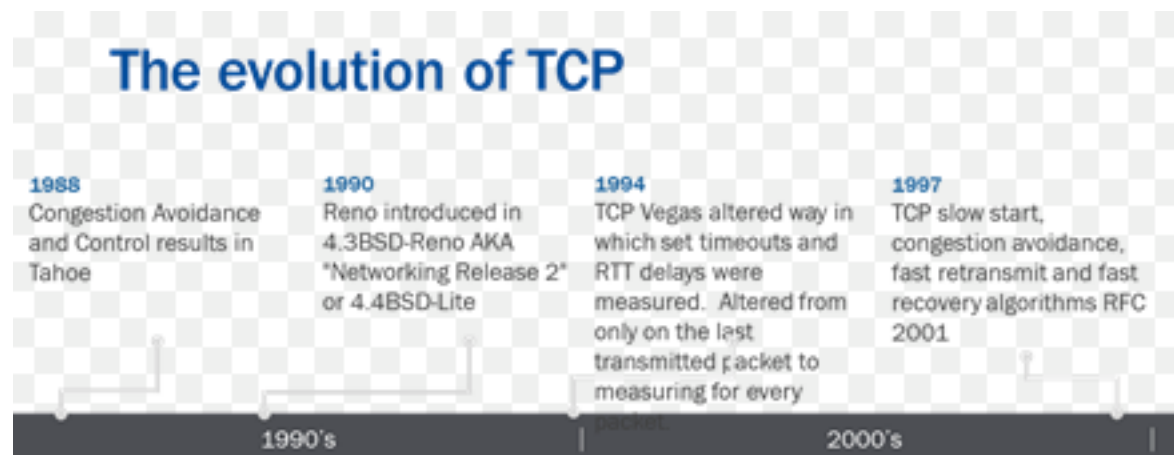
# TCP Congestion Control

- Two questions:
  1. How can we detect congestion?
  2. How to make flow rate sensitive to congestion level?
- TCP congestion control was introduced into the Internet in the late 1980s by Van Jacobson
  - TCP/IP protocol stack became operational in 1973
  - TCP Tahoe in 1988 (AIMD & Slow Start)
  - TCP Reno in 1990 (Fast Retransmit and Fast Recovery)
  - TCP Vegas in 1993 (Congestion Avoidance)



# TCP Congestion Control

- The main reason we can use the Internet successfully today



# TCP Congestion Control

- Recall resource allocation classifications
  1. Router-centric versus Host-centric
  2. Reservation-based versus Feedback-based
  3. Window-based versus Rate-based
- TCP uses host-centric, feedback-based and window-based resource allocation method.

nature of TCP communication:

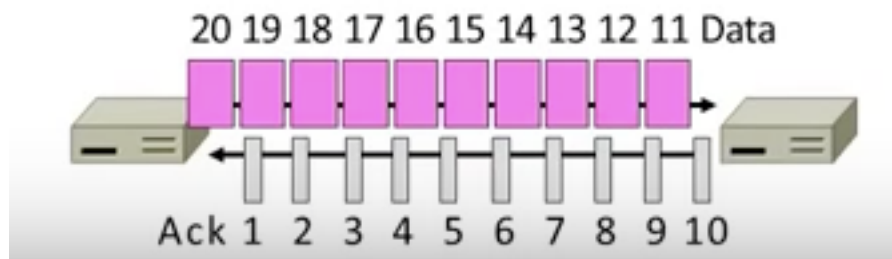
wait for first ACK of full window before I sent new set, allow to make full window by using self-clocking. ==> next page

# TCP Congestion Control

- So how can the source detect congestion so it can then adjust its flow rate accordingly?
  - Each time the source sends a packet to the destination over the network, an ACK signal is transmitted by the destination
  - If no ACK is received by the source a timeout will occur indicating packet is lost => congestion
  - On the other hand, the arrival of an ACK signals the source (sender) that one of its packets has left the network (reached the destination)
    - Therefore safe to insert a new packet into the network without adding to the level of congestion.
  - By using ACKs to pace the transmission of packets, TCP is said to be *self-clocking*.

# TCP Congestion Control

- By using ACKs to pace the transmission of packets, TCP is said to be *self-clocking*.

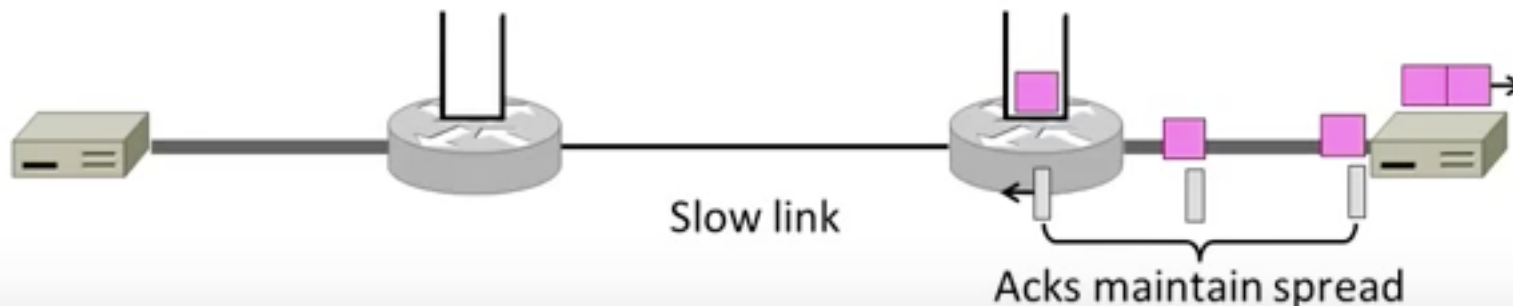
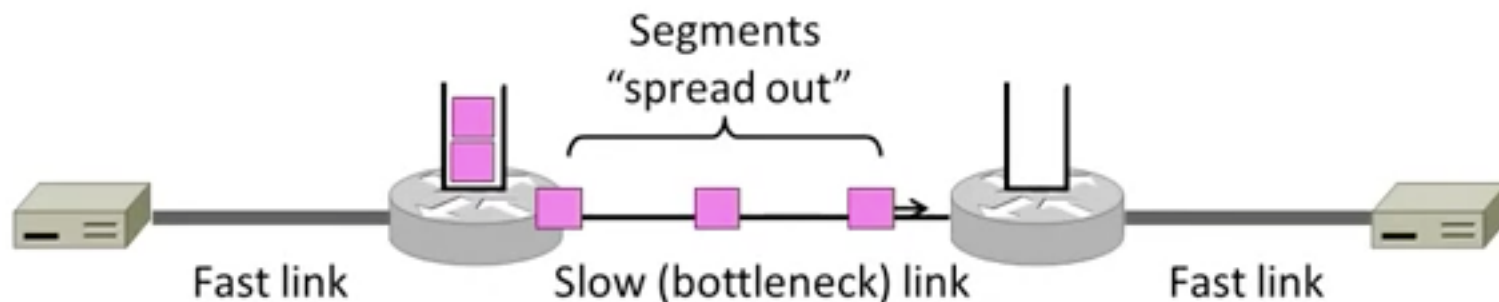
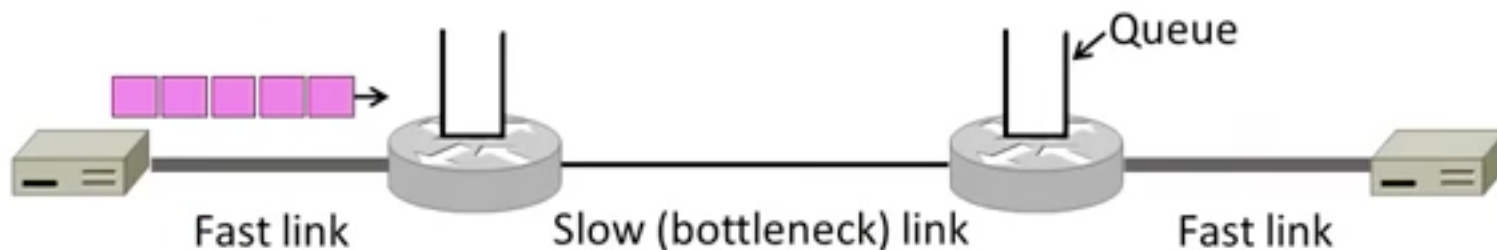


- What is the size of the sliding window here? **10 now**
- What if we have a congestion in the network due to a slow link?



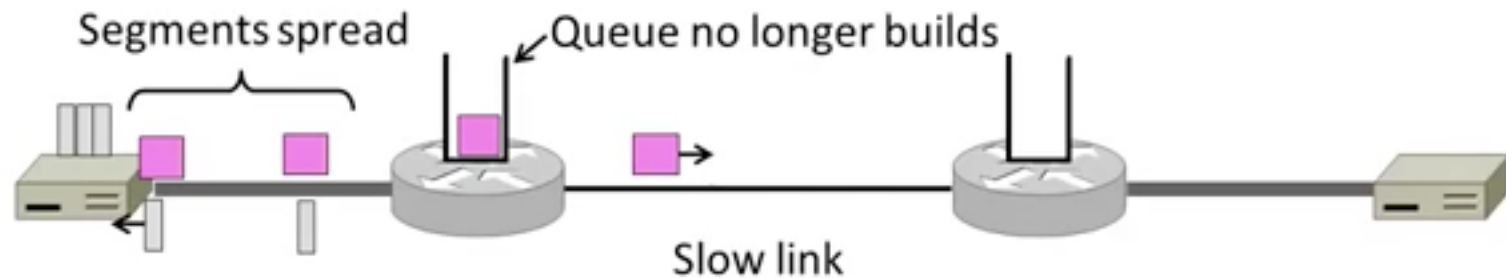
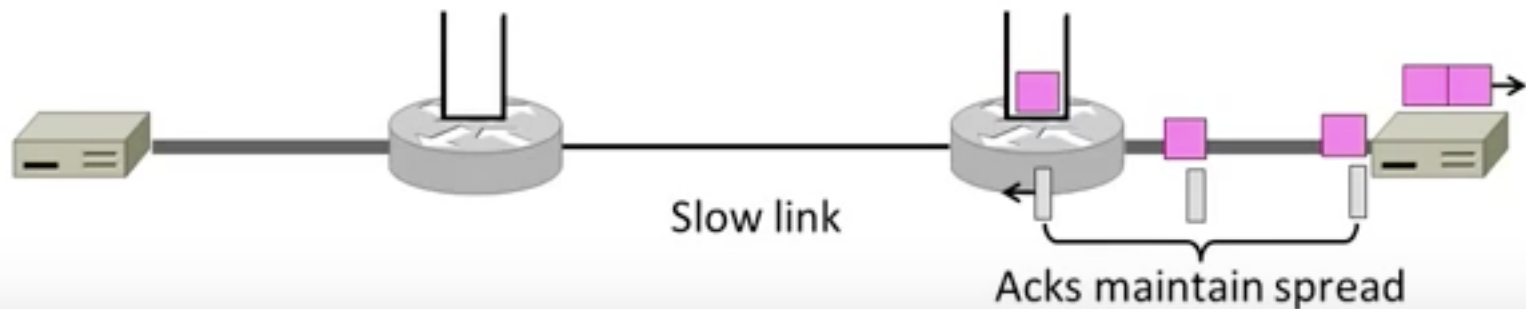
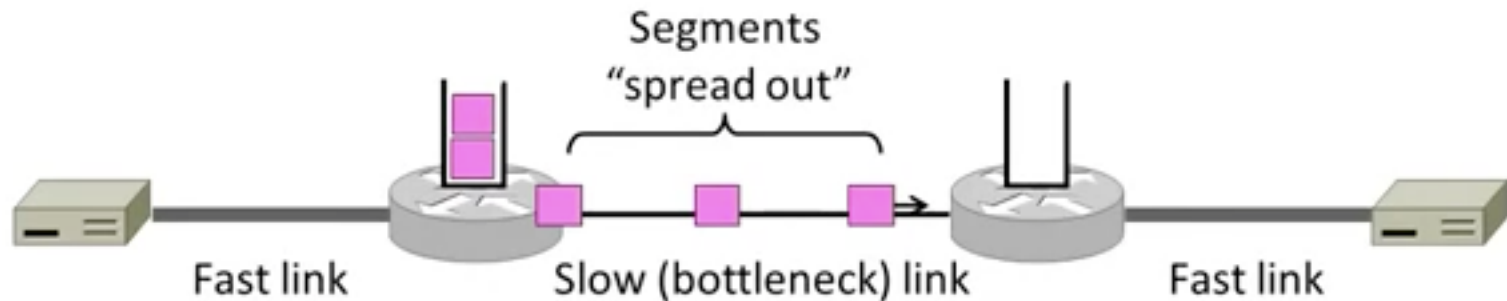
# TCP Congestion Control

- By using ACKs to pace the transmission of packets, TCP is said to be *self-clocking*.



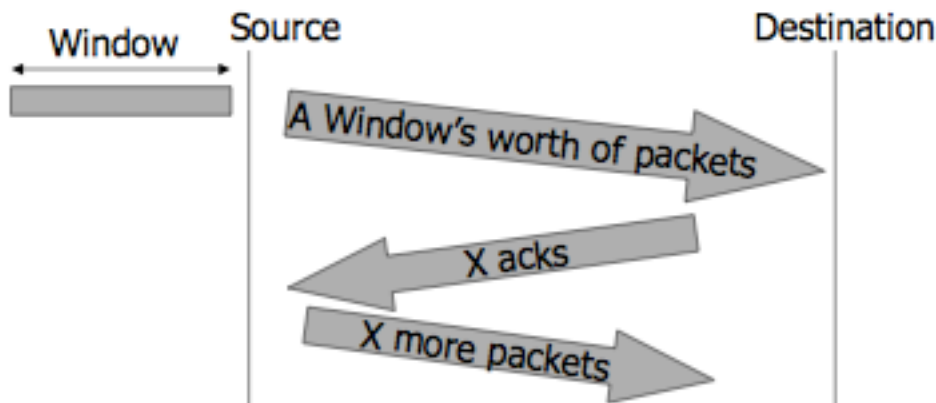
# TCP Congestion Control

- By using ACKs to pace the transmission of packets, TCP is said to be *self-clocking*.



# TCP Congestion Control

- Source's congestion window size determines the number of sender's packets in flight (in transit)



- Reduce window -> less packets in the network
  - Increase window -> more packets in the network
  - Congestion window (cwnd) is maintained by the sender, unlike advertised window which is maintained by the receiver
- Flow is controlled by destination, congestion is controlled by the source

# TCP Congestion Control

- Elements of congestion control in TCP
  - Additive increase, multiplicative decrease (AIMD)
  - Slow start
  - Fast retransmit and fast recovery



# TCP Congestion Control

- Additive Increase Multiplicative Decrease (AIMD)
  - TCP source uses *CongestionWindow* (cwnd) to limit how much data it is allowed to have in transit at a given time.
  - The congestion window is congestion control's counterpart to flow control's advertised window.
  - TCP is modified such that the **maximum** number of bytes of unacknowledged data allowed is now the **minimum of the congestion window and the advertised window**
    - $\text{MAX Bytes in Transit} = \text{MIN}(\text{cwnd}, \text{awnd})$

# TCP Congestion Control

## ■ AIMD Algorithm

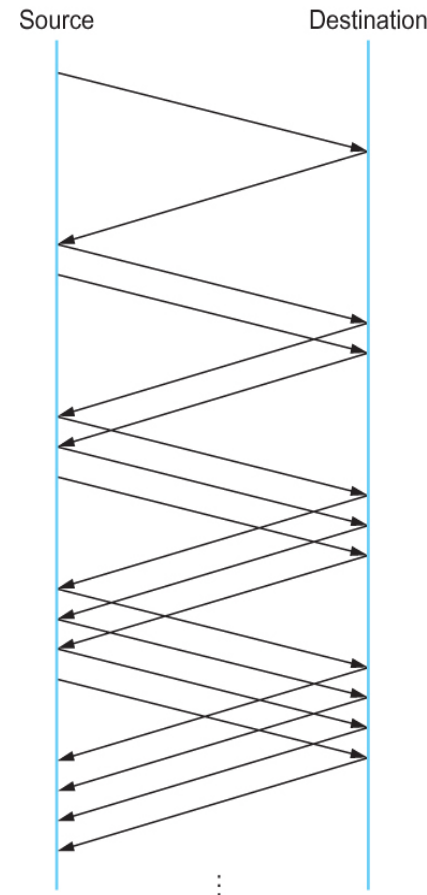
- Increment CongestionWindow (cwnd) by one packet per RTT (linear increase)

- Send 1 packet, receive 1 ack
- Send 2 packets, receive 2 acks
- Send 3 packets, receive 3 acks
- Send 4 packets, receive 3 acks

depending on network limit capacity could be 1000

- Divide CongestionWindow by two whenever a timeout occurs

- Recall that a timeout indicates a packet loss due to congestion
- Send 2 packets, receive 2 acks



# TCP Congestion Control

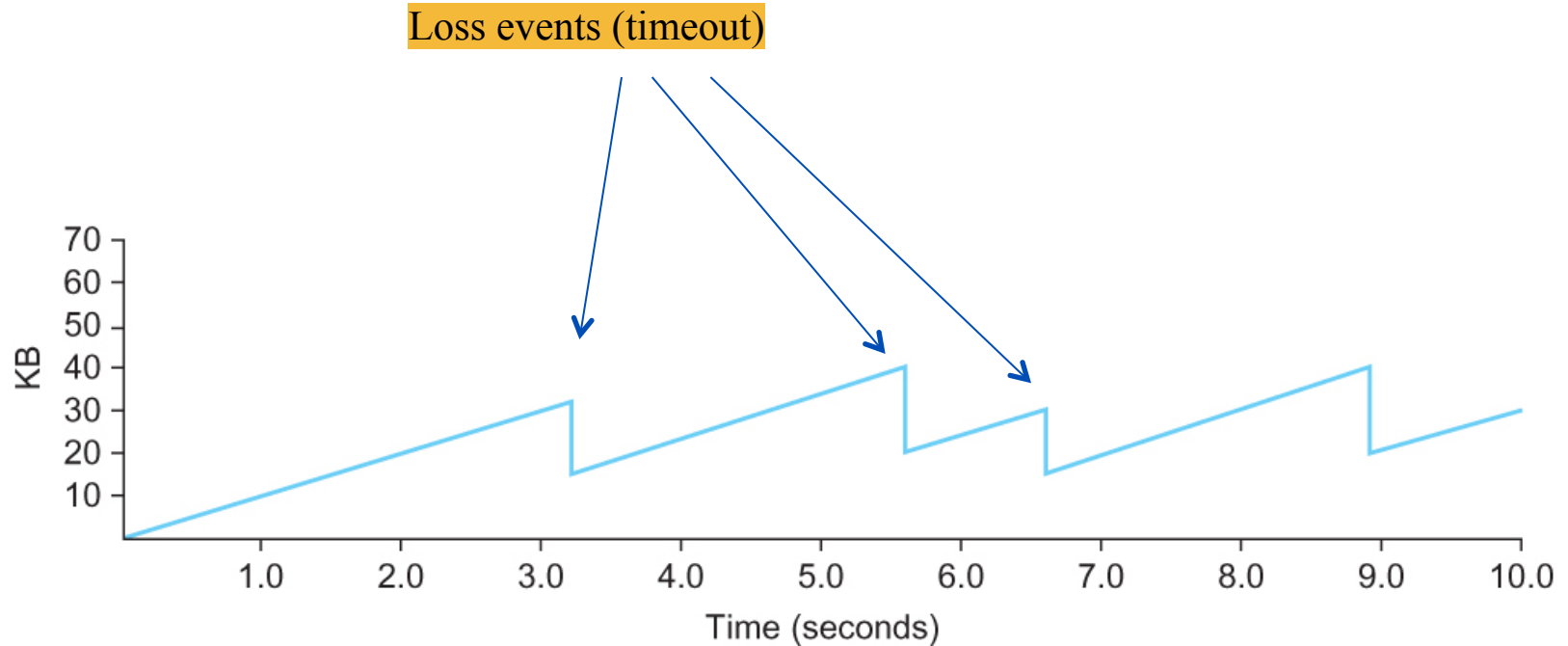
## ■ AIMD Algorithm

- Although CongestionWindow is defined in terms of bytes, it is easiest to understand multiplicative decrease if we think in terms of whole packets.
  - For example, suppose the CongestionWindow is currently set to 16 packets. If a loss is detected, CongestionWindow is set to 8.
  - Additional losses cause CongestionWindow to be reduced to 4, then 2, and finally to 1 packet.
  - CongestionWindow is not allowed to fall below the size of a single packet, or in TCP terminology, the *maximum segment size (MSS)*.

# TCP Congestion Control

- Additive Increase Multiplicative Decrease
  - TCP's effective window is revised as follows:
    - $\text{MaxWindow} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow})$
    - $\text{EffectiveWindow} = \text{MaxWindow}$
  - That is, MaxWindow replaces AdvertisedWindow.
  - Thus, a TCP source is allowed to send no faster than the slowest component—the network or the destination host—can accommodate.

# TCP Congestion Control



**FIGURE 6.9 Typical TCP sawtooth pattern.**

# TCP Congestion Control

- **AIMD**    **Additive Increase Multiplicative Decrease**
  - The additive increase is the right approach to use
    - When the source is operating close to the available capacity of the network.
  - But it takes too long to ramp up a connection when it is starting from scratch (a cold start).
    - The ideal case would be to determine the available network capacity quickly.

# TCP Congestion Control

- Slow Start
  - TCP therefore provides a second mechanism
    - Ironically called slow start, that is used to increase the congestion window rapidly from a cold start.
  - Slow start effectively increases the congestion window exponentially, rather than linearly.

# TCP Congestion Control

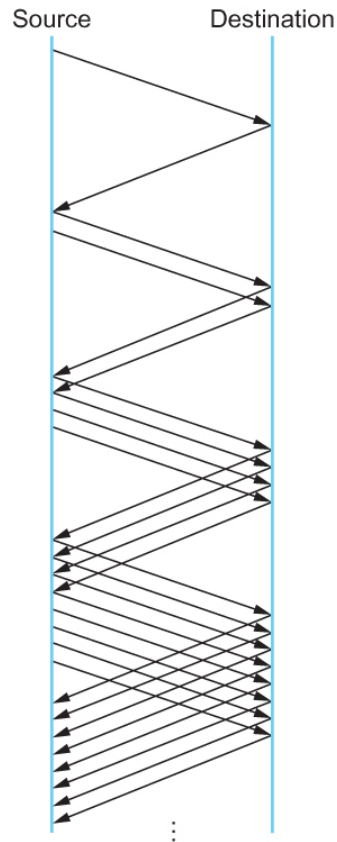
## ■ Slow Start

- Specifically, the **source starts** out by setting CongestionWindow to one packet.
  - $\text{cwnd} = 1$
- When the ACK for this packet arrives, TCP doubles the CongestionWindow and then sends two packets.
  - If ACK received then  $\text{cwnd} = \text{cwnd} \times 2 = 2$
- Upon receiving the corresponding two ACKs, **TCP** again **doubles the CongestionWindow** and next sends four packets.
  - If 2 ACKSs received then  $\text{cwnd} = \text{cwnd} \times 2 = 4$
- The end result is that TCP effectively doubles the number of packets it has in transit every RTT.



# TCP Congestion Control

- Slow Start



Packets in transit during slow start.

# TCP Congestion Control

## ■ Slow Start

- So how do we determine the capacity of the network as quickly as possible?
- First case:
  - At the very beginning of a connection the source has no idea how many packets it is going to be able to have in transit at a given time.
  - Slow start continues to double CongestionWindow each RTT until there is a loss, at which time a timeout causes ~~multiplicative decrease to divide CongestionWindow by 2.~~ congestionWindow to go down all the way to 1.

**slow start normal behavior: usually overshoot it till danger zone, detect loss, and come back to one regardless cold start or not.**

# TCP Congestion Control

## ■ Slow Start

- So how do we determine the capacity of the network quickly?
- **Second case** (a more delicate and precise method):
  - The source has a current (and useful) value of `CongestionWindow`; this is the value of `CongestionWindow` that existed prior to the last packet loss
  - The source halves this `CongestionWindow` that resulted in the last packet loss and call it the **`CongestionThreshold`**.
  - The source uses slow start again to rapidly increase the sending rate up to this threshold value, and then additive increase is used beyond this point.

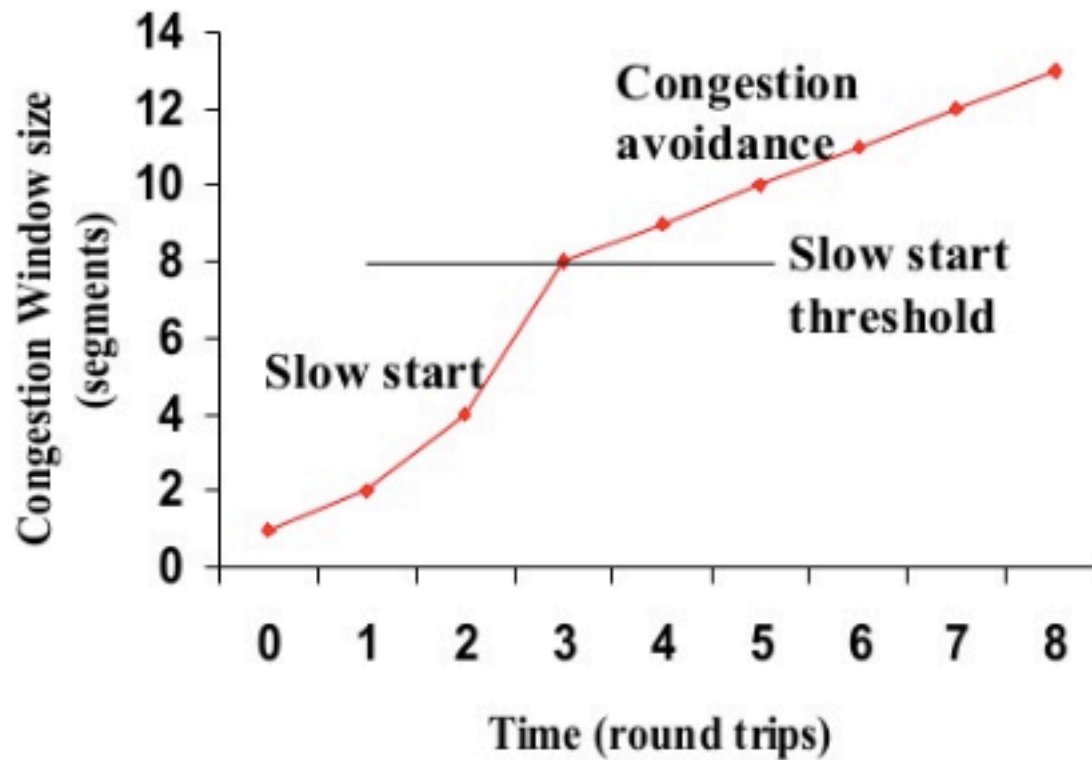
**`congestionThreshold` indicates start of the danger zone, congestion**

# TCP Congestion Control

- Elements of congestion control in TCP
  - Additive increase, multiplicative decrease (AIMD)
  - Slow start
  - Fast retransmit and fast recovery

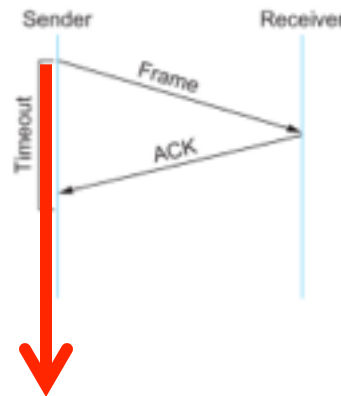
# TCP Congestion Control

## Typical TCP behaviour



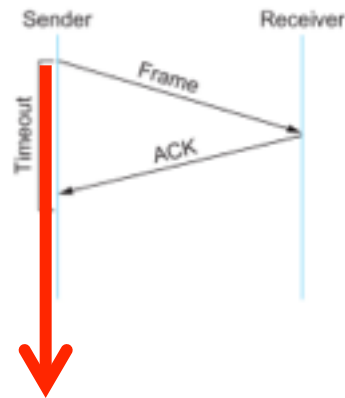
# TCP Congestion Control

- **Fast Retransmit** and Fast Recovery
  - The mechanisms described so far were part of the original proposal to add congestion control to TCP.
  - It was soon discovered, however, that using TCP timeouts to detect congestion led to long periods of time during which the connection went dead while waiting for a timer to expire.



# TCP Congestion Control

- Fast Retransmit and Fast Recovery
  - Because of this, a new mechanism called fast retransmit was added to TCP (TCP Reno)
  - Fast retransmit can trigger the retransmission of a dropped packet sooner than the regular timeout mechanism.



# TCP Congestion Control

- Fast Retransmit and Fast Recovery
  - The idea of fast retransmit is straightforward:
    - Every time a data packet arrives at the receiving side, the receiver responds with an acknowledgment, even if this sequence number has already been acknowledged.
  - Thus, when a packet arrives out of order, TCP resends the same acknowledgment it sent the last time.
    - Destination receives packet 5, acks packet 5
    - Destination receives packet 7, acks packet 5



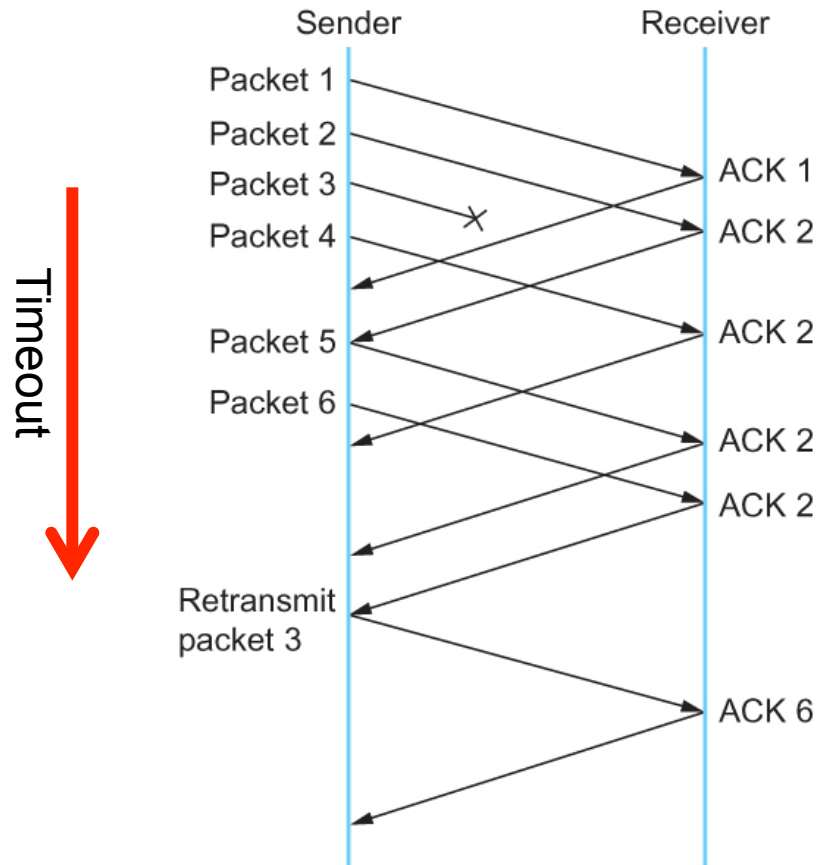
# TCP Congestion Control

- **Fast Retransmit and Fast Recovery**
  - To summarize:
    - Send an ACK on every packet
    - Send duplicate of last ACK when a packet is received out of order
    - Use duplicate ACKs to trigger retransmission instead of waiting for the longer timeout => Fast Retransmit

# TCP Congestion Control

- **Fast Retransmit and Fast Recovery**
  - Since it is also possible that the earlier packet has only been delayed rather than lost, the sender waits until it sees some number of duplicate ACKs and then retransmits the missing packet.
  - In practice, TCP waits until it has seen three duplicate ACKs before retransmitting the packet.
    - Even with three ACKs still faster retransmit than waiting for the timeout to occur

# TCP Congestion Control



**FIGURE 6.12 Fast retransmit based on duplicate ACKs.**

# TCP Congestion Control

- Fast Retransmit and **Fast Recovery**
  - When the fast retransmit mechanism signals congestion, rather than drop the congestion window all the way back to one packet and run slow start, it is possible to use the ACKs that are still in the pipe to clock the sending of packets.
    - After all, receiving of all those duplicate ACKs indicate that packets are reaching the other end except for that one missing
    - So sending more packets will not result in increased transit packets, but do this using the multiplicative decrease method
  - This mechanism (***fast recovery***), *effectively* removes the slow start phase and goes directly to the multiplicative decrease phase ( $cwnd = cwnd/2$ )

# TCP Congestion Avoidance

- Two questions:
  1. How can we detect congestion?
  2. How to make flow rate sensitive to congestion level?
- TCP congestion control was introduced into the Internet in the late 1980s by Van Jacobson
  - TCP Tahoe (AIMD & Slow Start)
  - TCP Reno (Fast Retransmit and Fast Recovery)
  - TCP Vegas (Congestion Avoidance)
- The main reason we can use the Internet successfully today

# TCP Congestion Avoidance

**TCP Vegas** is a **TCP** congestion avoidance algorithm that emphasizes packet delay, rather than packet loss, as a signal to help determine the rate at which to send packets. It was developed at the University of Arizona by Lawrence Brakmo and Larry L. Peterson.

[TCP Vegas - Wikipedia, the free encyclopedia](https://en.wikipedia.org/wiki/TCP_Vegas)  
[en.wikipedia.org/wiki/TCP\\_Vegas](https://en.wikipedia.org/wiki/TCP_Vegas) Wikipedia ▾



More about TCP Vegas

*Feedback*

University of Arizona researchers Larry Peterson and Lawrence Brakmo introduced TCP Vegas, named after the largest Nevada city, in which timeouts were set and round-trip delays were measured for every packet in the transmit buffer.

# TCP Congestion Avoidance

## TCP Vegas: New Techniques for Congestion Detection and Avoidance

Lawrence S. Brakmo  
Sean W. O'Malley  
Larry L. Peterson<sup>1</sup>

TR 94 04

### Abstract

Vegas is a new implementation of TCP that achieves between 40 and 70% better throughput, with one-half to one-fifth the losses, as compared to the implementation of TCP in the Reno distribution of BSD Unix. This paper motivates and describes the five key techniques employed by Vegas, and presents the results of a comprehensive experimental performance study—using both simulations and measurements of the actual Internet—of the Vegas and Reno implementations of TCP.

February 16, 1994

# Congestion Avoidance Mechanism

- TCP's strategy in Reno is to control congestion once it happens
- TCP repeatedly increases the to find the point at which congestion occurs, and then it backs off from this point.
- An appealing alternative is to predict when congestion is about to happen and reduce the rate
- We call such a strategy **congestion avoidance**, to distinguish it from congestion control.



# Congestion Avoidance Mechanism

- So the question is how to predict congestion in order to avoid it.
- Two possibilities:
  1. **Router-centric**
    - Routers explicitly notify sources about congestions
      - DECbit      Digital Equipment Corporation
    - Routers implicitly notify sources about congestions
      - RED Gateways      Random Early detection
  2. **Host-centric (source-based)**

TCP uses Host-Centric Feedback based instead of Window based instead of rate based

check 2 RTT > max & min of average seen so far

# Congestion Avoidance Mechanism

- DEC Bit
  - The first mechanism was developed for use on the Digital Network Architecture (DNA) at Digital Equipment Corporation (DEC)
    - A connectionless network with a connection-oriented transport protocol.
  - This mechanism could, therefore, also be applied to TCP and IP.
  - The idea here is to more evenly split the responsibility for congestion control between the routers and the end nodes.

# Congestion Avoidance Mechanism

## ■ DEC Bit

- Each router monitors the load it is experiencing and explicitly notifies the end nodes when congestion is about to occur.
- This notification is implemented by setting a binary congestion bit (DEC bit) in the packets that flow through the router.
- The destination host then copies this congestion bit into the ACK it sends back to the source.
- Finally, the source adjusts its sending rate so as to avoid congestion.

# Congestion Avoidance Mechanism

## ■ DEC Bit

When both endpoints support ECN, they mark their packets with 01 or 10. If the packet traverse a router that is experiencing congestion (and the router supports ECN) then it changes a code to 11 indicating congestion to the end point.

IPv4 Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP				ECN		Total Length																	
4	32	Identification																Flags				Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															

ECN is an optional feature that may be used between two ECN-enabled endpoints when the underlying network infrastructure (routers) also supports it.

# Congestion Avoidance Mechanism

- DEC Bit Router centric and explicit **vs RED (implicit)**
  - If **less than 50%** of the packets had the bit set, then the source increases its congestion window by one packet (increased linearly).
  - If **50% or more** of the last window's worth of packets had the congestion bit set, then the source decreases its congestion window to 0.875 times the previous value (decreased exponentially).
  - The *"increase by 1, decrease by 0.875"* rule was selected because additive increase/multiplicative decrease makes the mechanism stable.

# Congestion Avoidance Mechanism

- Random Early Detection (RED)
  - Similar to the DEC bit scheme in that each router is programmed to monitor its own queue length
  - When it detects that congestion is imminent, notifies the source to adjust its congestion window.
  - RED, invented by Sally Floyd and Van Jacobson in the early 1990s, differs from the DEC bit scheme in two major ways:

# Congestion Avoidance Mechanism

- Random Early Detection (RED)
  1. RED *implicitly notifies* the source of congestion *by dropping* one of its packets.
    - The source is, therefore, effectively notified by the subsequent timeout or duplicate ACK.
    - RED is designed to be used in conjunction with TCP, which currently detects congestion by means of timeouts or duplicate ACKs.

# Congestion Avoidance Mechanism

Router Centric too. Router Centric avoidance

- Random Early Detection (RED)
  - It drops the packet earlier than it would have to, so as to notify the source that it should decrease its congestion window sooner than it would normally have.
  - The router drops a few packets before it has exhausted its buffer space completely
    - a. So as to cause the **source to slow down.**
    - b. Hoping that this will mean it does not have to drop lots of packets later on.



# Congestion Avoidance Mechanism

- Source-based Congestion Avoidance
  - The general idea of these techniques is to watch for some sign from the network that some router's queue is building up and that congestion will happen soon if nothing is done about it.
  - For example, the source might notice that as packet queues build up in the network's routers, there is a measurable increase in the RTT for each successive packet it sends.

# Congestion Avoidance Mechanism

- Source-based Congestion Avoidance
  - One particular algorithm exploits this observation as follows:
    - The congestion window normally increases as in TCP, but every two round-trip delays the algorithm checks to see if the current RTT is greater than the average of the minimum and maximum RTTs seen so far. **continue if greater**
    - If it is, then the algorithm decreases the congestion window by one-eighth.

# Summary

- Different resource allocation mechanisms
- Different queuing mechanisms
- TCP Congestion Control mechanisms
- Congestion Avoidance mechanisms