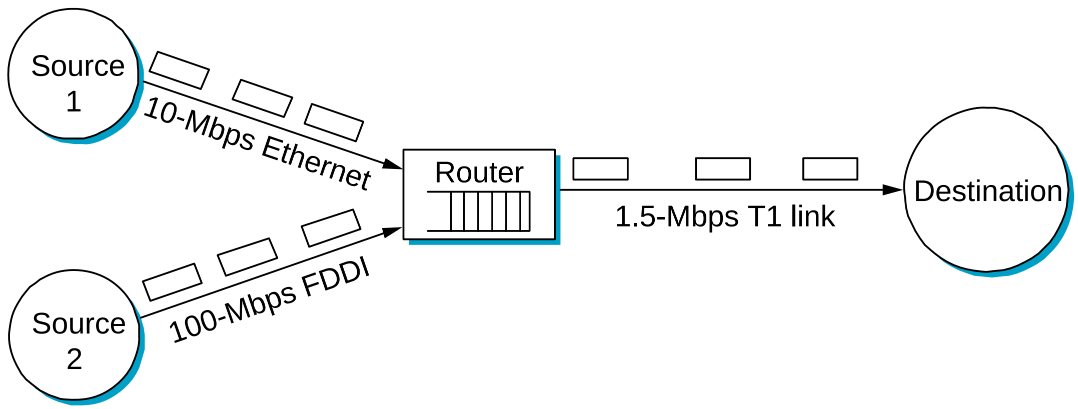
Lecture 14 : Congestion Control

Shankar Balachandran
Assistant Professor
Dept. of CSE, IIT Madras

Short Term Course on “Teaching Computer Networks Effectively”. Sponsored by AICTE.

Congestion Issues

- Two sides of the same coin
  - pre-allocate resources so at to avoid congestion
  - control congestion if (and when) it occurs



- Two points of implementation
  - hosts at the edges of the network (transport protocol)
  - routers inside the network (queuing discipline)
- Underlying service model
  - best-effort (assume for now)
  - multiple *qualities of service* (later)

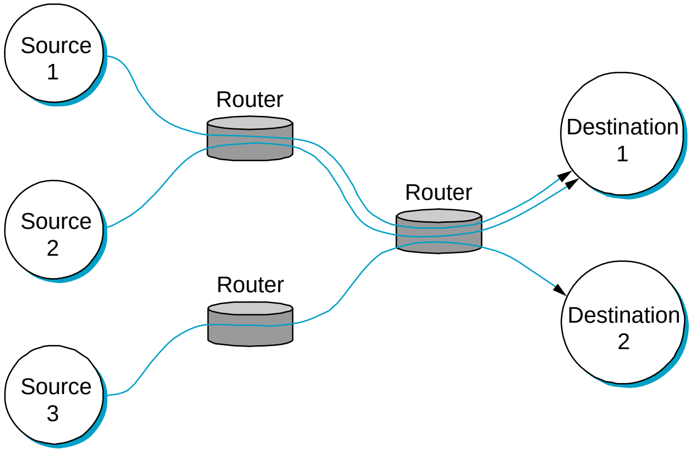
Congestion

- Congestion vs Access Control
  - Congestion is different from Access Control (MAC)
  - You can observe traffic and choose not to send data in local networks
- Can we always route around congested links?
  - Second order congestion possible
  - Oscillation
  - Lack of alternate routing paths

Framework

- Connectionless flows
  - sequence of packets sent between source/destination pair
  - maintain *soft state* at the routers
    - unlike connection oriented service, resources are not blocked
- Issues
  - Why analyze in a connectionless setup?
  - If datagrams are independent of each other, what's the point?

- Abstraction
  - Datagrams follow a *flow*
  - A sequence of packets follow the same route
- Flow
  - Can be used at various levels of granularity



Fairness and Efficiency

- Two running themes in congestion control
- Can we allocate efficiently?
  - Where do we allocate?
  - What resources do we allocate?
  - When can you do it ?
- Can we allocate resources fairly?
  - What is fairness?
  - Who should you be fair to?
  - How can you adapt to situations?

1. Service Model

- Best effort
  - Every one is a first class citizen
  - No specific treatment for any packet or connection
- QoS
  - Specific flows may need preferential action
- Applications need different guarantees
- Very challenging to come up with a unified model

2. Router centric vs Host Centric

- Address issues from inside the network
  - Routers, switches
  - Routers take responsibility for which to forward or drop
  - Inform hosts about the packets that are allowed
- Address from outside the network
  - In the hosts, inside the transport protocol
  - Observe the connections from outside
- Not completely mutually exclusive
  - Even if routers control, hosts must handle advisory messages
  - Even if hosts handle, routers must have at least simple policies

3. Reservation vs Feedback

- Reservation :
  - End hosts ask the network for a certain amount of capacity
  - Routers allocate
    - If not possible, routers can reject the flows
- Feedback :
  - Send data and observe what happens
  - Use this to control
- Reservation system goes with router-centric approach
- Feedback can go with either one

4. Window Based or Rate Based

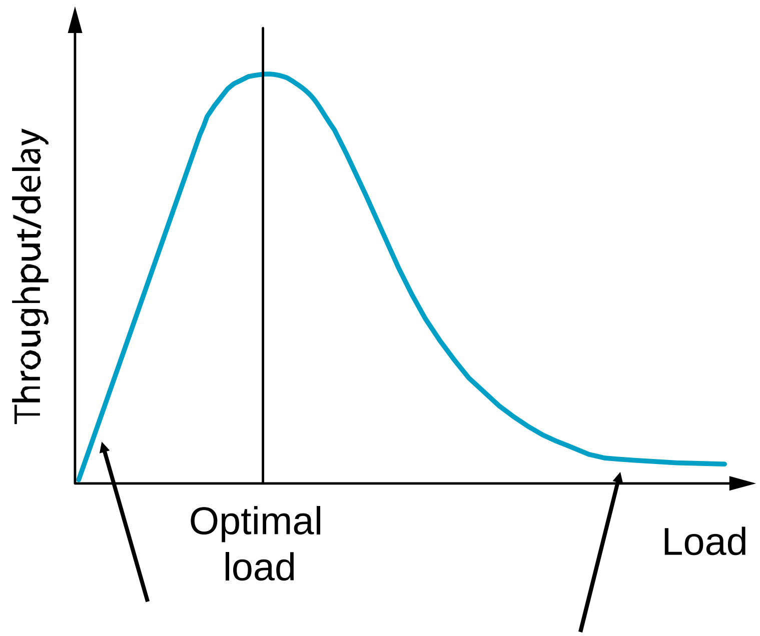
- Window
  - Like in TCP's *Advertised_Window*
  - Sender and receiver negotiate using buffer sizes as a metric
    - Even buffer size is abstracted away
- Rate
  - Bits/second
  - Logical in a reservation based system with QoS
    - Routers along the way can determine if the flow can be accepted with other commitments in mind.
    - E.g. Video

Two General Strategies

- Best effort + Feedback
  - Users must not be able to reserve
  - Congestion control is done at the hosts
  - Usually window-based
  - Model of the internet; Will see more
- QoS + Reservation
  - Significant router involvement
  - Explicit rates
  - Will see more too

Evaluation

- Fairness
- Throughput and Delay
- Power (ratio of throughput to delay)



More overhead; Very conservative

Longer Queues; Very aggressive

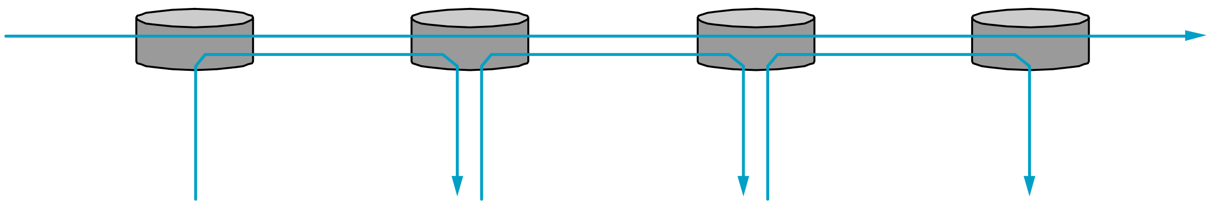
1. Power Ratio

- Not the best metric
- Depends on M/M/1* queuing model
  - Infinite queues
  - Not true in real world and hence drops packets
- Power is with respect to a single flow
  - Not quite clear what power is for multiple flows
- Very popular metric though; No competing metrics

*** - Exponential Arrival, Exponential Service Time, 1 Server**

2. Fairness

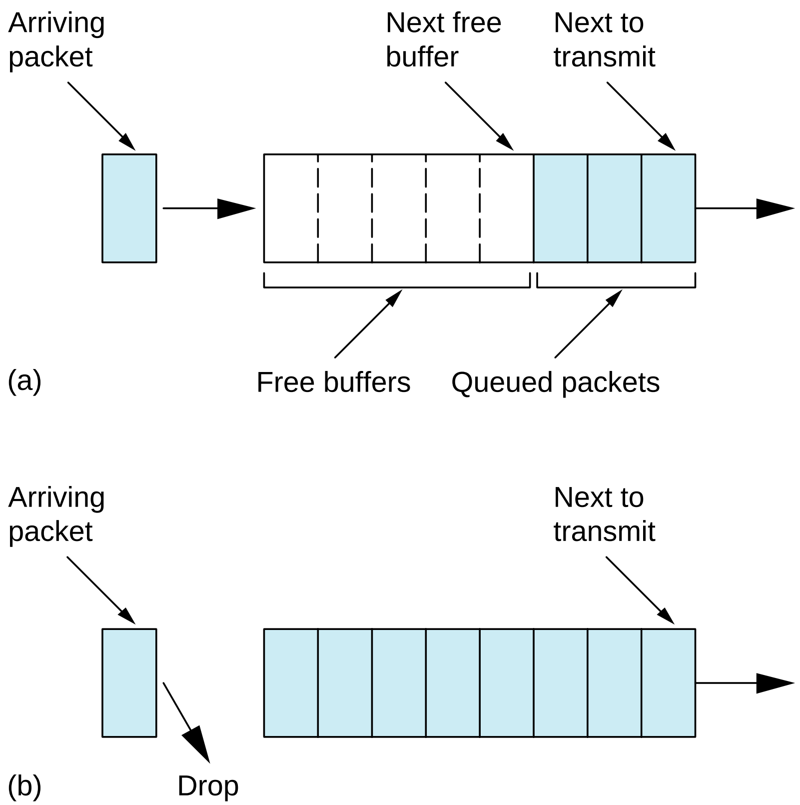
- What is fair?
  - Equal bandwidth?
    - Files don't require as much as videos
  - Equal share may not mean fairness
    - What about pathlength?



Jain's Fairness Index

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

Queuing Discipline (1)



Scheduling Discipline: FIFO

Drop Policy: tail drop

This is the most widely used queueing discipline in the Internet. It pushes the responsibility for resource allocation and congestion control to the edges of the network.

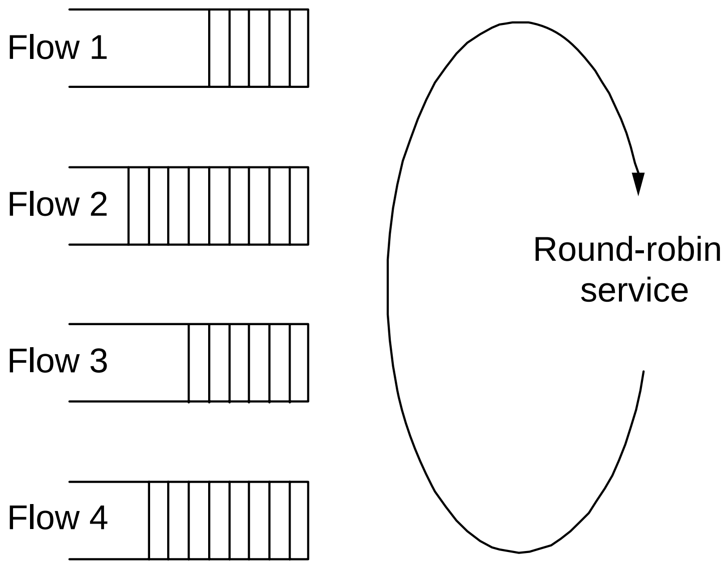
TCP detects and responds to congestion without much help from routing.

Schedule and Drop policies are different ideas.

Possible Variation : Priority Based

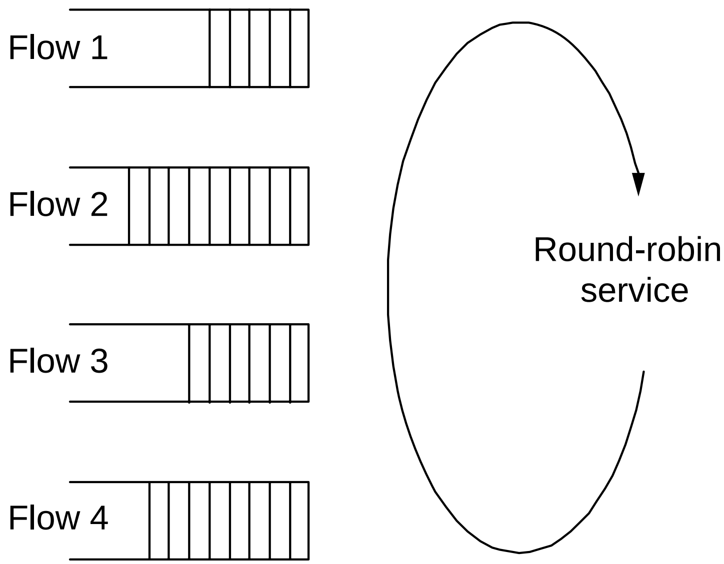
Queuing Discipline (2)

- Fair Queuing (FQ)
  - explicitly segregates traffic based on flows
  - ensures no flow captures more than its share of capacity
  - variation: weighted fair queuing (WFQ)
- Problems?



- Packets need not be of same length; One flow can hog resources
- What we need is a bit by bit mechanism.
  - Transmit one bit from each flow
  - Overhead?

FQ Algorithm : 1. For a Single Flow



- A flow is active when there's a packet in its queue.

$$S_i = \max(F_{i-1}, A_i)$$
$$F_i = S_i + P_i$$
$$F_i = \max(F_{i-1}, A_i) + P_i$$

$S_i =$ **TX start for packet *i***

$F_i =$ **TX end for packet *i***

$P_i =$ **Clock ticks to send *i***

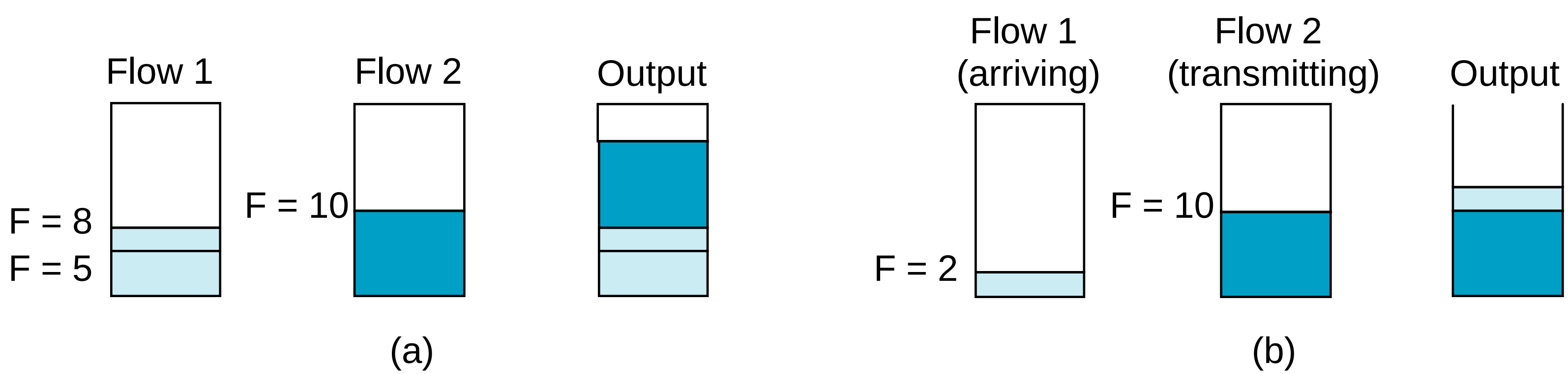
$A_i =$ **Arrival time for Packet *i***

- How do we determine arrival times? The clock we define advances by one tick when one bit from each active flow is transmitted.

- Compute all F_i: the next packet to transmit has the lowest F_i.

FQ Algorithm : 2. Multiple Flows

- For multiple flows
  - calculate F_i for each packet that arrives on each flow
  - treat all F_i's as timestamps
  - next packet to transmit is one with lowest timestamp
- Not perfect: can't preempt current packet
- Example



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

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)
```

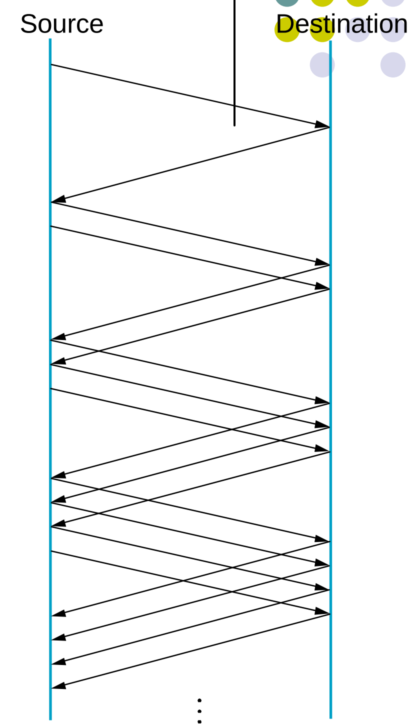
- Idea:
  - increase **`CongestionWindow`** when congestion goes down
  - decrease **`CongestionWindow`** when congestion goes up

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*)

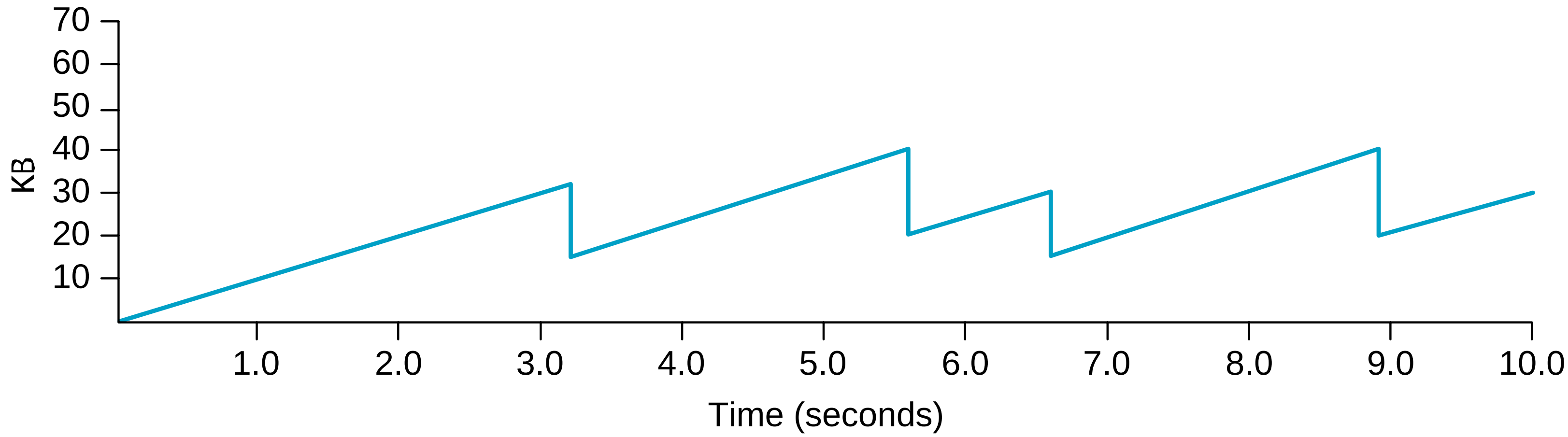


- In practice: increment a little for each ACK

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

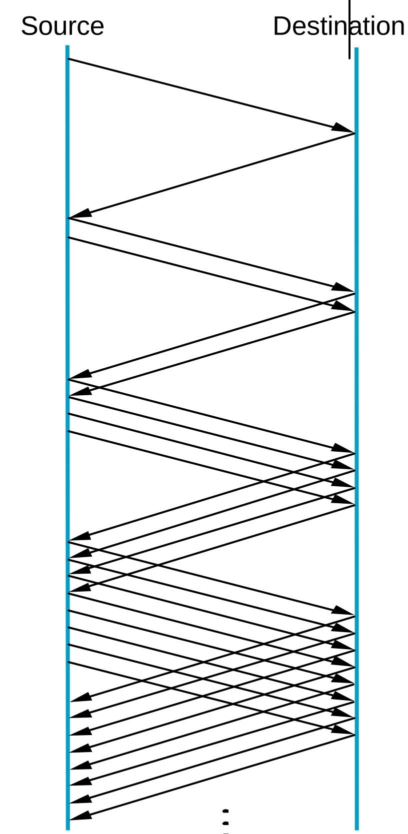
AIMD (cont)

- Trace: sawtooth behavior



Slow Start

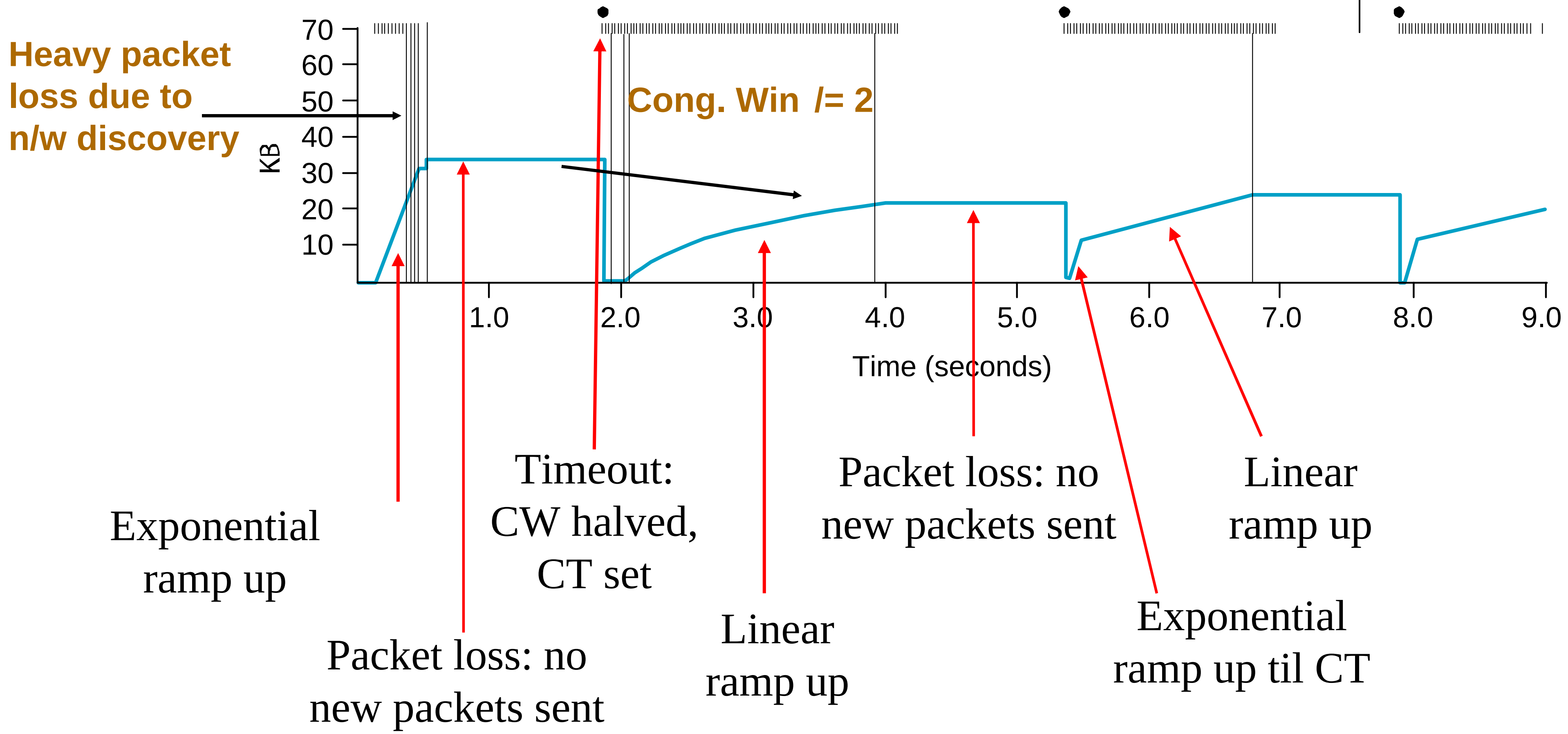
- Problem : Ramp up takes a lot of time in the beginning
- Objective: determine the available capacity first
- Idea:
  - begin with **`CongestionWindow`** = 1 packet
  - double **`CongestionWindow`** each RTT (increment by 1 packet for each ACK)



Slow Start (cont)

- Exponential growth, but slower than all at once
  - That's why it is called *Slow Start*
- Used…
  - when first starting connection
  - when connection goes dead waiting for timeout

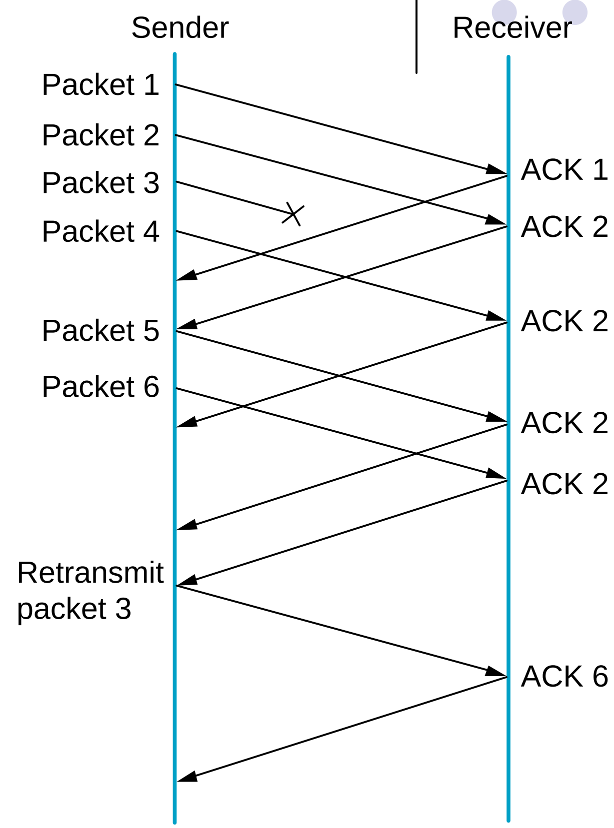
Trace



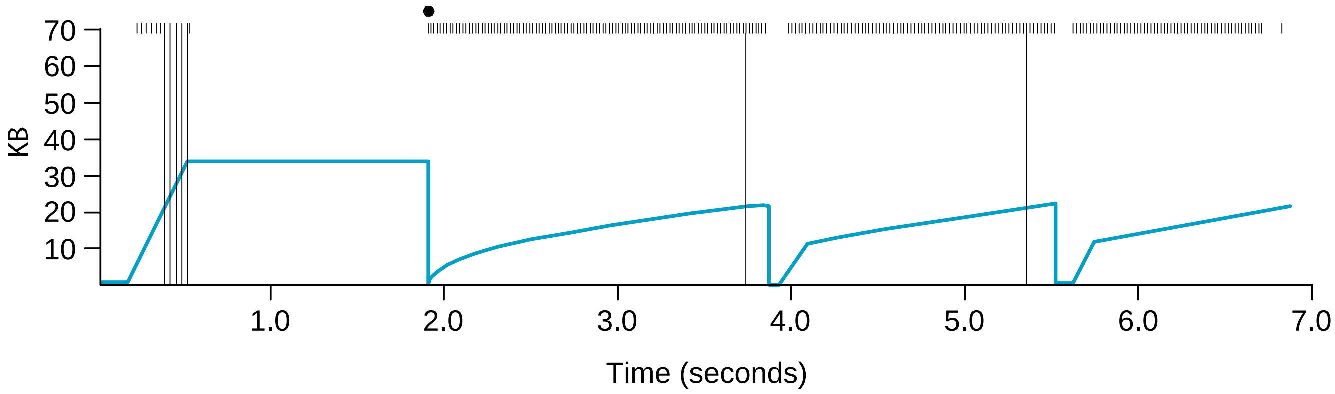
- Problem: lose up to half a **CongestionWindow**'s worth of data

Fast Retransmit and Fast Recovery

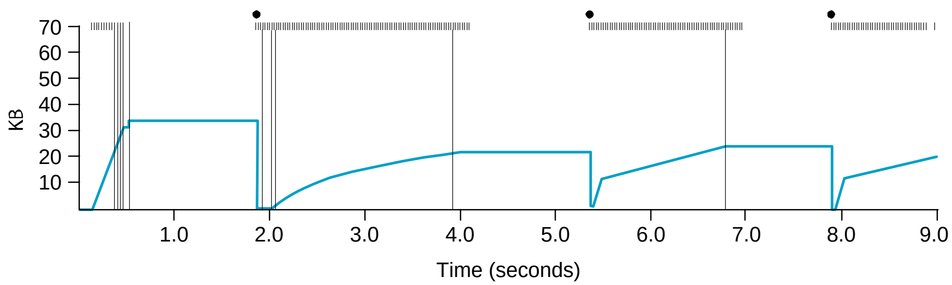
- Problem: coarse-grain TCP timeouts lead to idle periods
- Fast retransmit: use duplicate ACKs to trigger retransmission
  - Receiver gets out of order
  - Sends ACK for previous one received in correct order
  - Sender sees a duplicate ACK
    - Implies packet loss
  - Delayed or lost?
    - Wait for a few more duplicates; Then retransmit



Results



Compare this to Slow Start :



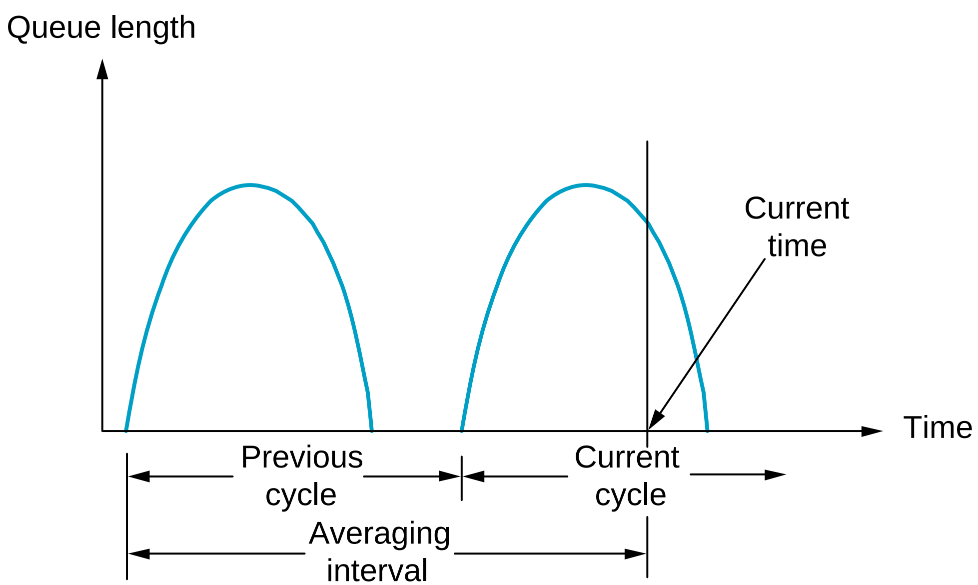
- Fast recovery
  - skip the slow start phase
  - go directly to half the last successful **`CongestionWindow (ssthresh)`**
  - **do not drop all the way to 1**

Congestion Avoidance

- TCP's strategy
  - control congestion once it happens
  - repeatedly increase load in an effort to find the point at which congestion occurs, and then back off
- Alternative strategy
  - predict when congestion is about to happen
  - reduce rate before packets start being discarded
  - call this congestion *avoidance*, instead of congestion *control*
- Two possibilities
  - router-centric: DECbit and RED Gateways
  - host-centric: TCP Vegas

DECbit

- Add binary congestion bit to each packet header
- Router
  - monitors average queue length over last busy+idle cycle



  - set congestion bit if average queue length > 1
  - attempts to balance throughput (queuing) against delay (idle time)

End Hosts

- Destination echoes bit back to source
- Source records how many packets resulted in set bit
- If less than 50% of last window's worth had bit set
  - increase **`CongestionWindow`** by 1 packet
- If 50% or more of last window's worth had bit set
  - decrease **`CongestionWindow`** by 0.875 times
- AIMD

Random Early Detection (RED)

- Notification is implicit
  - just drop the packet (done in RED)
    - TCP will timeout; Host will know
  - could make explicit by marking the packet
    - done in DECbit
- Early
  - Do it before congestion happens
- Random drop
  - rather than wait for queue to become full, drop each arriving packet with some *drop probability* whenever the queue length exceeds some *drop level*

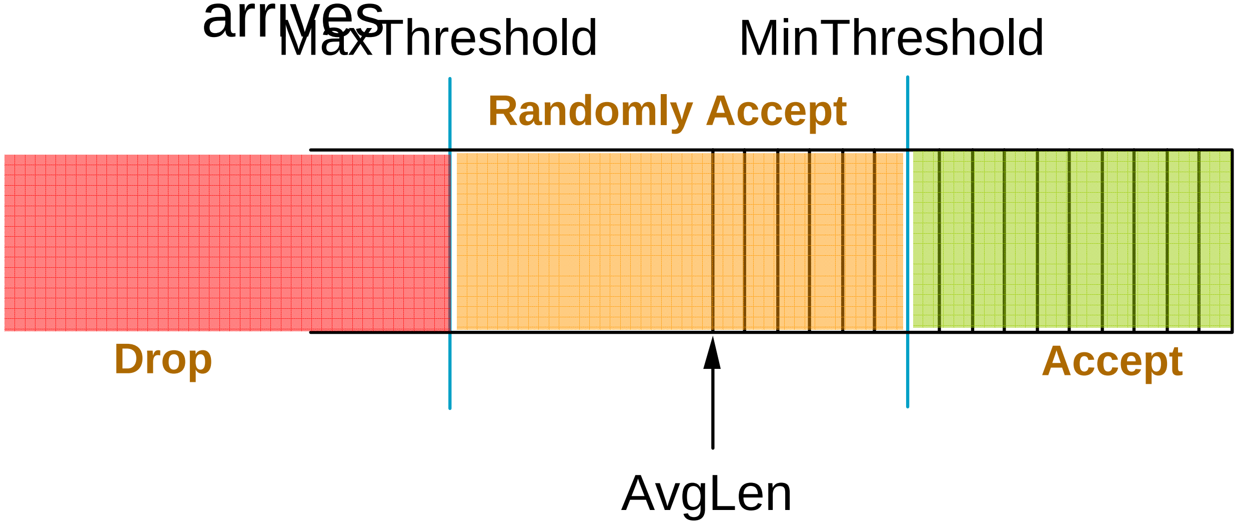
RED Details

- Compute average queue length

  `AvgLen = (1 - Weight) * AvgLen + Weight * SampleLen`

  0 < **`Weight`** < 1 (usually 0.002)

  `SampleLen` is queue length each time a packet arrives



RED Details (cont)

- Two queue length thresholds

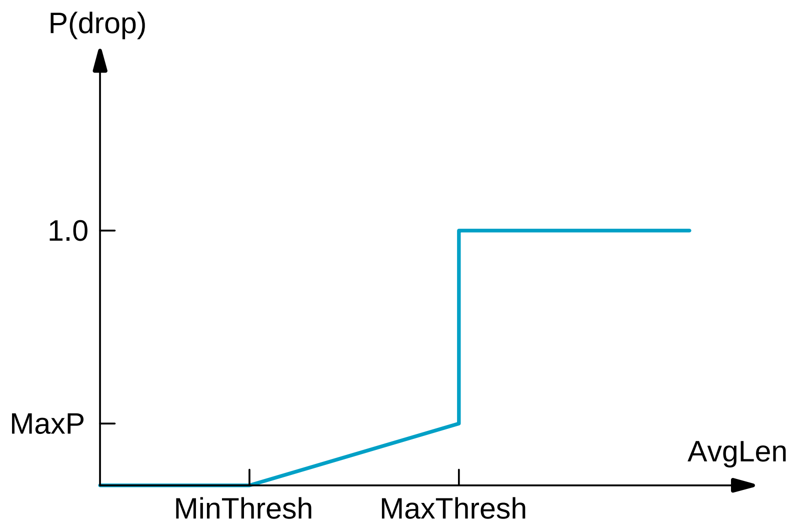
```
if AvgLen <= MinThreshold then
    enqueue the packet
if MinThreshold < AvgLen < MaxThreshold then
    calculate probability P
    drop arriving packet with probability P
if ManThreshold <= AvgLen then
    drop arriving packet
```

RED Details (cont)

- Computing probability P

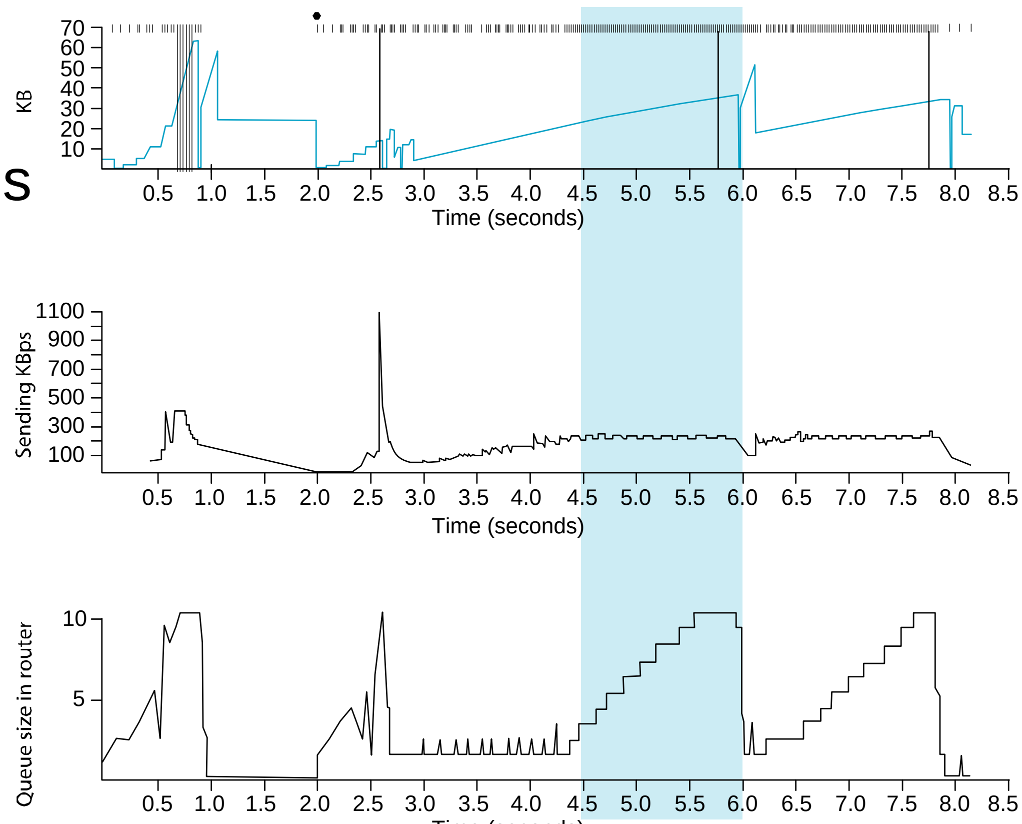
```
TempP = MaxP * (AvgLen - MinThreshold)/
                (MaxThreshold -
 MinThreshold)
P = TempP/(1 - count * TempP)
```

- Drop Probability Curve



TCP Vegas

- Idea: source watches for some sign that router's queue is building up and congestion will happen too; e.g.,
  - RTT grows
  - sending rate flattens



Algorithm

- Let **`BaseRTT`** be the minimum of all measured RTTs (commonly the RTT of the first packet)
- If not overflowing the connection, then

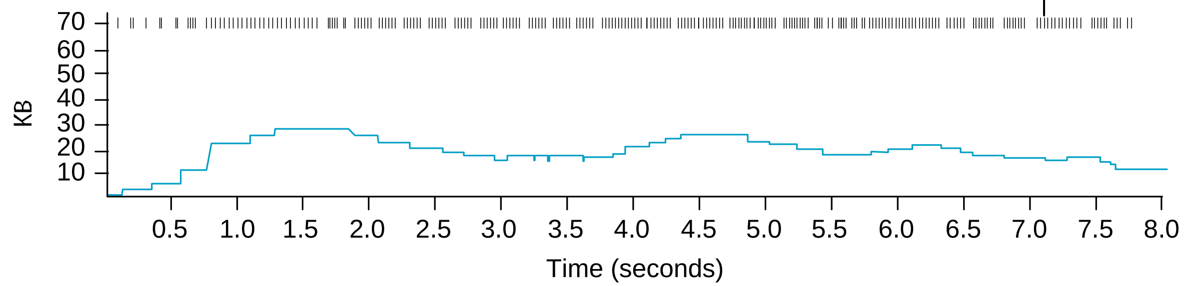
```
    ExpectRate = CongestionWindow/BaseRTT
```

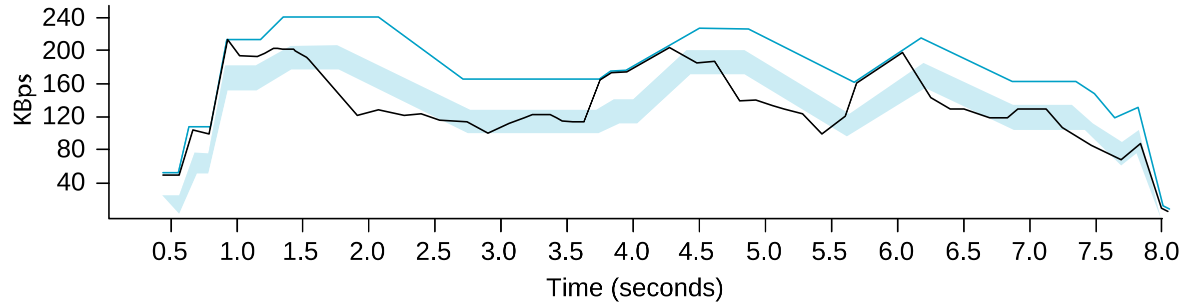
- Source calculates sending rate (**`ActualRate`**) once per RTT
- Source compares **`ActualRate`** with **`ExpectRate`**

```
Diff = ExpectedRate - ActualRate
if Diff < α
    increase CongestionWindow linearly
else if Diff > β
    decrease CongestionWindow linearly
else
    leave CongestionWindow unchanged
```

Algorithm (cont)

- Parameters
  - α = 1 packet
  - β = 3 packets





- Even faster retransmit
  - keep fine-grained timestamps for each packet
  - check for timeout on first duplicate ACK

Summary

- Congestion control
- Host vs router centric, best effort vs QoS, reservation vs feedback
- Congestion Window
- Congestion Avoidance