Chapter 6
Congestion Control and Resource Allocation

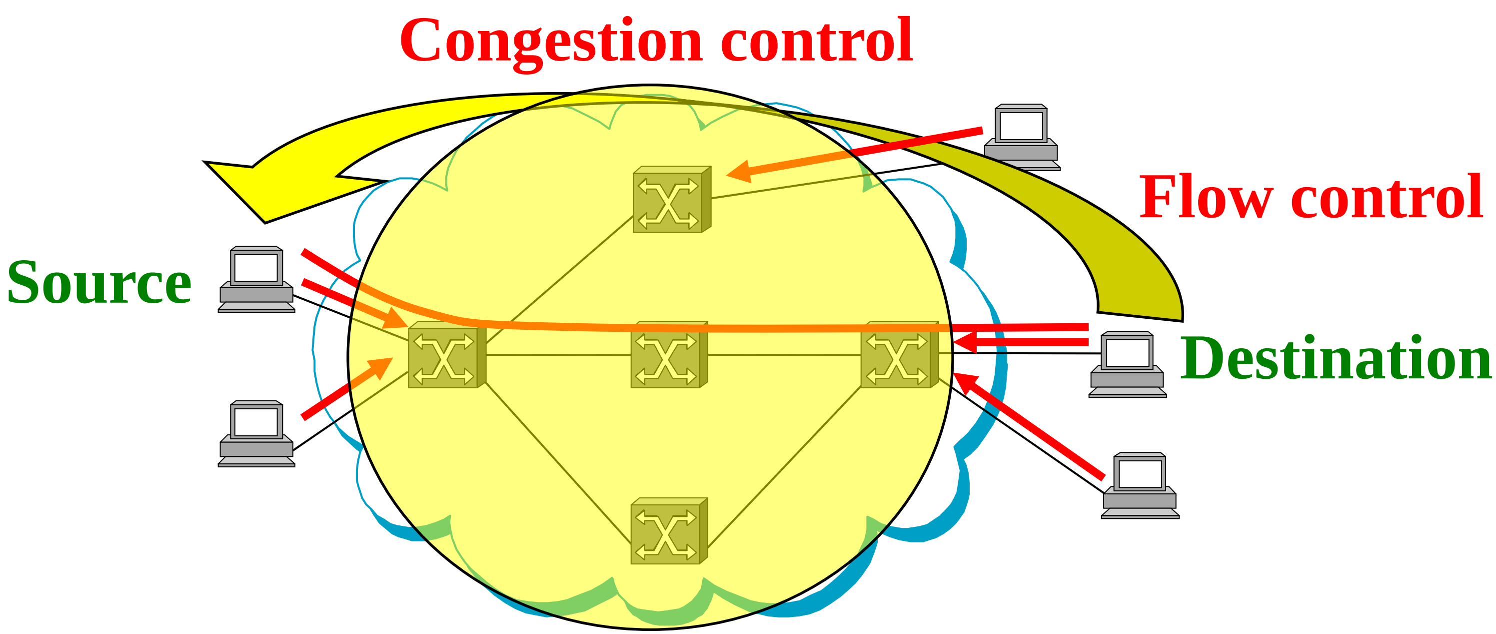
Congestion Control and Resource Allocation

- The issue is “how to **effectively** and **fairly** allocate resources among a collection of competing users?”
- The resources being shared include
  - The **bandwidth** of the links
  - The **buffers** on the routers or switches
- **Congested:** when too many packets are contending for the same link, the **queue overflows** and packets have to be **dropped frequently**
  - A **congestion-control mechanism** is introduced
- Congestion may be avoided by using good **resource allocation mechanism**
  - Possibly make congestion-control unnecessary

Issues in Resource Allocation

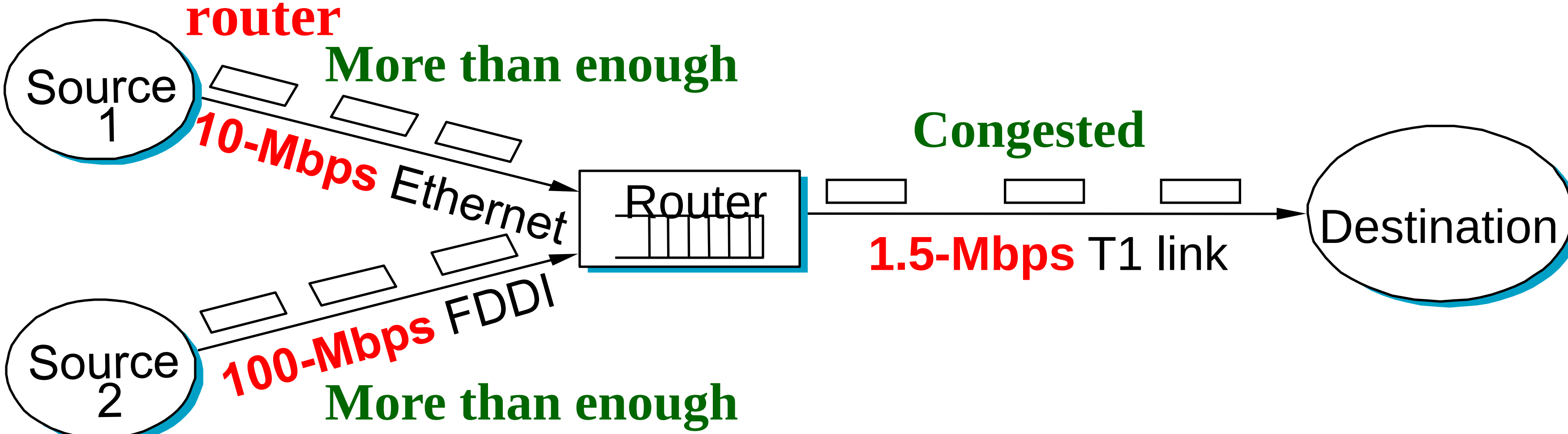
Issues in Resource Allocation

- The difference between **flow control** and **congestion control**
  - Flow control involves keeping **a fast sender** from overrunning a slow **receiver**
  - Congestion control is intended to keep **a set of senders** from sending too much data into **the network**



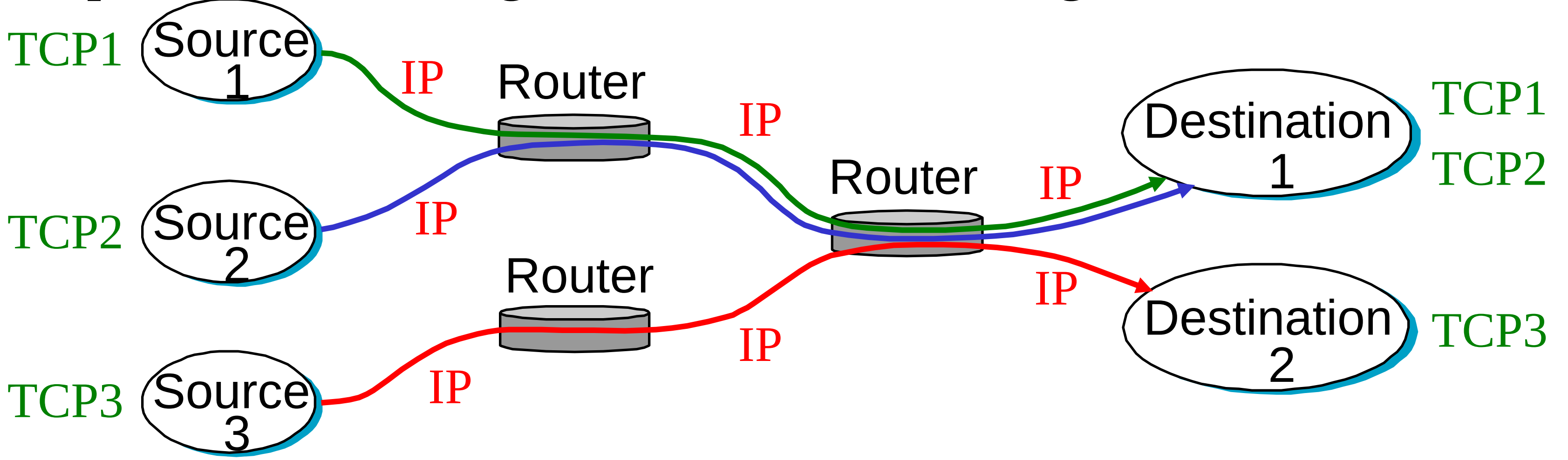
Network Model

- A packet-switched network consists of multiple links and routers (or switches)
- A given source may have **more than enough capacity** on the the **immediate** outgoing link to send a packet
- In the middle of a network, the packets encounter a link that is being used by many **different traffic sources**
  - The congested router is sometimes call the **bottleneck router**



Connectionless Flows

- For the Internet, the network is essentially **connectionless**, with any **connection-oriented** service implemented in the transport protocol that is running on the end hosts
  - **IP** provides a **connectionless** datagram delivery service
  - **TCP** implements an end-to-end **connection abstraction**
- **Flow:** a sequence of packets sent between a **source/destination** pair and following **the same route** through the network



Connectionless Flows

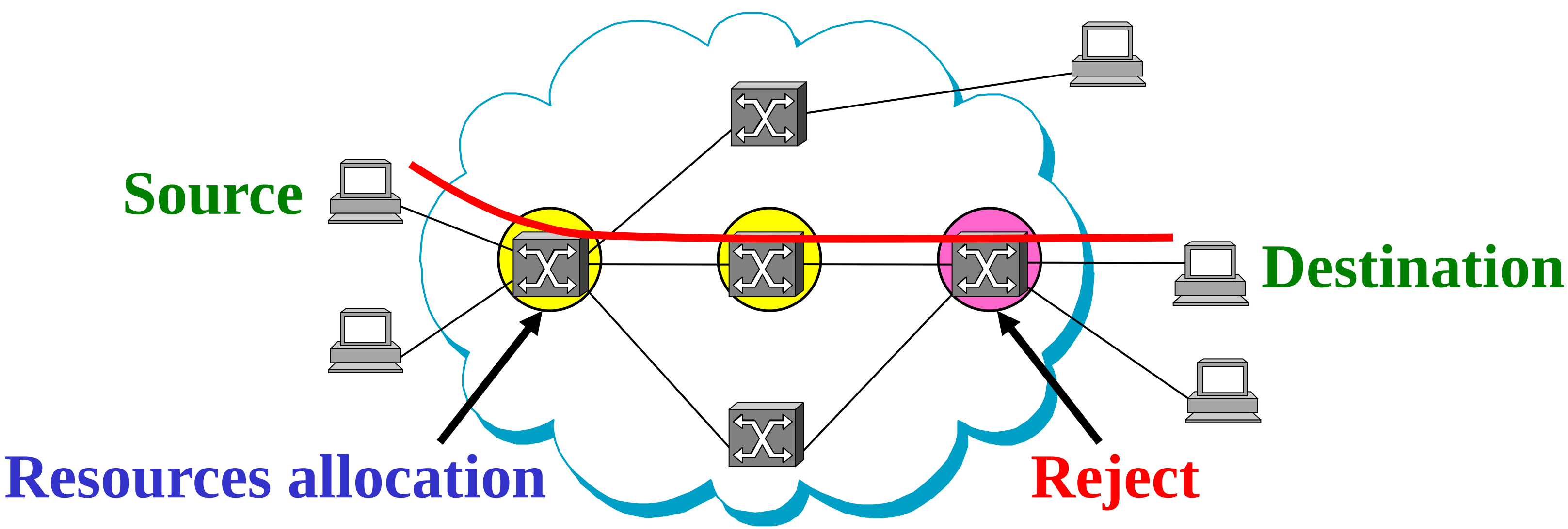
- Each router maintains **soft state and hard state information** for each flow
- **Soft state** is **not** explicitly created and removed by **signaling**
  - That can be used to make **resource allocation** decisions about the packets that belong to the flow
- **Hard state** is explicitly created and removed by **signaling**
  - That is generally used to make the packets being **correctly routed** from the source to the destination
- The correct operation of the network does not depend on soft state information, but the router with this information can **better handle the packets**
  - Resource allocation and congestion control

Router-Centric Versus Host-Centric

- The **Router-Centric** mechanism addresses the problem from **inside** the network
  - Each **router** takes responsibility for deciding **when** packets are forwarded and selecting **which** packets are dropped
- The **Host-Centric** mechanism addresses the problem from the **edges** of the network
  - The **end hosts** observe the network conditions and adjust their behavior accordingly
- These two groups are **not mutually exclusive**
  - A Router-Centric network still expects the end hosts to adhere to any **advisory messages** **sent by the routers**
  - A Host-Centric network still has some policy for deciding **which packets to drop** when their queues do overflow

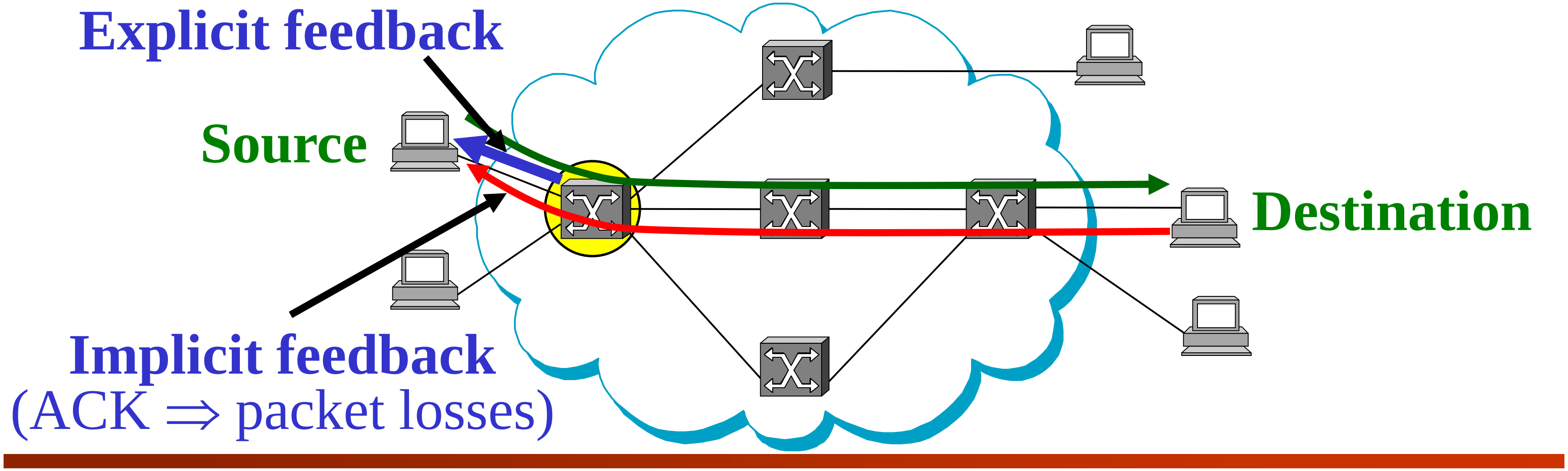
Reservation-Based

- In a **Reservation-Based** system, the end host asks for a certain amount of capacity at the time a flow is established
  - Each router allocates **enough resources** to satisfy the request
  - If the request cannot be satisfied at some router, the router **rejects** the flow



Feedback-Based

- In a **Feedback-Based** system, the end hosts begin sending data **without reserving any capacity** and **adjust their sending rate** according to the received feedback
  - **Explicit feedback:** it is according to a message from a congested router
  - **Implicit feedback:** it is according to the **externally observable behavior** of the network (e.g. **packet losses**)



Reservation-Based Versus Feedback-Based

- A **Reservation-Based** system always implies **a router-centric** resource allocation mechanism
  – Each router is responsible for keeping track the resource
- A **Feedback-Based** system can imply **either a router- or host-centric** mechanism
  – If the feedback is **explicit**, the router is involved
  – If the feedback is **implicit**, the routers **silently drop packets** when they become congested

Window-Based Versus Rate-Based

- Both **flow-control** and **resource allocation** mechanisms need a way to express to the sender
- For **Window-Based** mechanism (such as TCP), the receiver advertises a window to the sender
  - This window corresponds to how much **buffer space** the receiver has
- For **Rate-Based** mechanism, a sender's behavior is controlled by using a **rate**
- **Rate-based** characterization of flows is a good choice in a reservation-based system
  - Supports different qualities of service **(QoS)**

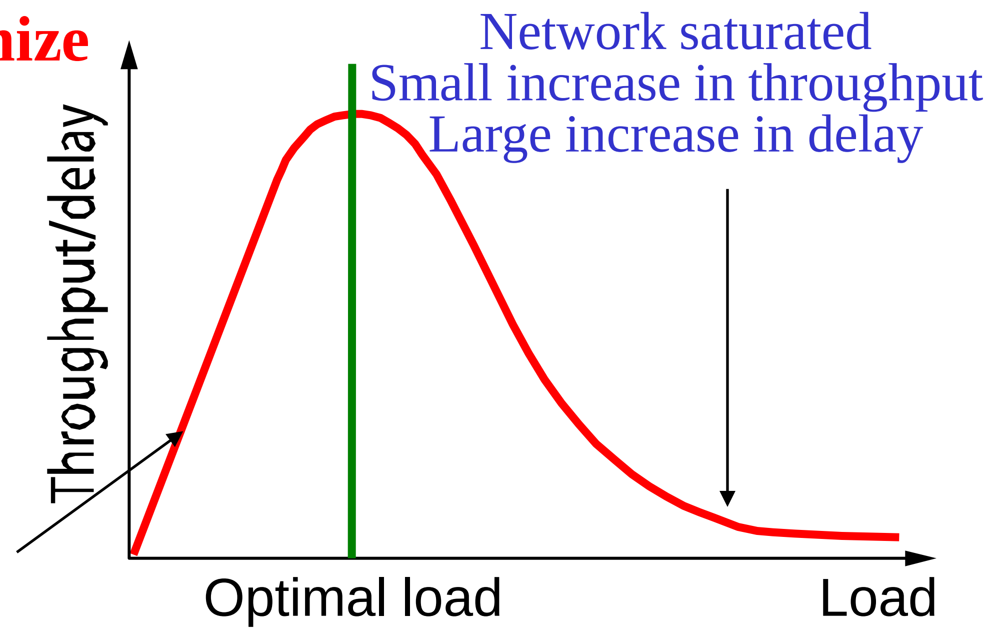
Evaluation Criteria (Effectiveness)

- A network should **effectively** and **fairly** allocate its resources
- For evaluating effectiveness, two metrics of networking are considered: **throughput** and **delay**
- One way to increase throughput is to allow **as many** packets into the network **as possible**
  - Drive the utilization of all the links **up to 100%**
  - Increase the length of the **queues** at each router
    - Longer queues mean packets are **delayed longer** in the network ⇒ a large delay
- We may use the ratio of throughput to delay as a metric for evaluating the effectiveness of a resource allocation scheme
  - **Power = Throughput / Delay**

Evaluation Criteria (Effectiveness)

- Power is based on an **M/M/1 queuing network** that assumes **infinite queues**
  - Real networks have only **finite buffers**
- Power is typically defined relative to a **single connection**
  - Real networks have **multiple, competing connections**

- The objective is to **maximize** the Power ratio
  - Ideally, the resource allocation mechanism would operate at the **peak** of this curve

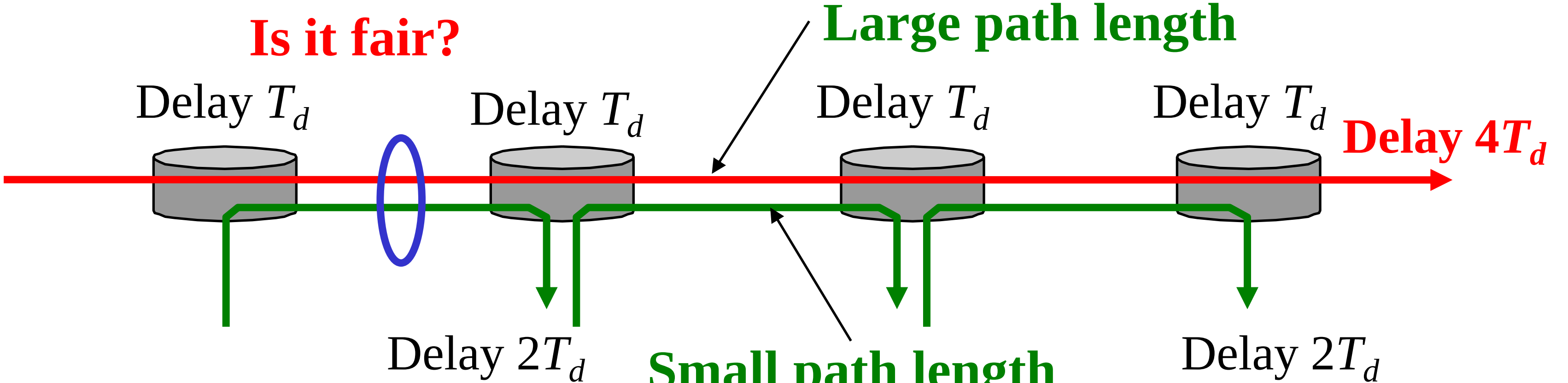


Evaluation Criteria (Fairness)

- A **reservation-based** resource allocation scheme provides an explicit way to create **controlled unfairness**
- In the **absence** of explicit information, we would like for each flow to receive an **equal share** of the bandwidth
- But **equal shares** may not equate to **fair shares**
  - Should we consider the length of the paths being compared?

If the same bandwidth is allocated,

Is it fair?



Evaluation Criteria (Fairness)

- **Raj Jain** has proposed a metric that can be used to **quantify the fairness** of a congestion control mechanism
- Given a set of flow throughputs **(x_1, x_2, …, x_n)** bits/second
- A **fairness index function** is defined as

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

  – Between 0 and 1, with **1 representing greatest fairness**

- All ***n*** flows receive a throughput of 1 unit of data per second
  – $\boldsymbol{x_1 = x_2 = \ldots = x_n}$

$$f(x_1, x_2, \cdots, x_n) = n^2 / n \times n = 1$$

Evaluation Criteria (Fairness)

- **Fairness:** all flows have the same throughput
  - $\boldsymbol{x_1 = x_2 = \ldots = x_n}$
- Suppose one flow receives a throughput of $1+\Delta$

$$f(x_1, x_2, \cdots, x_n) = (n^2 + 2n\Delta + \underline{\Delta^2}) / (n^2 + 2n\Delta + \underline{n\Delta^2}) < 1$$

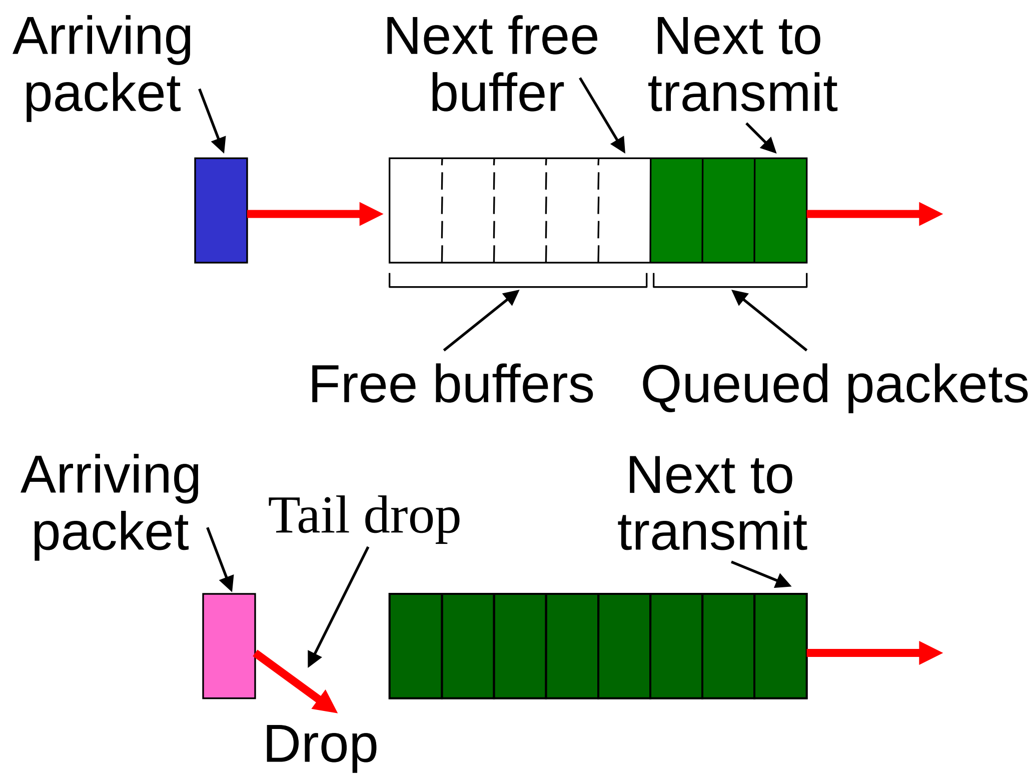
  - No matter Δ is positive or negative, the index **drops** below 1

Queuing Disciplines

FIFO (First-In-First-Out)

- The idea of **FIFO** queuing is "The first packet that arrives at a router is the first packet to be transmitted"
  - Also called **first-come-first-served (FCFS)** queuing

FIFO queue

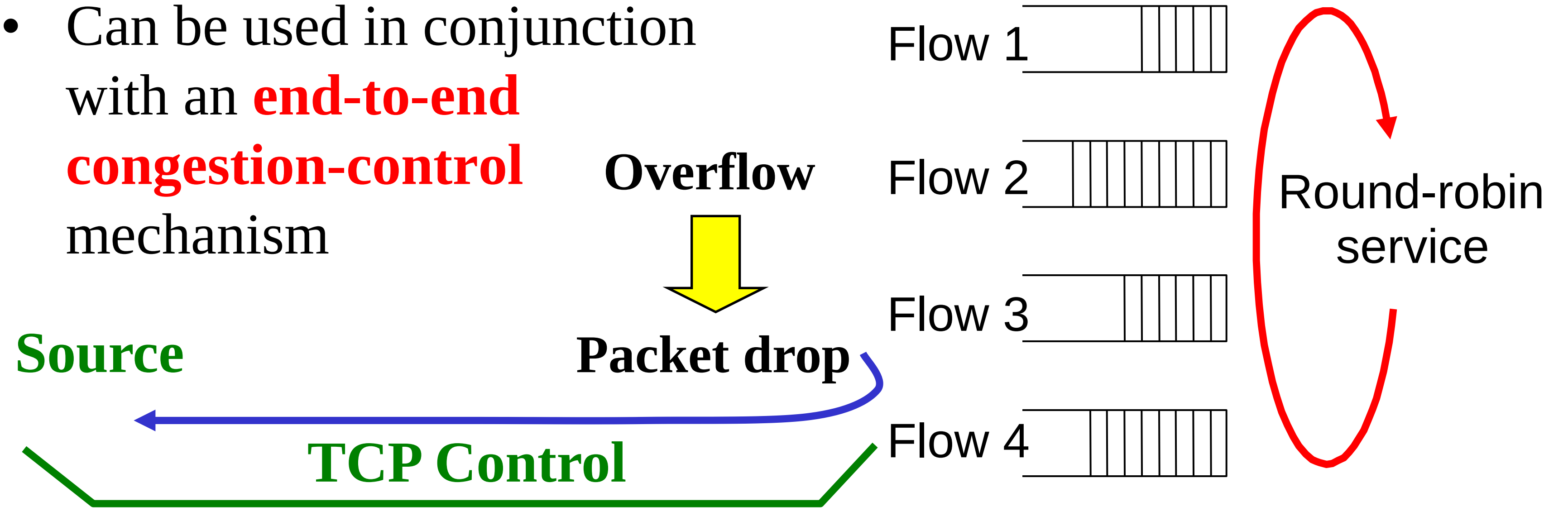


FIFO

- FIFO queuing is **"FIFO with tail drop"** (most widely used)
  - **FIFO** (scheduling discipline): determines the **order** in which packets are transmitted
  - **Tail drop** (drop policy): determines which packets get dropped
- A simple variation on basic FIFO queuing is **priority queuing**
  - Make each packet with priority (carried in the IP **Type of Service (TOS)** field)
  - The router always transmits packets out of the **highest-priority queue** if the queue is nonempty
  - Then moves on to the **next priority queue**
  - Within each priority, packets are still **FIFO**

FQ (Fair Queuing)

- The main problem with FIFO queuing is that it **does not discriminate** between **different traffic sources**
- **FQ** maintains a **separate queue** for each flow currently being handled by the router
  - Services these queues in a **round-robin** manner
- Can be used in conjunction with an **end-to-end congestion-control** mechanism

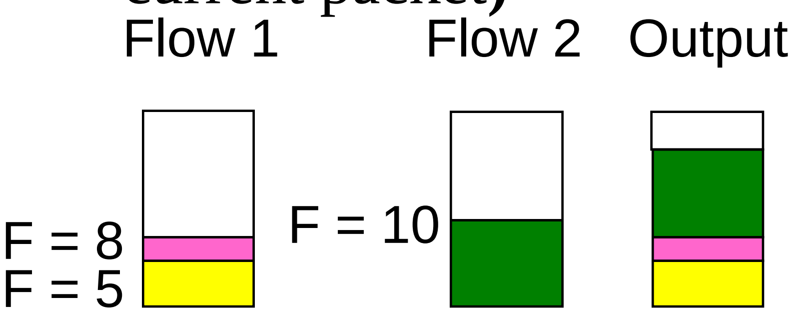


FQ (Fair Queuing)

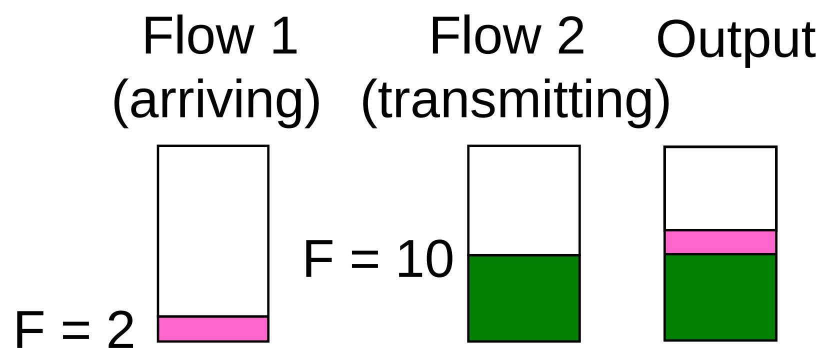
- The packets being processed at a router are not necessarily the same length
  - Takes **packet length** into consideration
- Let P_i denote the length of packet *i*, S_i denote the time when the router starts to transmit packet *i*, and F_i denote the time when the router finishes transmitting packet *i*
  - $F_i = S_i + P_i$
- Let A_i denote the time that packet *i* arrives at the router
  - $S_i = \max(F_{i-1}, A_i)$
  - $F_i = \max(F_{i-1}, A_i) + P_i$

FQ (Fair Queuing)

- All the F_i are treated as **timestamps**
  - The next packet to transmit is the packet with **lowest** timestamp
  - However, a **newly** arriving packet **cannot preempt** a packet that is currently being transmitted
- If the link is fully loaded and there are ***n* flows**, each flow shares **1/*n* of the link bandwidth** (Not perfect: can't preempt current packet)



Shorter packets are sent first



Sending of longer packet is completed first

TCP Congestion Control

TCP Congestion Control

- **TCP** applies an **end-to-end** congestion control mechanism
- The essential strategy of TCP is to send packets into the network **without a reservation**
  - To react to **observable events** that occur
- The idea of TCP congestion control is for each source to determine how much **capacity** is available in the network
- By using **ACKs** to pace the transmission of packets
  - TCP is said to be **self-clocking**

Additive Increase/Multiplicative Decrease

- TCP maintains a new state variable for each connection
  - Called **CongestionWindow**
  - Used by the **source** to limit how much data it is allowed to have in transit at a given time
- The TCP's **effective window**:
  - **MaxWindow =**

    MIN (CongestionWindow, AdvertisedWindow)
  - **EffectiveWindow =**

    MaxWindow – (LastByteSent–LastByteAcked)

Flow Control (Advertised Window)

- **AdvertiseWindow** is sent by the receiver
- To avoid overflowing the receive buffer, the sender computes an effective window that limits how much data it can send:
  - **EffectiveWindow =** (flow control)

    AdvertiseWindow–(LastByteSent–LastByteAcked)
  - **EffectiveWindow =** (congestion control)

    MaxWindow – (LastByteSent–LastByteAcked)
- **CongestionWindow ≥ AdvertiseWindow:** original "Sliding Window Algorithm"
- **CongestionWindow < AdvertiseWindow:** control by network congestion

Additive Increase/Multiplicative Decrease

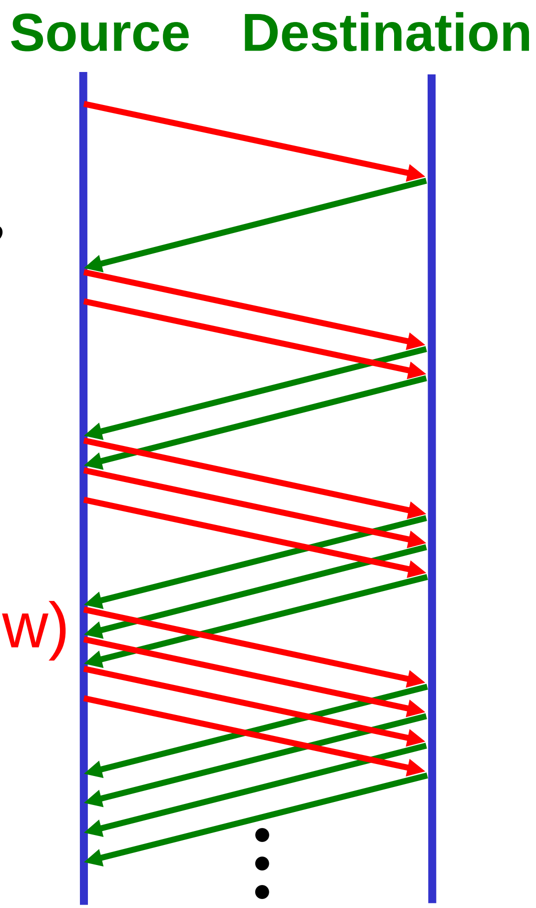
- How TCP learns an appropriate value for **CongestionWindow**
  - The TCP source sets the **CongestionWindow** based on the level of congestion in the network
  - **Decrease** the congestion window when congestion goes **up**
  - **Increase** the congestion window when congestion goes **down**
- The mechanism is called **Additive Increase/Multiplicative Decrease (AIMD)**

Additive Increase/Multiplicative Decrease

- How does the source determine that the network is **congested**?
  - The main reason of packets are not delivered, i.e. **timeout**, is that packet were dropped **due to congestion**
- TCP interprets timeouts as **a sign of congestion** and then **reduces** the transmission rate
- **Multiplicative Decrease:**
  - Each time a timeout occurs, the source sets **CongestionWindow** to **half** of its previous value
    - **CW** = 8; timeout ⇒ **CW** = 4; timeout ⇒ **CW** = 2; timeout ⇒ **CW** = 1
  - **CongestionWindow** is **not allowed** to fall below the size of a single packet, i.e. the **maximum segment size (MSS)**

Additive Increase/Multiplicative Decrease

- It also needs to increase the **CongestionWindow** to take advantage of **newly available** bandwidth in the network
- **Additive Increase:**
  - Every time the source **successfully** sends a CongestionWindow's worth of packets, it **adds** the equivalent of one packet to CongestionWindow
  - Specifically, each time an ACK arrives the congestion window is incremented



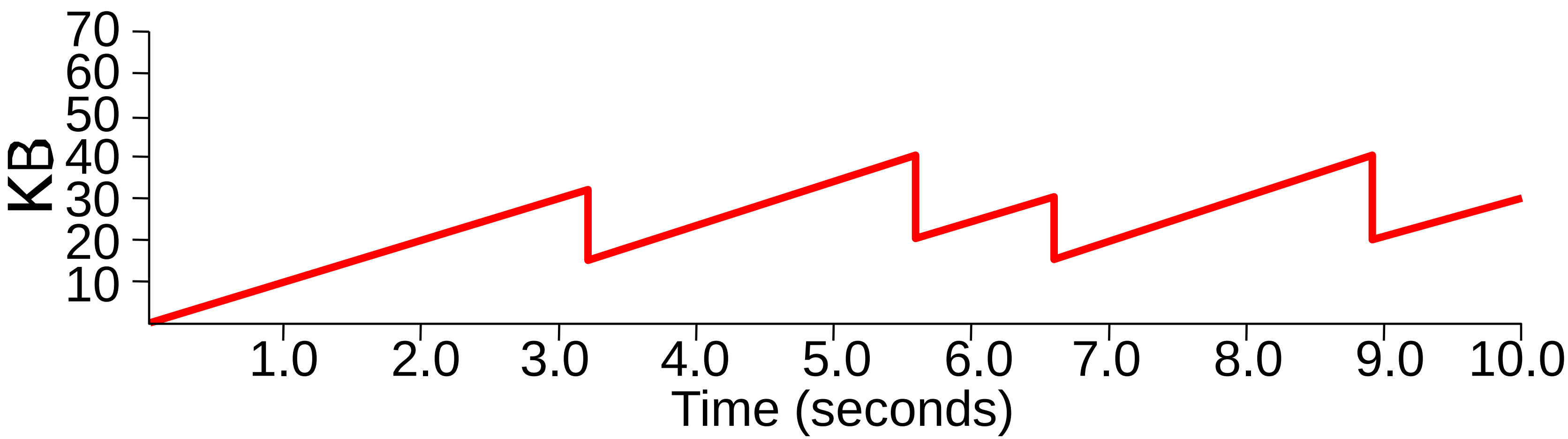
Increment = MSS×(MSS/CongestionWindow)

CongestionWindow + = Increment

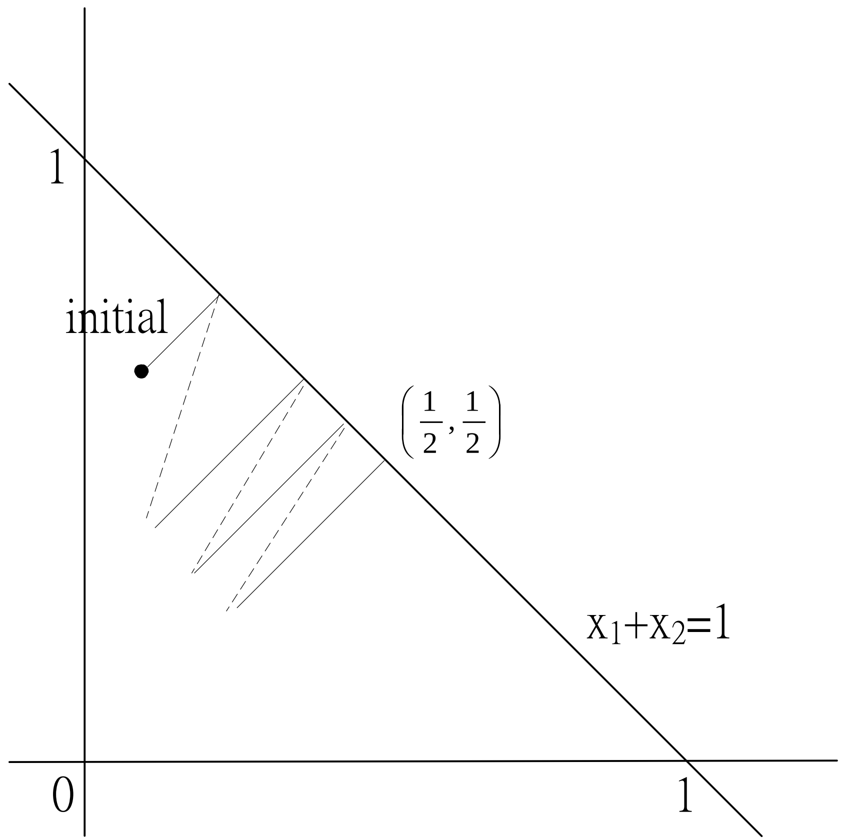
Fraction = MSS/CongestionWindow

Additive Increase/Multiplicative Decrease

- This pattern of continually increasing and decreasing the congestion window continuous throughout the connection
  - **A sawtooth pattern**
- The source is willing to **reduce** its congestion window at a **much faster** rate than it is willing to **increase** this window
  - The consequences of having too large a window are much worse than those of it being too small

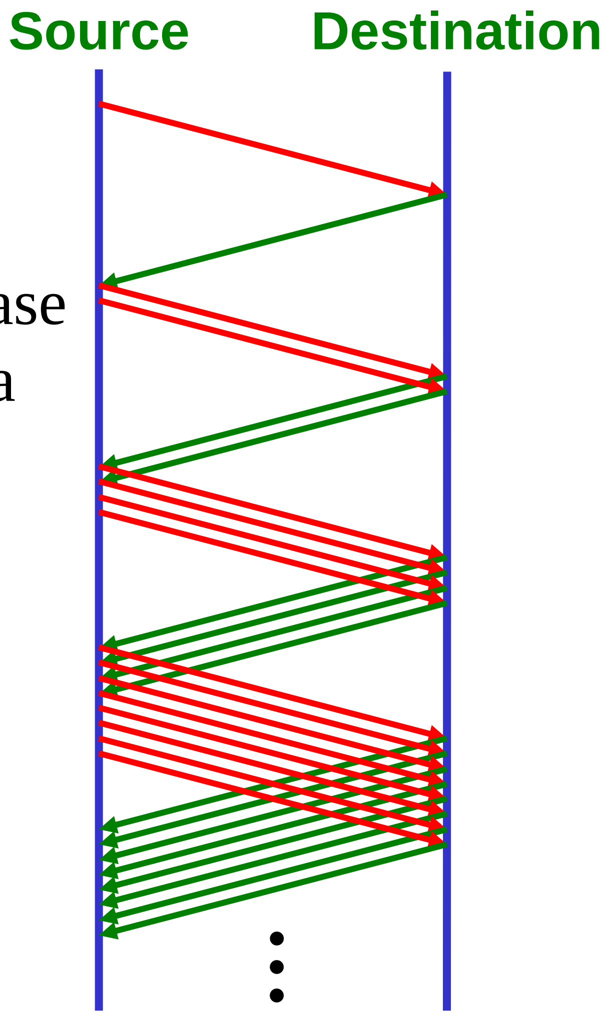


AIMD Fairness



Slow Start

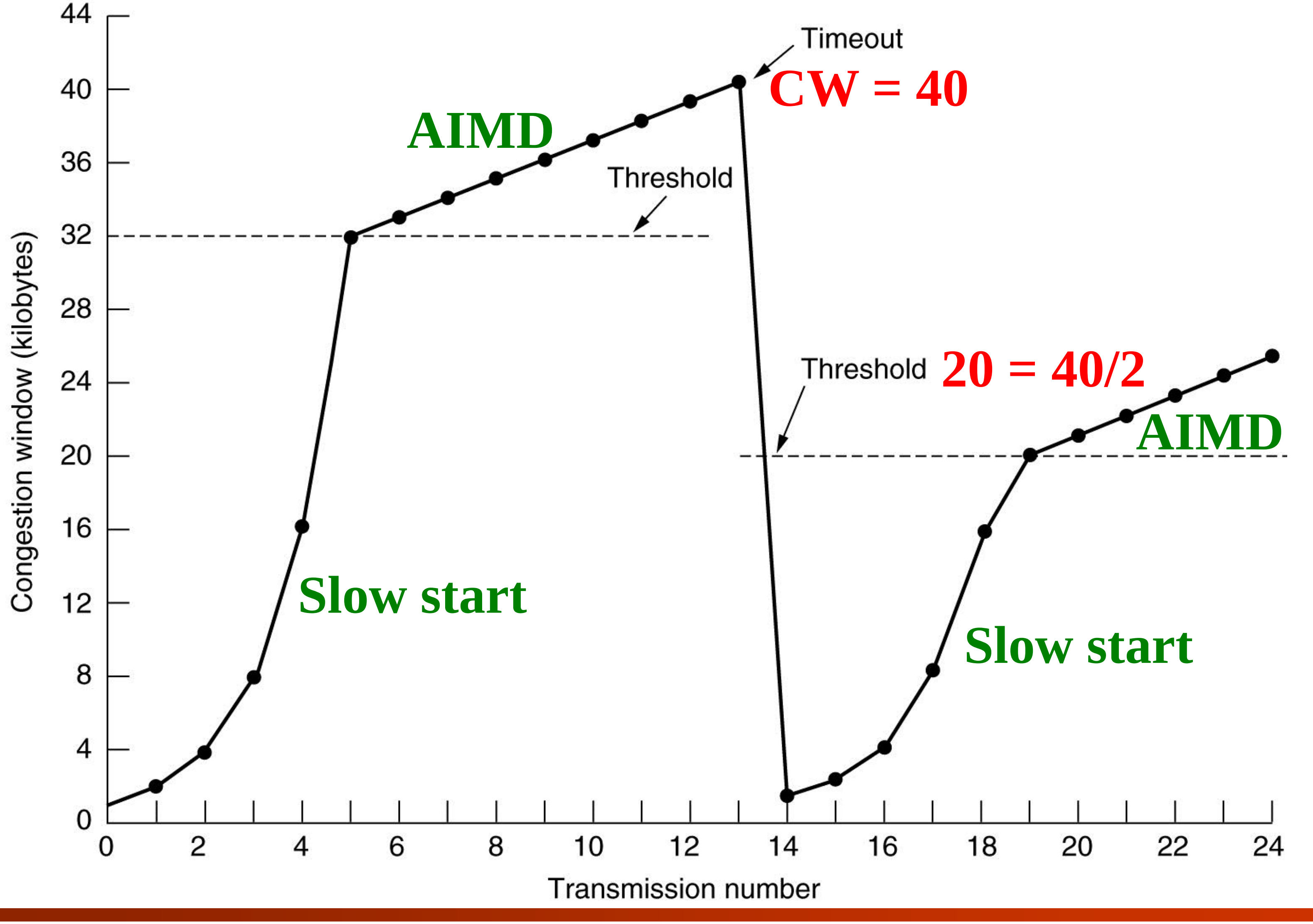
- The AIMD **takes too long to ramp up** a connection from its start to the available bandwidth
  - TCP provides a **second mechanism**, called **slow start**, that is used to increase the congestion window **rapidly** from a cold start
    - **Exponentially** rather than linearly
- For each received ACK, TCP increments CongestionWindow by **2**
  - More **rapidly** than **Additive Increase**



Slow Start

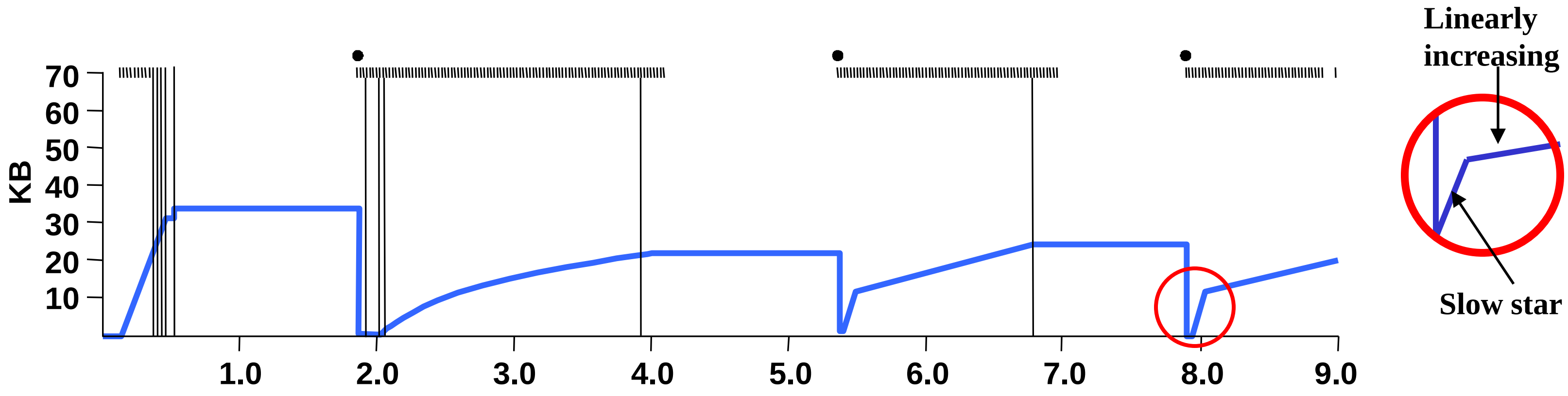
- Two different situations in which slow start runs
- The **very beginning** of a connection
  - The source has **no idea** about the available bandwidth
  - **Double** congestion window for each RTT until **there is a packet loss**
- The connection **goes dead** while **waiting for a timeout to occur** (no ACK comes back, and no packet can be transmitted)
  - The source uses **slow start** to restart the flow of data
  - **Target congestion window** (**CongestionThreshold, CT**)
    - Set to the value of CongestionWindow, that existed prior to the last packet loss, divided by 2
  - After CongestionWindow **has reached the target**, the additive increase (AIMD) is used beyond this point

Slow Start


44
40
36
32
28
24
20
16
12
8
4
0
Congestion window (kilobytes)
0
2
4
6
8
10
12
14
16
18
20
22
24
Transmission number
Timeout
CW = 40
AIMD
Threshold
Threshold
20 = 40/2
AIMD
Slow start
Slow start

Slow Start (cont)

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



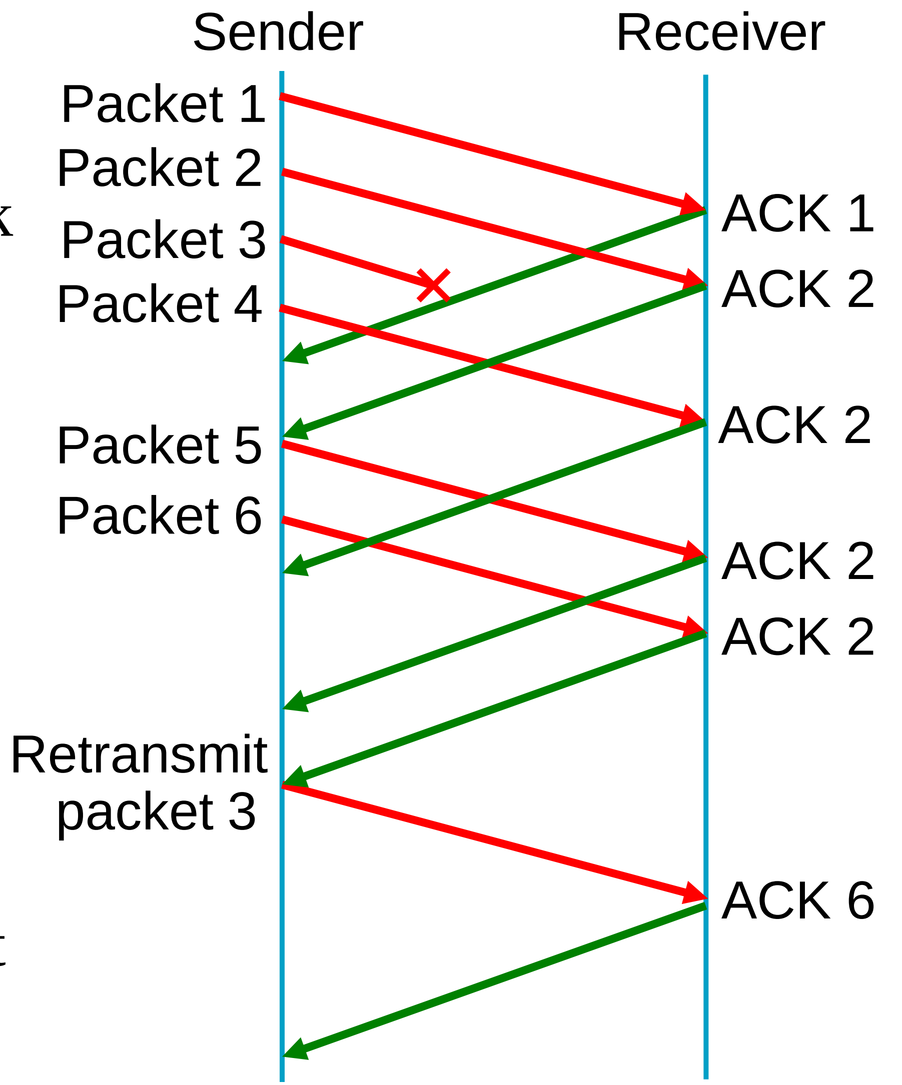
- Problem: lose up to half a **`CongestionWindow`**'s worth of data

Fast Retransmit

- TCP timeouts lead to long periods of time during which the connection went dead while **waiting for a timer to expire**
  - The **fast retransmit mechanism** was added to TCP
  - Triggers the retransmission of a **dropped packet** sooner than the regular timeout mechanism
- Every time a data packet arrives at the receiving side, the receiver responses with a **acknowledgment**
  - If a packet arrives out of order, TCP resends the **last ACK**
  - This **duplicate ACK** suggests that an earlier packet might have been **lost** (it is possible that it just only be delayed)
- In practice, TCP waits until it has seen **three** duplicate ACKs before retransmitting the packet

Fast Retransmit

- The destination receives packets 1 and 2
- Packet 3 is lost in the network
- When packet 4 is received, the destination resents ACK2
- …
- When three duplicate ACK2 is received by the sender
  - **Retransmit packet 3**
- When packet 3 is received, the destination sends a cumulative ACK up to packet 6

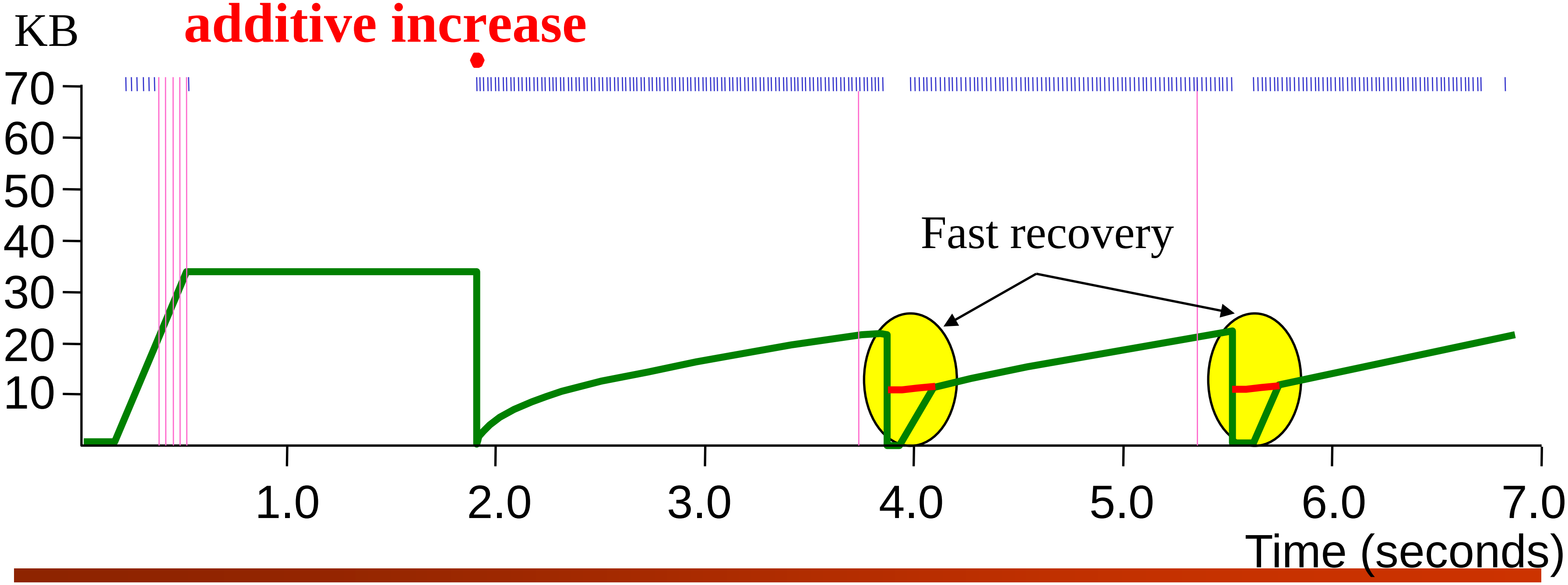


Fast Retransmit

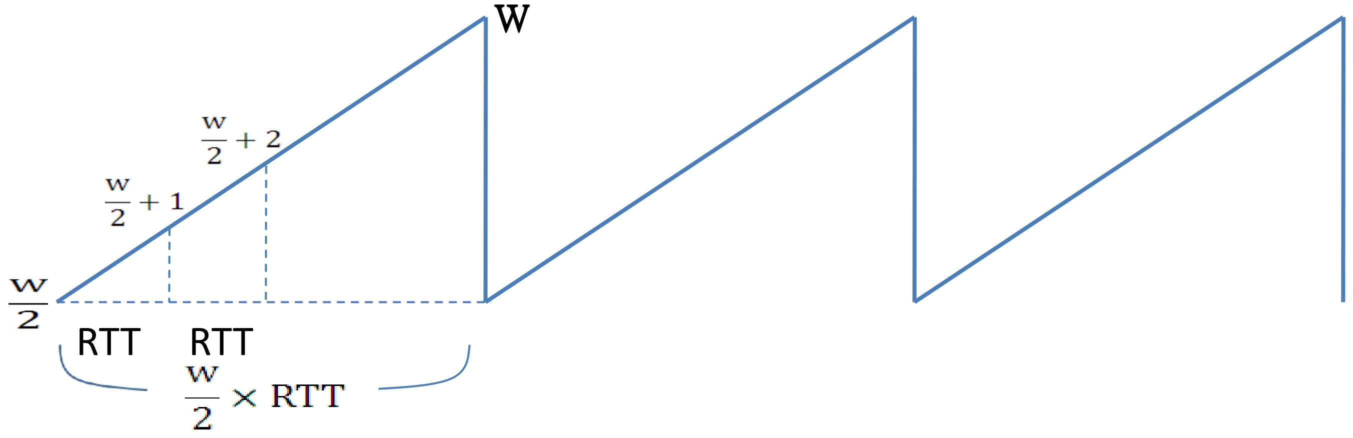
- For TCP with **fast retransmit** mechanism
  - The long period which the congestion window **stays flat** and no packets are sent **have been eliminated**
  - Result in roughly a **20% improvement** in the throughput

Fast Recovery

- **Fast recovery:** uses the ACKs that are still in the pipe to clock the sending of packets
  - **Remove the slow start phase** between when fast retransmit detects a lost packet and additive increase begins
  - Simply **cuts the congestion window in half** and **resumes additive increase**



TCP Equation



- Total number of packets transmitted in a period

$$= \frac{w}{2} + \left(\frac{w}{2}+1\right) + \left(\frac{w}{2}+2\right) + \cdots + w \approx \frac{3}{8} w^2$$

- One packet dropped in a period

$$\rho \approx \frac{1}{\frac{3}{8} w^2} \quad \Rightarrow \quad w \approx \sqrt{\frac{8}{3}\frac{1}{\rho}}$$

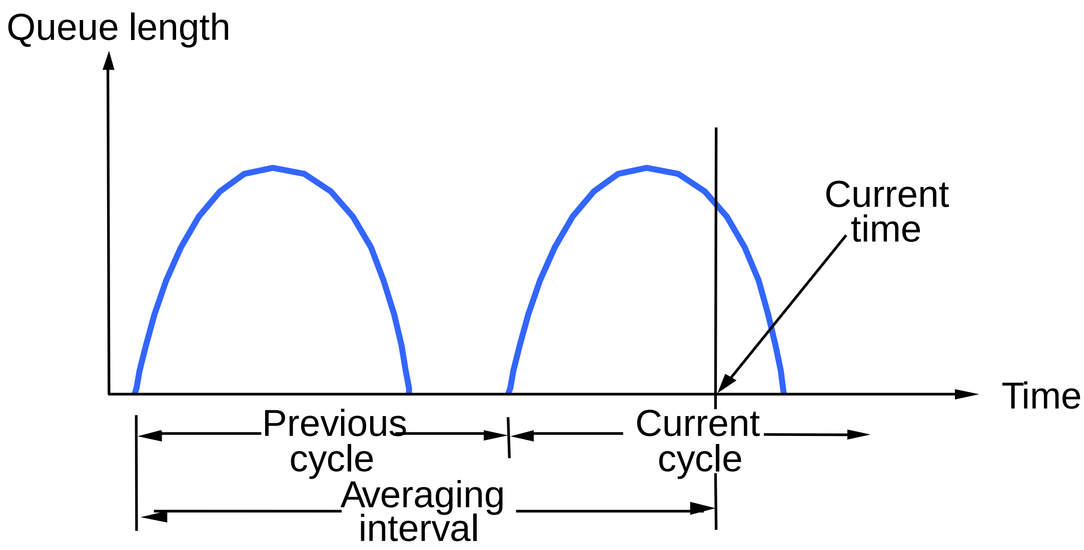
- $\mathrm{RATE} \approx \dfrac{\frac{3}{8} w^2}{\frac{w}{2}\mathrm{RTT}} \approx \dfrac{3}{4}\dfrac{w}{\mathrm{RTT}} \approx \dfrac{1}{\mathrm{RTT}}\dfrac{1}{\sqrt{\rho}}\sqrt{\dfrac{3}{2}} \approx \dfrac{1.22}{\mathrm{RTT}\sqrt{\rho}}$

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 throughout against delay

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

Random Early Detection (RED)

- Notification is implicit
  - just drop the packet (TCP will timeout)
  - could make explicit by marking the packet
- Early 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*

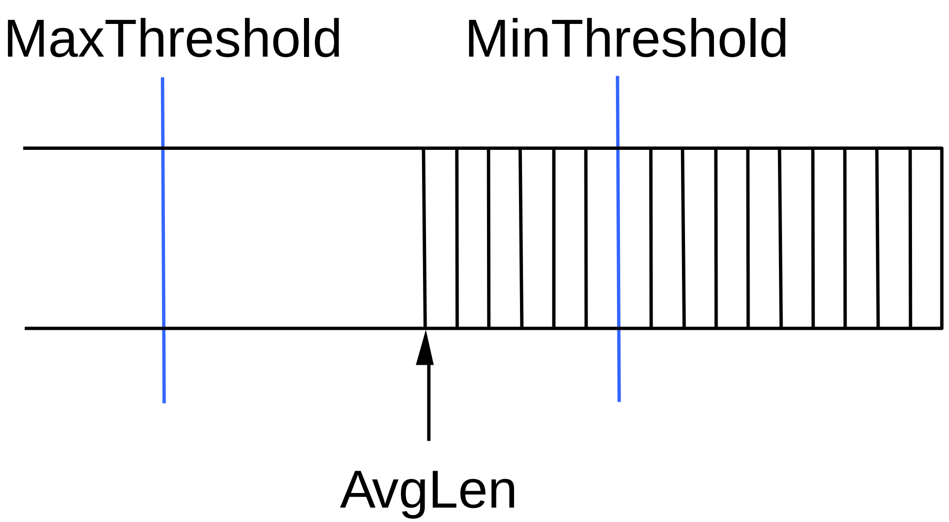
RED Details

- 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

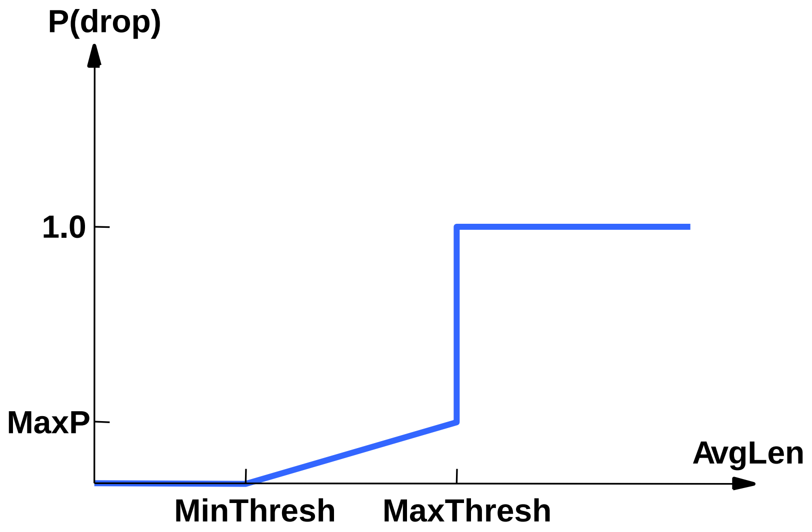
```
if AvgLen <= MinThreshold then
   enqueue the packet
if MinThreshold < AvgLen < MaxThreshold then
   calculate probability P
   drop arriving packet with probability P
if MaxnThreshold <= AvgLen then
   drop arriving packet
```

RED Details (cont)

- Computing probability P

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

- Drop Probability Curve



Tuning RED

- Probability of dropping a particular flow's packet(s) is roughly proportional to the share of the bandwidth that flow is currently getting
- **`MaxP`** is typically set to *0.02*, meaning that when the average queue size is halfway between the two thresholds, the gateway drops roughly one out of *50* packets.
- If traffic is bursty, then **`MinThreshold`** should be sufficiently large to allow link utilization to be maintained at an acceptably high level
- Difference between two thresholds should be larger than the typical increase in the calculated average queue length in one RTT; setting **`MaxThreshold`** to twice **`MinThreshold`** is reasonable for traffic

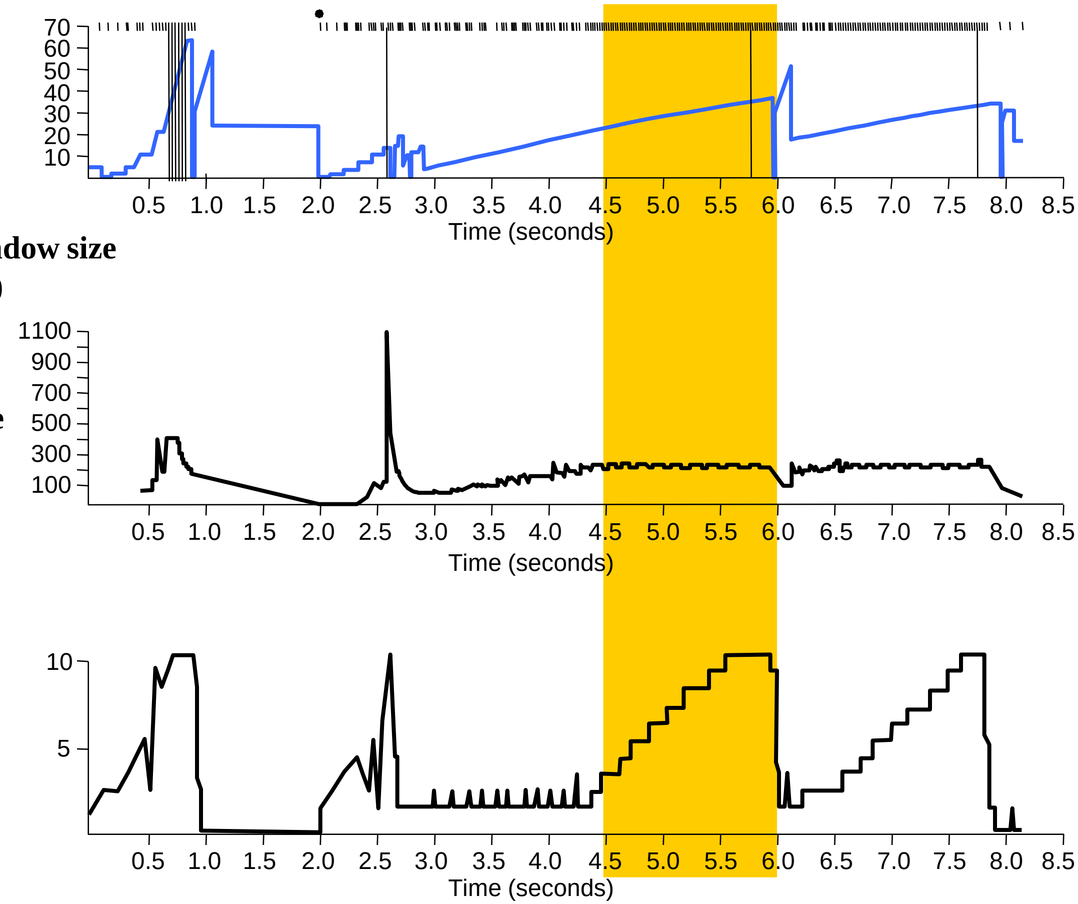
TCP Vegas

- 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

Congestion window size (KB)

Sending rate (KB)

Queue size in router



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
```

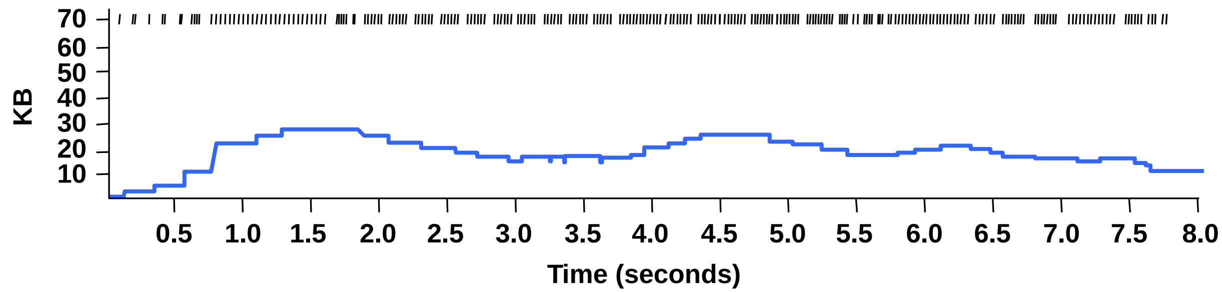
- Source calculates sending rate (**`ActualRate`**) once per RTT
- Source compares **`ActualRate`** with **`ExpectRate`**

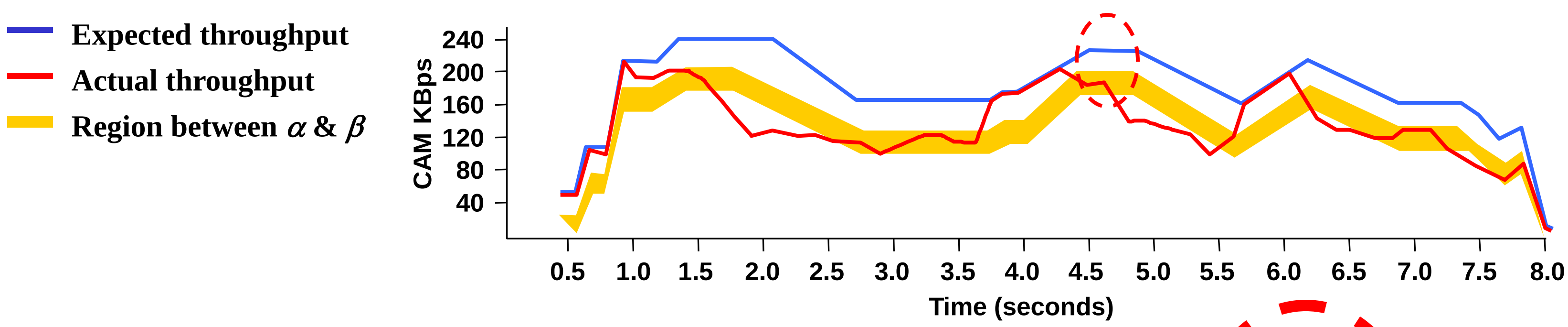
```
Diff = ExpectedRate - ActualRate
if Diff < α
    increase CongestionWindow linearly
else if Diff > β
    decrease CongestionWindow linearly
else
    leave CongestionWindow unchanged
```

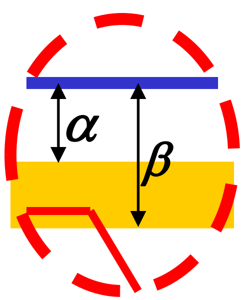
Algorithm (cont)

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

Congestion window size







Quality of Service

Quality of Service

- Packet-switched networks have offered the promise of supporting **multimedia applications**
  - One obstacle is the need for higher-bandwidth links
  - Improvements in coding have reduced the bandwidth needs
- **Audio or video services** are **delay sensitive** (real-time applications). $\Rightarrow$ **timely delivery** is very important
  - Need some sort of assurance from the network that data is likely to **arrive on time**
- **Non-real-time applications:** use an end-to-end **retransmission** strategy to make sure that data arrives **correctly**
  - Such a strategy cannot provide timeliness
  - Retransmission $\Rightarrow$ **long latency**

Quality of Service

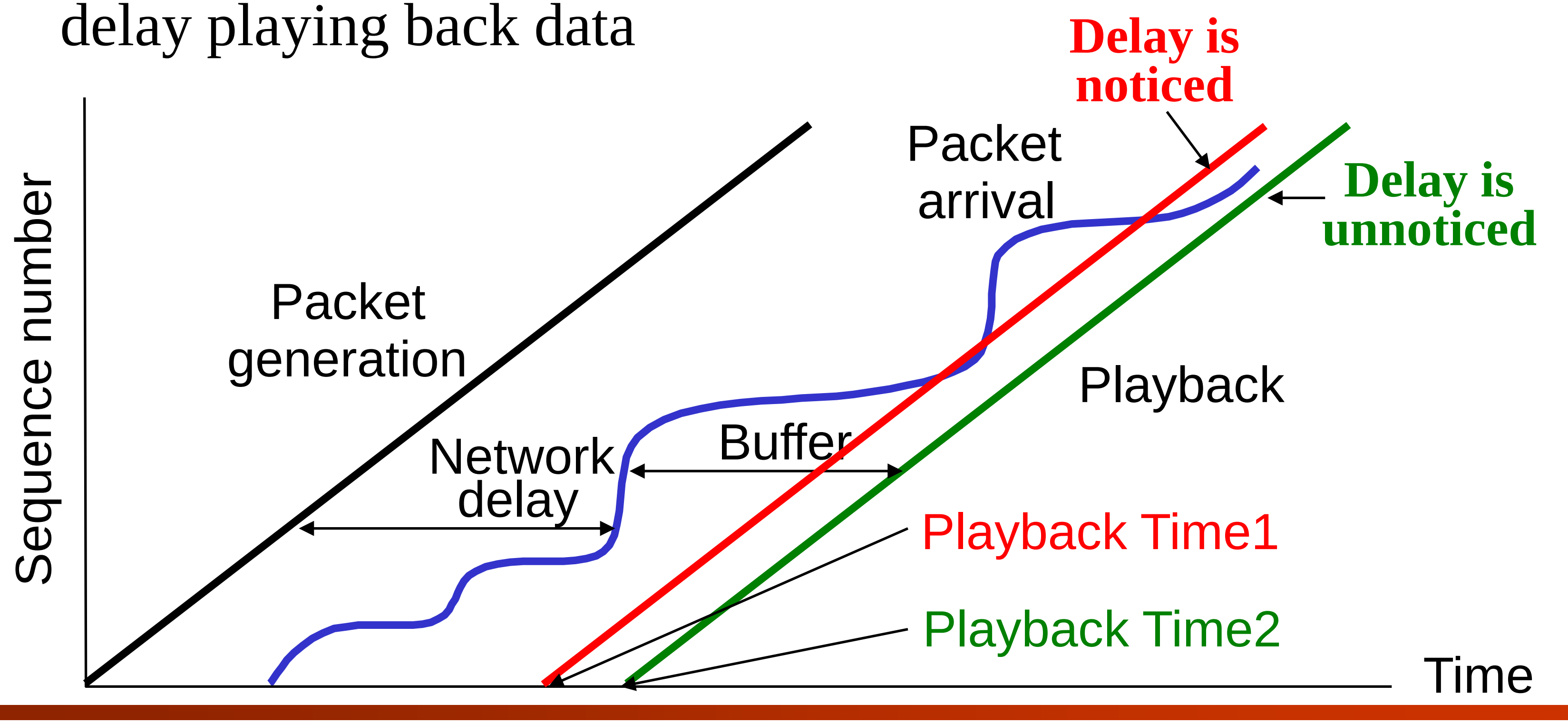
- **Timely arrival** must be provided by the **network** itself (routers), not just at the network edges (hosts)
- **Best-effort model** is **not sufficient** for real-time applications
  - A new service model is required
  - Applications that need **higher assurances** can ask the network for them
- The network will treat some packets **(real-time)** differently from others **(non-real-time)**
- A network that can provide these different levels of service is often said to support **quality of service (QoS)**

Application Requirements (Real-Time)

- The data must be **played back** at some appropriate rate
  - Each sample has a particular **playback time** in the receiver
- If data arrives **after** its appropriate playback time, either delayed or dropped, it is essentially **useless**
- It is **impossible** to make sure that all samples take exactly the same amount of time to traverse the network
  - The queue lengths **vary with time** $\Rightarrow$ so does **delays**
- To **buffer** up some amount of data in reserve, thereby always providing a store of packets waiting to be played back

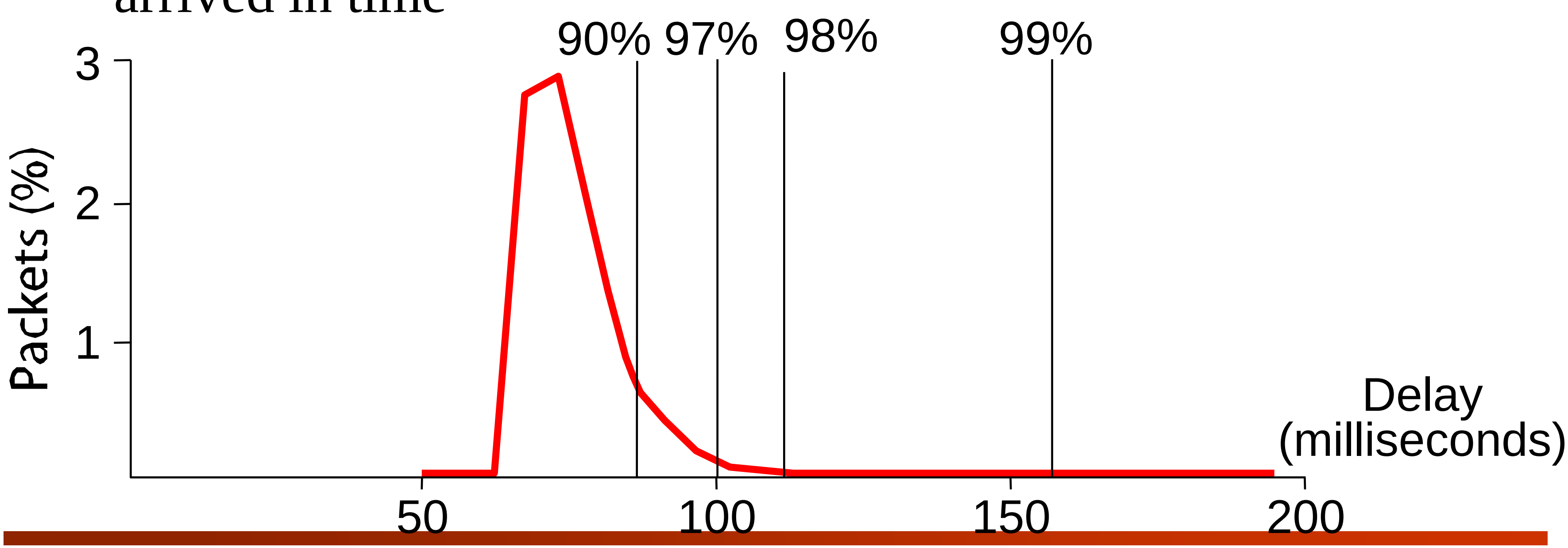
Application Requirements (Real-Time)

- As long as the playback line is far enough to the right in time, **the variation in network delay** is never noticed by the application
- For audio applications, there are limits to **how far** we can delay playing back data



Application Requirements (Real-Time)

- We can measure the **one-way delay** over a certain path across the Internet
  - The key factor is the **variability** of the delay
- Set the playback point to **100 ms** ⇒ 3% packets arrive too late
- Set the playback point to **200 ms** ⇒ ensures that all packets arrived in time



Approaches to QoS Support

- Two broad categories of approaches that support QoS:
  - **Fine-grained approaches:** provide QoS to **individual** applications or flows
    - **RSVP (Resource Reservation Protocol)**
  - **Coarse-grained approaches:** provide QoS to large classes of data or **aggregated** traffic
    - **Differentiated Services**

RSVP (Resource Reservation Protocol)

Integrated Services (RSVP)

- Service classes:
  - **Guaranteed service:** the network should guarantee that the **maximum packet delay** has some specified value
    - The application can set the **playback point** so that no packet will ever arrive after its playback time
  - **Controlled load service:** **tolerant, adaptive** applications
    - Uses a **queuing mechanism** to isolate the controlled load traffic from other traffic
    - Uses some form of **admission control** to limit the total amount of controlled load traffic such that the load is kept reasonable low
    - **Adjusts the playback point** as network delay varies, and controls a reasonable packet loss rate

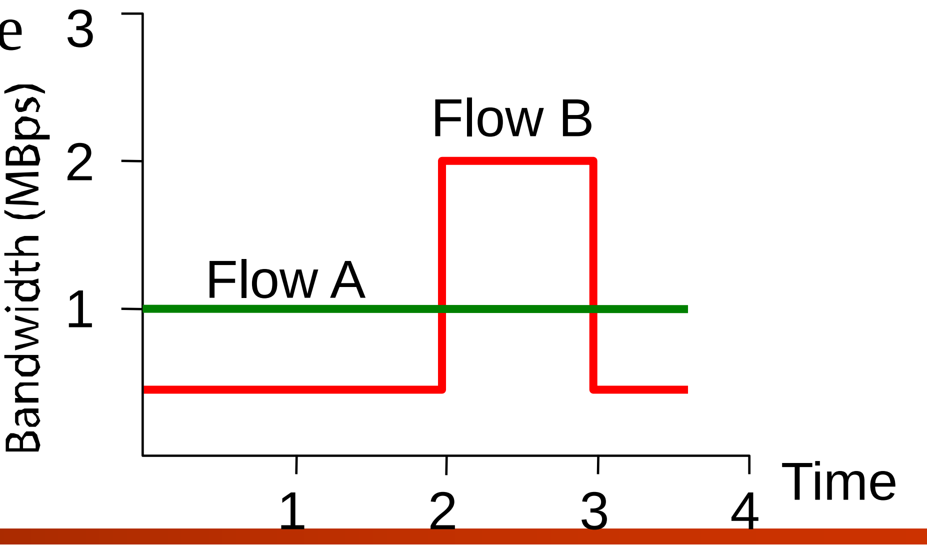
Integrated Services (RSVP)

- Four major parts for providing integrated services:
  - **Flowspec:** the set of information that a user provides to the network regarding the service
  - **Admission control:** the process of deciding when to **say no** to a **requesting user**
  - **Resource reservation:** the mechanism by which the users of the network and the components of the network **exchange information**, such as requests for service
  - **Packet scheduling:** the mechanism of managing the way packets are **queued** and **scheduled for transmission** in the switches and routers

Flowspec

- There are two separable parts to the flowspec:
  - **RSpec:** describes the **service requested** from the network
    - For a controlled load service: **no additional parameters**
    - For a guaranteed service: specify a **delay** target or bound
  - **TSpec:** describes the flow's **traffic characteristics**
    - Specify the **requesting bandwidth**: the **average** bit rate and the **peak** bit rate

The same average bit rate
Different peak bit rates



Admission Control

- Admission control looks at the **TSpec** and **RSpec** of the flow and tries to decide if the desired service can be provided
  - Given the currently **available resources**
  - Without causing any **previously admitted** flow to receive **worse service** than it had requested
- Admission control is very dependent on the **type of requested service** and on the **queuing discipline** employed in the routers
  - For a **guaranteed service:** a good algorithm to make a **definitive yes/no decision** is required
  - For a **controlled load service:** it may be based on **heuristics**
    - E.g., current delays are **far inside the bounds** $\Rightarrow$ It should be able to admit another flow without difficulty

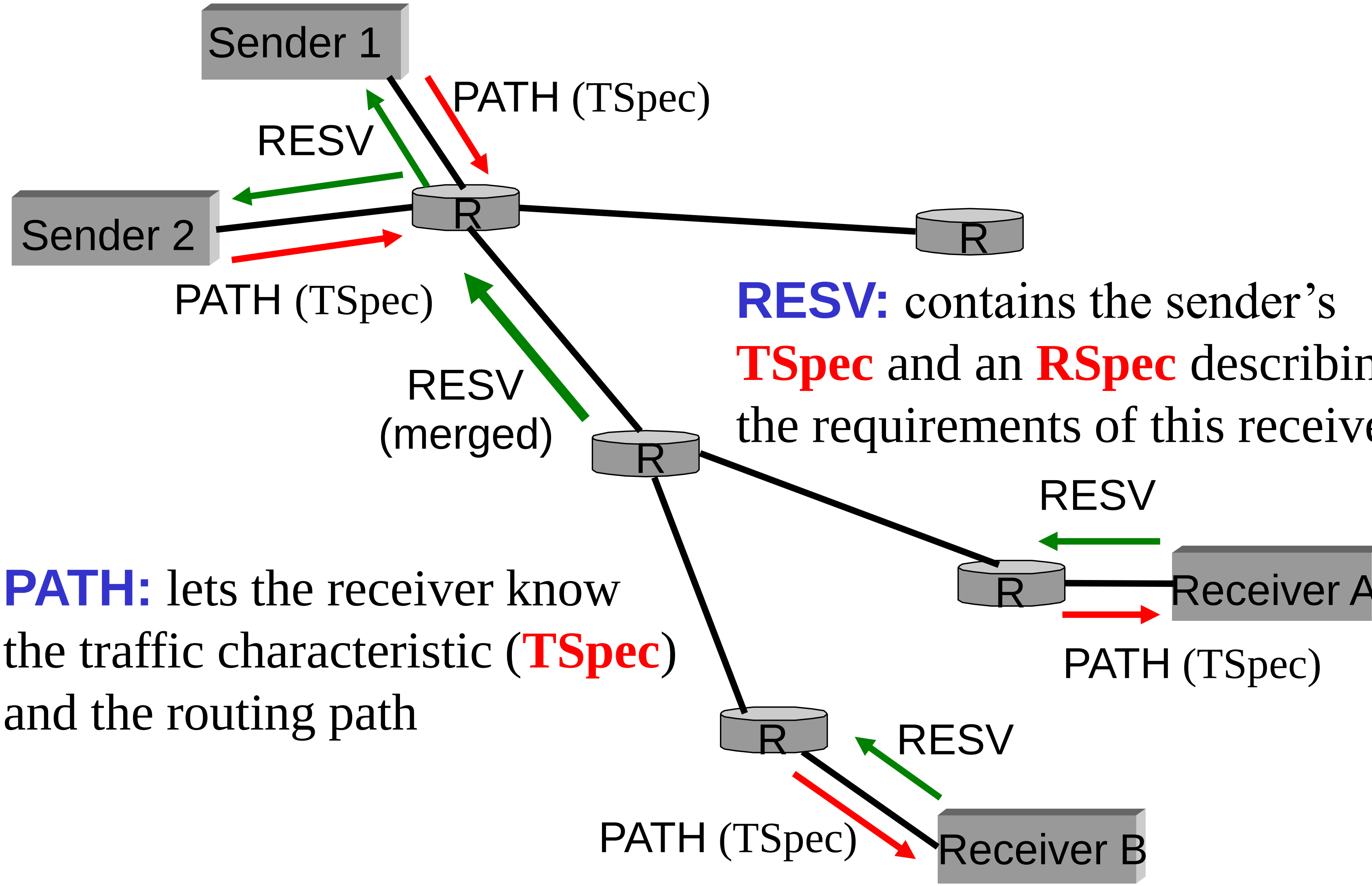
Admission Control

- **Policing:** is a function applied on a **per-packet basis** to make sure that a flow conforms to the **TSpec** that was used to make the reservation
  - There are several options, the obvious one being to **drop offending packets**
  - Another option is to **drop the offending packets first** if any packets are needed to drop

Resource Reservation (RSVP)

- The most important protocol is **Resource Reservation Protocol (RSVP)**
  - Maintains the **robustness** of connectionless networks by using the idea of **soft state** in the routers
  - Supports **multicast** flows just as effectively as **unicast** flows
- Consider the case of one sender and one receiver trying to get a reservation for the traffic flowing between them
  - The receiver needs to know **what traffic** the sender is likely to send (to make an appropriate reservation)
  - It needs to know **what path** the packets will follow (to establish a reservation **at each router on the path**)

Resource Reservation (RSVP)



RESV: contains the sender's **TSpec** and an **RSpec** describing the requirements of this receiver

PATH: lets the receiver know the traffic characteristic (**TSpec**) and the routing path

Packet Scheduling

- For the routers to actually deliver the packets, all the routers on the path must
  - **Classifying packets:** **classify** each packet with the appropriate **reservation** so that it can be handled correctly
  - **Packet scheduling:** **manage** the packets in the **queues** according to the requested service
- Classifying packets is done by examining up to **five fields** in the packet: **the source address, destination address, protocol number, source port, and destination port**
  - Based on this information, the packet can be placed in the appropriate class
- Packet scheduling is an area where implementers can try to do creative things to **realize the service model efficiently**

Scalability Issues

- In the **best-effort service** model, routers store **little or no state** about the individual flows passing through them
- RSVP raises the possibility that **every flow** passing through a router might have **a corresponding reservation**
  - Each of those reservations needs some amount of state
    - Stored in memory and refreshed periodically
  - The router needs to **classify, police, and queue** each of those flows
- Suppose that every flow on an OC-48 (2.5 Gbps) link represents a 64-Kbps audio stream
  - $2.5 \times 10^9 / 64 \times 10^3 = 39{,}000$ flows
  - Maintaining **per-flow** state may be **not practical**

$\Rightarrow$ **Differentiated services**

Differentiated Services

Differentiated Services

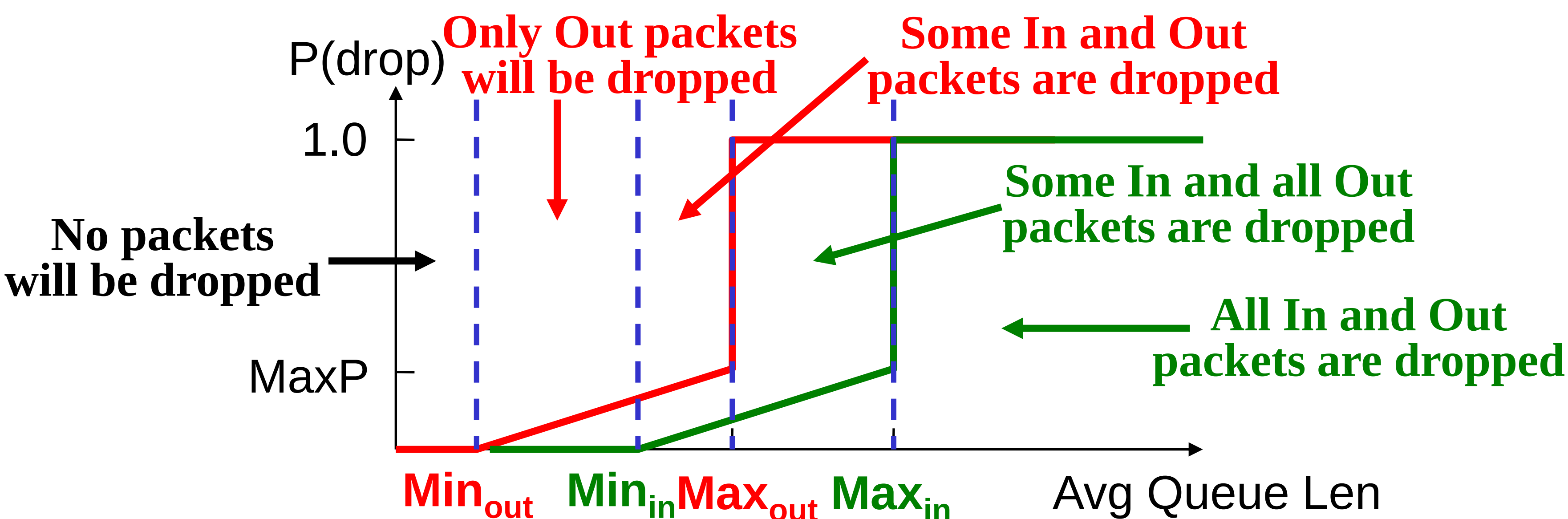
- **Differentiated services model (DiffServ)** allocates resources to a small number of classes of traffic
  - Some proposed approaches divide traffic into **two classes**
  - Enhance the best-effort service model by adding just one new class called **"premium"**
- Each router should figure out which packets are **premium** and which are regular old **best-effort**
  - Could be done by using a bit in the packet header
  - The **router at the edge** of an Internet service provider's network might set this bit for some packets
- In practice: based on the behavior of individual routers called **"per-hop behaviors" (PHBs)**

Differentiated Services (EF)

- **Six bits** (taken from the old **TOS** (type of service) **byte** of the IP header) have been allocated for **DiffServ code points (DSCP)**
  - Identify a particular PHB to be applied to a packet
- One of the simplest PHBs is **"expedited forwarding" (EF)**
  - The packets should be forwarded by the router with **minimal delay and loss**
  - One possible strategy is to give EF packets **strict priority** over all other packets
  - Another is to perform **weighted fair queuing**
    - The weight of EF set is sufficiently high
    - All EF packets can be delivered quickly

Differentiated Services (AF)

- Another PHB is known as **"assured forwarding" (AF)**
  - Based on an approach known as **"RED with In and Out" (RIO)** or **"Weighted RED"** (RED, random early detection)
  - A custom is allowed to send up to ***y* Mbps** of assured traffic
  - If it sends **less** than *y* Mbps, all packets are marked **"in"**
  - If it **exceeds** the rate, the excess packets are marked **"out"**



Differentiated Services (WFQ)

- A third way is **weighted fair queuing (WFQ):** it uses the DSCP value to determine which queue to put a packet into
  - Define the **best-effort queue** and the **premium queue**
  - Choose weights that makes the premium packets get a better service than the best-effort packets
- E.g. premium queue weight = **1**, best-effort queue weight = **4**
  - The bandwidth available to premium packets is

$$B_{premium} = W_{premium} / (W_{premium} + W_{best\_effort}) = 0.2$$

  - Reserve **20%** bandwidth of the link for premium packets
  - If the offered load of premium traffic is 10% of the link on average, the service of premium traffic will be very good