IF2230 Jaringan Komputer

Congestion Control and Resource Allocation

Robithoh Annur
Andreas Bara Timur
Monterico Andrian
Prof. Nana Rachmana

Problem

- We have seen enough layers of protocol hierarchy to 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
- Queuing Disciplines
- TCP Congestion Control
- Congestion Avoidance Mechanism
- Quality of Service

Congestion Control and Resource Allocation

- Resources
  - Bandwidth of the links
  - Buffers at the routers and switches
- Packets contend at a router for the use of a link, with each contending packet placed in a queue waiting for its turn to be transmitted over the link
- When too many packets are contending for the same link
  - The queue overflows
  - Packets get dropped
    - Network is congested!
- Network should provide a congestion control mechanism to deal with such 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
  - No need for congestion control
- Allocating resources with any precision is difficult
  - Resources are distributed throughout the network
- On the other hand, we can always let the sources send as much data as they want
  - Then recover from the congestion when it occurs
  - Easier approach but it can be disruptive because many packets many be discarded by the network before congestions can be controlled

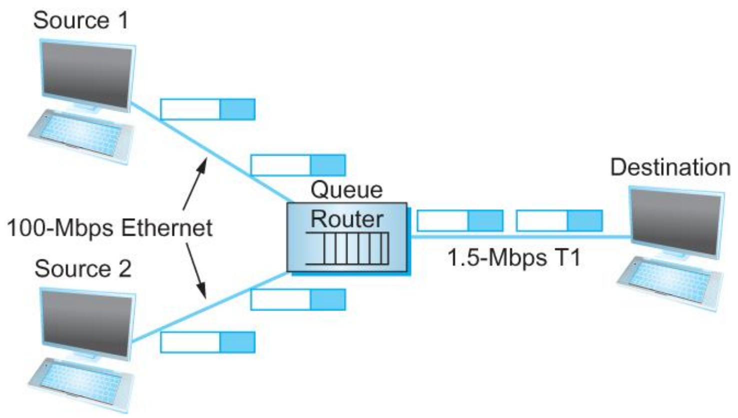
Congestion Control and Resource Allocation

- Congestion control and resource allocations involve both hosts and network elements such as routers

- 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 hosts' end
  - The congestion control mechanism paces how fast sources are allowed to send packets

Issues in Resource Allocation

- Network Model
  - Packet Switched Network
    - We consider resource allocation in a packet-switched network (or internet) consisting of multiple links and switches (or routers).
    - In such an environment, a given source may have more than enough capacity on the immediate outgoing link to send a packet, but somewhere in the middle of a network, its packets encounter a link that is being used by many different traffic sources



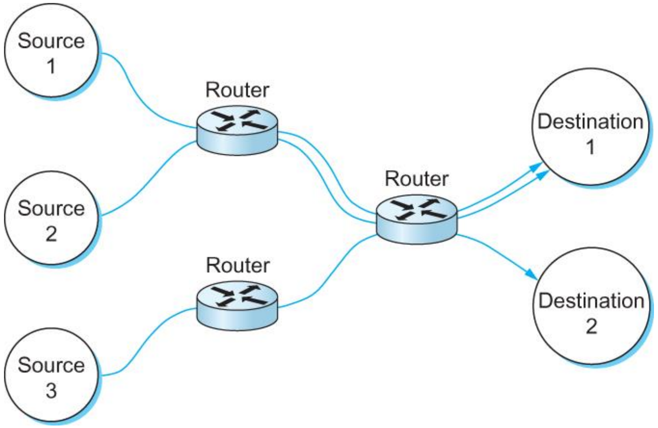
A potential bottleneck router.

Issues in Resource Allocation

- Network Model
  - Connectionless Flows
    - For much of our discussion, we assume that the network is essentially connectionless, with any connection-oriented service implemented in the transport protocol that is running on the end hosts.
      - We need to qualify the term “connectionless” because our classification of networks as being either connectionless or connection-oriented is a bit too restrictive; there is a gray area in between.
      - In particular, the assumption that all datagrams are completely independent in a connectionless network is too strong.
      - The datagrams are certainly switched independently, but it is usually the case that a stream of datagrams between a particular pair of hosts flows through a particular set of routers

Issues in Resource Allocation

- Network Model
  - Connectionless Flows
    - One of the powers of the flow abstraction is that flows can be defined at different granularities. For example, a flow can be host-to-host (i.e., have the same source/destination host addresses) or process-to-process (i.e., have the same source/destination host/port pairs).
    - In the latter case, a flow is essentially the same as a channel, as we have been using that term throughout this book. The reason we introduce a new term is that a flow is visible to the routers inside the network, whereas a channel is an end-to-end abstraction.



Multiple flows passing through a set of routers

Issues in Resource Allocation

- Network Model
  - Connectionless Flows
    - Because multiple related packets flow through each router, it sometimes makes sense to maintain some state information for each flow, information that can be used to make resource allocation decisions about the packets that belong to the flow. This state is sometimes called soft state.
    - The main difference between soft state and “hard” state is that soft state need not always be explicitly created and removed by signalling.
    - Soft state represents a middle ground between a purely connectionless network that maintains no state at the routers and a purely connection-oriented network that maintains hard state at the routers.
    - In general, the correct operation of the network does not depend on soft state being present (each packet is still routed correctly without regard to this state), but when a packet happens to belong to a flow for which the router is currently maintaining soft state, then the router is better able to handle the packet.

Issues in Resource Allocation

- Taxonomy
  - Router-centric versus Host-centric
    - In a router-centric design, each router takes responsibility for deciding when packets are forwarded and selecting which packets are to dropped, as well as for informing the hosts that are generating the network traffic how many packets they are allowed to send.
    - In a host-centric design, the end hosts observe the network conditions (e.g., how many packets they are successfully getting through the network) and adjust their behavior accordingly.
    - Note that these two groups are not mutually exclusive.

Issues in Resource Allocation

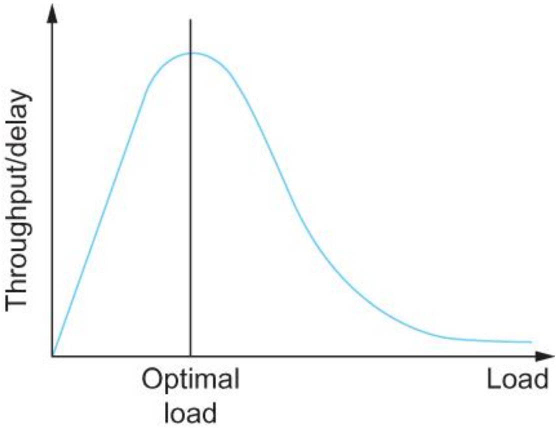
- Taxonomy
  - Reservation-based versus Feedback-based
    - In a reservation-based system, some entity (e.g., the end host) asks the network for a certain amount of capacity to be allocated for a flow.
      - Each router then allocates enough resources (buffers and/or percentage of the link's bandwidth) to satisfy this request. If the request cannot be satisfied at some router, because doing so would overcommit its resources, then the router rejects the reservation.
    - 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.
      - This feedback can either be *explicit (i.e., a congested router sends a "please* slow down" message to the host) or it can be *implicit (i.e., the end host adjusts its sending* rate according to the externally observable behavior of the network, such as packet losses).
  - 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.

Issues in Resource Allocation

- 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 and delay.
      - Clearly, we want as much throughput and as little delay as possible.
      - Unfortunately, these goals are often somewhat at odds with each other.
      - One sure way for a resource allocation algorithm to increase throughput is to allow as many packets into the network as possible, so as to drive the utilization of all the links up to 100%.
      - We would do this to avoid the possibility of a link becoming idle because an idle link necessarily hurts throughput.
      - The problem with this strategy is that increasing the number of packets in the network also increases the length of the queues at each router. Longer queues, in turn, mean packets are delayed longer in the network

Issues in Resource Allocation

- Evaluation Criteria
  - Effective Resource Allocation
    - To describe this relationship, some network designers have proposed using the ratio of throughput to delay as a metric for evaluating the effectiveness of a resource allocation scheme.
    - This ratio is sometimes referred to as the *power of the network.*
    - Power = Throughput/Delay



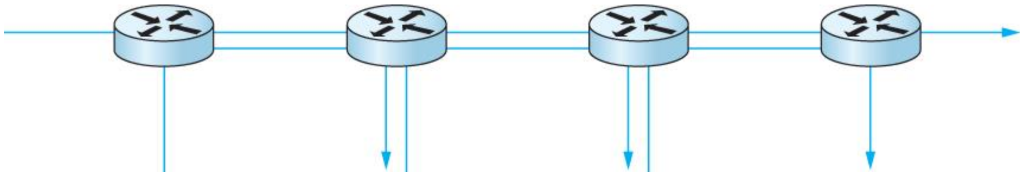
Ratio of throughput to delay as a function of load

Issues in Resource Allocation

- Evaluation Criteria
  - Fair Resource Allocation
    - The effective utilization of network resources is not the only criterion for judging a resource allocation scheme.
    - We must also consider the issue of fairness. However, we quickly get into murky waters when we try to define what exactly constitutes fair resource allocation.
    - For example, a reservation-based resource allocation scheme provides an explicit way to create controlled unfairness.
    - With such a scheme, 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.

Issues in Resource Allocation

- Evaluation Criteria
  - Fair Resource Allocation
    - In the absence of explicit information to the contrary, 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?
      - Consider this figure.



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

Issues in Resource Allocation

- 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.
      - Jain's fairness index is defined as follows. Given a set of flow throughputs ($x_1, x_2, \ldots, x_n$) (measured in consistent units such as bits/second), the following function assigns a fairness index to the flows:

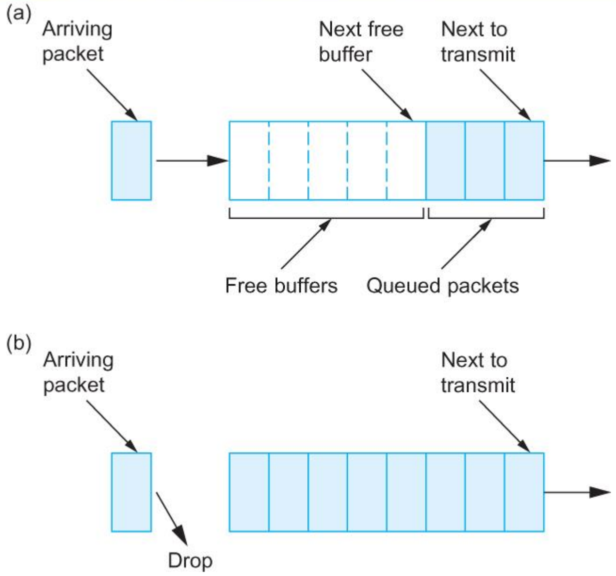
$$f(x_1, x_2, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2}$$

      - The fairness index always results in a number between 0 and 1, with 1 representing greatest fairness.

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
  - Given that the amount of buffer space at each router is finite, if a packet arrives and the queue (buffer space) is full, then the router discards that packet
  - This is done without regard to which flow the packet belongs to or how important the packet is. This is sometimes called *tail drop, since packets that arrive at* the tail end of the FIFO are dropped
  - Note that tail drop and FIFO are two separable ideas. FIFO is a *scheduling discipline*—it determines the order in which packets are transmitted. Tail drop is a *drop policy—it determines which packets get dropped*

Queuing Disciplines



(a) FIFO queuing; (b) tail drop at a FIFO queue.

Queuing Disciplines

- A simple variation on basic FIFO queuing is priority queuing. The idea is to mark each packet with a priority; the mark could be carried, for example, in the IP header.
- The routers then implement multiple FIFO queues, one for each priority class. The router always transmits packets out of the highest-priority queue if that queue is nonempty before moving on to the next priority queue.
- Within each priority, packets are still managed in a FIFO manner.

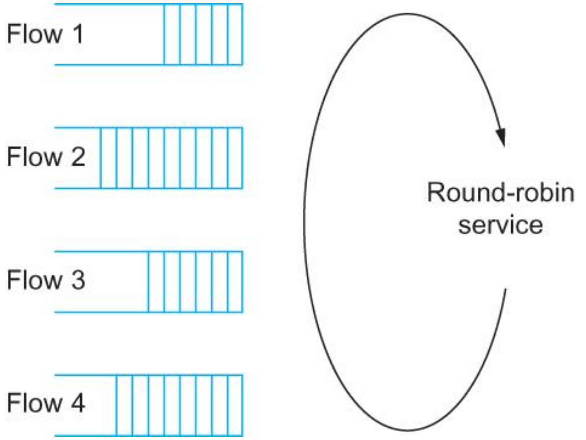
Queuing Disciplines

- Fair Queuing
  - The main problem with FIFO queuing is that it does not discriminate between different traffic sources, or it does not separate packets according to the flow to which they belong.
  - Fair queuing (FQ) is an algorithm that has been proposed to address this problem. The idea of FQ is to maintain a separate queue for each flow currently being handled by the router. The router then services these queues

    in a sort of round-robin,

Queuing Disciplines

- Fair Queuing



Round-robin service of four flows at a router

Queuing Disciplines

- Fair Queuing
  - The main complication with Fair Queuing is that the packets being processed at a router are not necessarily the same length.
  - To truly allocate the bandwidth of the outgoing link in a fair manner, it is necessary to take packet length into consideration.
    - For example, if a router is managing two flows, one with 1000-byte packets and the other with 500-byte packets (perhaps because of fragmentation upstream from this router), 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

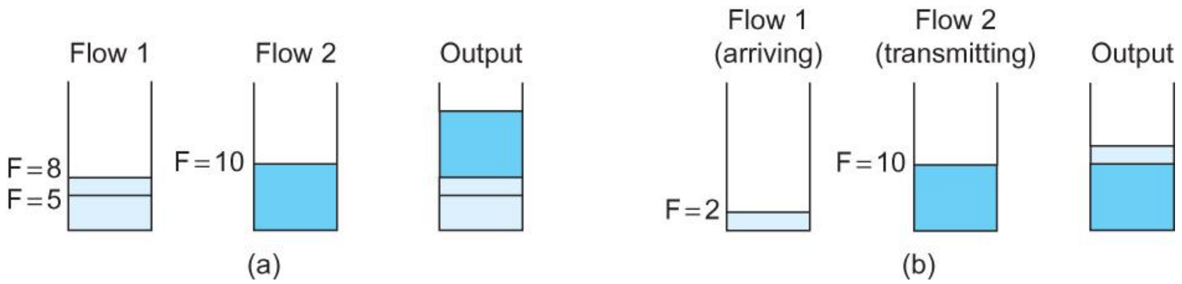
- Fair Queuing
  - What we really want is bit-by-bit round-robin; that is, the router transmits a bit from flow 1, then a bit from flow 2, and so on.
  - Clearly, it is not feasible to interleave the bits from different packets.
  - The FQ mechanism therefore simulates this behavior by first determining when a given packet would finish being transmitted if it were being sent using bit-by-bit round-robin, 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, $F_i = S_i + P_i$
  - 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 A_i denote the time that packet i arrives at the router
  - Then $S_i = \max(F_{i-1}, A_i)$
  - $F_i = \max(F_{i-1}, A_i) + P_i$

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
  - Next packet to transmit is always the packet that has the lowest timestamp
    - The packet that should finish transmission before all others



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

- TCP congestion control was introduced into the Internet in the late 1980s by Van Jacobson, roughly eight years after the TCP/IP protocol stack had become operational.
- Immediately preceding this time, the Internet was suffering from congestion collapse—
  - 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), and the hosts would time out and retransmit their packets, resulting in even more congestion

TCP Congestion Control

- The idea of TCP congestion control is for each source to determine how much capacity is available in the network, so that it knows how many packets it can safely have in transit.
  - Once a given source has this many packets in transit, it uses the arrival of an ACK as a signal that one of its packets has left the network, and that it is 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

- Additive Increase Multiplicative Decrease
  - TCP maintains a new state variable for each connection, called *CongestionWindow*, which is used by the source 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

TCP Congestion Control

- Additive Increase Multiplicative Decrease
  - TCP’s effective window is revised as follows:
    - MaxWindow = MIN(CongestionWindow, AdvertisedWindow)
    - EffectiveWindow = MaxWindow − (LastByteSent − LastByteAcked).
  - That is, MaxWindow replaces AdvertisedWindow in the calculation of EffectiveWindow.
  - 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

- Additive Increase Multiplicative Decrease
  - The problem, of course, is how TCP comes to learn an appropriate value for CongestionWindow.
  - Unlike the AdvertisedWindow, which is sent by the receiving side of the connection, there is no one to send a suitable CongestionWindow to the sending side of TCP.
    - The answer is that the TCP source sets the CongestionWindow based on the level of congestion it perceives to exist in the network.
    - This involves decreasing the congestion window when the level of congestion goes up and increasing the congestion window when the level of congestion goes down. Taken together, the mechanism is commonly called *additive increase/multiplicative decrease (AIMD)*

TCP Congestion Control

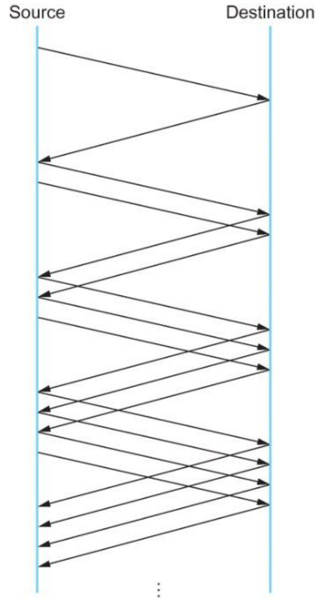
- Additive Increase Multiplicative Decrease
  - The key question, then, is how does the source determine that the network is congested and that it should decrease the congestion window?
    - The answer is based on the observation that the main reason packets are not delivered, and a timeout results, is that a packet was dropped due to congestion. It is rare that a packet is dropped because of an error during transmission.
    - Therefore, TCP interprets timeouts as a sign of congestion and reduces the rate at which it is transmitting.
    - Specifically, each time a timeout occurs, the source sets CongestionWindow to half of its previous value. This halving of the CongestionWindow for each timeout corresponds to the “multiplicative decrease” part of AIMD.

TCP Congestion Control

- Additive Increase Multiplicative Decrease
  - 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
  - A congestion-control strategy that only decreases the window size is obviously too conservative.
  - We also need to be able to increase the congestion window to take advantage of newly available capacity in the network.
  - This is the “additive increase” part of AIMD, and it works as follows.
    - Every time the source successfully sends a CongestionWindow’s worth of packets—that is, each packet sent out during the last RTT has been ACKed—it adds the equivalent of 1 packet to CongestionWindow.



Packets in transit during additive increase, with one packet being added each RTT.

TCP Congestion Control

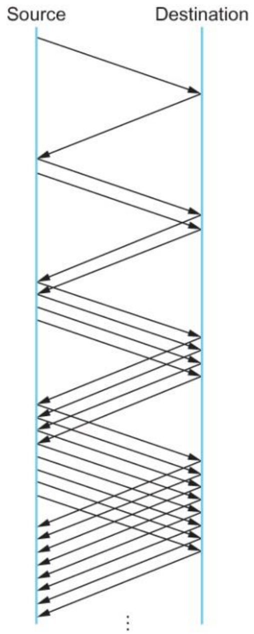
- Additive Increase Multiplicative Decrease
  - Note that in practice, TCP does not wait for an entire window's worth of ACKs to add 1 packet's worth to the congestion window, but instead increments CongestionWindow by a little for each ACK that arrives.
  - Specifically, the congestion window is incremented as follows each time an ACK arrives:
    - Increment = MSS × (MSS/CongestionWindow)
    - CongestionWindow+= Increment
    - That is, rather than incrementing CongestionWindow by an entire MSS bytes each RTT, we increment it by a fraction of MSS every time an ACK is received.
    - Assuming that each ACK acknowledges the receipt of MSS bytes, then that fraction is MSS/CongestionWindow.

TCP Congestion Control

- Slow Start
  - The additive increase mechanism just described 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.
  - 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 windowexponentially, rather than linearly.

TCP Congestion Control

- Slow Start
  - Specifically, the source starts out by setting CongestionWindow to one packet.
  - When the ACK for this packet arrives, TCP adds 1 to CongestionWindow and then sends two packets.
  - Upon receiving the corresponding two ACKs, TCP increments CongestionWindow by 2—one for each ACK—and next sends four packets.
  - The end result is that TCP effectively doubles the number of packets it has in transit every RTT.



Packets in transit during slow start.

TCP Congestion Control

- Slow Start
  - There are actually two different situations in which slow start runs.
    - The first is at the very beginning of a connection, at which time the source has no idea how many packets it is going to be able to have in transit at a given time.
      - In this situation, 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.
    - The second situation in which slow start is used is a bit more subtle; it occurs when the connection goes dead while waiting for a timeout to occur.
      - Recall how TCP's sliding window algorithm works—when a packet is lost, the source eventually reaches a point where it has sent as much data as the advertised window allows, and so it blocks while waiting for an ACK that will not arrive.
      - Eventually, a timeout happens, but by this time there are no packets in transit, meaning that the source will receive no ACKs to "clock" the transmission of new packets.
      - The source will instead receive a single cumulative ACK that reopens the entire advertised window, but as explained above, the source then uses slow start to restart the flow of data rather than dumping a whole window's worth of data on the network all at once.

TCP Congestion Control

- Slow Start
  - Although the source is using slow start again, it now knows more information than it did at the beginning of a connection.
  - Specifically, the source has a current (and useful) value of CongestionWindow; this is the value of CongestionWindow that existed prior to the last packet loss, divided by 2 as a result of the loss.
  - We can think of this as the “target” congestion window.
  - Slow start is used to rapidly increase the sending rate up to this value, and then additive increase is used beyond this point.

TCP Congestion Control

- Slow Start
  - Notice that we have a small bookkeeping problem to take care of, in that we want to remember the “target” congestion window resulting from multiplicative decrease as well as the “actual” congestion window being used by slow start.
  - To address this problem, TCP introduces a temporary variable to store the target window, typically called CongestionThreshold, that is set equal to the CongestionWindow value that results from multiplicative decrease.
  - The variable CongestionWindow is then reset to one packet, and it is incremented by one packet for every ACK that is received until it reaches.
  - CongestionThreshold, at which point it is incremented by one packet per RTT.

TCP Congestion Control

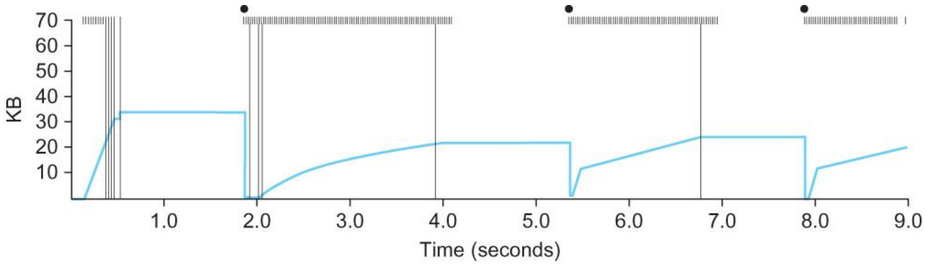
- Slow Start
  - In other words, TCP increases the congestion window as defined by the following code fragment:

```
{
        u_int cw = state->CongestionWindow;
        u_int incr = state->maxseg;
        if (cw > state->CongestionThreshold)
                incr = incr * incr / cw;
        state->CongestionWindow = MIN(cw + incr, TCP_MAXWIN);
}
```

    - where state represents the state of a particular TCP connection and TCP MAXWIN defines an upper bound on how large the congestion window is allowed to grow.

TCP Congestion Control

- Slow Start



Behavior of TCP congestion control. Colored line = value of CongestionWindow over time; solid bullets at top of graph = timeouts; hash marks at top of graph = time when each packet is transmitted; vertical bars = time when a packet that was eventually retransmitted was first transmitted.

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 the coarse-grained implementation of TCP timeouts led to long periods of time during which the connection went dead while waiting for a timer to expire.
  - Because of this, a new mechanism called *fast retransmit* was added to TCP.
  - Fast retransmit is a heuristic that sometimes triggers 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— that is, TCP cannot yet acknowledge the data the packet contains because earlier data has not yet arrived—TCP resends the same acknowledgment it sent the last time.

TCP Congestion Control

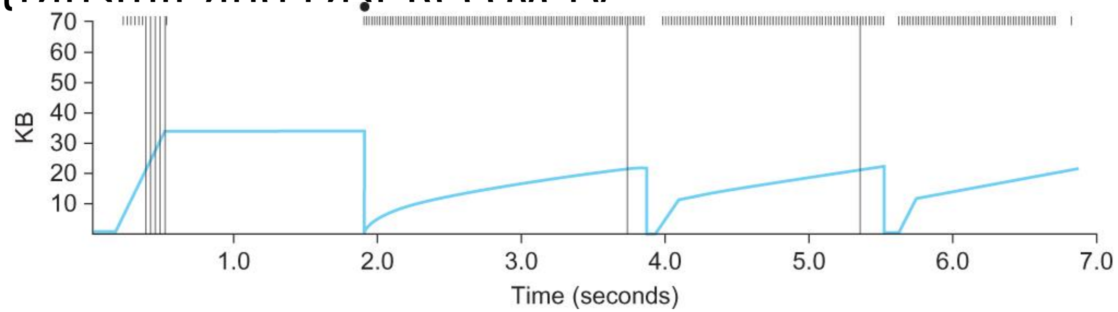
- Fast Retransmit and Fast Recovery
  - This second transmission of the same acknowledgment is called a *duplicate ACK.*
  - When the sending side sees a duplicate ACK, it knows that the other side must have received a packet out of order, which suggests that an earlier packet might have been lost.
  - 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.

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.
  - This mechanism, which is called *fast recovery, effectively* removes the slow start phase that happens between when fast retransmit detects a lost packet and additive increase begins.

TCP Congestion Control

- Fast Retransmit and Fast Recovery



Trace of TCP with fast retransmit. Colored line = CongestionWindow;
solid bullet = timeout; hash marks = time when each packet is transmitted;
vertical bars = time when a packet that was eventually retransmitted was first transmitted.

Congestion Avoidance Mechanism

- It is important to understand that TCP's strategy is to control congestion once it happens, as opposed to trying to avoid congestion in the first place.
- In fact, TCP repeatedly increases the load it imposes on the network in an effort to find the point at which congestion occurs, and then it backs off from this point.
- An appealing alternative, but one that has not yet been widely adopted, is to predict when congestion is about to happen and then to reduce the rate at which hosts send data just before packets start being discarded.
- We call such a strategy *congestion avoidance, to* distinguish it from *congestion control.*

Congestion Avoidance Mechanism

- DEC Bit
  - The first mechanismwas developed for use on the Digital Network Architecture (DNA), a connectionless network with a connection-oriented transport protocol.
  - This mechanism could, therefore, also be applied to TCP and IP.
  - As noted above, the idea here is to more evenly split the responsibility for congestion control between the routers and the end nodes.
  - 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 in the packets that flow through the router; hence the name DECbit.
  - 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

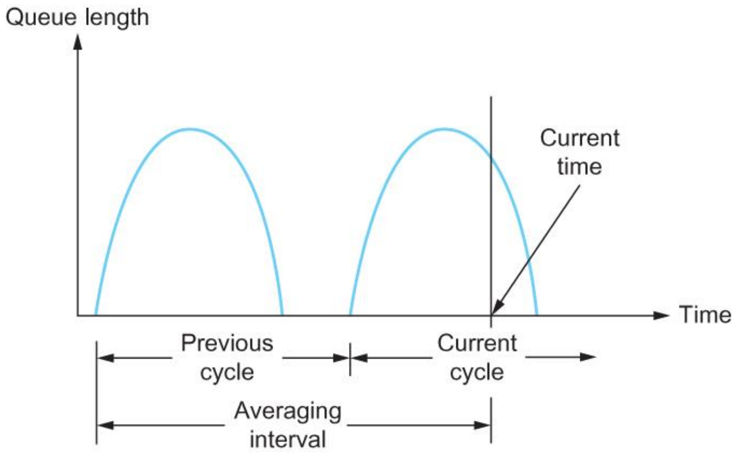
- DEC Bit
  - A single congestion bit is added to the packet header. A router sets this bit in a packet if its average queue length is greater than or equal to 1 at the time the packet arrives.
  - This average queue length is measured over a time interval that spans the last busy+idle cycle, plus the current busy cycle.
  - Essentially, the router calculates the area under the curve and divides this value by the time interval to compute the average queue length.
  - Using a queue length of 1 as the trigger for setting the congestion bit is a trade-off between significant queuing (and hence higher throughput) and increased idle time (and hence lower delay).
  - In other words, a queue length of 1 seems to optimize the power function.

Congestion Avoidance Mechanism

- DEC Bit
  - The source records how many of its packets resulted in some router setting the congestion bit.
  - In particular, the source maintains a congestion window, just as in TCP, and watches to see what fraction of the last window's worth of packets resulted in the bit being set.
  - If less than 50% of the packets had the bit set, then the source increases its congestion window by one packet.
  - 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.

Congestion Avoidance Mechanism

- DEC Bit
  - The value 50% was chosen as the threshold based on analysis that showed it to correspond to the peak of the power curve. The “increase by 1, decrease by 0.875” rule was selected because additive increase/multiplicative decrease makes the mechanism stable.



Computing average queue length at a router

Congestion Avoidance Mechanism

- Random Early Detection (RED)
  - A second mechanism, called *random early detection (RED),* is similar to the DECbit scheme in that each router is programmed to monitor its own queue length, and when it detects that congestion is imminent, to notify the source to adjust its congestion window. RED, invented by Sally Floyd and Van Jacobson in the early 1990s, differs from the DECbit scheme in two major ways:

Congestion Avoidance Mechanism

- Random Early Detection (RED)
  - The first is that rather than explicitly sending a congestion notification message to the source, RED is most commonly implemented such that it *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 some other means of detecting packet loss such as duplicate ACKs).

Congestion Avoidance Mechanism

- Random Early Detection (RED)
  - As the “early” part of the RED acronym suggests, the gateway 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.
  - In other words, the router drops a few packets before it has exhausted its buffer space completely, so as to cause the source to slow down, with the hope that this will mean it does not have to drop lots of packets later on.

Congestion Avoidance Mechanism

- Random Early Detection (RED)
  - The second difference between RED and DECbit is in the details of how RED decides when to drop a packet and what packet it decides to drop.
  - To understand the basic idea, consider a simple FIFO queue. Rather than wait for the queue to become completely full and then be forced to drop each arriving packet, we could decide to drop each arriving packet with some *drop probability* whenever the queue length exceeds some *drop level.*
  - This idea is *called early random drop. The RED algorithm defines the details of how to monitor the queue length and* when to drop a packet.

Congestion Avoidance Mechanism

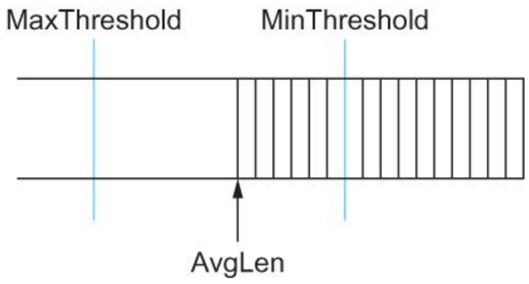
- Random Early Detection (RED)
  - First, RED computes an average queue length using a weighted running average similar to the one used in the original TCP timeout computation. That is, AvgLen is computed as
    - AvgLen = (1 − Weight) × AvgLen + Weight × SampleLen
    - where 0 < Weight < 1 and SampleLen is the length of the queue when a sample measurement is made.
  - In most software implementations, the queue length is measured every time a new packet arrives at the gateway.
  - In hardware, it might be calculated at some fixed sampling interval.

Congestion Avoidance Mechanism

- Random Early Detection (RED)
  - Second, RED has two queue length thresholds that trigger certain activity: MinThreshold and MaxThreshold.
  - When a packet arrives at the gateway, RED compares the current AvgLen with these two thresholds, according to the following rules:
    - if AvgLen $\leq$ MinThreshold
      - → queue the packet
    - if MinThreshold < AvgLen < MaxThreshold
      - → calculate probability P
      - → drop the arriving packet with probability P
    - if MaxThreshold $\leq$ AvgLen
      - → drop the arriving packet

Congestion Avoidance Mechanism

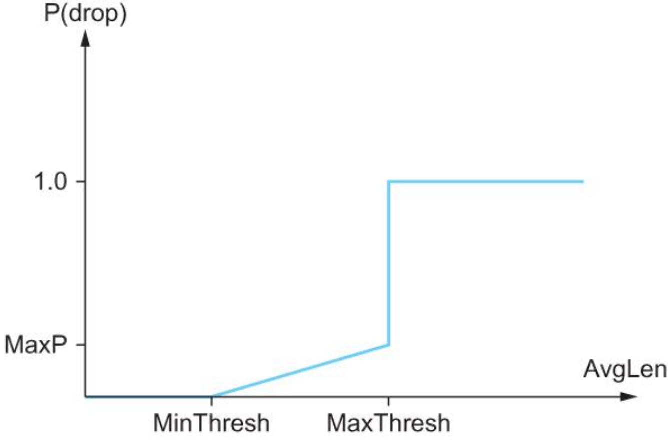
- Random Early Detection (RED)
  - P is a function of both AvgLen and how long it has been since the last packet was dropped.
  - Specifically, it is computed as follows:
    - TempP = MaxP × (AvgLen − MinThreshold)/(MaxThreshold − MinThreshold)
    - P = TempP/(1 − count × TempP)



RED thresholds on a FIFO queue

Congestion Avoidance Mechanism

- Random Early Detection (RED)



Drop probability function for RED

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.
  - 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.
    - If it is, then the algorithm decreases the congestion window by one-eighth.

Congestion Avoidance Mechanism

- Source-based Congestion Avoidance
  - A second algorithm does something similar. The decision as to whether or not to change the current window size is based on changes to both the RTT and the window size.
  - The window is adjusted once every two round-trip delays based on the product
    - (CurrentWindow − OldWindow)×(CurrentRTT − OldRTT)
    - If the result is positive, the source decreases the window size by one-eighth;
    - if the result is negative or 0, the source increases the window by one maximum packet size.
    - Note that the window changes during every adjustment; that is, it oscillates around its optimal point.

Quality of Service

- For many years, packet-switched networks have offered the promise of supporting multimedia applications, that is, those that combine audio, video, and data.
- After all, once digitized, audio and video information become like any other form of data—a stream of bits to be transmitted. One obstacle to the fulfillment of this promise has been the need for higher-bandwidth links.
- Recently, however, improvements in coding have reduced the bandwidth needs of audio and video applications, while at the same time link speeds have increased.

Quality of Service

- There is more to transmitting audio and video over a network than just providing sufficient bandwidth, however.
- Participants in a telephone conversation, for example, expect to be able to converse in such a way that one person can respond to something said by the other and be heard almost immediately.
- Thus, the timeliness of delivery can be very important. We refer to applications that are sensitive to the timeliness of data as *real-time applications.*
- Voice and video applications tend to be the canonical examples, but there are others such as industrial control—you would like a command sent to a robot arm to reach it before the arm crashes into something.
- Even file transfer applications can have timeliness constraints, such as a requirement that a database update complete overnight before the business that needs the data resumes on the next day.

Quality of Service

- The distinguishing characteristic of real-time applications is that they need some sort of assurance *from the network that data is likely to arrive on time (for some definition* of "on time").
- Whereas a non-real-time application can use an end-to-end retransmission strategy to make sure that data arrives *correctly, such a strategy cannot provide* timeliness.
- This implies that the network will treat some packets differently from others—something that is not done in the best-effort model.
- A network that can provide these different levels of service is often said to support quality of service (QoS).

Quality of Service

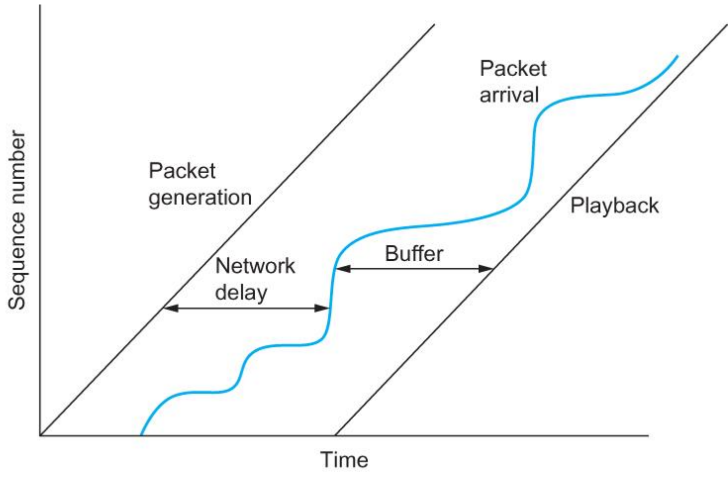
- Real-Time Applications
  - Data is generated by collecting samples from a microphone and digitizing them using an A →D converter
  - The digital samples are placed in packets which are transmitted across the network and received at the other end
  - At the receiving host the data must be played back at some appropriate rate

Quality of Service

- Real-Time Applications
  - For example, if voice samples were collected at a rate of one per 125 μs, they should be played back at the same rate
  - We can think of each sample as having a particular playback time
  - The point in time at which it is needed at the receiving host
  - In this example, each sample has a playback time that is 125 μs later than the preceding sample
  - If data arrives after its appropriate playback time, it is useless

Quality of Service

- Real-Time Applications
  - For some audio applications, there are limits to how far we can delay playing back data
  - It is hard to carry on a conversation if the time between when you speak and when your listener hears you is more than 300 ms
  - We want from the network a guarantee that all our data will arrive within 300 ms
  - If data arrives early, we buffer it until playback time



A playback buffer

- Taxonomy of Real-Time Applications
  - The first characteristic by which we can categorize applications is their tolerance of loss of data, where “loss” might occur because a packet arrived too late to be played back as well as arising from the usual causes in the network.
  - On the one hand, one lost audio sample can be interpolated from the surrounding samples with relatively little effect on the perceived audio quality. It is only as more and more samples are lost that quality declines to the point that the speech becomes incomprehensible.

- Taxonomy of Real-Time Applications
  - On the other hand, a robot control program is likely to be an example of a real-time application that cannot tolerate loss—losing the packet that contains the command instructing the robot arm to stop is unacceptable.
  - Thus, we can categorize real-time applications as *tolerant or intolerant* depending on whether they can tolerate occasional loss

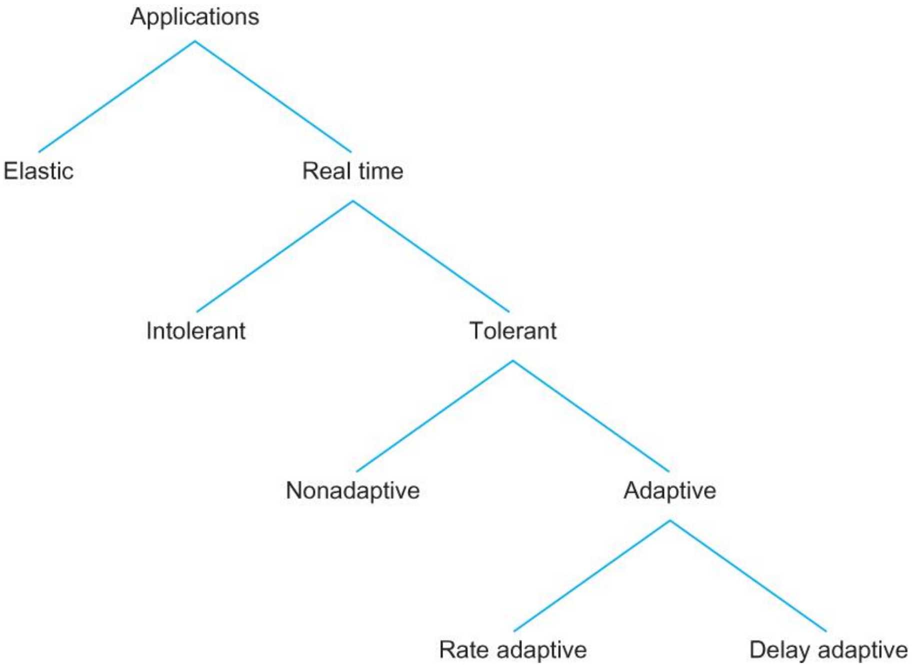
- Taxonomy of Real-Time Applications
  - A second way to characterize real-time applications is by their adaptability.
    - For example, an audio application might be able to adapt to the amount of delay that packets experience as they traverse the network.
      - If we notice that packets are almost always arriving within 300 ms of being sent, then we can set our playback point accordingly, buffering any packets that arrive in less than 300 ms.
      - Suppose that we subsequently observe that all packets are arriving within 100 ms of being sent.
      - If we moved up our playback point to 100 ms, then the users of the application would probably perceive an improvement. The process of shifting the playback point would actually require us to play out samples at an increased rate for some period of time.

Quality of Service

- Taxonomy of Real-Time Applications
  - We call applications that can adjust their playback point *delay-adaptive applications.*
  - Another class of adaptive applications are *rate adaptive. For example, many* video coding algorithms can trade off bit rate versus quality. Thus, if we find that the network can support a certain bandwidth, we can set our coding parameters accordingly.
  - If more bandwidth becomes available later, we can change parameters to increase the quality.

Quality of Service

- Taxonomy of Real-Time Applications



Quality of Service

- Approaches to QoS Support
  - *fine-grained approaches, which provide QoS to individual applications or flows*
  - *coarse-grained approaches, which provide QoS to large classes of data or aggregated* traffic

  - In the first category we find “Integrated Services,” a QoS architecture developed in the IETF and often associated with RSVP (Resource Reservation Protocol).
  - In the second category lies “Differentiated Services,” which is probably the most widely deployed QoS mechanism.

- Integrated Services (RSVP)
  - The term "Integrated Services" (often called IntServ for short) refers to a body of work that was produced by the IETF around 1995–97.
  - The IntServ working group developed specifications of a number of *service classes designed to meet the needs of some of the* application types described above.
  - It also defined how RSVP could be used to make reservations using these service classes.

Quality of Service

- Integrated Services (RSVP)
  - Service Classes
    - Guaranteed Service
      - The network should guarantee that the maximum delay that any packet will experience has some specified value
    - Controlled Load Service
      - The aim of the controlled load service is to emulate a lightly loaded network for those applications that request the service, even though the network as a whole may in fact be heavily loaded

Quality of Service

- Integrated Services (RSVP)
  - Overview of Mechanisms
    - Flowspec
      - With a best-effort service we can just tell the network where we want our packets to go and leave it at that, a real-time service involves telling the network something more about the type of service we require
      - The set of information that we provide to the network is referred to as a *flowspec.*
    - Admission Control
      - When we ask the network to provide us with a particular service, the network needs to decide if it can in fact provide that service. The process of deciding when to say no is called *admission control*.
    - Resource Reservation
      - We need a mechanism by which the users of the network and the components of the network itself exchange information such as requests for service, flowspecs, and admission control decisions. We refer to this process as *resource reservation*

Quality of Service

- Integrated Services (RSVP)
  - Overview of Mechanisms
    - Packet Scheduling
      - Finally, when flows and their requirements have been described, and admission control decisions have been made, the network switches and routers need to meet the requirements of the flows.
      - A key part of meeting these requirements is managing the way packets are queued and scheduled for transmission in the switches and routers.
      - This last mechanism is *packet scheduling.*

- Integrated Services (RSVP)
  - Flowspec
    - There are two separable parts to the flowspec:
      - The part that describes the flow's traffic characteristics (called the *TSpec)* and
      - The part that describes the service requested from the network (the *RSpec).*

      - The RSpec is very service specific and relatively easy to describe.
      - For example, with a controlled load service, the RSpec is trivial: The application just requests controlled load service with no additional parameters.
      - With a guaranteed service, you could specify a delay target or bound.

- Integrated Services (RSVP)
  - Flowspec
    - Tspec
      - We need to give the network enough information about the bandwidth used by the flow to allow intelligent admission control decisions to be made
      - For most applications, the bandwidth is not a single number
        - » It varies constantly
      - A video application will generate more bits per second when the scene is changing rapidly than when it is still
        - » Just knowing the long term average bandwidth is not enough

Quality of Service

- Integrated Services (RSVP)
  - Flowspec
    - Suppose 10 flows arrive at a switch on separate ports and they all leave on the same 10 Mbps link
    - If each flow is expected to send no more than 1 Mbps
      - No problem
    - If these are variable bit applications such as compressed video
      - They will occasionally send more than the average rate
    - If enough sources send more than average rates, then the total rate at which data arrives at the switch will be more than 10 Mbps
    - This excess data will be queued
    - The longer the condition persists, the longer the queue will get

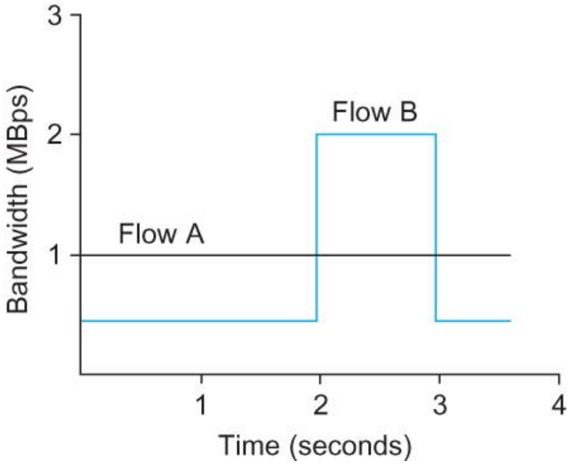
- Integrated Services (RSVP)
  - Flowspec
    - One way to describe the Bandwidth characteristics of sources is called a Token Bucket Filter
    - The filter is described by two parameters
      - A token rate *r*
      - A bucket depth *B*
    - To be able to send a byte, a token is needed
    - To send a packet of length *n*, *n* tokens are needed
    - Initially there are no tokens
    - Tokens are accumulated at a rate of *r* per second
    - No more than *B* tokens can be accumulated

- Integrated Services (RSVP)
  - Flowspec
    - We can send a burst of as many as *B* bytes into the network as fast as we want, but over significant long interval we cannot send more than *r* bytes per second
    - This information is important for admission control algorithm when it tries to find out whether it can accommodate new request for service

Quality of Service

– Flowspec

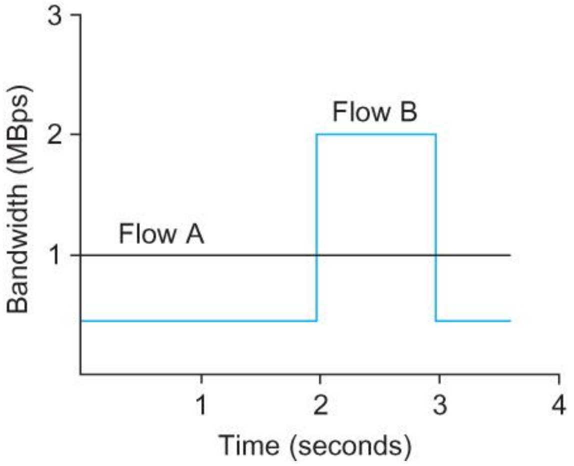
- The figure illustrates how a token bucket can be used to characterize a flow's Bandwidth requirement
- For simplicity, we assume each flow can send data as individual bytes rather than as packets
- Flow A generates data at a steady rate of 1 MBps
  - So it can be described by a token bucket filter with a rate *r* = 1 MBps and a bucket depth of 1 byte
  - This means that it receives tokens at a rate of 1 MBps but it cannot store more than 1 token, it spends them immediately



Quality of Service

– Flowspec

- Flow B sends at a rate that averages out to 1 MBps over the long term, but does so by sending at 0.5 MBps for 2 seconds and then at 2 MBps for 1 second
- Since the token bucket rate *r* is a long term average rate, flow B can be described by a token bucket with a rate of 1 MBps
- Unlike flow A, however flow B needs a bucket depth *B* of at least 1 MB, so that it can store up tokens while it sends at less than 1 MBps to be used when it sends at 2 MBps
- For the first 2 seconds, it receives tokens at a rate of 1 MBps but spends them at only 0.5 MBps,
  - So it can save up $2 \times 0.5 = 1$ MB of tokens which it spends at the 3rd second



- Integrated Services (RSVP)
  - Admission Control
    - The idea behind admission control is simple: When some new flow wants to receive a particular level of service, admission control looks at the TSpec and RSpec of the flow and tries to decide if the desired service can be provided to that amount of traffic, given the currently available resources, without causing any previously admitted flow to receive worse service than it had requested. If it can provide the service, the flow is admitted; if not, then it is denied.

Quality of Service

- Integrated Services (RSVP)
  - Reservation Protocol
    - While connection-oriented networks have always needed some sort of setup protocol to establish the necessary virtual circuit state in the switches, connectionless networks like the Internet have had no such protocols.
    - However we need to provide a lot more information to our network when we want a real-time service from it.
    - While there have been a number of setup protocols proposed for the Internet, the one on which most current attention is focused is called Resource Reservation Protocol (RSVP).

Quality of Service

- Integrated Services (RSVP)
  - Reservation Protocol
    - One of the key assumptions underlying RSVP is that it should not detract from the robustness that we find in today’s connectionless networks.
    - Because connectionless networks rely on little or no state being stored in the network itself, it is possible for routers to crash and reboot and for links to go up and down while end-to-end connectivity is still maintained.
    - RSVP tries to maintain this robustness by using the idea of *soft state in the routers.*
    - Another important characteristic of RSVP is that it aims to support multicast flows just as effectively as unicast flows
    - Initially, consider the case of one sender and one receiver trying to get a reservation for traffic flowing between them.

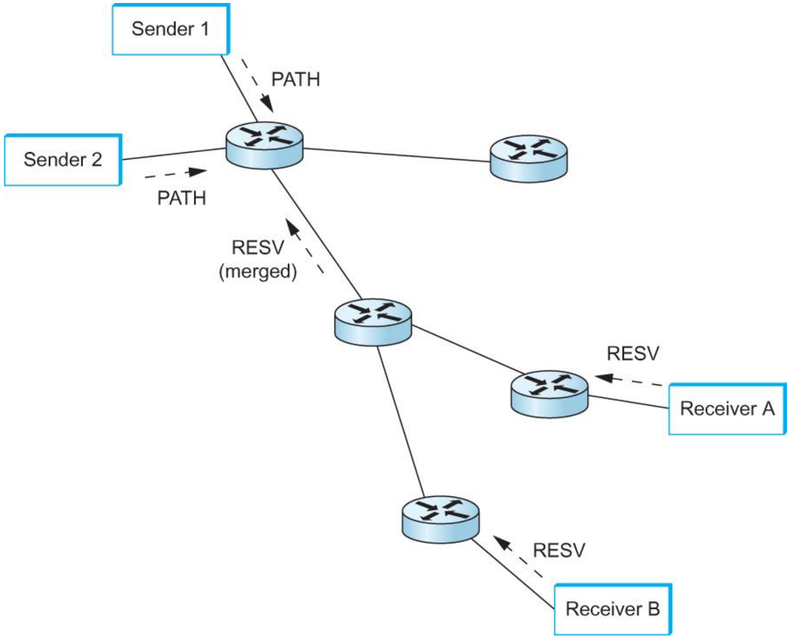
- Integrated Services (RSVP)
  - Reservation Protocol
    - There are two things that need to happen before a receiver can make the reservation.
    - First, the receiver needs to know what traffic the sender is likely to send so that it can make an appropriate reservation. That is, it needs to know the sender's TSpec.
    - Second, it needs to know what path the packets will follow from sender to receiver, so that it can establish a resource reservation at each router on the path. Both of these requirements can be met by sending a message from the sender to the receiver that contains the TSpec.
    - Obviously, this gets the TSpec to the receiver. The other thing that happens is that each router looks at this message (called a PATH message) as it goes past, and it figures out the *reverse path that will be used to send* reservations from the receiver back to the sender in an effort to get the reservation to each router on the path.

Quality of Service

- Integrated Services (RSVP)
  - Reservation Protocol
    - Having received a PATH message, the receiver sends a reservation back “up” the multicast tree in a RESV message.
    - This message contains the sender’s TSpec and an RSpec describing the requirements of this receiver.
    - Each router on the path looks at the reservation request and tries to allocate the necessary resources to satisfy it. If the reservation can be made, the RESV request is passed on to the next router.
    - If not, an error message is returned to the receiver who made the request. If all goes well, the correct reservation is installed at every router between the sender and the receiver.
    - As long as the receiver wants to retain the reservation, it sends the same RESV message about once every 30 seconds.

Quality of Service

- Integrated Services (RSVP)
  - Reservation Protocol



Making reservations on a multicast tree

Quality of Service

- Integrated Services (RSVP)
  - Packet Classifying and Scheduling
    - Once we have described our traffic and our desired network service and have installed a suitable reservation at all the routers on the path, the only thing that remains is for the routers to actually deliver the requested service to the data packets. There are two things that need to be done:
      - Associate each packet with the appropriate reservation so that it can be handled correctly, a process known as *classifying packets.*
      - Manage the packets in the queues so that they receive the service that has been requested, a process known as packet *scheduling.*

Quality of Service

- Differentiated Services
  - Whereas the Integrated Services architecture allocates resources to individual flows, the Differentiated Services model (often called DiffServ for short) allocates resources to a small number of classes of traffic.
  - In fact, some proposed approaches to DiffServ simply divide traffic into two classes.
  - Suppose that we have decided to enhance the best-effort service model by adding just one new class, which we'll call "premium."
  - Clearly we will need some way to figure out which packets are premium and which are regular old best effort.
  - Rather than using a protocol like RSVP to tell all the routers that some flow is sending premium packets, it would be much easier if the packets could just identify themselves to the router when they arrive. This could obviously be done by using a bit in the packet header—if that bit is a 1, the packet is a premium packet; if it's a 0, the packet is best effort

Quality of Service

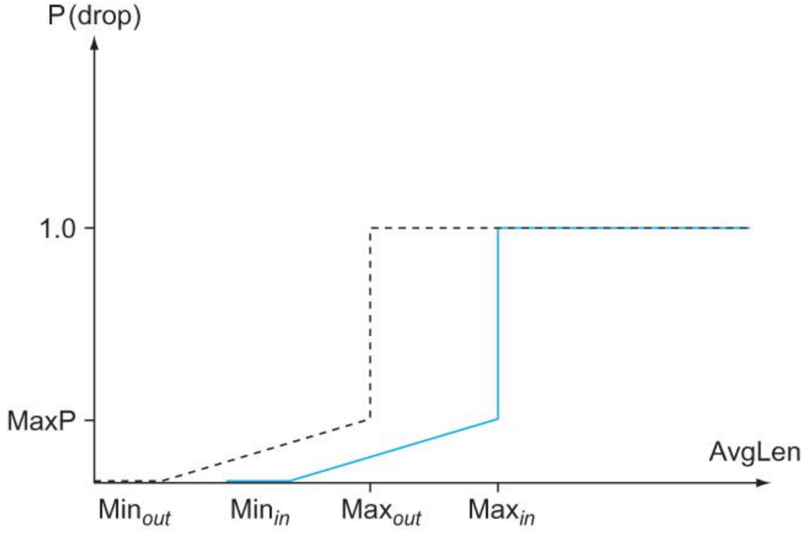
- Differentiated Services
  - With this in mind, there are two questions we need to address:
    - Who sets the premium bit, and under what circumstances?
    - What does a router do differently when it sees a packet with the bit set?
  - There are many possible answers to the first question, but a common approach is to set the bit at an administrative boundary.
  - For example, the router at the edge of an Internet service provider’s network might set the bit for packets arriving on an interface that connects to a particular company’s network.
  - The Internet service provider might do this because that company has paid for a higher level of service than best effort.

Quality of Service

- Differentiated Services
  - Assuming that packets have been marked in some way, what do the routers that encounter marked packets do with them?
  - Here again there are many answers. In fact, the IETF standardized a set of router behaviors to be applied to marked packets. These are called “per-hop behaviors” (PHBs), a term that indicates that they define the behavior of individual routers rather than end-to-end services
  - The Expedited Forwarding (EF) PHB
    - One of the simplest PHBs to explain is known as “expedited forwarding” (EF). Packets marked for EF treatment should be forwarded by the router with minimal delay and loss.
    - The only way that a router can guarantee this to all EF packets is if the arrival rate of EF packets at the router is strictly limited to be less than the rate at which the router can forward EF packets.

- Differentiated Services
  - The Assured Forwarding (AF) PHB
    - The “assured forwarding” (AF) PHB has its roots in an approach known as “RED with In and Out” (RIO) or “Weighted RED,” both of which are enhancements to the basic RED algorithm.
    - For our two classes of traffic, we have two separate drop probability curves. RIO calls the two classes “in” and “out” for reasons that will become clear shortly.
    - Because the “out” curve has a lower MinThreshold than the “in” curve, it is clear that, under low levels of congestion, only packets marked “out” will be discarded by the RED algorithm. If the congestion becomes more serious, a higher percentage of “out” packets are dropped, and then if the average queue length exceeds Min_{in}, RED starts to drop “in” packets as well.

- Differentiated Services
  - The Assured Forwarding (AF) PHB



RED with In and Out drop probabilities

Summary

- We have discussed different resource allocation mechanisms
- We have discussed different queuing mechanisms
- We have discussed TCP Congestion Control mechanisms
- We have discussed Congestion Avoidance mechanisms
- We have discussed Quality of Services