UNIT 3

Congestion Control Algorithms

Congestion refers to the state when network nodes or links become overloaded with data, leading to a degradation in performance and increased delays. Network congestion can occur due to high demand, insufficient bandwidth, or inefficient routing.

When a network becomes congested, several issues may arise:

1. **Increased Latency:** Congestion can cause delays in the transmission of data packets, resulting in increased latency or lag. This can impact real-time applications, such as video conferencing or online gaming, where timely delivery of data is crucial.
2. **Packet Loss:** In congested networks, packets may be dropped or lost due to limited buffer space or overwhelmed network devices. Packet loss can lead to data retransmissions, affecting the overall efficiency and reliability of network communication.
3. **Reduced Throughput:** Network congestion can decrease the available bandwidth for each user or device, resulting in reduced data transfer rates. This can impact tasks that require high data throughput, such as large file transfers or streaming media.
4. **Unfair Resource Allocation:** Congestion can lead to an unfair distribution of network resources among users or applications. Certain connections or services may dominate the available bandwidth, causing others to suffer from limited access and poor performance.

1. General Principles of Congestion Control

Many problems in complex systems, such as computer networks, can be viewed from a control theory point of view. This approach leads to dividing all solutions into two groups:
open loop and closed loop.

(i)Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Tools for doing open-loop control include deciding when to accept new traffic, deciding when to discard packets and which ones, and making scheduling decisions at various points in the network. All of these have in common the fact that they make decisions without regard to the current state of the network.

(ii) closed loop solutions are based on the concept of a feedback loop. This approach has three parts when applied to congestion control:

1. Monitor the system to detect when and where congestion occurs.
2. Pass this information to places where action can be taken.
3. Adjust system operation to correct the problem.

1. A variety of metrics can be used to monitor the subnet for congestion. Chief among these are the percentage of all packets discarded for lack of buffer space, the average queue lengths, the number of packets that time out and are retransmitted, the average packet delay, and the standard deviation of packet delay. In all cases, rising numbers indicate growing congestion.
2. The second step in the feedback loop is to transfer the information about the congestion from the point where it is detected to the point where something can be done about it. The obvious way is for the router detecting the congestion to send a packet to the traffic source or sources, announcing the problem. Of course, these extra packets increase the load at precisely the moment that more load is not needed, namely, when the subnet is congested.
3. Still another approach is to have hosts or routers periodically send probe packets out to explicitly ask about congestion. This information can then be used to route traffic around problem areas. Some radio stations have helicopters flying around their cities to report on road congestion to make it possible for their mobile listeners to route their packets (cars) around hot spots.

The open loop algorithms are further divided into ones that act at the source versus ones that act at the destination. The closed loop algorithms are also divided into two subcategories: explicit feedback versus implicit feedback.

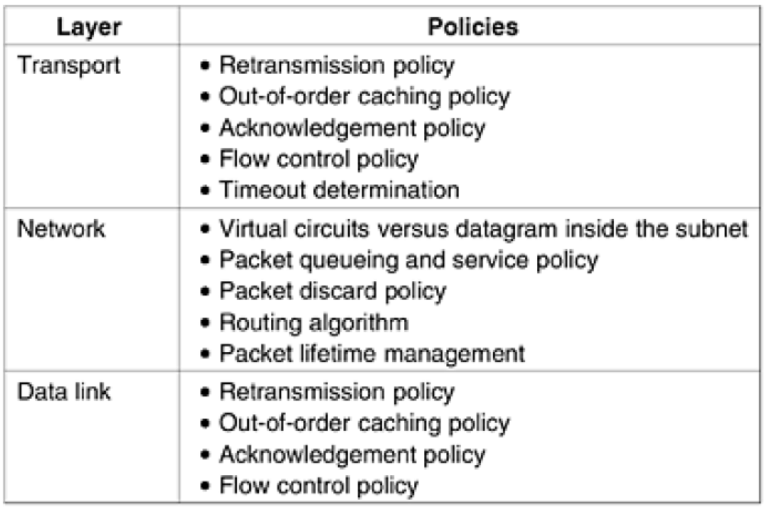
- In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source.
- In implicit algorithms, the source deduces the existence of congestion by making local observations, such as the time needed for acknowledgements to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. Two solutions come to mind: increase the resources or decrease the load. For example, the subnet may start using dial-up telephone lines to temporarily increase the bandwidth between certain points. On satellite systems, increasing transmission power often gives higher bandwidth. Splitting traffic over multiple routes instead of always using the best one may also effectively increase the bandwidth. Finally, spare routers that are normally used only as backups (to make the system fault tolerant) can be put on-line to give more capacity when serious congestion appears.
However, sometimes it is not possible to increase the capacity, or it has already been increased to the limit. The only way then to beat back the congestion is to decrease the load. Several ways exist to reduce the load, including denying service to some users, degrading service to some or all users, and having users schedule their demands in a more predictable way.

2. Congestion Prevention Policies

The open loop systems are designed to minimize congestion in the first place, rather than letting it happen and reacting after the fact. They try to achieve their goal by using appropriate policies at various levels.

Figure 5-26. Policies that affect congestion.



Layer	Policies
Transport	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy • Timeout determination
Network	• Virtual circuits versus datagram inside the subnet • Packet queueing and service policy • Packet discard policy • Routing algorithm • Packet lifetime management
Data link	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy

i) At the data link layer the retransmission policy is concerned with how fast a sender times out and what it transmits upon timeout. A jumpy sender that times out quickly and retransmits all outstanding packets using go back n will put a heavier load on the system than will a leisurely sender that uses selective repeat. Closely related to this is the buffering policy. If receivers routinely discard all out-of-order packets, these packets will have to be transmitted again later, creating extra load. With respect to congestion control, selective repeat is clearly better than go back n. Acknowledgement policy also affects congestion. If each packet is acknowledged immediately, the acknowledgement packets generate extra traffic. However, if acknowledgements are saved up to piggyback onto reverse traffic, extra timeouts and retransmissions may result. A tight flow control scheme (e.g., a small window) reduces the data rate and thus helps fight congestion.

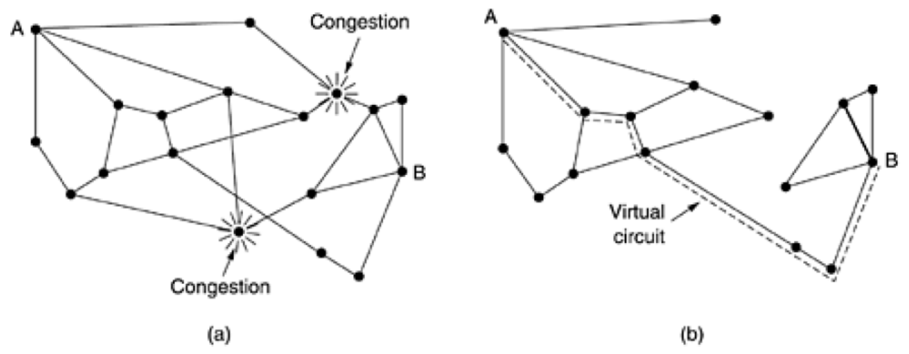
ii) At the network layer, the choice between using virtual circuits and using datagrams affects congestion since many congestion control algorithms work only with virtual-circuit subnets. Packet queueing and service policy relates to whether routers have one queue per input line, one queue per output line, or both. It also relates to the order in which packets are processed. Discard policy is the rule telling which packet is dropped when there is no space. A good policy can help alleviate congestion and a bad one can make it worse. A good routing algorithm can help avoid congestion by spreading the traffic over all the lines, whereas a bad one can send too much traffic over already congested lines. Finally, packet lifetime management deals with how long a packet may live before being discarded. If it is too long, lost packets may clog up the works for a long time, but if it is too short, packets may sometimes time out before reaching their destination, thus inducing retransmissions.

iii) In the transport layer, the same issues occur as in the data link layer, but in addition, determining the timeout interval is harder because the transit time across the network is less predictable than the transit time over a wire between two routers. If the timeout interval is too short, extra packets will be sent unnecessarily. If it is too long, congestion will be reduced but the response time will suffer whenever a packet is lost.

3. Congestion Control in Virtual-Circuit Subnets

One technique that is widely used to keep congestion that has already started from getting worse is admission control. The idea is simple: once congestion has been signaled, no more virtual circuits are set up until the problem has gone away. Thus, attempts to set up new transport layer connections fail. Letting more people in just makes matters worse. While this approach is crude, it is simple and easy to carry out. In the telephone system, when a switch gets overloaded, it also practices admission control by not giving dial tones. An alternative approach is to allow new virtual circuits but carefully route all new virtual circuits around problem areas.

Figure 5-27. (a) A congested subnet. (b) A redrawn subnet that eliminates the congestion. A virtual circuit from A to B is also shown.



Suppose that a host attached to router A wants to set up a connection to a host attached to router B. Normally, this connection would pass through one of the congested routers. To avoid this situation, we can redraw the subnet as shown in Fig. 5-27(b), omitting the congested routers and all of their lines. The dashed line shows a possible route for the virtual circuit that avoids the congested routers.

Another strategy relating to virtual circuits is to negotiate an agreement between the host and subnet when a virtual circuit is set up. This agreement normally specifies the volume and shape of the traffic, quality of service required, and other parameters. To keep its part of the agreement, the subnet will typically reserve resources along the path when the circuit is set up. These resources can include table and buffer space in the routers and bandwidth on the lines. In this way, congestion is unlikely to occur on the new virtual circuits because all the necessary resources are guaranteed to be available.

This kind of reservation can be done all the time as standard operating procedure or only when the subnet is congested. A disadvantage of doing it all the time is that it tends to waste resources. If six virtual circuits that might use 1 Mbps all pass through the same physical 6- Mbps line, the line has to be marked as full, even though it may rarely happen that all six virtual circuits are transmitting full blast at the same time. Consequently, the price of the congestion control is unused (i.e., wasted) bandwidth in the normal case.

4. Congestion Control in Datagram Subnets

Let us now turn to some approaches that can be used in datagram subnets (and also in virtual circuit subnets). Each router can easily monitor the utilization of its output lines and other resources. For example, it can associate with each line a real variable, u, whose value, between 0.0 and 1.0, reflects the recent utilization of that line. To maintain a good estimate of u, a sample of the instantaneous line utilization, f (either 0 or 1), can be made periodically and u updated according to

$$u_{\text{new}} = a u_{\text{old}} + (1 - a) f$$

where the constant ‘a’ determines how fast the router forgets recent history.
Whenever u moves above the threshold, the output line enters a "warning" state. Each newly arriving packet is checked to see if its output line is in warning state. If it is, some action is taken. The action taken can be one of several alternatives.

The Warning Bit

The old DECNET architecture signaled the warning state by setting a special bit in the packet's header. So does frame relay. When the packet arrived at its destination, the transport entity copied the bit into the next acknowledgement sent back to the source. The source then cut back on traffic.
As long as the router was in the warning state, it continued to set the warning bit, which meant that the source continued to get acknowledgements with it set. The source monitored the fraction of acknowledgements with the bit set and adjusted its transmission rate accordingly. As long as the warning bits continued to flow in, the source continued to decrease its transmission rate. When they slowed to a trickle, it increased its transmission rate. Note that since every router along the path could set the warning bit, traffic increased only when no router was in trouble.

Choke Packets

The previous congestion control algorithm is fairly subtle. It uses a roundabout means to tell the source to slow down. Why not just tell it directly? In this approach, the router sends a choke packet back to the source host, giving it the destination found in the packet. The original packet is tagged (a header bit is turned on) so that it will not generate any more choke packets farther along the path and is then forwarded in the usual way.

The purpose of choke packets is to provide feedback to the sender, notifying them of the network congestion and signalling them to reduce their data transmission rate.

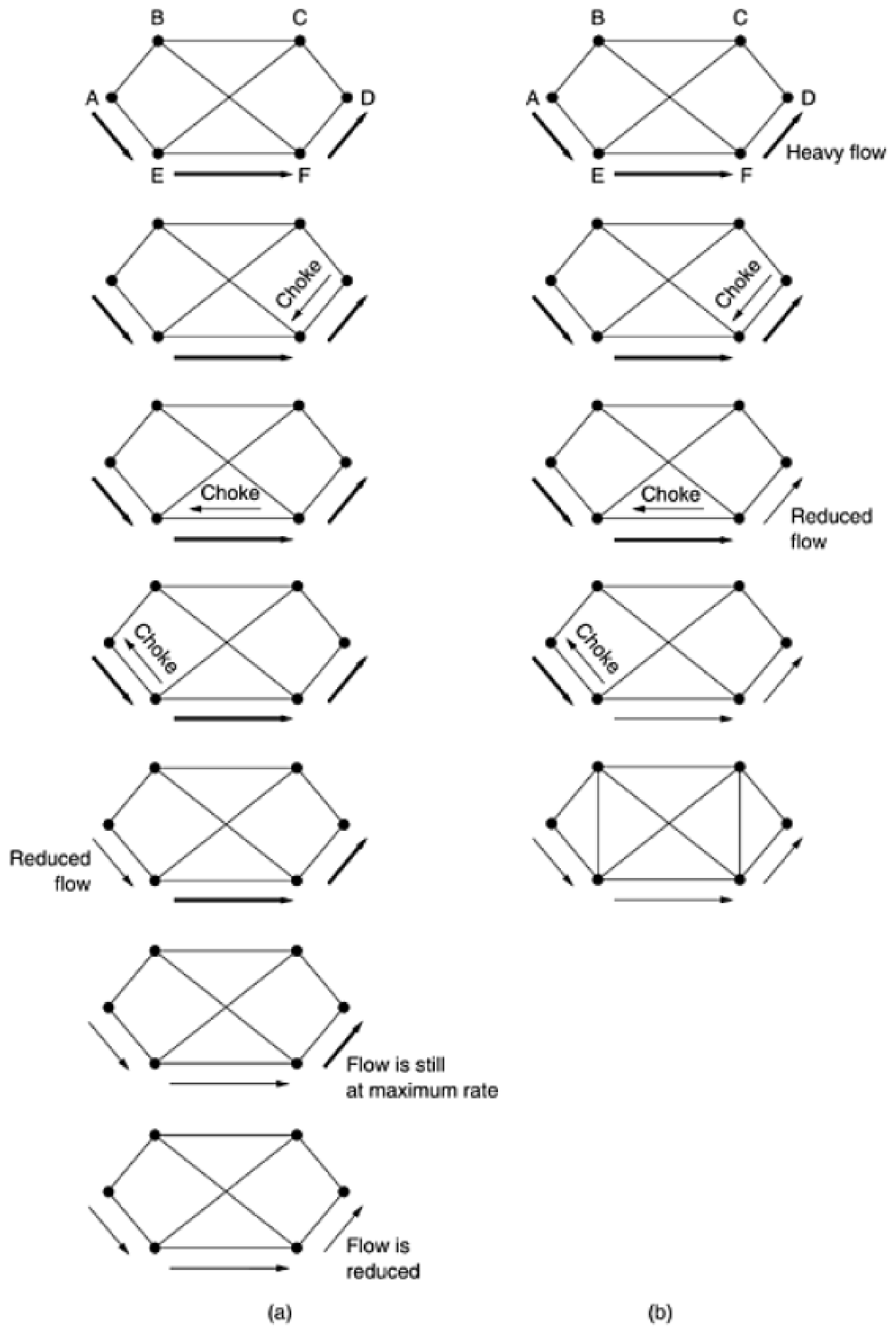
When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination by X percent. Since other packets aimed at the same destination are probably already under way and will generate yet more choke packets, the host should ignore choke packets referring to that destination for a fixed time interval. After that period has expired, the host listens for more choke packets for another interval. If one arrives, the line is still congested, so the host reduces the flow still more and begins ignoring choke packets again. If no choke packets arrive during the listening period, the host may increase the flow again. The feedback implicit in this protocol can help prevent congestion yet not throttle any flow unless trouble occurs.

Hosts can reduce traffic by adjusting their policy parameters, for example, their window size. Typically, the first choke packet causes the data rate to be reduced to 0.50 of its previous rate, the next one causes a reduction to 0.25, and so on. Increases are done in smaller increments to prevent congestion from reoccurring quickly.

Hop-by-Hop Choke Packets

At high speeds or over long distances, sending a choke packet to the source hosts does not work well because the reaction is so slow. Consider, for example, a host in San Francisco (router A in Fig. 5-28) that is sending traffic to a host in New York (router D in Fig. 5-28) at 155 Mbps. If the New York host begins to run out of buffers, it will take about 30 msec for a choke packet to get back to San Francisco to tell it to slow down. The choke packet propagation is shown as the second, third, and fourth steps in Fig. 5-28(a). In those 30 msec, another 4.6 megabits will have been sent. Even if the host in San Francisco completely shuts down immediately, the 4.6 megabits in the pipe will continue to pour in and have to be dealt with. Only in the seventh diagram in Fig. 5-28(a) will the New York router notice a slower flow.

Figure 5-28. (a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through.



An alternative approach is to have the choke packet take effect at every hop it passes through, as shown in the sequence of Fig. 5-28(b). Here, as soon as the choke packet reaches F, F is required to reduce the flow to D. Doing so will require F to devote more buffers to the flow, since the source is still sending away at full blast, but it gives D immediate relief. In the next step, the choke packet reaches E, which tells E to reduce the flow to F. This action puts a greater demand on E's buffers but gives F immediate relief. Finally, the choke packet reaches A and the flow genuinely slows down. The net effect of this hop-by-hop scheme is to provide quick relief at the point of congestion at the price of using up more buffers upstream. In this way, congestion can be nipped in the bud without losing any packets.

5. Load Shedding, Jitter Control

Load Shedding

When none of the above methods make the congestion disappear, routers can bring out the heavy artillery: load shedding. Load shedding is a fancy way of saying that when routers are being flooded by packets that they cannot handle, they just throw them away.

A router drowning in packets can just pick packets at random to drop, but usually it can do better than that. Which packet to discard may depend on the applications running. For file transfer, an old packet is worth more than a new one because dropping packet 6 and keeping packets 7 through 10 will cause a gap at the receiver that may force packets 6 through 10 to be retransmitted (if the receiver routinely discards out-of-order packets). In a 12-packet file, dropping 6 may require 7 through 12 to be retransmitted, whereas dropping 10 may require only 10 through 12 to be retransmitted. In contrast, for multimedia, a new packet is more important than an old one. The former policy (old is better than new) is often called wine and the latter (new is better than old) is often called milk.

A step above this in intelligence requires cooperation from the senders. For many applications, some packets are more important than others. For example, certain algorithms for compressing video periodically transmit an entire frame and then send subsequent frames as differences from the last full frame. In this case, dropping a packet that is part of a difference is preferable to dropping one that is part of a full frame. As another example, consider transmitting a document containing ASCII text and pictures. Losing a line of pixels in some image is far less damaging than losing a line of readable text.

To implement an intelligent discard policy, applications must mark their packets in priority classes to indicate how important they are. If they do this, then when packets have to be discarded, routers can first drop packets from the lowest class, then the next lowest class, and so on. Of course, unless there is some significant incentive to mark packets as anything other than VERY IMPORTANT— NEVER, EVER DISCARD, nobody will do it.

The incentive might be in the form of money, with the low-priority packets being cheaper to send than the high-priority ones. Alternatively, senders might be allowed to send high-priority packets under conditions of light load, but as the load increased they would be discarded, thus encouraging the users to stop sending them.

Another option is to allow hosts to exceed the limits specified in the agreement negotiated when the virtual circuit was set up (e.g., use a higher bandwidth than allowed), but subject to the condition that all excess traffic be marked as low priority. Such a strategy is actually not a bad idea, because it makes more efficient use of idle resources, allowing hosts to use them as long as nobody else is interested, but without establishing a right to them when times get tough.

Random Early Detection

It is well known that dealing with congestion after it is first detected is more effective than letting it gum up the works and then trying to deal with it. This observation leads to the idea of discarding packets before all the buffer space is really exhausted. A popular algorithm for doing this is called RED (Random Early Detection). In some transport protocols (including TCP), the response to lost packets is for the source to slow down. The reasoning behind this logic is that TCP was designed for wired networks and wired networks are very reliable, so lost packets are mostly due to buffer overruns rather than transmission errors. This fact can be exploited to help reduce congestion.

By having routers drop packets before the situation has become hopeless (hence the "early" in the name), the idea is that there is time for action to be taken before it is too late. To determine when to start discarding, routers maintain a running average of their queue lengths.

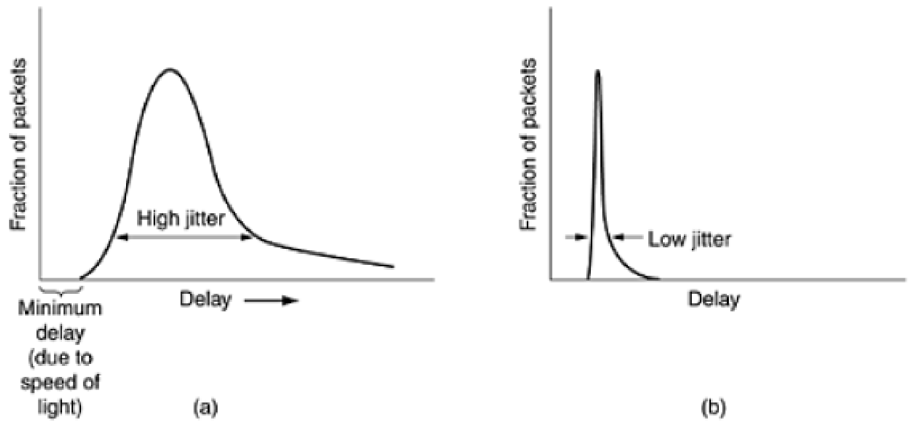
When the average queue length on some line exceeds a threshold, the line is said to be congested and action is taken. Since the router probably cannot tell which source is causing most of the trouble, picking a packet at random from the queue that triggered the action is probably as good as it can do.

How should the router tell the source about the problem? One way is to send it a choke packet. A problem with that approach is that it puts even more load on the already congested network. A different strategy is to just discard the selected packet and not report it. The source will eventually notice the lack of acknowledgement and take action. Since it knows that lost packets are generally caused by congestion and discards, it will respond by slowing down instead of trying harder. This implicit form of feedback only works when sources respond to lost packets by slowing down their transmission rate. In wireless networks, where most losses are due to noise on the air link, this approach cannot be used.

Jitter Control

For applications such as audio and video streaming, it does not matter much if the packets take 20 msec or 30 msec to be delivered, as long as the transit time is constant. The variation (i.e., standard deviation) in the packet arrival times is called jitter. High jitter, for example, having some packets taking 20 msec and others taking 30 msec to arrive will give an uneven quality to the sound or movie. Jitter is illustrated in Fig. 5-29. In contrast, an agreement that 99 percent of the packets be delivered with a delay in the range of 24.5 msec to 25.5 msec might be acceptable.

Figure 5-29. (a) High jitter. (b) Low jitter.



The range chosen must be feasible, of course. It must take into account the speed-of-light transit time and the minimum delay through the routers and perhaps leave a little slack for some inevitable delays.

The jitter can be bounded by computing the expected transit time for each hop along the path. When a packet arrives at a router, the router checks to see how much the packet is behind or ahead of its schedule. This information is stored in the packet and updated at each hop. If the packet is ahead of schedule, it is held just long enough to get it back on schedule. If it is behind schedule, the router tries to get it out the door quickly.

In fact, the algorithm for determining which of several packets competing for an output line should go next can always choose the packet furthest behind in its schedule. In this way, packets that are ahead of schedule get slowed down and packets that are behind schedule get speeded up, in both cases reducing the amount of jitter.

In some applications, such as video on demand, jitter can be eliminated by buffering at the receiver and then fetching data for display from the buffer instead of from the network in real time. However, for other applications, especially those that require real-time interaction between people such as Internet telephony and videoconferencing, the delay inherent in buffering is not acceptable.

6. Quality Of Service: Requirements

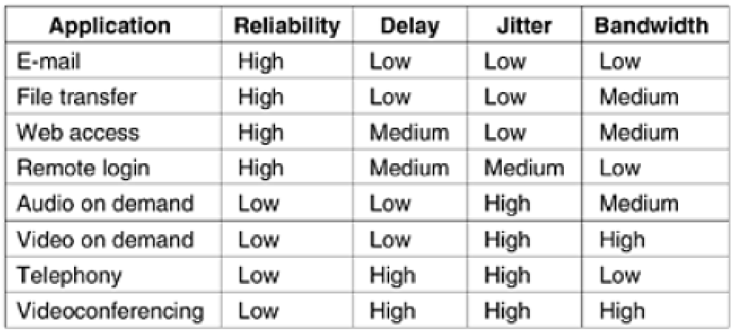
The techniques we looked at in the previous sections are designed to reduce congestion and improve network performance. However, with the growth of multimedia networking, often these ad hoc measures are not enough. Serious attempts at guaranteeing quality of service through network and protocol design are needed.0 A stream of packets from a source to a destination is called a flow. In a connection-oriented network, all the packets belonging to a flow follow the same route; in a connectionless network, they may follow different routes. The needs of each flow can be characterized by four primary parameters: reliability, delay, jitter, and bandwidth. Together these determine the QoS (Quality of Service) the flow requires.

Key Differences Between Jitter and Latency

1. The delay produced in the departure and arrival of the IP packet from the source to destination is known as latency. Conversely, jitter it the variation in the delay produced by the packets transmission.
2. Congestion in the network can cause jitter while latency can be produced through propagation delay, switching, routing and buffering.
3. The jitter can be prevented by using timestamps. In contrast, the latency can be reduced by using multiple connections to the internet.

Several common applications and the stringency of their requirements are listed in Fig. 5-30.

Figure 5-30. How stringent the quality-of-service requirements are.



Application	Reliability	Delay	Jitter	Bandwidth
E-mail	High	Low	Low	Low
File transfer	High	Low	Low	Medium
Web access	High	Medium	Low	Medium
Remote login	High	Medium	Medium	Low
Audio on demand	Low	Low	High	Medium
Video on demand	Low	Low	High	High
Telephony	Low	High	High	Low
Videoconferencing	Low	High	High	High

The first four applications have stringent requirements on reliability. No bits may be delivered incorrectly. This goal is usually achieved by check summing each packet and verifying the checksum at the destination. If a packet is damaged in transit, it is not acknowledged and will be retransmitted eventually. This strategy gives high reliability. The four final (audio/video) applications can tolerate errors, so no checksums are computed or verified.

File transfer applications, including e-mail and video, are not delay sensitive. If all packets are delayed uniformly by a few seconds, no harm is done. Interactive applications, such as Web surfing and remote login, are more delay sensitive. Real-time applications, such as telephony and videoconferencing have strict delay requirements. If all the words in a telephone call are each delayed by exactly 2.000 seconds, the users will find the connection unacceptable. On the other hand, playing audio or video files from a server does not require low delay.

The first three applications are not sensitive to the packets arriving with irregular time intervals between them. Remote login is somewhat sensitive to that, since characters on the screen will appear in little bursts if the connection suffers much jitter. Video and especially audio are extremely sensitive to jitter. If a user is watching a video over the network and the frames are all delayed by exactly 2.000 seconds, no harm is done. But if the transmission time varies randomly between 1 and 2 seconds, the result will be terrible. For audio, a jitter of even a few milliseconds is clearly audible. Finally, the applications differ in their bandwidth needs, with e-mail and remote login not needing much, but video in all forms needing a great deal. ATM networks classify flows in four broad categories with respect to their QoS demands as follows:

1. Constant bit rate (e.g., telephony).
2. Real-time variable bit rate (e.g., compressed videoconferencing).
3. Non-real-time variable bit rate (e.g., watching a movie over the Internet).
4. Available bit rate (e.g., file transfer).

These categories are also useful for other purposes and other networks. Constant bit rate is an attempt to simulate a wire by providing a uniform bandwidth and a uniform delay. Variable bit rate occurs when video is compressed, some frames compressing more than others. Thus, sending a frame with a lot of detail in it may require sending many bits whereas sending a shot of a white wall may compress extremely well. Available bit rate is for applications, such as email,that are not sensitive to delay or jitter.

7. Techniques for Achieving Good Quality of Service

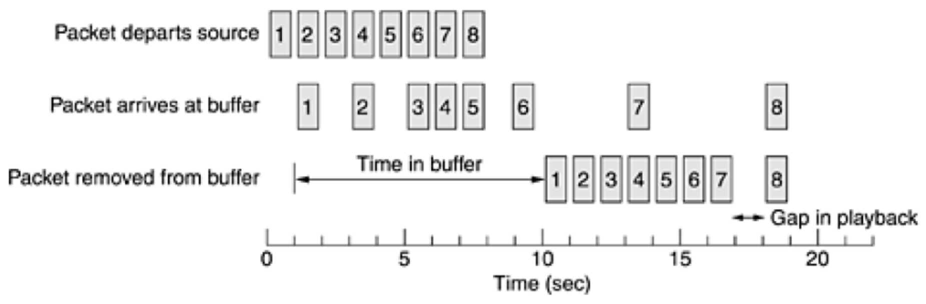
1.Overprovisioning

An easy solution is to provide so much router capacity, buffer space, and bandwidth that the packets just fly through easily. The trouble with this solution is that it is expensive. As time goes on and designers have a better idea of how much is enough, this technique may even become practical. To some extent, the telephone system is overprovisioned. It is rare to pick up a telephone and not get a dial tone instantly. There is simply so much capacity available there that demand can always be met.

2.Buffering

Flows can be buffered on the receiving side before being delivered. Buffering them does not affect the reliability or bandwidth, and increases the delay, but it smooths out the jitter. For audio and video on demand, jitter is the main problem, so this technique helps a lot. In Fig. 5-31 we see a stream of packets being delivered with substantial jitter. Packet 1 is sent from the server at t = 0 sec and arrives at the client at t = 1 sec. Packet 2 undergoes more delay and takes 2 sec to arrive. As the packets arrive, they are buffered on the client machine.

Figure 5-31. Smoothing the output stream by buffering packets.



At t = 10 sec, playback begins. At this time, packets 1 through 6 have been buffered so that they can be removed from the buffer at uniform intervals for smooth play. Unfortunately, packet 8 has been delayed so much that it is not available when its play slot comes up, so playback must stop until it arrives, creating an annoying gap in the music or movie. This problem can be alleviated by delaying the starting time even more, although doing so also requires a larger buffer. Commercial Web sites that contain streaming audio or video all use players that buffer for about 10 seconds before starting to play.

3.Traffic Shaping

In the above example, the source outputs the packets with a uniform spacing between them, but in other cases, they may be emitted irregularly, which may cause congestion to occur in the network. Nonuniform output is common if the server is handling many streams at once, and it also allows other actions, such as fast forward and rewind, user authentication, and so on. Also, the approach we used here (buffering) is not always possible, for example, with videoconferencing. However, if something could be done to make the server (and hosts in general) transmit at a uniform rate, quality of service would be better.

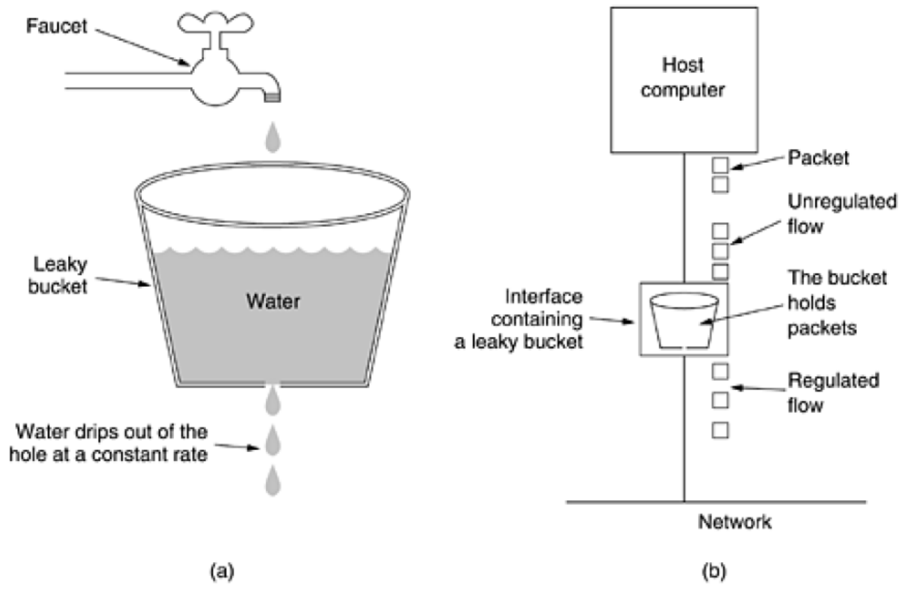
Traffic shaping is about regulating the average rate (and burstiness) of data transmission. It smooths out the traffic on the server side, rather than on the client side. In contrast, the sliding window protocols limit the amount of data in transit at once, not the rate at which it is sent. When a connection is set up, the user and the subnet (i.e., the customer and the carrier) agree on a certain traffic pattern (i.e., shape) for that circuit. Sometimes this is called a service level agreement. As long as the customer fulfils her part of the bargain and only sends packets according to the agreed-on contract, the carrier promises to deliver them all in a timely fashion. Traffic shaping reduces congestion and thus helps the carrier live up to its promise. Such agreements are not so important for file transfers but are of great importance for real-time data, such as audio and video connections, which have stringent quality-of-service requirements.

In effect, with traffic shaping the customer says to the carrier: My transmission pattern will look like this; can you handle it? If the carrier agrees, the issue arises of how the carrier can tell if the customer is following the agreement and what to do if the customer is not. Monitoring a traffic flow is called traffic policing. Agreeing to a traffic shape and policing it afterward are easier with virtual-circuit subnets than with datagram subnets. However, even with datagram subnets, the same ideas can be applied to transport layer connections.

4. The Leaky Bucket Algorithm

Imagine a bucket with a small hole in the bottom, as illustrated in Fig. 5-32(a). No matter the rate at which water enters the bucket, the outflow is at a constant rate, ρ, when there is any water in the bucket and zero when the bucket is empty. Also, once the bucket is full, any additional water entering it spills over the sides and is lost (i.e., does not appear in the output stream under the hole).

Figure 5-32. (a) A leaky bucket with water. (b) A leaky bucket with packets.



The same idea can be applied to packets, as shown in Fig. 5-32(b). Conceptually, each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more processes within the host try to send a packet when the maximum number is already queued, the new packet is abruptly discarded. This arrangement can be built into the hardware interface or simulated by the host operating system. It was first proposed by Turner (1986) and is called the leaky bucket algorithm. In fact, it is nothing other than a single-server queueing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. Again, this can be enforced by the interface card or by the operating system. This mechanism turns an uneven flow of packets from the user processes inside the host into an even flow of packets onto the network, smoothing out bursts and greatly reducing the chances of congestion.

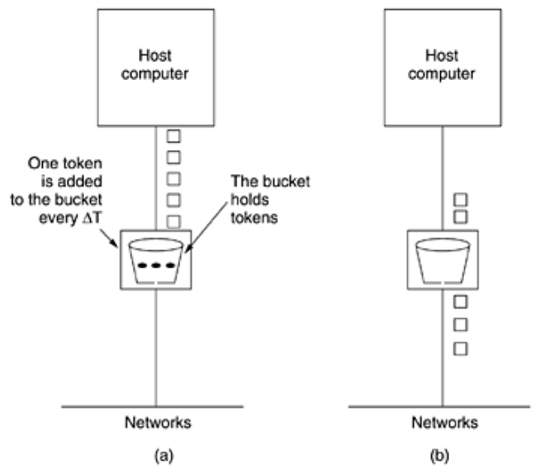
When the packets are all the same size (e.g., ATM cells), this algorithm can be used as described. However, when variable-sized packets are being used, it is often better to allow a fixed number of bytes per tick, rather than just one packet. Thus, if the rule is 1024 bytes per tick, a single 1024-byte packet can be admitted on a tick, two 512-byte packets, four 256-byte packets, and so on. If the residual byte count is too low, the next packet must wait until the next tick.

Implementing the original leaky bucket algorithm is easy. The leaky bucket consists of a finite queue. When a packet arrives, if there is room on the queue it is appended to the queue; otherwise, it is discarded. At every clock tick, one packet is transmitted (unless the queue is empty). The byte-counting leaky bucket is implemented almost the same way. At each tick, a counter is initialized to n. If the first packet on the queue has fewer bytes than the current value of the counter, it is transmitted, and the counter is decremented by that number of bytes. Additional packets may also be sent, as long as the counter is high enough. When the counter drops below the length of the next packet on the queue, transmission stops until the next tick, at which time the residual byte count is reset and the flow can continue.

5.The Token Bucket Algorithm

The leaky bucket algorithm enforces a rigid output pattern at the average rate, no matter how bursty the traffic is. For many applications, it is better to allow the output to speed up somewhat when large bursts arrive, so a more flexible algorithm is needed, preferably one that never loses data. One such algorithm is the token bucket algorithm. In this algorithm, the leaky bucket holds tokens, generated by a clock at the rate of one token every ΔT sec. In Fig.5-34(a) we see a bucket holding three tokens, with five packets waiting to be transmitted. For a packet to be transmitted, it must capture and destroy one token. In Fig. 5-34(b) we see that three of the five packets have gotten through, but the other two are stuck waiting for two more tokens to be generated.

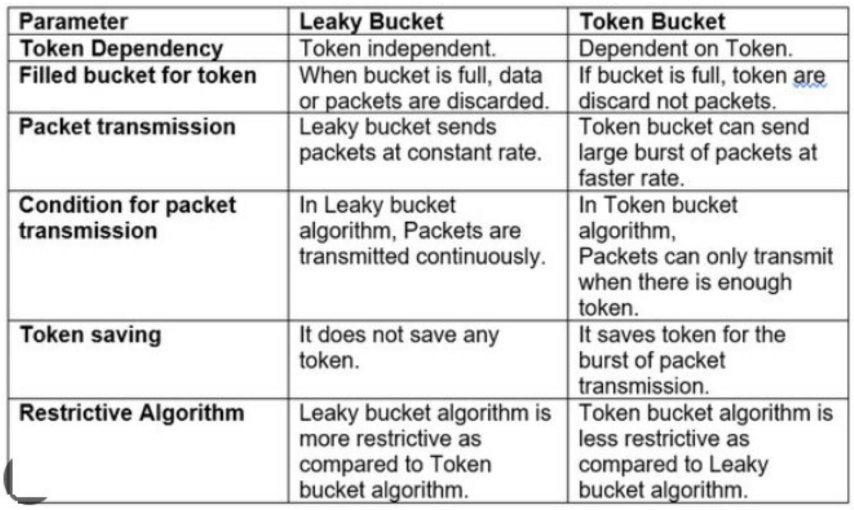
Figure 5-34. The token bucket algorithm. (a) Before. (b) After.



The token bucket algorithm provides a different kind of traffic shaping than that of the leaky bucket algorithm. The leaky bucket algorithm does not allow idle hosts to save up permission to send large bursts later. The token bucket algorithm does allow saving, up to the maximum size of the bucket, n. This property means that bursts of up to n packets can be sent at once, allowing some burstiness in the output stream and giving faster response to sudden bursts of input.

Another difference between the two algorithms is that the token bucket algorithm throws away tokens (i.e., transmission capacity) when the bucket fills up but never discards packets. In contrast, the leaky bucket algorithm discards packets when the bucket fills up. Here, too, a minor variant is possible, in which each token represents the right to send not one packet, but k bytes. A packet can only be transmitted if enough tokens are available to cover its length in bytes. Fractional tokens are kept for future use.

The leaky bucket and token bucket algorithms can also be used to smooth traffic between routers, as well as to regulate host output as in our examples. However, one clear difference is that a token bucket regulating a host can make the host stop sending when the rules say it must. Telling a router to stop sending while its input keeps pouring in may result in lost data. The implementation of the basic token bucket algorithm is just a variable that counts tokens. The counter is incremented by one every ΔT and decremented by one whenever a packet is sent. When the counter hits zero, no packets may be sent. In the byte-count variant, the counter is incremented by k bytes every ΔT and decremented by the length of each packet sent.



Parameter	Leaky Bucket	Token Bucket
Token Dependency	Token independent.	Dependent on Token.
Filled bucket for token	When bucket is full, data or packets are discarded.	If bucket is full, token are discard not packets.
Packet transmission	Leaky bucket sends packets at constant rate.	Token bucket can send large burst of packets at faster rate.
Condition for packet transmission	In Leaky bucket algorithm, Packets are transmitted continuously.	In Token bucket algorithm, Packets can only transmit when there is enough token.
Token saving	It does not save any token.	It saves token for the burst of packet transmission.
Restrictive Algorithm	Leaky bucket algorithm is more restrictive as compared to Token bucket algorithm.	Token bucket algorithm is less restrictive as compared to Leaky bucket algorithm.

6.Resource Reservation

Being able to regulate the shape of the offered traffic is a good start to guaranteeing the quality of service. However, effectively using this information implicitly means requiring all the packets of a flow to follow the same route. Spraying them over routers at random makes it hard to guarantee anything. As a consequence, something similar to a virtual circuit has to be set up from the source to the destination, and all the packets that belong to the flow must follow this route.

Once we have a specific route for a flow, it becomes possible to reserve resources along that route to make sure the needed capacity is available. Three different kinds of resources can potentially be reserved:

1. Bandwidth.
2. Buffer space.
3. CPU cycles.

,The first one, bandwidth, is the most obvious. If a flow requires 1 Mbps and the outgoing line has a capacity of 2 Mbps, trying to direct three flows through that line is not going to work. Thus, reserving bandwidth means not oversubscribing any output line.

A second resource that is often in short supply is buffer space. When a packet arrives, it is usually deposited on the network interface card by the hardware itself. The router software then has to copy it to a buffer in RAM and queue that buffer for transmission on the chosen outgoing line. If no buffer is available, the packet has to be discarded since there is no place to put it. For a good quality of service, some buffers can be reserved for a specific flow so that flow does not have to compete for buffers with other flows. There will always be a buffer available when the flow needs one, up to some maximum.

Finally, CPU cycles are also a scarce resource. It takes router CPU time to process a packet, so a router can process only a certain number of packets per second. Making sure that the CPU is not overloaded is needed to ensure timely processing of each packet.

At first glance, it might appear that if it takes, say, 1 µsec to process a packet, a router can process 1 million packets/sec. This observation is not true because there will always be idle periods due to statistical fluctuations in the load. If the CPU needs every single cycle to get its work done, losing even a few cycles due to occasional idleness creates a backlog it can never get rid of.

7.Admission Control

Now we are at the point where the incoming traffic from some flow is well shaped and can potentially follow a single route in which capacity can be reserved in advance on the routers along the path. When such a flow is offered to a router, it has to decide, based on its capacity and how many commitments it has already made for other flows, whether to admit or reject the flow. The decision to accept or reject a flow is not a simple matter of comparing the (bandwidth, buffers, cycles) requested by the flow with the router's excess capacity in those three dimensions. It is a little more complicated than that.

To start with, although some applications may know about their bandwidth requirements, few know about buffers or CPU cycles, so at the minimum, a different way is needed to describe flows. Next, some applications are far more tolerant of an occasional missed deadline than others. Finally, some applications may be willing to haggle about the flow parameters and others may not. For example, a movie viewer that normally runs at 30 frames/sec may be willing to drop back to 25 frames/sec if there is not enough free bandwidth to support 30 frames/sec. Similarly, the number of pixels per frame, audio bandwidth, and other properties may be adjustable. Because many parties may be involved in the flow negotiation (the sender, the receiver, and all the routers along the path between them), flows must be described accurately in terms of specific parameters that can be negotiated. A set of such parameters is called a flow specification. Typically, the sender (e.g., the video server) produces a flow specification proposing the parameters it would like to use. As the specification propagates along the route, each router examines it and modifies the parameters as need be. The modifications can only reduce the flow, not increase it (e.g., a lower data rate, not a higher one). When it gets to the other end, the parameters can be established. As an example of what can be in a flow specification, consider the example of Fig. 5-35, it has five parameters, the first of which, the Token bucket rate, is the number of bytes per second that are put into the bucket. This is the maximum sustained rate the sender may transmit, averaged over a long time interval.

Figure 5-35. An example flow specification.

Parameter	Unit
Token bucket rate	Bytes/sec
Token bucket size	Bytes
Peak data rate	Bytes/sec
Minimum packet size	Bytes
Maximum packet size	Bytes

The second parameter is the size of the bucket in bytes. If, for example, the Token bucket rate is 1 Mbps and the Token bucket size is 500 KB, the bucket can fill continuously for 4 sec before it fills up (in the absence of any transmissions). Any tokens sent after that are lost. The third parameter, the Peak data rate, is the maximum tolerated transmission rate, even for brief time intervals. The sender must never exceed this rate.

The last two parameters specify the minimum and maximum packet sizes, including the transport and network layer headers (e.g., TCP and IP). The minimum size is important because processing each packet takes some fixed time, no matter how short. A router may be prepared to handle 10,000 packets/sec of 1 KB each, but not be prepared to handle 100,000 packets/sec of 50 bytes each, even though this represents a lower data rate. The maximum packet size is important due to internal network limitations that may not be exceeded. For example, if part of the path goes over an Ethernet, the maximum packet size will be restricted to no more than 1500 bytes no matter what the rest of the network can handle.

An interesting question is how a router turns a flow specification into a set of specific resource reservations. That mapping is implementation specific and is not standardized. Suppose that a router can process 100,000 packets/sec. If it is offered a flow of 1 MB/sec with minimum and maximum packet sizes of 512 bytes, the router can calculate that it might get 2048 packets/sec from that flow. In that case, it must reserve 2% of its CPU for that flow, preferably more to avoid long queueing delays. If a router's policy is never to allocate more than 50% of its CPU (which implies a factor of two delay, and it is already 49% full, then this flow must be rejected. Similar calculations are needed for the other resources.

The tighter the flow specification, the more useful it is to the routers. If a flow specification states that it needs a Token bucket rate of 5 MB/sec but packets can vary from 50 bytes to 1500 bytes, then the packet rate will vary from about 3500 packets/sec to 105,000 packets/sec. The router may panic at the latter number and reject the flow, whereas with a minimum packet size of 1000 bytes, the 5 MB/sec flow might have been accepted.

8.Proportional Routing

Most routing algorithms try to find the best path for each destination and send all traffic to that destination over the best path. A different approach that has been proposed to provide a higher quality of service is to split the traffic for each destination over multiple paths. Since routers generally do not have a complete overview of network-wide traffic, the only feasible way to split traffic over multiple routes is to use locally-available information. A simple method is to divide the traffic equally or in proportion to the capacity of the outgoing links. However, more sophisticated algorithms are also available.

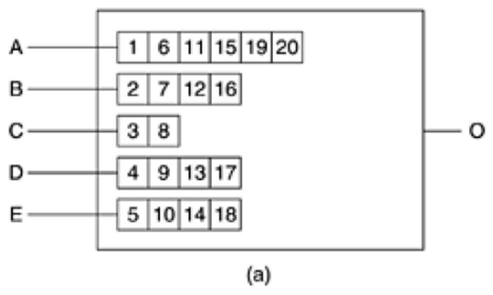
9.Packet Scheduling

If a router is handling multiple flows, there is a danger that one flow will hog too much of its capacity and starve all the other flows. Processing packets in the order of their arrival means that an aggressive sender can capture most of the capacity of the routers its packets traverse, reducing the quality of service for others. To prevent such attempts, various packet scheduling algorithms have been devised.

One of the first ones was the **fair queueing algorithm**. The essence of the algorithm is that routers have separate queues for each output line, one for each flow. When a line becomes idle, the router scans the queues round robin, taking the first packet on the next queue. In this way, with n hosts competing for a given output line, each host gets to send one out of every n packets. Sending more packets will not improve this fraction.

Although a start, the algorithm has a problem: it gives more bandwidth to hosts that use large packets than to hosts that use small packets. Demers et al. (1990) suggested an improvement in which the round robin is done in such a way as to simulate a byte-by-byte round robin, instead of a packet-by-packet round robin. In effect, it scans the queues repeatedly, byte-for byte, until it finds the tick on which each packet will be finished. The packets are then sorted in order of their finishing and sent in that order. The algorithm is illustrated in Fig. 5-36.

Figure 5-36. (a) A router with five packets queued for line O. (b) Finishing times for the five packets.



Packet	Finishing time
C	8
B	16
D	17
E	18
A	20

(b)

In Fig. 5-36(a) we see packets of length 2 to 6 bytes. At (virtual) clock tick 1, the first byte of the packet on line A is sent. Then goes the first byte of the packet on line B, and so on. The first packet to finish is C, after eight ticks. The sorted order is given in Fig. 5-36(b). In the absence of new arrivals, the packets will be sent in the order listed, from C to A.

One problem with this algorithm is that it gives all hosts the same priority. In many situations, it is desirable to give video servers more bandwidth than regular file servers so that they can be given two or more bytes per tick. This modified algorithm is called **weighted fair queueing** and is widely used. Sometimes the weight is equal to the number of flows coming out of a machine, so each process gets equal bandwidth.

8. Integrated Services

Between 1995 and 1997, IETF put a lot of effort into devising an architecture for streaming multimedia. [IETF stands for the Internet Engineering Task Force. It is an open, international community of network designers, operators, vendors, and researchers who work together to develop and promote internet standards. The IETF plays a critical role in shaping the technologies and protocols that form the foundation of the internet.] This work resulted in over two dozen RFCs, starting with RFCs 2205–2210. [RFC stands for "Request for Comments." It is a series of documents published by the Internet Engineering Task Force (IETF) and other organizations to propose and describe internet standards, protocols, procedures, and other technical specifications. RFCs serve as the primary means through which internet standards and protocols are defined and documented]. The generic name for this work is flow-based algorithms or integrated services.

Integrated Services (IntServ), also known as Integrated Services Model, is a quality of service (QoS) architecture developed to provide guaranteed and controlled service levels for individual data flows over an IP network. It was introduced to address the requirements of real-time and multimedia applications that are sensitive to delay, jitter, and packet loss. IntServ is based on the idea of reserving network resources in advance to ensure the required QoS for specific flows.

It was aimed at both unicast and multicast applications. An example of the former is a single user streaming a video clip from a news site. An example of the latter is a collection of digital television stations broadcasting their programs as streams of IP packets to many receivers at various locations. In many multicast applications, groups can change membership dynamically, for example, as people enter a video conference and then get bored and switch to a soap opera or the croquet channel. Under these conditions, the approach of having the senders reserve bandwidth in advance does not work well, since it would require each sender to track all entries and exits of
its audience. For a system designed to transmit television with millions of subscribers, it would not work at all.

RSVP—The Resource reSerVation Protocol

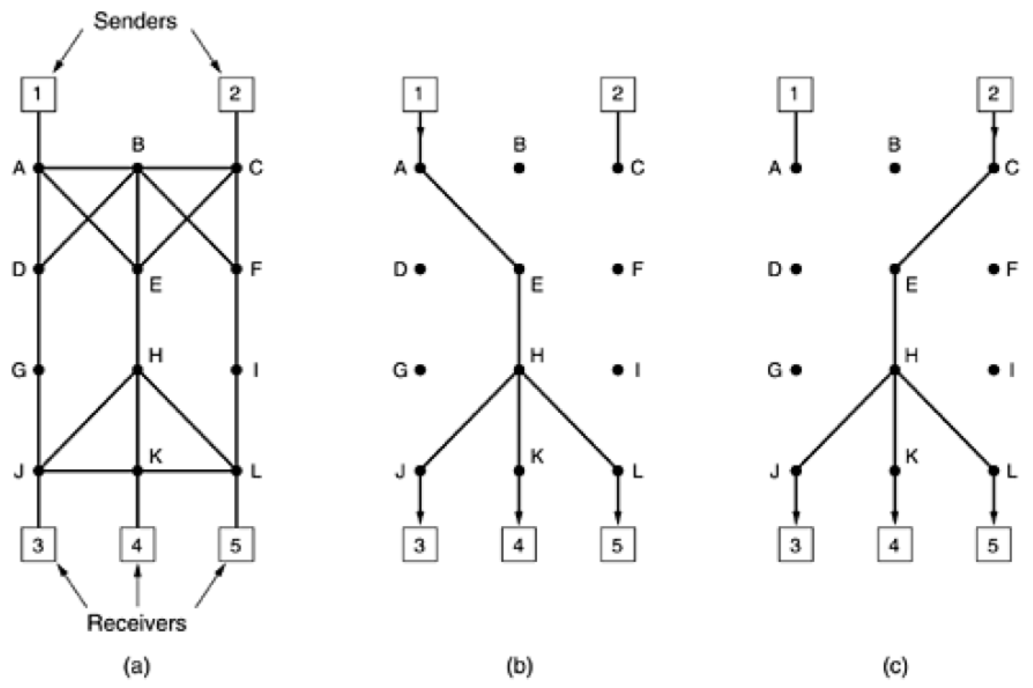
RSVP stands for "Resource Reservation Protocol." It is a signalling protocol used in computer networks to establish and maintain resource reservations for specific data flows, ensuring Quality of Service (QoS) guarantees in IP-based networks. RSVP is a crucial component of the Integrated Services (IntServ) architecture, which aims to provide end-to-end QoS for individual data flows.

The main IETF protocol for the integrated services architecture is RSVP. It is described in RFC 2205 and others. This protocol is used for making the reservations; other protocols are used for sending the data. RSVP allows multiple senders to transmit to multiple groups of receivers, permits individual receivers to switch channels freely, and optimizes bandwidth use while at the same time eliminating congestion.

In its simplest form, the protocol uses multicast routing using spanning trees. Each group is assigned a group address. To send to a group, a sender puts the group's address in its packets. The standard multicast routing algorithm then builds a spanning tree covering all group members. The routing algorithm is not part of RSVP. The only difference from normal multicasting is a little extra information that is multicast to the group periodically to tell the routers along the tree to maintain certain data structures in their memories.

As an example, consider the network of Fig. 5-37(a). Hosts 1 and 2 are multicast senders, and hosts 3, 4, and 5 are multicast receivers. In this example, the senders and receivers are disjoint, but in general, the two sets may overlap. The multicast trees for hosts 1 and 2 are shown in Fig. 5-37(b) and Fig. 5-37(c), respectively.

Figure 5-37. (a) A network. (b) The multicast spanning tree for host 1. (c) The multicast spanning tree for host 2.

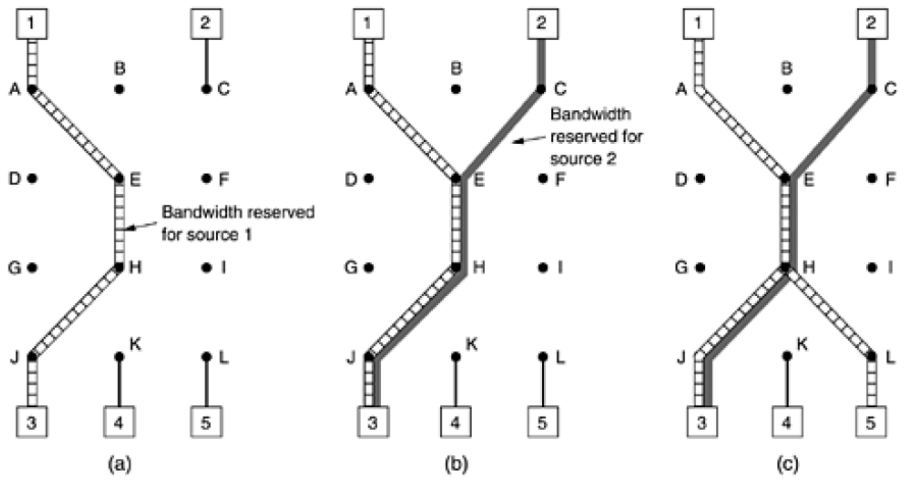


To get better reception and eliminate congestion, any of the receivers in a group can send a reservation message up the tree to the sender. The message is propagated using the reverse path forwarding algorithm. At each hop, the router notes the reservation and reserves the necessary bandwidth. If insufficient bandwidth is available, it reports back failure.

By the time the message gets back to the source, bandwidth has been reserved all the way from the sender to the receiver making the reservation request along the spanning tree.

An example of such a reservation is shown in Fig. 5-38(a). Here host 3 has requested a channel to host 1. Once it has been established, packets can flow from 1 to 3 without congestion. Now consider what happens if host 3 next reserves a channel to the other sender, host 2, so the user can watch two television programs at once. A second path is reserved, as illustrated in Fig. 5-38(b). Note that two separate channels are needed from host 3 to router E because two independent streams are being transmitted.

Figure 5-38. (a) Host 3 requests a channel to host 1. (b) Host 3 then requests a second channel, to host 2. (c) Host 5 requests a channel to host 1.


1
2
A
B
C
D
E
F
G
H
I
J
K
L
3
4
5
Bandwidth reserved for source 1
Bandwidth reserved for source 2
(a)
(b)
(c)

Finally, in Fig. 5-38(c), host 5 decides to watch the program being transmitted by host 1 and also makes a reservation. First, dedicated bandwidth is reserved as far as router H. However, this router sees that it already has a feed from host 1, so if the necessary bandwidth has already been reserved, it does not have to reserve any more. Note that hosts 3 and 5 might have asked for different amounts of bandwidth (e.g., 3 has a black-and-white television set, so it does not want the color information), so the capacity reserved must be large enough to satisfy the greediest receiver.

When making a reservation, a receiver can (optionally) specify one or more sources that it wants to receive from. It can also specify whether these choices are fixed for the duration of the reservation or whether the receiver wants to keep open the option of changing sources later. The routers use this information to optimize bandwidth planning. In particular, two receivers are only set up to share a path if they both agree not to change sources later on. The reason for this strategy in the fully dynamic case is that reserved bandwidth is decoupled from the choice of source. Once a receiver has reserved bandwidth, it can switch to another source and keep that portion of the existing path that is valid for the new source. If host 2 is transmitting several video streams, for example, host 3 may switch between them at will without changing its reservation: the routers do not care what program the receiver is watching.

9. Differentiated Services

Flow-based algorithms have the potential to offer good quality of service to one or more flows because they reserve whatever resources are needed along the route. However, they also have a downside. They require an advance setup to establish each flow, something that does not scale well when there are thousands or millions of flows. Also, they maintain internal per-flow state in the routers, making them vulnerable to router crashes. "internal per-flow state in the routers" refers to the maintenance of specific information or data related to individual flows within the routers of a network. This per-flow state allows routers to make intelligent decisions about how to handle and forward network traffic for each flow passing through them. Finally, the changes required to the router code are substantial and involve complex router-to-router exchanges for setting up the flows. As a consequence, few implementations of RSVP or anything like it exist yet. For these reasons, IETF has also devised a simpler approach to quality of service, one that can be largely implemented locally in each router without advance setup and without having the whole path involved. This approach is known as **class-based** (as opposed to flow-based) **quality of service.** IETF has standardized an architecture for it, called differentiated services.

Differentiated services (DS) can be offered by a set of routers forming an administrative domain (e.g., an ISP or a telco). The administration defines a set of service classes with corresponding forwarding rules. If a customer signs up for DS, customer packets entering the domain may carry a Type of Service field in them, with better service provided to some classes (e.g., premium service) than to others. Traffic within a class may be required to conform to some specific shape, such as a leaky bucket with some specified drain rate. An operator with a nose for business might charge extra for each premium packet transported or might allow up to N premium packets per month for a fixed additional monthly fee. Note that this scheme requires no advance setup, no resource reservation, and no time-consuming end-to-end negotiation for each flow, as with integrated services. This makes DS relatively easy to implement.

Class-based service also occurs in other industries. For example, package delivery companies often offer overnight, two-day, and three-day service. Airlines offer first class, business class, and cattle class service. Long-distance trains often have multiple service classes. Even the Paris subway has two service classes. For packets, the classes may differ in terms of delay, jitter, and probability of being discarded in the event of congestion, among other possibilities (but probably not roomier Ethernet frames).

Flow-based Quality of Service (QoS) and Class-based Quality of Service are two different approaches used to manage and prioritize network traffic based on different criteria. Here's a comparison of the two:

Flow-Based Quality of Service:

1. **Individual Flow Treatment:** Flow-based QoS focuses on managing and prioritizing traffic on a per-flow basis. Each flow represents a unidirectional sequence of packets sharing common characteristics, such as source and destination IP addresses, ports, and protocol type.
2. **Per-Flow State:** Flow-based QoS requires routers or network devices to maintain per-flow state information. This information includes data specific to each flow, which allows the network to apply different QoS policies to each flow independently.
3. **Granular Control:** Flow-based QoS offers more granular control over the treatment of individual flows, allowing for fine-tuning of QoS policies based on the specific requirements of each flow.
4. **Complexity:** Managing per-flow state and applying differentiated treatment to each flow can add complexity to the network devices. It may require additional memory and processing resources to maintain the state and enforce QoS policies.

Class-Based Quality of Service:

1. **Traffic Classification and Aggregation:** Class-based QoS involves classifying traffic into different classes based on certain criteria, such as IP address ranges, protocol type, port numbers, or any other packet header information. Traffic is then aggregated into these classes.
2. **Per-Class Treatment:** Instead of maintaining per-flow state, class-based QoS treats all flows within a specific class in a similar manner. Each class is associated with a specific Quality of Service treatment, known as the Per-Hop Behavior (PHB).
3. **Simplified State Management:** Class-based QoS simplifies state management by grouping similar flows into classes. This reduces the complexity associated with maintaining per-flow state.
4. **Scalability:** Class-based QoS is more scalable compared to flow-based QoS, especially in large networks with numerous flows. It allows routers to apply QoS policies based on class membership rather than individual flows.
5. **Limited Granularity:** While class-based QoS is simpler to implement and scale, it may offer less granularity compared to flow-based QoS. Fine-tuning QoS treatment for specific flows within a class may not be possible with this approach.

Use Cases:

- **Flow-Based QoS:** Flow-based QoS is suitable when there is a need to provide highly differentiated treatment to individual flows. It is often used for real-time applications like VoIP and video conferencing, where each flow requires specific QoS guarantees.
- **Class-Based QoS:** Class-based QoS is more commonly used when traffic can be aggregated into classes with similar requirements. It is efficient for prioritizing different types of traffic in a more general way, such as giving higher priority to mission-critical applications or bulk data transfers.

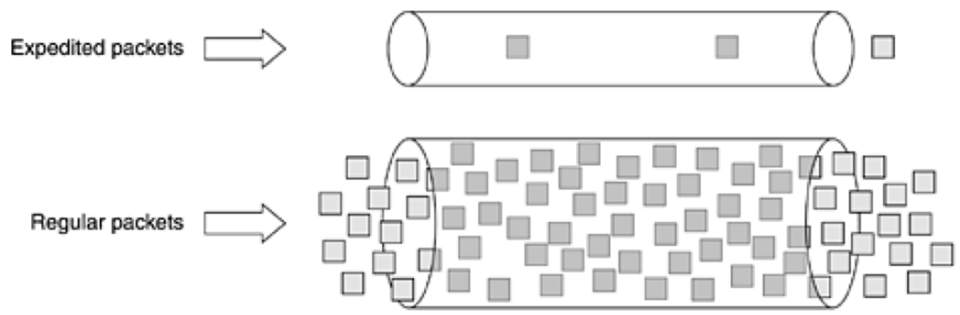
To make the difference between flow-based quality of service and class-based quality of service clearer, consider an example: Internet telephony. With a flow-based scheme, each telephone call gets its own resources and guarantees. With a class-based scheme, all the telephone calls together get the resources reserved for the class telephony. These resources cannot be taken away by packets from the file transfer class or other classes, but no telephone call gets any private resources reserved for it alone.

Expedited Forwarding

The choice of service classes is up to each operator, but since packets are often forwarded between subnets run by different operators, IETF is working on defining network-independent service classes. The simplest class is expedited forwarding.

The idea behind expedited forwarding is very simple. Two classes of services are available: regular and expedited. The vast majority of the traffic is expected to be regular, but a small fraction of the packets are expedited. The expedited packets should be able to transit the subnet as though no other packets were present. A symbolic representation of this "two-tube" system is given in Fig. 5-39. Note that there is still just one physical line. The two logical pipes shown in the figure represent a way to reserve bandwidth, not a second physical line.

Figure 5-39. Expedited packets experience a traffic-free network.



One way to implement this strategy is to program the routers to have two output queues for each outgoing line, one for expedited packets and one for regular packets. When a packet arrives, it is queued accordingly. Packet scheduling should use something like weighted fair queueing. For example, if 10% of the traffic is expedited and 90% is regular, 20% of the bandwidth could be dedicated to expedited traffic and the rest to regular traffic. Doing so would give the expedited traffic twice as much bandwidth as it needs in order to provide low delay for it. This allocation can be achieved by transmitting one expedited packet for every four regular packets (assuming the size distribution for both classes is similar). In this way, it is hoped that expedited packets see an unloaded subnet, even when there is, in fact, a heavy load.

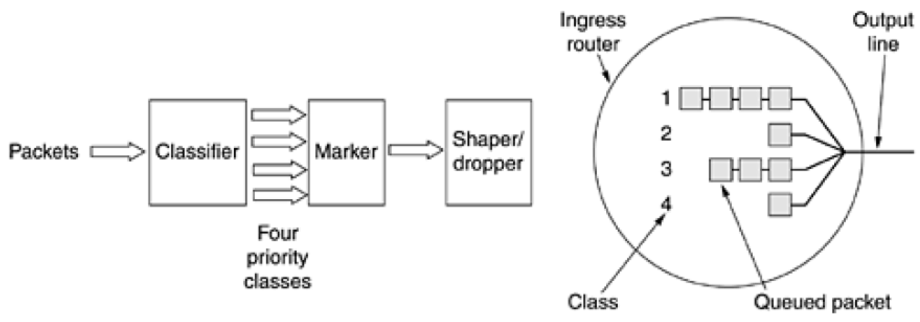
Assured Forwarding

A somewhat more elaborate scheme for managing the service classes is called assured forwarding. It specifies that there shall be four priority classes, each class having its own resources. In addition, it defines three discard probabilities for packets that are undergoing congestion: low, medium, and high. Taken together, these two factors define 12 service classes. Figure 5-40 shows one-way packets might be processed under assured forwarding.

Step 1

classify the packets into one of the four priority classes. This step might be done on the sending host (as shown in the figure) or in the ingress (first) router. The advantage of doing classification on the sending host is that more information is available about which packets belong to which flows there.

Figure 5-40. A possible implementation of the data flow for assured forwarding.



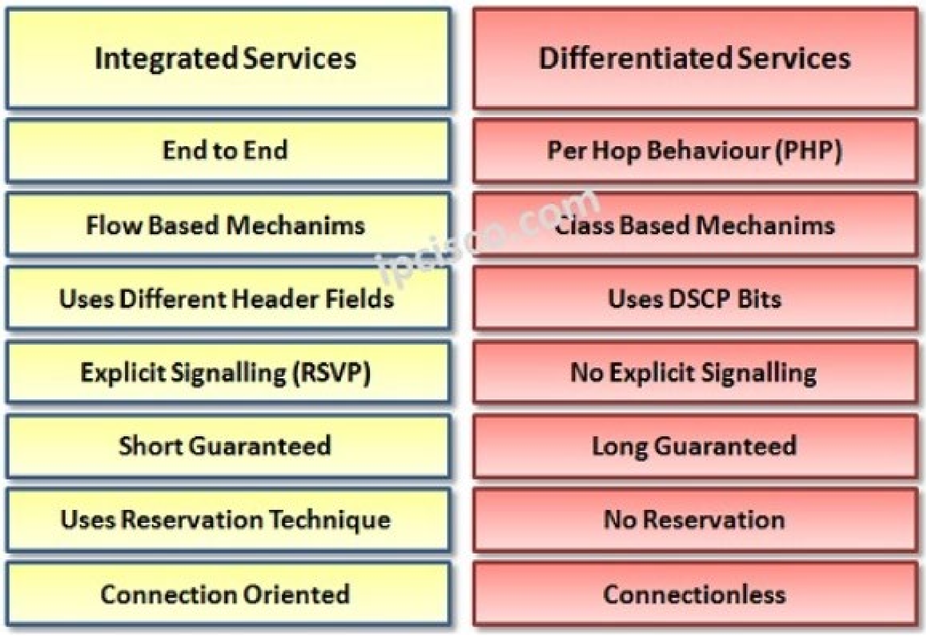
Step 2

mark the packets according to their class. A header field is needed for this purpose. Fortunately, an 8-bit Type of service field is available in the IP header. RFC 2597 specifies that six of these bits are to be used for the service class, leaving coding room for historical service classes and future ones.

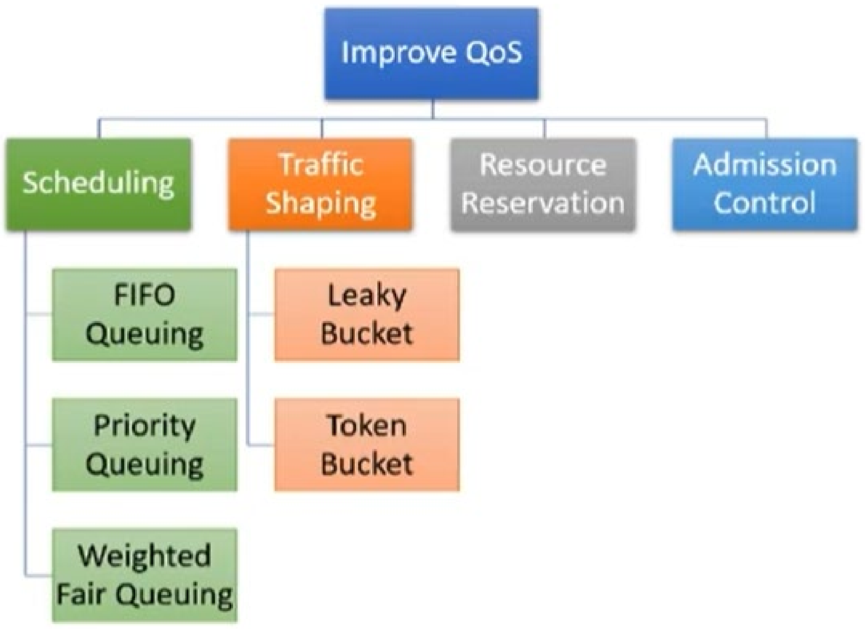
Step 3

pass the packets through a shaper/dropper filter that may delay or drop some of them to shape the four streams into acceptable forms, for example, by using leaky or token buckets. If there are too many packets, some of them may be discarded here, by discard category. More elaborate schemes involving metering or feedback are also possible.

In this example, these three steps are performed on the sending host, so the output stream is now fed into the ingress router. It is worth noting that these steps may be performed by special networking software or even the operating system, to avoid having to change existing applications.



Integrated Services	Differentiated Services
End to End	Per Hop Behaviour (PHP)
Flow Based Mechanims	Class Based Mechanims
Uses Different Header Fields	Uses DSCP Bits
Explicit Signalling (RSVP)	No Explicit Signalling
Short Guaranteed	Long Guaranteed
Uses Reservation Technique	No Reservation
Connection Oriented	Connectionless


Improve QoS
Scheduling
Traffic Shaping
Resource Reservation
Admission Control
FIFO Queuing
Priority Queuing
Weighted Fair Queuing
Leaky Bucket
Token Bucket