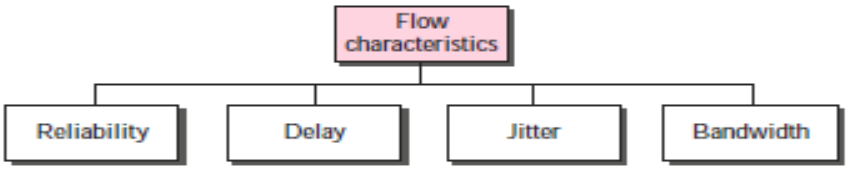
Quality of Service (QoS)

The Internet was originally designed for best-effort service without guarantee of predictable performance. **Best-effort service** is often sufficient for a traffic that is **not sensitive to delay**, such as file transfers and e-mail. Such a traffic is called **elastic** because it can **stretch** to work under **delay conditions**; it is also called **available bit rate** because applications can speed up or slow down according to the available bit rate.

The real-time traffic generated by some multimedia applications is delay sensitive and therefore requires guaranteed and predictable performance. **Quality of service (QoS) is an internetworking issue that refers to a set of techniques and mechanisms that guarantee the performance of the network to deliver predictable service to an application program.**

DATA-FLOW CHARACTERISTICS

Traditionally, four types of characteristics are attributed to a flow: **reliability, delay, jitter, and bandwidth.** To provide quality of server (QoS).



Reliability

Reliability is a characteristic that a flow needs **in order to deliver the packets safe and sound to the destination.** Lack of reliability means losing a packet or acknowledgment, which entails retransmission. However, the sensitivity of different application programs to reliability varies. For example, reliable transmission is more important for electronic mail, file transfer, and Internet access than for telephony or audio conferencing.

Delay

Source-to-destination delay is another flow characteristic. Again, **applications can tolerate delay in different degrees.** In this case, telephony, audio conferencing, video conferencing, and remote logging need minimum delay, while delay in file transfer or e-mail is less important.

Jitter

Jitter is the variation in delay for packets belonging to the same flow. For example, if four packets depart at times 0, 1, 2, 3 and arrive at 20, 21, 22, 23, all have the same delay, 20 units of time. On the other hand, if the above four packets arrive at 21, 23, 24, and 28, they will have different delays. For applications such as audio and video, the first case is completely acceptable; the second case is not. For these applications, it does not matter if the packets arrive with a short or long delay as long as the delay is the same for all packets. These types of applications do not tolerate jitter.

Bandwidth

Different applications need different bandwidths. In video conferencing we need to send millions of bits per second to refresh a color screen while the total number of bits in an e-mail may not reach even a million.

Table : *Sensitivity of applications to flow characteristics*

Application	*Reliability*	*Delay*	*Jitter*	*Bandwidth*
FTP	High	Low	Low	Medium
HTTP	High	Medium	Low	Medium
Audio-on-demand	Low	Low	High	Medium
Video-on-demand	Low	Low	High	High
Voice over IP	Low	High	High	Low
Video over IP	Low	High	High	High

Flow Classes

Based on the flow characteristics, we can classify flows into groups, with each group having similar levels of characteristics.

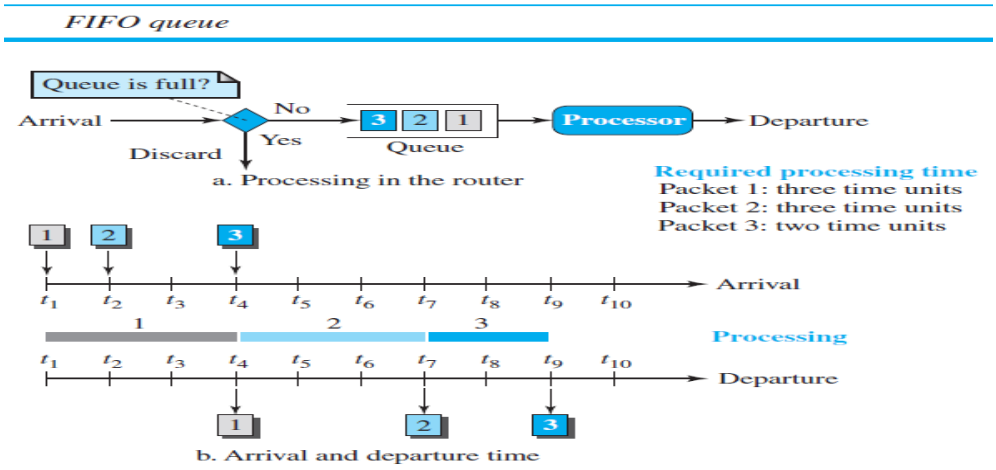
Techniques to Improve QoS

Some techniques that can be used to improve the quality of service. We briefly discuss four common methods: **scheduling, traffic shaping, admission control, and resource reservation.**

1) Scheduling

Packets from different flows arrive at a switch or router for processing. **A good scheduling technique treats the different flows in a fair and appropriate manner.** Several scheduling techniques are designed to improve the quality of service. We discuss three of them here: **FIFO queuing, priority queuing, and weighted fair queuing.**

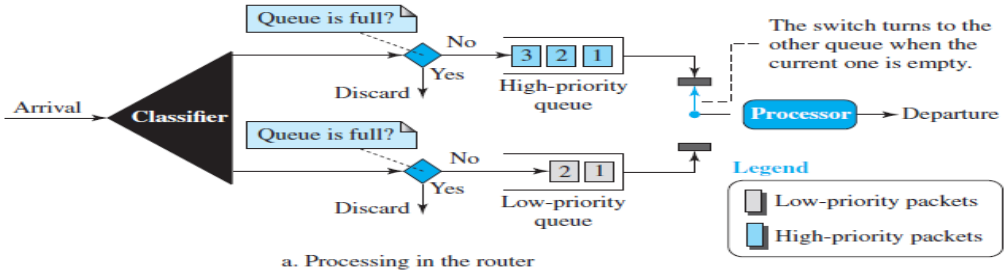
A. **In first-in, first-out (FIFO)** queuing, packets wait in a buffer (queue) until the node (router) is ready to process them. If the average arrival rate is higher than the average processing rate, the queue will fill up and new packets will be discarded.



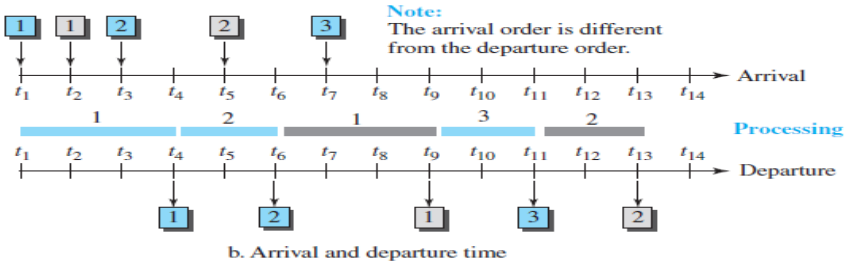
B. Priority Queuing

In priority queuing, packets **are first assigned to a priority class.** Each priority class **has its own queue.** The packets in the highest-priority queue are processed first. Packets in the lowest-priority queue are processed last. Note that the system does not stop serving a queue until it is empty. A packet priority is determined from a specific field in the packet header: the **ToS field** of an **IPv4 header**, the **priority field** of **IPv6**, a priority number assigned to **a destination address**, or a priority number assigned to **an application** (destination port number), and so on.

Figure shows priority queuing with two priority levels (for simplicity). **A priority queue can provide better QoS than the FIFO queue because higher-priority traffic, such as multimedia, can reach the destination with less delay.** However, there is **a potential drawback.** If there is a continuous flow in a high-priority queue, the packets in the lower-priority queues will never have a chance to be processed. This is a condition called **starvation**. Severe starvation may result in dropping of some packets of lower priority. In the figure, the packets of higher priority are sent out before the packets of lower priority.



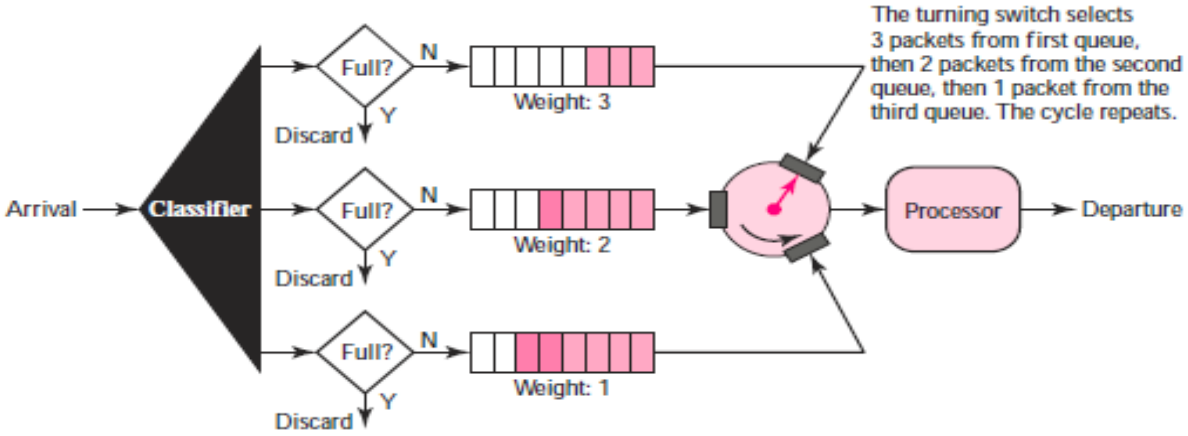
a. Processing in the router



b. Arrival and departure time

C. Weighted Fair Queuing

A better scheduling method is **weighted fair queuing.** In this technique, the packets are still assigned to different classes and admitted to different queues. **The queues, however, are weighted based on the priority of the queues; higher priority means a higher weight**. The system processes packets in each queue **in a round-robin fashion** with the number of packets selected from each queue based on the corresponding weight. For example, if the weights are 3, 2, and 1, three packets are processed from the first queue, two from the second queue, and one from the third queue. If the system does not impose priority on the classes, all weights can be equal. In this way, we have fair queuing with priority.

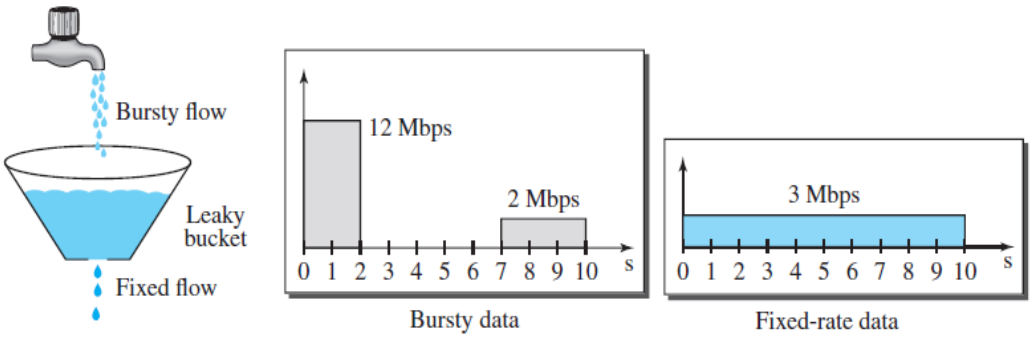


2) Traffic Shaping

Traffic shaping is a mechanism to **control** the **amount** and the **rate** of **the traffic** sent to the network. Two techniques can shape traffic: leaky bucket and token bucket.

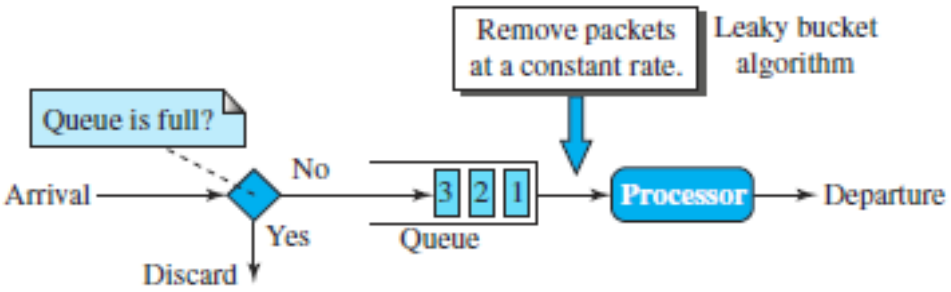
A. Leaky Bucket

In networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate.



In the figure, we assume that the network has committed a bandwidth of **3 Mbps** for a host. **The use of the leaky bucket shapes the input traffic to make it conform to this commitment.** The host sends a burst of data at a rate of 12 Mbps for 2 seconds, for a total of 24 Mb of data. The host is silent for 5 seconds and then sends data at a rate of 2 Mbps for 3 seconds, for a total of 6 Mb of data. In all, the host has sent 30 Mb of data in 10 seconds. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10 seconds. **Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host**. We can also see that the leaky bucket **may prevent congestion**.

A simple leaky bucket implementation is shown in the following Figure. A FIFO queue holds the packets. If the traffic consists of fixed-size packets, the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.



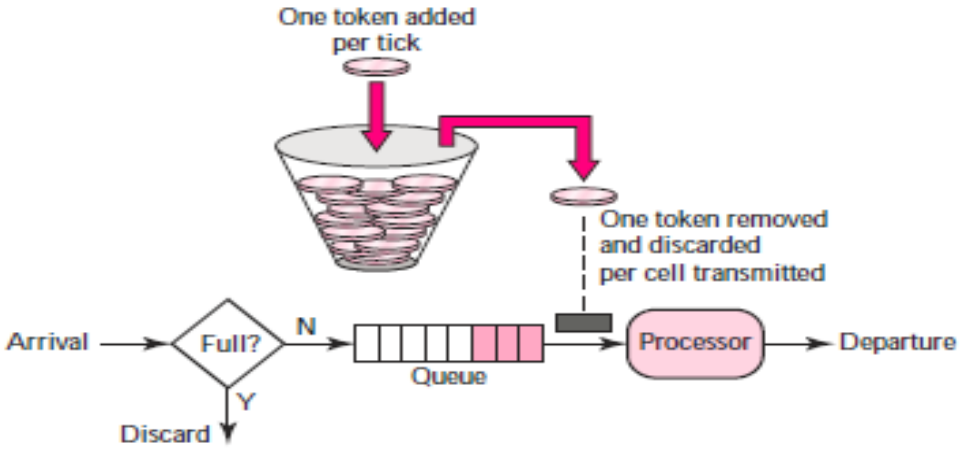
The following is an algorithm for variable-length packets:

1- Initialize a counter to **n** at the tick of the clock.
2- If **n** is **greater than** the **size of the packet**, **send** the packet and **decrement** the counter by the packet size. **Repeat** this step **until** the counter value is **smaller** than the packet size.
3- Reset the counter to **n** and go to step 1.

A leaky bucket algorithm shapes bursty traffic into fixed-rate traffic by averaging the data rate. It may drop the packets if the bucket is full.

B. Token Bucket

The leaky bucket is very restrictive. It does not credit an idle host. For example, if a host is not sending for a while, its bucket becomes empty. Now if the host has bursty data, the leaky bucket allows only an average rate. The time when the host was idle is not taken into account. **On the other hand, the token bucket algorithm allows idle hosts to accumulate credit for the future in the form of tokens.** For each tick of the clock, the system sends **n** tokens to the bucket. The system removes one token for every cell (or byte) of data sent. For example, if **n** is **100** and the host is idle for **100 ticks**, the bucket collects 10,000 tokens. Now the host can consume all these tokens in **one tick** with **10,000 cells**, or the host takes **1,000 ticks** with **10 cells** per tick. In other words, the host can send bursty data as long as the bucket is not empty.



The token bucket can easily be implemented with a counter. The token is initialized to zero. Each time a token is added, the counter is incremented by 1. Each time a unit of data is sent, the counter is decremented by 1. When the counter is zero, the host cannot send data.

The token bucket allows bursty traffic at a regulated maximum rate.

C. Combining Token Bucket and Leaky Bucket

The two techniques can be combined to credit an idle host and at the same time regulate the traffic. The **leaky bucket** is applied **after** the **token bucket**; the **rate of the leaky bucket** needs to be **higher than** the **rate of tokens dropped in the bucket**.

3) Resource Reservation

A flow of data needs resources such as **a buffer, bandwidth, CPU time, and so on.** The quality of service is improved if these resources are reserved beforehand. One QoS model called **Integrated Services**, which depends heavily on resource reservation to improve the quality of service.

4) Admission Control

Admission control refers to the mechanism used by a router, or a switch, to accept or reject a flow based on predefined parameters called flow specifications. Before a router accepts a flow for processing, it checks the flow specifications to see if its capacity (**in terms of bandwidth, buffer size, CPU speed, etc.**) and its previous commitments to other flows can handle the new flow.