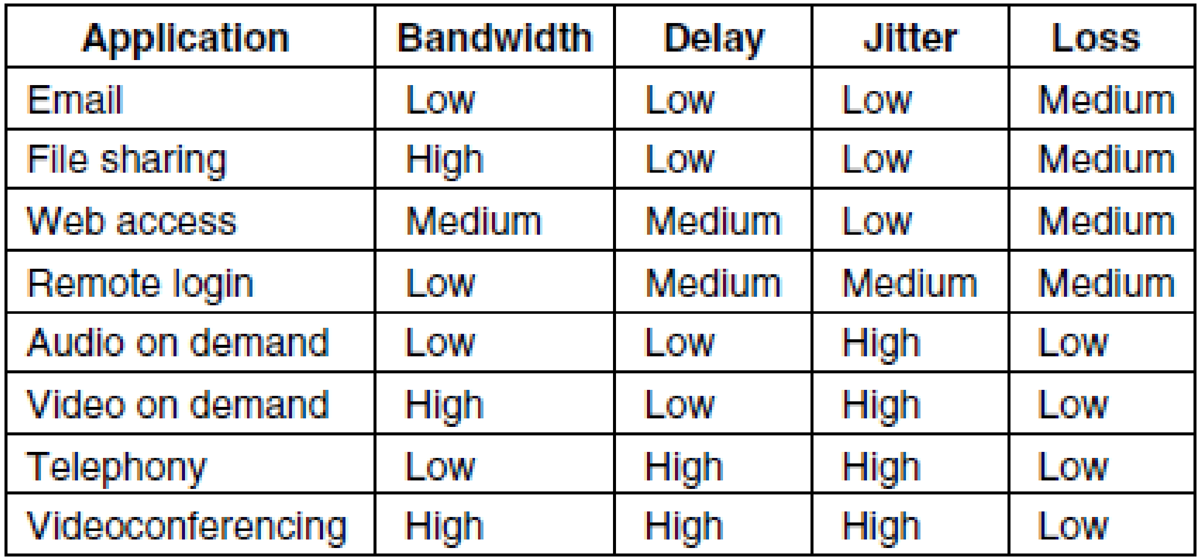
Quality of Service

- 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.

Stringency of **applications’ quality-of-service requirements.**



Application	Bandwidth	Delay	Jitter	Loss
Email	Low	Low	Low	Medium
File sharing	High	Low	Low	Medium
Web access	Medium	Medium	Low	Medium
Remote login	Low	Medium	Medium	Medium
Audio on demand	Low	Low	High	Low
Video on demand	High	Low	High	Low
Telephony	Low	High	High	Low
Videoconferencing	High	High	High	Low

- The first four applications have stringent requirements on reliability.
- No bits may be delivered incorrectly.
- This goal is usually achieved by checksumming 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.
- 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.
- 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.

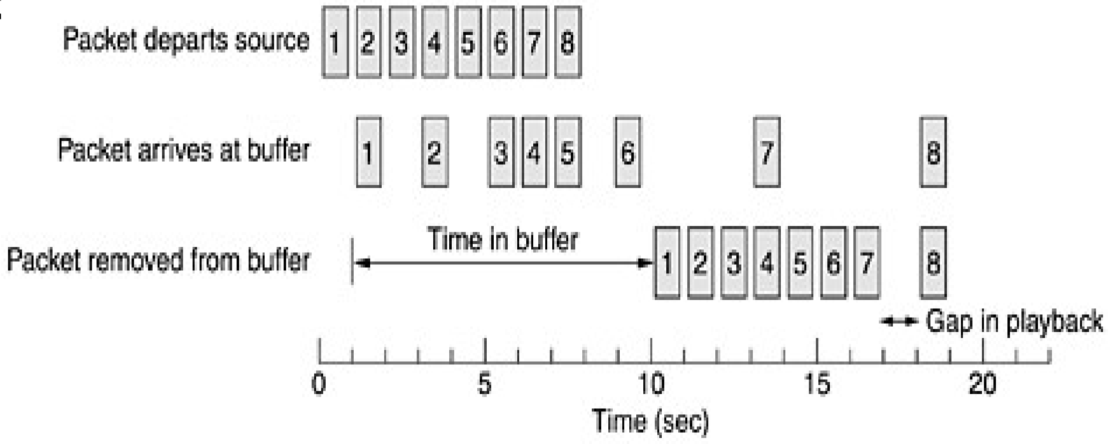
Techniques for Achieving Good Quality of Service

1.Overprovisioning

- An easy solution is to provide so much router capacity, buffer space, and bandwidth that the packets flow easily.
- The trouble with this solution is that it is expensive.

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



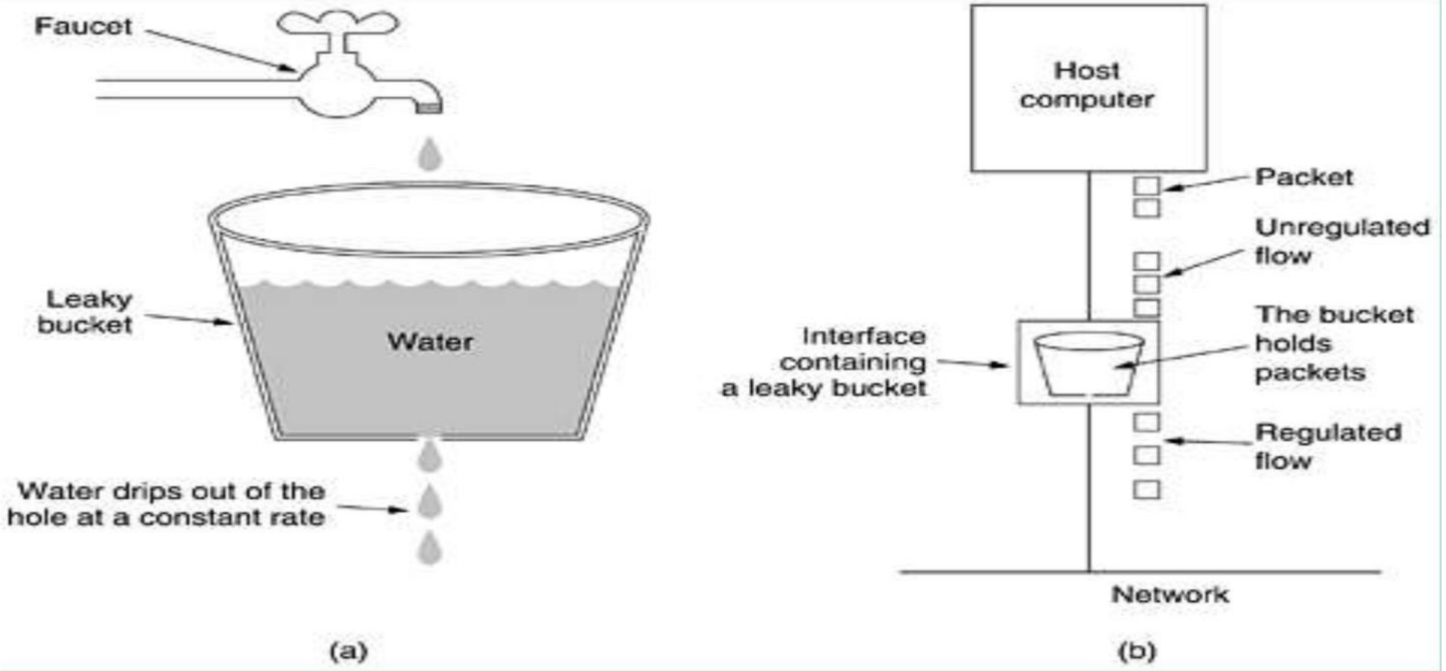
3.Traffic Shaping

- Traffic shaping make the server (and hosts in general) transmit at a uniform rate, smooths out the traffic on the server side, rather than on the client side.
- Traffic shaping is about regulating the average *rate* of data transmission. When a connection is set up, the user and the subnet agree on a certain traffic pattern (i.e., shape) for that circuit, which is called a **service level agreement**.
- Traffic shaping reduces congestion. 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.
- 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.

Leaky Bucket Algorithm

- It is a single-server queuing system with constant service time.
- Imagine a bucket with a small hole in the bottom, as illustrated in Fig. (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.

Leaky Bucket



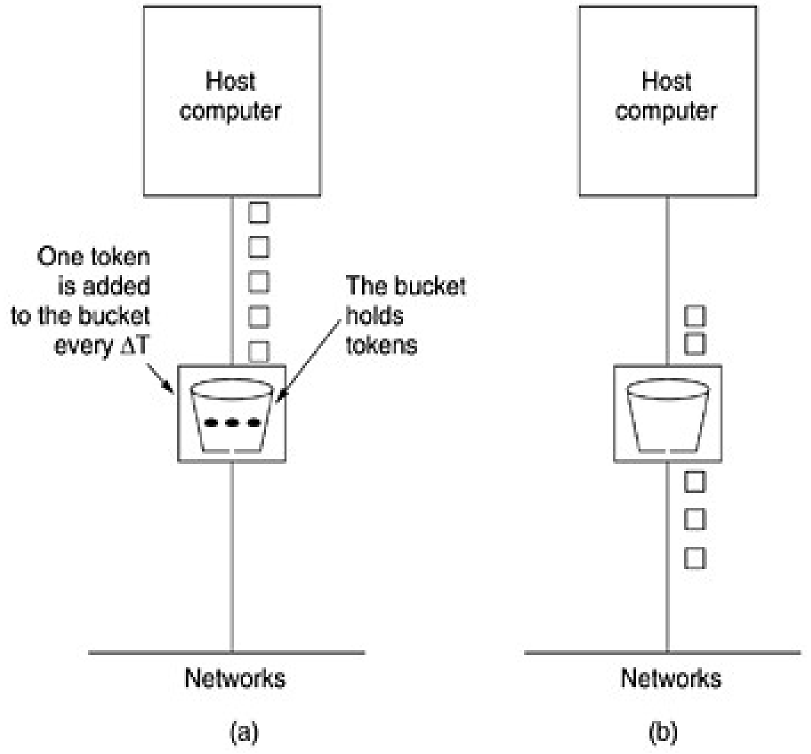
Leaky Bucket Algorithm

- The same idea can be applied to packets, as shown in Fig. (b).
- 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.
- The host is allowed to put one packet per clock tick onto the network. 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.

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 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**.

The Token Bucket Algorithm



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. (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. (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.
- 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.

The Token Bucket Algorithm

- 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, 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.
- 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.

Comparison

- Leaky bucket does not taken into account idle host , if a host is not sending for a while, its bucket becomes empty.
- If the host has bursty data, leaky bucket allows only average rate.
- Token bucket takes into account the idle time , with each clock tick the tokens are added to bucket, when the data needs to be send, it collects token from bucket and then send the data packet.

Comparison

- 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.
- The leaky bucket and token bucket algorithms can also be used to smooth traffic between routers, as well as to regulate host output.

Resource Reservation

- A flow of data needs resources such as buffer, bandwidth, CPU time, and so on .
- **QoS can be improved if these resources are reserved beforehand**
- Three different kinds of resources can potentially be reserved:
  - Bandwidth.
  - Buffer space.
  - CPU cycles.

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.
- Routers or switches puts **restrictions on the admission of packets from host**.

- Before a router accepts the flow , it checks the flow for specifications in terms of bandwidth , buffer size ,cpu speed etc.