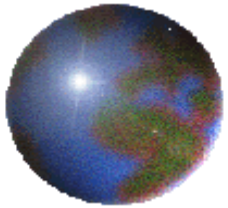


CEN445
Network Protocols & Algorithms

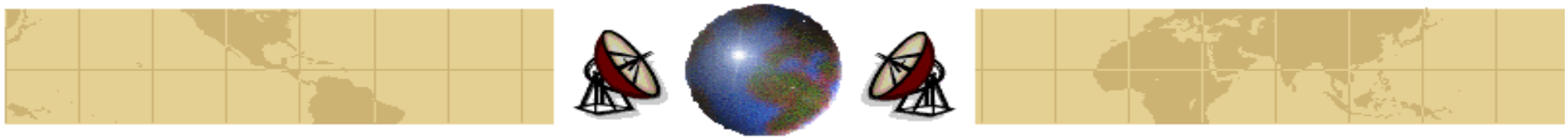
Chapter 5

Network Layer

Prepared by

Dr. Mohammed Amer Arafah

Summer 2008

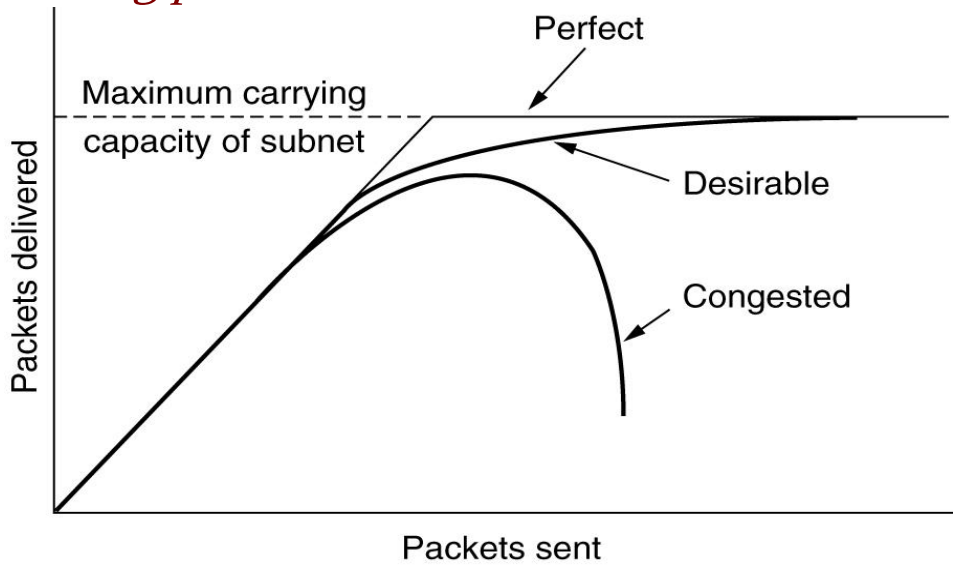


Congestion Control Algorithms

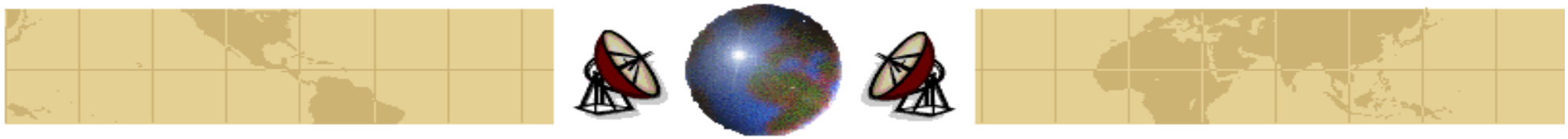
- General Principles of Congestion Control
- Congestion Prevention Policies
- Congestion Control in Virtual-Circuit Subnets
- Load Shedding
- Jitter Control

Congestion

- When too many packets are present is (a part of) the subnet, performance degrades. This situation is called **congestion**.
- When the number of packets dumped into the subnet by the hosts is within its carrying capacity, the number delivered is proportional to the number sent. However, *as traffic increases too far, the routers are no longer able to cope, and they begin losing packets*.



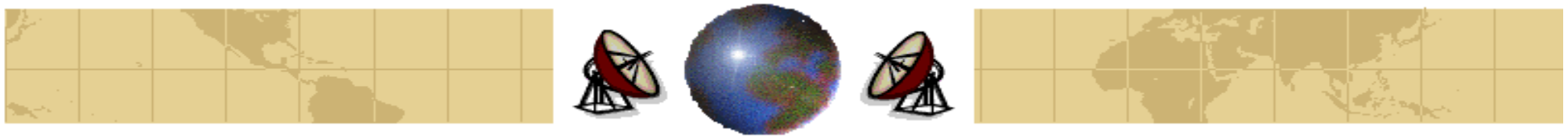
When too much traffic is offered, congestion sets in and performance degrades sharply.



Congestion Control Algorithms

Factors Causing Congestion:

- If *packets arriving on several input lines and all need the same output line*, a queue will build up. If there is insufficient memory, packets will be lost. Adding up memory, congestion may get worse. The reason is that the time for packets to get to front of the queue, they have already timed out, and duplicates have been sent.
- If the *routers' CPUs are slow*, queues can build up.
- *Low bandwidth lines*.



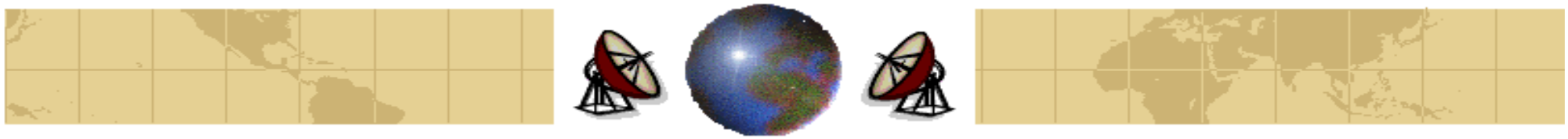
Comparison between Congestion Control and Flow Control

First: Congestion Control

- Congestion control has to do with making sure that the subnet is able to carry the offered traffic.
- It is a global issue, involving the behavior of all hosts, all the routers, the store-and-forward mechanism within the routers, and others.

Second: Flow Control

- Flow control relates to the point-to-point traffic between a given sender and a given receiver. Its job is to make sure that a faster sender cannot continuously transmit data faster than the receiver can absorb it.
- Flow control involves a direct feedback from the receiver to the sender.



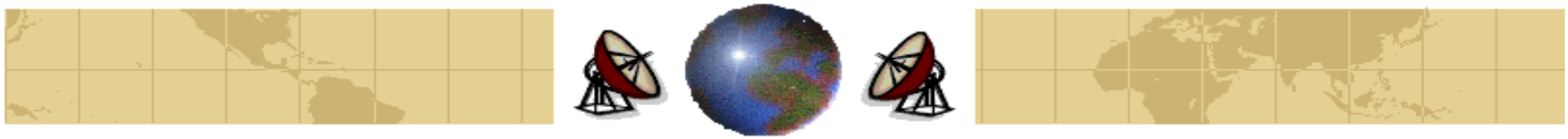
Comparison between Congestion Control and Flow Control

Example 1:

- Consider a fiber optic network with a capacity of 1000 gigabit/sec on which supercomputer is trying to transfer a file to a personal computer at 1Gbps. Although there is no congestion control, flow control is needed to force the supercomputer to give the personal computer a chance to breathe.

Example 2:

- Consider a store-and-forward network with 1-Mbps lines and 1000 large computers, half of them are trying to transfer files at 100 kbps to the other half. Here the problem is not that of fast senders overpowering slow receivers, but that the total offered traffic exceeds what the network can handle.

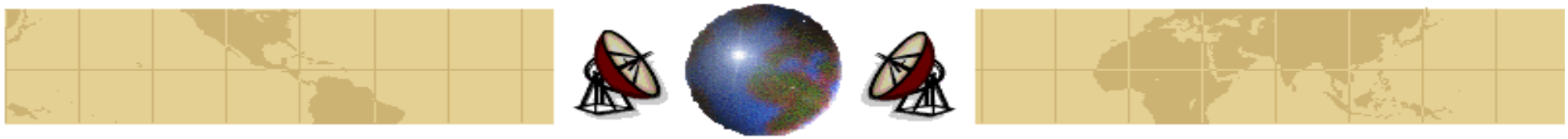


General Principles of Congestion Control

- Many problems in complex systems. Such as computer networks, can be viewed from a *control theory* point of view. The solutions can be either:

I. Open Loop Solutions:

- Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in first place.
- The tools for doing open control include:
  - Deciding when to accept new traffic.
  - Deciding when to discard packets and which ones.
  - Making scheduling decisions at various points in the network.

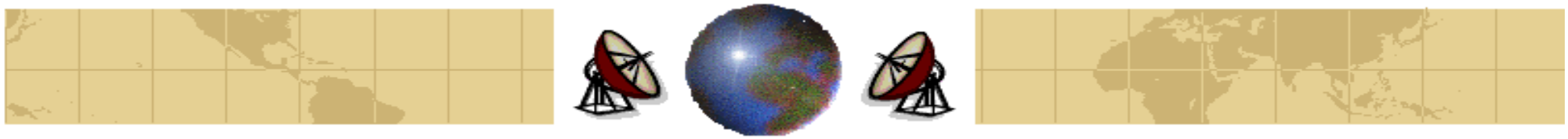


General Principles of Congestion Control

II. Closed Loop Solutions:

- Open loop solutions are based on the concept of a feedback loop. This approach has three parts when applied to congestion control:

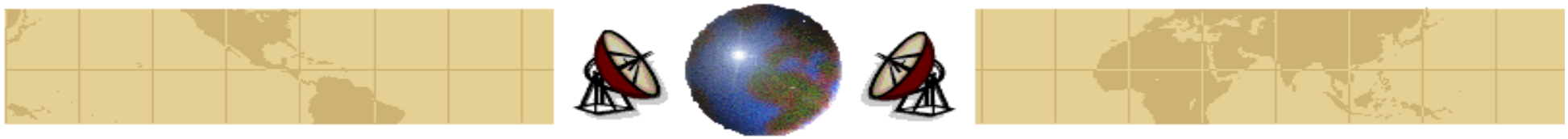
1. *Monitor the system:* Detect when and where congestion occurs.
2. *Pass information to where action can be taken.*
3. *Adjust system operation* to correct the problem



Closed Loop Solutions

1. Monitor the system to detect when and where congestion occurs:

- Various metrics can be used to monitor the subnet for congestion such as:
  - The *percentage of all packets discarded* for lack of memory 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*.

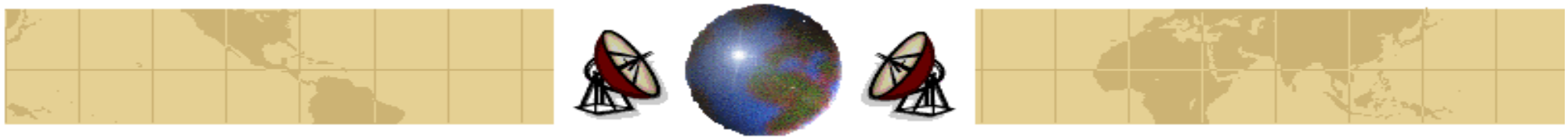


Closed Loop Solutions

2. Transfer the information about congestion from the point where it is detected to places where action can be taken:

- The router, detecting the congestion, sends a "*warning*" packet to the traffic source or sources.
- Other possibility is *to reserve a bit or field in every packet* for routers to fill in whenever congestion gets above some threshold level.
- Another approach is to have hosts or routers *send probe packets out periodically* to explicitly ask about congestion.

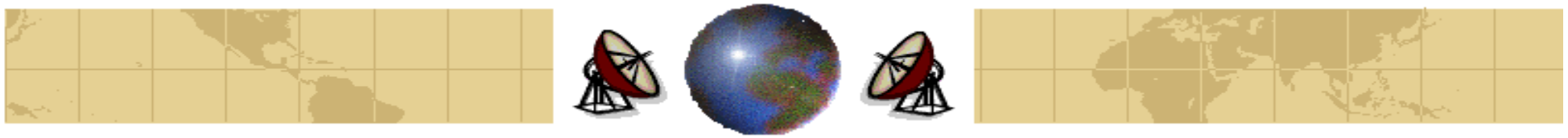
3. Adjust system operation to correct the problem using the appropriate congestion control.



Closed Loop Solutions

The *closed loop algorithm* can be either:

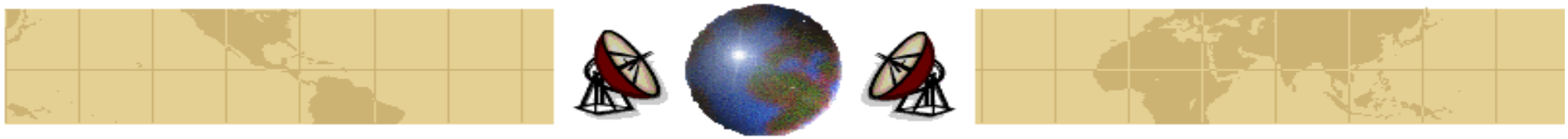
- **1. Explicit:** Packets are sent back from the point of congestion to warn the source.
- **2. Implicit:** The source deduces the existence of congestion by making local observations, such as the time needed for acknowledgements to come back.



Congestion Prevention Policies

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

Policies that affect congestion.



Congestion Prevention Policies – Data Link Layer

1. Retransmission Policy:

- It deals with *how fast a sender times out* and *what it transmit upon timeout*.
- A jumpy sender that times out quickly and retransmits all outstanding packets using **go-back *N*** will put heavier load on the system than the sender uses **selective repeat**.

2. Out-of-Order Caching Policy:

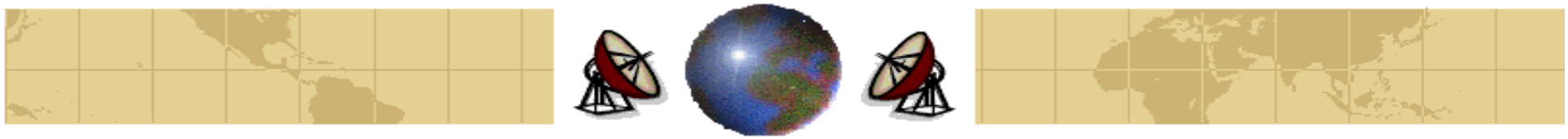
- If the receivers routinely *discard all out-of-order packets*, these packets will have to be *retransmitted again* later, creating extra load.

3. Acknowledgement Policy:

- If each packet is *acknowledged immediately*, the acknowledgement packets generate extra load. However, if acknowledgments are saved up to *piggback* onto reserve traffic, extra timeouts and retransmissions may result.

4. Flow Control Policy:

- A tight control scheme reduces the data rate and thus *helps fight congestion*.



Congestion Prevention Policies – Network Layer

1. Virtual Circuit versus Datagram.

2. Packet Queueing and Service Policy:

- Router may have *one queue per input line*, *one queue per output line*, *or both*.
- It also relates to the order packets are processed.

3. Packet Discard Policy:

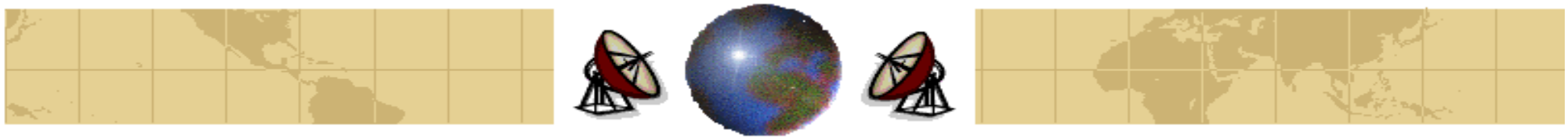
- It tells *which packet is dropped* when there is no place.

4. Routing Policy:

- Good routing algorithm *spreads the traffic over all the lines*.

5. Packet Lifetime Management:

- It deals with *how long a packet may live* before being discarded.
  - If it is *too long*, lost packets waste the network's bandwidth.
  - If it is *too short*, packets may be discarded before reaching their destination.



Congestion Prevention Policies – Transport Layer

1. Retransmission Policy

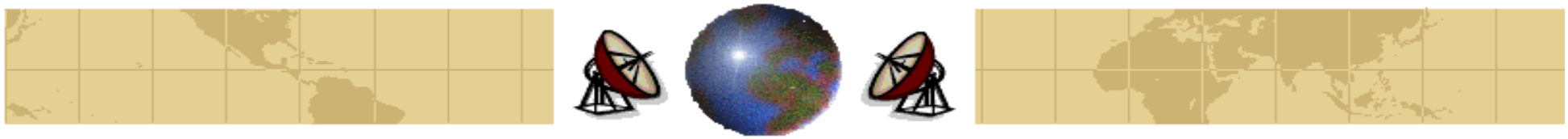
2. Out-of-Order Caching Policy

3. Acknowledgement Policy

4. Flow Control Policy

5. Timeout Determination:

- Determining the *timeout interval* is harder because the transit time across the network is less predictable than the transit over a wire between two routers.
  - If it is *too short*, extra packets will be sent unnecessary.
  - If it is *too long*, congestion will be reduced, but the response time will suffer whenever packet is lost.



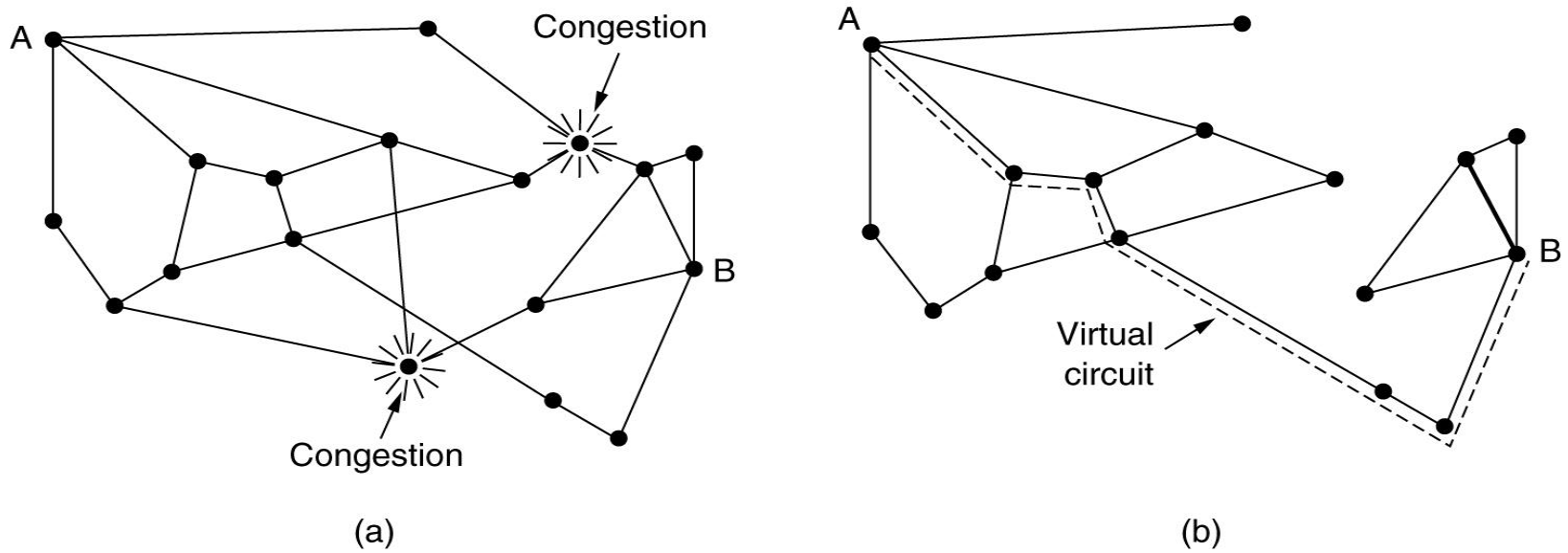
Congestion Control in Virtual Circuit Subnets

1. Admission Control:

- 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.
- In a telephone system, when a switch gets overloaded, it also practices admission control, by not giving dial tones.
- This approach is *crude*, but it is *simple and easy to carry out*.

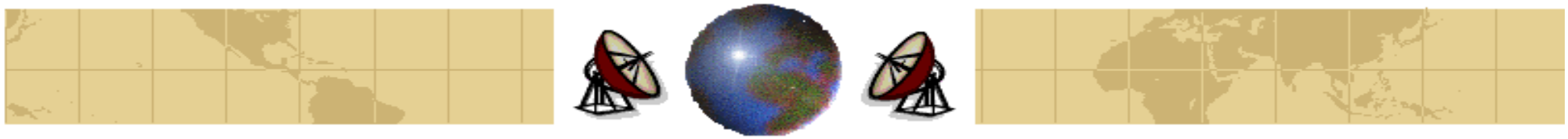
Congestion Control in Virtual Circuit Subnets

2. Routing All New Virtual Circuits around the Problem Areas:



(a) A congested subnet.

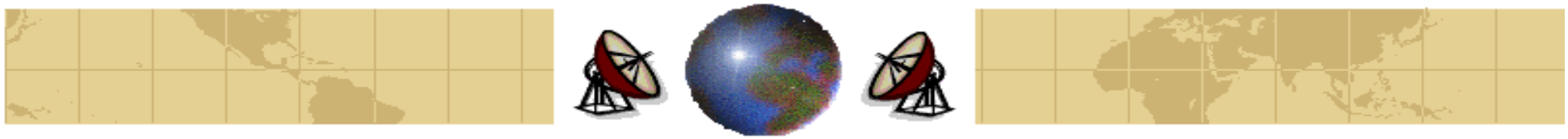
(b) A redrawn subnet, eliminates congestion and a virtual circuit from A to B.



Congestion Control in Virtual Circuit Subnets

3. Negotiating for 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 (QoS) required*, and other parameters.
- To keep its part of the agreement, the subnet will *reserve resources* along path when the circuit is set up.
- These resources can include *table and buffer space in the routers* and *bandwidth in the lines*.
- The drawback of this method that it tends to *waste resources*. For example, if six virtual circuits that might use 1 Mbps all pass through the same physical 6-Mbps line, the line has to marked as full, even though it may rarely happen that all six virtual circuits are transmitting at the same time.



Choke Packets

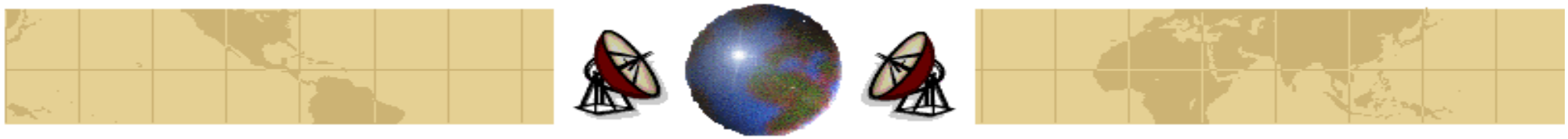
- **Choke packets** are used in both **virtual circuit** and **datagram subnets**.
- Each router can easily monitor the *utilization of its output lines*. For example, it can associate with each line a real variable, *u*, whose value, between 0.0 and 1.0, reflects the recent utilization of that line.

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

where *f* is the *instantaneous line utilization* that can be made periodically, and its value either 0 or 1, and

a is a constant that determines how fast the router forgets recent history.

- Whenever *u* moves *above the threshold*, the output line enters a *"***warning***"* state.



Choke Packets

- Each newly arriving packet is checked to see if its output line is in **warning** state. If so, *the router sends a choke packet back to the source host*. The *original packet is tagged* so that it will not generate any more choke packets further along the path, and is then forwarded in the usual way.

- When *the source gets the choke packet*, it is required to reduce the traffic sent to the specified destination by *X* percent. The source also *ignores the choke packets* referring to that destination for a fixed time interval. After that period is expired, if another choke packet arrives, the host *reduces the flow more*, and so on.

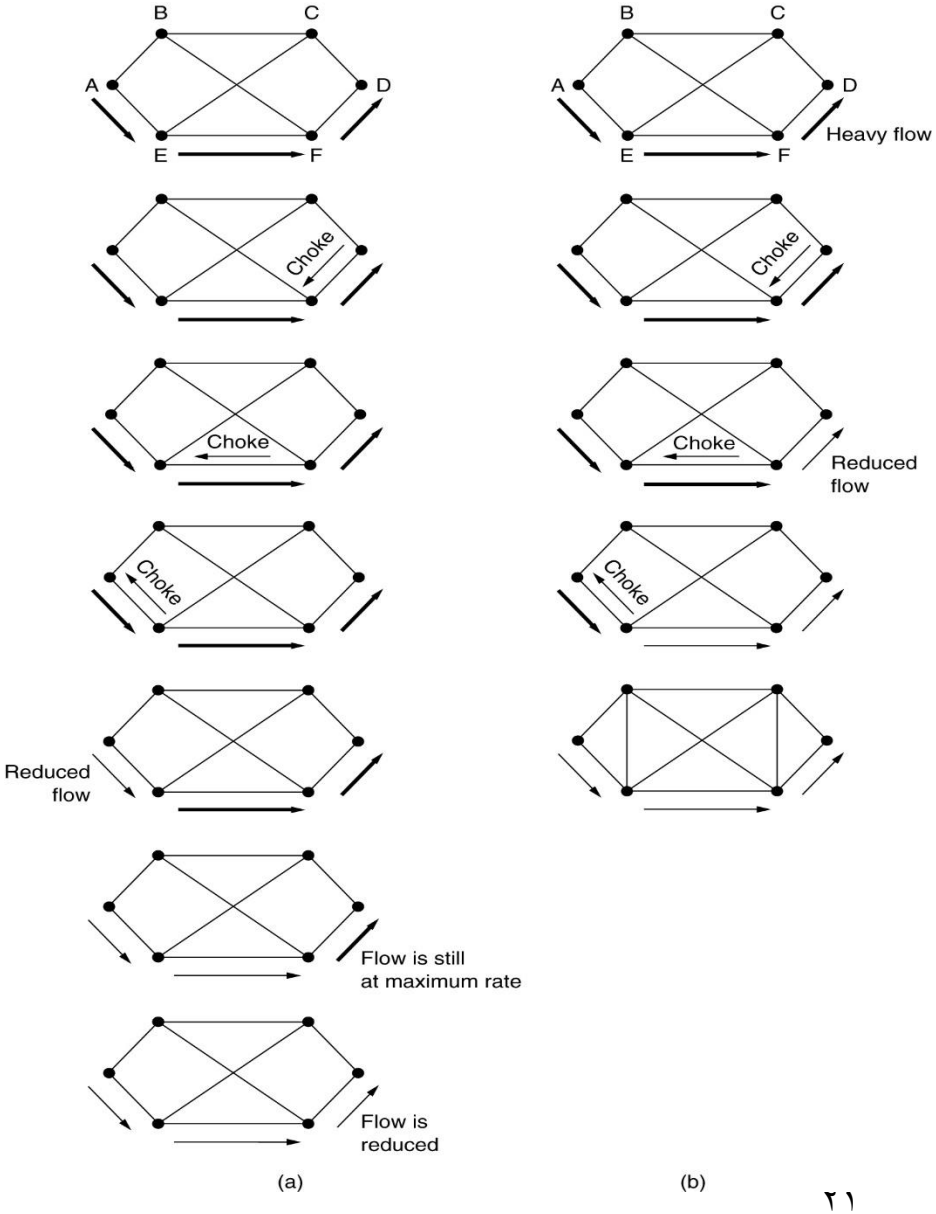
- Router can maintain several thresholds. Depending on which threshold has been crossed, the choke packet can contain a **mild** warning, a **stern** warning, or an **ultimatum**.

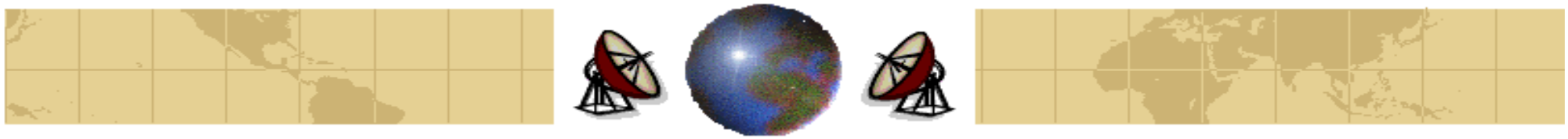
- Another variation is to use queue lengths or buffer utilization instead of line utilization.

Hop-by-Hop Choke Packets

(a) A choke packet that affects only the source.

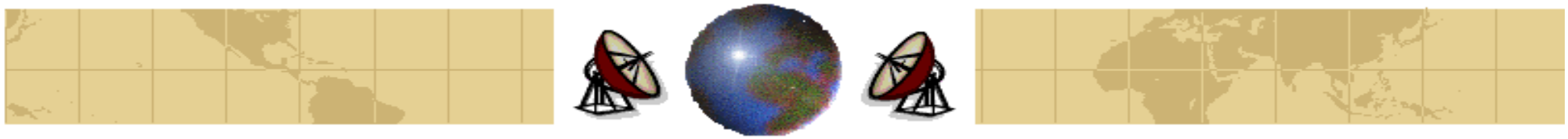
(b) A choke packet that affects each hop it passes through.





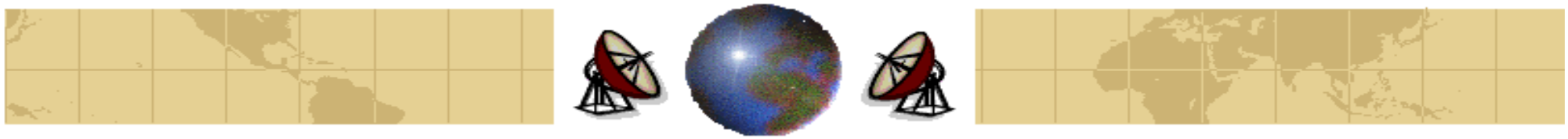
Hop-by-Hop Choke Packets

- For long distances, sending a *choke packet* to the source is *not efficient* because the choke packet may take long time to the source and during that period the source may send a lot of packets.
- An alternative approach to have the choke packets *take effect at every hop* it passes though.
- This hop-by-hop scheme provides *quick relief at the point of congestion* at the price of using more buffers upstream.



Load Shedding

- The basic idea of **load shedding** scheme is that when routers have a lot of packets that they cannot handle, they just throw them away.
- A router drowning in packets can just pick packets *at random* to discard, 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 the new ones* if the receiver routinely discards out-of-order packets.
- For **Multimedia**, a *new packet is more important than an old one*.
- For **compressed video**, a full frame is first sent. Then the subsequent frames as differences from the last full frame are sent. *Dropping a packet that is part of a difference is preferable to dropping one that is part of a full frame*.

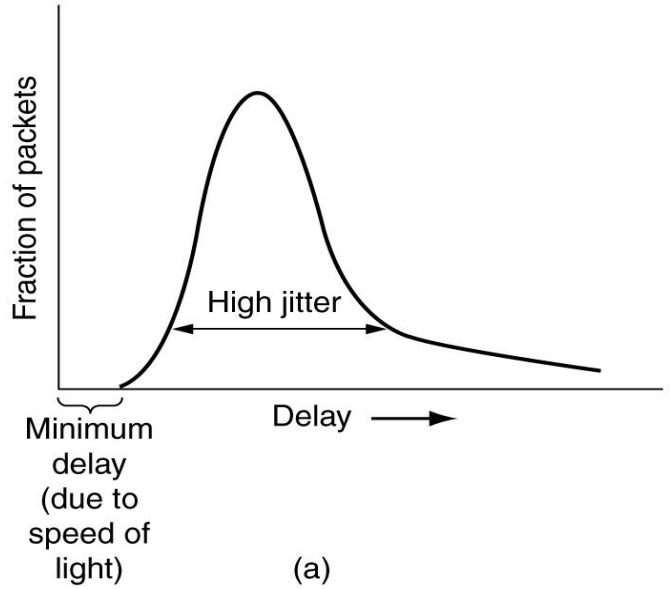


Load Shedding

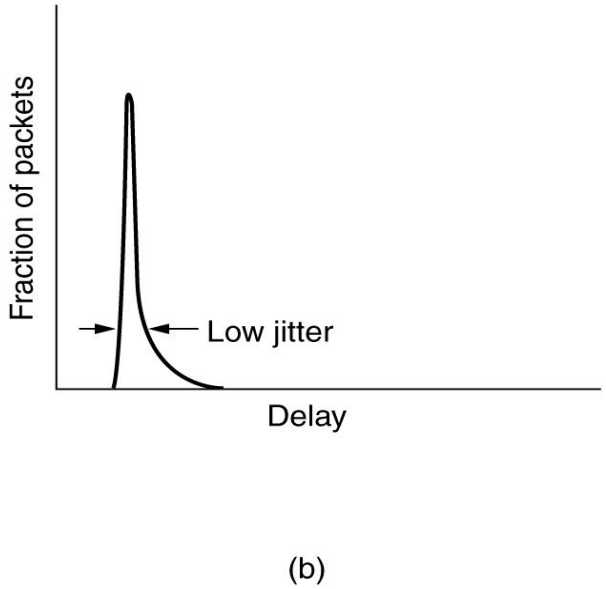
- For **documents the containing ASCII test and pictures**, *losing a line of pixels in some images is far less damaging than losing a line of readable text*.
- To implement an **intelligent discard policy**, *applications must mark their packets in priority classes to indicate how important they are*.
- Another option is *to allow hosts to exceed the limit specified in the agreement negotiated when the virtual circuit is set up*, but subject to condition that **all excess traffic is marked as low priority**.

Jitter Control

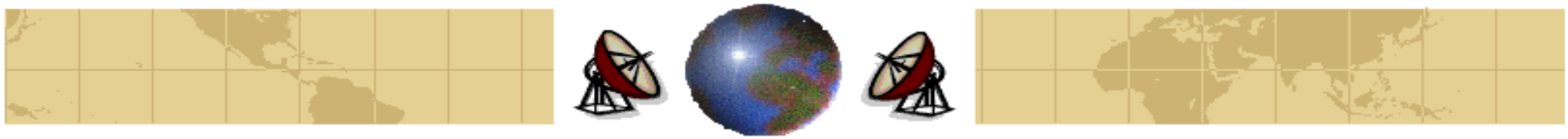
- **Jitter** is the *amount of variation in the end-to end packet transit time*.
- For applications such as audio and video streaming, it does not matter much if the packets take 20 msec or 30 msec to be delivered, as long as the transmit time is constant.



(a) High jitter.

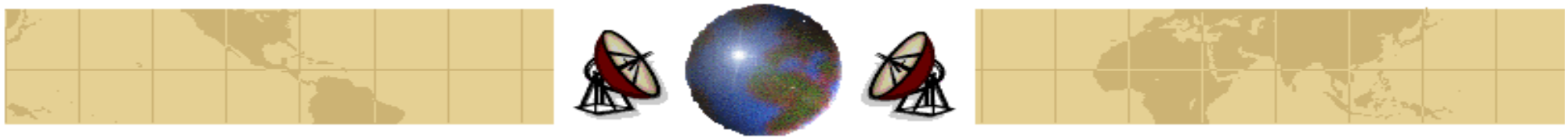


(b) Low jitter.



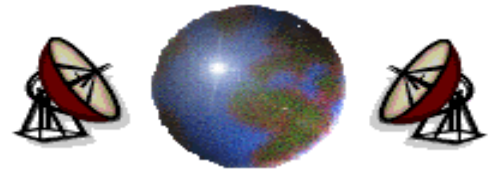
Jitter Control

- 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**. The information is stored in the packet and updated at each hop.
- If the packet is *ahead of schedule*, it is held long enough to get it back on schedule.
- If the packet is *behind schedule*, the router tries to get it out quickly. 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 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 from the network. However, for *real-time* applications such as Internet telephony and videoconferencing, the *buffering is not acceptable*.



Quality of Service

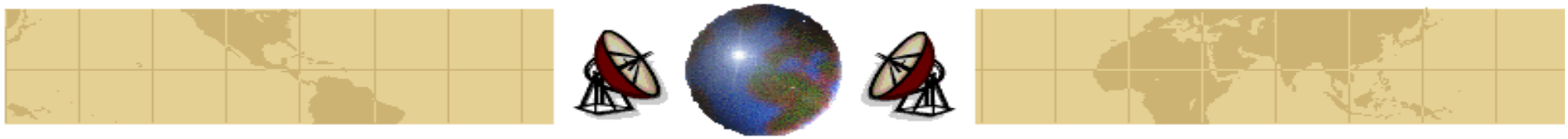
- Requirements
- Techniques for Achieving Good Quality of Service
- Integrated Services
- Differentiated Services
- Label Switching and MPLS



Requirements

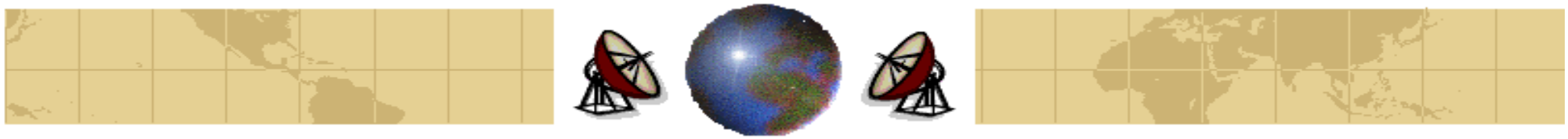
Application	Reliability	Delay	Jitter	Bandwidth
E-mail	High	Low	Low	Low
File transfer	High	Low	Low	Medium
Web access	High	Medium	Low	Medium
Remote login	High	Medium	Medium	Low
Audio on demand	Low	Low	High	Medium
Video on demand	Low	Low	High	High
Telephony	Low	High	High	Low
Videoconferencing	Low	High	High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

How stringent the quality-of-service requirements are.



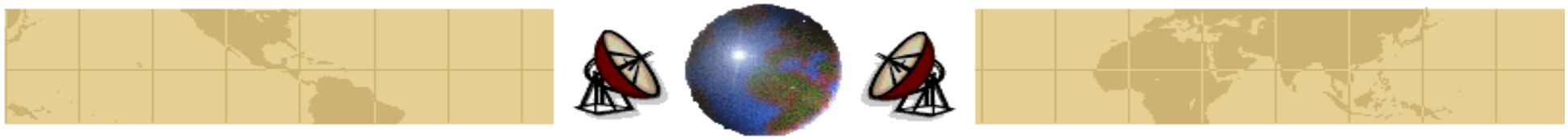
Requirements

- ATM networks classify flows in *five broad categories* with respect to their QoS demands as follows:
  - **Constant bit rate** (e.g., telephony).
  - **Real-time variable bit rate** (e.g., compressed videoconferencing).
  - **Non-Real-time variable bit rate** (e.g., watching movie over the Internet).
  - **Available bit rate** (e.g., file transfer).
  - **Unspecified bit rate** (e.g., Background file transfer).



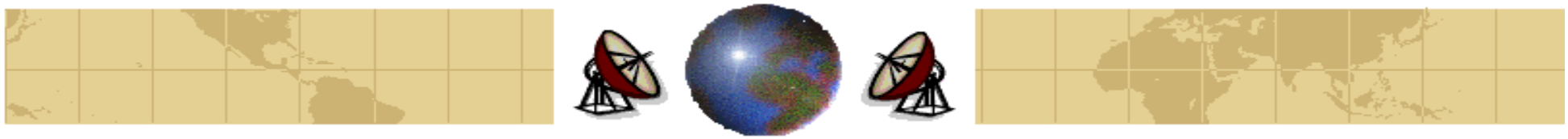
Requirements

Service Characteristic	CBR	RT-VBR	NRT-VBR	ABR	UBR
Bandwidth guarantee	Constant	Average	Average	Minimum	No
Suitable for real-time traffic	Yes	Yes	No	No	No
Suitable for bursty traffic	No	YES	Yes	Yes	Yes
Feedback about congestion	No	No	No	Yes	No



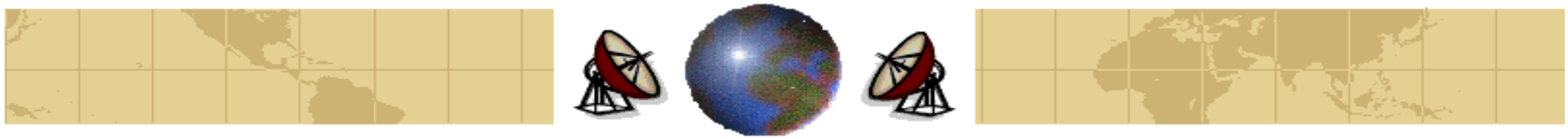
Techniques for Achieving Good Quality of Service

1. Overprovisioning
2. Buffering
3. Traffic Shaping
    1. Leaky Bucket Algorithm
    2. Token Bucket Algorithm
4. Resource reservation
5. Admission Control
6. Proportional Routing
7. Packet Scheduling



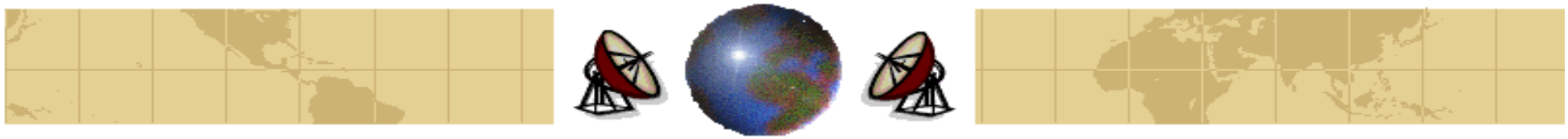
1. Overprovisioning

- To provide so much router capacity, buffer space, and bandwidth that the packets just fly through easily!
- There is simply *so much capacity available* there that demand can always be met. But it is *expensive* solution,
- As times 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.

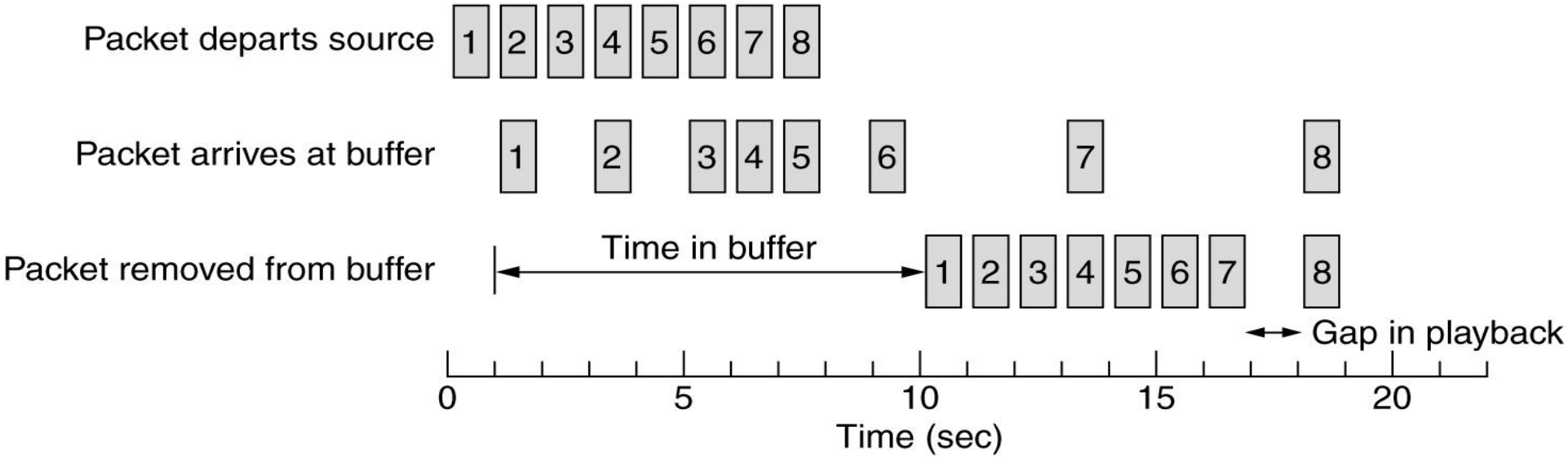


2. Buffering

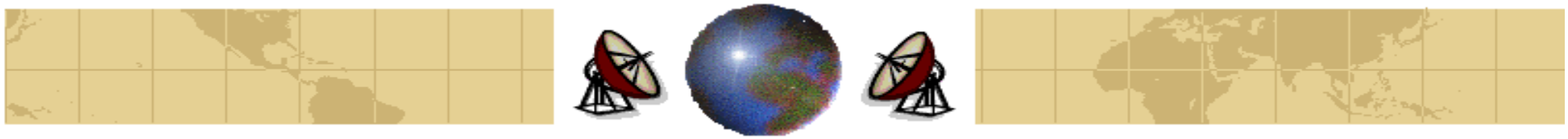
- Buffering *smoothes out the jitter*.
- For *audio and video on-demand*, jitter is the main problem, so this technique helps a lot.
- In the following figure, a stream of packets being *delivered with substantial jitter*. As the packets arrive, they are *buffered* on the client machine.
- When *playback* begins, packets 1 through 6 have been buffered so they can be removed from the buffer at uniform rate for *smooth play*. Unfortunately, packet 8 has been delayed so it is not available when its play slot comes up, so play back must stop until it arrives, creating an *annoying gap*.
- This problem can be alleviated by *delaying the starting time even more*, although doing so requires a larger buffer.
- Commercial Web sites use players that buffer for about 10 seconds before starting to play.



2. Buffering



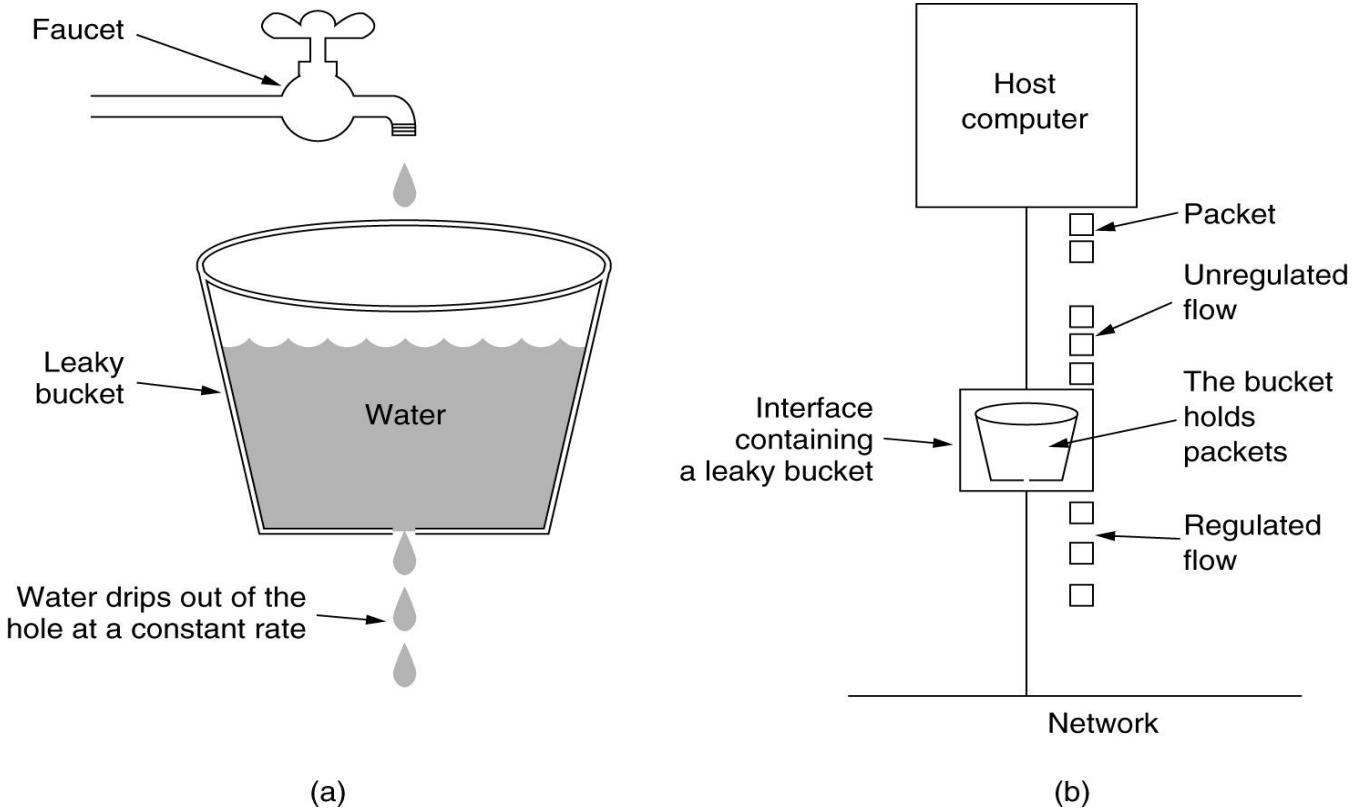
Smoothing the output stream by buffering packets.



3. Traffic Shaping

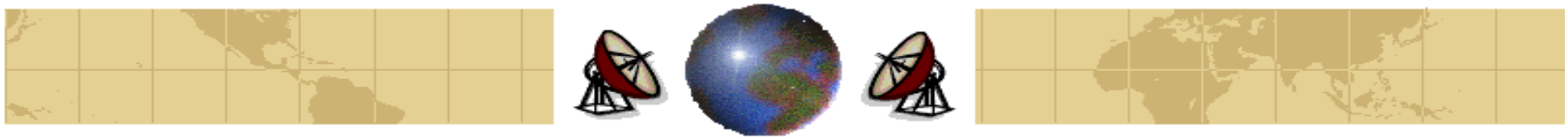
- **Bursty traffic** is one of the main causes of congestion. If hosts could be made to transmit at a uniform rate, congestion would be less common.
- **Traffic shaping** is about *regulating the average rate of data transmission*. It smoothes out the traffic on the server-side, rather than on the client-side (e.g., videoconferencing requires traffic shaping, but not buffering).
- When a virtual-circuit is set up, the user and the subnet (carrier) agree on a certain traffic shape for that circuit. This agreement is called **traffic contract or service level agreement (SLA)**.
- As long as the customer sends packets according to the agreed upon contract, the *carrier promises to deliver them all in timely fashion*.
- Traffic shaping is very important for *real-time data*, such as audio and video.
- **Traffic policing** is to monitor the traffic flow.
- Traffic shaping and policing mechanisms are easier with virtual circuit subnets than with datagram subnets.

3.1 Leaky Bucket Algorithm



(a) A leaky bucket with water.

(b) a leaky bucket with packets.



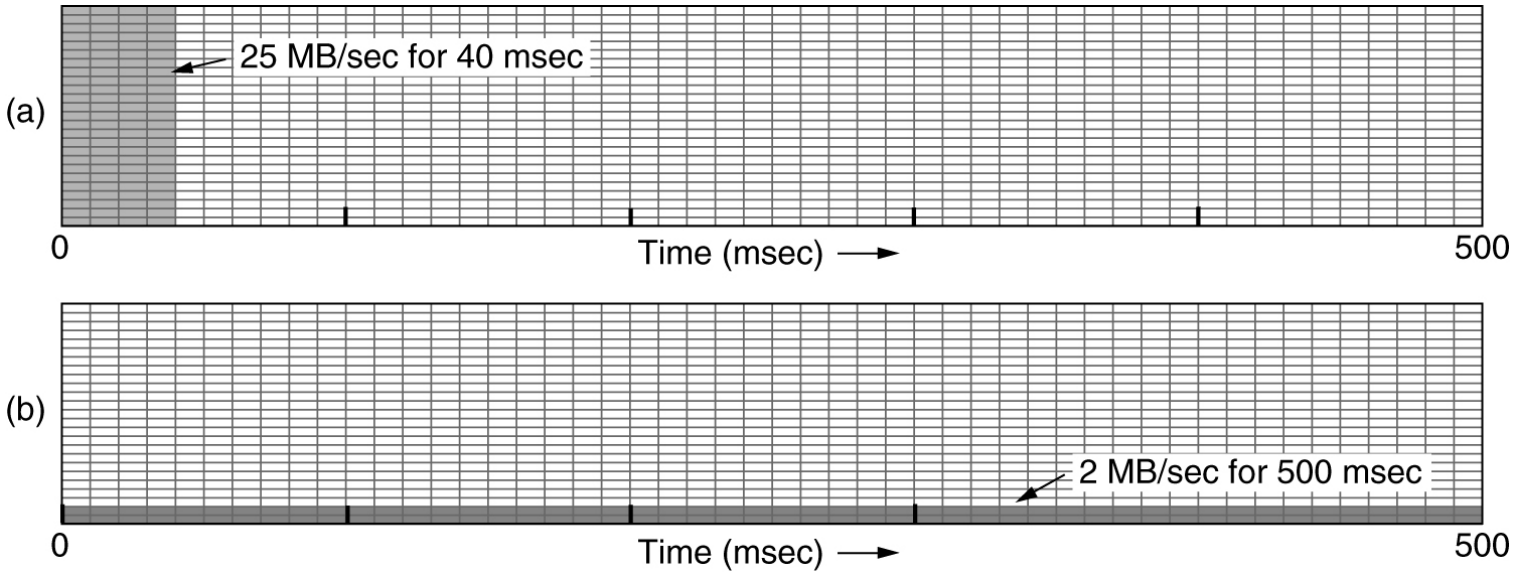
3.1 Leaky Bucket Algorithm

- Imagine a *bucket with a small hole in the bottom*. No matter at what rate water enters the bucket, the **outflow is at a constant rate, ρ,** when there is any water in the bucket, and zero when bucket is empty. Also, once the bucket is full, any additional water entering it spills over the sides and is lost.
- The same idea can be applied to packets. *Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal 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 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.

3.1 Leaky Bucket Algorithm

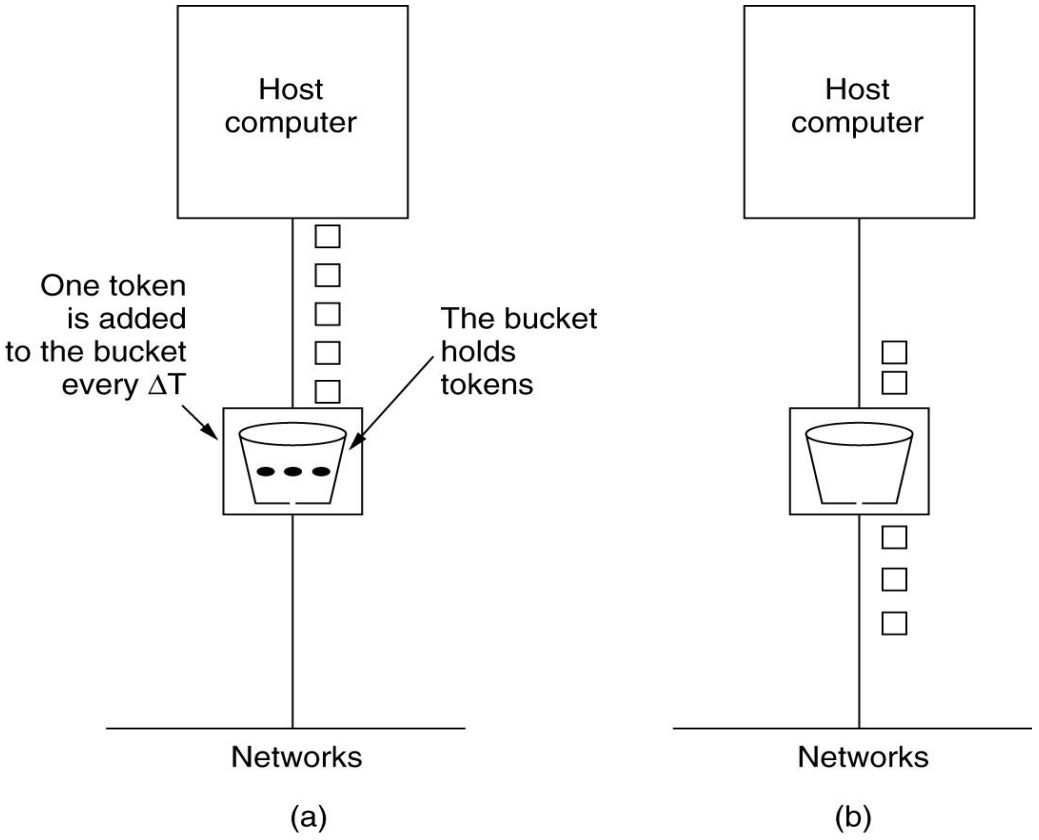
Example:

- The input to the leaky bucket running at 25 MByte/sec for 40 msec. However, the output draining out at uniform rate of 2 MByte/sec for 500 msec.

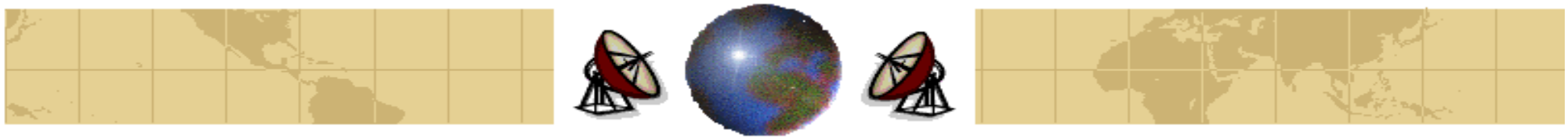


(a) Input to a leaky bucket.
(b) Output from a leaky bucket.

3.2 Token Bucket Algorithm

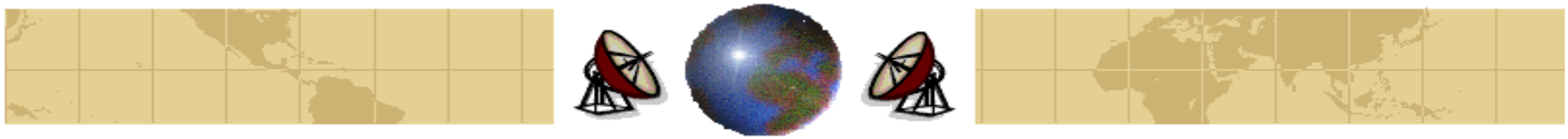


(a) Before (b) After



3.2 Token Bucket Algorithm

- It is better to allow the output to speed up somewhat when large burst arrives, so a more flexible algorithm is needed. 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.
- The implementation of the basic token bucket algorithm is *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 is 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.
- The token bucket algorithm allows saving, up to the maximum size of the bucket, n. *This property means that bursts up to n packets can be sent at once, allowing some burstiness in the output stream and giving faster response to sudden burst input.*



3.2 Token Bucket Algorithm

- The token bucket algorithm throws away tokens when the bucket fills up but never discard packets. In contrast, the leaky bucket algorithm discards packets when the bucket fills up.
- If we call the burst length S sec, the token bucket capacity C bytes, the token arrival rate ρ bytes/sec, and the maximum output rate M bytes/sec, then the output burst contains a maximum of $C + \rho S$ bytes.
- We know that the number of bytes in a maximum-speed burst of length S seconds is MS.
- Hence we have:

$$C + \rho S = MS$$

- We can solve this equation to get:

$$S = \frac{C}{M - \rho}$$

3.2 Token Bucket Algorithm

Example:

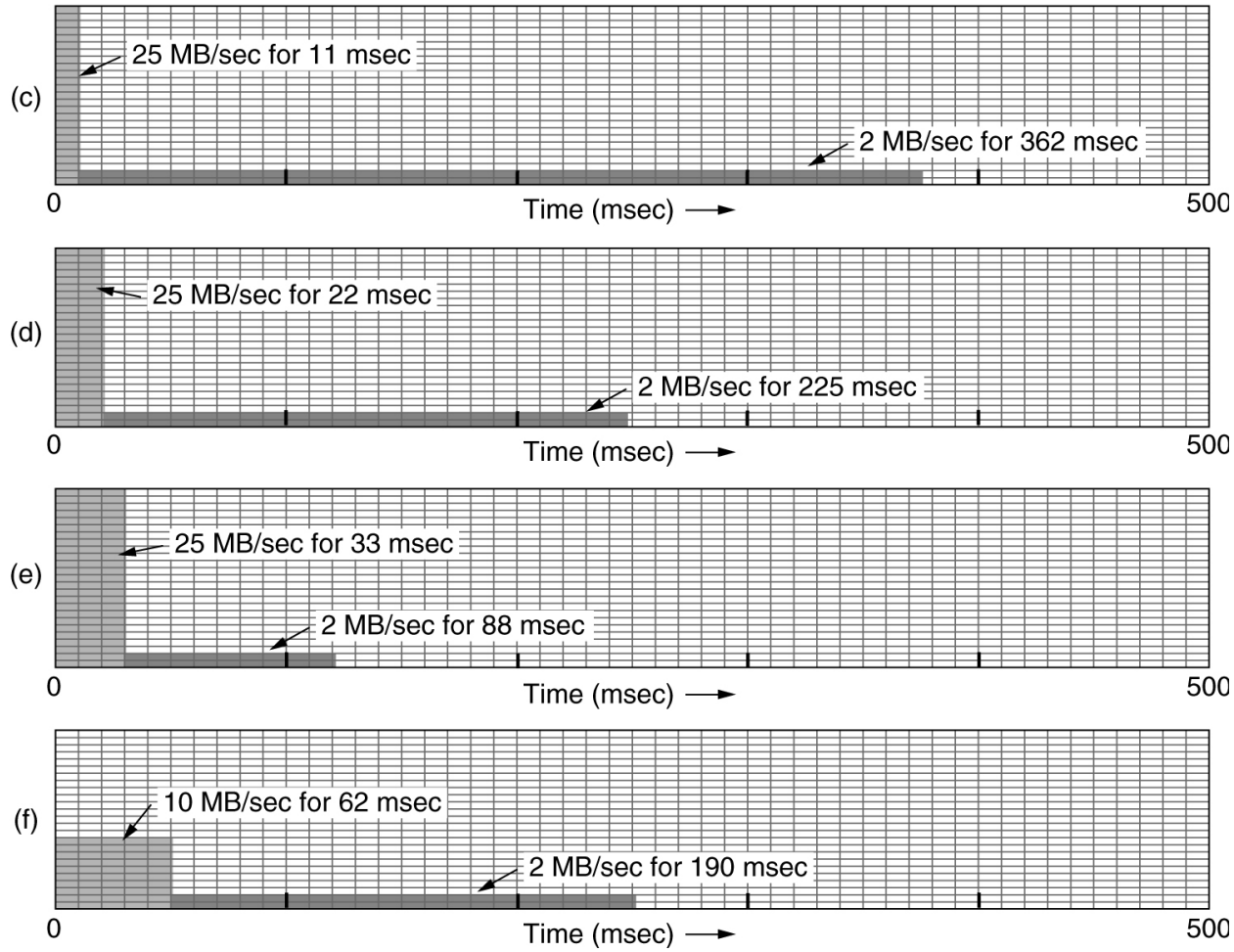
Output from a token bucket with capacities of:

(c) 250 KB,

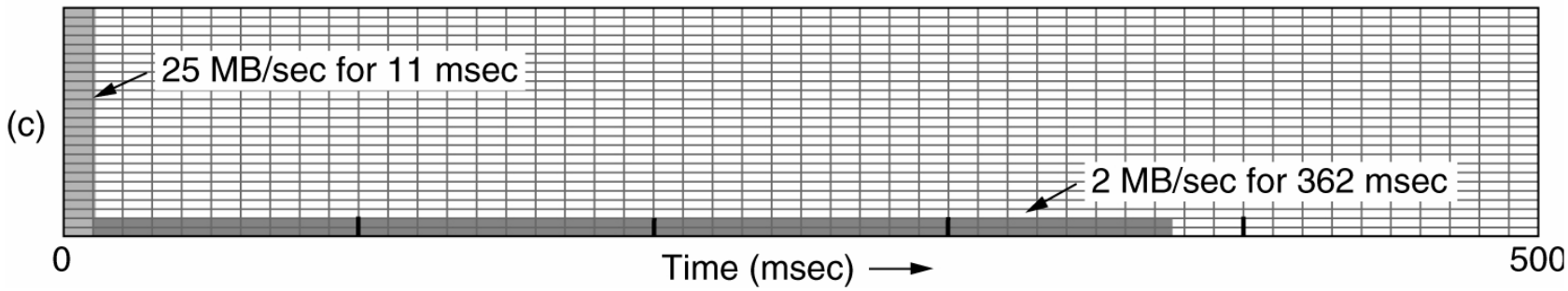
(d) 500 KB,

(e) 750 KB,

(f) Output from a 500KB token bucket feeding a 10-MB/sec leaky bucket.



3.2 Token Bucket Algorithm



C=250 KB

$$S = \frac{C}{M - \rho}$$

$$S = \frac{250}{25 - 2}$$

$$\Rightarrow S = 10.8696\ msec$$

$$\text{Number of Transferred bytes during the burst} = 0.0108696 \times 25 \times 10^6 = 271739\ bytes$$

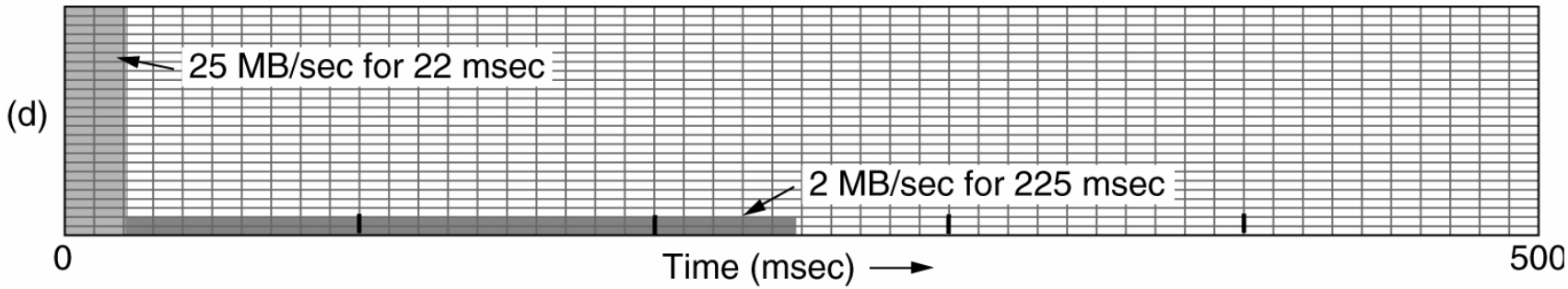
$$\Rightarrow \text{Remaining number of bytes} = 1000000 - 271739 = 728260\ bytes$$

$$\text{Time to transmit the remaining file at uniform rate} = \frac{728260}{2 \times 10^6} = 364.13\ msec$$

$$\text{Total Time to transmit the file} = 10.8696 + 364.13 = 375\ msec$$

$$\text{Time to refill the token bucket} = \frac{C}{\rho} = \frac{250000}{2 \times 10^6} = 125\ msec$$

3.2 Token Bucket Algorithm



C=500 KB

$$S = \frac{C}{M - \rho}$$

$$S = \frac{500}{25 - 2}$$

$$\Rightarrow S = 21.739\ m\text{sec}$$

$$\text{Number of Transferred bytes during the burst} = 0.021739 \times 25 \times 10^6 = 543478\ bytes$$

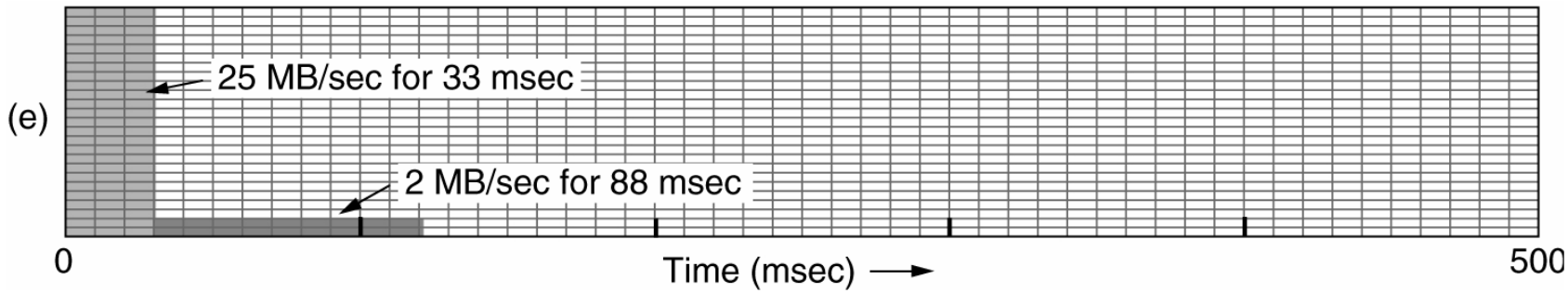
$$\Rightarrow \text{Remaining number of bytes} = 1000000 - 543478 = 456521\ bytes$$

$$\text{Time to transmit the remaining file at uniform rate} = \frac{456522}{2 \times 10^6} = 228.261\ m\text{sec}$$

$$\text{Total Time to transmit the file} = 21.739 + 228.261 = 250\ m\text{sec}$$

$$\text{Time to refill the token bucket} = \frac{C}{\rho} = \frac{500000}{2 \times 10^6} = 250\ m\text{sec}$$

3.2 Token Bucket Algorithm



C=750 KB

$$S = \frac{C}{M - \rho}$$

$$S = \frac{750}{25 - 2}$$

$$\Rightarrow S = 32.609\ m\text{sec}$$

$$\text{Number of Transferred bytes during the burst} = 0.032609 \times 25 \times 10^6 = 815217\ bytes$$

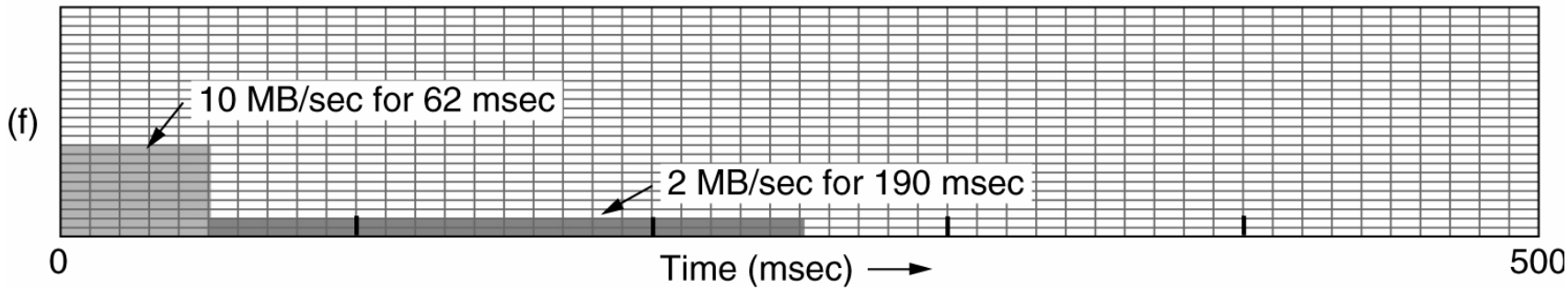
$$\Rightarrow \text{Remaining number of bytes} = 1000000 - 815217 = 184783\ bytes$$

$$\text{Time to transmit the remaining file at uniform rate} = \frac{184783}{2 \times 10^6} = 92.391\ m\text{sec}$$

$$\text{Time to transmit the file} = 32.609 + 92.391 = 125\ m\text{sec}$$

$$\text{Time to refill the token bucket} = \frac{C}{\rho} = \frac{7500000}{2 \times 10^6} = 375\ m\text{sec}$$

3.2 Token Bucket Algorithm



Output from a C=500KB token bucket feeding a ρ_2=10 MB/sec leaky bucket

$$S = \frac{C}{\rho_2 - \rho_1}$$

$$S = \frac{500}{10 - 2}$$

$$\Rightarrow S = 62.5\, msec$$

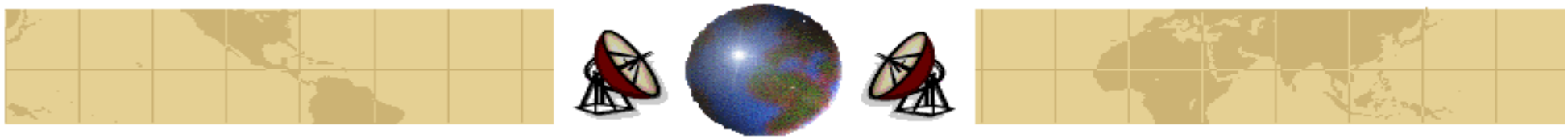
$$\text{Number of Transferred bytes during the burst} = 0.0625 \times 10 \times 10^6 = 650000\ bytes$$

$$\Rightarrow \text{Remaining number of bytes} = 1000000 - 650000 = 375000\ bytes$$

$$\text{Time to transmit the remaining file at uniform rate} = \frac{375000}{2 \times 10^6} = 187.5\, msec$$

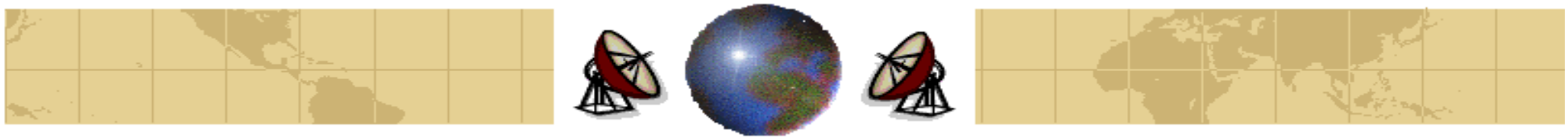
$$\text{Time to transmit the file} = 62.5 + 187.5 = 250\ msec$$

$$\text{Time to refill the token bucket} = \frac{C}{\rho} = \frac{500000}{2 \times 10^6} = 250\ msec$$



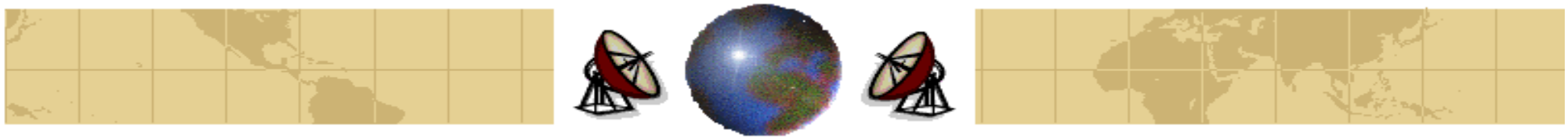
4. Resource Reservation

- Being able to regulate the shape of the offered traffic is a good start to guaranteeing the quality of Service.
- Once we have a specific route, it becomes possible to reserve resources along the path to make sure the needed capacity is available.
- Three different resources an be reserved:
  - Bandwidth.
  - Buffer space.
  - CPU cycles.



5. Admission Control

- Now the incoming traffic is well-shaped and follow a single route in which capacity can be reserved in advance on the routers along the path.
- When a new flow is offered to a router, it has to decide, based on its capacity and haw many connections 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 parameters because:
  - Some application may know the bandwidth requirements, but *few know about buffers or CPU cycles*.
  - Some applications are *tolerant to missed deadlines* than others.
  - Some applications may be willing to *bargain about the flow specifications* and others may not.

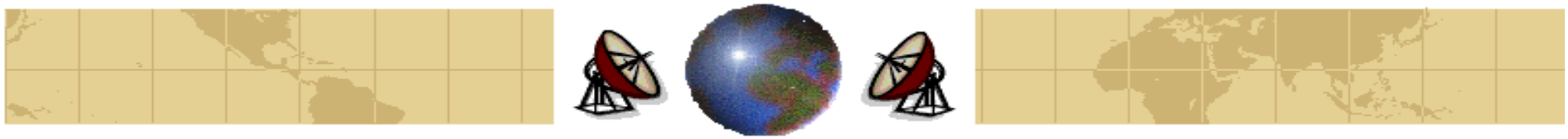


Flow Specification

- Traffic shaping is most effective when the sender, the receiver, and subnet all agree to it. To get agreement, it is necessary to specify the traffic pattern in a precise way. This agreement is called a flow specification.
- A flow specification consists of a data structure that describes both the pattern of the injected traffic and the quality of service desired by the applications.

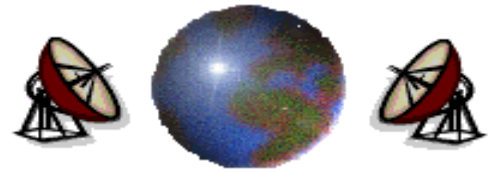
Parameter	Unit
Token bucket rate	Bytes/sec
Token bucket size	Bytes
Peak data rate	Bytes/sec
Minimum packet size	Bytes
Maximum packet size	Bytes

An example of flow specification



Flow Specification

- Before the connection is established or before a sequence of datagrams are sent, the source gives the flow specification to the subnet for approval. The subnet can either accept it, reject it, or come back with counterproposal (“I cannot give you 100 msec average delay; can you live with 150 msec?”). Once the sender and subnet have struck a deal, the sender can ask the receiver if it agrees.



Flow Specification

First: Characteristics of the Input Traffic

1. Maximum Packet Size (bytes).

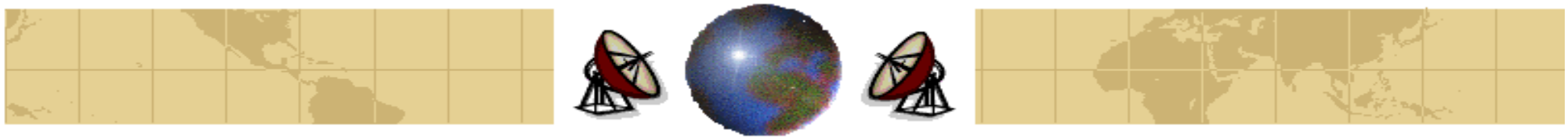
2. Token bucket rate (bytes/sec).

3. Token bucket size (byte):

- If the token bucket rate is ρ bytes/sec and the bucket size is C bytes, then in the time interval Δt, the maximum number of bytes that may be sent is $C + \rho \Delta t$.

4. Maximum Transmission rate:

- It is the top rate the host is capable of producing under any condition and implicitly specifies the shortest time interval in which the token bucket could be emptied.



Flow Specification

Second: Service Desired

1. Loss sensitivity (bytes).

2. Loss Interval (msec):

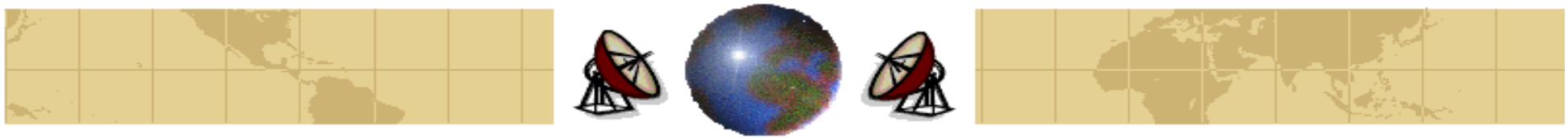
- Loss sensitivity and loss interval parameters represent the numerator and denominator of fraction giving the maximum acceptable loss (*e.g.*, 1 byte/hour).

3. Burst loss sensitivity (Packets):

- It tells how many consecutive lost packets can be tolerated.

4. Minimum delay noticed (msec):

- It tells how long a packet can be delayed without the application noticed.



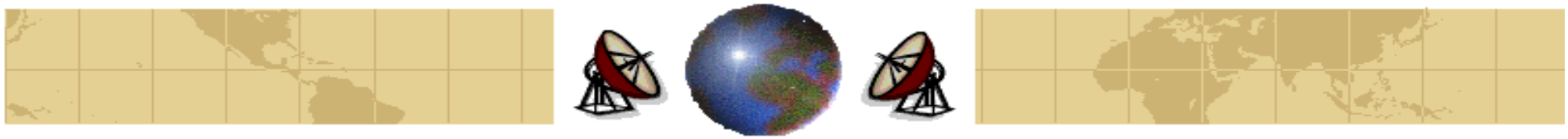
Flow Specification

5. Maximum delay variation (msec):

- Some applications are not sensitive to the actual delay but are highly sensitive to the jitter, that is, the amount of variation in the end-to-end packet transit time.
- It is two times the number of microseconds a packet's delay may vary from the average. For example, a value of 2000 microseconds means that a packet may be up to 1 msec early or late, but not more.

6. Quality of Guarantee:

- It indicates whether or not the application really means it.

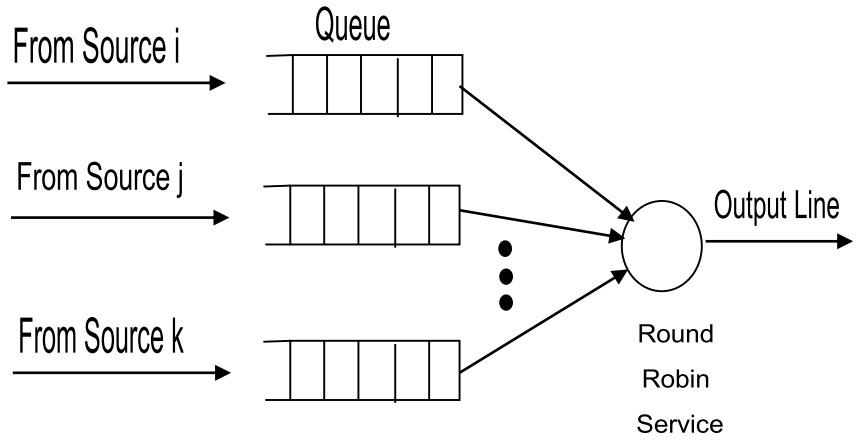


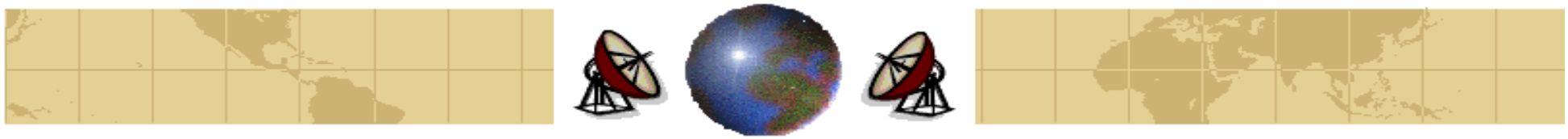
6. Proportional Routing

- 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 of routing algorithms is to *split the traffic for each destinations over the multiple paths*.
- A simple method is to divide the traffic equally or in proportion to the capacity of the outgoing links.

7. Packet Scheduling

- The problem of using chock packets is that the action to be taken by the source hosts is *voluntary*.
- To get around this problem, a **fair queueing algorithm** is proposed.
- Each router has multiple queues for each output line, one for each source. When the *line becomes idle*, the *router scans the queues round robin*, taking the first packet on the next queue. Some ATM switches use this algorithm.

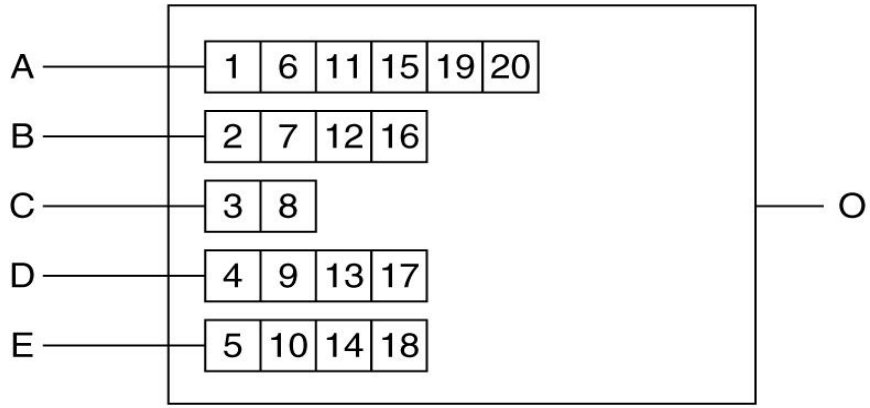




7. Packet Scheduling

- The *fair queueing algorithm* has a problem: it gives more bandwidth to hosts that use large packets than to hosts that use small packets.
- It is suggested to use a *byte-by-byte round robin*, instead of packet-by-packet round robin. It *scans* the queues repeatedly, byte-by-byte, until each packet is finished. The packets are then *sorted* in order of their finishing and sent in that order.
- The problem of this algorithm is that it *gives all hosts the same priority*. It is desirable to give file servers more bandwidth than clients. The modified algorithm is called **weighted fair queueing**.

7. Packet Scheduling



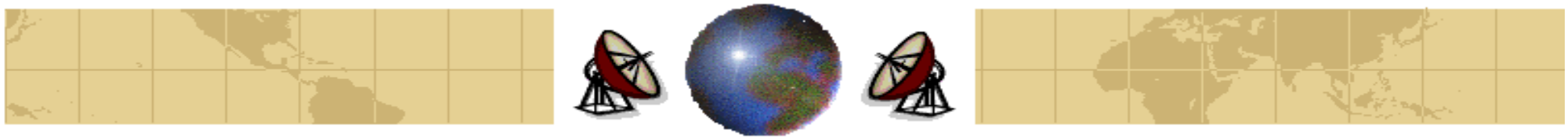
(a)

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

(b)

(a) A router with five packets queued for line O

(b) Finishing times for the five packets



Integrated Services

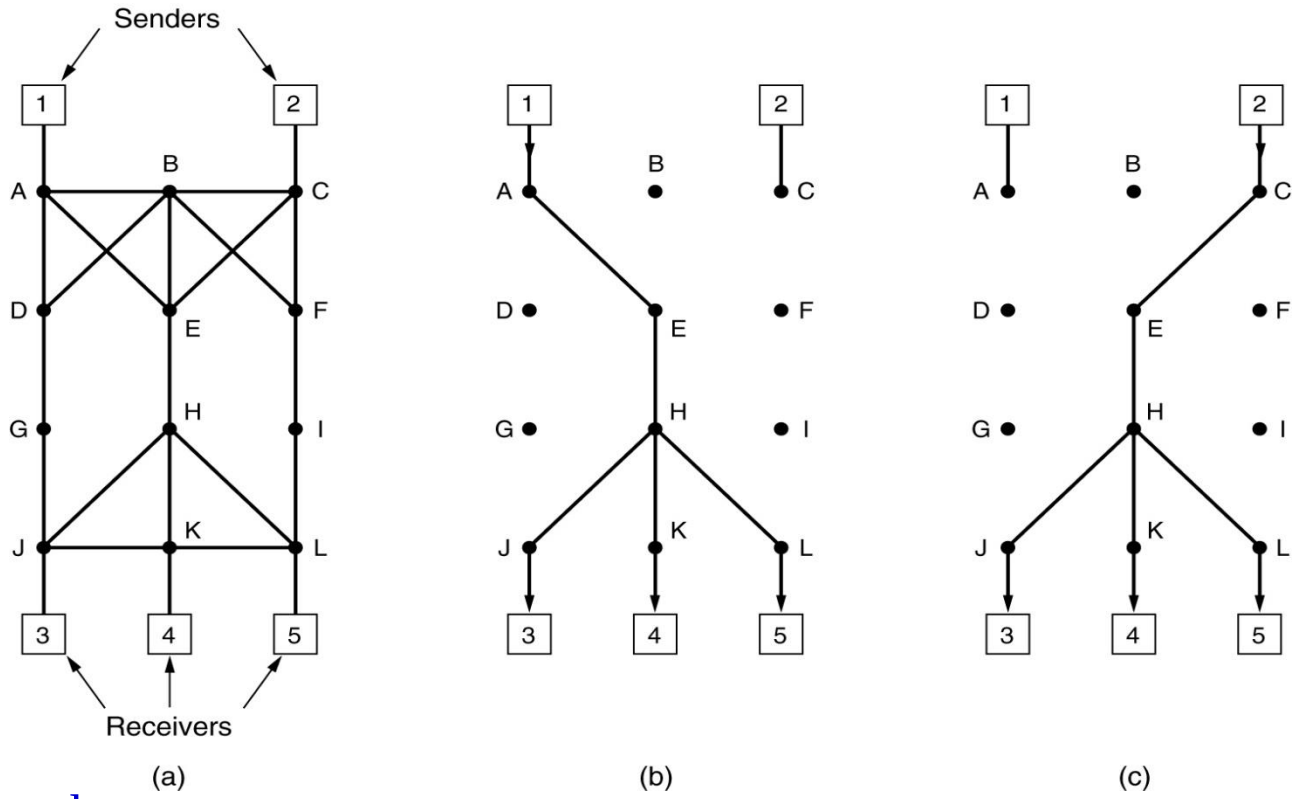
- **Integrated services** architecture was aimed for *unicast* and *multicast* applications. An example of the unicast is a single user streaming a video clip from a news site. An example of the multicast is a collection of digital television stations broadcasting their programs as stream of IP packets to many receivers at various locations.

- In many multicast applications, groups can change membership dynamically. The approach of having the senders reserve bandwidth in advance does not work.

- The main protocol for the Integrated services architecture is the **Resource reSerVation Protocol (RSVP)**.

- RSVP allows multiple senders to transmit to multiple groups of receivers, permits individual receivers to switch channels freely, and optimizes the bandwidth use while at the same time eliminating congestion.

RSVP-The ReSerVation Protocol

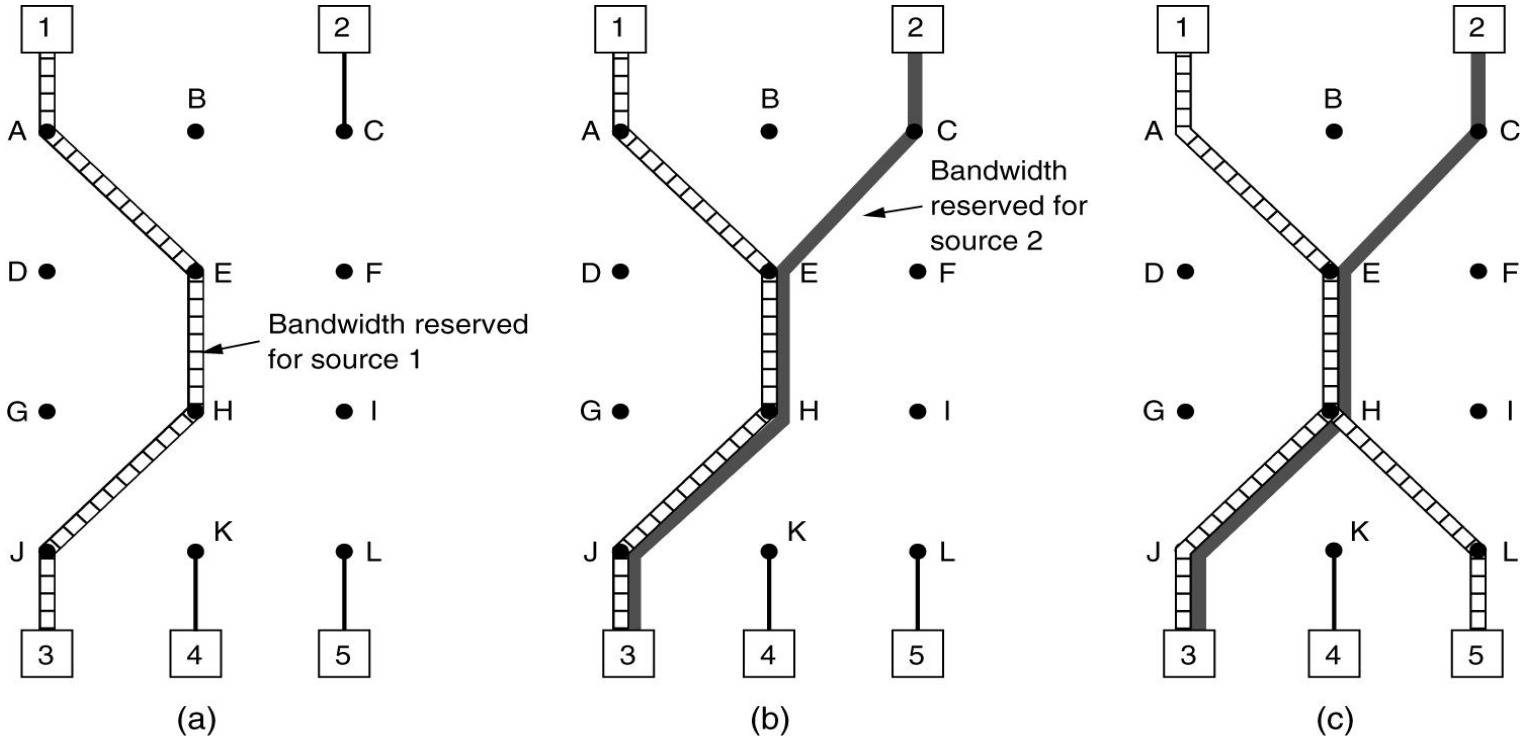


(a) A network

(b) The multicast spanning tree for host 1

(c) The multicast spanning tree for host 2

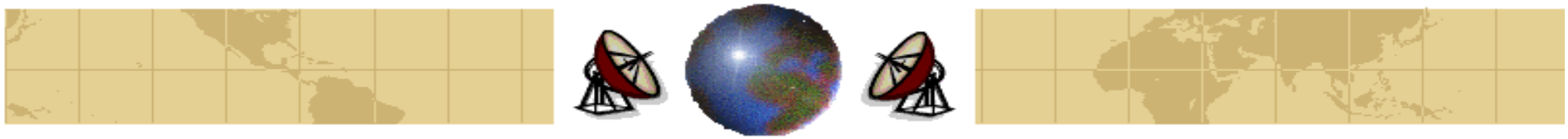
RSVP-The ReSerVation Protocol



(a) Host 3 requests a channel to host 1.

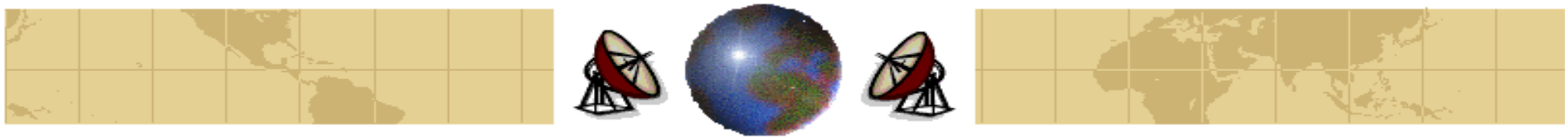
(b) Host 3 then requests a second channel, to host 2.

(c) Host 5 requests a channel to host 1.



Differentiated Service

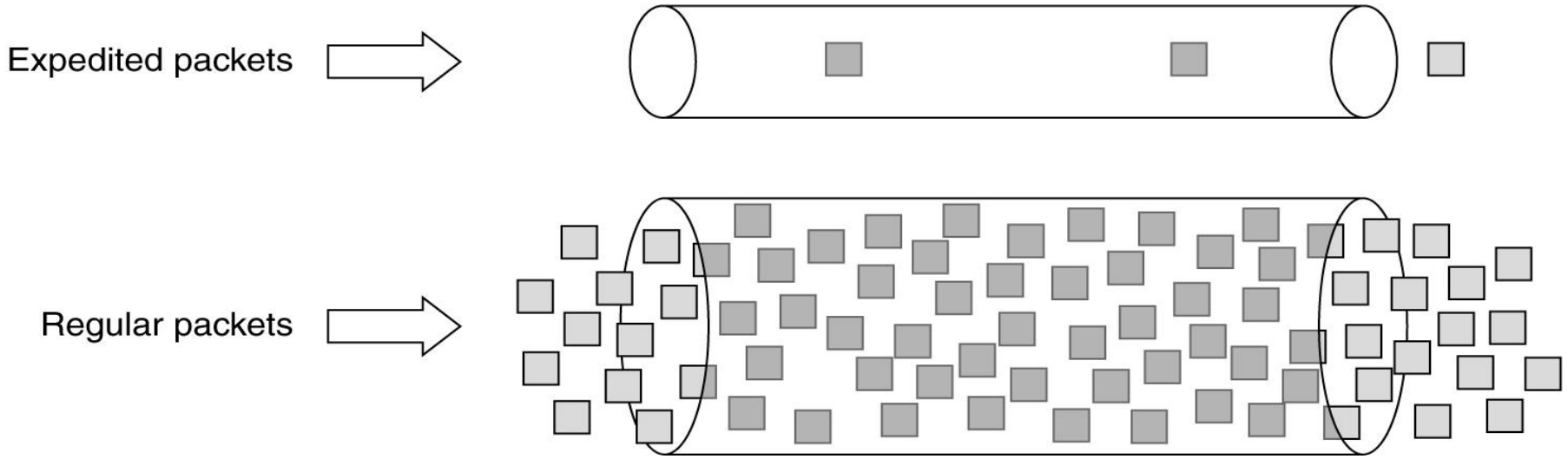
- **Differentiated service (DS)** architecture can be implemented locally in each router without advance setup and without having the whole path involved.
- DS can be offered by a set of routers forming an *administrative domain* (e.g., STC). The *administration* defines a set of service classes with corresponding forwarding rules.
- If a customer signs up for DS, customer packets entering the domain may carry a *Type of Service* (ToS) Field.



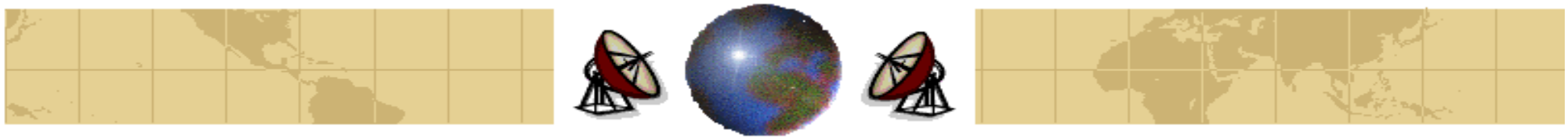
Expedited Forwarding

- The choice of service classes is up to each operator, but packets are often forwarded among subnets run by different operators.
- **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 a s though no other packets were present*.
- The *implementation* of this strategy is to program the routers to have two output queues for each outgoing line, one for expedited packets and one for the regular packets. When a packet arrives, it is queued accordingly. If 10% of the traffic is expedited and 90% is regular, 20% of the bandwidth is could be dedicated to expedited traffic and the rest for the regular traffic.

Expedited Forwarding



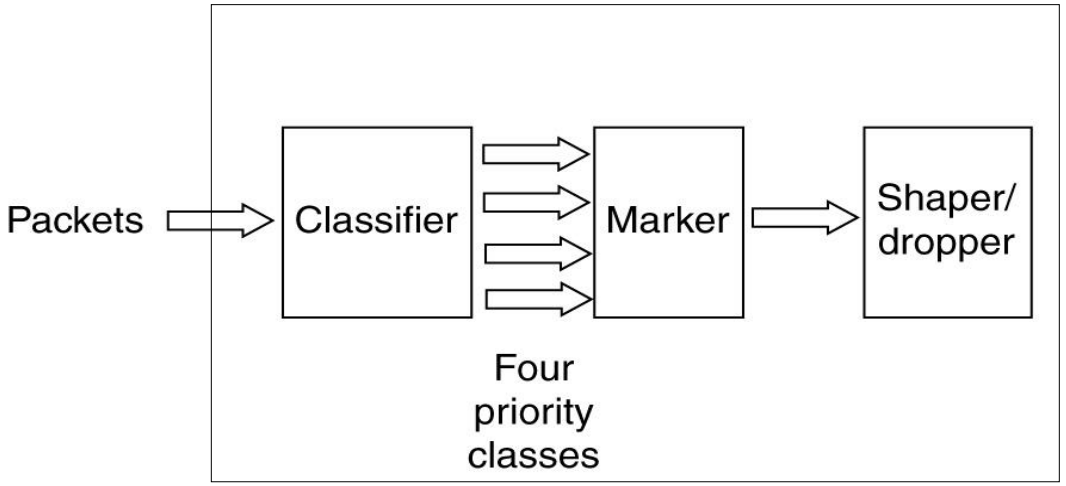
Expedited packets experience a traffic-free network



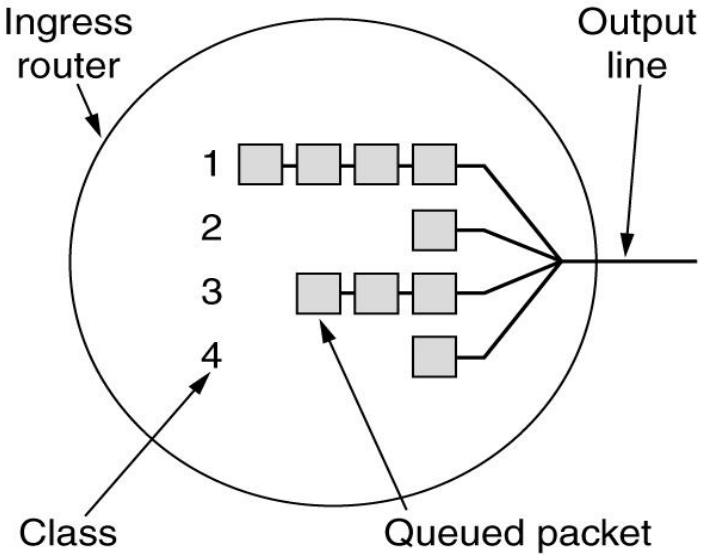
Assured Forwarding

- **Assured forwarding** has *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*.
- A possible *implementation* of data flow for assured forwarding:
  - **Step 1** is to classify the packets into one of four priority lasses. This can be done on the sending host or in the ingress router.
  - **Step 2** is to mark the packets according their class. A *header field* is needed for this purpose (e.g., 8-bit *Type of Service* is available in the IP header).
  - **Step 3** is to pass the packets through a shaper/dropper filter that may delay or drop some of them to shape the four streams into acceptable forms (e.g., Leaky or token buckets).

Assured Forwarding

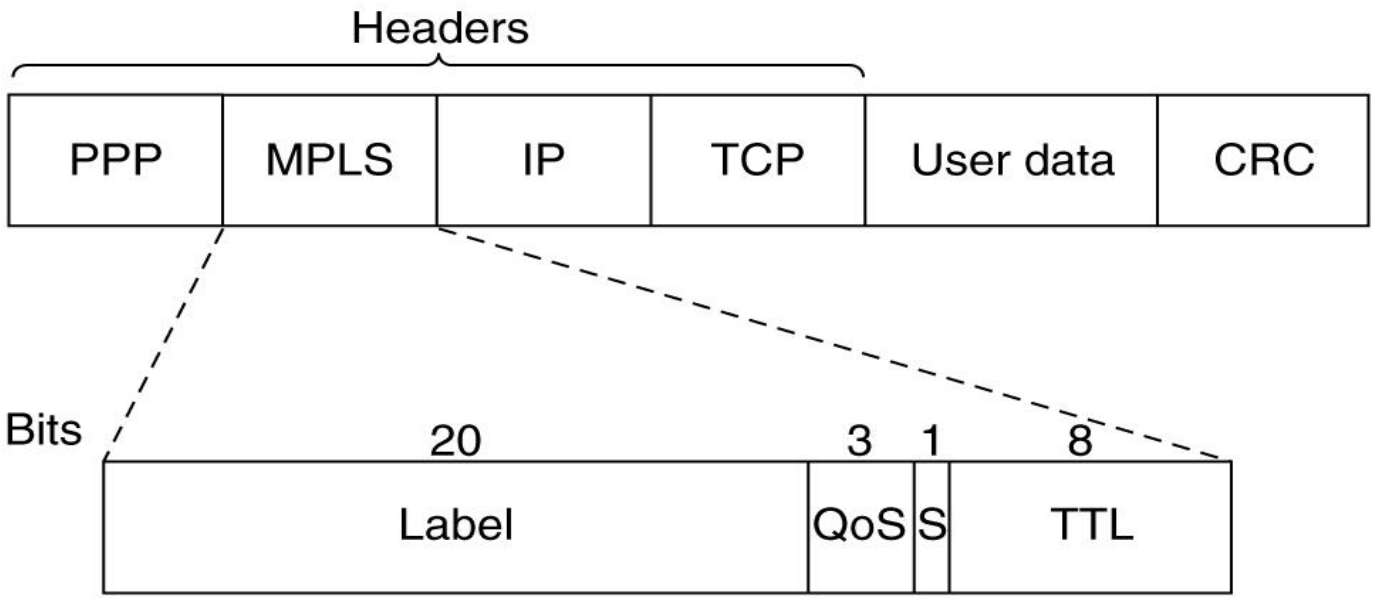


Sending Host



A possible implementation of the data flow for assured forwarding

Label Switching and MPLS



Transmitting a TCP segment using IP, MPLS, and PPP