**Congestion**

Congestion in a network may occur if the load on the network(the number of packets sent to the network) is greater than the capacity of the network(the number of packets a network can handle). Congestion control refers to the mechanisms and techniques to control the congestion and keep the load below the capacity .

• When too many packets are pumped into the system, congestion occur leading into degradation of performance.

• Congestion tends to feed upon itself and back ups.

• Congestion shows lack of balance between various networking equipments.

• It is a global issue.

When too many packets are present in a subnet or a part of subnet, performance degrades. This situation is called congestion. When number of packets dumped into the subnet by the hosts is within its carrying capacity, they are all delivered (except for a few that contain transmission errors), and 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. At very high traffics, performance collapses completely and almost no packets are delivered.

**Causes of Congestion**

 When there are more input lines and less or single output lines.

 When there is slow router i.e., if routers CPU‘s, are slow

 If the router has no free buffers i.e., insufficient memory to hold queue of packets.

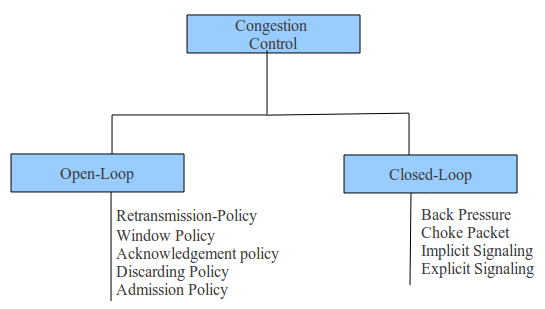
 If the components used in subnet (link, router, switches, etc) have different traffics carrying and switching capacities, then congestion occurs.

 If the bandwidths of the lines are low, it can‘t carry large volume of packets and caused

congestion. Hence, congestion cannot be eradicated but can be controlled.

**Congestion Control Algorithms**

In general, we can divide congestion control mechanisms into two broad categories: open-loop congestion control (prevention) and closed-loop congestion control (removal) as shown in Figure



**Open Loop Congestion Control:**

In open-loop congestion control, policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination

**Retransmission Policy**

Retransmission is sometimes unavoidable. If the sender feels that a sent packet is lost or corrupted, the packet needs to be retransmitted. Retransmission in general may increase congestion in the network. However, a good retransmission policy can prevent congestion. The retransmission policy and the retransmission timers must be designed to optimize efficiency and at the same time prevent congestion. For example, the retransmission policy used by TCP is designed to prevent or alleviate congestion.

**Window Policy**

The type of window at the sender may also affect congestion. The Selective Repeat window is better than the Go-Back-N window for congestion control. In the Go-Back-N window, when the timer for a packet times out, several packets may be resent, although some may have arrived safe and sound at the receiver. This duplication may make the congestion worse. The Selective Repeat window, on the other hand, tries to send the specific packets that have been lost or corrupted.

**Acknowledgment Policy :**

The acknowledgment policy imposed by the receiver may also affect congestion. If the receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion. Several approaches are used in this case. A receiver may send an acknowledgment only if it has a packet to be sent or a special timer expires. A receiver may decide to acknowledge only N packets at a time. We need to know that the acknowledgments are also part of the load in a network. Sending fewer acknowledgments means imposing less load on the network.

**Discarding Policy :**

A good discarding policy by the routers may prevent congestion and at the same time may not harm the integrity of the transmission. For example, in audio transmission, if the policy is to discard less sensitive packets when congestion is likely to happen, the quality of sound is still preserved and congestion is prevented or alleviated.

**Admission Policy :**

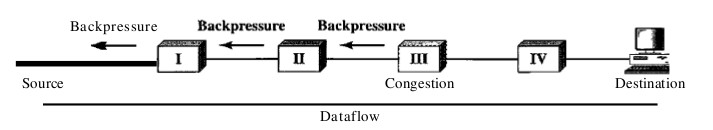
An admission policy, which is a quality-of-service mechanism, can also prevent congestion in virtual-circuit networks. Switches in a flow first check the resource requirement of a flow before admitting it to the network. A router can deny establishing a virtual- circuit connection if there is congestion in the network or if there is a possibility of future congestion.

**Closed-Loop Congestion Control**

Closed-loop congestion control mechanisms try to alleviate congestion after it happens. Several mechanisms have been used by different protocols.

**Back-pressure:**

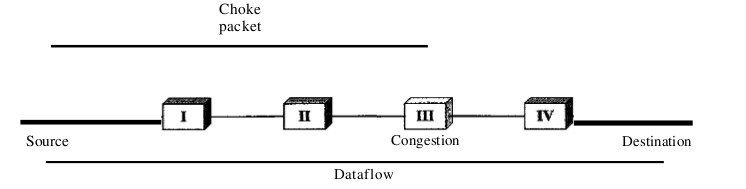
The technique of backpressure refers to a congestion control mechanism in which a congested node stops receiving data from the immediate upstream node or nodes. This may cause the upstream node or nodes to become congested, and they, in turn, reject data from their upstream nodes or nodes. And so on. Backpressure is a node-to-node congestion control that starts with a node and propagates, in the opposite direction of data flow, to the source. The backpressure technique can be applied only to virtual circuit networks, in which each node knows the upstream node from which a flow of data is corning.



Node III in the figure has more input data than it can handle. It drops some packets in its input buffer and informs node II to slow down. Node II, in turn, may be congested because it is slowing down the output flow of data. If node II is congested, it informs node I to slow down, which in turn may create congestion. If so, node I informs the source of data to slow down. This, in time, alleviates the congestion. Note that the pressure on node III is moved backward to the source to remove the congestion. None of the virtual-circuit networks we studied in this book use backpressure. It was, however, implemented in the first virtual-circuit network, X.25. The technique cannot be implemented in a datagram network because in this type of network, a node (router) does not have the slightest knowledge of the upstream router.

**Choke Packet**

A choke packet is a packet sent by a node to the source to inform it of congestion. Note the difference between the backpressure and choke packet methods. In backpresure, the warning is from one node to its upstream node, although the warning may eventually reach the source station. In the choke packet method, the warning is from the router, which has encountered congestion, to the source station directly. The intermediate nodes through which the packet has traveled are not warned. We have seen an example of this type of control in ICMP. When a router in the Internet is overwhelmed datagrams, it may discard some of them; but it informs the source . host, using a source quench ICMP message. The warning message goes directly to the source station; the intermediate routers, and does not take any action. Figure shows the idea of a choke packet.



**Implicit Signaling**

In implicit signaling, there is no communication between the congested node or nodes and the source. The source guesses that there is a congestion somewhere in the network from other symptoms. For example, when a source sends several packets and there is no acknowledgment for a while, one assumption is that the network is congested. The delay in receiving an acknowledgment is interpreted as congestion in the network; the source should slow down. We will see this type of signaling when we discuss TCP congestion control later in the chapter.

**Explicit Signaling**

The node that experiences congestion can explicitly send a signal to the source or destination. The explicit

signaling method, however, is different from the choke packet method. In the choke packet method, a separate packet is used for this purpose; in the explicit signaling method, the signal is included in the packets that carry data. Explicit signaling, as we will see in Frame Relay congestion control, can occur in either the forward or the backward direction.

**Backward Signaling** A bit can be set in a packet moving in the direction opposite to the congestion. This bit can warn the source that there is congestion and that it needs to slow down to avoid the discarding of packets. **Forward Signaling** A bit can be set in a packet moving in the direction of the congestion. This bit can warn the destination that there is congestion. The receiver in this case can use policies, such as slowing down the acknowledgments, to alleviate the congestion.

**Traffic Shaping**

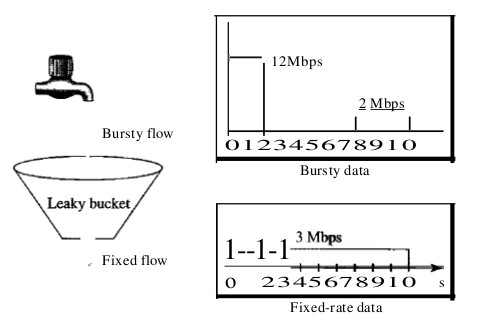
Traffic shaping is a mechanism to control the amount and the rate of the traffic sent to the network. Two techniques can shape traffic:

1.Leaky bucket and

2. Token bucket.

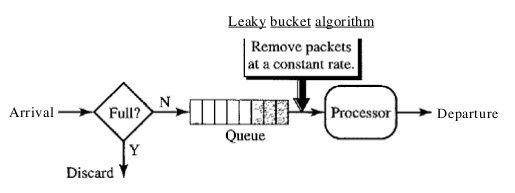
1. **Leaky Bucket**

If a bucket has a small hole at the bottom, the water leaks from the bucket at a constant rate as long as there is water in the bucket. The rate at which the water leaks does not depend on the rate at which the water is input to the bucket unless the bucket is empty. The input rate can vary, but the output rate remains constant. Similarly, 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. Figure shows a leaky bucket and its effects.



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. In Figure 24.19 the host sends a burst of data at a rate of 12 Mbps for 2 s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in lOs. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10 s. 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.



*Leaky Bucket Implementation*

A simple leaky bucket implementation is shown in Figure 24.20. A FIFO queue holds the packets. If the traffic consists of fixed-size packets (e.g., cells in ATM networks), 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.

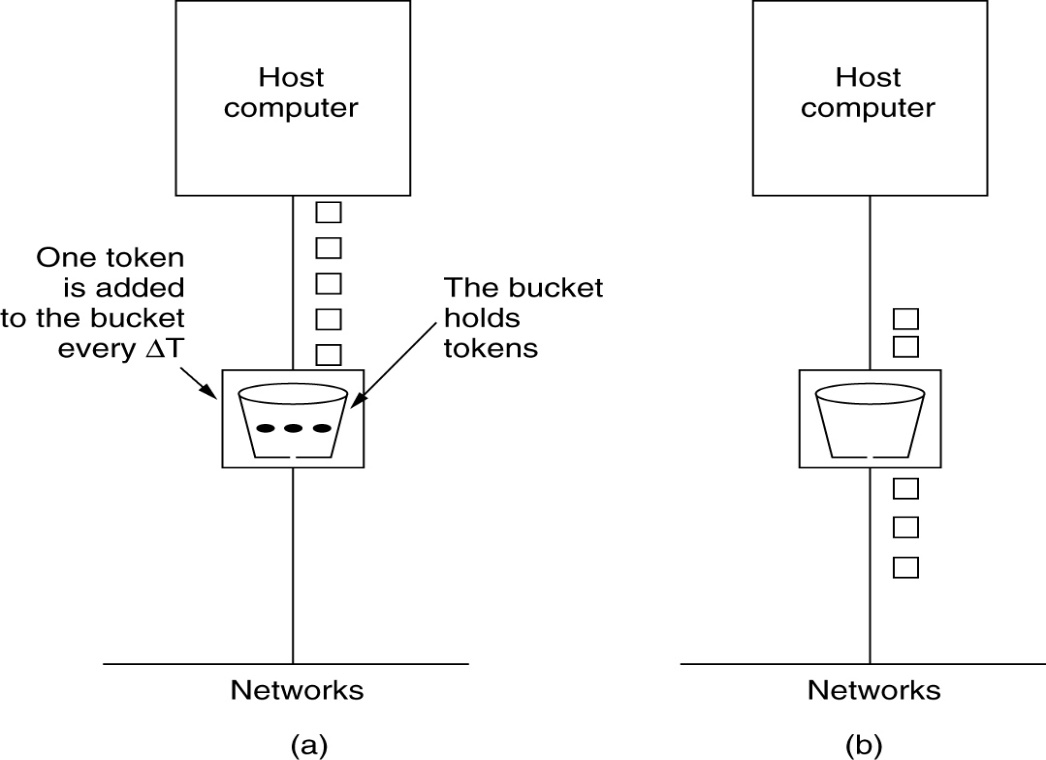
Repeat this step until n is smaller than the packet size.

3. Reset the counter 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.*

**2. Token Bucket**

* In contrast to the LB, the Token Bucket (TB) algorithm, allows the output rate to vary, depending on the size of the burst.
* In the TB algorithm, the bucket holds tokens. To transmit a packet, the host must capture and destroy one token.
* Tokens are generated by a clock at the rate of one token every Dt sec.
* Idle hosts can capture and save up tokens (up to the max. size of the bucket) in order to send larger bursts later.



(a) Before (b) After

**Token bucket operation**

* TB accumulates fixed size tokens in a token bucket
* Transmits a packet (from data buffer, if any are there) or arriving packet if the sum of the token sizes in the bucket add up to packet size
* More tokens are periodically added to the bucket (at rate Dt). If tokens are to be added when the bucket is full, they are discarded

**Token bucket properties**

* Does not bound the peak rate of small bursts, because bucket may contain enough token to cover a complete burst size
* Performance depends only on the sum of the data buffer size and the token bucket size

**Token bucket – example**

* 2 tokens of size 100 bytes added each second to the token bucket of capacity 500 bytes
  + Avg. rate = 200 bytes/sec, burst size = 500 bytes
  + Packets bigger than 500 bytes will never be sent
  + Peak rate is unbounded – i.e., 500 bytes of burst can be transmitted arbitrarily fast

**Leaky Bucket vs Token Bucket**

* LB discards packets; TB does not. TB discards tokens.
* With TB, a packet can only be transmitted if there are enough tokens to cover its length in bytes.
* LB sends packets at an average rate. TB allows for large bursts to be sent faster by speeding up the output.
* TB allows saving up tokens (permissions) to send large bursts. LB does not allow saving.

