UNIT – III

ROUTING ALGORITHMS

2. Discuss about different routing algorithms in detail. (or)

Discuss shortest path routing. (Or)

What is flooding? Discuss. (Or)

Differentiate and explain adaptive and nonadaptive routing algorithms. (Or)

Describe hierarchical Broadcast and Multicasting routing.

(Nov'11, May'10, Dec'08, Nov'07, Dec'05, Dec'04)

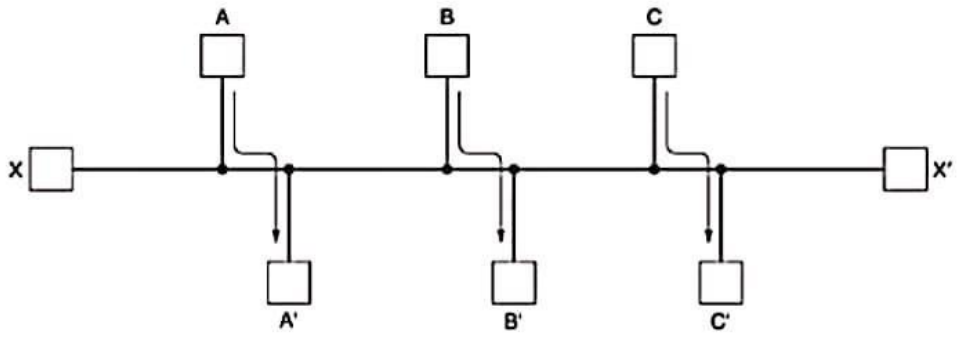
ROUTING ALGORITHMS

The **routing algorithm** is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on.

PROPERTIES OF ROUTING ALGORITHM:

Correctness, simplicity, robustness, stability, fairness, and optimality

FAIRNESS AND OPTIMALITY.



Fairness and optimality may sound obvious, but as it turns out, they are often contradictory goals. There is enough traffic between A and A', between B and B', and between C and C' to saturate the horizontal links. To maximize the total flow, the X to X' traffic should be shut off altogether. Unfortunately, X and X' may not see it that way. Evidently, some compromise between global efficiency and fairness to individual connections is needed.

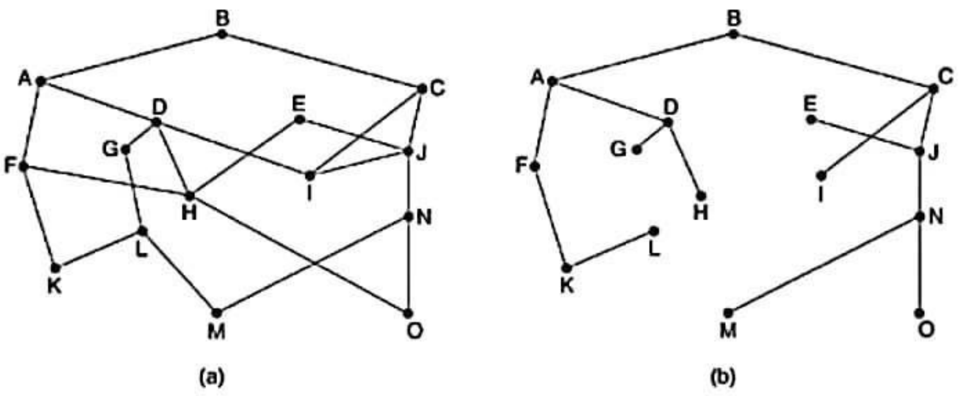
CATEGORY OF ALGORITHM

- Routing algorithms can be grouped into two major classes: **nonadaptive and adaptive.**
- **Nonadaptive algorithms** do not base their routing decisions on measurements or estimates of the current traffic and topology. Instead, the choice of the route to use to get from I to J is computed in advance, off-line, and downloaded to the routers when the network is booted.
- This procedure is sometimes called **Static routing**.

- **Adaptive algorithms**, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well
- This procedure is sometimes called **dynamic routing**

THE OPTIMALITY PRINCIPLE

- If router J is on the optimal path from router I to router K, then the optimal path from J to K also falls along the same route.
- The set of optimal routes from all sources to a given destination form a tree rooted at the destination. Such a tree is called a sink tree.



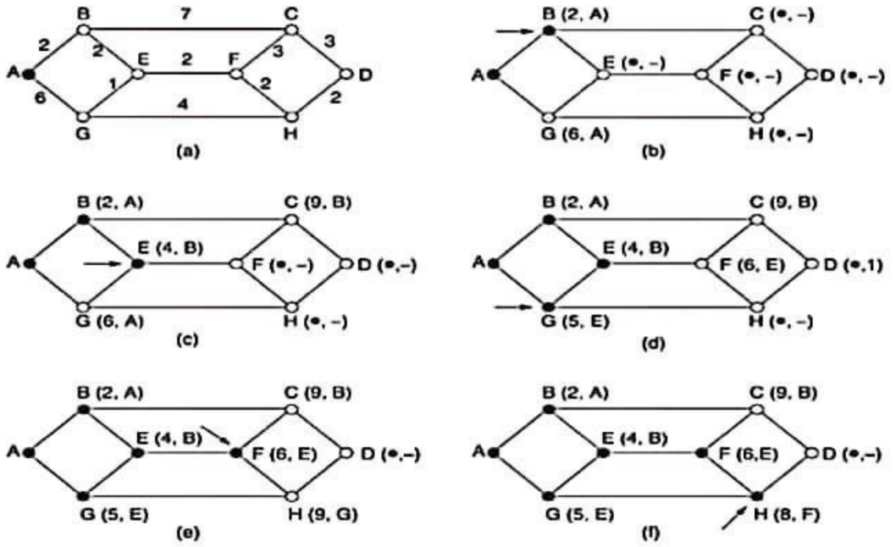
(a) A subnet. (b) A sink tree for router **B.**

- As a direct consequence of the optimality principle, we can see that the set of optimal routes from all sources to a given destination form a tree rooted at the destination.
- Such a tree is called a **sink tree** where the distance metric is the number of hops. Note that a sink tree is not necessarily unique; other trees with the same path lengths may exist.
- The goal of all routing algorithms is to discover and use the sink trees for all routers.

SHORTEST PATH ROUTING

- A technique to study routing algorithms: The idea is to build a graph of the subnet, with each node of the graph representing a router and each arc of the graph representing a communication line (often called a link).
- To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.
- One way of measuring path length is the number of hops. Another metric is the geographic distance in kilometers. Many other metrics are also possible. For example, each arc could be labeled with the mean queuing and transmission delay for some standard test packet as determined by hourly test runs.
- In the general case, the labels on the arcs could be computed as a function of the distance, bandwidth, average traffic, communication cost, mean queue length, measured

delay, and other factors. By changing the weighting function, the algorithm would then compute the "shortest" path measured according to any one of a number of criteria or to a combination of criteria.



The first five steps used in computing the shortest path from* A *to* D. *The arrows indicate the working node.

- To illustrate how the labelling algorithm works, look at the weighted, undirected graph of Fig. 5-7(a), where the weights represent, for example, distance.
- We want to find the shortest path from *A* to *D*. We start out by marking node *A* as permanent, indicated by a filled-in circle.
- Then we examine, in turn, each of the nodes adjacent to *A* (the working node), relabeling each one with the distance to *A*.
- Whenever a node is relabelled, we also label it with the node from which the probe was made so that we can reconstruct the final path later.
- Having examined each of the nodes adjacent to *A*, we examine all the tentatively labelled nodes in the whole graph and make the one with the smallest label permanent, as shown in Fig. 5-7(b).
- This one becomes the new working node.

We now start at *B* and examine all nodes adjacent to it. If the sum of the label on *B* and the distance from *B* to the node being considered is less than the label on that node, we have a shorter path, so the node is relabelled.

After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively-labelled node with the smallest value. This node is made permanent and becomes the working node for the next round. Figure 5-7 shows the first five steps of the algorithm.

- To see why the algorithm works, look at Fig. 5-7(c). At that point we have just made *E* permanent. Suppose that there were a shorter path than *ABE*, say *AXYZE*. There are two possibilities: either node *Z* has already been made permanent, or it has not been. If it has, then *E* has already been probed (on the round following the one when *Z* was made permanent), so the *AXYZE* path has not escaped our attention and thus cannot be a shorter path.

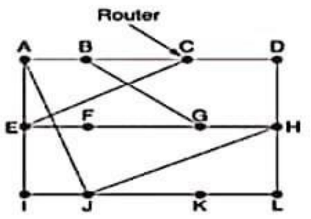
- Now consider the case where Z is still tentatively labelled. Either the label at *Z* is greater than or equal to that at *E*, in which case *AXYZE* cannot be a shorter path than *ABE*, or it is less than that of *E*, in which case *Z* and not *E* will become permanent first, allowing *E* to be probed from Z.
- This algorithm is given in Fig. 5-8. The global variables *n* and *dist* describe the graph and are initialized before *shortest path* is called. The only difference between the program and the algorithm described above is that in Fig. 5-8, we compute the shortest path starting at the terminal node, *t*, rather than at the source node, *s*. Since the shortest path from *t* to *s* in an undirected graph is the same as the shortest path from *s* to *t*, it does not matter at which end we begin (unless there are several shortest paths, in which case reversing the search might discover a different one). The reason for searching backward is that each node is labelled with its predecessor rather than its successor. When the final path is copied into the output variable, *path*, the path is thus reversed. By reversing the search, the two effects cancel, and the answer is produced in the correct order

FLOODING

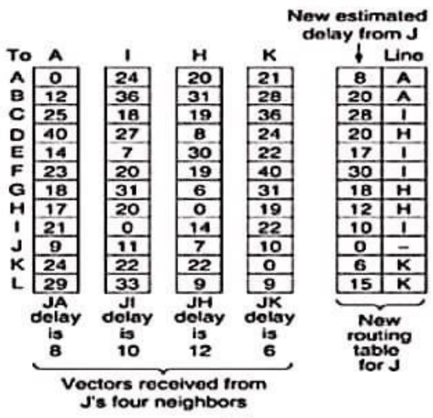
- Another static algorithm is **flooding**, in which every incoming packet is sent out on every outgoing line except the one it arrived on.
- Flooding obviously generates vast numbers of duplicate packets, in fact, an infinite number unless some measures are taken to damp the process.
- One such measure is to have a hop counter contained in the header of each packet, which is decremented at each hop, with the packet being discarded when the counter reaches zero.
- Ideally, the hop counter should be initialized to the length of the path from source to destination. If the sender does not know how long the path is, it can initialize the counter to the worst case, namely, the full diameter of the subnet.

DISTANCE VECTOR ROUTING

- **Distance vector routing** algorithms operate by having each router maintain a table (i.e, a vector) giving the best known distance to each destination and which line to use to get there.
- These tables are updated by exchanging information with the neighbors.
- The distance vector routing algorithm is sometimes called by other names, most commonly the distributed **Bellman-Ford** routing algorithm and the **Ford-Fulkerson** algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962).
- It was the original ARPANET routing algorithm and was also used in the Internet under the name RIP.



(a)



To	A	I	H	K	New estimated delay from J	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	–
K	24	22	22	0	6	K
L	29	33	9	9	15	K
	JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6		

Vectors received from J's four neighbors

New routing table for J

(b)

(a) A subnet. (b) Input from A, I, H, K, *and the new routing table for* J.

- Part (a) shows a subnet. The first four columns of part (b) show the delay vectors received from the neighbours of router *J*.
- *A* claims to have a 12-msec delay to *B*, a 25-msec delay to *C*, a 40-msec delay to *D*, etc. Suppose that *J* has measured or estimated its delay to its neighbours, *A*, *I*, *H*, and *K* as 8, 10, 12, and 6 msec, respectively.

Each node constructs a one-dimensional array containing the "distances"(costs) to all other nodes and distributes that vector to its immediate neighbors.

1. The starting assumption for distance-vector routing is that each node knows the cost of the link to each of its directly connected neighbors.
2. A link that is down is assigned an infinite cost.

Example.

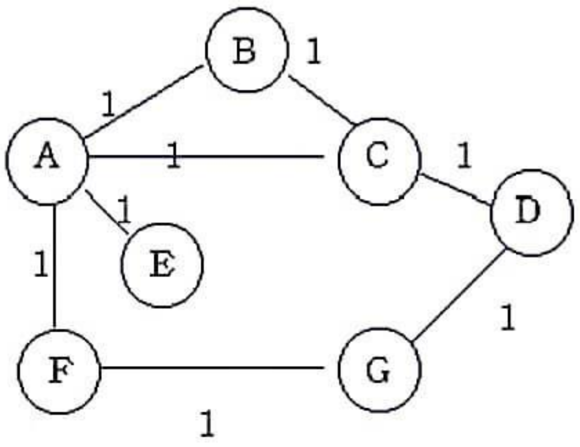


Table 1. Initial distances stored at each node(global view).

Information Stored at Node	**Distance to Reach Node** A	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

We can represent each node's knowledge about the distances to all other nodes as a table like the one given in Table 1.

Note that each node only knows the information in one row of the table.

1. Every node sends a message to its directly connected neighbors containing its personal list of distance. (for example, **A** sends its information to its neighbors **B,C,E**, and **F**.)
2. If any of the recipients of the information from **A** find that **A** is advertising a path shorter than the one they currently know about, they update their list to give the new path length and note that they should send packets for that destination through **A**. (node **B** learns from **A** that node **E** can be reached at a cost of 1; **B** also knows it can reach **A** at a cost of 1, so it adds these to get the cost of reaching **E** by means of **A**. **B** records that it can reach **E** at a cost of 2 by going through **A**.)
3. After every node has exchanged a few updates with its directly connected neighbors, all nodes will know the least-cost path to all the other nodes.
4. In addition to updating their list of distances when they receive updates, the nodes need to keep track of which node told them about the path that they used to calculate the cost, so that they can create their forwarding table. (for example, **B** knows that it was **A** who said " I can reach **E** in one hop" and so **B** puts an entry in its table that says " To reach **E**, use the link to **A**.)

Table 2. final distances stored at each node (global view).

Information	**Distance to Reach Node**						
Stored at Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

In practice, each node's forwarding table consists of a set of triples of the form:

(Destination, Cost, NextHop).

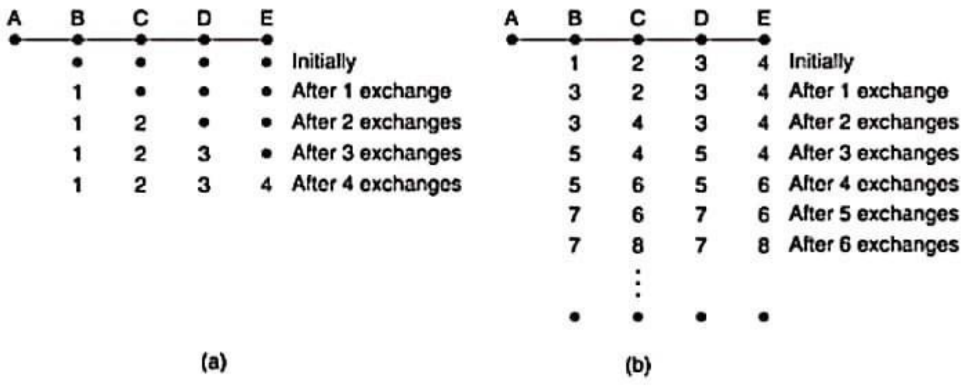
For example, Table 3 shows the complete routing table maintained at node B for the network in figure1.

Table 3. Routing table maintained at node B.

Destination	Cost	NextHop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	3	A

THE COUNT-TO-INFINITY PROBLEM

The count-to-infinity problem.



A	B	C	D	E	
	•	•	•	•	Initially
	1	•	•	•	After 1 exchange
	1	2	•	•	After 2 exchanges
	1	2	3	•	After 3 exchanges
	1	2	3	4	After 4 exchanges

(a)

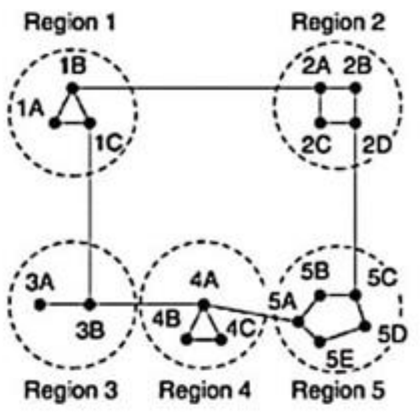
A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	•	•	•	•	

(b)

- Consider the five-node (linear) subnet of Fig. 5-10, where the delay metric is the number of hops. Suppose *A* is down initially and all the other routers know this. In other words, they have all recorded the delay to *A* as infinity.
- Now let us consider the situation of Fig. 5-10(b), in which all the lines and routers are initially up. Routers *B*, *C*, *D*, and *E* have distances to *A* of 1, 2, 3, and 4, respectively. Suddenly *A* goes down, or alternatively, the line between *A* and *B* is cut, which is effectively the same thing from *B*'s point of view.

HIERARCHICAL ROUTING

- The routers are divided into what we will call regions, with each router knowing all the details about how to route packets to destinations within its own region, but knowing nothing about the internal structure of other regions.
- For huge networks, a two-level hierarchy may be insufficient; it may be necessary to group the regions into clusters, the clusters into zones, the zones into groups, and so on, until we run out of names for aggregations.



(a)

Full table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

Hierarchical table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

- Figure 5-15 gives a quantitative example of routing in a two-level hierarchy with five regions.
- The full routing table for router 1*A* has 17 entries, as shown in Fig. 5-15(b).
- When routing is done hierarchically, as in Fig. 5-15(c), there are entries for all the local routers as before, but all other regions have been condensed into a single router, so all traffic for region 2 goes via the 1*B* -2*A* line, but the rest of the remote traffic goes via the 1*C* -3*B* line.
- Hierarchical routing has reduced the table from 17 to 7 entries. As the ratio of the number of regions to the number of routers per region grows, the savings in table space increase.

CONGESTION CONTROL

Congestion: Too many packets present in the subnet causes congestion. It will result in the degradation of performance.

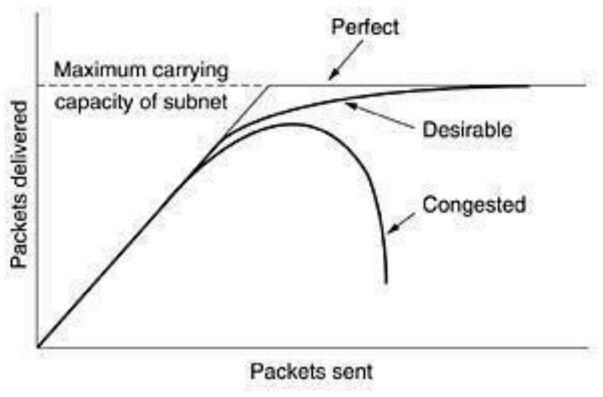


Figure: When too much traffic is offered, congestion sets in and performance degrades sharply

- ✓ When the number of packets dumped into the subnet by the hosts is within its carrying capacity, they areall delivered and the number is proportional to the number sent.
- ✓ However as traffic increases too far, the routers are no longer able to cope and they begin to lose packets.This tends to make matters worse. At very high traffic, performance collapses completely and almost no packets are delivered.

CAUSES OF THE CONGESTION

1. Mismatch in the speed at which the packets are dumped into the router and the speed atwhich they are delivered.
2. Packets may arrive at four or more different lines are send out in single line which may build up congestion. Here the solution is resources may be increased which can handle the huge amount of data or the load can be deceased to avoid congestion. It can also be handledby splitting the output into many lines.
3. Insufficient memory
4. Slow processor is another reason for congestion build up.
5. Low bandwidth can also cause congestion.

<u>GENERAL PRINCIPLES OF CONGESTION CONTROL</u>

Congestion occurs as there are two approaches for congestion control

1. **OPEN LOOP:** Control before the congestion occurs. Different methods are
    a. <u>Retransmission policy</u>: Retransmits the data and timers kept to optimize efficiency.
    b. <u>Window policy</u>: Selective repeat is used here than Go-Back-N
    c. <u>Acknowledgement policy</u>: Does not acknowledge every packet
    d. <u>Discarding policy</u>: The packets are discarded and they are not informed back to the sender to avoid congestion. When the sender does not receive any acknowledgementthe same packets are sending again.
    e. <u>Admission policy</u>: Check the packets for the resource requirement of a flow before admitting tothe network.
2. **CLOSED LOOP:** it detects the congestion, pass the information to where the action can be taken and adjust the system to correct the problem (detect ,feedback, correct). Different methods are
    a. <u>Back pressure</u>: Inform the previous router to reduce the rate of outgoing packets
    b. <u>Choke packet</u>: Packet send back to the sender to inform about the congestion.
    c. <u>Implicit signaling</u>: Sender slows down the sending rate by detecting implicit signal concerningcongestion.
    d. <u>Explicit signaling</u>: Backward or forward signaling done by packets send from congestion towarn the source.

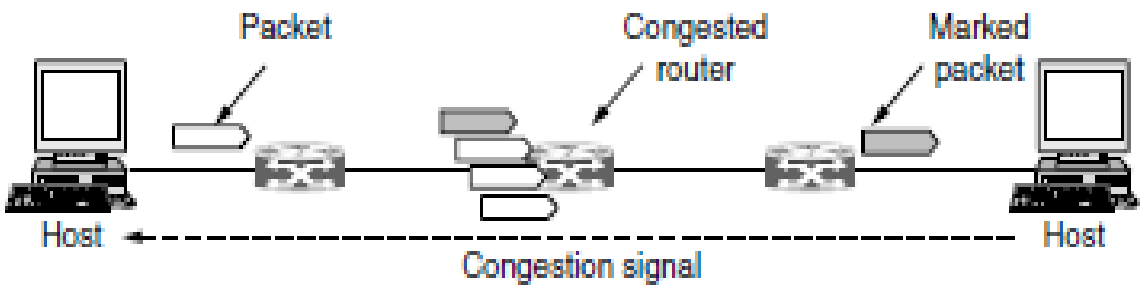


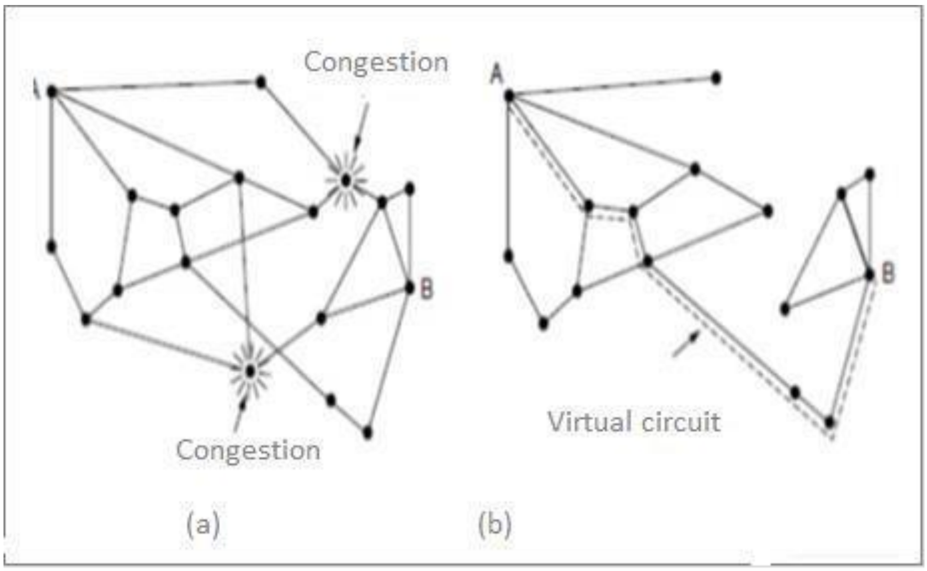
Figure: Explicit Congestion signaling

ADMISSION CONTROL

1. It is one of techniques that is widely used in virtual-circuit networks to keep congestion at bay. The idea is do not set up a new virtual circuit unless the network can carry the added traffic without becoming congested.
2. Admission control can also be combined with traffic aware routing by considering routes around traffic hotspots as part of the setup procedure.

Example

Take two networks (a) A congestion network and (b) The portion of the network that is not congested. A virtual circuit A to B is also shown below −



Explanation

Step 1 − Suppose a host attached to router A wants to set up a connection to a host attached to router B. Normally this connection passes through one of the congested routers.

Step 2 − To avoid this situation, we can redraw the network as shown in figure (b), removing the congested routers and all of their lines.

Step 3 − The dashed line indicates a possible route for the virtual circuit that avoids the congested routers.

CONGESTION CONTROL ALGORITHMS

What is **congestion?**

A state occurring in network layer when the message traffic is so heavy that it slows down network response time.

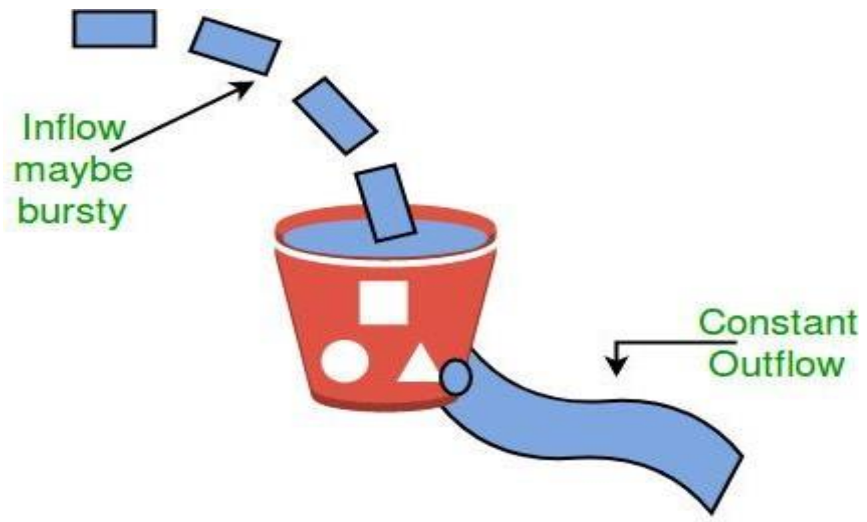
<u>**Effects of Congestion**</u>

a. As delay increases, performance decreases.
b. If delay increases, retransmission occurs, making situation worse.

<u>CONGESTION CONTROL ALGORITHMS</u>

- **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 constant rate. When the bucket is full with water additional water entering spills over the sides and is lost.



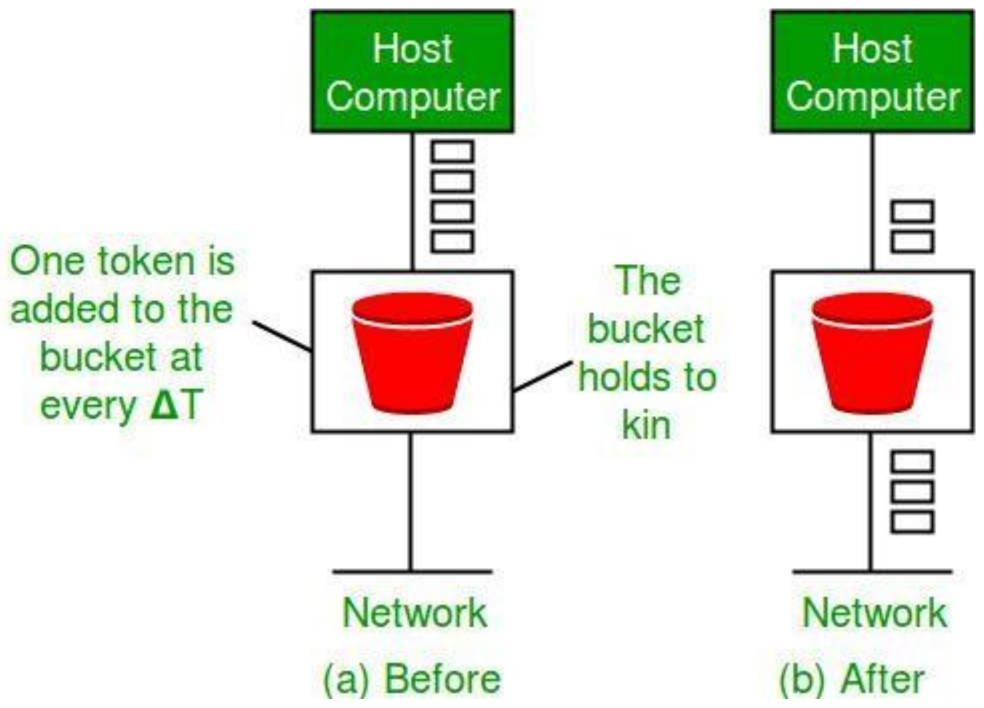
Similarly, each network interface contains a leaky bucket and the following steps are involved in leaky bucket algorithm:

1. When host wants to send packet, packet is thrown into the bucket.
2. The bucket leaks at a constant rate, meaning the network interface transmits packets at a constant rate.
3. Bursty traffic is converted to a uniform traffic by the leaky bucket.
4. In practice the bucket is a finite queue that outputs at a finite rate.

- **Token bucket Algorithm**

  Need of token bucket Algorithm:-

  1. The leaky bucket algorithm enforces output pattern at the average rate, no matter how bursty the traffic is.
  2. So in order to deal with the bursty traffic we need a flexible algorithm so that the data is not lost.
  3. One such algorithm is token bucket algorithm.



Steps of this algorithm can be described as follows:

1. In regular intervals tokens are thrown into the bucket.
2. The bucket has a maximum capacity.
3. If there is a ready packet, a token is removed from the bucket, and the packet is sent.
4. If there is no token in the bucket, the packet cannot be sent.