

UNIT III

Network Layer: Network Layer Design issues, store and forward packet switching connection less and connection oriented networks-routing algorithms-optimality principle, shortest path, flooding, Distance Vector Routing, Count to Infinity Problem, Hierarchical Routing, Congestion control algorithms, admission control.

1. Explain the design issues of the Network layer. /

Discuss the services provided to the transport layer by the Network layer. /

How Connection Oriented and Connection Less Services are implemented? Explain. /

Compare the virtual circuits and datagram within the subnet.

2. Discuss about different routing algorithms in detail. /

Discuss shortest path routing. /

What is flooding? Discuss. /

Differentiate and explain adaptive and nonadaptive routing algorithms. /

Describe hierarchical Broadcast and Multicasting routing.

3. Discuss about different congestion control algorithms in detail /

Discuss the reasons for occurrence of congestion. Suggest some algorithms to control congestion.

4. What is meant by fragmentation? Discuss & Write short notes on:

(a) Tunnelling

(b) Internetwork routing

(c) Fragmentation.

5. Explain about the network layer in the internet in detail /

What is IP? Explain IPV 4 header.

**1. Explain the design issues of the Network layer. /
Discuss the services provided to the transport layer by the Network layer. /
How Connection Oriented and Connection Less Services are implemented?
Explain. /**

**Compare the virtual circuits and datagram within the
subnet.**

(Nov'09, Nov'07, Dec,06)

NETWORK LAYER DESIGN ISSUES

In the following sections we will provide an introduction to some of the issues that the designers of the network layer must grapple with. These issues include the service provided to the transport layer and the internal design of the subnet.

STORE-AND-FORWARD PACKET SWITCHING

- The network layer protocols operation can be seen in [Fig. 5-1](#).
- The major components of the system are the carrier's equipment (routers connected by transmission lines), shown inside the shaded oval.
- The customers' equipment, shown outside the oval. Host *H1* is directly connected to one of the carrier's routers, *A*, by a leased line. In contrast, *H2* is on a LAN with a router, *F*, owned and operated by the customer. This router also has a leased line to the carrier's equipment.
- We have shown *F* as being outside the oval because it does not belong to the carrier, but in terms of construction, software, and protocols, it is probably no different from the carrier's routers. Whether it belongs to the subnet is arguable, but for the purposes of this chapter, routers on customer premises are considered part of the subnet.

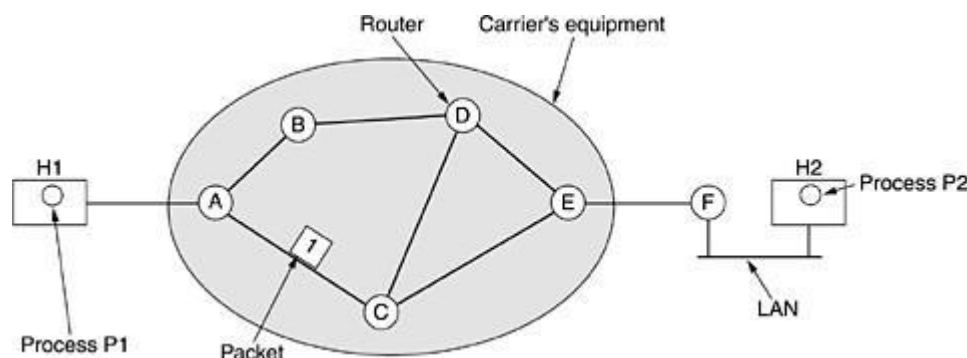


Figure 5-1. The environment of the network layer protocols.

- A host with a packet to send transmits it to the nearest router, either on its own LAN or over a point-to-point link to the carrier.

- The packet is stored there until it has fully arrived so the checksum can be verified. Then it is forwarded to the next router along the path until it reaches the destination host, where it is delivered.
- This mechanism is **store-and-forward packet switching**.

SERVICES PROVIDED TO THE TRANSPORT LAYER

The network layer provides services to the transport layer at the network layer/transport layer interface. The network layer services have been designed with the following goals in mind.

1. The services should be independent of the router technology.
 2. The transport layer should be shielded from the number, type, and topology of the routers present.
 3. The network addresses made available to the transport layer should use a uniform numbering plan, even across LANs and WANs.
- There is a discussion centers on whether the network layer should provide connection-oriented service or connectionless service.
 - In their view (based on 30 years of actual experience with a real, working computer network), the subnet is inherently unreliable, no matter how it is designed. Therefore, the hosts should accept the fact that the network is unreliable and do error control (i.e., error detection and correction) and flow control themselves.
 - This viewpoint leads quickly to the conclusion that the network service should be connectionless, with primitives SEND PACKET and RECEIVE PACKET and little else.
 - In particular, no packet ordering and flow control should be done, because the hosts are going to do that anyway, and there is usually little to be gained by doing it twice.
 - Furthermore, each packet must carry the full destination address, because each packet sent is carried independently of its predecessors, if any.
 - The other camp (represented by the telephone companies) argues that the subnet should provide a reliable, connection-oriented service.
 - These two camps are best exemplified by the Internet and ATM. The Internet offers connectionless network-layer service; ATM networks offer connection-oriented network-layer service. However, it is interesting to note that as quality-of-service guarantees are becoming more and more important, the Internet is evolving. In particular, it is starting to acquire properties normally associated with connection-oriented service, as we will see later. Actually, we got an inkling of this evolution during our study of VLANs in Chap. 4.

IMPLEMENTATION OF CONNECTIONLESS SERVICE

- Two different organizations are possible, depending on the type of service offered.
- If connectionless service is offered, packets are injected into the subnet individually and routed independently of each other. No advance setup is needed. In this context, the packets are frequently called **datagrams** (in analogy with telegrams) and the subnet is called a **datagram subnet**.
- If connection-oriented service is used, a path from the source router to the destination router must be established before any data packets can be sent. This connection is called a **VC (virtual circuit)**, in analogy with the physical circuits set up by the telephone system, and the subnet is called a **virtual-circuit subnet**.
- Let us now see how a datagram subnet works. Suppose that the process *P1* in **Fig. 5-2** has a long message for *P2*. It hands the message to the transport layer with instructions to deliver it to process *P2* on host *H2*. The transport layer code runs on *H1*, typically within the operating system. It prepends a transport header to the front of the message and hands the result to the network layer, probably just another procedure within the operating system.

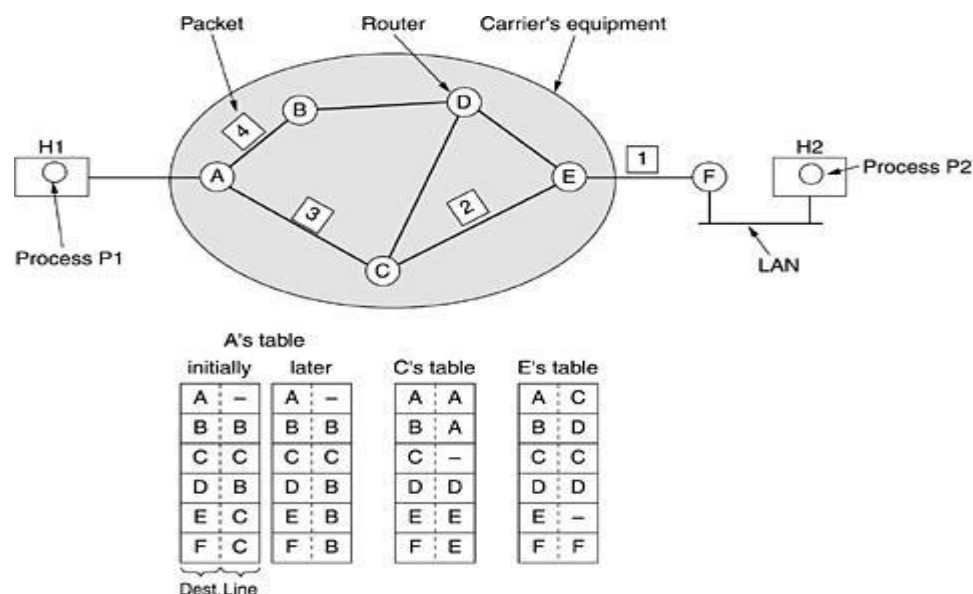


Figure 5-2. Routing within a datagram subnet

- Let us assume that the message is four times longer than the maximum packet size, so the network layer has to break it into four packets, 1, 2, 3, and 4 and sends each of them in turn to router *A* using some point-to-point protocol,
- For example, PPP. At this point the carrier takes over. Every router has an internal table telling it where to send packets for each possible destination. Each table entry is a pair consisting of a destination and the outgoing line to use for that destination. Only directly-connected lines can be used.

- For example, in [Fig. 5-2](#), *A* has only two outgoing lines—to *B* and *C*—so every incoming packet must be sent to one of these routers, even if the ultimate destination is some other router. *A*'s initial routing table is shown in the figure under the label "initially". As they arrived at *A*, packets 1, 2, and 3 were stored briefly (to verify their checksums). Then each was forwarded to *C* according to *A*'s table. Packet 1 was then forwarded to *E* and then to *F*. When it got to *F*, it was encapsulated in a data link layer frame and sent to *H2* over the LAN. Packets 2 and 3 follow the same route.
- However, something different happened to packet 4. When it got to *A* it was sent to router *B*, even though it is also destined for *F*. For some reason, *A* decided to send packet 4 via a different route than that of the first three. Perhaps it learned of a traffic jam somewhere along the *ACE* path and updated its routing table, as shown under the label "later."
- The algorithm that manages the tables and makes the routing decisions is called the **routing algorithm**.

IMPLEMENTATION OF CONNECTION-ORIENTED SERVICE

- For connection-oriented service, we need a virtual-circuit subnet.
- Let us see how that works.
- The idea behind virtual circuits is to avoid having to choose a new route for every packet sent, as in [Fig. 5-2](#). Instead, when a connection is established, a route from the source machine to the destination machine is chosen as part of the connection setup and stored in tables inside the routers.
- That route is used for all traffic flowing over the connection, exactly the same way that the telephone system works.
- When the connection is released, the virtual circuit is also terminated. With connection-oriented service, each packet carries an identifier telling which virtual circuit it belongs to.
- As an example, consider the situation of [Fig. 5-3](#). Here, host *H1* has established connection 1 with host *H2*. It is remembered as the first entry in each of the routing tables. The first line of *A*'s table says that if a packet bearing connection identifier 1 comes in from *H1*, it is to be sent to router *C* and given connection identifier 1. Similarly, the first entry at *C* routes the packet to *E*, also with connection identifier 1.

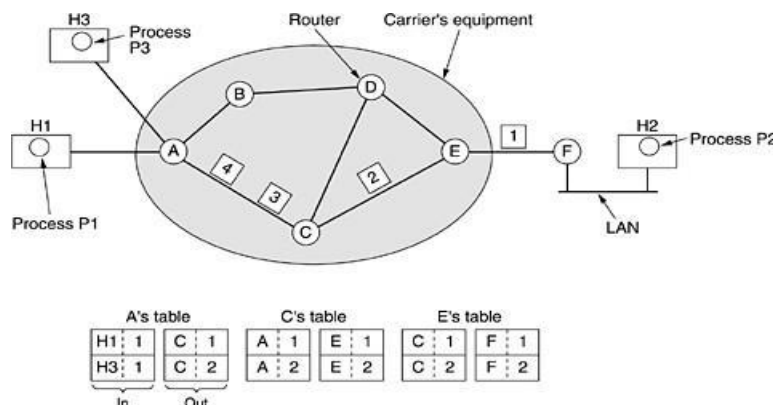


Figure 5-3. Routing within a virtual-circuit subnet.

- Now let us consider what happens if *H3* also wants to establish a connection to *H2*. It chooses connection identifier 1 (because it is initiating the connection and this is its only connection) and tells the subnet to establish the virtual circuit. This leads to the second row in the tables. Note that we have a conflict here because although *A* can easily distinguish connection 1 packets from *H1* from connection 1 packets from *H3*, *C* cannot do this. For this reason, *A* assigns a different connection identifier to the outgoing traffic for the second connection. Avoiding conflicts of this kind is why routers need the ability to replace connection identifiers in outgoing packets. In some contexts, this is called label switching.

COMPARISON OF VIRTUAL-CIRCUIT AND DATAGRAM SUBNETS

The major issues are listed in [Fig. 5-4](#), although purists could probably find a counter example for everything in the figure.

Issue	Datagram subnet	Virtual-circuit subnet
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Routers do not hold state information about connections	Each VC requires router table space per connection
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow it
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Quality of service	Difficult	Easy if enough resources can be allocated in advance for each VC
Congestion control	Difficult	Easy if enough resources can be allocated in advance for each VC

Figure 5-4. Comparison of datagram and virtual-circuit subnets.

2. Discuss about different routing algorithms in detail.

(or)

Discuss shortest path

(

routing.

O

What is flooding? Discuss. (Or) r)

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)

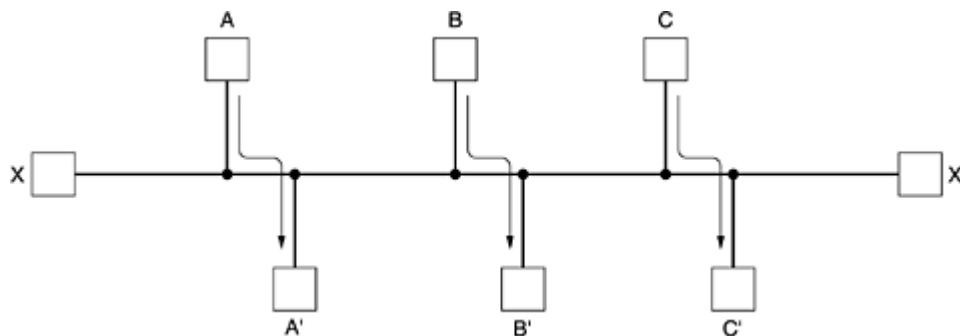
ROUTING ALGORITHMS

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.

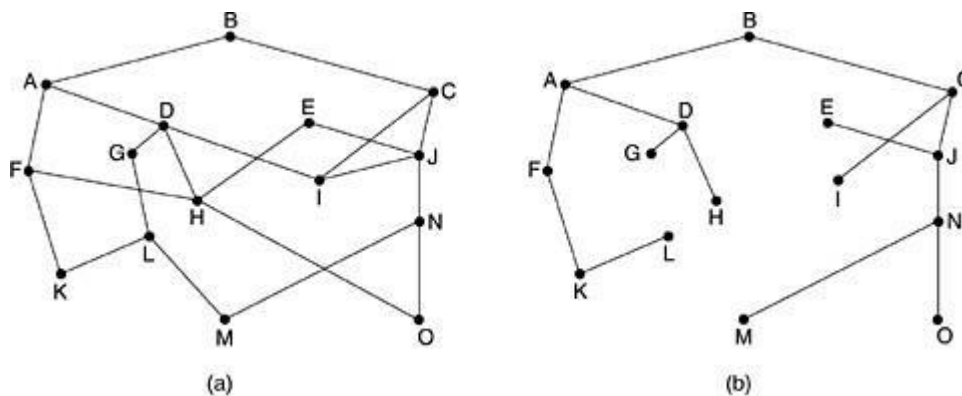
CATEGORY OF ALGORITHM

- 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.



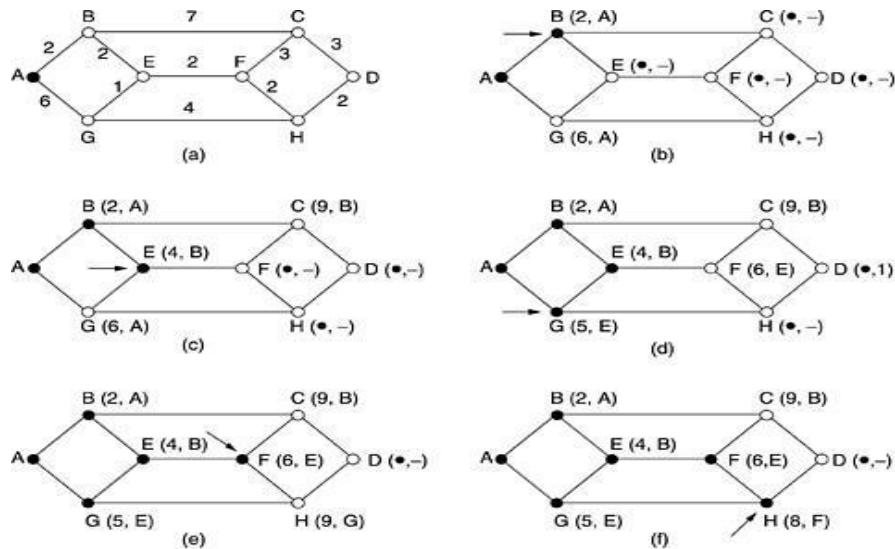
(c) A subnet. (b) A sink tree for router B.

- 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.

```

#define MAX_NODES 1024          /* maximum number of nodes */
#define INFINITY 1000000000     /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES]; /* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                /* the path being worked on */
  int predecessor;              /* previous node */
  int length;                   /* length from source to this node */
  enum {permanent, tentative} label; /* label state */
} state[MAX_NODES];

int i, k, min;
struct state *p;

for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
  p->predecessor = -1;
  p->length = INFINITY;
  p->label = tentative;
}
state[t].length = 0; state[t].label = permanent;
k = t;
do {
  for (i = 0; i < n; i++)
    if (dist[k][i] != 0 && state[i].label == tentative) {
      if (state[k].length + dist[k][i] < state[i].length) {
        state[i].predecessor = k;
        state[i].length = state[k].length + dist[k][i];
      }
    }

  /* Find the tentatively labeled node with the smallest label. */
  k = 0; min = INFINITY;
  for (i = 0; i < n; i++)
    if (state[i].label == tentative && state[i].length < min) {
      min = state[i].length;
      k = i;
    }
  state[k].label = permanent;
} while (k != s);

/* Copy the path into the output array. */
i = 0; k = s;
do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}

```

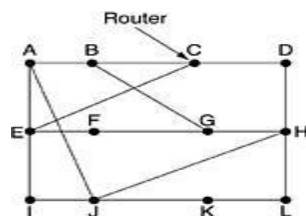
Figure 5-8. Dijkstra's algorithm to compute the shortest path through a graph.

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	
8	A
20	A
28	I
20	H
17	I
30	I
18	H
12	H
10	I
0	-
6	K
15	K

(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.

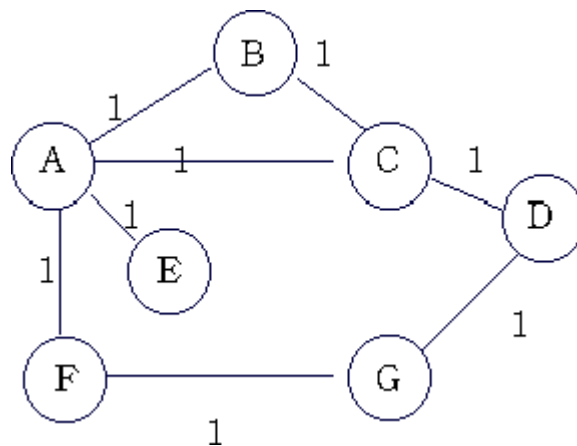


Table 1. Initial distances stored at each node(global view).

Information	Distance to Reach Node						
Stored at 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).

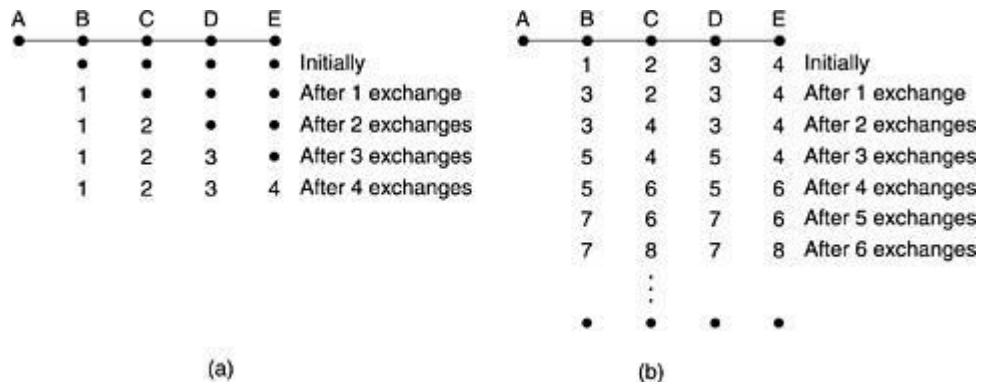
For example, Table 3 shows the complete routing table maintained at node **B** for the network in figure1.

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.



- 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.

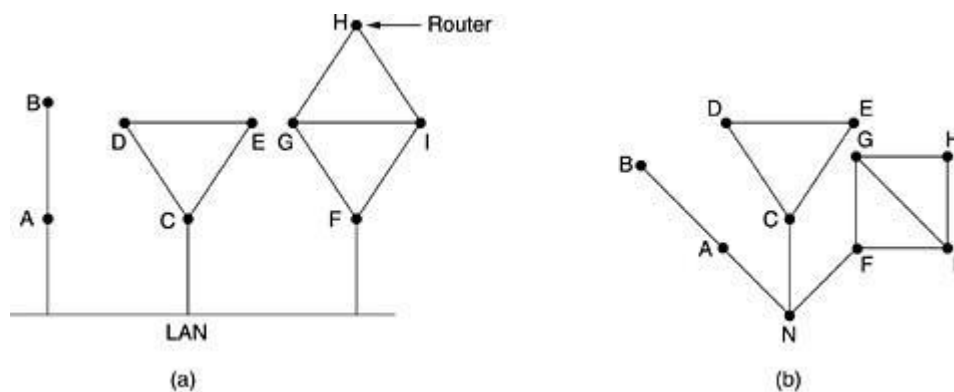
LINK STATE ROUTING

The idea behind link state routing is simple and can be stated as five parts. Each router must do the following:

1. Discover its neighbors and learn their network addresses.
2. Measure the delay or cost to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router

Learning about the Neighbours

When a router is booted, its first task is to learn who its neighbours are. It accomplishes this goal by sending a special HELLO packet on each point-to-point line. The router on the other end is expected to send back a reply telling who it is.



(a)

Nine routers and a LAN. (b)

A graph model of (a). (b)

Measuring Line Cost

- The link state routing algorithm requires each router to know, or at least have a reasonable estimate of, the delay to each of its neighbors. The most direct way to determine this delay is to send over the line a special ECHO packet that the other side is required to send back immediately.
- By measuring the round-trip time and dividing it by two, the sending router can get a reasonable estimate of the delay.
- For even better results, the test can be conducted several times, and the average used. Of course, this method implicitly assumes the delays are symmetric, which may not always be the case.

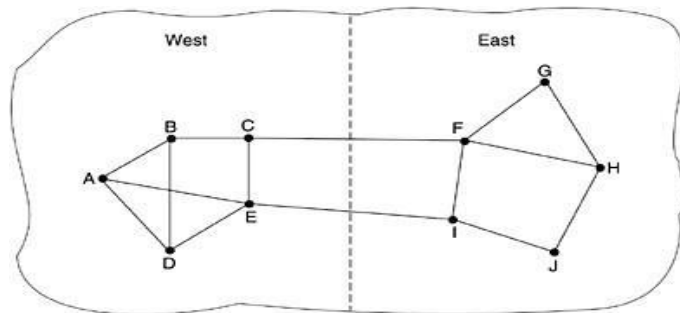
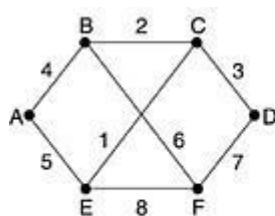


Figure: A subnet in which the East and West parts are connected by two lines.

- Unfortunately, there is also an argument against including the load in the delay calculation. Consider the subnet of [Fig. 5-12](#), which is divided into two parts, East and West, connected by two lines, *CF* and *EI*.

Building Link State Packets



(a)

Link		State		Packets	
A	B	C	D	E	F
Seq.	Seq.	Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

(b)

(a) A subnet. (b) The link state packets for this subnet.

- Once the information needed for the exchange has been collected, the next step is for each router to build a packet containing all the data.
- The packet starts with the identity of the sender, followed by a sequence number and age (to be described later), and a list of neighbours.
- For each neighbour, the delay to that neighbour is given.
- An example subnet is given in [Fig. 5-13\(a\)](#) with delays shown as labels on the lines. The corresponding link state packets for all six routers are shown in [Fig. 5-13\(b\)](#).

Distributing the Link State Packets

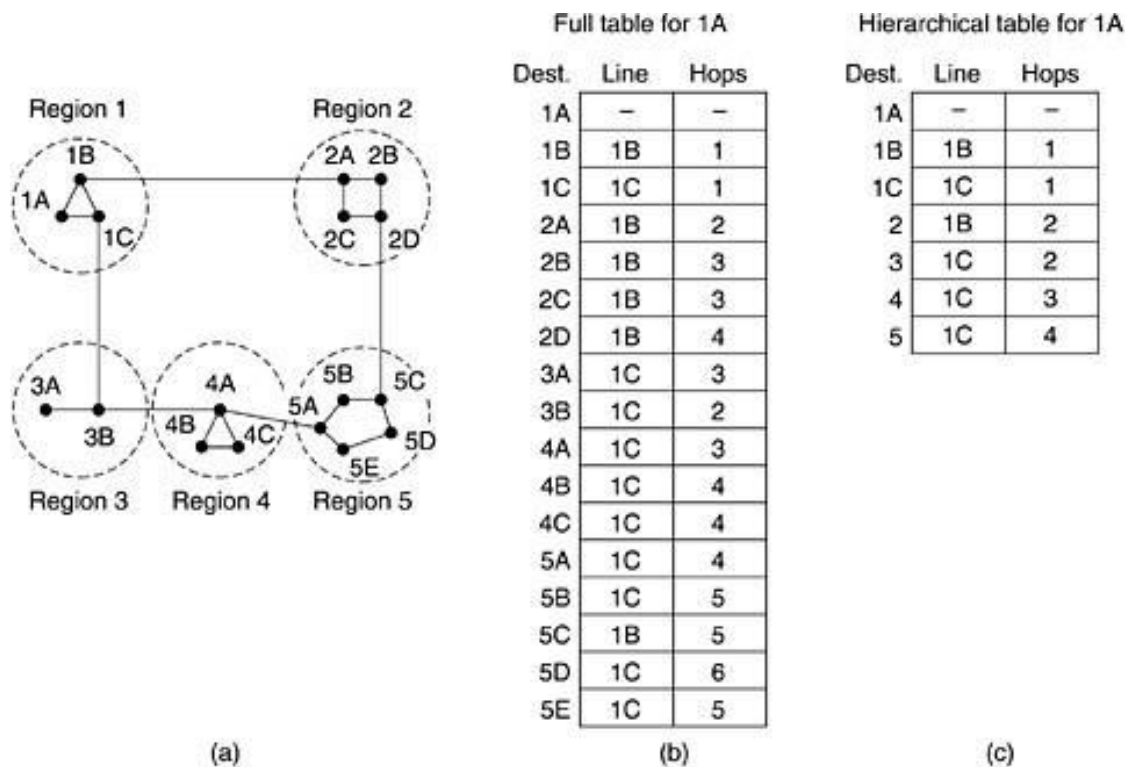
Source	Seq.	Age	Send flags			ACK flags			Data
			A	C	F	A	C	F	
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

The packet buffer for router B in [Fig. 5-13](#).

- In Fig. 5-14, the link state packet from *A* arrives directly, so it must be sent to *C* and *F* and acknowledged to *A*, as indicated by the flag bits.
- Similarly, the packet from *F* has to be forwarded to *A* and *C* and acknowledged to *F*.

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.



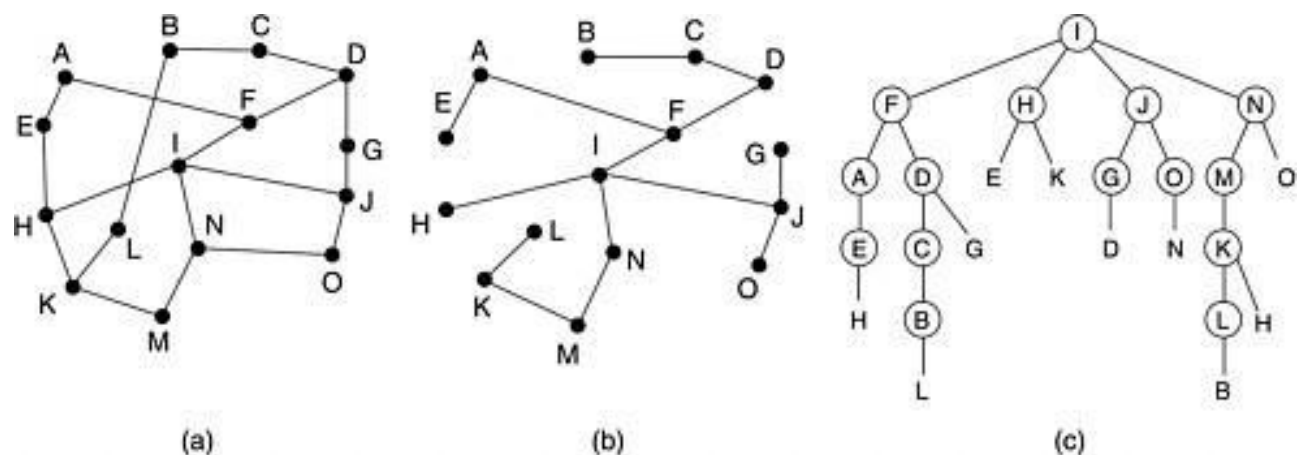
- Figure 5-15 gives a quantitative example of routing in a two-level hierarchy with five regions.
- The full routing table for router 1A has 17 entries, as shown in Fig. 5-15(b).
- When routing is done hierarchically, as in Fig. 5-15(c), there are entries for all the local routers as before, but all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

BROADCAST ROUTING

Sending a packet to all destinations simultaneously is called broadcasting.

- 1) The source simply sends a distinct packet to each destination. Not only is the method wasteful of bandwidth, but it also requires the source to have a complete list of all destinations.
- 2) Flooding.

The problem with flooding as a broadcast technique is that it generates too many packets and consumes too much bandwidth.

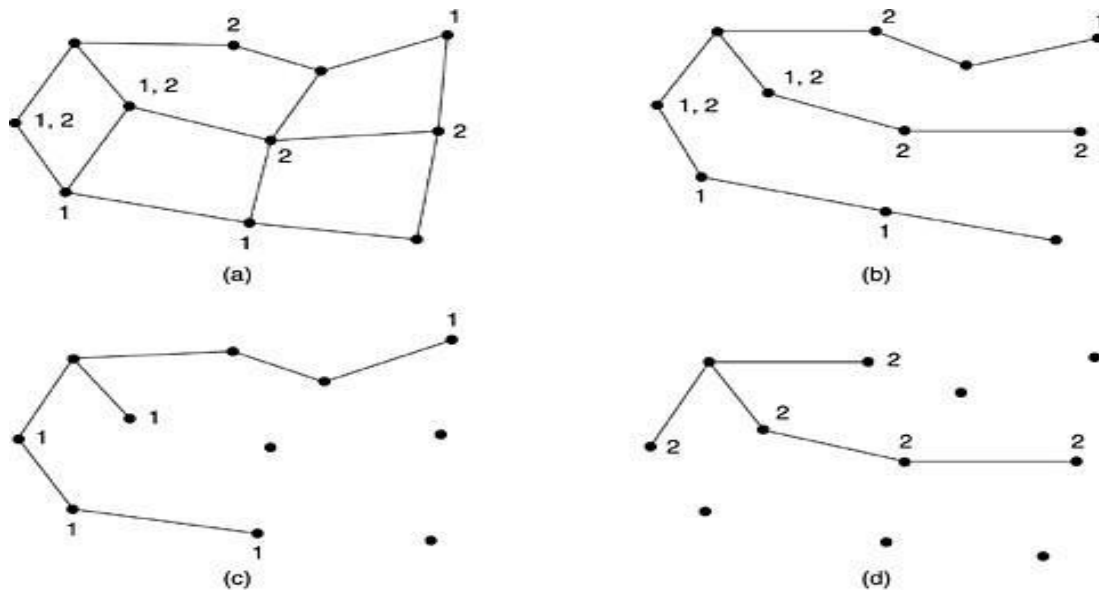


Reverse path forwarding. (a) A subnet. (b) A sink tree. (c) The tree built by reverse path forwarding.

Part (a) shows a subnet, part (b) shows a sink tree for router I of that subnet, and part (c) shows how the reverse path algorithm works.

- When a broadcast packet arrives at a router, the router checks to see if the packet arrived on the line that is normally used for sending packets to the source of the broadcast. If so, there is an excellent chance that the broadcast packet itself followed the best route from the router and is therefore the first copy to arrive at the router.
- This being the case, the router forwards copies of it onto all lines except the one it arrived on. If, however, the broadcast packet arrived on a line other than the preferred one for reaching the source, the packet is discarded as a likely duplicate.

MULTICAST ROUTING



- To do multicast routing, each router computes a spanning tree covering all other routers. For example, in [Fig. 5-17\(a\)](#) we have two groups, 1 and 2.
- Some routers are attached to hosts that belong to one or both of these groups, as indicated in the figure.
- A spanning tree for the leftmost router is shown in [Fig. 5-17\(b\)](#). When a process sends a multicast packet to a group, the first router examines its spanning tree and prunes it, removing all lines that do not lead to hosts that are members of the group.
- In our example, [Fig. 5-17\(c\)](#) shows the pruned spanning tree for group 1. Similarly, [Fig. 5-17\(d\)](#) shows the pruned spanning tree for group 2. Multicast packets are forwarded only along the appropriate spanning tree.

CONGESTION CONTROL ALGORITHMS

- When too many packets are present in (a part of) the subnet, performance degrades. This situation is called **congestion**.
- [Figure 5-25](#) depicts the symptom. When the number of packets dumped into the subnet by the hosts is within its carrying capacity, they are all delivered (except for a few that are afflicted with 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. This tends to make matters worse. At very high traffic, performance collapses completely and almost no packets are delivered.

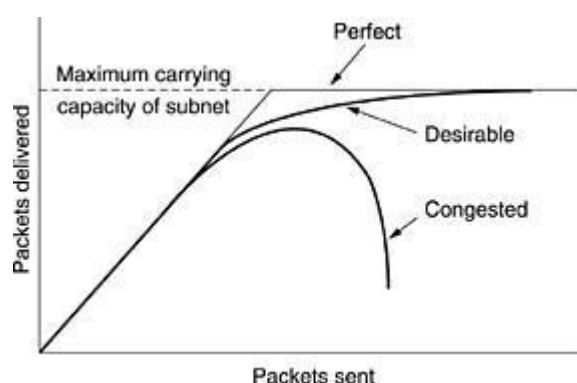


Figure 5-25. When too much traffic is offered, congestion sets in and performance degrades sharply.

- Congestion can be brought on by several factors. If all of a sudden, streams of packets begin arriving on three or four input lines and all need the same output line, a queue will build up.
- If there is insufficient memory to hold all of them, packets will be lost.
- Slow processors can also cause congestion. If the routers' CPUs are slow at performing the bookkeeping tasks required of them (queuing buffers, updating tables, etc.), queues can build up, even though there is excess line capacity. Similarly, low-bandwidth lines can also cause congestion.

APPROACHES TO CONGESTION CONTROL

- Many problems in complex systems, such as computer networks, can be viewed from a control theory point of view. This approach leads to dividing all solutions into two groups: open loop and closed loop.

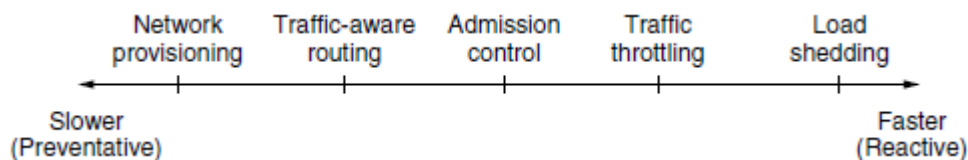


Figure: Timescales Of Approaches To Congestion Control

- Open loop solutions attempt to solve the problem by good design.
- Tools for doing open-loop control include deciding when to accept new traffic, deciding when to discard packets and which ones, and making scheduling decisions at various points in the network.
- Closed 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 to detect when and where congestion occurs.
 2. Pass this information to places where action can be taken.
 3. Adjust system operation to correct the problem.
- A variety of metrics can be used to monitor the subnet for congestion. Chief among these are the percentage of all packets discarded for lack of buffer 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. In all cases, rising numbers indicate growing congestion.
- The second step in the feedback loop is to transfer the information about the congestion from the point where it is detected to the point where something can be done about it.

- In all feedback schemes, the hope is that knowledge of congestion will cause the hosts to take appropriate action to reduce the congestion.
- The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. Two solutions come to mind: increase the resources or decrease the load.

CONGESTION PREVENTION POLICIES

The methods to control congestion by looking at open loop systems. These systems are designed to minimize congestion in the first place, rather than letting it happen and reacting after the fact. They try to achieve their goal by using appropriate policies at various levels. In Fig. 5-26 we see different data link, network, and transport policies that can affect congestion (Jain, 1990).

Layer	Policies
Transport	<ul style="list-style-type: none"> • Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy • Timeout determination
Network	<ul style="list-style-type: none"> • Virtual circuits versus datagram inside the subnet • Packet queueing and service policy • Packet discard policy • Routing algorithm • Packet lifetime management
Data link	<ul style="list-style-type: none"> • Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy

Figure 5-26. Policies that affect congestion.

The data link layer Policies.

- The **retransmission policy** is concerned with how fast a sender times out and what it transmits upon timeout. A jumpy sender that times out quickly and retransmits all outstanding packets using go back n will put a heavier load on the system than will a leisurely sender that uses selective repeat.
- Closely related to this is the **buffering policy**. If receivers routinely discard all out-of-order packets, these packets will have to be transmitted again later, creating extra load. With respect to congestion control, selective repeat is clearly better than go back n.
- **Acknowledgement policy** also affects congestion. If each packet is acknowledged immediately, the acknowledgement packets generate extra traffic. However, if acknowledgements are saved up to piggyback onto reverse traffic, extra timeouts and retransmissions may result. A tight flow control scheme (e.g., a small window) reduces the data rate and thus helps fight congestion.

The **network layer Policies**.

- The choice between using **virtual circuits** and using **datagrams** affects congestion since many congestion control algorithms work only with virtual-circuit subnets.
- **Packet queueing and service policy** relates to whether routers have one queue per input line, one queue per output line, or both. It also relates to the order in which packets are processed (e.g., round robin or priority based).
- **Discard policy** is the rule telling which packet is dropped when there is no space.
- A good **routing algorithm** can help avoid congestion by spreading the traffic over all the lines, whereas a bad one can send too much traffic over already congested lines.
- **Packet lifetime management** deals with how long a packet may live before being discarded. If it is too long, lost packets may clog up the works for a long time, but if it is too short, packets may sometimes time out before reaching their destination, thus inducing retransmissions.

The **transport layer Policies**,

- The **same issues occur as in the data link layer**, but in addition, determining the **timeout interval** is harder because the transit time across the network is less predictable than the transit time over a wire between two routers. If the timeout interval is too short, extra packets will be sent unnecessarily. If it is too long, congestion will be reduced but the response time will suffer whenever a packet is lost.

ADMISSION CONTROL

- One technique that is widely used to keep congestion that has already started from getting worse is **admission control**.
- Once congestion has been signaled, no more virtual circuits are set up until the problem has gone away.
- An alternative approach is to allow new virtual circuits but carefully route all new virtual circuits around problem areas. For example, consider the subnet of [Fig. 5-27\(a\)](#), in which two routers are congested, as indicated.

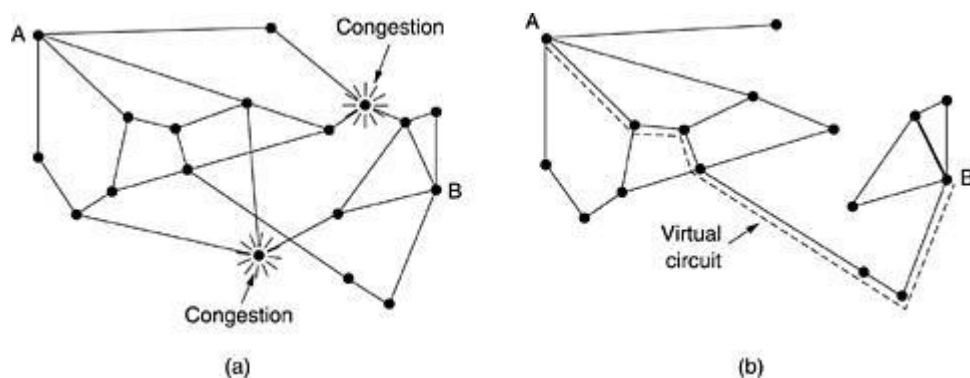
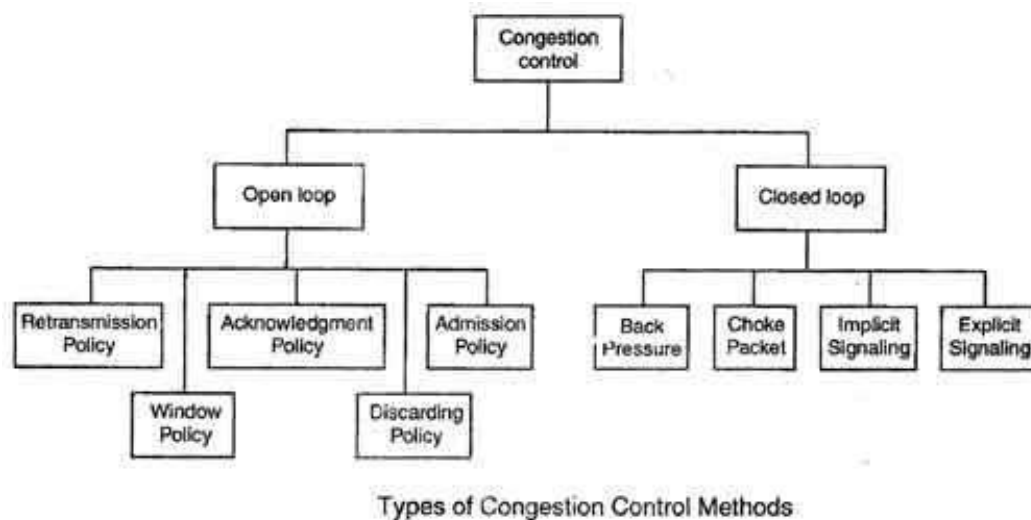


Figure 5-27. (a) A congested subnet. (b) A redrawn subnet that eliminates the congestion. A virtual circuit from A to B is also shown.

Suppose that a host attached to router *A* wants to set up a connection to a host attached to router *B*. Normally, this connection would pass through one of the congested routers. To avoid this situation, we can redraw the subnet as shown in [Fig. 5-27\(b\)](#), omitting the congested routers and all of their lines. The dashed line shows a possible route for the virtual circuit that avoids the congested routers.

How to correct the Congestion Problem:

Congestion Control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened. Congestion control mechanisms are divided into two categories, one category prevents the congestion from happening and the other category removes congestion after it has taken place.



These two categories are:

1. Open loop
2. Closed loop

Open Loop Congestion Control

- In this method, policies are used to prevent the congestion before it happens.
- Congestion control is handled either by the source or by the destination.
- The various methods used for open loop congestion control are:

1. Retransmission Policy

- The sender retransmits a packet, if it feels that the packet it has sent is lost or corrupted.
- However retransmission in general may increase the congestion in the network. But we need to implement good retransmission policy to prevent congestion.
- The retransmission policy and the retransmission timers need to be designed to optimize efficiency and at the same time prevent the congestion.

2. Window Policy

- To implement window policy, selective reject window method is used for congestion control.
- Selective Reject method is preferred over Go-back-n window as in Go-back-n method, when timer for a packet times out, several packets are resent, although some may have arrived safely at the receiver. Thus, this duplication may make congestion worse.
- Selective reject method sends only the specific lost or damaged packets.

3. Acknowledgement Policy

- The acknowledgement 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.
- Acknowledgments also add to the traffic load on the network. Thus, by sending fewer acknowledgements we can reduce load on the network.
- To implement it, several approaches can be used:
 1. A receiver may send an acknowledgement only if it has a packet to be sent.
 2. A receiver may send an acknowledgement when a timer expires.
 3. A receiver may also decide to acknowledge only N packets at a time.

4. Discarding Policy

- A router may discard less sensitive packets when congestion is likely to happen.
- Such a discarding policy may prevent congestion and at the same time may not harm the integrity of the transmission.

5. Admission Policy

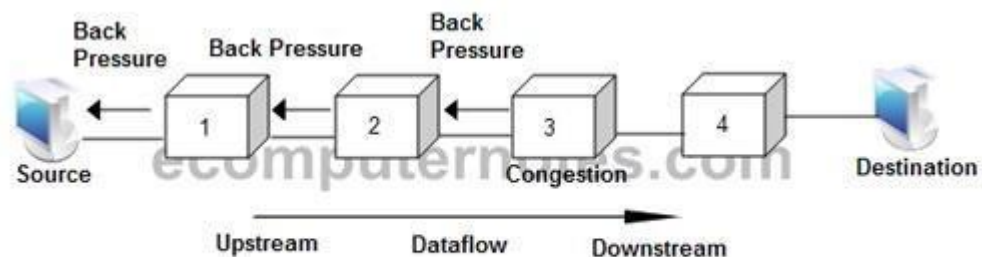
- An admission policy, which is a quality-of-service mechanism, can also prevent congestion in virtual circuit networks.
- Switches in a flow first check the resource requirement of a flow before admitting it to the network.
- A router can deny establishing a virtual circuit connection if there is congestion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

Closed Loop Congestion Control

- Closed loop congestion control mechanisms try to remove the congestion after it happens.
- The various methods used for closed loop congestion control are:

1. Backpressure

- Backpressure is a node-to-node congestion control that starts with a node and propagates, in the opposite direction of data flow.



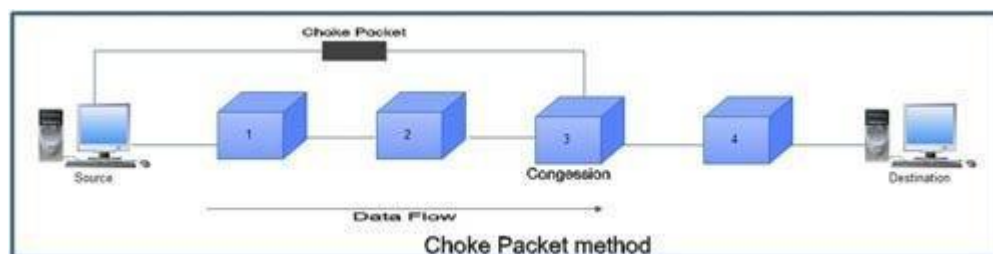
Backpressure Method

- The backpressure technique can be applied only to virtual circuit networks. In such virtual circuit each node knows the upstream node from which a data flow is coming.
- In this method of congestion control, the 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 node or nodes.

- As shown in fig node 3 is congested and it stops receiving packets and informs its upstream node 2 to slow down. Node 2 in turns may be congested and informs node 1 to slow down. Now node 1 may create congestion and informs the source node to slow down. In this way the congestion is alleviated. Thus, the pressure on node 3 is moved backward to the source to remove the congestion.

2. Choke Packet

- In this method of congestion control, congested router or node sends a special type of packet called choke packet to the source to inform it about the congestion.
- Here, congested node does not inform its upstream node about the congestion as in backpressure method.
- In choke packet method, congested node sends a warning directly to the source station *i.e.* the intermediate nodes through which the packet has traveled are not warned.



3. Implicit Signaling

- In implicit signaling, there is no communication between the congested node or nodes and the source.
- The source guesses that there is congestion somewhere in the network when it does not receive any acknowledgment. Therefore the delay in receiving an acknowledgment is interpreted as congestion in the network.
- On sensing this congestion, the source slows down.
- This type of congestion control policy is used by TCP.

4. Explicit Signaling

- In this method, the congested nodes explicitly send a signal to the source or destination to inform about the congestion.
- Explicit signaling is different from the choke packet method. In choke packet method, a separate packet is used for this purpose whereas in explicit signaling method, the signal is included in the packets that carry data .
- Explicit signaling can occur in either the forward direction or the backward direction .

- In backward signaling, a bit is set in a packet moving in the direction opposite to the congestion. This bit warns the source about the congestion and informs the source to slow down.
- In forward signaling, a bit is set in a packet moving in the direction of congestion. This bit warns the destination about the congestion. The receiver in this case uses policies such as slowing down the acknowledgements to remove the congestion.

