NETWORK LAYER

- The network layer is concerned with getting packets from the source all the way to the destination
- The network layer is the lowest layer that deals with end-to-end transmission.
- To achieve its goals, the network layer must know about the topology of the network (i.e., the set of all routers and links) and choose appropriate paths through
- it, even for large networks It must also take care when choosing routes to avoid overloading some of the communication lines and routers while leaving others idle.
- Finally, when the source and destination are in different networks, new problems occur. It is up to the network layer to deal with them.

Network Layer Design Issues

- Store-and-Forward Packet Switching
- Services Provided to the Transport Layer
- Implementation of Connectionless Service
- Implementation of Connection-Oriented Service
- Comparison of Virtual-Circuit and Datagram Subnets

Store-and-Forward Packet Switching

The major components of the network are the ISP's equipment

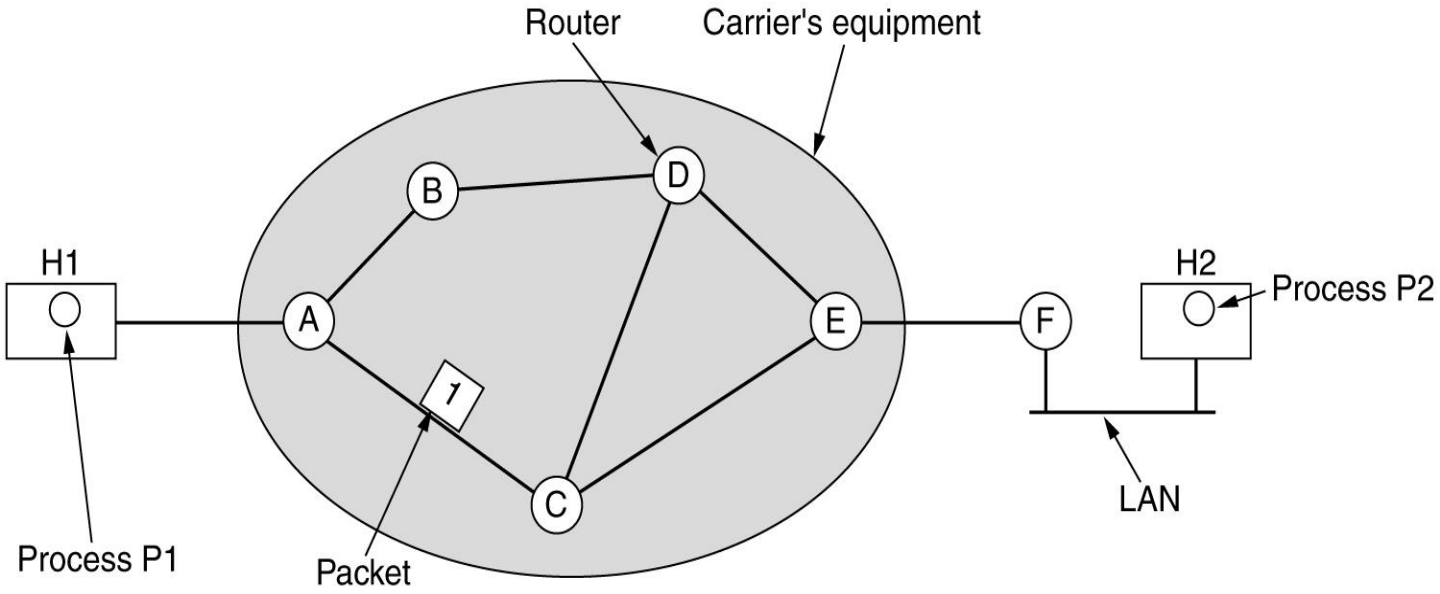
(routers connected by transmission lines), shown inside the shaded oval, and the customers' equipment, shown outside the oval. Host *H1* is directly connected to one of the ISP's routers, *A*, perhaps as a home computer that is plugged into a DSL modem.

In contrast, *H2* is on a LAN, which might be an office Ethernet, with a router, *F*, owned and operated by the customer.

This router has a leased line to the ISP's equipment.

We have shown *F* as being outside the oval because it does not belong to the ISP.

The routers on customer premises are considered part of the ISP network because they run the same algorithms as the ISP's routers.



- 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 ISP.
- The packet is stored there until it has fully arrived and the link has finished its processing by verifying the checksum.
- 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 services need to be carefully 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.

- The network layer should provide connection oriented service or connectionless service.
- The network is inherently unreliable, no matter how it is designed.
- 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.
- Each packet must carry the full destination address, because each packet sent is carried independently of its predecessors, if any.
- The network should provide a reliable, connection-oriented service.
- The quality of service is the dominant factor, and without connections in the network, quality of service is very difficult to achieve, especially for real-time traffic such as voice and video.
- Data networks, such as X.25 in the 1970s and its successor Frame Relay in the 1980s, were connection-oriented.
- However, since the days of the ARPANET and the early Internet, connectionless network layers have grown tremendously in popularity. The IP protocol is now an ever-present symbol of success.
- Two examples of connection-oriented technologies are MPLS (MultiProtocol Label Switching) and VLANs

Implementation of Connectionless Service

If connectionless service is offered, packets are injected into the network individually and routed independently of each other.

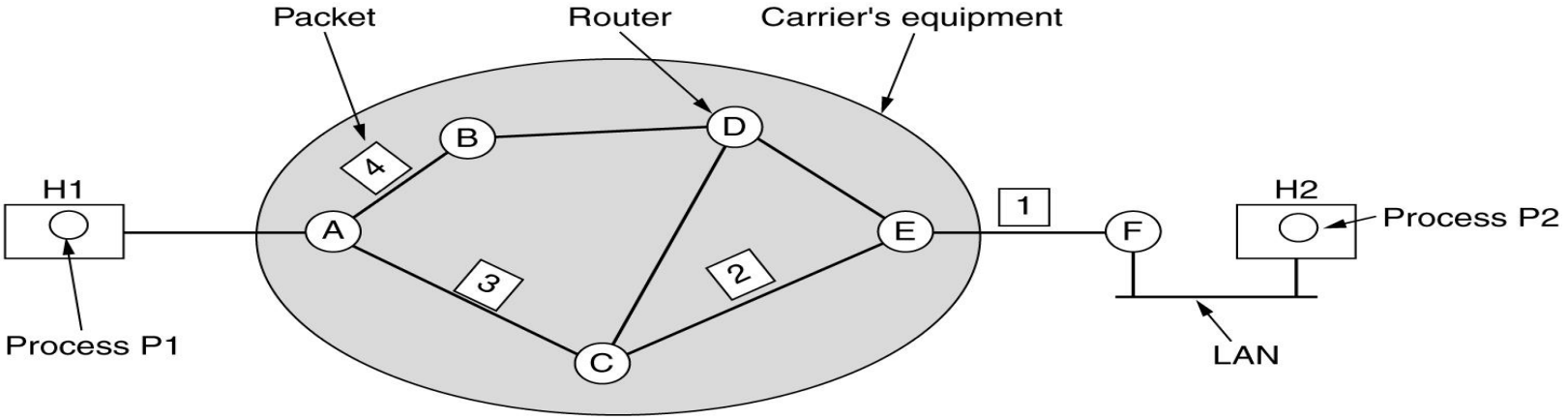
No advance setup is needed.

In this context, the packets are frequently called **datagrams** (in analogy with telegrams) and the network is called a **datagram network**.

If connection-oriented service is used, a path from the source router all the way 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 network is called a **virtual-circuit network**.

Suppose that the process *P1* 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.



A's table

Dest.	initially	later
A	–	–
B	B	B
C	C	C
D	B	B
E	C	B
F	C	B

C's table

Dest.	Line
A	A
B	A
C	–
D	D
E	E
F	E

E's table

Dest.	Line
A	C
B	D
C	C
D	D
E	–
F	F

Let us assume for this example 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 send each of them in turn to router *A* using some point-to-point protocol.

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, *A* has only two outgoing lines—to *B* and to *C*—so every incoming packet must be sent to one of these routers, even if the ultimate destination is to some other router.

Packet 1,2 and 3

A's initial routing table is shown in the figure under the label "initially."

At *A*, packets 1, 2, and 3 are stored briefly, having arrived on the incoming link and had their checksums verified.

Then each packet is forwarded according to *A*'s table, onto the outgoing link to *C* within a new frame.

Packet 1 is then forwarded to *E* and then to *F*.

When it gets to *F*, it is sent within a frame over the LAN to *H2*.

Packets 2 and 3 follow the same route.

packet 4

When it gets to *A* it is 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 packets.

Perhaps it has 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**

IP (Internet Protocol), which is the basis for the entire Internet, is the dominant example of a connectionless network service.

Each packet carries a destination IP address that routers use to individually forward each packet.

The addresses are 32 bits in IPv 4 packets and 128 bits in IPv6 packets.

Implementation of Connection-Oriented Service

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.

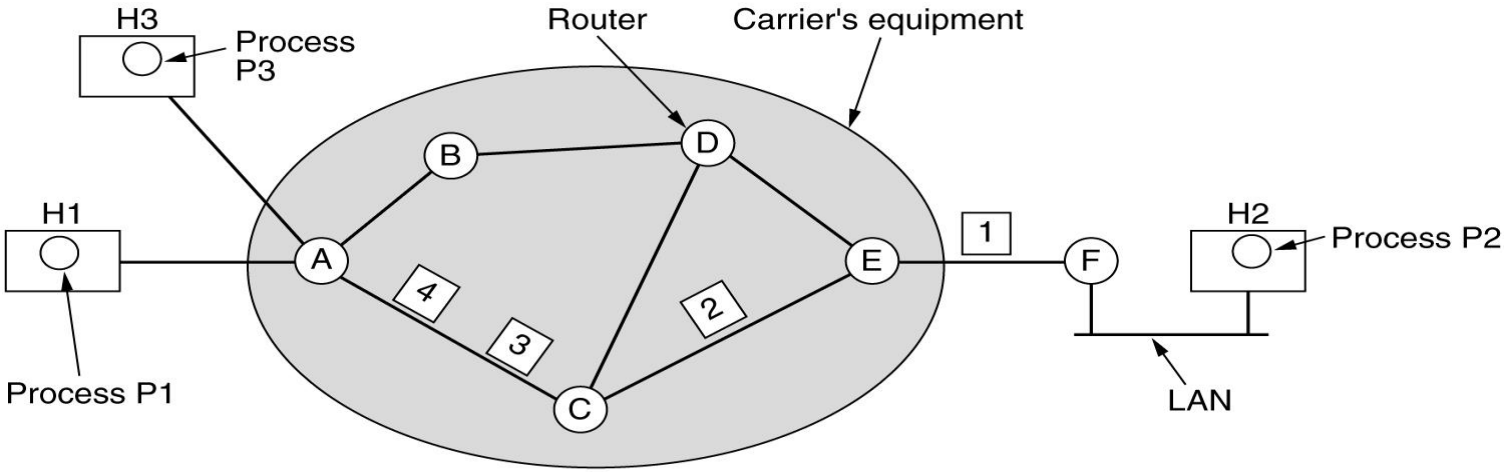
Host *H1* has established connection 1 with host *H2*.

This connection 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.

Routing within a virtual-circuit network



A's table

In		Out	
H1	1	C	1
H3	1	C	2

C's table

A	1	E	1
A	2	E	2

E's table

C	1	F	1
C	2	F	2

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 network 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 process is called **label switching**.

An example of a connection-oriented network service is **MPLS** (**MultiProtocol Label Switching**).

It is used within ISP networks in the Internet, with IP packets wrapped in an MPLS header having a 20-bit connection identifier or label.

MPLS is often hidden from customers, with the ISP establishing long-term connections for large amounts of traffic, but it is increasingly being used to help when quality of service is important but also with other ISP traffic management tasks.

Comparison of Virtual-Circuit and Datagram Networks

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

ROUTING ALGORITHMS

The part of the Network Layer responsible for deciding on which output line to transmit an incoming packet.

If the network uses datagrams internally, this decision must be made a new for every arriving data packet since the best route may have changed since last time.

If the network uses virtual circuits internally, routing decisions are made only when a new virtual circuit is being set up.

Thereafter, data packets just follow the already established route.

The latter case is sometimes called **session routing** because a route remains in force for an entire session (e.g., while logged in over a VPN).

It is sometimes useful to make a distinction between routing, which is making the decision which routes to use, and forwarding, which is what happens when a packet arrives.

A router as having two processes inside it.

- One of them handles each packet as it arrives, looking up the outgoing line to use for it in the routing tables. This process is **forwarding**.
- The other process is responsible for filling in and updating the routing tables.

Algorithm properties

correctness, simplicity, robustness, stability, fairness, optimality and scalability.

Correctness and simplicity hardly require comment, but the need for robustness may be less obvious at first.

- Once a major network comes on the air, it may be expected to run continuously for years without system-wide failures.
- During that period there will be hardware and software failures of all kinds.
- Hosts, routers, and lines will fail repeatedly, and the topology will change many times.
- The routing algorithm should be able to cope with changes in the topology and traffic without requiring all jobs in all hosts to be aborted.
- Imagine the havoc if the network needed to be rebooted every time some router crashed!

Stability is also an important goal for the routing algorithm.

- There exist routing algorithms that never converge to a fixed set of paths, no matter how long they run.
- A stable algorithm reaches equilibrium and stays there.
- It should converge quickly too, since communication may be disrupted until the routing algorithm has reached equilibrium.

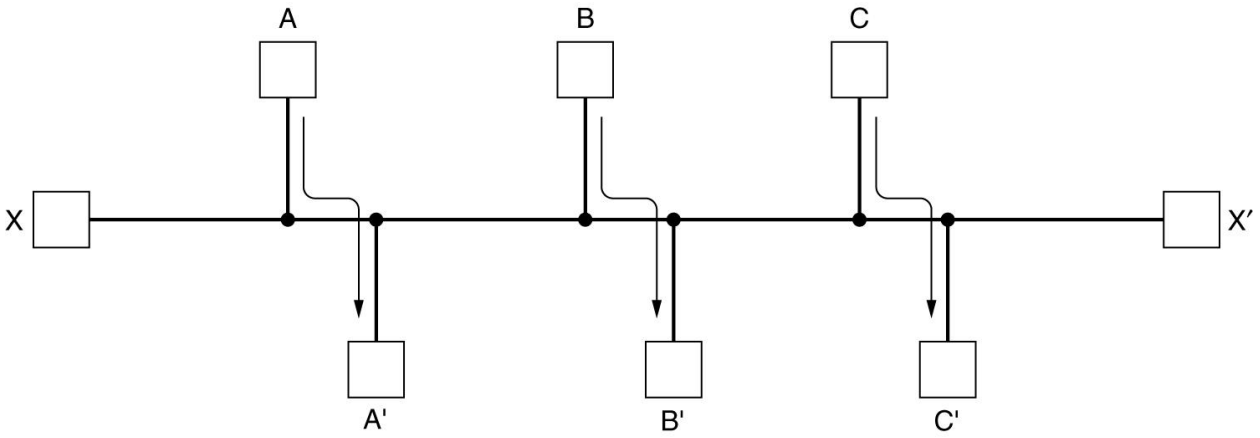
Fairness and efficiency may sound traffic between A and A', between B and B', and between C and C' to saturate the horizontal links.

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

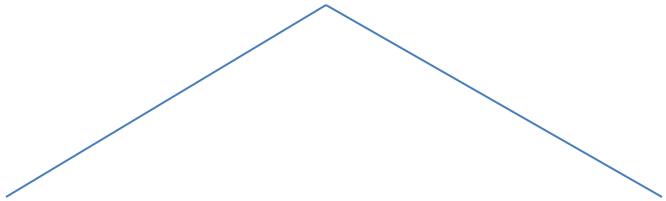
needed.

Minimizing the mean packet delay is an obvious candidate to send traffic through the network effectively, but so is maximizing total network throughput.



Conflict between fairness and optimality.

Routing algorithms



nonadaptive adaptive

Non-Adaptive Routing

- routing computed in advance and off-line
- Flooding
- static routing using shortest path algorithms
- it does not respond to failures
- Static routing is mostly useful for situations in which the routing choice is clear

Adaptive Routing

based on current measurements of traffic and/or topology.

centralized

isolated

Distributed

Dynamic Routing

Change their routing decisions to reflect changes in the topology, and sometimes changes in the traffic as well.

Routing Algorithms

- The Optimality Principle
- Shortest Path Routing
- Flooding
- Distance Vector Routing
- Link State Routing
- Hierarchical Routing
- Broadcast Routing
- Multicast Routing
- Routing for Mobile Hosts
- Routing in Ad Hoc Networks

Optimality Principle

optimality principle (Bellman, 1957).

It states that 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.

To see this, call the part of the route from *I* to *J* *r*1 and the rest of the route *r* 2.

If a route better than *r* 2 existed from *J* to *K*, it could be concatenated with *r* 1 to improve the route from *I* to *K*, contradicting our statement that *r* 1*r* 2 is optimal.

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.**

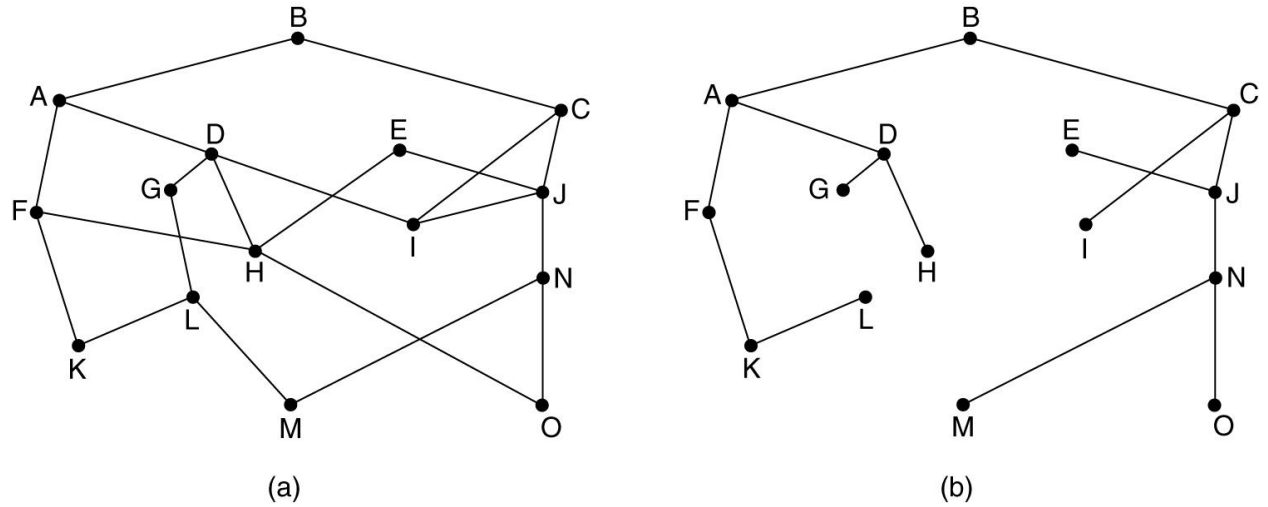


Figure 5-6. (a) A network. (b) A sink tree for router *B*.

DAG (**Directed Acyclic Graph**)

DAGshave no loops. We will use sink trees as a convenient shorthand for both cases.

Both cases also depend on the technical assumption that the paths do not interfere with each other so, for example, a traffic jam on one path will not cause another path to divert.

Since a sink tree is indeed a tree, it does not contain any loops, so each packet will be delivered within a finite and bounded number of hops.

Shortest Path Algorithm

A graph of the network, with each node of the graph representing a router and each edge of the graph representing a communication line, or link.

To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.

The concept of a **shortest path** deserves some explanation.

One way of measuring path length is the number of hops.

Using this metric, the paths *ABC* and *ABE* in Fig. are equally long.

Another metric is the geographic distance in kilometers, in which case *ABC* is clearly much longer than *ABE* (assuming the figure is drawn to scale).

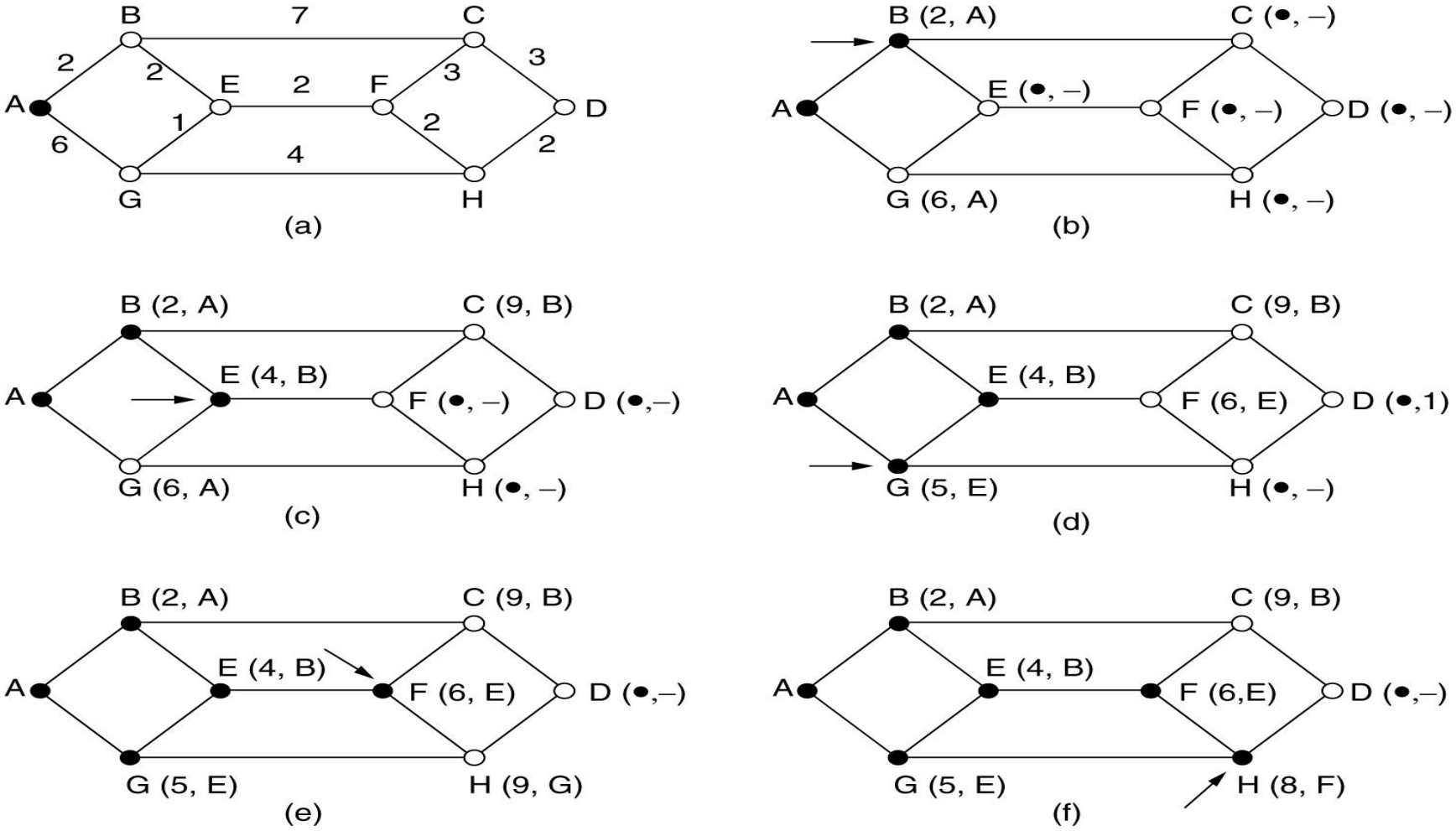


Figure 5-7. The first six steps used in computing the shortest path from *A* to *D*.

The arrows indicate the working node.

For example, each edge could be labeled with the mean delay of a standard test packet, as measured by hourly runs.

With this graph labeling, the shortest path is the fastest path rather than the path with the fewest edges or kilometers.

In the general case, the labels on the edges could be computed as a function of the distance, bandwidth, average traffic, communication cost, 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.

Dijkstra-Shortest Path Algorithm

- **Dijkstra**(1959) and finds the shortest paths between a source and all destinations in the network.
- Each node is labeled (in parentheses) with its distance from the source node along the best known path.
- The distances must be non-negative, as they will be if they are based on real quantities like bandwidth and delay.
- Initially, no paths are known, so all nodes are labeled with infinity.
- As the algorithm proceeds and paths are found, the labels may change, reflecting better paths.
- A label may be either tentative or permanent.
- Initially, all labels are tentative.
- When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

- To illustrate how the labeling 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 relabeled, we also label it with the node from which the probe was made so that we can reconstruct the final path later.
- If the network had more than one shortest path from *A* to *D* and we wanted to find all of them, we would need to remember all of the probe nodes that could reach a node with the same distance.

- Having examined each of the nodes adjacent to *A*, we examine all the tentatively labeled 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
- 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 labeled 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 six steps of the algorithm.

- This algorithm is given in Fig. 5-8. The global variables *n* and *dist* describethe 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 paths from *t* to *s* in an undirected graph are the same as the shortest paths from *s* to *t*, it does not matter at which end we begin.

- The reason for searching backward is that each node is labeled with its predecessor rather than its successor.

- When the final path is copied into the output variable, *path*,
- the path is thus reversed.

- The two reversal effects cancel, and the answer is produced in the correct order.

5.2.3 Flooding

- When a routing algorithm is implemented, each router must make decisions based on local knowledge, not the complete picture of the network.

- A simple local technique 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 that 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 network.

- Flooding with a hop count can produce an exponential number of duplicate packets as the hop count grows and routers duplicate packets they have seen before.

- A better technique for damming the flood is to have routers keep track of which packets have been flooded, to avoid sending them out a second time.

- One way to achieve this goal is to have the source router put a sequence number in each packet it receives from its hosts.

- Each router then needs a list per source router telling which sequence numbers originating at that source have already been seen.

- If an incoming packet is on the list, it is not flooded.

Dijkstra's algorithm to compute the shortest path through a graph.

```
#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;                                /* k is the initial working node */
```

```
  do {                                          /* Is there a better path from k? */
      for (i = 0; i < n; i++)                   /* this graph has n nodes */
          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);
}
```

To prevent the list from growing without bound, each list should be augmented by a counter, *k*, meaning that all sequence numbers through *k* have been seen.

When a packet comes in, it is easy to check if the packet has already been flooded (by comparing its sequence number to *k*; if so, it is discarded.

Furthermore, the full list below *k* is not needed, since *k* effectively summarizes it.

Flooding is not practical for sending most packets, but it does have some important uses.

1.It ensures that a packet is delivered to every node in the network.

- This may be wasteful if there is a single destination that needs the packet, but it is effective for broadcasting information.
- In wireless networks, all messages transmitted by a station can be received by all other stations within its radio range, which is, in fact, flooding, and some algorithms utilize this property.

2.Flooding is tremendously robust.

- Even if large numbers of routers are blown to bits (e.g., in a military network located in a war one), flooding will find a path if one exists, to get a packet to its destination.
- Flooding also requires little in the way of setup.
- The routers only need to know their neighbors.
- This means that flooding can be used as a building block for other routing algorithms that are more efficient but need more in the way of setup.
- Flooding can also be used as a metric against which other routing algorithms can be compared. Flooding always chooses the shortest path because it chooses every possible path in parallel.
- Consequently, no other algorithm can produce a shorter delay.

5.2.4 Distance Vector Routing

Computer networks generally use dynamic routing algorithms that are more complex than flooding, but more efficient because they find shortest paths for the current topology.

Two dynamic algorithms in particular, distance vector routing and link state routing, are the most popular.

A **distance vector routing** algorithm operates by having each router maintain a table (i.e., a vector) giving the best known distance to each destination and which link to use to get there.

These tables are updated by exchanging information with the neighbors.

Eventually, every router knows the best link to reach each destination.

The distance vector routing algorithm is sometimes called by other names, most commonly the distributed **Bellman-Ford** routing 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.

- In distance vector routing, each router maintains a routing table indexed by, and containing one entry for each router in the network.
- This entry has two parts:
- the preferred outgoing line to use for that destination and an estimate of the distance to that destination.
- The distance might be measured as the number of hops or other metrics.
- The router is assumed to know the "distance" to each of its neighbors.
- If the metric is hops, the distance is just one hop.
- If the metric is propagation delay, the router can measure it directly with special ECHO packets that the receiver just time stamps and sends back as fast as it can.

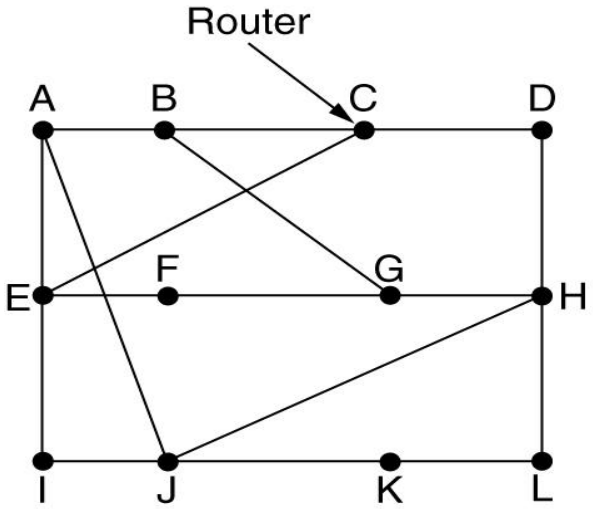
Example

- Assume that delay is used as a metric and that the router knows the delay to each of its neighbors.
- Once every *T* msec, each router sends to each neighbor a list of its estimated delays to each destination.
- It also receives a similar list from each neighbor.
- Imagine that one of these tables has just come in from neighbor *X*, with *Xi* being *X*'s estimate of how long it takes to get to router *i*.
- If the router knows that the delay to *X* is *m* msec, it also knows that it can reach router *i* via *X* in *Xi* + *m* msec.
- By performing this calculation for each neighbor, a router can find out which estimate seems the best and use that estimate and the corresponding link in its new routing table.

Note that the old routing table is not used in the calculation.

- Fig. 5-9. Part (a) shows a network.
- The first four columns of part (b) show the delay vectors received from the neighbors 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 neighbors, *A*, *I*, *H*, and *K*, as 8, 10, 12, and 6 msec, respectively.
- Consider how *J* computes its new route to router *G*.
- It knows that it can get to *A* in 8 msec, and furthermore *A* claims to be able to get to *G* in 18 msec, so *J* knows it can count on a delay of 26 msec to *G* if it forwards packets bound for *G* to *A*.
- Similarly, it computes the delay to *G* via *I*, *H*, and *K* as 41 (31 + 10), 18 (6 + 12), and 37 (31 + 6) msec, respectively.
- The best of these values is 18, so it makes an entry in its routing table that the delay to *G* is 18 msec and that the route to use is via *H*.
- The same calculation is performed for all the other destinations,
- with the new routing table shown in the last column of the figure.

Figure 5-9. (a) A network. (b) Input from *A*, *I*, *H*, *K*, and the new routing table for *J*.



(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	New routing table for J	

Vectors received from J's four neighbors

(b)

The Count-to-Infinity Problem

- The settling of routes to best paths across the network is called **convergence**.
- Distance vector routing is useful as a simple technique by which routers can collectively compute shortest paths, but it has a serious drawback in practice:
- although it converges to the correct answer, it may do so slowly.

Drawback

- Consider a router whose best route to destination *X* is long.
- If, on the next exchange, neighbor *A* suddenly reports a short delay to *X*, the router just switches over to using the line to *A* to send traffic to *X*.

Advantage

consider the five-node (linear) network

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.

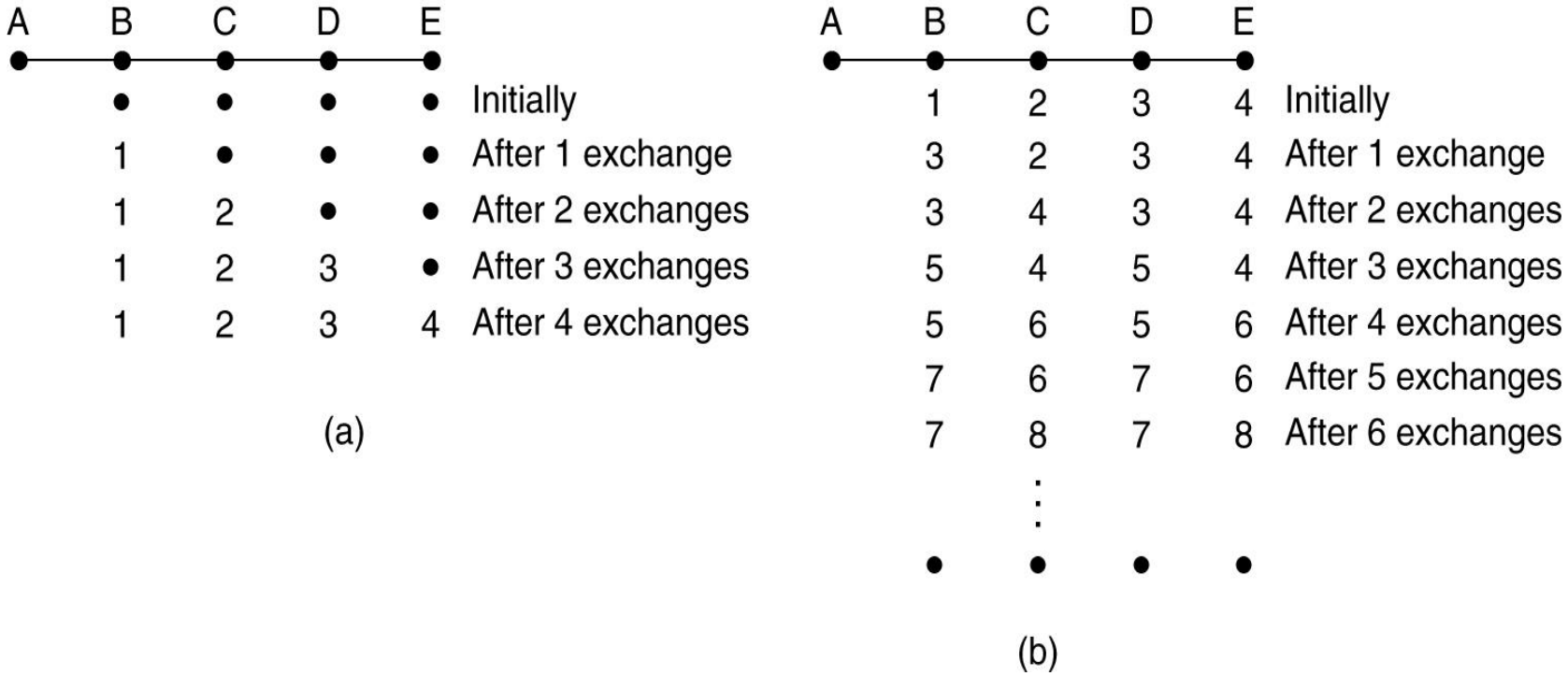


Figure 5-10. The count-to-infinity problem.

- When *A* comes up, the other routers learn about it via the vector exchanges.
- For simplicity, we will assume that there is a gigantic gong somewhere that is struck periodically to initiate a vector exchange at all routers simultaneously.
- At the time of the first exchange, *B* learns that its left-hand neighbor has zero delay to *A*.
- *B* now makes an entry in its routing table indicating that *A* is one hop away to the left.
- All the other routers still think that *A* is down. At this point, the routing table entries for *A* are as shown in the second row of Fig. 5-10(a).
- On the next exchange, *C* learns that *B* has a path of length 1 to *A*, so it updates its routing table to indicate a path of length 2, but *D* and *E* do not hear the good news until later.
- Clearly, the good news is spreading at the rate of one hop per exchange.
- In a network whose longest path is of length *N* hops, within *N* exchanges everyone will know about newly revived links and routers.

- Now let us consider the situation of Fig. 5-10(b), in which all the links and routers are initially up.
- Routers *B*, *C*, *D*, and *E* have distances to *A* of 1, 2, 3, and 4 hops, respectively.
- Suddenly, either *A* goes down or the link between *A* and *B* is cut (which is effectively the same thing from *B*'s point of view).
- At the first packet exchange, *B* does not hear anything from *A*.
- Fortunately, *C* says "Do not worry; I have a path to *A* of length 2."
- Little does *B* suspect that *C*'s path runs through *B* itself.
- For all *B* knows, *C* might have ten links all with separate paths to *A* of length 2.
- As a result, *B* thinks it can reach *A* via *C*, with a path length of 3.
- *D* and *E* do not update their entries for *A* on the first exchange.
- On the second exchange, *C* notices that each of its neighbors claims to have a path to *A* of length 3.
- It picks one of them at random and makes its new distance to *A* 4, as shown in the third row of Fig. 5-10(b).
- Subsequent exchanges produce the history shown in the rest of Fig. 5-10(b).

- From this figure, it should be clear why bad news travels slowly:
- no router ever has a value more than one higher than the minimum of all its neighbors.
- Gradually, all routers work their way up to infinity, but the number of exchanges required depends on the numerical value used for infinity.
- For this reason, it is wise to set infinity to the longest path plus 1.
- This problem is known as the **count-to-infinity** problem.
- There have been many attempts to solve it, for example, preventing routers from advertising their best paths back to the neighbors from which they heard them with the split horizon with poisoned reverse rule discussed in RFC 1058.
- However, none of these heuristics work well in practice despite the colorful names.
- The core of the problem is that when *X* tells *Y* that it has a path somewhere, *Y* has no way of knowing whether it itself is on the path.

5.2.5 Link State Routing

- Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing.
- The primary problem that caused its demise was that the algorithm often took too long to converge after the network topology changed (due to the count-to-infinity problem).
- Consequently, it was replaced by an entirely new algorithm, now called **link state routing**.
- Variants of link state routing called IS-IS and OSPF are the routing algorithms that are most widely used inside large networks and the Internet today.

The idea behind link state routing is fairly simple and can be stated as five parts.
Each router must do the following things to make it work:

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

1. Learning about the Neighbors

When a router is booted, its first task is to learn who its neighbors 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 giving its name.

These names must be globally unique because when a distant router later hears that three routers are all connected to *F*, it is essential that it can determine whether all three mean the same *F*.

When two or more routers are connected by a broadcast link (e.g., a switch, ring, or classic Ethernet), the situation is slightly more complicated.

Fig. 5-11(a) illustrates a broadcast LAN to which three routers, *A*, *C*, and *F*, are directly connected.

Each of these routers is connected to one or more additional routers, as shown.

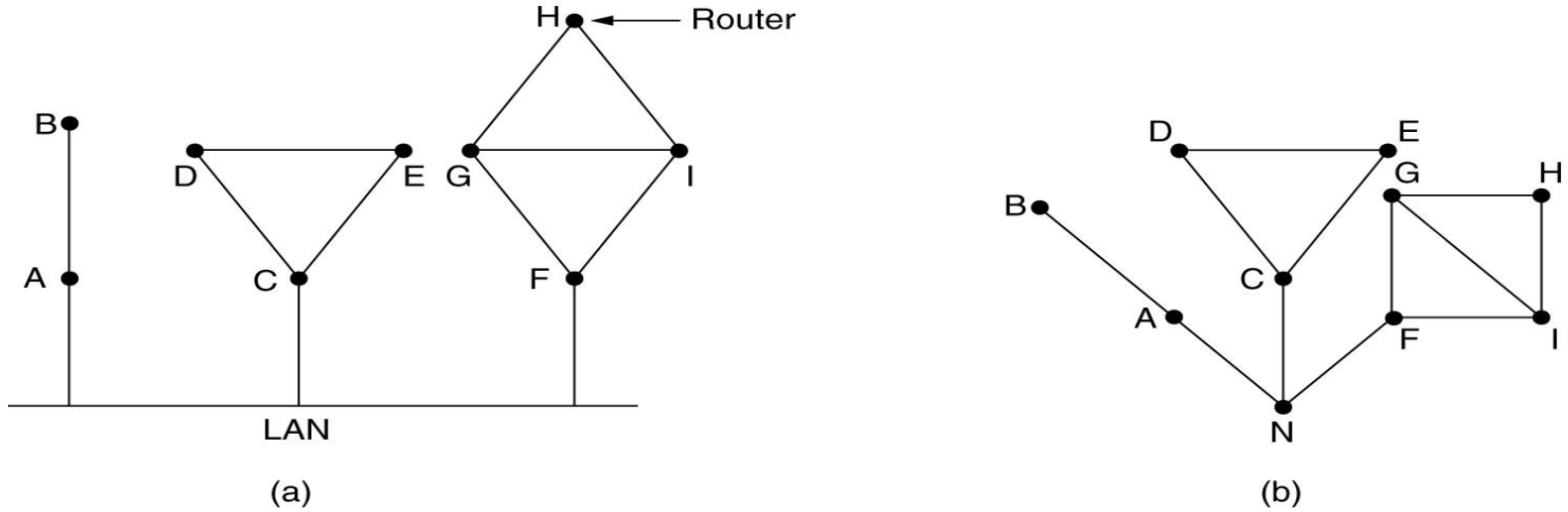


Figure 5-11. (a) Nine routers and a broadcast LAN. (b) A graph model of (a).

- The broadcast LAN provides connectivity between each pair of attached routers.
- However, modeling the LAN as many point-to-point links increases the size of the topology and leads to wasteful messages.
- A better way to model the LAN is to consider it as a node itself, as shown in Fig. 5-11(b).
- Here, we have introduced a new, artificial node, *N*, to which *A*, *C*, and *F* are connected.
- One **designated router** on the LAN is selected to play the role of *N* in the routing protocol.
- The fact that it is possible to go from *A* to *C* on the LAN is represented by the path *ANC* here.

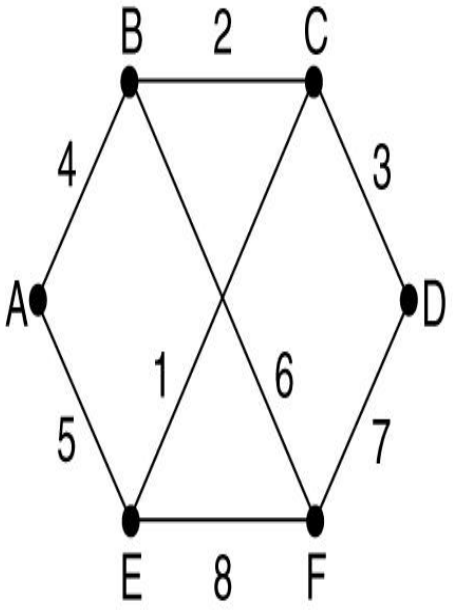
2. Setting Link Costs

- The link state routing algorithm requires each link to have a distance or cost metric for finding shortest paths.
- The cost to reach neighbors can be set automatically, or configured by the network operator.
- A common choice is to make the cost inversely proportional to the bandwidth of the link.
- For example, 1-Gbps Ethernet may have a cost of 1 and 100-Mbps Ethernet a cost of 10.
- This makes higher-capacity paths better choices.
- If the network is geographically spread out, the delay of the links may be factored into the cost so that paths over shorter links are better choices.
- 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.

3. Building Link State Packets

- 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 and a list of neighbors.
- The cost to each neighbor is also given.
- An example network is presented in Fig. 5-12(a) with costs shown as labels on the lines.
- The corresponding link state packets for all six routers are shown in Fig. 5- 12(b).
- Building the link state packets is easy. The hard part is determining when to build them.
- One possibility is to build them periodically, that is, at regular intervals.
- Another possibility is to build them when some significant event occurs, such as a line or neighbor going down or coming back up again or changing its properties appreciably.

Figure 5-12. (a) A network. (b) The link state packets for this network.



(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)

Distributing the Link State Packets

The trickiest part of the algorithm is distributing the link state packets.

All of the routers must get all of the link state packets quickly and reliably. If different routers are using different versions of the topology, the routes they compute can have inconsistencies such as loops, unreachable machines, and other problems.

The fundamental idea is to use flooding to distribute the link state packets to all routers.

To keep the flood in check, each packet contains a sequence number that is incremented for each new packet sent.

Routers keep track of all the (source router, sequence) pairs they see.

When a new link state packet comes in, it is checked against the list of packets already seen.

If it is new, it is forwarded on all lines except the one it arrived on. If it is a duplicate, it is discarded.

If a packet with a sequence number lower than the highest one seen so far ever arrives, it is rejected as being obsolete as the router has more recent data.

This algorithm has a few problems, but they are manageable.

1.If the sequence numbers wrap around, confusion will reign.
The solution here is to use a 32-bit sequence number.
With one link state packet per second, it would take 137 years to wrap around,so this possibility can be ignored.

2.If a router ever crashes, it will lose track of its sequence number.
If it starts again at 0, the next packet it sends will be rejected as a duplicate.

3. If a sequence number is ever corrupted and 65,540 is received instead of 4 (a 1-bit error), packets 5 through 65,540 will be rejected as obsolete, since the current sequence number will be thought to be 65,540.

- The solution to all these problems is to include the age of each packet after the sequence number and decrement it once per second.
- When the age hits zero, the information from that router is discarded.
- Normally, a new packet comes in, say, every 10 sec, so router information only times out when a router is down (or six consecutive packets have been lost, an unlikely event).
- The *Age* field is also decremented by each router during the initial flooding process, to make sure no packet can get lost and live for an indefinite period of time (a packet whose age is zero is discarded).

Some refinements to this algorithm make it more robust. When a link state

packet comes in to a router for flooding, it is not queued for transmission immediately. Instead, it is put in a holding area to wait a short while in case more

links are coming up or going down. If another link state packet from the same

source comes in before the first packet is transmitted, their sequence numbers are compared. If they are equal, the duplicate is discarded. If they are different, the

older one is thrown out. To guard against errors on the links, all link state packets

are acknowledged.

Some refinements to this algorithm make it more robust.

When a link state packet comes in to a router for flooding, it is not queued for transmission immediately.

Instead, it is put in a holding area to wait a short while in case more links are coming up or going down.

If another link state packet from the same source comes in before the first packet is transmitted, their sequence numbers are compared.

If they are equal, the duplicate is discarded.

If they are different, the older one is thrown out.

To guard against errors on the links, all link state packets are acknowledged.

The data structure used by router *B* for the network shown in Fig. 5-12(a) is depicted in Fig. 5-13.

Each row here corresponds to a recently arrived, but as yet not fully processed, link state packet.

The table records where the packet originated, its sequence number and age, and the data.

In addition, there are send and acknowledgement flags for each of *B*'s three links (to *A*, *C*, and *F*, respectively).

The send flags mean that the packet must be sent on the indicated link.

The acknowledgement flags mean that it must be acknowledged there.

Figure 5-13. The packet buffer for router *B* in Fig. 5-12(a).

Source	Seq.	Age	Send flags A	Send flags C	Send flags F	ACK flags A	ACK flags C	ACK flags F	Data
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	

In Fig. 5-13, 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*.

However, the situation with the third packet, from *E*, is different.

It arrives twice, once via *EAB* and once via *EFB*. Consequently, it has to be sent only to *C* but must be acknowledged to both *A* and *F*, as indicated by the bits.

If a duplicate arrives while the original is still in the buffer, bits have to be changed.

For example, if a copy of *C*'s state arrives from *F* before the fourth entry in the table has been forwarded, the six bits will be changed to 100011 to indicate that the packet must be acknowledged to *F* but not sent there.

Computing the New Routes

Once a router has accumulated a full set of link state packets, it can construct the entire network graph because every link is represented.

Every link is, in fact, represented twice, once for each direction.

The different directions may even have different costs.

The shortest-path computations may then find different paths from router *A* to *B* than from router *B* to *A*.

Now Dijkstra's algorithm can be run locally to construct the shortest paths to all possible destinations.

The results of this algorithm tell the router which link to use to reach each destination.

This information is installed in the routing tables, and normal operation is resumed.

Compared to distance vector routing, link state routing requires more memory and computation.

For a network with *n* routers, each of which has *k* neighbors,

the memory required to store the input data is proportional to *kn*, which is at least as large as a routing table listing all the destinations.

Also, the computation time grows faster than *kn*, even with the most efficient data structures, an issue in large networks.

Nevertheless, in many practical situations, link state routing works

well because it does not suffer from slow convergence problems.

Link state routing is widely used in actual networks. Many ISPs use the **IS-IS** (**Intermediate System-Intermediate System**) link state protocol (Oran, 1990).

It was designed for an early network called DECnet, later adopted by ISO for use with the OSI protocols and then modified to handle other protocols as well, most notably, IP.

OSPF (**Open Shortest Path First**) is the other main link state protocol.

- It was designed by IETF several years after IS-IS and adopted many of the innovations designed for IS-IS.
- These innovations include a self-stabilizing method of flooding link state updates, the concept of a designated router on a LAN, and the method of computing and supporting path splitting and multiple metrics.
- As a consequence, there is very little difference between IS-IS and OSPF.
- The most important difference is that IS-IS can carry information about multiple network layer protocols at the same time (e.g., IP, IPX, and AppleTalk).
- OSPF does not have this feature, and it is an advantage in large multiprotocol environments.

5.2.6 Hierarchical Routing

- As networks grow in size, the router routing tables grow proportionally.
- Not only is router memory consumed by ever-increasing tables, but more CPU time is needed to scan them and more bandwidth is needed to send status reports about them.
- At a certain point, the network may grow to the point where it is no longer feasible for every router to have an entry for every other router, so the routing will have to be done hierarchically, as it is in the telephone network.
- When hierarchical routing is used, the routers are divided into **regions**.
- Each router knows all the details about how to route packets to destinations within its own region but knows nothing about the internal structure of other regions.
- When different networks are interconnected, it is natural to regard each one as a separate region to free the routers in one network from having to know the topological structure of the other ones.
- 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.
- As an example of a multilevel hierarchy, consider how a packet might be routed from Berkeley, California, to Malindi, Kenya.
- The Berkeley router would know the detailed topology within California but would send all out-of-state traffic to the Los Angeles router.

The Los Angeles router would be able to route traffic directly to other domestic routers but would send all foreign traffic to New York.

The New York router would be programmed to direct all traffic to the router in the destination country responsible for handling foreign traffic, say, in Nairobi.

Finally, the packet would work its way down the tree in Kenya until it got to Malindi.

Figure 5-14 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. 5-14(b).

When routing is done hierarchically, as in Fig. 5-14(c), there are entries for all the local routers, as before, but all other regions are 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.

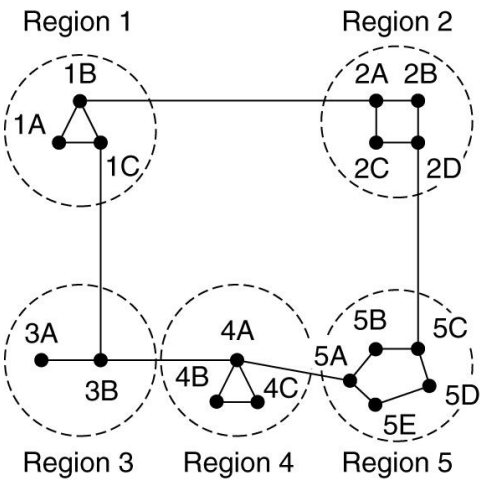
As the ratio of the number of regions to the number of routers per region grows, the savings in table space increase.

Unfortunately, these gains in space are not free.

There is a penalty to be paid:

increased path length.

For example, the best route from *1A* to *5C* is via region 2, but with hierarchical routing all traffic to region 5 goes via region 3, because that is better for most destinations in region 5.



(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 5-14. Hierarchical routing.

When a single network becomes very large, an interesting question is "how many levels should the hierarchy have?"

For example, consider a network with 720 routers.

If there is no hierarchy, each router needs 720 routing table entries.

If the network is partitioned into 24 regions of 30 routers each, each router needs 30 local entries plus 23 remote entries for a total of 53 entries.

If a three-level hierarchy is chosen, with 8 clusters each containing 9 regions of 10 routers, each router needs 10 entries for local routers, 8 entries for routing to other regions within its own cluster, and 7 entries for distant clusters, for a total of 25 entries.

Kamoun and Kleinrock (1979) discovered that the optimal number of levels for an N router network is ln N, requiring a total of e ln N entries per router.

They have also shown that the increase in effective mean path length caused by hierarchical routing is sufficiently small that it is usually acceptable.

5.2.7 Broadcast Routing

- In some applications, hosts need to send messages to many or all other hosts.
- For example, a service distributing weather reports, stock market updates, or live radio programs might work best by sending to all machines and letting those that are interested read the data.
- Sending a packet to all destinations simultaneously is called **broadcasting**.
- Various methods have been proposed for doing it.
- One broadcasting method that requires no special features from the network is for the source to simply send a distinct packet to each destination.
- Not only is the method wasteful of bandwidth and slow, but it also requires the source to have a complete list of all destinations.
- This method is not desirable in practice, even though it is widely applicable.
- An improvement is **multidestination routing**, in which each packet contains either a list of destinations or a bit map indicating the desired destinations.
- When a packet arrives at a router, the router checks all the destinations to determine the set of output lines that will be needed.
- An output line is needed if it is the best route to at least one of the destinations. The router generates a new copy of the packet for each output line to be used and includes in each packet only those destinations that are to use the line.
- In effect, the destination set is partitioned among the output lines.
- After a sufficient number of hops, each packet will carry only one destination like a normal packet.

- Multidestination routing is like using separately addressed packets, except that when several packets must follow the same route, one of them pays full fare and the rest ride free.
- The network bandwidth is therefore used more efficiently.
- However, this scheme still requires the source to know all the destinations, plus it is as much work for a router to determine where to send one multidestination packet as it is for multiple distinct packets.
- We have already seen a better broadcast routing technique: flooding.
- When implemented with a sequence number per source, flooding uses links efficiently with a decision rule at routers that is relatively simple.
- Although flooding is illsuited for ordinary point-to-point communication, it rates serious consideration for broadcasting.
- However, it turns out that we can do better still once the shortest path routes for regular packets have been computed.
- The idea for **reverse path forwarding** is elegant and remarkably simple once it has been pointed out (Dalal and Metcalfe, 1978).
- When a broadcast packet arrives at a router, the router checks to see if the packet arrived on the link that is normally used for sending packets *toward* 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 links except the one it arrived on.
- If, however, the broadcast packet arrived on a link other than the preferred one for reaching the source, the packet is discarded as a likely duplicate.

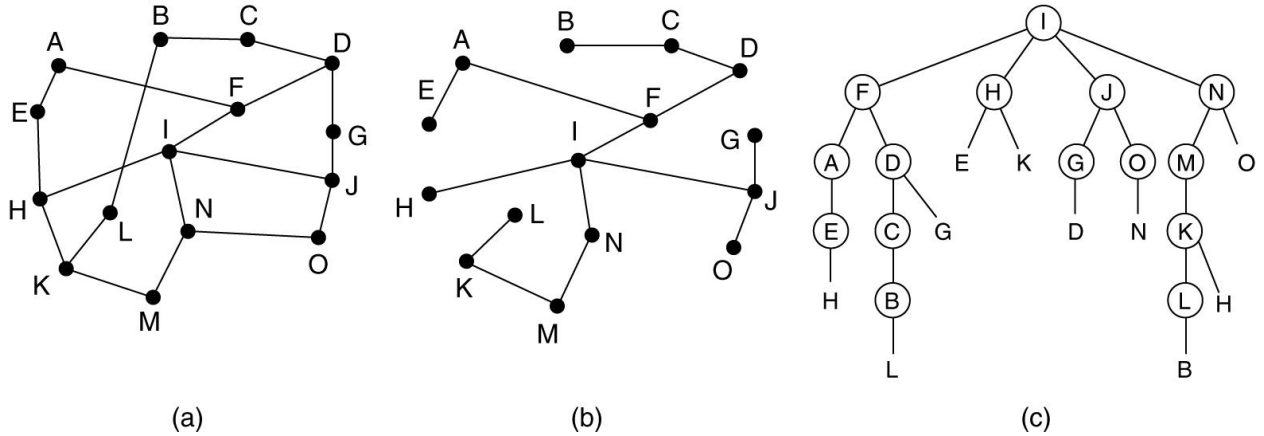


Figure 5-15. Reverse path forwarding. (a) A network. (b) A sink tree. (c) The tree built by reverse path forwarding.

- On the first hop, *I* sends packets to *F*, *H*, *J*, and *N*, as indicated by the second row of the tree.
- Each of these packets arrives on the preferred path to *I* (assuming that the preferred path falls along the sink tree) and is so indicated by a circle around the letter.
- On the second hop, eight packets are generated, two by each of the routers that received a packet on the first hop.
- As it turns out, all eight of these arrive at previously unvisited routers, and five of these arrive along the preferred line.
- Of the six packets generated on the third hop, only three arrive on the preferred path (at *C*, *E*, and *K*); the others are duplicates.
- After five hops and 24 packets, the broadcasting terminates, compared with four hops and 14 packets had the sink tree been followed exactly.

The principal advantage of reverse path forwarding is that it is efficient while being easy to implement.

It sends the broadcast packet over each link only once in each direction, just as in flooding, yet it requires only that routers know how to reach all destinations, without needing to remember sequence numbers (or use other mechanisms to stop the flood) or list all destinations in the packet.

Our last broadcast algorithm improves on the behavior of reverse path forwarding.

It makes explicit use of the sink tree—or any other convenient spanning

tree—for the router initiating the broadcast.

A **spanning tree** is a subset of the network that includes all the routers but contains no loops.

Sink trees are spanning trees.

If each router knows which of its lines belong to the spanning tree, it can copy an incoming broadcast packet onto all the spanning tree lines except the one it arrived on.

This method makes excellent use of bandwidth, generating the absolute minimum number of packets necessary to do the job.

In Fig. 5-15, for example, when the sink tree of part (b) is used as the spanning tree, the broadcast packet is sent with the minimum 14 packets.

The only problem is that each router must have knowledge of some spanning tree for the method to be applicable.

Sometimes this information is available (e.g., with link state routing, all routers know the complete topology, so they can compute a spanning tree) but sometimes it is not (e.g., with distance vector routing).

5.2.8 Multicast Routing

A multiplayer game or live video of a sports event streamed to many viewing locations, send packets to multiple receivers.

Unless the group is very small, sending a distinct packet to each receiver is expensive.

On the other hand, broadcasting a packet is wasteful if the group consists of, say,

1000 machines on a million-node network, so that most receivers are not interested in the message (or worse yet, they are definitely interested but are not supposed to see it).

Thus, we need a way to send messages to well-defined groups that are numerically large in size but small compared to the network as a whole.

Sending a message to such a group is called **multicasting**, and the routing algorithm used is called **multicast routing**.

All multicasting schemes require some way to create and destroy groups and to identify which routers are members of a group.

How these tasks are accomplished is not of concern to the routing algorithm.

For now, we will assume that each group is identified by a multicast address and that routers know the groups to which they belong.

We will revisit group membership when we describe the network layer of the Internet.

Multicast routing schemes build on the broadcast routing schemes.

we have already studied, sending packets along spanning trees to deliver the packets to the members of the group while making efficient use of bandwidth.

However, the best spanning tree to use depends on whether the group is dense, with receivers scattered over most of the network, or sparse, with much of the network not belonging to the group.

If the group is dense, broadcast is a good start because it efficiently gets the packet to all parts of the network.

But broadcast will reach some routers that are not members of the group, which is wasteful.

The solution explored by Deering and Cheriton (1990) is to prune the broadcast spanning tree by removing links that do not lead to members.

The result is an efficient multicast spanning tree.

As an example, consider the two groups, 1 and 2, in the network shown in

Fig. 5-16(a).

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. 5-16(b).

This tree can be used for broadcast but is overkill for multicast, as can be seen from the two pruned versions that are shown next.

InFig. 5-16(c), all the links that do not lead to hosts that are members of group 1 have been removed.

The result is the multicast spanning tree for the leftmost router to send to group 1.

Packets are forwarded only along this spanning tree, which is more efficient than the broadcast tree because there are 7 links instead of 10.

Fig. 5-16(d) shows the multicast spanning tree after pruning for group 2.

It is efficient too, with only five links this time.

It also shows that different multicast groups have different spanning trees.

Various ways of pruning the spanning tree are possible.

The simplest one can be used if link state routing is used and each router is aware of the complete topology, including which hosts belong to which groups.

Each router can then construct its own pruned spanning tree for each sender to the group in question by constructing a sink tree for the sender as usual and then removing all links that do

not connect group members to the sink node.

MOSPF (**Multicast OSPF**) is an example of a link state protocol that works in this way (Moy, 1994).

With distance vector routing, a different pruning strategy can be followed.

The basic algorithm is reverse path forwarding.

However, whenever a router with no hosts interested in a particular group and no connections to other routers receives a multicast message for that group, it responds with a PRUNE message, telling the neighbor that sent the message not to send it any more multicasts from the sender for that group.

When a router with no group members among its own hosts has received such messages on all the lines to which it sends the multicast, it, too, can respond with a PRUNE message.

In this way, the spanning tree is recursively pruned. **DVMRP** (**Distance Vector Multicast Routing Protocol**) is an example of a multicast routing protocol that works this way (Waitzman et al., 1988).

Pruning results in efficient spanning trees that use only the links that are actually needed to reach members of the group.

One potential disadvantage is that it is lots of work for routers, especially for large networks. Suppose that a network has *n* groups, each with an average of *m* nodes.

At each router and for each group, *m* pruned spanning trees must be stored, for a total of *mn* trees. For example, Fig. 5-16(c) gives the spanning tree for the leftmost router to send to group 1.

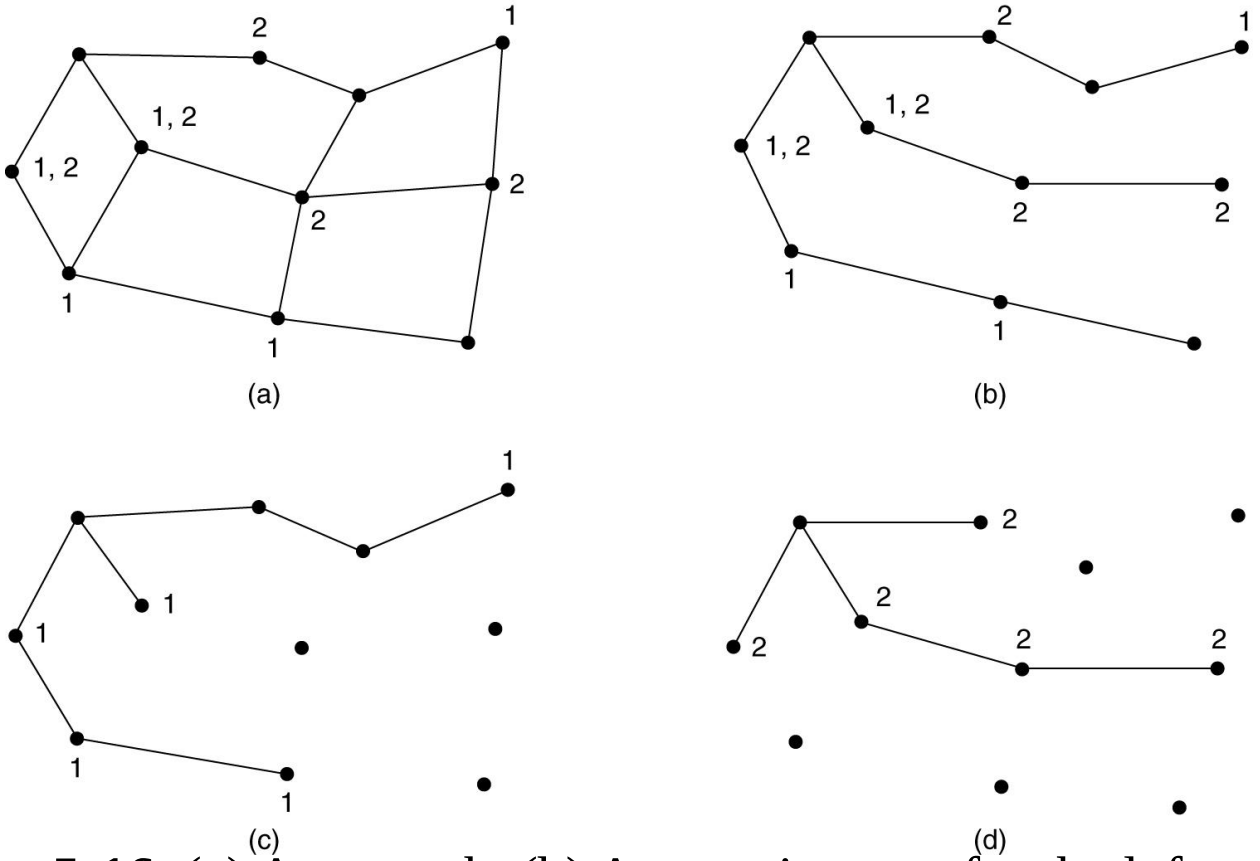


Figure 5-16. (a) A network. (b) A spanning tree for the leftmost router. (c) A multicast tree for group 1. (d) A multicast tree for group 2.

The spanning tree for the rightmost router to send to group 1 (not shown) will look quite different, as packets will head directly for group members rather than via the left side of the graph.

This in turn means that routers must forward packets destined to group 1 in different directions depending on which node is sending to the group.

When many large groups with many senders exist, considerable storage is needed to store all the trees.

An alternative design uses core-based trees to compute a single spanning tree for the group (Ballardie et al., 1993).

All of the routers agree on a root (called the core or rendezvous point) and build the tree by sending a packet from each member to the root.

The tree is the union of the paths traced by these packets.

Fig. 5-17(a) shows a core-based tree for group 1.

To send to this group, a sender sends a packet to the core.

When the packet reaches the core, it is forwarded down the tree.

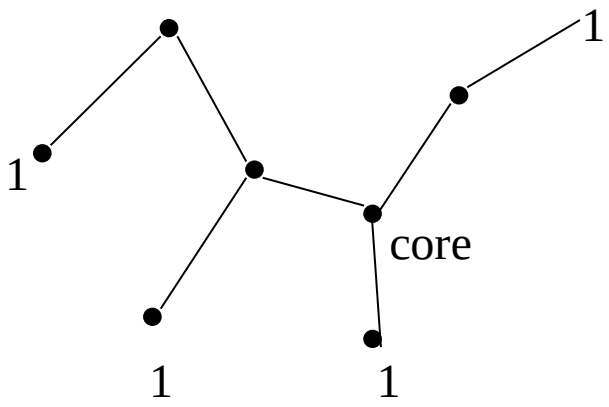
This is shown in Fig. 5-17(b) for the sender on the righthand side of the network.

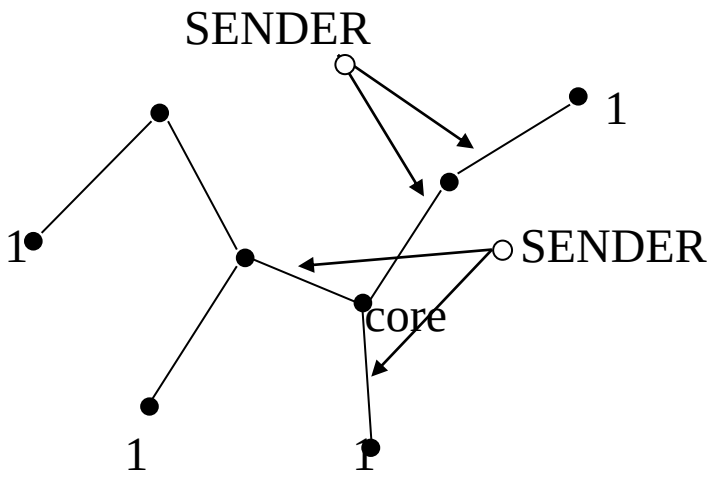
As a performance optimization, packets destined for the group do not need to reach the core before they are multicast.

As soon as a packet reaches the tree, it can be forwarded up toward the root, as well as down all the other

branches.

This is the case for the sender at the top of Fig. 5-17(b).


1
1
core
1
1


SENDER
1
1
SENDER
core
1
1

Having a shared tree is not optimal for all sources.

For example, in Fig. 5- 17(b), the packet from the sender on the righthand side reaches the top-right group member via the core in three hops, instead of directly.

The inefficiency depends on where the core and senders are located, but often it is reasonable when the core is in the middle of the senders.

When there is only a single sender, as in a video that is streamed to a group, using the sender as the core is optimal.

Also of note is that shared trees can be a major savings in storage costs, messages sent, and computation.

Each router has to keep only one tree per group, instead of *m* trees.

Further, routers that are not part of the tree do no work at all to support the group.

For this reason, shared tree approaches like core-based trees are used for multicasting to sparse groups in the Internet as part of popular protocols such as **PIM** (**Protocol Independent Multicast**) (Fenner et al., 2006).

5.2.9 Anycast Routing

So far, we have covered delivery models in which a source sends to a single destination (called **unicast**), to all destinations (called broadcast), and to a group of destinations (called multicast).

Another delivery model, called **anycast** is sometimes also useful.

In anycast, a packet is delivered to the nearest member of a group (Partridge et al., 1993).

Schemes that find these paths are called **anycast routing**.

Why would we want anycast?

Sometimes nodes provide a service, such as time of day or content distribution for which it is getting the right information all that matters, not the node that is contacted; any node will do.

For example, anycast is used in the Internet as part of DNS?

Luckily, we will not have to devise new routing schemes for anycast because regular distance vector and link state routing can produce anycast routes.

Suppose we want to anycast to the members of group 1.

They will all be given the address "1," instead of different addresses.

Distance vector routing will distribute vectors as usual, and nodes will choose the shortest path to destination 1.

This will result in nodes sending to the nearest instance of destination 1.

The routes are shown in Fig. 5-18(a).

This procedure works because the routing protocol does not realize that there are multiple instances of destination 1.

That is, it believes that all the instances of node 1 are the same node, as in the topology shown in Fig. 5-18(b).

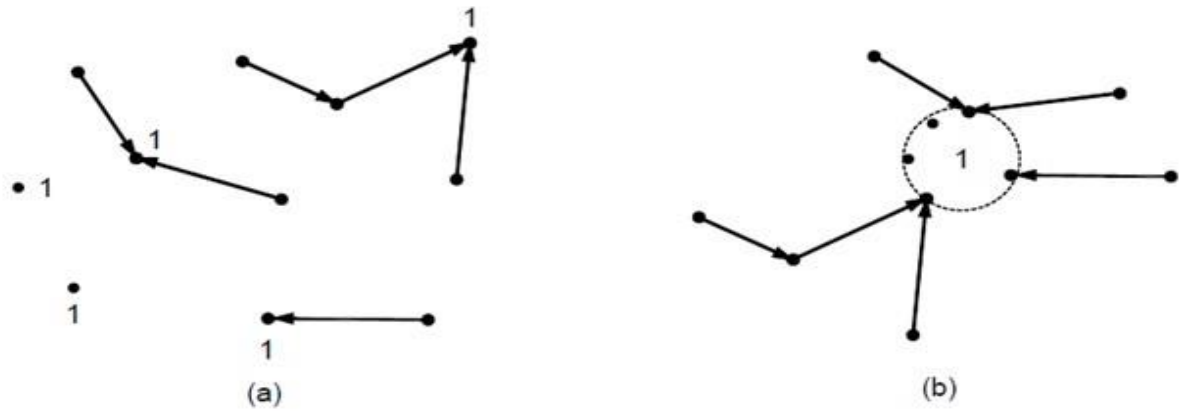


Figure 5-18. (a) Anycast routes to group 1. (b) Topology seen by the routing protocol.

This procedure works for link state routing as well, although there is the added consideration that the routing protocol must not find seemingly short paths that pass through node 1.

This would result in jumps through hyperspace, since the instances of node 1 are really nodes located in different parts of the network.

However, link state protocols already make this distinction between routers and hosts.

5.2.10 Routing for Mobile Hosts

Millions of people use computers while on the go, from truly mobile situations

with wireless devices in moving cars, to nomadic situations in which laptop

computers are used in a series of different locations.

We will use the term **mobile hosts** to mean either category, as distinct from stationary hosts that never move.

Increasingly, people want to stay connected wherever in the world they may be, as

easily as if they were at home.

These mobile hosts introduce a new complication: to route a packet to a mobile host, the network first has to find it.

The model of the world that we will consider is one in which all hosts are assumed

to have a permanent **home location** that never changes.

Each hosts also has a permanent home address that can be used to determine its home location, analogous to the way the telephone number 1-212-5551212 indicates the United States (country code 1) and Manhattan (212).

The routing goal in systems with mobile hosts is to make it possible to send packets to mobile hosts using their fixed home addresses and have the packets efficiently reach them wherever they may be.

The trick, of course, is to find them.

A different model would be to recompute routes as the mobile host moves and the topology changes.

We could then simply use the routing schemes described earlier in this section.

However, with a growing number of mobile hosts, this model would soon lead to the entire network endlessly computing new routes.

Using the home addresses greatly reduces this burden.

Another alternative would be to provide mobility above the network layer, which is what typically happens with laptops today.

When they are moved to new Internet locations, laptops acquire new network addresses.

There is no association between the old and new addresses; the network does not know that they belonged to the same laptop.

In this model, a laptop can be used to browse the Web, but other hosts cannot send packets to it (for example, for an incoming call), without building a higher layer location service,

for example, signing into Skype *again* after moving.

Moreover, connections cannot be maintained while the host is moving; new connections must be started up instead.

Network-layer mobility is useful to fix these problems.

The basic idea used for mobile routing in the Internet and cellular networks is for the mobile host to tell a host at the home location where it is now.

This host, which acts on behalf of the mobile host, is called the **home agent**.

Once it knows where the mobile host is currently located, it can forward packets so that they are

delivered.

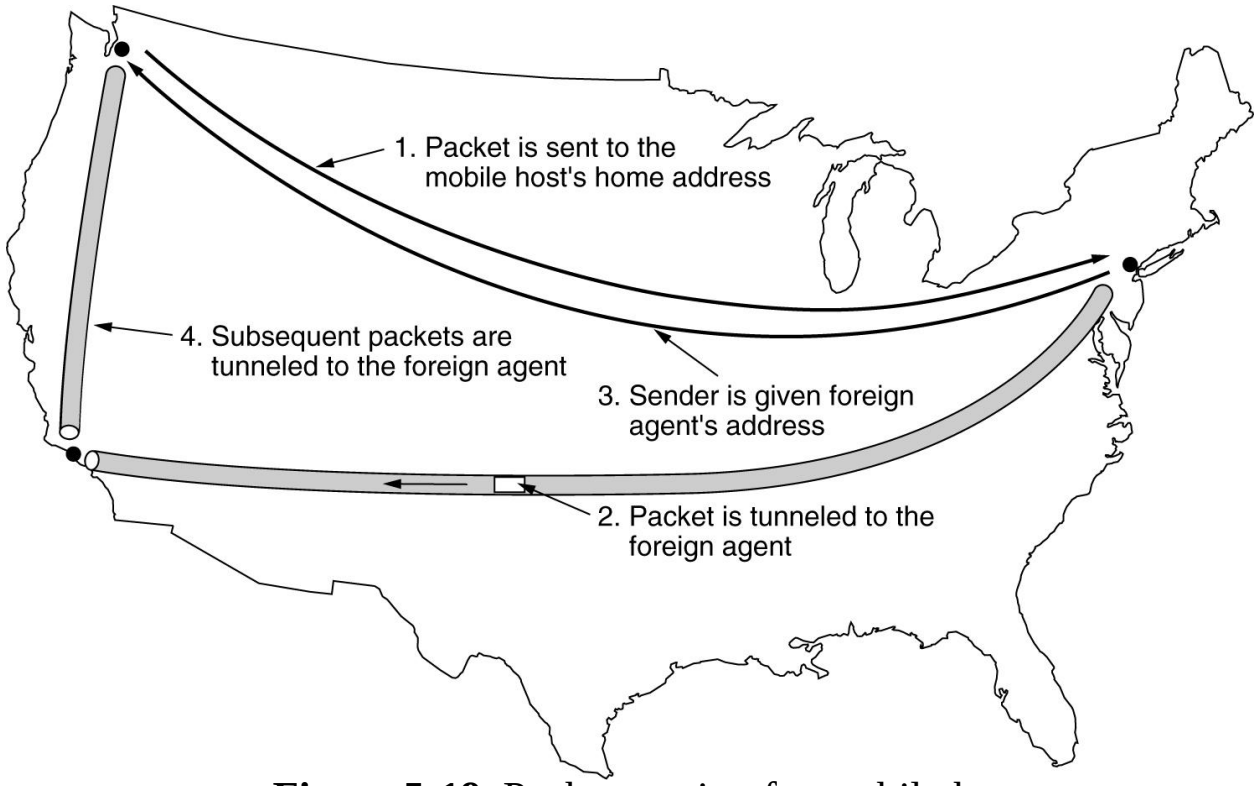


Figure 5-19. Packet routing for mobile hosts.

REFERENCES

- Andrew S. Tanenbaum, David J. Wetherall, “Computer Networks“, 5th Edition, Pearson Education Publ. - 2011
- Miller, ”Data and Network Communications”, Viaks Publ., 2001.
- William A Shay, “Understanding data communications and Networks”, 2nd Edition, Vikas Publ., 2001.
- http://my.fit.edu/~vkepuska/ece4561/0132127067_ppt-125189/Chapter5-NetworkLayer.ppt