# ROUTING ALGORITHMS

## ROUTING ALGORITHMS :

The main function of the network layer is routing packets from the source machine to the destination machine. In most subnets, packets will require multiple hops to make the journey. The only notable exception is for broadcast networks, but even here routing is an issue if the source and destination are not on the same network. The algorithms that choose the routes and the data structures that they use are a major area of network layer design.

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

# ROUTING ALGORITHMS

If the subnet uses datagrams internally, this decision must be made new for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new virtual circuit is being set up. Thereafter, data packets just follow the previously established route. The latter case is sometimes called session routing because a route remains in force for an entire user session.

One can think of 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. That is where the routing algorithm comes into play.

# ROUTING ALGORITHMS

Regardless of whether routes are chosen independently for each packet or only when new connections are established, certain properties are desirable in a routing algorithm: correctness, simplicity, robustness, stability, fairness, and optimality.

# ROUTING ALGORITHMS

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* (for all *I* and *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. Adaptive algorithms differ in where they get their information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every *T* sec, when the load changes or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).

# ROUTING ALGORITHMS

## 1. The Optimality Principle :

Before we get into specific algorithms, it may be helpful to note that one can make a general statement about optimal routes without regard to network topology or traffic. This statement is known as the optimality principle.

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* r1 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.

# ROUTING ALGORITHMS

## 3 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.

# ROUTING ALGORITHMS

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

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 is a duplicate; if so, it is discarded. Furthermore, the full list below *k* is not needed, since *k* effectively summarizes it.

# ROUTING ALGORITHMS

A variation of flooding that is slightly more practical is selective flooding. In this algorithm the routers do not send every incoming packet out on every line, only on those lines that are going approximately in the right direction.

Flooding is not practical in most applications, but it does have some uses. For example, in military applications, where large numbers of routers may be blown to bits at any instant, the tremendous robustness of flooding is highly desirable. In distributed database applications, it is sometimes necessary to update all the databases concurrently, in which case flooding can be useful. 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.

# 5 Link State Routing :

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. Two primary problems caused its demise. First, since the delay metric was queue length, it did not take line bandwidth into account when choosing routes. Initially, all the lines were 56 kbps, so line bandwidth was not an issue, but after some lines had been upgraded to 230 kbps and others to 1.544 Mbps, not taking bandwidth into account was a major problem. Of course, it would have been possible to change the delay metric to factor in line bandwidth, but a second problem also existed, namely, the algorithm often took too long to converge (the count-to-infinity problem). For these reasons, it was replaced by an entirely new algorithm, now called link state routing. Variants of link state routing are now widely used.

# ROUTING ALGORITHMS

The idea behind link state routing is simple and can be stated as five parts.

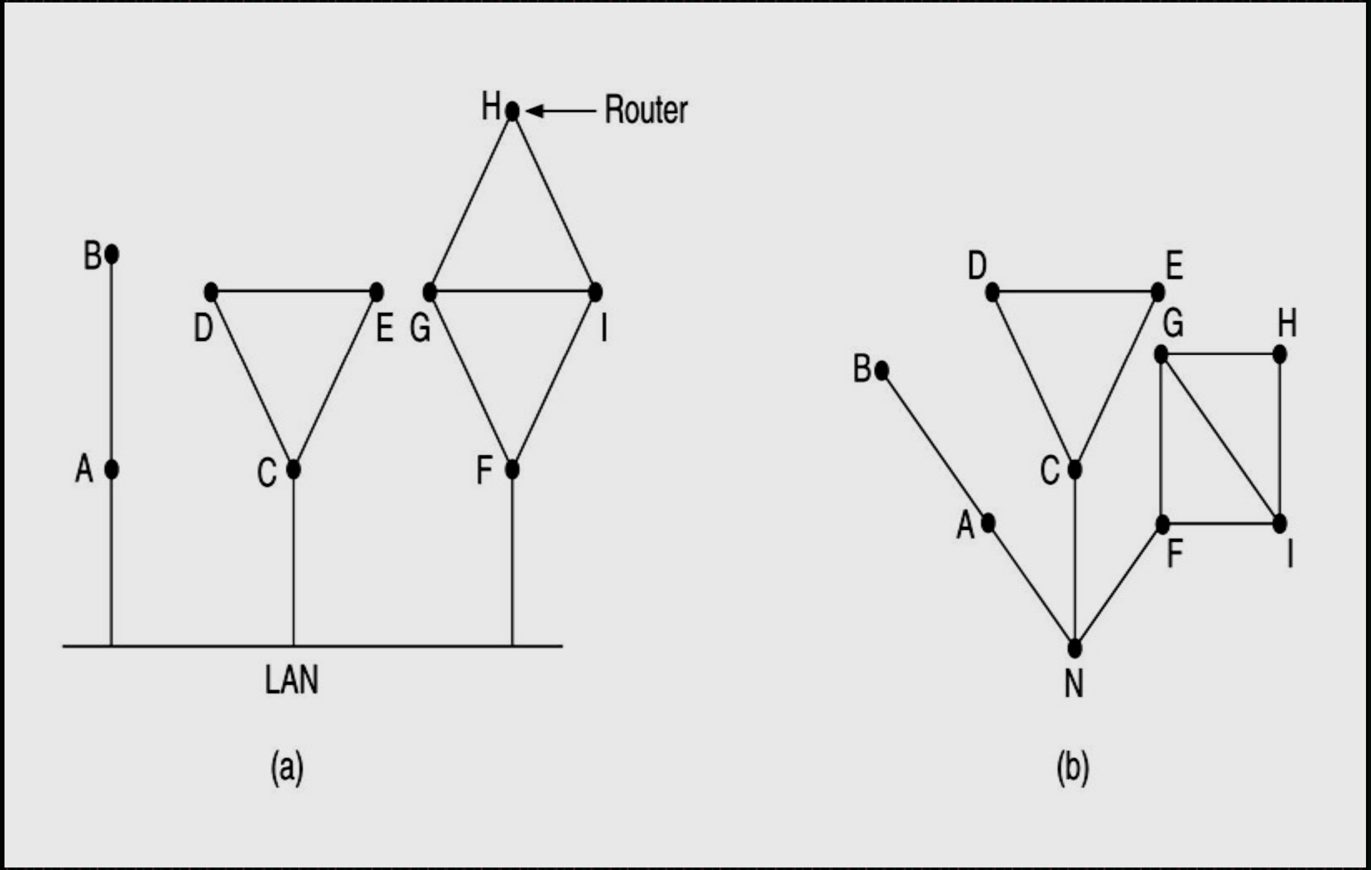
Each router must do the following:

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.

In effect, the complete topology and all delays are experimentally measured and distributed to every router. Then Dijkstra's algorithm can be run to find the shortest path to every other router.

## 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 telling who it is. 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*.



(a) Nine routers and a LAN (b) A graph model of (a)

# ROUTING ALGORITHMS

When two or more routers are connected by a LAN, the situation is slightly more complicated. Fig. (a) illustrates a 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.

One way to model the LAN is to consider it as a node itself, as shown in Fig. (b). Here we have introduced a new, artificial node, N, to which A, C, and F are connected. The fact that it is possible to go from A to C on the LAN is represented by the path ANC here.

## *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.**

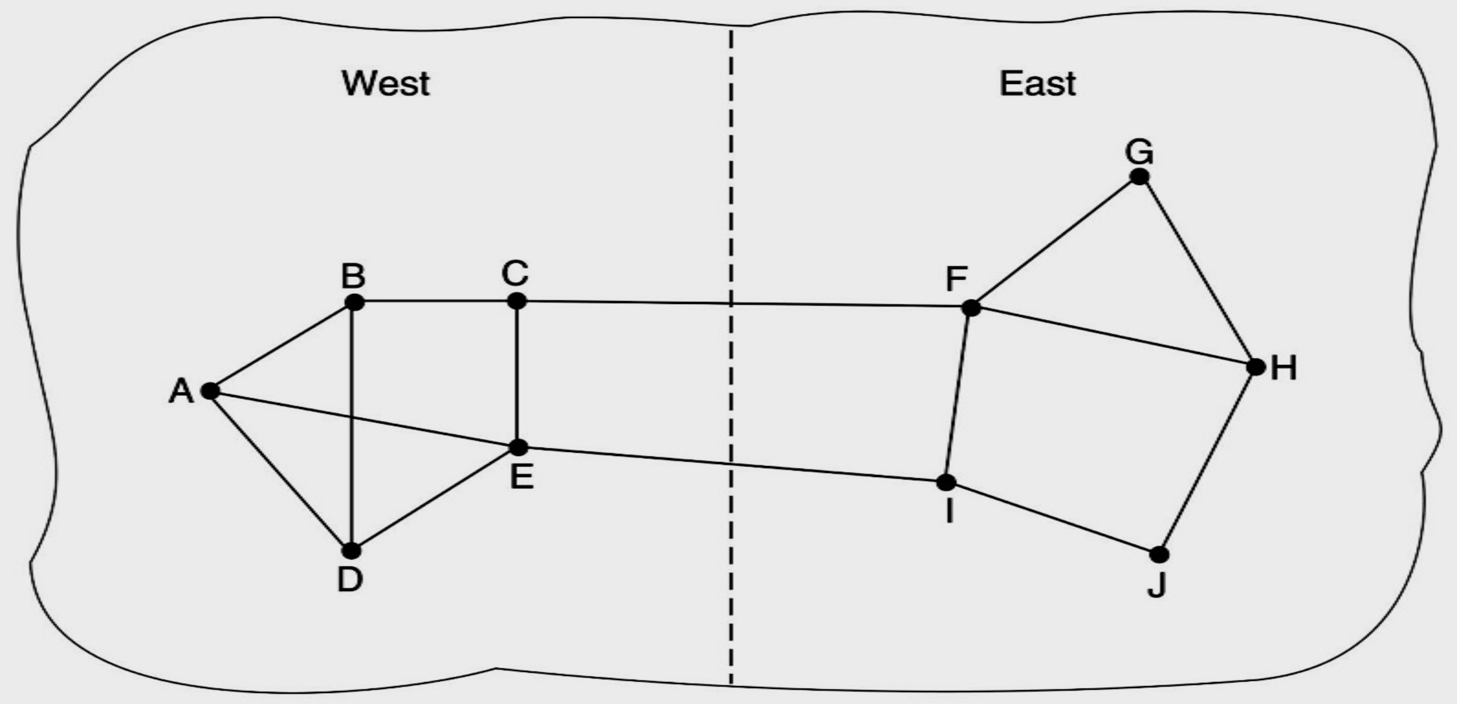
# ROUTING ALGORITHMS

An interesting issue is whether to take the load into account when measuring the delay. To factor the load in, the round-trip timer must be started when the ECHO packet is queued. To ignore the load, the timer should be started when the ECHO packet reaches the front of the queue.

Arguments can be made both ways. Including traffic-induced delays in the measurements means that when a router has a choice between two lines with the same bandwidth, one of which is heavily loaded all the time and one of which is not, the router will regard the route over the unloaded line as a shorter path. This choice will result in better performance.

# ROUTING ALGORITHMS

Unfortunately, there is also an argument against including the load in the delay calculation. Consider the subnet of Fig. A, which is divided into two parts, East and West, connected by two lines, *CF* and *EI.*



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

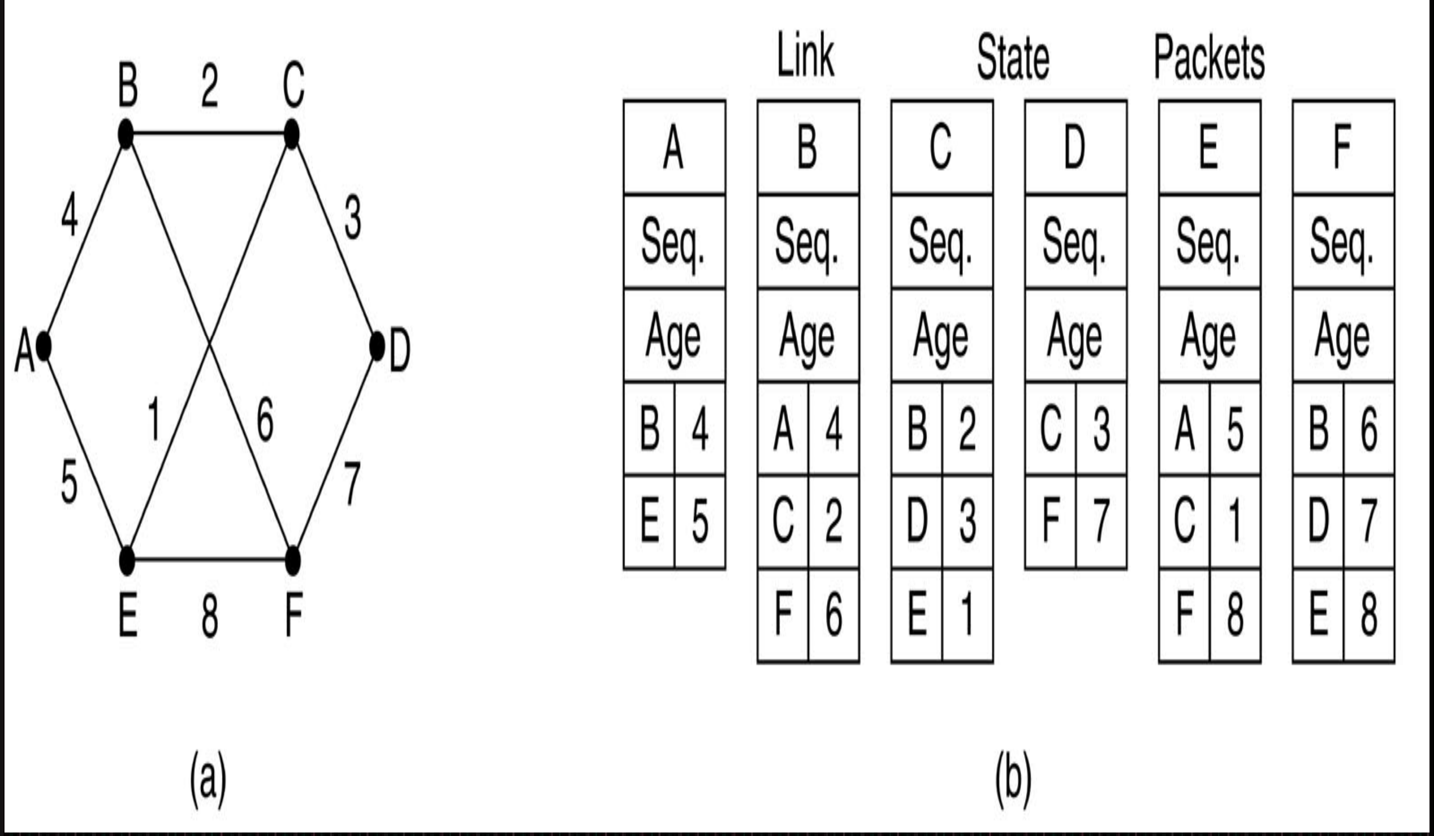
# ROUTING ALGORITHMS

Suppose that most of the traffic between East and West is using line *CF*, and as a result, this line is heavily loaded with long delays. Including queueing delay in the shortest path calculation will make *EI* more attractive. After the new routing tables have been installed, most of the East-West traffic will now go over *EI*, overloading this line.

Consequently, in the next update, *CF* will appear to be the shortest path. As a result, the routing tables may oscillate wildly, leading to erratic routing and many potential problems. If load is ignored and only bandwidth is considered, this problem does not occur. Alternatively, the load can be spread over both lines, but this solution does not fully utilize the best path. Nevertheless, to avoid oscillations in the choice of best path, it may be wise to distribute the load over multiple lines, with some known fraction going over each line.

# 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 (to be described later), and a list of neighbors. For each neighbor, the delay to that neighbor is given. An example subnet is given in Fig. (a) with delays shown as labels on the lines. The corresponding link state packets for all six routers are shown in Fig. (b).



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 |

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

# ROUTING ALGORITHMS

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.

# Distributing the Link State Packets:

The trickiest part of the algorithm is distributing the link state packets reliably. As the packets are distributed and installed, the routers getting the first ones will change their routes. Consequently, the different routers may be using different versions of the topology, which can lead to inconsistencies, loops, unreachable machines, and other problems.

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

# ROUTING ALGORITHMS

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 since the router has more recent data.

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

# ROUTING ALGORITHMS

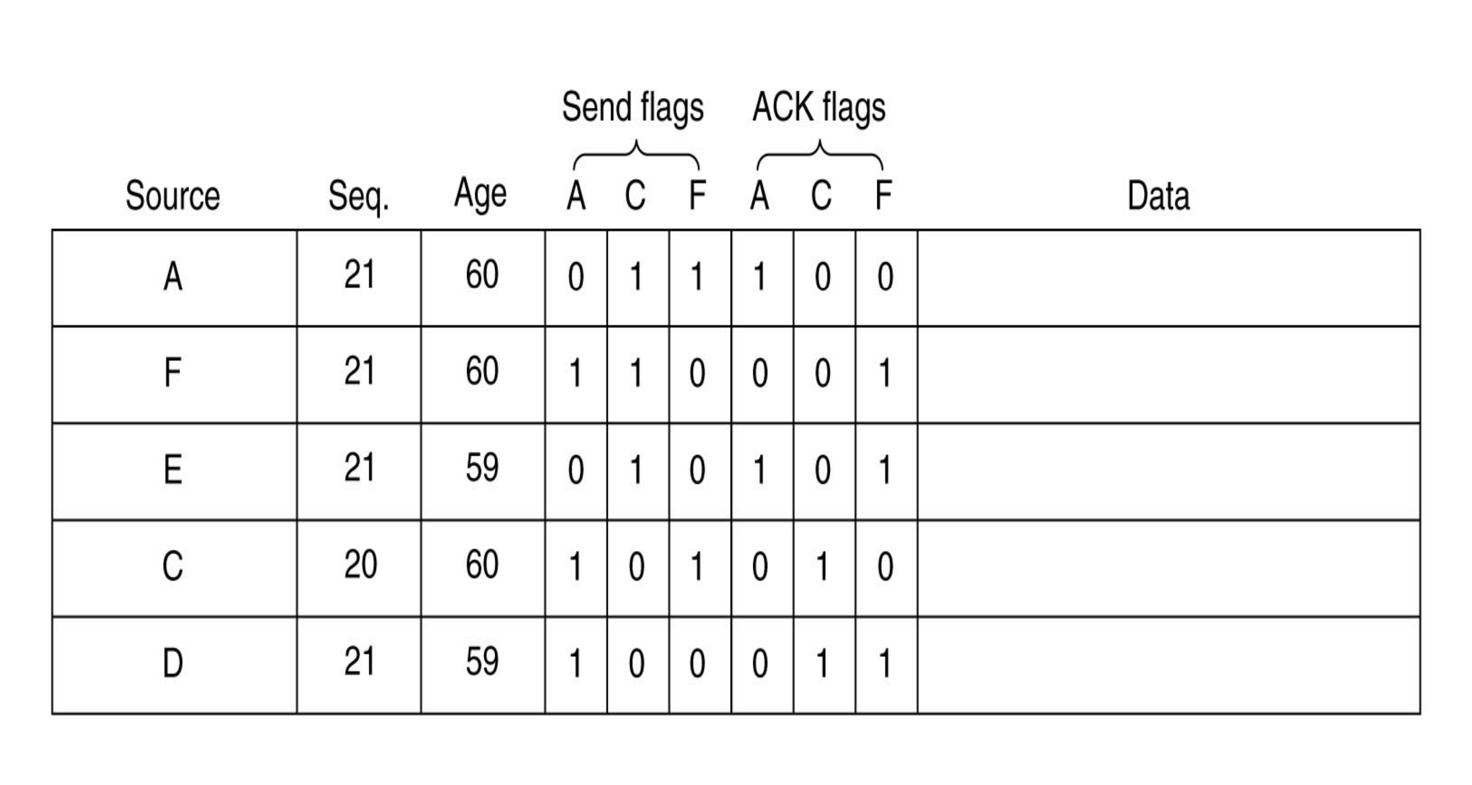
Second, if a router ever crashes, it will lose track of its sequence number. If it starts again at 0, the next packet will be rejected as a duplicate.

Third, if a sequence number is ever corrupted and 65,540 is received instead of 4 (a l-bit error), packets 5 through 65,540 will be rejected as obsolete, since the current sequence number is 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).

# ROUTING ALGORITHMS

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 first put in a holding area to wait a short while. 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 router-router lines, all link state packets are acknowledged. When a line goes idle, the holding area is scanned in round-robin order to select a packet or acknowledgement to send.



|  |  |  | Send flags |  |  | ACK flags |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| Source | Seq. | Age | A | C | F | A | C | 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 |  |

The packet buffer for router B

# ROUTING ALGORITHMS

The data structure used by router *B* for the subnet shown in Fig. (a) is depicted in this figure. 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 lines (to *A,* C, and *F,* respectively).

The send flags mean that the packet must be sent on the indicated line. The acknowledgement flags mean that it must be acknowledged there.

In this figure,, 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.*

# ROUTING ALGORITHMS

However, the situation with the third packet, from *E,* is different. It arrived twice, once via *EAB* and once via *EFB.* Consequently, it has to be sent only to C but 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 subnet graph because every link is represented. Every link is, in fact, represented twice, once for each direction. The two values can be averaged or used separately.

Now Dijkstra's algorithm can be run locally to construct the shortest path to all possible destinations. The results of this algorithm can be installed in the routing tables, and normal operation resumed.

For a subnet with *n* routers, each of which has *k* neighbors, the memory required to store the input data is proportional to *kn.* For large subnets, this can be a problem. Also, the computation time can be an issue. Nevertheless, in many practical situations, link state routing works well.

# ROUTING ALGORITHMS

However, problems with the hardware or software can cause havoc with this algorithm (also with other ones). For example, if a router claims to have a line it does not have or forgets a line it does have, the subnet graph will be incorrect. If a router fails to forward packets or corrupts them while forwarding them, trouble will arise.

Finally, if it runs out of memory or does. the routing calculation wrong, bad things will happen. As the subnet grows into the range of tens or hundreds of thousands of nodes, the probability of some router failing occasionally becomes nonnegligible. The trick is to try to arrange to limit the damage when the inevitable happens.

# ROUTING ALGORITHMS

Link state routing is widely used in actual networks, so a few words about some example protocols using it are in order. The OSPF protocol, which is widely used in the Internet, uses a link state algorithm.

Another link state protocol is IS-IS (Intermediate System-Intermediate System), which was designed for DECnet and later adopted by ISO for use with its connection less network layer protocol, CLNP. Since then it has been modified to handle other protocols as well, most notably, IP. IS-IS is used in some Internet backbones (including the old NSFNET backbone) and in some digital cellular systems such as CDPD. Novell NetWare uses a minor variant of IS-IS (NLSP) for routing IPX packets.

# ROUTING ALGORITHMS

Basically IS-IS distributes a picture of the router topology, from which the shortest paths are computed. Each router announces, in its link state information, which network layer addresses it can reach directly. These addresses can be IP, IPX, AppleTalk, or any other addresses. IS-IS can even support multiple network layer protocols at the same time.

Many of the innovations designed for IS-IS were adopted by OSPF (OSPF was designed several years after IS-IS). These 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 is encoded in such a way that it is easy and natural to simultaneously carry information about multiple network layer protocols, a feature OSPF does not have. This advantage is especially valuable in large multi protocol environments.