RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula Computer Science & Engineering

Computer Network [CS – 602]

Topic covered : Network Layer: Need, Services Provided , Design issues, Routing algorithms: Least Cost Routing algorithm, Dijkstra's algorithm, Bellman-ford algorithm, Hierarchical outing, Broadcast Routing, Multicast Routing. IP Addresses, Header format, Packet forwarding, Fragmentation and reassembly, ICMP, Comparative study of IPv4 & IPv6

NETWORK LAYER

The network layer is concerned with getting packets from the source all the way to the destination. Getting to the destination may require making many hops at intermediate routers along the way. This function clearly contrasts with that of the data link layer, which has the more modest goal of just moving frames from one end of a wire to the other. Thus, the network layer is the lowest layer that deals with end-to-end transmission.

Services provided by Network layer 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.

Given these goals, the designers of the network layer have a lot of freedom in writing detailed specifications of the services to be offered to the transport layer. This freedom often degenerates into a raging battle between two warring factions. the network layer should provide connection-oriented service or connectionless service. One camp (represented by the Internet community) argues that the routers' job is moving packets around and nothing else. 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 redecessors, if any.

The other camp (represented by the telephone companies) argues that the subnet should provide a reliable, connection-oriented service. They claim that 100 years of successful experience with the worldwide telephone system is an excellent guide. In this view, quality of service is the dominant factor, and without connections in the subnet, quality of service is very difficult to achieve, especially for real-time traffic such as voice and video.

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.

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 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.

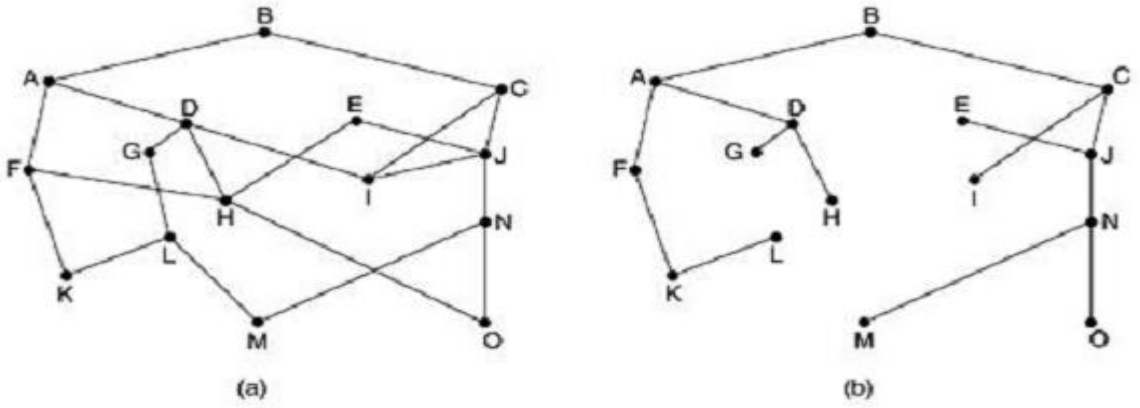
Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get their information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every ΔT 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). In the following sections we will discuss a variety of routing algorithms, both static and dynamic.

Principle of Optimality

One can make a general statement about optimal routes without regard to networktopology 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 Jr1 and the rest of the route r2. If a route better than r2 existed from J to K, it could be concatenated with r1 to improve the route from I to K, contradicting our statement that r1r2 is optimal.

As a direct consequence of the optimality principle, one 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 and is illustrated in Fig.6, 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.

Figure 6. (a) A subnet. (b) A sink tree for router B.



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. Links and routers can go down and come back up during operation, so different routers may have different ideas about the current topology. The optimality principle and the sink tree provide a benchmark against which other routing algorithms can be measured.

Static Routing Algorithm

Example of Static Algorithm are :

1. Shortest Path Routing 2. Flooding 3. Flow Based Routing

Shortest Path Routing Algorithm.

There are many algorithms for computing shortest path between two nodes. One of them is Dijkstra Algorithm. The other one is Bellman-Ford Algorithm.

Dijkstra Algorithm :

It is used for computing the shortest path from the root node to every other node in the network. 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. Using this metric, the paths ABC and ABE in Fig.7 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).

Several algorithms for computing the shortest path between two nodes of a graph are known. This one is due to Dijkstra (1959). Each node is labeled (in parentheses) with its distance from the source node along the best known path. 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.7(a) , where the weights represent, for example, distance. We want to find the shortest path from A to D. Mark node A as permanent, indicated by a filled-in circle. Then 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, label it with the node from which the probe was made so that one can reconstruct the final path later.

Having examined each of the nodes adjacent to A, examineall the tentatively labeled nodes in the whole graph and make the one with the smallest label permanent, as shown in Fig.7(b). This becomes the new working node. 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, is is the shorter path, so the node is relabeled.

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.

Fig.7 shows the first five steps of the algorithm. To see why the algorithm works, consider Fig.7(c). At that point E is made 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 labeled. 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.

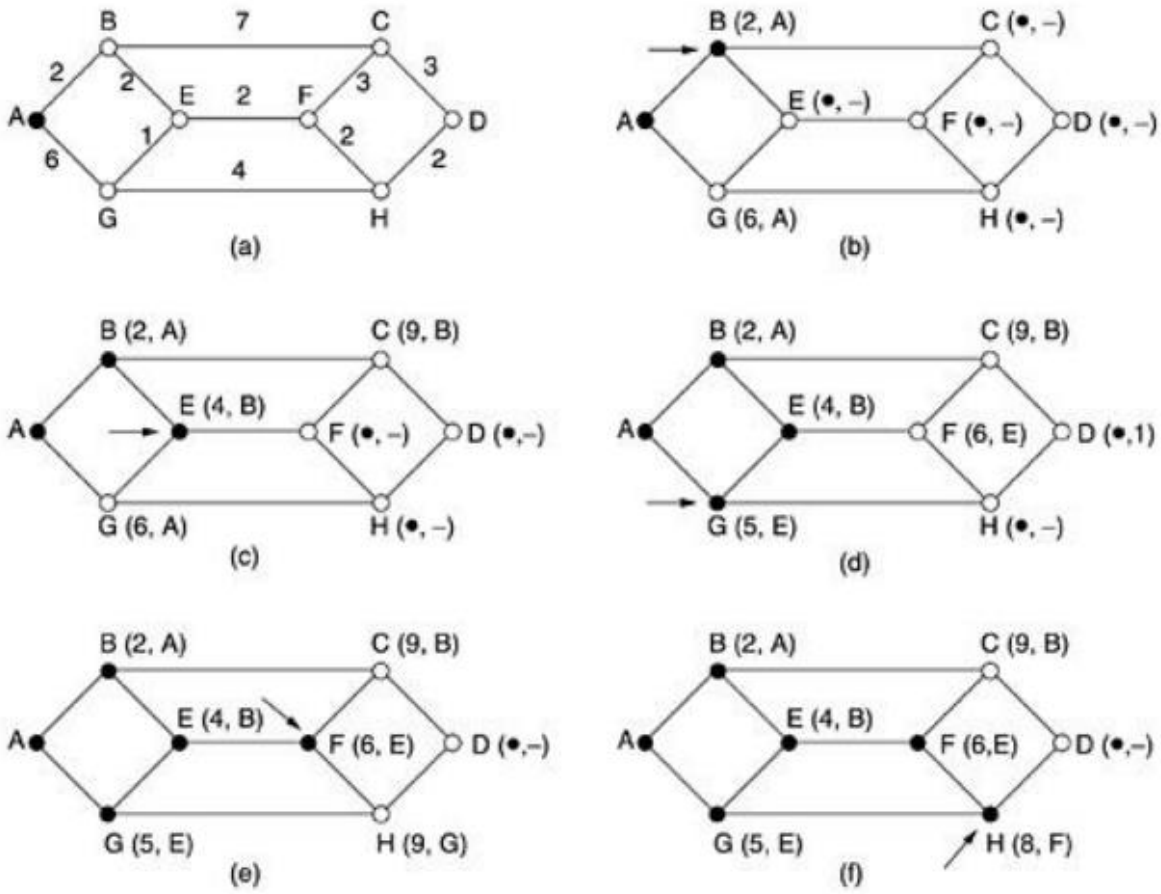


Fig.7 The first five steps used in computing the shortest path from A to D. The arrows indicate the working node.

Flooding Static Routing Algorithm

Flooding is a static routing algorithm, in which every incoming packet is sent out on every outgoing line except the one it arrived on. Flooding obviously generates vast numbers ofduplicate 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.

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. 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. 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. There is usually little point in sending a westbound packet on an eastbound line unless the topology is extremely peculiar and the router is sure of this fact.

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. A fourth possible use of flooding is 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.

Distance Vector Routing Algorithm

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;** 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 subnet. This entry contains two parts: the preferred outgoing line to use for that destination and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

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 queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets that the receiver just timestamps and sends back as fast as it can.

As an 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 line in its new routing table. Note that the old routing table is not used in the calculation.

This updating process is illustrated in Fig.9 Part (a) shows a subnet. 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,

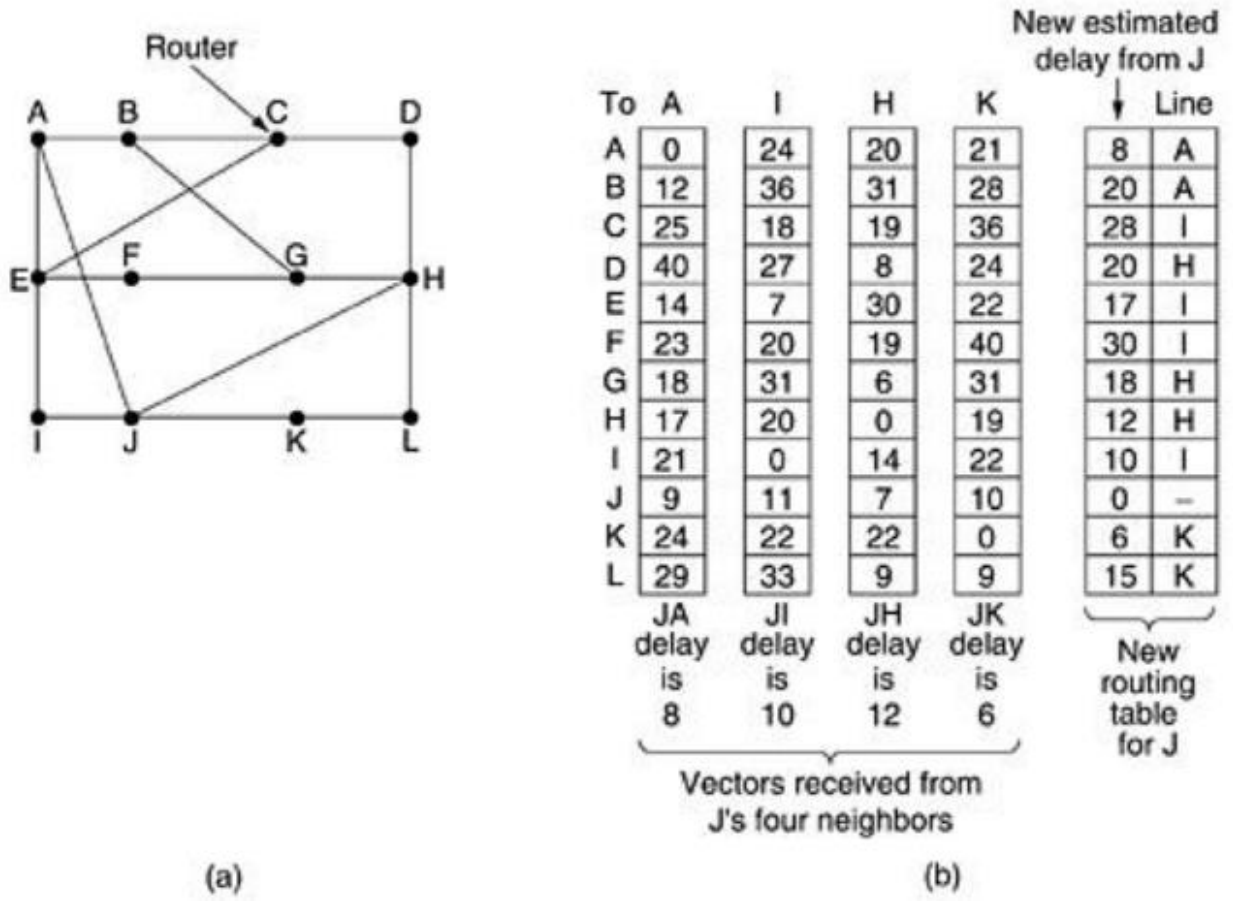


Fig.9 (a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.

Consider how J computes its new route to router G. It knows that it can get to A in 8 msec, an 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 it forwards packets bound for G to A. Similarly, it computes the delay to G via I, H, and K as 4 (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.

Count-to-Infinity Problem

Distance vector routing works in theory but has a serious drawback in practice. Although it converges to the correct answer, it may do so slowly. Consider a router whose best route todestination X is large. I 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. In one vector exchange, the good news is processed.

To see how fast good news propagates, consider the five-node (linear) subnet of Fig.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. When A comes up, the other routers learn about it via the vector exchanges. For simplicity assume that there is a gigantic going 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 neighbor has zero delay to A. B now makes an entry in its routing table 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.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 subnet whose longest path is of length N hops, within N exchanges everyone will know about newly-revived lines and routers.

Now consider the situation of Fig.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.

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 know that C's path runs through B itself. For all B knows, C might have ten lines 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. 10(b). Subsequent exchanges produce the history shown in the rest of Fig. 10(b). If the metric is time delay, there is no well-defined upper bound, so a high value is needed to prevent a path with a long delay from being treated as down. This problem is known as the **count-to-infinity problem.** There have been a few attempts to solve it (such as split horizon with poisoned reverse in RFC 1058), but none of these work well in general. 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.

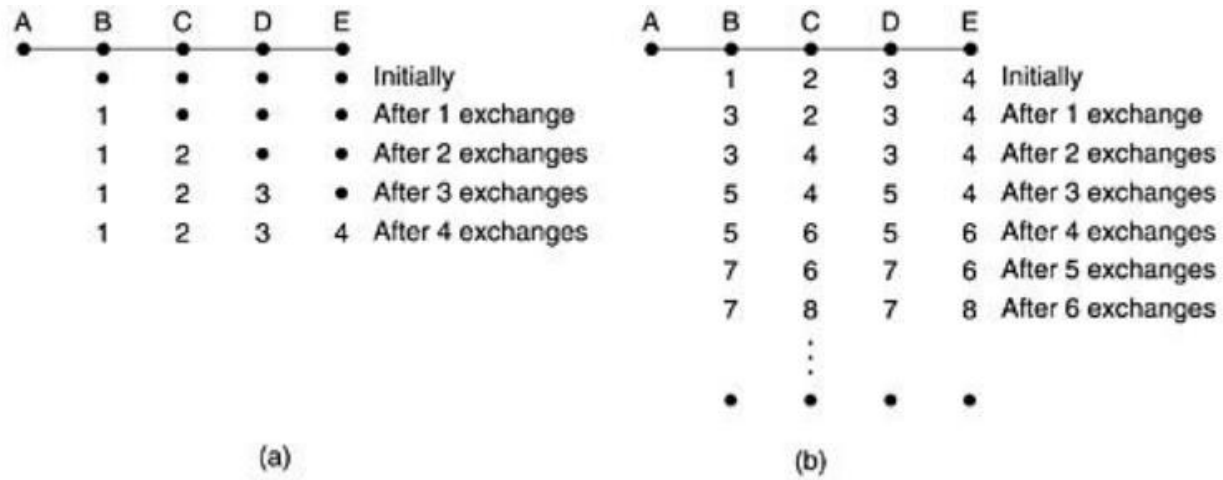


Fig.10: The count-to-infinity problem

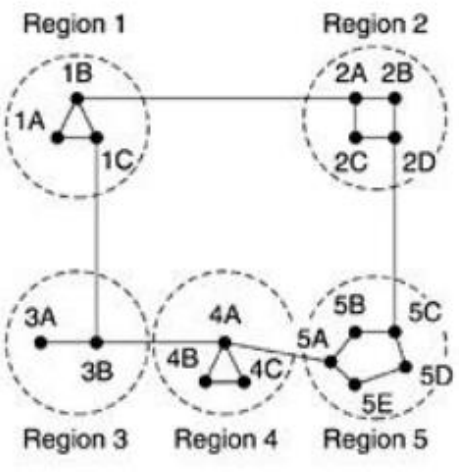
Hierarchical Routing

Hierarchical routing is an algorithm for routing packets hierarchically. 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, 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. When different networks are interconnected, it is natural to regard each one as a separate region in order 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. **Fig.11a** 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.11 (b).** When routing is done hierarchically, as in **Fig.11 (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.



(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)

Fig.11 Hierarchical Routing.

Unfortunately, these gains in space are not free. There is a penalty to be paid, and this penalty is in the form of 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.

When a single network becomes very large, an interesting question is: How many levels should the hierarchy have? For example, consider a subnet with 720 routers. If there is no hierarchy, each router needs 720 routing table entries.

If the subnet 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 eight 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.

Reverse Path Forwarding

The idea, called Reverse Path forwarding, is remarkably simple once it has been pointed out. 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.

An example of reverse path forwarding is shown in Fig.12. 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. 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 both reasonably efficient and easy to implement. It does not require routers to know about spanning trees, nor does it have the overhead of a destination list or bit map in each broadcast packet as does multidestination addressing. Nor does it require any special mechanism to stop the process, as flooding does (either a hop counter in each packet and a priori knowledge of the subnet diameter, or a list of packets already seen per source).

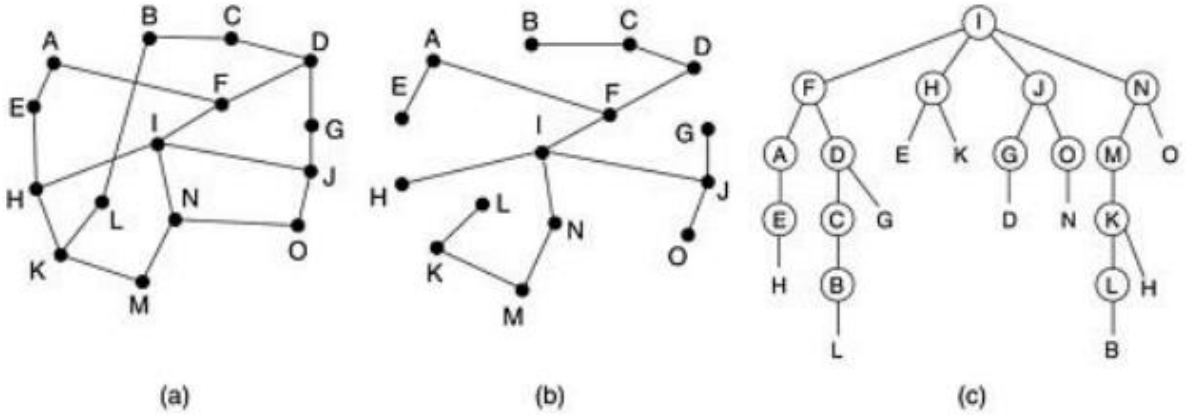


Fig.12. Reverse path forwarding. (a) A subnet. (b) A sink tree. (c) The tree built by reverse path forwarding.

Broadcast Routing Algorithm

Sending a packet to all destinations simultaneously is called broadcasting. 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 broadcasting 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 subnet is for the source to simply send 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. Flooding is another obvious candidate. Although flooding is ill-suited for ordinary point to-point communication, for broadcasting it might rate serious consideration, especially if none of the methods described below are applicable. The problem with flooding as a broadcast technique is the same problem it has as a point-to-point routing algorithm: it generates too many packets and consumes too much bandwidth.

A third algorithm is multidestination routing. If this method is used, 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 and can be treated as a normal packet. Multidestination routing is like separately

addressed packets, except that when several packets must follow the same route, one of them pays full fare and the rest ride free.

A fourth broadcast algorithm makes explicit use of the sink tree for the router initiating the broadcast—or any other convenient spanning tree for that matter. A spanning tree is a subset of the subnet that includes all the routers but contains no loops. 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. 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) but sometimes it is not (e.g., with distance vector routing).

The last broadcast algorithm is an attempt to approximate the behaviour of the previous one, even when the routers do not know anything at all about spanning trees. The idea, called reverse path forwarding, is remarkably simple once it has been pointed out. 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 or not.

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 Algorithm

Sending a message to multiple receivers with a single send operation is called multicasting. Some applications require that widely-separated processes work together in groups, for example, a group of processes implementing a distributed database system. In these situations, it is frequently necessary for one process to send a message to all the other members of the group. If the group is small, it can just send each other member a point-to-point message.

If the group is large, this strategy is expensive. Sometimes broadcasting can be used, but using broadcasting to inform 1000 machines on a million-node network is inefficient because most receivers are not interested in the message Sending a message to such a group is called multicasting, and its routing algorithm is called multicast routing.

Multicasting requires group management. Some way is needed to create and destroy groups, and to allow processes to join and leave groups. It is important that routers know which of their hosts belong to which groups. Either hosts must inform their routers about changes in group membership, or routers must query their hosts periodically. Either way, routers learn about which of their hosts are in which groups. Routers tell their neighbours, so the information propagates through the subnet. To do multicast routing, each router computes a spanning tree covering all other routers.

For example, in Fig. 14(a) there are 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. 14(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. 14(c) shows the pruned spanning tree for group 1. Similarly, Fig. 14(d) shows the pruned spanning tree for group 2. Multicast packets are forwarded only along the appropriate spanning tree.

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. Then the spanning tree can be pruned, starting at the end of each path, working toward the root, and removing all routers that do not belong to the group in question. 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 sender not to send it any more multicasts for that group. When a router with no group members among its own hosts has received such messages on all its lines, it, too, can respond with a PRUNE message. In this way, the subnet is recursively pruned.

One potential disadvantage of this algorithm is that it scales poorly to large networks. Suppose that a network has n groups, each with an average of m members. For each group, m pruned spanning trees must be stored, for a total of mn trees.

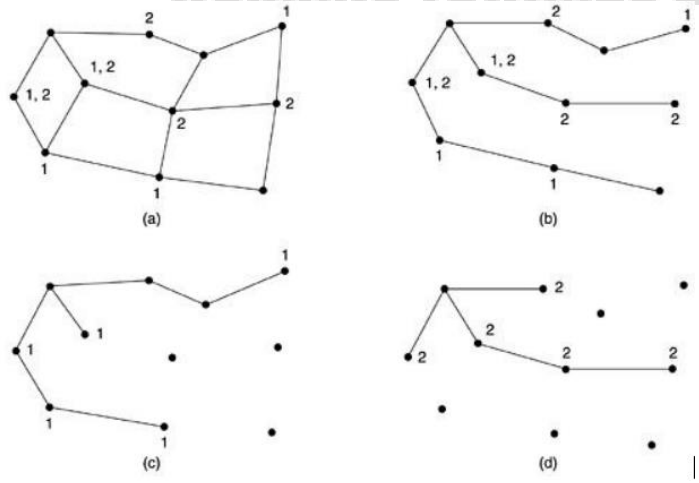
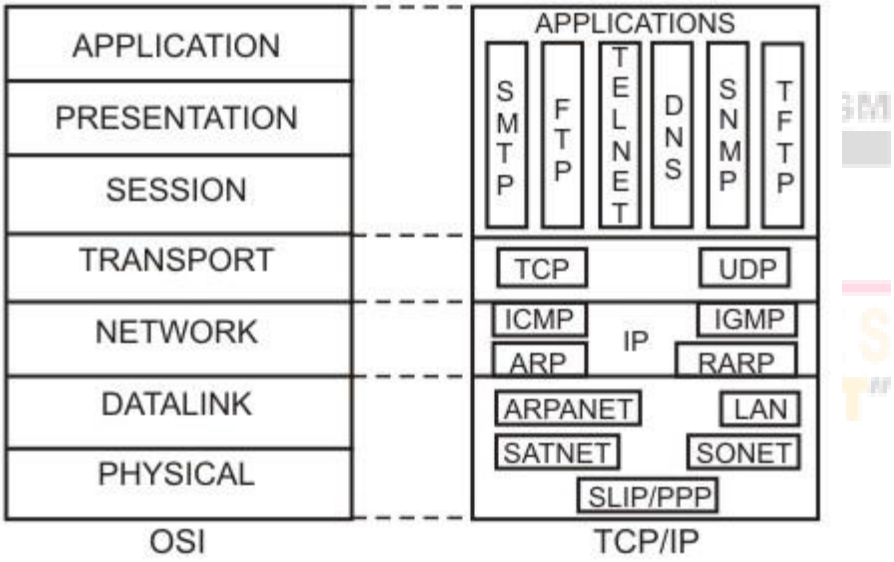


Fig.14 (a) A network. (b) A spanning tree for the leftmost router. (c) A multicast tree for group 1. (d) A multicast tree for group 2.

Addressing

To send a packet from a source node to a destination node correctly through a network, the packet must contain enough information about the destination address. It is also common to include the source address, so that retransmission can be done, if necessary. The addressing scheme used for this purpose has considerable effect on routing.

There are two possible approaches used for addressing; *flat* and *hierarchical*. In *flat addressing* every possible node is assigned a unique number. When a new node is added to the network, it must be given an address within the allowed address range. Addressing used in Ethernet is an example of flat addressing, where addresses (48-bits long) are allocated centrally, blocks of addresses are apportioned to manufactures, so that no two devices in the world will have the same address. Flat addressing has the advantage that if a node is moved from one location to another, it can retain its unique address.



In *hierarchical addressing*, each address consists of a number of fields; as each field is inspected, the packet is taken nearer to the destination. This is very similar to the addressing used in postal system. A significant advantage of hierarchical addressing is that it is possible to relate a hierarchical address structure to the topology of the network, so that routing is simplified. This scheme has the disadvantage that if a host moves from one location to another, a new address needs to be allocated to it, in the same manner that an address change is required as we change house.

IP Addressing

Every host and router on the internet is provided with a unique standard form of network address, which encodes its network number and host number. The combination is unique; no two nodes have the same IP addresses. The IP addresses are 32-bit long having the formats shown in

Fig 6.2.2. The three main address formats are assigned with network addresses (net id) and host address (host id) fields of different sizes.

The class A format allows up to 126 networks with 16 million hosts each. Class B allows up to 16,382 networks with up to 64 K hosts each. Class C allows 2 million networks with up to 254 hosts each. The Class D is used for multicasting in which a datagram is directed to multiple hosts. Addresses beginning with 11110 are reserved for future use. Network addresses are usually written in dotted decimal notation, such as 126.12.15.220, where each byte is written in decimal number corresponding to the binary value. Figure 6.2.3 illustrates how the dotted decimal representation is obtained for a particular IP address in binary form. Range of IP addresses for different classes is given in Fig. 6.2.4.
Some IP addresses, which are used in special situations such as the same host, a host the same network, broadcast on the same network, broadcast on a distant network, or loopback are given in Fig. 6.2.5. This approach of representing IP addresses in terms of classes is known as *classful addressing*. In mid 90's another approach known as *classlees addressing* has been proposed, which may supersede the existing classful addressing approach in future.

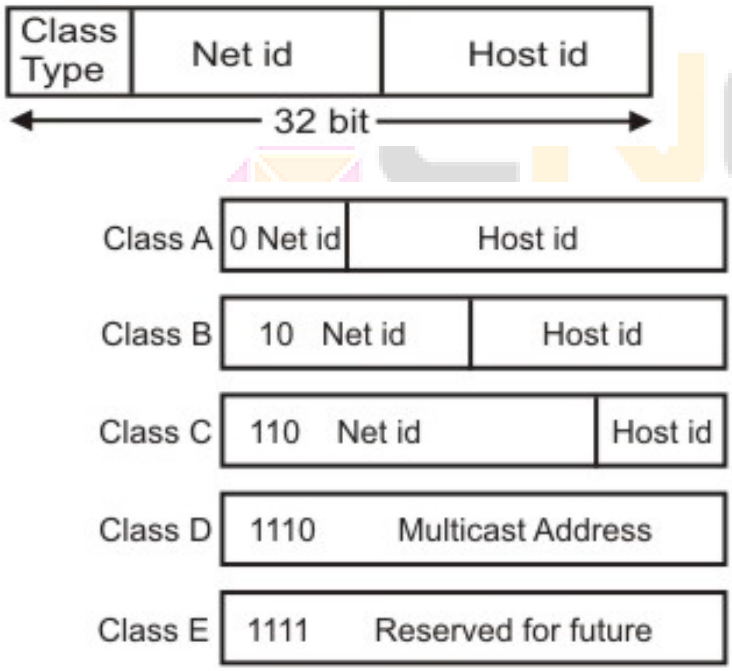


Figure 6.2.2 IP address formats

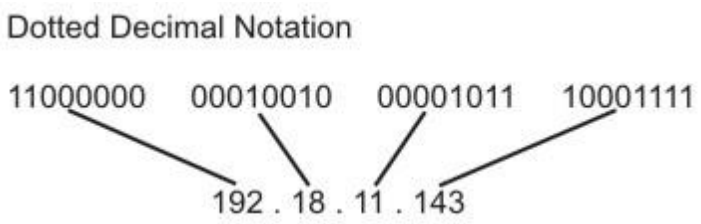


Figure 6.2.3 Dotted decimal representation

Range of Host Addresses

Class A	1.0.0.0	to	127.255.255.255
Class B	128.0.0.0	to	191.255.255.255
Class C	192.0.0.0	to	233.255.255.255
Class D	244.0.0.0	to	239.255.255.255
Class E	240.0.0.0	to	247.255.255.255

Figure 6.2.4 Dotted decimal notation of the IP addresses

00000000 00000000 00000000 00000000		This host
0000 00000 00	hostid	A host on this network
11111111 11111111 11111111 11111111		Broadcast on this network
netid	1111.....1111	Broadcast on a distant network
127	Anything	Loopback

Figure 6.2.5 Special IP addresses

IP Datagram

As we have mentioned earlier, IP is an unreliable and connectionless *best-effort* delivery service protocol. By best effort we mean that there is no error and flow control. However, IP performs error detection and discards a packet, if it is corrupted. To achieve reliability, it is necessary to combine it with a reliable protocol such as TCP. Packets in IP layer are called *datagrams*. The IP header provides information about various functions the IP performs. The IP header format is shown in Fig. 6.2.11. The 20 to 60 octets of header has a number of fields to provide:

-Source and destination IP addresses
-Non transparent fragmentation
-Error checking
-Priority
-Security
-Source routing option
-Route Recording option
-Stream identification
-Time stamping

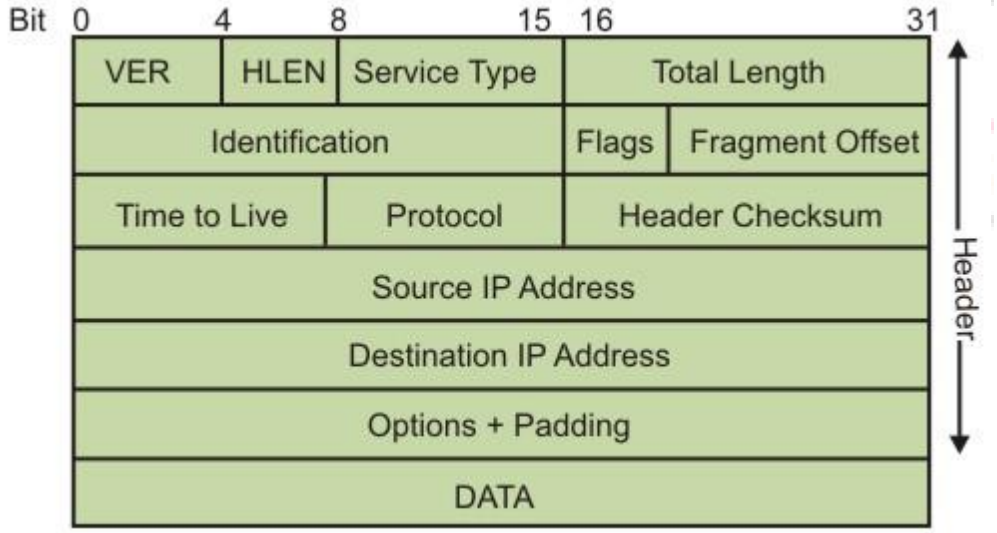


Figure 6.2.11 IP packet format

A brief description of each of the fields are given below:

- VER (4 bits): Version of the IP protocol in use (typically 4).
  HLEN (4 bits): Length of the header, expressed as the number of 32-bit words. Minimum size is 5, and maximum 15.
- Total Length (16 bits): Length in bytes of the datagram, including headers. Maximum datagram size is (216) 65536 bytes.

- Service Type (8 bits): Allows packet to be assigned a priority. Router can use this field to route packets. Not universally used.
- Time to Live (8 bits): Prevents a packet from traveling forever in a loop. Senders set a value that is decremented at each hop. If it reaches zero, packet is discarded.
- Protocol: Defines the higher level protocol that uses the service of the IP layer
- Source IP address (32 bits): Internet address of the sender.
- Destination IP address (32 bits): Internet address of the destination.
- Identification, Flags, Fragment Offset: Used for handling fragmentation.
- Options (variable width): Can be used to provide more functionality to the IP datagram
- Header Checksum (16 bits):

  ■ Covers only the IP header.

  Steps:

  -Header treated as a sequence of 16-bit integers
  -The integers are all added using ones complement arithmetic
  -Ones complement of the final sum is taken as the checksum
  -Datagram is discarded in case of mismatch in checksum values

Multiplexing and Demultiplexing

IP datagram can encapsulate data from several higher-level protocols such as TCP, UDP, ICMP, etc. The Protocol field in the datagram specifies the final destination protocol to which IP datagram to be delivered. When the datagram arrives at the destination, the information in this field is used to perform demultiplex the operation. The multiplexing and demultiplexing operations are shown in Fig. 6.2.1

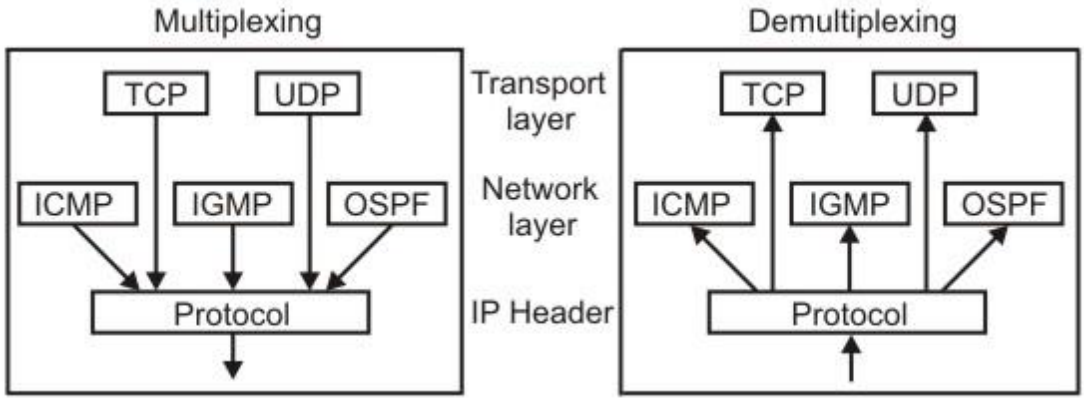


Figure 6.2.1 Multiplexing and demultiplexing in the IP layer

Fragmentation and Reassembly

Each network imposes a limit on maximum size, known as *maximum transfer unit* (MTU) of a packet because of various reasons. One approach is to prevent the problem to occur in the first place, i.e. send packets smaller than the MTU. Second approach is to deal with the problem using fragmentation. When a gateway connects two networks that have different maximum and or minimum packet sizes, it is necessary to allow the gateway to break packets up into fragments, sending each one as an internet packet. The technique is known as *fragmentation*. The following fields of an IP datagram are related to fragmentation:

- **Identification**: A16-bit field identifies a datagram originating from the source host.
- **Flags**: There are 3 bits, the first bit is reserved, the second bit is *do not fragment* bit, and the last bit *is more fragment* bit.
- **Fragmentation offset**: This 13-bit field shows the relative position of the segment with respect to the complete datagram measured in units of 8 bytes.

Figure 6.2.6 shows a fragmentation example, where a packet is fragmented into packets of 1600 bytes. So, the offset of the second fragmented packet is 1600/8 = 200 and the offset of the third fragmented packet is 400 and so on.

The reverse process, known as *reassembly*, which puts the fragments together, is a more difficult task. There are two opposing strategies for performing the re-assembly. In the first case, the fragmentation in one network is made transparent to any subsequent networks. This requires that packets to be reassembled before sending it to subsequent networks as shown in Fig. 6.2.6(a). This strategy is used in ATM. As re-assembly requires sufficient buffer space for storage of all the fragments, this approach has large storage overhead. To overcome this problem in the second strategy, re-assembly is done only at the ultimate destination. This approach does not require large buffer but additional fields are to be added to each packet for independent addressing and to indicate the fragment number as shown in Fig. 6.2.6(b).

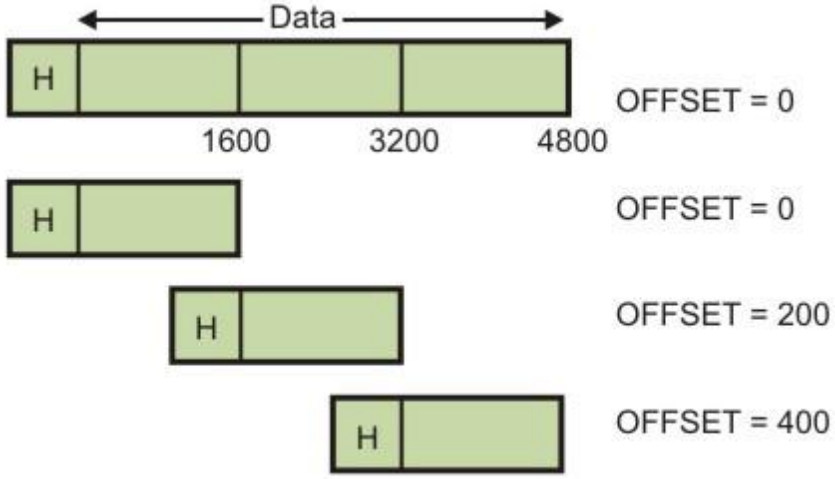
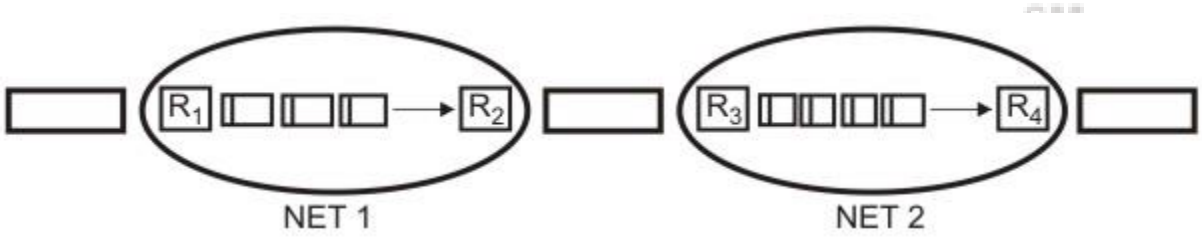


Figure 6.2.6 Fragmentation example



(a) Figure 6.2.6 (a) Transparent Fragmentation (ATM)

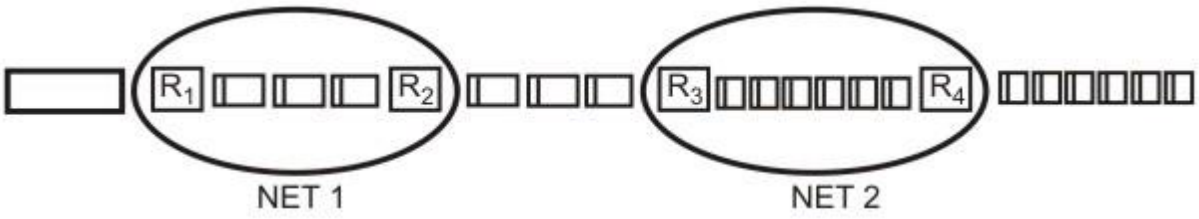


Fig (b) Nontransparent fragmentation (IP)

ICMP (*Internet Control Message Protocol*)

To make efficient use of the network resources, IP was designed to provide unreliable and connectionless best-effort datagram delivery service. As a consequence, IP has no error-control mechanism and also lacks mechanism for host and management queries. A companion protocol

known as *Internet Control Message Protocol* (ICMP), has been designed to compensate these two deficiencies. ICMP messages can be broadly divided into two broad categories: error reporting messages and query messages as follows.

- Error reporting Messages: Destination unreachable, Time exceeded, Source quench, Parameter problems, Redirect
- Query: Echo request and reply, Timestamp request and reply, Address mask request and reply

The frame formats of these query and messages are shown in Fig. 6.2.5.

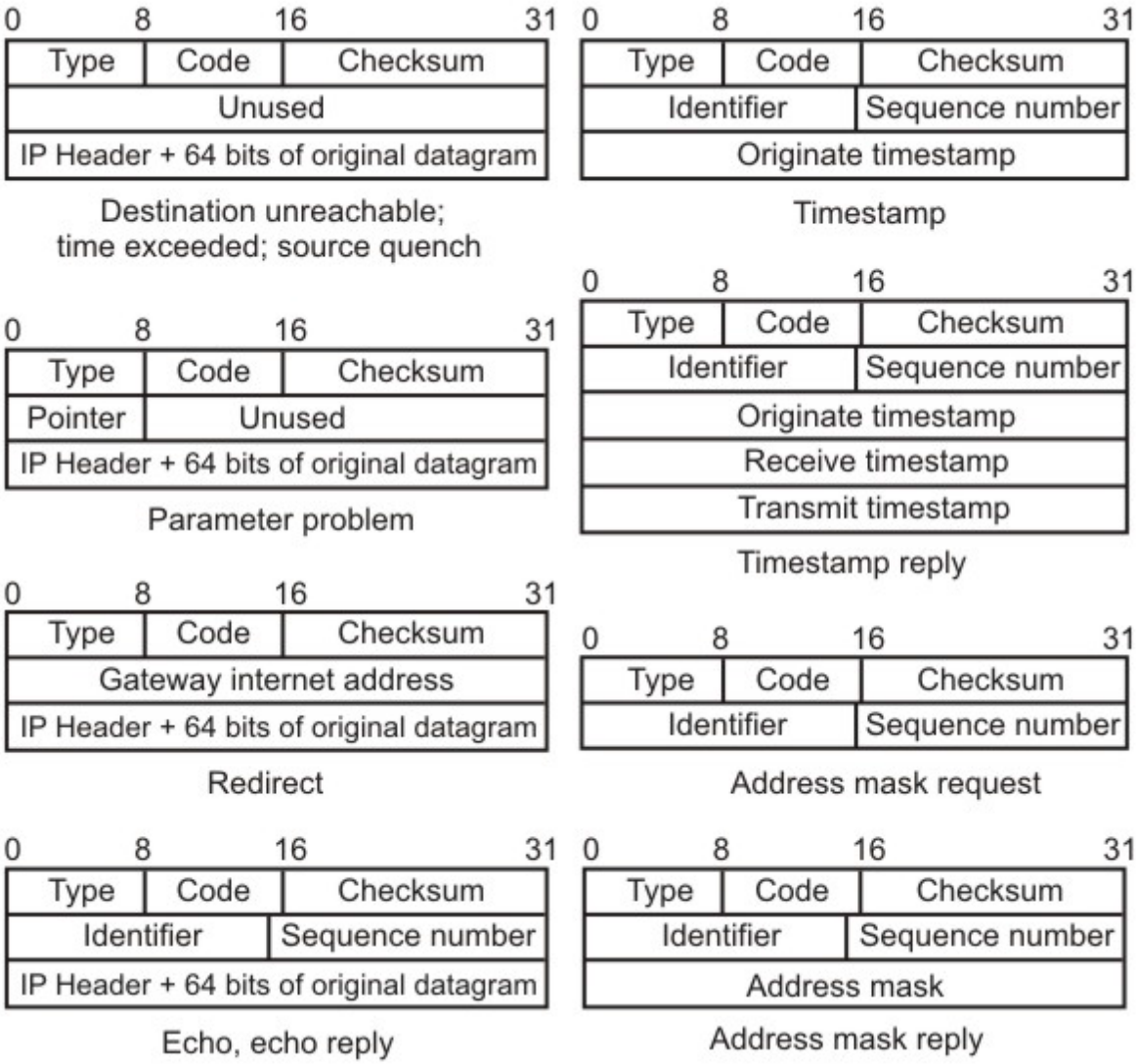


Figure 6.2.5 ICMP message formats

IPV6

The network layer that is present in use in commonly referred to as IPv4. Although IPv4 is well designed and has helped the internet to grow rapidly, it has some deficiencies, These deficiencies has made it unsuitable for the fast growing internet. To overcome these deficiencies, Internet Protocol, Version 6 protocol has been proposed and it has evolved into a standard. Important features of IPv6 are highlighted below:

- IPv6 uses 128-bit address instead of 32-bit address to provide larger address space
- Uses more flexible header format, which simplifies and speeds up the routing process
- Basic header followed by extended header
- Resource Allocation options, which was not present in IPv4
- Provision of new/future protocol options
- Support for security with the help of encryption and authentication
- Support for fragmentation at source

Comparative Study between IPv4 and IPv6:

IPv4 and IPv6 are internet protocol version 4 and internet protocol version 6, IP version 6 is the new version of Internet Protocol, which is way better than IP version 4 in terms of complexity and efficiency.

Difference Between IPv4 and IPv6:

IPV4	IPV6
IPv4 has 32-bit address length	IPv6 has 128-bit address length
It Supports Manual and DHCP address configuration	It supports Auto and renumbering address configuration
In IPv4 end to end connection integrity is Unachievable	In IPv6 end to end connection integrity is Achievable
It can generate 4.29×109 address space	Address space of IPv6 is quite large it can produce 3.4×1038 address space
Security feature is dependent on application	IPSEC is inbuilt security feature in the IPv6 protocol
Address representation of IPv4 in decimal	Address Representation of IPv6 is in hexadecimal
Fragmentation performed by Sender and forwarding routers	In IPv6 fragmentation performed only by sender

IPV4	IPV6
In IPv4 Packet flow identification is not available	In IPv6 packetflow identification are Available and uses flow label field in the header
In IPv4 checksumfield is available	In IPv6 checksumfield is not available
It has broadcast Message Transmission Scheme	In IPv6 multicast and any cast message transmission scheme is available
In IPv4 Encryption and Authentication facility not provided	In IPv6 Encryption and Authentication are provided
IPv4 has header of 20-60 bytes.	IPv6 has header of 40 bytes fixed

References:

1.Andrew S. Tanenbaum, David J. Wetherall, “Computer Networks” Pearson Education.

2 Douglas E Comer, “Internetworking WithTcp/Ip Principles, Protocols, And Architecture - Volume I” 6 th Edition,Pearson Education

3.DimitriBertsekas, Robert Gallager, “Data Networks”, PHI Publication, Second Edition.