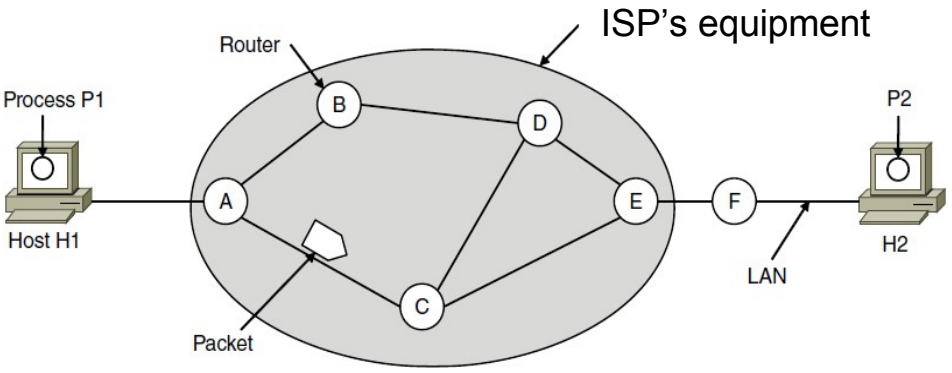
UNIT – 4

The 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.

4.1 NETWORK LAYER DESIGN ISSUES:

4.1.1 Store-and-Forward Packet Switching:



The major components of the system are the carrier'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 carrier's routers, A, by a leased line. In contrast, H2 is on a LAN with a router, F, owned and operated by the customer.

This equipment is used as follows. A host with a packet to send transmits it to the nearest router, either on its own LAN or over a point-to-point link to the carrier. The packet is stored there until it has fully arrived so the checksum can be verified. Then it is forwarded to the next router along the path until it reaches the destination host, where it is delivered. This mechanism is **store-and-forward packet switching.**

4.1.2 Services Provided to the Transport Layer:

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

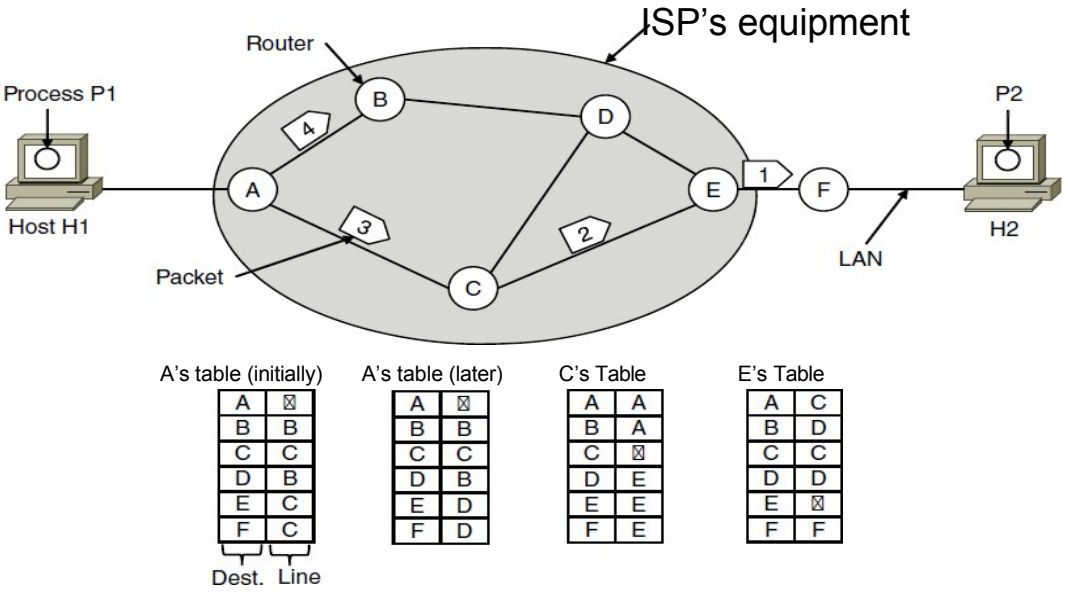
1. The services should be independent of the router technology.
2. The transport layer should be shielded from the number, type, and topology of the routers present.
3. The network addresses made available to the transport layer should use a uniform numbering plan, even across LANs and WANs.

The network service should be connectionless, with primitives SEND PACKET and RECEIVE PACKET and little else. In particular, no packet ordering and flow control should be done, because the hosts are going to do that anyway, and there is usually little to be gained by doing it twice. Furthermore, each packet must carry the full destination address, because each packet sent is carried independently of its predecessors, if any.

4.1.3 Implementation of Connectionless Service:

In connectionless service, packets are injected into the subnet individually and routed independently of each other. No advance setup is needed. In this context, the packets are frequently called **datagrams** and the subnet is called a **datagram subnet.**

Let us now see how a datagram subnet works,



The process P1 on host H1 has a long message for P2 on host H2. The network layer has to break a message into four packets, 1, 2, 3, and 4 and sends each of them in turn to router A. A has only two outgoing lines to B and C so every incoming packet must be sent to one of these routers, even if the ultimate destination is some other router. A's initial routing table is shown in the figure under the label "initially."

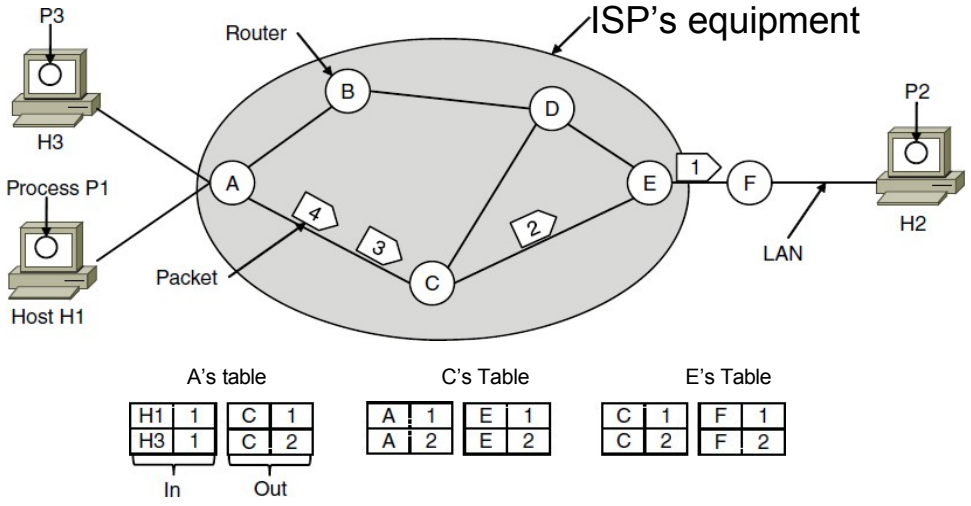
As they arrived at A, packets 1, 2, and 3 were stored briefly (to verify their checksums). Then each was forwarded to C according to A's table. Packet 1 was then forwarded to E and then to F. When it got to F, it was encapsulated in a data link layer frame and sent to H2 over the LAN. Packets 2 and 3 follow the same route.

However, something different happened to packet 4. When it got to A it was sent to router B, even though it is also destined for F. For some reason, perhaps it learned of a traffic jam somewhere along the ACE path and updated its routing table, as shown under the label "later." The algorithm that manages the tables and makes the routing decisions is called the routing algorithm.

4.1.4 Implementation of Connection-Oriented Service:

In connection-oriented service, a path from the source router to the destination router must be established before any data packets can be sent. This connection is called a **VC (virtual circuit),** and the subnet is called a **virtual-circuit subnet.**

The idea behind virtual circuits is to avoid having to choose a new route for every packet sent. Instead, when a connection is established, a route from the source machine to the destination machine is chosen as part of the connection setup and stored in tables inside the routers.



Here, host H1 has established connection 1 with host H2. It is remembered as the first entry in each of the routing tables. The first line of A's table says that if a packet bearing connection

identifier 1 comes in from H1, it is to be sent to router C and given connection identifier 1. Similarly, the first entry at C routes the packet to E, also with connection identifier 1. Similarly if H3 wants to connect to H2 they have to do the same procedure and it has to use different connection identifier in above example host H3 uses connection identifier as one.

4.1.5 Comparison of Virtual-Circuit and Datagram Subnets:

Issue	Datagram network	Virtual-circuit network
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

4.2 ROUTING ALGORITHMS:

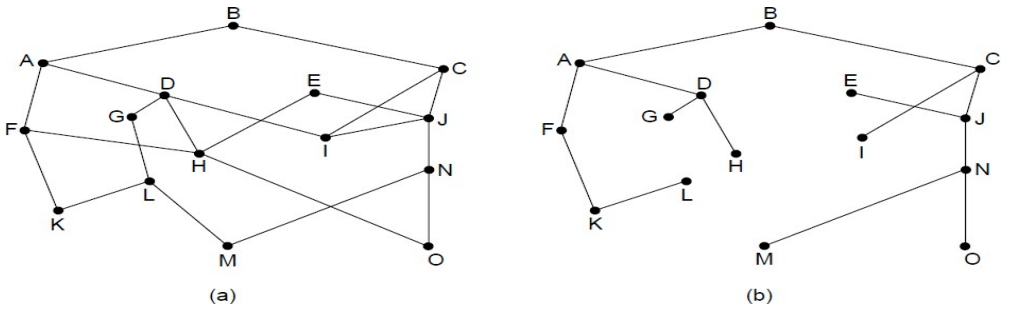
The main function of the network layer is routing packets from the source machine to the destination machine. In most subnets, packets will require multiple hops to make the journey. The routing algorithm is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagrams internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time.

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. This procedure is sometimes called **Dynamic routing.**

4.2.1 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.
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 below Fig. where the distance metric is the number of hops. Note that a sink tree is not necessarily unique; 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.



4.2.2 Shortest Path Routing:

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 below 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).

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

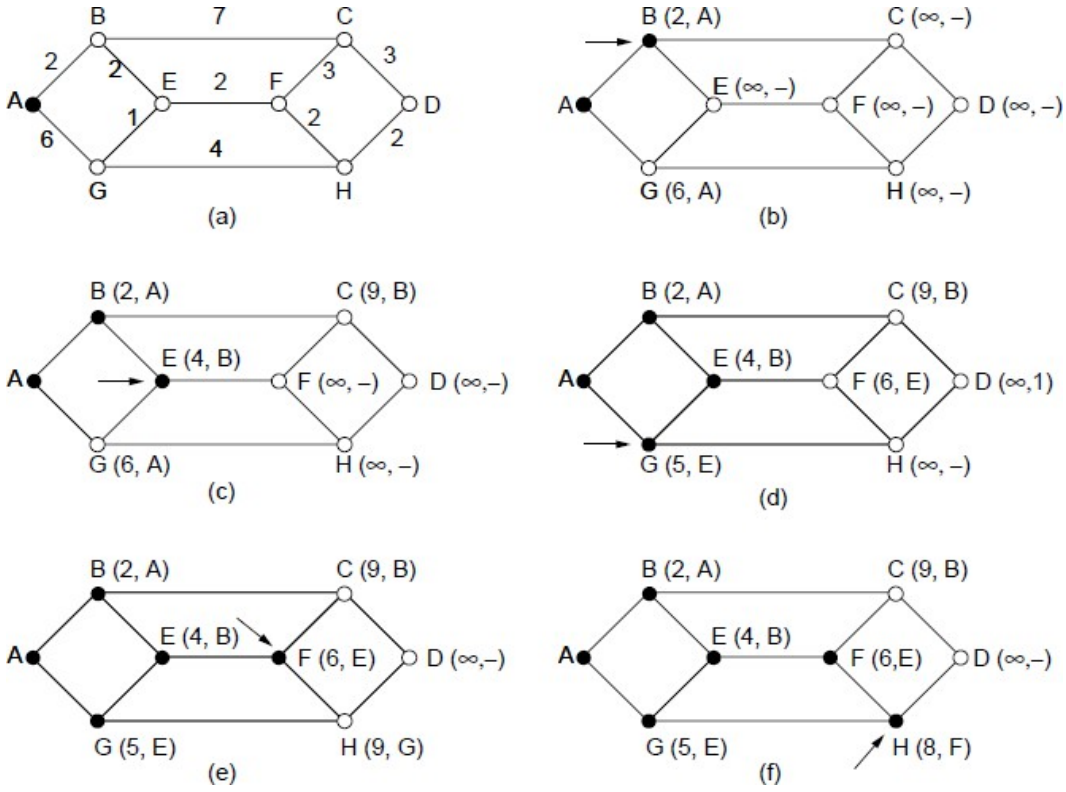
label may be either tentative or permanent. Initially, all labels are tentative. When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

To illustrate how the labeling algorithm works, look at the weighted, undirected graph of **Fig.(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. 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.(b).** This one becomes the **new working node**.

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

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

Now consider the case where Z is still tentatively 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.



4.2.3 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.

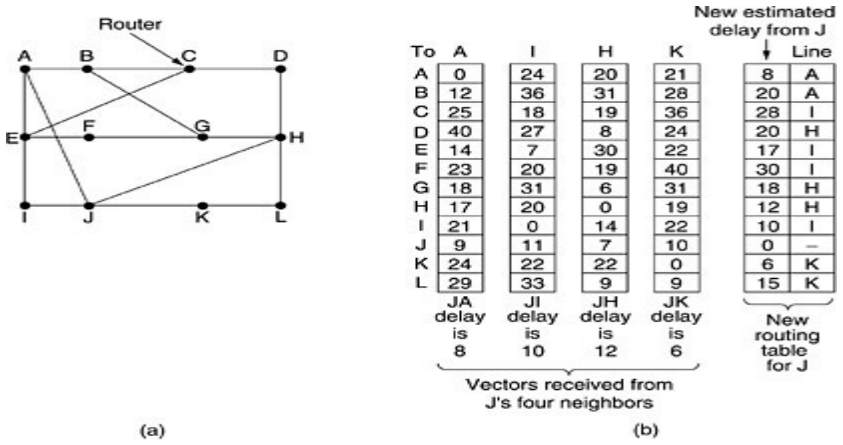
An alternative technique for damming the flood is to keep track of which packets have been flooded, to avoid sending them out a second time. Achieve this goal is to have the source router put **a sequence number** in each packet it receives from its hosts. Each router then needs a list per source router telling which sequence numbers originating at that source have already been seen.

4.2.4 Intra- and Interdomain Routing:

Today, an internet can be so large that one routing protocol cannot handle the task of updating the routing tables of all routers. For this reason, an internet is divided into autonomous systems. An **autonomous system (AS)** is a group of networks and routers under the authority of a single administration. Routing inside an autonomous system is referred to as **intradomain** routing. Routing between autonomous systems is referred to as **interdomain** routing.

4.2.5 Distance Vector Routing:

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



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

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

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

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

Example.

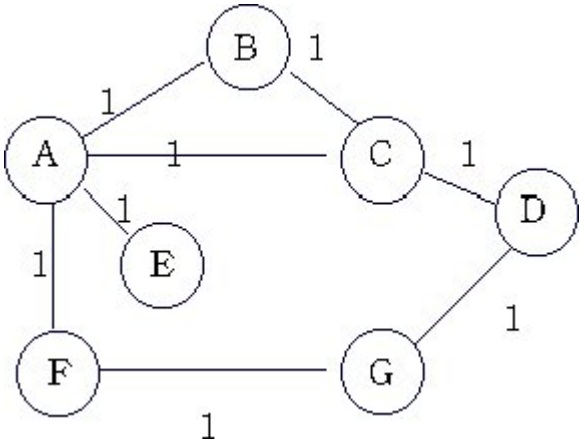


Table 1. Initial distances stored at each node(global view).

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

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

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

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

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

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

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

(Destination, Cost, NextHop).

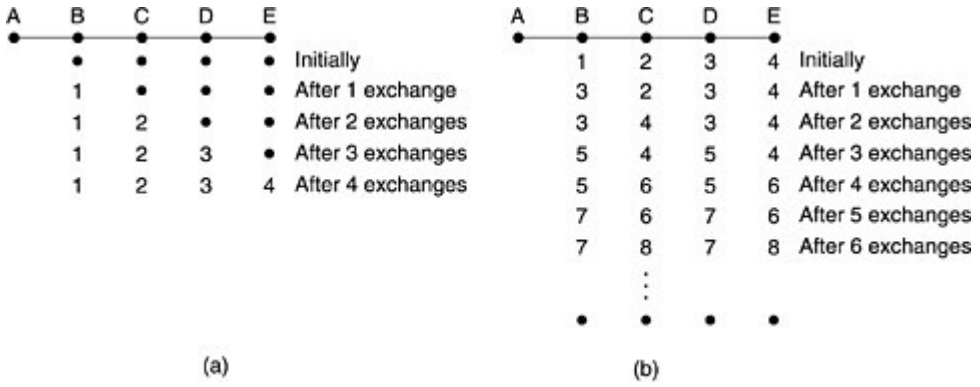
For example, Table 3 shows the complete routing table maintained at node B for the network in figure1.

Table 3. Routing table maintained at node B.

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

THE COUNT-TO-INFINITY PROBLEM

The count-to-infinity problem.



- Consider the five-node (linear) subnet of above fig, where the delay metric is the number of hops. Suppose *A* is down initially and all the other routers know this. In other words, they have all recorded the delay to *A* as infinity.
- Now let us consider the situation of Fig (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.

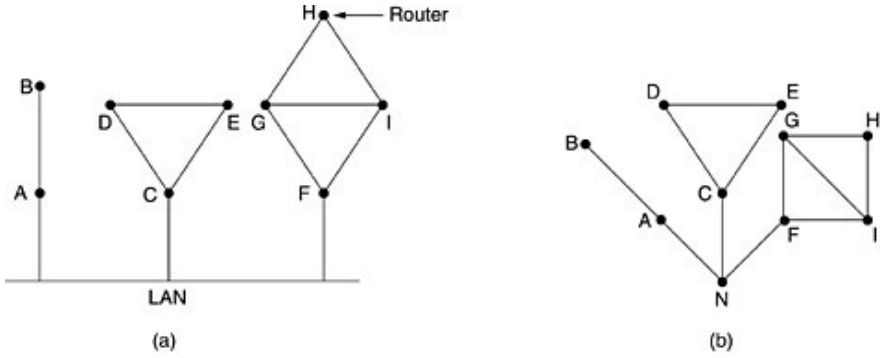
4.2.6 LINK STATE ROUTING:

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

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

Learning about the Neighbours

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



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

Measuring Line Cost

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

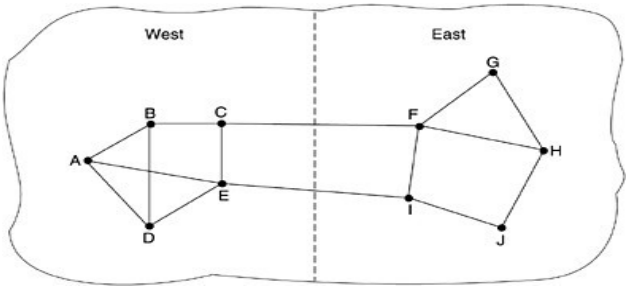
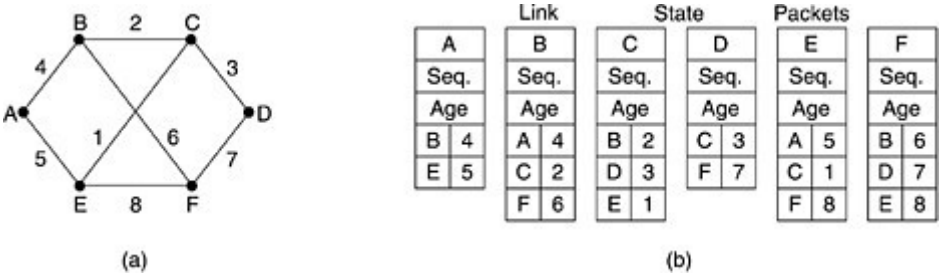


Figure: A subnet in which the East and West parts are connected by two lines.

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

Building Link State Packets



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

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

Distributing the Link State Packets

			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.**

- In above Fig. 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*.

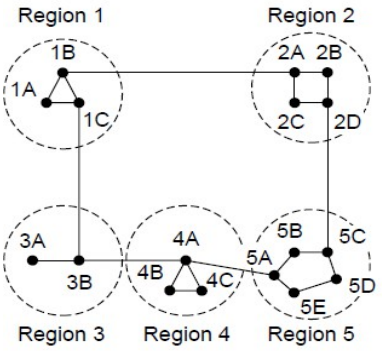
4.2.7 Hierarchical Routing:

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.

When hierarchical routing is used, the routers are divided into what we will call **regions**, with each router knowing all the details about how to route packets to destinations within its own region, but knowing nothing about the internal structure of other regions.

For huge networks, a two-level hierarchy may be insufficient; it may be necessary to group the **regions** into **clusters**, the **clusters** into **zones**, the **zones** into **groups**, and so on, until we run out of names for aggregations.

Below Fig. (a) 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. (b). When routing is done hierarchically, as in Fig. (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)

4.3 CONGESTION CONTROL ALGORITHMS

- When too many packets are present in (a part of) the subnet, performance degrades. This situation is called **congestion**.
- Below Figure depicts the symptom. When the number of packets dumped into the subnet by the hosts is within its carrying capacity, they are all delivered (except for a few that are afflicted with transmission errors) and the number delivered is proportional to the number sent.
- However, as traffic increases too far, the routers are no longer able to cope and they begin losing packets. This tends to make matters worse. At very high traffic, performance collapses completely and almost no packets are delivered.

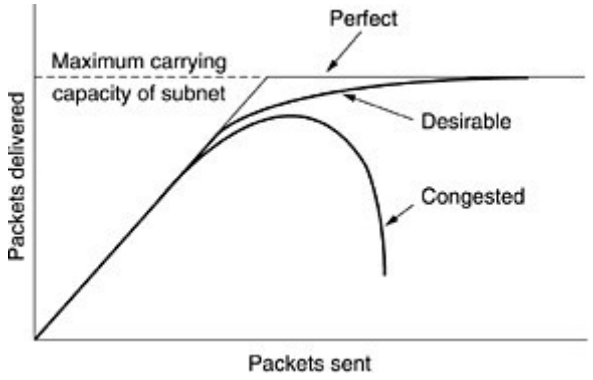


Fig:. When too much traffic is offered, congestion sets in and performance degrades sharply.

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

4.3.1 APPROACHES TO CONGESTION CONTROL

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

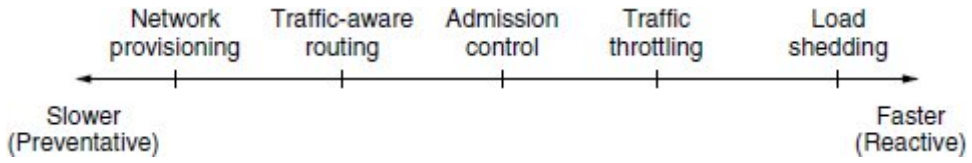


Fig: Timescales Of Approaches To Congestion Control

- Open loop solutions attempt to solve the problem by good design.
- Tools for doing open-loop control include deciding when to accept new traffic, deciding when to discard packets and which ones, and making scheduling decisions at various points in the network.
- Closed loop solutions are based on the concept of a feedback loop.
- This approach has three parts when applied to congestion control:
  1. Monitor the system to detect when and where congestion occurs.
  2. Pass this information to places where action can be taken.
  3. Adjust system operation to correct the problem.
- A variety of metrics can be used to monitor the subnet for congestion. Chief among these are the percentage of all packets discarded for lack of buffer space, the average queue lengths, the number of packets that time out and are retransmitted, the average packet delay,

and the standard deviation of packet delay. In all cases, rising numbers indicate growing congestion.

- The second step in the feedback loop is to transfer the information about the congestion from the point where it is detected to the point where something can be done about it.
- In all feedback schemes, the hope is that knowledge of congestion will cause the hosts to take appropriate action to reduce the congestion.
- The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. Two solutions come to mind: increase the resources or decrease the load.

4.3.2 CONGESTION PREVENTION POLICIES

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

Layer	Policies
Transport	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy • Timeout determination
Network	• Virtual circuits versus datagram inside the subnet • Packet queueing and service policy • Packet discard policy • Routing algorithm • Packet lifetime management
Data link	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy

Figure 5-26. Policies that affect congestion.

The **data link layer Policies.**

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

retransmissions may result. A tight flow control scheme (e.g., a small window) reduces the data rate and thus helps fight congestion.

The **network layer Policies.**

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

The **transport layer Policies**,

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

4.3.3 ADMISSION CONTROL

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

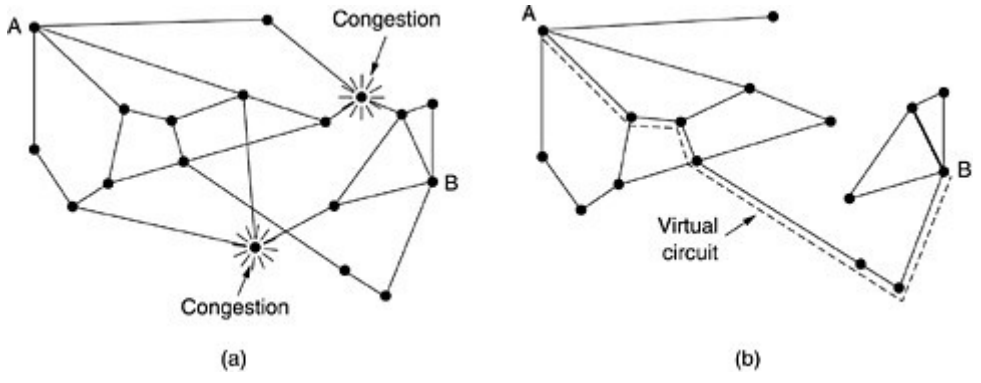


Figure 5-27. (a) A congested subnet. (b) A redrawn subnet that eliminates the congestion. A virtual circuit from A to B is also shown.

Suppose that a host attached to router *A* wants to set up a connection to a host attached to router *B*. Normally, this connection would pass through one of the congested routers. To avoid

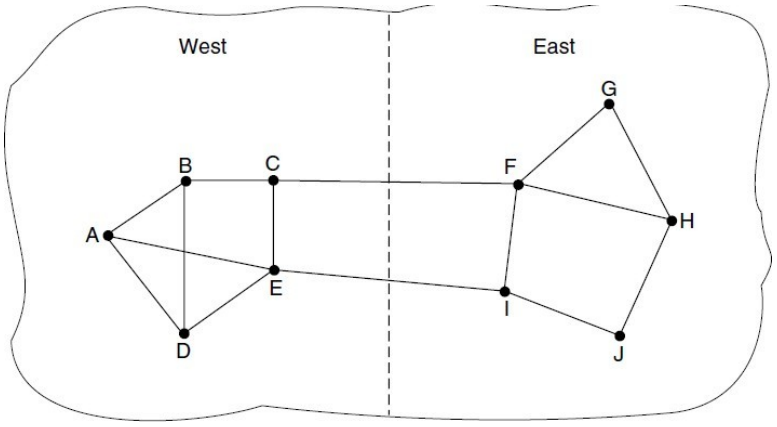
this situation, we can redraw the subnet as shown in Fig. 5-27(b), omitting the congested routers and all of their lines. The dashed line shows a possible route for the virtual circuit that avoids the congested routers.

4.3.4 TRAFFIC AWARE ROUTING

These schemes adapted to changes in topology, but not to changes in load. The goal in taking load into account when computing routes is to shift traffic away from hotspots that will be the first places in the network to experience congestion.

The most direct way to do this is to set the link weight to be a function of the (fixed) link bandwidth and propagation delay plus the (variable) measured load or average queuing delay. Least-weight paths will then favor paths that are more lightly loaded, all else being equal.

Consider the network of Fig. 5-23, which is divided into two parts, East and West, connected by two links, *CF* and *EI*. Suppose that most of the traffic between East and West is using link *CF*, and, as a result, this link is heavily loaded with long delays. Including queueing delay in the weight used for 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*, loading this link. 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 and propagation delay are considered, this problem does not occur. Attempts to include load but change weights within a narrow range only slow down routing oscillations. Two techniques can contribute to a successful solution. The first is multipath routing, in which there can be multiple paths from a source to a destination. In our example this means that the traffic can be spread across both of the East to West links. The second one is for the routing scheme to shift traffic across routes slowly enough that it is able to converge.

4.3.5 TRAFFIC THROTTLING

- Each router can easily monitor the utilization of its output lines and other resources. For example, it can associate with each line a real variable, *u*, whose value, between 0.0 and 1.0, reflects the recent utilization of that line. To maintain a good estimate of *u*, a sample of the instantaneous line utilization, *f* (either 0 or 1), can be made periodically and *u* updated according to

$$u_{new} = au_{old} + (1 - a)f$$

where the constant a determines how fast the router forgets recent history.

Whenever u moves above the threshold, the output line enters a "warning" state. Each newly-arriving packet is checked to see if its output line is in warning state. If it is, some action is taken. The action taken can be one of several alternatives, which we will now discuss.

4.3.6 THE WARNING BIT

- The old DECNET architecture signaled the warning state by setting a special bit in the packet's header.
- When the packet arrived at its destination, the transport entity copied the bit into the next acknowledgement sent back to the source. The source then cut back on traffic.
- As long as the router was in the warning state, it continued to set the warning bit, which meant that the source continued to get acknowledgements with it set.
- The source monitored the fraction of acknowledgements with the bit set and adjusted its transmission rate accordingly. As long as the warning bits continued to flow in, the source continued to decrease its transmission rate. When they slowed to a trickle, it increased its transmission rate.
- Note that since every router along the path could set the warning bit, traffic increased only when no router was in trouble.

4.3.7 CHOKE PACKETS

- In this approach, the router sends a **choke packet** back to the source host, giving it the destination found in the packet.
- The original packet is tagged (a header bit is turned on) so that it will not generate any more choke packets farther along the path and is then forwarded in the usual way.
- When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination by X percent. Since other packets aimed at the same destination are probably already under way and will generate yet more choke packets, the host should ignore choke packets referring to that destination for a fixed time interval. After that period has expired, the host listens for more choke packets for another interval. If one arrives, the line is still congested, so the host reduces the flow still more and begins ignoring choke packets again. If no choke packets arrive during the listening period, the host may increase the flow again.
- The feedback implicit in this protocol can help prevent congestion yet not throttle any flow unless trouble occurs.
- Hosts can reduce traffic by adjusting their policy parameters.
- Increases are done in smaller increments to prevent congestion from reoccurring quickly.
- Routers can maintain several thresholds. Depending on which threshold has been crossed, the choke packet can contain a mild warning, a stern warning, or an ultimatum.

4.3.8 HOP-BY-HOP BACK PRESSURE

- At high speeds or over long distances, sending a choke packet to the source hosts does not work well because the reaction is so slow.

Consider, for example, a host in San Francisco (router *A* in Fig. 5-28) that is sending traffic to a host in New York (router *D* in Fig. 5-28) at 155 Mbps. If the New York host begins to run out of buffers, it will take about 30 msec for a choke packet to get back to San Francisco to tell it to slow down. The choke packet propagation is shown as the second, third, and fourth steps in Fig. 5-28(a). In those 30 msec, another 4.6 megabits will have been sent. Even if the host in San Francisco completely shuts down immediately, the 4.6 megabits in the pipe will continue to pour in and have to be dealt with. Only in the seventh diagram in Fig. 5- 28(a) will the New York router notice a slower flow.

An alternative approach is to have the choke packet take effect at every hop it passes through, as shown in the sequence of Fig. 5-28(b). Here, as soon as the choke packet reaches *F*, *F* is required to reduce the flow to *D*. Doing so will require *F* to devote more buffers to the flow, since the source is still sending away at full blast, but it gives *D* immediate relief, like a headache remedy in a television commercial. In the next step, the choke packet reaches *E*, which tells *E* to reduce the flow to *F*. This action puts a greater demand on *E*'s buffers but gives *F* immediate relief. Finally, the choke packet reaches *A* and the flow genuinely slows down.

The net effect of this hop-by-hop scheme is to provide quick relief at the point of congestion at the price of using up more buffers upstream. In this way, congestion can be nipped in the bud without losing any packets.

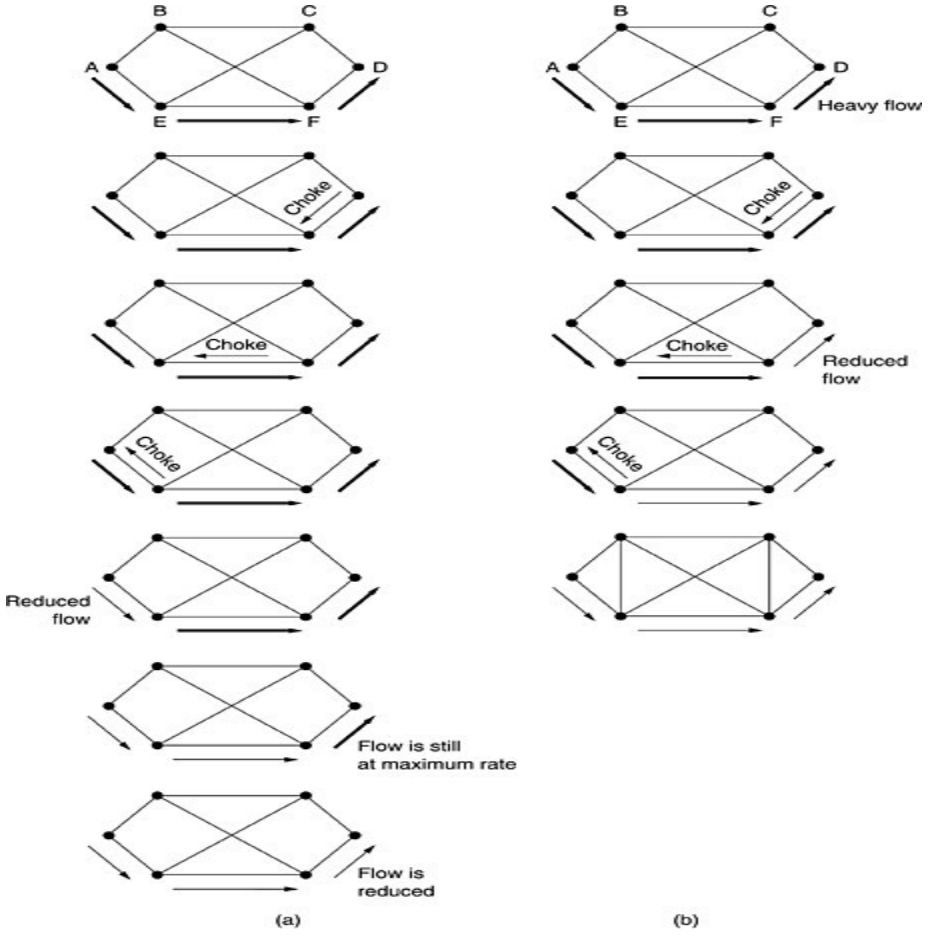


Figure 5-28. (a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through.

4.3.9 LOAD SHEDDING

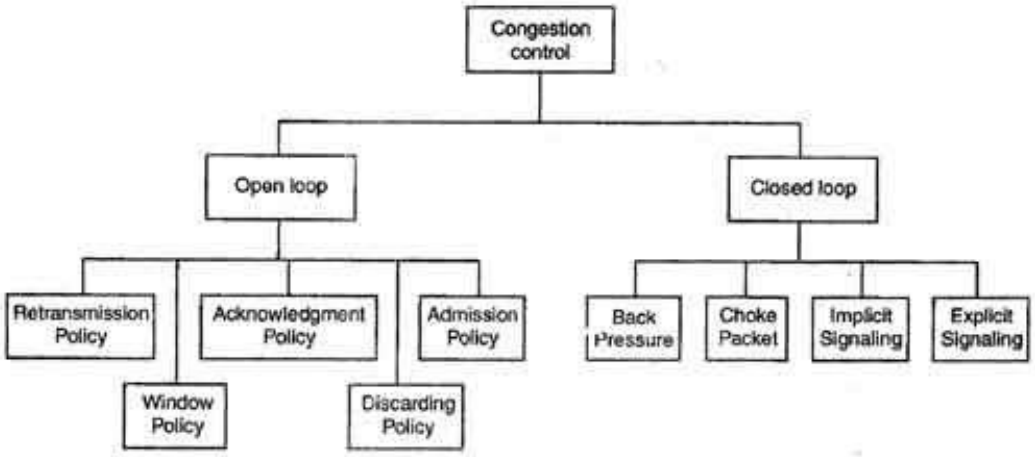
- When none of the above methods make the congestion disappear, routers can bring out the heavy artillery: load shedding.
- **Load shedding** is a fancy way of saying that when routers are being in undated by packets that they cannot handle, they just throw them away.
- A router drowning in packets can just pick packets at random to drop, but usually it can do better than that.
- Which packet to discard may depend on the applications running.
- To implement an intelligent discard policy, applications must mark their packets in priority classes to indicate how important they are. If they do this, then when packets have to be discarded, routers can first drop packets from the lowest class, then the next lowest class, and so on.

4.3.10 RANDOM EARLY DETECTION

- It is well known that dealing with congestion after it is first detected is more effective than letting it gum up the works and then trying to deal with it. This observation leads to the idea of discarding packets before all the buffer space is really exhausted. A popular algorithm for doing this is called **RED** (**Random Early Detection)**.
- In some transport protocols (including TCP), the response to lost packets is for the source to slow down. The reasoning behind this logic is that TCP was designed for wired networks and wired networks are very reliable, so lost packets are mostly due to buffer overruns rather than transmission errors. This fact can be exploited to help **reduce congestion**.
- By having routers drop packets before the situation has become hopeless (hence the "early" in the name), the idea is that there is time for action to be taken before it is too late. To determine when to start discarding, routers maintain a running average of their queue lengths. When the average queue length on some line exceeds a threshold, the line is said to be congested and action is taken.

4.3.12 How to correct the Congestion Problem:

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



Types of Congestion Control Methods

These two categories are:

1. Open loop
2. Closed loop

Open Loop Congestion Control

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

1. Retransmission Policy

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

2. Window Policy

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

3. Acknowledgement Policy

• The acknowledgement policy imposed by the receiver may also affect congestion.
• If the receiver does not acknowledge every packet it receives it may slow down the sender and help prevent congestion.

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

4. Discarding Policy

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

5. Admission Policy

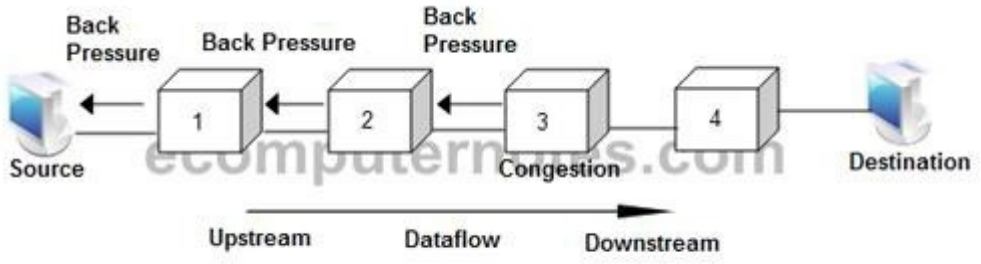
• An admission policy, which is a quality-of-service mechanism, can also prevent congestion in virtual circuit networks.
• Switches in a flow first check the resource requirement of a flow before admitting it to the network.
• A router can deny establishing a virtual circuit connection if there is congestion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

Closed Loop Congestion Control

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

1. Backpressure

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



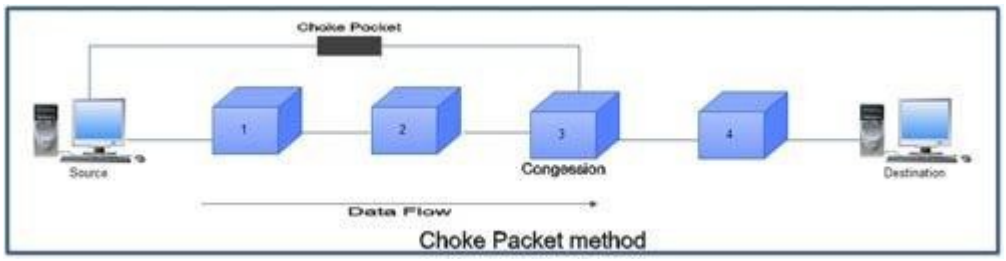
Backpressure Method

• The backpressure technique can be applied only to virtual circuit networks. In such virtual circuit each node knows the upstream node from which a data flow is coming.
• In this method of congestion control, the congested node stops receiving data from the immediate upstream node or nodes.
• This may cause the upstream node on nodes to become congested, and they, in turn, reject data from their upstream node or nodes.
• As shown in fig node 3 is congested and it stops receiving packets and informs its upstream node 2 to slow down. Node 2 in turns may be congested and informs node 1 to slow down.

Now node 1 may create congestion and informs the source node to slow down. In this way the congestion is alleviated. Thus, the pressure on node 3 is moved backward to the source to remove the congestion.

2. Choke Packet

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



Choke Packet method

3. Implicit Signaling

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

4. Explicit Signaling

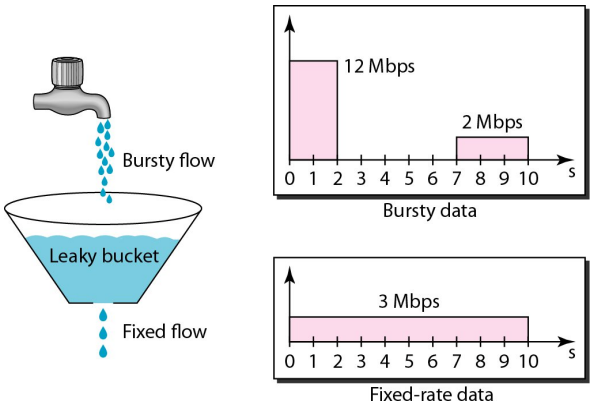
• In this method, the congested nodes explicitly send a signal to the source or destination to inform about the congestion.
• Explicit signaling is different from the choke packet method. In choke packed method, a separate packet is used for this purpose whereas in explicit signaling method, the signal is included in the packets that carry data .
• Explicit signaling can occur in either the forward direction or the backward direction .
• In backward signaling, a bit is set in a packet moving in the direction opposite to the congestion. This bit warns the source about the congestion and informs the source to slow down.
• In forward signaling, a bit is set in a packet moving in the direction of congestion. This bit warns the destination about the congestion. The receiver in this case uses policies such as slowing down the acknowledgements to remove the congestion.

4.3.13 Traffic Control Algorithm

Traffic shaping is a mechanism to control the amount and the rate of the traffic sent to the network. Two techniques can shape traffic: leaky bucket and token bucket.

1. ***Leaky Bucket***

If a bucket has a small hole at the bottom, the water leaks from the bucket at a constant rate as long as there is water in the bucket. The rate at which the water leaks does not depend on the rate at which the water is input to the bucket unless the bucket is empty. The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate. Below Figure shows a leaky bucket and its effects.

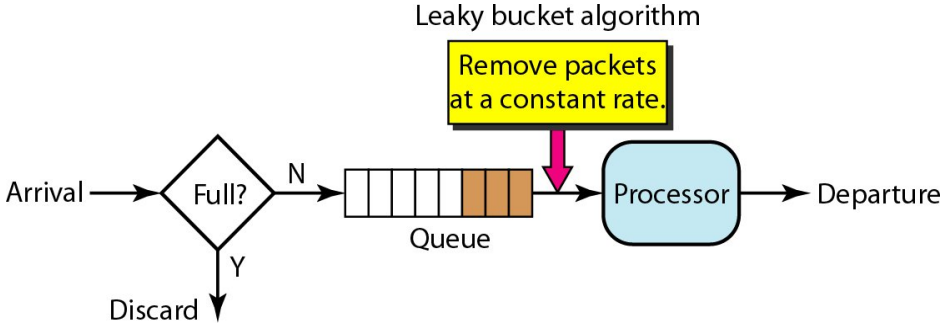


In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment. In Figure 24.19 the host sends a burst of data at a rate of 12 Mbps for 2 s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in l0s. The leaky bucket smooth's the traffic by sending out data at a rate of 3 Mbps during the same 10 s. Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion.

A simple leaky bucket implementation is shown in below Figure. A FIFO queue holds the packets. If the traffic consists of fixed-size packets, the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.

The following is an algorithm for variable-length packets:

1. Initialize a counter to *n* at the tick of the clock.
2. If *n* is greater than the size of the packet, send the packet and decrement the counter by the packet size. Repeat this step until *n* is smaller than the packet size.
3. Reset the counter and go to step 1.

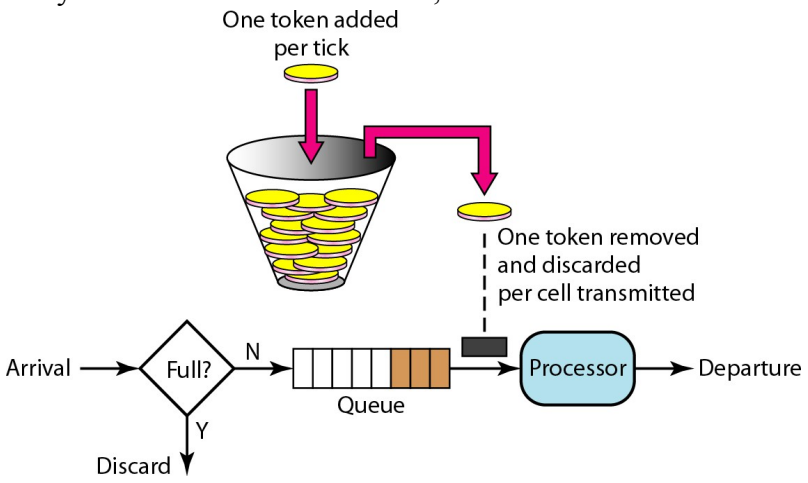


2. ***Token Bucket***

The leaky bucket is very restrictive. It does not credit an idle host. For example, if a host is not sending for a while, its bucket becomes empty. Now if the host has bursty data, the leaky bucket allows only an average rate. The time when the host was idle is not taken into account. On the other hand, the token bucket algorithm allows idle hosts to accumulate credit for the future in the form of tokens. For each tick of the clock, the system sends *n* tokens to the bucket. The system removes one token for every cell (or byte) of data sent. For example, if *n* is 100 and the host is idle for 100 ticks,

the bucket collects 10,000 tokens. Now the host can consume all these tokens in one tick with 10,000 cells, or the host takes 1000 ticks with 10 cells per tick. In other words, the host can send bursty data as long as the bucket is not empty. Below Figure shows the idea.

The token bucket can easily be implemented with a counter. The token is initialized to zero. Each time a token is added, the counter is incremented by 1. Each time a unit of data is sent, the counter is decremented by 1. When the counter is zero, the host cannot send data.



4.4 INTERNETWORKING

4.4.1 How Networks Differ:

Networks can differ in many ways. Some of the differences, such as different modulation techniques or frame formats, are internal to the physical and data link layers. These differences will not concern us here. Instead, in Fig. 5-38 we list some of the differences that can be exposed to the network layer. It is papering over these differences that makes internetworking more difficult than operating within a single network.

Item	Some Possibilities
Service offered	Connectionless versus connection oriented
Addressing	Different sizes, flat or hierarchical
Broadcasting	Present or absent (also multicast)
Packet size	Every network has its own maximum
Ordering	Ordered and unordered delivery
Quality of service	Present or absent; many different kinds
Reliability	Different levels of loss
Security	Privacy rules, encryption, etc.
Parameters	Different timeouts, flow specifications, etc.
Accounting	By connect time, packet, byte, or not at all

Figure 5-38. Some of the many ways networks can differ.

4.4.2 How Networks Can Be Connected

There are two basic choices for connecting different networks: we can build devices that translate or convert packets from each kind of network into packets for each other network, or, like good computer scientists, we can try to solve the problem by adding a layer of indirection and building a common layer on top of the different networks. In either case, the devices are placed at the boundaries between networks.

Let us first explore at a high level how interconnection with a common network layer can be used to interconnect dissimilar networks. An internet comprised of 802.11, MPLS, and Ethernet networks is shown in Fig. 5-39(a). Suppose that the source machine on the 802.11 network wants to send a packet to the destination machine on the Ethernet network. Since these technologies are different, and they are further separated by another kind of network (MPLS), some added processing is needed at the boundaries between the networks.

Because different networks may, in general, have different forms of addressing, the packet carries a network layer address that can identify any host across the three networks. The first boundary the packet reaches is when it transitions from an 802.11 network to an MPLS network. 802.11 provides a connectionless service, but MPLS provides a connection-oriented service. This means that a virtual circuit must be set up to cross that network. Once the packet has traveled along the virtual circuit, it will reach the Ethernet network. At this boundary, the packet may be too large to be carried, since 802.11 can work with larger frames than Ethernet. To handle this problem, the packet is divided into fragments, and each fragment is sent separately. When the fragments reach the destination, they are reassembled. Then the packet has completed its journey.

The protocol processing for this journey is shown in Fig. 5-39(b). The source accepts data from the transport layer and generates a packet with the common network layer header, which is IP in this example. The network header contains the ultimate destination address, which is used to determine that the packet should be sent via the first router. So the packet is encapsulated in an 802.11 frame whose destination is the first router and transmitted. At the router, the packet is removed from the frame's data field and the 802.11 frame header is discarded. The router now examines the IP address in the packet and looks up this address in its routing table. Based on this address, it decides to send the packet to the second router next. For this part of the path, an MPLS virtual circuit must be established to the second router and the packet must be encapsulated with MPLS headers that travel this circuit. At the far end, the MPLS header is discarded and the network address is again consulted to find the next network layer hop. It is the destination itself. Since the packet is too long to be sent over Ethernet, it is split into two portions. Each of these portions is put into the data field of an Ethernet frame and sent to the Ethernet address of the destination. At the destination, the Ethernet header is stripped from each of the frames, and the contents are reassembled. The packet has finally reached its destination.

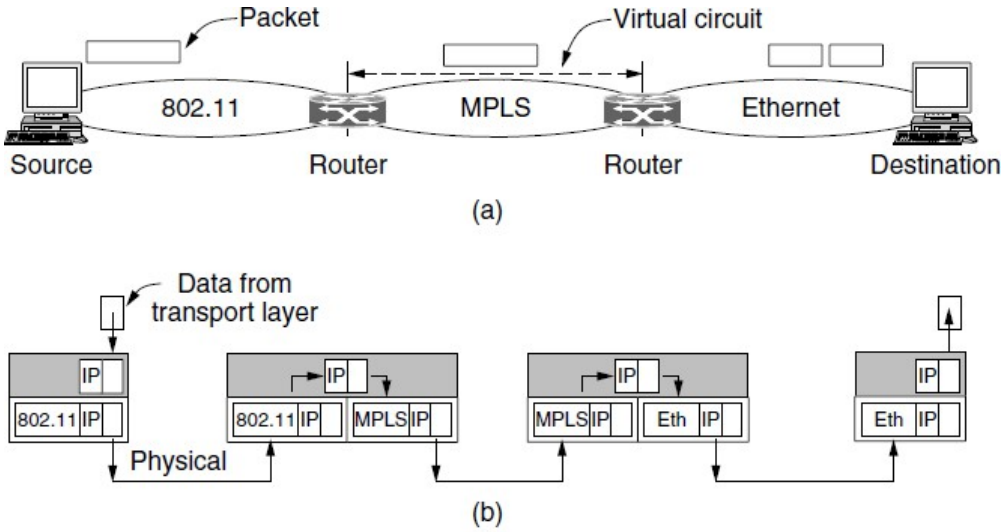


Figure 5-39. (a) A packet crossing different networks. (b) Network and link layer protocol processing.

4.4.3 Tunneling

- Handling the general case of making two different networks interwork is exceedingly

difficult. However, there is a common special case that is manageable.

- This case is where the source and destination hosts are on the same type of network, but there is a different network in between.
- As an example, think of an international bank with a TCP/IP-based Ethernet in Paris, a TCP/IP-based Ethernet in London, and a non-IP wide area network (e.g., ATM) in between, as shown in Fig. 5-47.

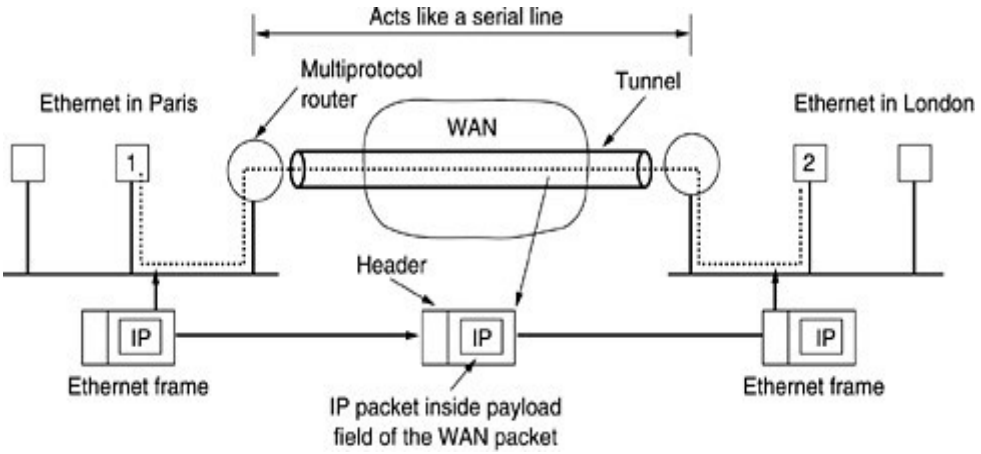


Figure 5-47. Tunneling a packet from Paris to London.

- The solution to this problem is a technique called **tunneling**.
- To send an IP packet to host 2, host 1 constructs the packet containing the IP address of host 2, inserts it into an Ethernet frame addressed to the Paris multiprotocol router, and puts it on the Ethernet. When the multiprotocol router gets the frame, it removes the IP packet, inserts it in the payload field of the WAN network layer packet, and addresses the latter to the WAN address of the London multiprotocol router. When it gets there, the London router removes the IP packet and sends it to host 2 inside an Ethernet frame.
- The WAN can be seen as a big tunnel extending from one multiprotocol router to the other. The IP packet just travels from one end of the tunnel to the other, snug in its nice box. Neither do the hosts on either Ethernet. Only the multiprotocol router has to understand IP and WAN packets. In effect, the entire distance from the middle of one multiprotocol router to the middle of the other acts like a serial line.
- Consider a person driving her car from Paris to London. Within France, the car moves under its own power, but when it hits the English Channel, it is loaded into a high-speed train and transported to England through the Chunnel (cars are not permitted to drive through the Chunnel). Effectively, the car is being carried as freight, as depicted in Fig. 5-48. At the far end, the car is let loose on the English roads and once again continues to move under its own power. Tunneling of packets through a foreign network works the same way.

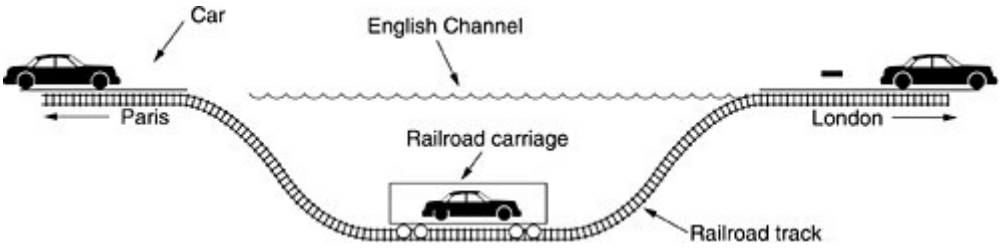


Figure 5-48. Tunneling a car from France to England.

4.4.4 Internetwork Routing

- Routing through an internetwork is similar to routing within a single subnet, but with some added complications.
- Consider, for example, the internetwork of Fig. 5-49(a) in which five networks are connected by six (possibly multiprotocol) routers. Making a graph model of this situation is complicated by the fact that every router can directly access (i.e., send packets to) every other router connected to any network to which it is connected. For example, *B* in Fig. 5-49(a) can directly access *A* and *C* via network 2 and also *D* via network 3. This leads to the graph of Fig. 5-49(b).

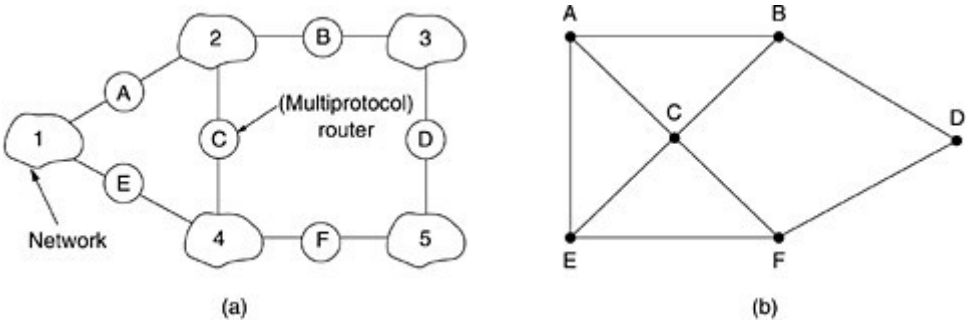


Figure 5-49. (a) An internetwork. (b) A graph of the internetwork.

- Once the graph has been constructed, known routing algorithms, such as the distance vector and link state algorithms, can be applied to the set of multiprotocol routers.
- This gives a two-level routing algorithm: within each network an **interior gateway protocol** is used, but between the networks, an **exterior gateway protocol** is used (''gateway'' is an older term for ''router'').
- Network in an internetwork is independent of all the others, it is often referred to as an **Autonomous System** (**AS**).
- A typical internet packet starts out on its LAN addressed to the local multiprotocol router (in the MAC layer header). After it gets there, the network layer code decides which multiprotocol router to forward the packet to, using its own routing tables. If that router can be reached using the packet's native network protocol, the packet is forwarded there directly. Otherwise it is tunneled there, encapsulated in the protocol required by the

intervening network. This process is repeated until the packet reaches the destination network.

- One of the differences between internetwork routing and intranet work routing is that internetwork routing may require crossing international boundaries. Various laws suddenly come into play, such as Sweden's strict privacy laws about exporting personal data about Swedish citizens from Sweden. Another example is the Canadian law saying that data traffic originating in Canada and ending in Canada may not leave the country. This law means that traffic from Windsor, Ontario to Vancouver may not be routed via nearby Detroit, even if that route is the fastest and cheapest.

Another difference between interior and exterior routing is the cost. Within a single network, a single charging algorithm normally applies. However, different networks may be under different managements, and one route may be less expensive than another. Similarly, the quality of service offered by different networks may be different, and this may be a reason to choose one route over another.

4.4.5 Fragmentation

Each network imposes some maximum size on its packets. These limits have various causes, among them:

1. Hardware (e.g., the size of an Ethernet frame).
2. Operating system (e.g., all buffers are 512 bytes).
3. Protocols (e.g., the number of bits in the packet length field).
4. Compliance with some (inter)national standard.
5. Desire to reduce error-induced retransmissions to some level.
6. Desire to prevent one packet from occupying the channel too long.

- From the above factors maximum payloads range from 48 bytes (ATM cells) to 65,515 bytes (IP packets), although the payload size in higher layers is often larger.
- If the original source packet is too large to be handled by the destination network? The routing algorithm can hardly bypass the destination.
- The only solution to the problem is to allow gateways to break up packets into **fragments**, sending each fragment as a separate internet packet. Packet-switching networks, too, have trouble putting the fragments back together again.

Two opposing strategies exist for recombining the fragments back into the original packet.

The **first strategy** is to make fragmentation caused by a "small-packet" network transparent to any subsequent networks through which the packet must pass on its way to the ultimate destination. This option is shown in Fig. 5-50(a). In this approach, the small-packet network has gateways (most likely, specialized routers) that interface to other networks. When an oversized packet arrives at a gateway, the gateway breaks it up into fragments. Each fragment is addressed to the same exit gateway, where the pieces are recombined. In this way passage

through the small-packet network has been made transparent. Subsequent networks are not even aware that fragmentation has occurred.

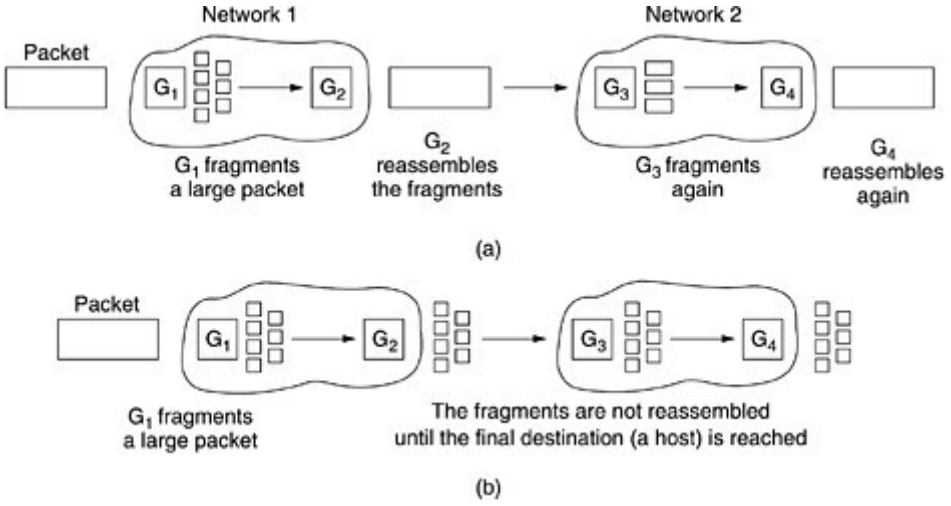


Figure 5-50. (a) Transparent fragmentation. (b) Nontransparent fragmentation.

- Transparent fragmentation is straightforward but has some problems.
- For one thing, the exit gateway must know when it has received all the pieces, so either a count field or an ''end of packet'' bit must be provided.
- For another thing, all packets must exit via the same gateway. By not allowing some fragments to follow one route to the ultimate destination and other fragments a disjoint route, some performance may be lost.
- A last problem is the overhead required to repeatedly reassemble and then refragment a large packet passing through a series of small-packet networks. ATM requires transparent fragmentation.

The **other fragmentation strategy** is to refrain from recombining fragments at any intermediate gateways. Once a packet has been fragmented, each fragment is treated as though it were an original packet. All fragments are passed through the exit gateway (or gateways), as shown in Fig. 5-50(b). Recombination occurs only at the destination host. IP works this way.

- Nontransparent fragmentation also has some problems.
- For example, it requires *every* host to be able to do reassembly.
- Yet another problem is that when a large packet is fragmented, the total overhead increases because each fragment must have a header. Whereas in the first method this overhead disappears as soon as the small-packet network is exited, in this method the overhead remains for the rest of the journey.
- An advantage of nontransparent fragmentation, however, is that multiple exit gateways can now be used and higher performance can be achieved.
- When a packet is fragmented, the fragments must be numbered in such a way that the original data stream can be reconstructed.

- One way of numbering the fragments is to use a tree. If packet 0 must be split up, the pieces are called 0.0, 0.1, 0.2, etc. If these fragments themselves must be fragmented later on, the pieces are numbered 0.0.0, 0.0.1, 0.0.2, . . . , 0.1.0, 0.1.1, 0.1.2, etc.
- No duplicates are generated anywhere, this scheme is sufficient to ensure that all the pieces can be correctly reassembled at the destination, no matter what order they arrive in.
- However, if even one network loses or discards packets, end-to-end retransmissions are needed, with unfortunate effects for the numbering system.
- Suppose that a 1024-bit packet is initially fragmented into four equal-sized fragments, 0.0, 0.1, 0.2, and 0.3. Fragment 0.1 is lost, but the other parts arrive at the destination. Eventually, the source time out and retransmits the original packet again.
- When a packet is fragmented, all the pieces are equal to the elementary fragment size except the last one, which may be shorter. An internet packet may contain several fragments, for efficiency reasons.
- The internet header must provide the original packet number and the number of the (first) elementary fragment contained in the packet.
- A bit indicating that the last elementary fragment contained within the internet packet is the last one of the original packet.

This approach requires two sequence fields in the internet header:

- The original packet number and the fragment number. There is clearly a trade-off between the size of the elementary fragment and the number of bits in the fragment number. Because the elementary fragment size is presumed to be acceptable to every network, subsequent fragmentation of an internet packet containing several fragments causes no problem. The ultimate limit here is to have the elementary fragment be a single bit or byte, with the fragment number then being the bit or byte offset within the original packet, as shown in Fig. 5-51.

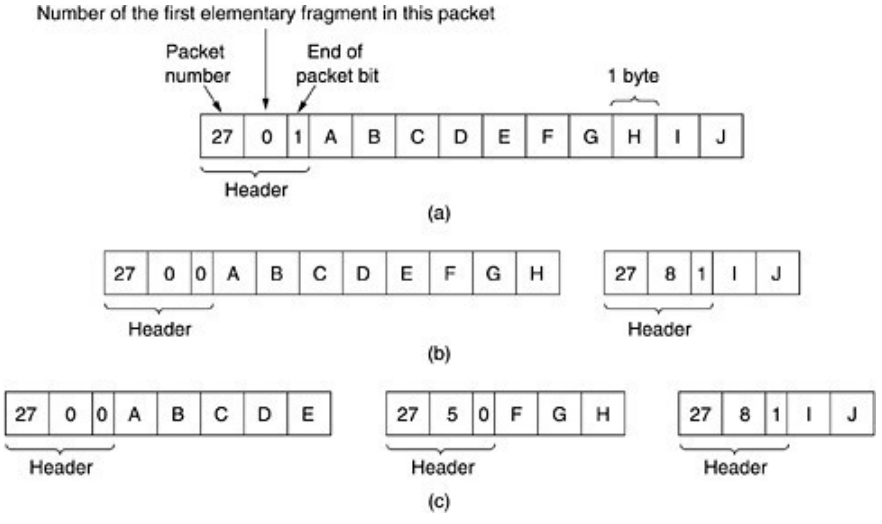


Figure 5-51. Fragmentation when the elementary data size is 1 byte. (a) Original packet,

containing 10 data bytes. (b) Fragments after passing through a network with maximum packet size of 8 payload bytes plus header. (c) Fragments after passing through a size 5 gateway.

Some internet protocols take this method even further and consider the entire transmission on a virtual circuit to be one giant packet, so that each fragment contains the absolute byte number of the first byte within the fragment.

4.4.6 The Network Layer in The Internet

Top 10 principles

1. **Make sure it works.** Do not finalize the design or standard until multiple prototypes have successfully communicated with each other. All too often designers first write a

   1000-page standard, get it approved, then discover it is deeply flawed and does not work. Then they write version 1.1 of the standard. This is not the way to go.
2. **Keep it simple.** When in doubt, use the simplest solution. William of Occam stated this principle (Occam's razor) in the 14th century. Put in modern terms: fight features. If a feature is not absolutely essential, leave it out, especially if the same effect can be achieved by combining other features.
3. **Make clear choices.** If there are several ways of doing the same thing, choose one. Having two or more ways to do the same thing is looking for trouble. Standards often have multiple options or modes or parameters because several powerful parties insist that their way is best. Designers should strongly resist this tendency. Just say no.
4. **Exploit modularity.** This principle leads directly to the idea of having protocol stacks, each of whose layers is independent of all the other ones. In this way, if circumstances that require one module or layer to be changed, the other ones will not be affected.
5. **Expect heterogeneity.** Different types of hardware, transmission facilities, and applications will occur on any large network. To handle them, the network design must be simple, general, and flexible.
6. **Avoid static options and parameters.** If parameters are unavoidable (e.g., maximum packet size), it is best to have the sender and receiver negotiate a value than defining fixed choices.
7. **Look for a good design; it need not be perfect.** Often the designers have a good design but it cannot handle some weird special case. Rather than messing up the design, the designers should go with the good design and put the burden of working around it on the people with the strange requirements.
8. **Be strict when sending and tolerant when receiving.** In other words, only send packets that rigorously comply with the standards, but expect incoming packets that may not be fully conformant and try to deal with them.
9. **Think about scalability.** If the system is to handle millions of hosts and billions of users

effectively, no centralized databases of any kind are tolerable and load must be spread as evenly as possible over the available resources.

10. **Consider performance and cost.** If a network has poor performance or outrageous costs, nobody will use it.

- At the network layer, the Internet can be viewed as a collection of subnetworks or **Autonomous Systems** (**ASes**) that are interconnected.
- There is no real structure, but several major backbones exist. These are constructed from high-bandwidth lines and fast routers. Attached to the backbones are regional (midlevel) networks, and attached to these regional networks are the LANs at many universities, companies, and Internet service providers.
- A sketch of this quasi-hierarchical organization is given in Fig. 5-52.

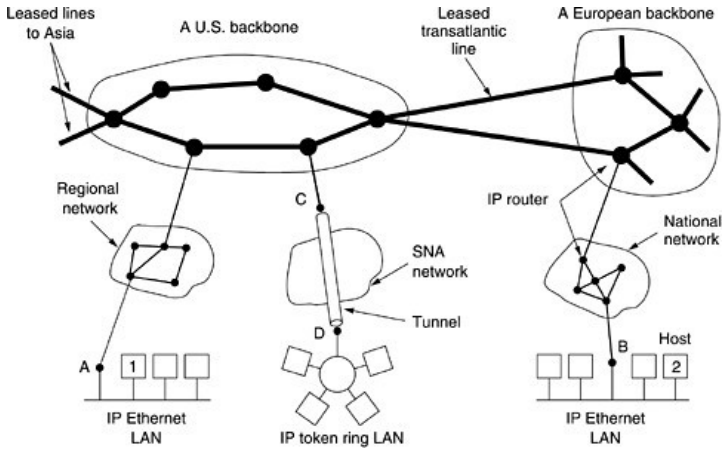
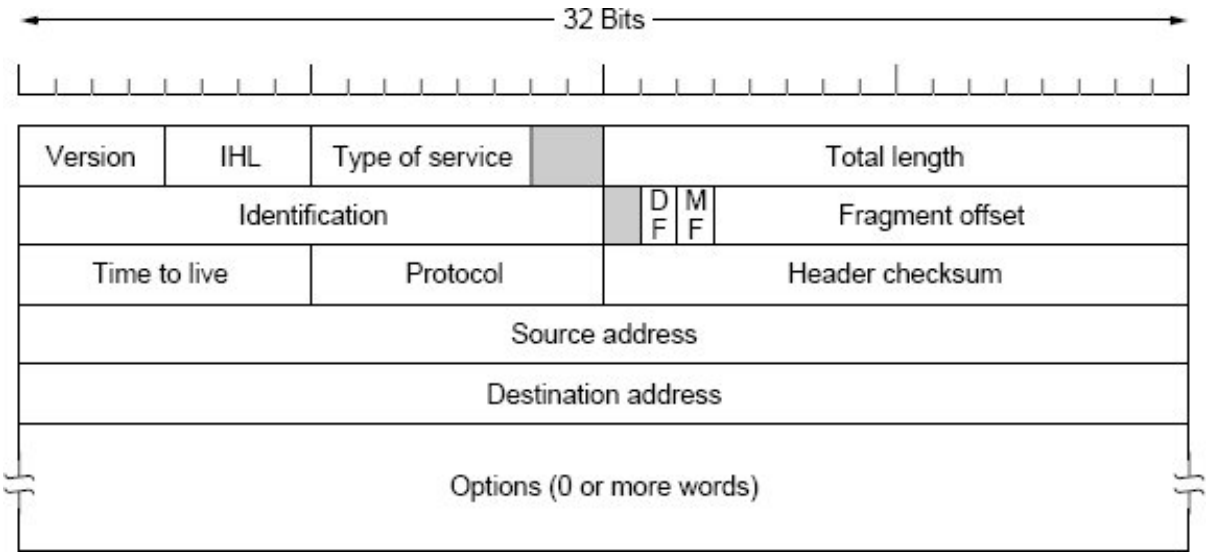


Figure 5-52. The Internet is an interconnected collection of many networks.

- The glue that holds the whole Internet together is the network layer protocol, **IP** (**Internet Protocol**). Unlike most older network layer protocols, it was designed from the beginning with internetworking in mind.
- The network layer job is to provide a best-efforts (i.e., not guaranteed) way to transport datagram from source to destination, without regard to whether these machines are on the same network or whether there are other networks in between them.
- The transport layer takes data streams and breaks them up into datagram's.
- Each datagram is transmitted through the Internet, possibly being fragmented into smaller units as it goes.
- When all the pieces finally get to the destination machine, they are reassembled by the network layer into the original datagram.
- This datagram is then handed to the transport layer, which inserts it into the receiving process' input stream.

4.4.7 The IP Protocol

- An IP datagram consists of a header part and a text part. The header has a 20-byte fixed part and a variable length optional part.

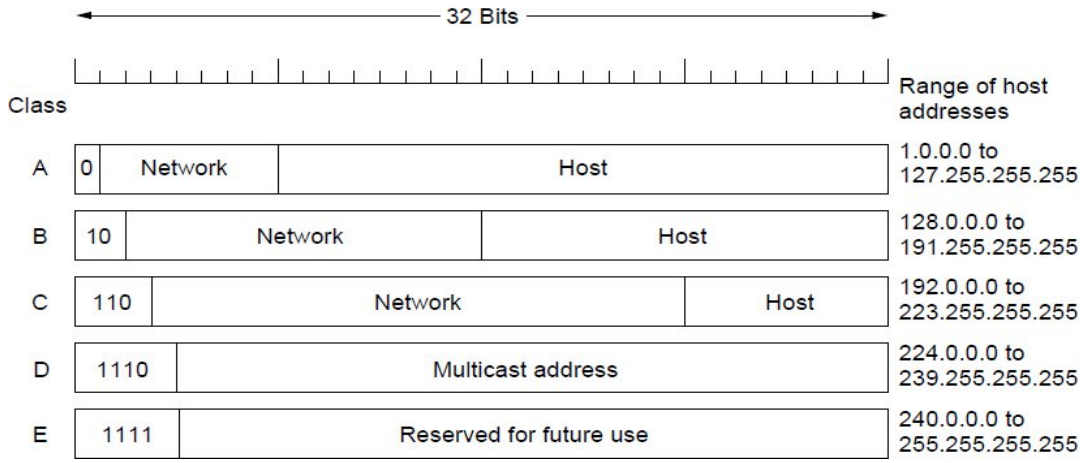


- The ***Version*** field keeps track of which version of the protocol the datagram belongs to.
- The header length is not constant, a field in the header, ***IHL***, is provided to tell how long the header is, in 32-bit words.
- The ***Type of service*** field is one of the few fields that have changed its meaning (slightly) over the years. It was and is still intended to distinguish between different classes of service.

- The ***Total length*** includes everything in the datagram—both header and data. The maximum length is 65,535 bytes.
- The ***Identification*** field is needed to allow the destination host to determine which datagram a newly arrived fragment belongs to. All the fragments of a datagram contain the same *Identification* value. Next comes an unused bit and then two 1-bit fields.
- ***DF*** stands for Don't Fragment. It is an order to the routers not to fragment the datagram because the destination is incapable of putting the pieces back together again.
- ***MF*** stands for More Fragments. All fragments except the last one have this bit set. It is needed to know when all fragments of a datagram have arrived.
- The ***Fragment offset*** tells where in the current datagram this fragment belongs. All fragments except the last one in a datagram must be a multiple of 8 bytes, the elementary fragment unit. Since 13 bits are provided, there is a maximum of 8192 fragments per datagram, giving a maximum datagram length of 65,536 bytes, one more than the ***Total length*** **field.**
- The ***Time to live*** field is a counter used to limit packet lifetimes. It is supposed to count time in seconds, allowing a maximum lifetime of 255 sec.
- When the network layer has assembled a complete datagram, it needs to know what to do with it. The ***Protocol*** field tells it which transport process to give it to. TCP is one possibility, but so are UDP and some others. The numbering of protocols is global across the entire Internet.
- The ***Header checksum*** verifies the header only.
- The ***Source address*** **and** ***Destination address*** indicate the network number and host number.
- The ***Options*** field was designed to provide an escape to allow subsequent versions of the protocol to include information not present in the original design, to permit experimenters to try out new ideas, and to avoid allocating header bits to information that is rarely needed.
- The options are variable length.

4.4.8 IP Address

IP addresses were divided into the five categories listed in Fig. This allocation has come to be called classful addressing. It is no longer used, but references to it in the literature are still common.



The class A, B, C, and D formats allow for up to 128 networks with 16 million hosts each, 16,384 networks with up to 64K hosts, and 2 million networks (e.g., LANs) with up to 256 hosts each (although a few of these are special). Also supported is multicast, in which a datagram is directed to multiple hosts. Addresses beginning with 1111 are reserved for future use.

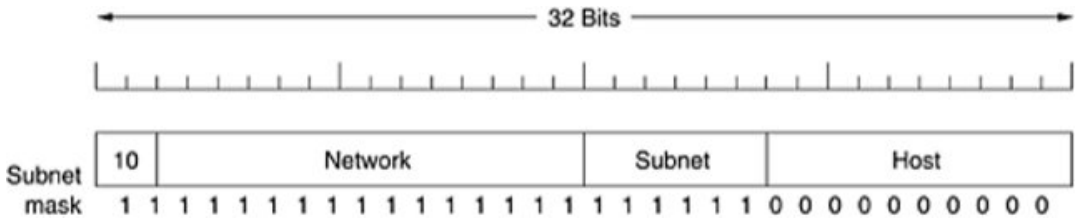
Network addresses, which are 32-bit numbers, are usually written in dotted decimal notation. In this format, each of the 4 bytes is written in decimal, from 0 to 255. For example, the 32-bit hexadecimal address C0290614 is written as 192.41.6.20. The lowest IP address is 0.0.0.0 and the highest is 255.255.255.255.

Subnets:

As we have seen, all the hosts in a network must have the same network number. This property of IP addressing can cause problems as networks grow. The solution is to allow a network to be split into several parts for internal use but still act like a single network to the outside world.

In the Internet literature, the parts of the network (in this case, Ethernets) are called **subnets.**

To implement subnetting, the main router needs a **subnet mask** that indicates the split between network + subnet number and host, as shown in Fig. Subnet masks are also written in dotted decimal notation, with the addition of a slash followed by the number of bits in the network + subnet part. For the example of Fig. the subnet mask can be written as 255.255.252.0. An alternative notation is /22 to indicate that the subnet mask is 22 bits long.



CIDR—Classless Inter Domain Routing:

The basic idea behind CIDR, which is described in RFC 1519, is to allocate the remaining IP addresses in **variable-sized blocks**, without regard to the classes. If a site needs, say, 2000 addresses, it is given a block of 2048 addresses on a 2048-byte boundary.

Using CIDR, each IP address has a **network prefix** that identifies either one or several network gateways. The length of the network prefix in IPv4 CIDR is also specified as part of

the IP address and varies depending on the number of bits needed, rather than any arbitrary class assignment structure. A destination IP address or route that describes many possible destinations has a shorter prefix and is said to be less specific. A longer prefix describes a destination gateway more specifically.

CIDR notation: **a.b.c.d/n**

Where n = number of network IDs

Host ID = 32-n

IP = $2^{(32-n)}$

Example: 20.10.20.100/20

Network ID = 20.

Host ID = 32-20 = 12.

IP = $2^{(32-20)}$ = 4096.

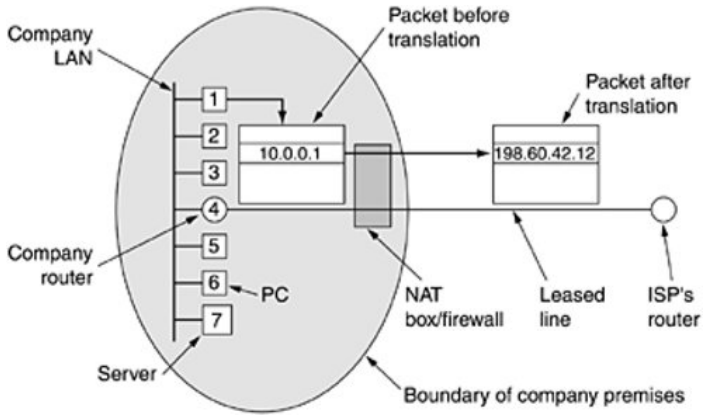
NAT—Network Address Translation:

IP addresses are scarce. An ISP might have a /16 (formerly class B) address, giving it 65,534 host numbers. If it has more customers than that, it has a problem. This quick fix came in the form of **NAT (Network Address Translation),** which is described in RFC 3022 and which we will summarize below.

The basic idea behind NAT is to assign each company a single IP address (or at most, a small number of them) for Internet traffic. Within the company, every computer gets a unique IP address, which is used for routing intramural traffic. However, when a packet exits the company and goes to the ISP, an address translation takes place. To make this scheme possible, three ranges of IP addresses have been declared as private.

10.0.0.0 – 10.255.255.255/8 (16,777,216 hosts)
172.16.0.0 – 172.31.255.255/12 (1,048,576 hosts)
192.168.0.0 – 192.168.255.255/16 (65,536 hosts)

The operation of NAT is shown in Fig. Within the company premises, every machine has a unique address of the form 10.x.y.z. However, when a packet leaves the company premises, it passes through a NAT box that converts the internal IP source address, 10.0.0.1 in the figure, to the company's true IP address, 198.60.42.12 in this example.



4.4.9 IP Version 6

Internet Protocol version 6 is a new addressing protocol designed to incorporate all the possible requirements of future Internet. This protocol as its predecessor IPv4, works on the Network Layer (Layer-3). Along with its offering of an enormous amount of logical address space, this protocol has ample features to address the shortcoming of IPv4.

The major goals of IPV6 are:

1. Support billions of hosts, even with inefficient address allocation.

2. Reduce the size of the routing tables.
3. Simplify the protocol, to allow routers to process packets faster.
4. Provide better security (authentication and privacy).
5. Pay more attention to the type of service, particularly for real-time data.
6. Aid multicasting by allowing scopes to be specified.
7. Make it possible for a host to roam without changing its address.
8. Allow the protocol to evolve in the future.
9. Permit the old and new protocols to coexist for years.

First and foremost, IPv6 has longer addresses than IPv4. They are 128 bits long, which solves the problem that IPv6 set out to solve: providing an effectively unlimited supply of Internet addresses.

The second major improvement of IPv6 is the simplification of the header. It contains only seven fields (versus 13 in IPv4). This change allows routers to process packets faster and thus improves throughput and delay.

The third major improvement is better support for options. This change was essential with the new header because fields that previously were required are now optional (because they are not used so often). In addition, the way options are represented is different, making it simple for routers to skip over options not intended for them. This feature speeds up packet processing time.

A fourth area in which IPv6 represents a big advance is in security. Authentication and privacy are key features of the new IP.

The Main IPv6 Header

The IPv6 header is shown in Fig. 5-56. The Version field is always 6 for IPv6 (and 4 for IPv4). During the transition period from IPv4, which has already taken more than a decade, routers will be able to examine this field to tell what kind of acket they have.

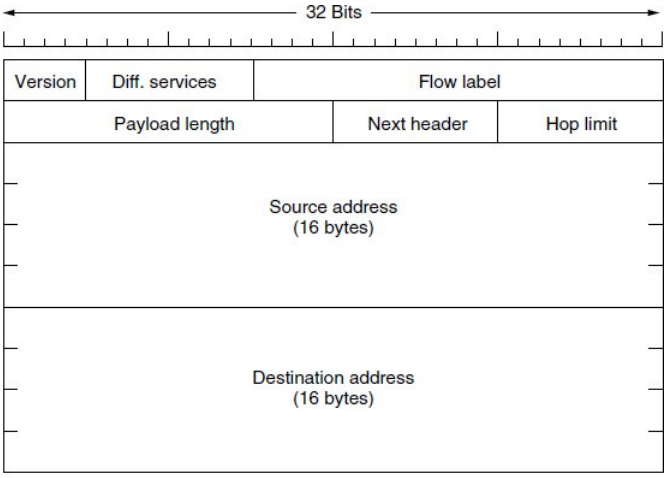


Figure 5-56. The IPv6 fixed header (required).

The *Differentiated services* field (originally called *Traffic class*) is used to distinguish the class of service for packets with different real-time delivery requirements. It is used with the differentiated service architecture for quality of service in the same manner as the field of the same name in the IPv4 packet.

The *Flow label* field provides a way for a source and destination to mark groups of packets that have the same requirements and should be treated in the same way by the network, forming a pseudo connection. The flow can be set up in advance and given an identifier.
The *Payload length* field tells how many bytes follow the 40-byte header of Fig. 5-56. The name was changed from the IPv4 *Total length* field because the meaning was changed slightly: the 40 header bytes are no longer counted as part of the length (as they used to be). This change means the payload can now be 65,535 bytes instead of a mere 65,515 bytes.
The *Next header* field lets the cat out of the bag. The reason the header could be simplified is that there can be additional (optional) extension headers. This field tells which of the (currently) six extension headers, if any, follow this one.
The *Hop limit* field is used to keep packets from living forever. It is, in practice, the same as the *Time to live* field in IPv4, namely, a field that is decremented on each hop.
Next the *Source address* and *Destination address* fields which contains 1228 bit address.
A new notation has been devised for writing 16-byte addresses. They are written as eight groups of four hexadecimal digits with colons between the groups, like this:

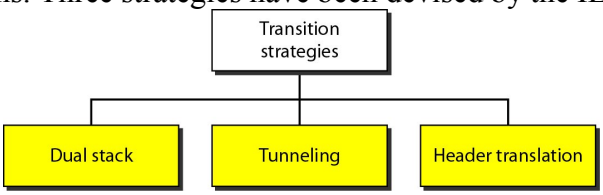
8000:0000:0000:0000:0123:4567:89AB:CDEF

Since many addresses will have many zeros inside them, three optimizations have been authorized. First, leading zeros within a group can be omitted, so 0123 can be written as 123. Second, one or more groups of 16 zero bits can be replaced by a pair of colons. Thus, the above address now becomes

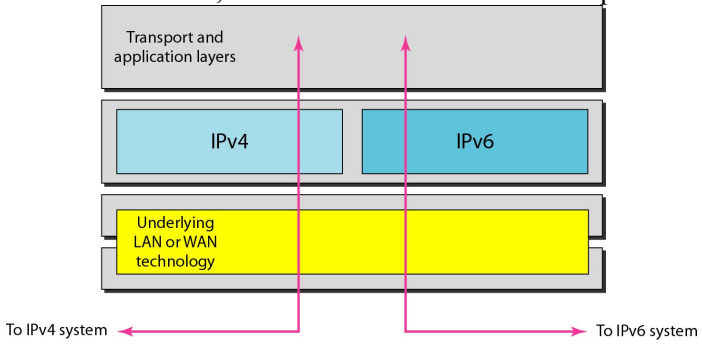
8000::123:4567:89AB:CDEF

Transition from IPV4 to IPV6:

Because of the huge number of systems on the Internet, the transition from IPv4 to IPv6 cannot happen suddenly. It takes a considerable amount of time before every system in the Internet can move from IPv4 to IPv6. The transition must be smooth to prevent any problems between IPv4 and IPv6 systems. Three strategies have been devised by the IETF to help the transition.

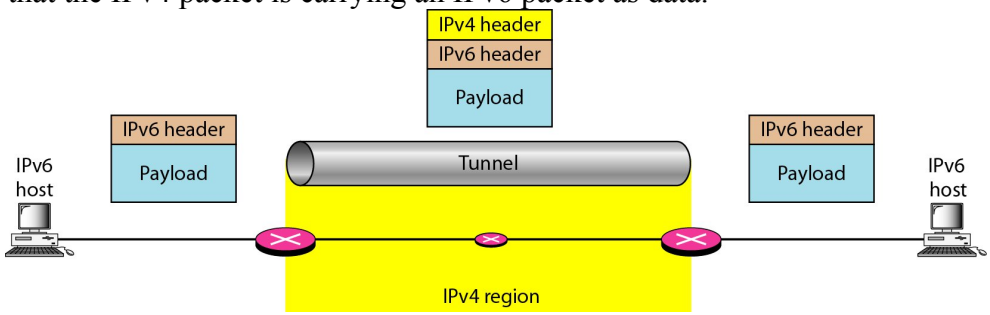


1. **Dual Stack**
   It is recommended that all hosts, before migrating completely to version 6, have a dual stack of protocols. In other words, a station must run IPv4 and IPv6 simultaneously until all the Internet uses IPv6. See below Figure for the layout of a dual-stack configuration.
   To determine which version to use when sending a packet to a destination, the source host queries the DNS. If the DNS returns an IPv4 address, the source host sends an IPv4 packet. If the DNS returns an IPv6 address, the source host sends an IPv6 packet.



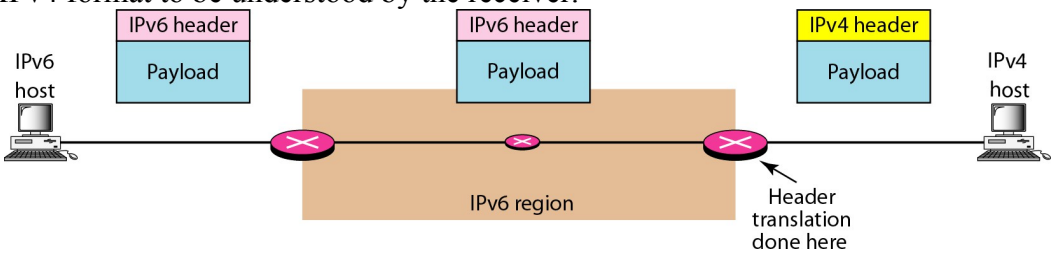
2. **Tunneling**

Tunneling is a strategy used when two computers using IPv6 want to communicate with each other and the packet must pass through a region that uses IPv4. To pass through this region, the packet must have an IPv4 address. So the IPv6 packet is encapsulated in an IPv4 packet when it enters the region, and it leaves its capsule when it exits the region. It seems as if the IPv6 packet goes through a tunnel at one end and emerges at the other end. To make it clear that the IPv4 packet is carrying an IPv6 packet as data.



3. **Header Translation**

Header translation is necessary when the majority of the Internet has moved to IPv6 but some systems still use IPv4. The sender wants to use IPv6, but the receiver does not understand IPv6. Tunneling does not work in this situation because the packet must be in the IPv4 format to be understood by the receiver.



Comparison of IPV4 & IPV6

IPv4 Address	IPv6 Address
Address Length – 32 bits	128 bits
Address Representation - decimal	hexadecimal
Internet address classes	Not applicable in IPv6
Multicast addresses (224.0.0.0/4)	IPv6 multicast addresses (FF00::/8)
Broadcast addresses	Not applicable in IPv6
Unspecified address is 0.0.0.0	Unspecified address is ::
Loopback address is 127.0.0.1	Loopback address is ::1
Public IP addresses	Global unicast addresses
Private IP addresses (10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16)	Site-local addresses (FEC0::/10)
Autoconfigured addresses (169.254.0.0/16)	Link-local addresses (FE80::/64)

4.4.10 Internet Control Protocols:

In addition to IP, which is used for data transfer, the Internet has several companion control protocols that are used in the network layer. They include ICMP, ARP, and DHCP.

3. **IMCP—The Internet Control Message Protocol**

The operation of the Internet is monitored closely by the routers. When something unexpected occurs during packet processing at a router, the event is reported to the sender by the **ICMP** (**Internet Control Message Protocol**). ICMP is also used to test the Internet. About a dozen types of ICMP messages are defined. Each ICMP message type is carried encapsulated in an IP packet. The most important ones are listed in Fig. 5-60.

Message type	Description
Destination unreachable	Packet could not be delivered
Time exceeded	Time to live field hit 0
Parameter problem	Invalid header field
Source quench	Choke packet
Redirect	Teach a router about geography
Echo and echo reply	Check if a machine is alive
Timestamp request/reply	Same as Echo, but with timestamp
Router advertisement/solicitation	Find a nearby router

Figure 5-60. The principal ICMP message types.

The DESTINATION UNREACHABLE message is used when the router cannot locate the destination or when a packet with the *DF* bit cannot be delivered because a ‘‘small-packet’’ network stands in the way.

The TIME EXCEEDED message is sent when a packet is dropped because it’s *TTL (Time to live)* counter has reached zero. This event is a symptom that packets are looping, or that the counter values are being set too low.

The PARAMETER PROBLEM message indicates that an illegal value has been detected in a header field. This problem indicates a bug in the sending host’s IP software or possibly in the software of a router transited.

The SOURCE QUENCH message was long ago used to throttle hosts that were sending too many packets. When a host received this message, it was expected to slow down. It is rarely used anymore because when congestion occurs, these packets tend to add more fuel to the fire and it is unclear how to respond to them.

The REDIRECT message is used when a router notices that a packet seems to be routed incorrectly. It is used by the router to tell the sending host to update to a better route.

The ECHO and ECHO REPLY messages are sent by hosts to see if a given destination is reachable and currently alive. Upon receiving the ECHO message, the destination is expected to send back an ECHO REPLY message. These messages are used in the **ping** utility that checks if a host is up and on the Internet.

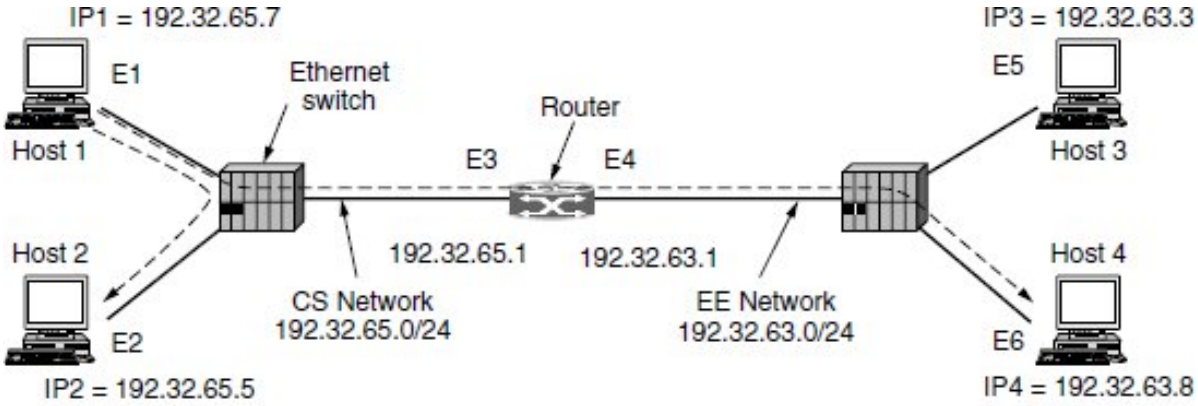
The TIMESTAMP REQUEST and TIMESTAMP REPLY messages are similar, except that the arrival time of the message and the departure time of the reply are recorded in the reply. This facility can be used to measure network performance.

The ROUTER ADVERTISEMENT and ROUTER SOLICITATION messages are used to let hosts find nearby routers. A host needs to learn the IP address of at least one router to be able to send packets off the local network.

4. **ARP—The Address Resolution Protocol**

Although every machine on the Internet has one or more IP addresses, these addresses are not sufficient for sending packets. Data link layer NICs (Network Interface Cards) such as Ethernet cards do not understand Internet addresses. In the case of Ethernet, every NIC ever manufactured comes equipped with a unique 48-bit Ethernet address. Manufacturers of Ethernet NICs request a block of Ethernet addresses from IEEE to ensure that no two NICs have the same address. The NICs send and receive frames based on 48-bit Ethernet addresses. They know nothing at all about 32-bit IP addresses.

The question now arises, how do IP addresses get mapped onto data link layer addresses, such as Ethernet? To explain how this works, let us use the example of Fig. 5-61, in which a small university with two /24 networks is illustrated. One network (CS) is a switched Ethernet in the Computer Science Dept. It has the prefix 192.32.65.0/24. The other LAN (EE), also switched Ethernet, is in Electrical Engineering and has the prefix 192.32.63.0/24. The two LANs are connected by an IP router. Each machine on an Ethernet and each interface on the router has a unique Ethernet address, labeled *E1* through *E6*, and a unique IP address on the CS or EE network.



Frame	Source IP	Source Eth.	Destination IP	Destination Eth.
Host 1 to 2, on CS net	IP1	E1	IP2	E2
Host 1 to 4, on CS net	IP1	E1	IP4	E3
Host 1 to 4, on EE net	IP1	E4	IP4	E6

Figure 5-61. Two switched Ethernet LANs joined by a router.

Let us start out by seeing how a user on host 1 sends a packet to a user on host 2 on the CS network. Let us assume the sender knows the name of the intended receiver, possibly something like *eagle.cs.uni.edu*. The first step is to find the IP address for host 2. This lookup is performed by DNS.

. The upper layer software on host 1 now builds a packet with 192.32.65.5 in the *Destination address* field and gives it to the IP software to transmit. The IP software can look at the address and see that the destination is on the CS network, (i.e., its own network). However, it still needs some way to find the destination's Ethernet address to send the frame. One solution is to have a configuration file somewhere in the system that maps IP addresses onto Ethernet addresses. While this solution is certainly possible, for organizations with thousands of machines keeping all these files up to date is an error-prone, time-consuming job.

A better solution is for host 1 to output a broadcast packet onto the Ethernet asking who owns IP address 192.32.65.5. The broadcast will arrive at every machine on the CS Ethernet,

and each one will check its IP address. Host 2 alone will respond with its Ethernet address (*E2*). In this way host 1 learns that IP address 192.32.65.5 is on the host with Ethernet address *E2*. The protocol used for asking this question and getting the reply is called **ARP** (**Address Resolution Protocol**).

At this point, the IP software on host 1 builds an Ethernet frame addressed to E2, puts the IP packet (addressed to 192.32.65.5) in the payload field, and dumps it onto the Ethernet. The IP and Ethernet addresses of this packet are given in Fig. 5-61.

Now let us look at Fig. 5-61 again, only this time assume that host 1 wants to send a packet to host 4 (192.32.63.8) on the EE network. Host 1 will see that the destination IP address is not on the CS network. It knows to send all such off-network traffic to the router, which is also known as the **default gateway**. By convention, the default gateway is the lowest address on the network (198.31.65.1). To send a frame to the router, host 1 must still know the Ethernet address of the router interface on the CS network. It discovers this by sending an ARP broadcast for 198.31.65.1, from which it learns *E3*. It then sends the frame. The same lookup mechanisms are used to send a packet from one router to the next over a sequence of routers in an Internet path.

When the Ethernet NIC of the router gets this frame, it gives the packet to the IP software. It knows from the network masks that the packet should be sent onto the EE network where it will reach host 4. If the router does not know the Ethernet address for host 4, then it will use ARP again. The table in Fig. 5-61 lists the source and destination Ethernet and IP addresses that are present in the frames as observed on the CS and EE networks. Observe that the Ethernet addresses change with the frame on each network while the IP addresses remain constant.

It is also possible to send a packet from host 1 to host 4 without host 1 knowing that host 4 is on a different network. The solution is to have the router answer ARPs on the CS network for host 4 and give its Ethernet address, *E3*, as the response. It is not possible to have host 4 reply directly because it will not see the ARP request (as routers do not forward Ethernet-level broadcasts). The router will then receive frames sent to 192.32.63.8 and forward them onto the EE network. This solution is called **proxy ARP**.

5. **DHCP—The Dynamic Host Configuration Protocol**

With DHCP, every network must have a DHCP server that is responsible for configuration. When a computer is started, it has a built-in Ethernet or other link layer address embedded in the NIC, but no IP address. Much like ARP, the computer broadcasts a request for an IP address on its network. It does this by using a DHCP DISCOVER packet. This packet must reach the DHCP server. If that server is not directly attached to the network, the router will be configured to receive DHCP broadcasts and relay them to the DHCP server, wherever it is located.

When the server receives the request, it allocates a free IP address and sends it to the host in a DHCP OFFER packet (which again may be relayed via the router). To be able to do this work even when hosts do not have IP addresses, the server identifies a host using its Ethernet address (which is carried in the DHCP DISCOVER packet)

An issue that arises with automatic assignment of IP addresses from a pool is for how long an IP address should be allocated. If a host leaves the network and does not return its IP address to the DHCP server, that address will be permanently lost. After a period of time, many addresses may be lost. To prevent that from happening, IP address assignment may be for a fixed period of time, a technique called **leasing**. Just before the lease expires, the host must ask for a DHCP renewal. If it fails to make a request or the request is denied, the host may no longer use the IP address it was given earlier.