COMPUTER NETWORKS
UNIT III

Circuit switching vs. packet switching / Packet switched networks – IP – ARP – RARP – DHCP – ICMP – Queueing discipline – Routing algorithms – RIP – OSPF – Subnetting – CIDR – Interdomain routing – BGP – Ipv6 – Multicasting – Congestion avoidance in network layer

CIRCUIT SWITCHING AND PACKET SWITCHING

PACKET SWITCHING

- A packet switch is a device with several inputs and outputs leading to and from the hosts that the switch interconnects.

- The core job of a switch is to take packets that arrive on an input and *forward (or switch) them to the right output so that they* will reach their appropriate destination.

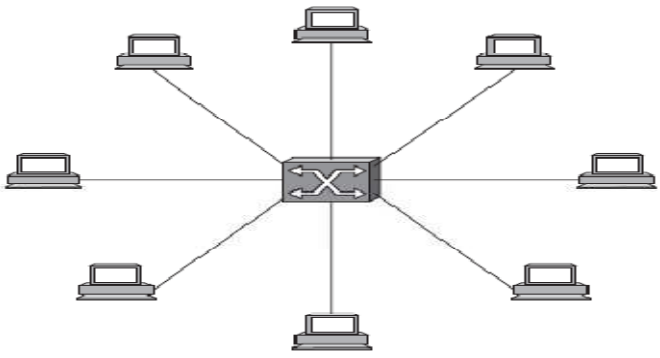
- There are a variety of ways that the switch can determine the ―right‖ output for a packet, which can be broadly categorized as connectionless and connection-oriented approachesA switch is a multi-input, multi-output device, which transfers packets from an input to one or more outputs. Thus, a switch adds the star topology to the point-to-point link, bus (Ethernet), and ring (802.5 and FDDI) topologies.

- A star topology has several attractive properties:
- Even though a switch has a fixed number of inputs and outputs, which limits the number of hosts that can be connected to a single switch, large networks can be built by interconnecting a number of switches.

- We can connect switches to each other and to hosts using point-to-point links, which typically means that we can build networks of large geographic scope.

- Adding a new host to the network by connecting it to a switch does not necessarily mean that the hosts already connected will get worse performance from the network.

A switch provides a star topology

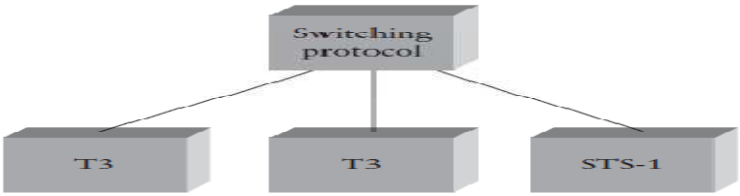


- Switched networks are considered more *scalable.*

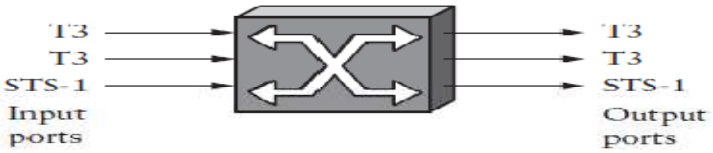
- A switch's primary job is to receive incoming packets on one of its links and to transmit them on some other link.

- This function is sometimes referred to as either *switching or forwarding, and in terms of the OSI architecture, it is the main function* of the network layer.

Example protocol graph running on a switch.



Shows the protocol graph that would run on a switch that is connected to two T3 links and one STS-1 SONET link.



In this figure, we have split the input and output halves of each link, and we refer to each input or output as a *port.* In other words, this example switch has three input ports and three output ports.

TWO COMMON APPROCHES FOR SWITCHING AND FORWADING

- The first is the *datagram or connectionless approach. The second is the virtual circuit or connection-oriented approach.*

- A third approach, *source routing, is less common than these other two, but it* is simple to explain and does have some useful applications.

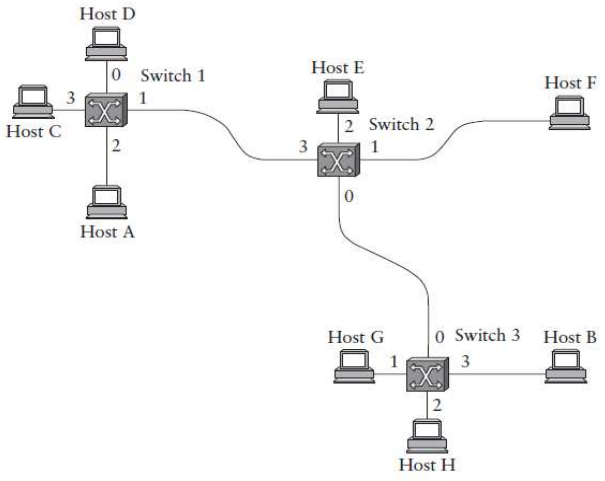
All host must have *globally unique identifier.*

Datagrams

- You just make sure that every packet contains enough information to enable any switch to decide how to get it to its destination. That is, every packet contains the complete destination address.

- To decide how to forward a packet, a switch consults a *forwarding table (sometimes called a routing table).*

Datagram forwarding: an example network



Forwarding table for switch 2.

Destination	Port
A	3
B	0
C	3
D	3
E	2
F	1
G	0
H	0

ROUTING THROUGH DATAGRAM

- This particular table shows the forwarding information that switch 2 needs to forward datagram's in the example network.

- It is pretty easy to figure out such a table when you have a complete map of a simple network like that depicted here; we could imagine a network operator configuring the tables statically.

- It is a lot harder to create the forwarding tables in large, complex networks with dynamically changing topologies and multiple paths between destinations.

- That harder problem is known as *routing.*

Characteristics Of The Connectionless(datagram) Networks

- A host can send a packet anywhere at any time, since any packet that turns up at a switch can be immediately forwarded.

- When a host sends a packet, it has no way of knowing if the network is capable of delivering it or if the destination host is even up and running.
- Each packet is forwarded independently of previous packets that might have been sent to the same destination. Thus, two successive packets from host A to host B may follow completely different paths.

- A switch or link failure might not have any serious effect on communication if it is possible to find an alternate route around the failure and to update the forwarding table accordingly.

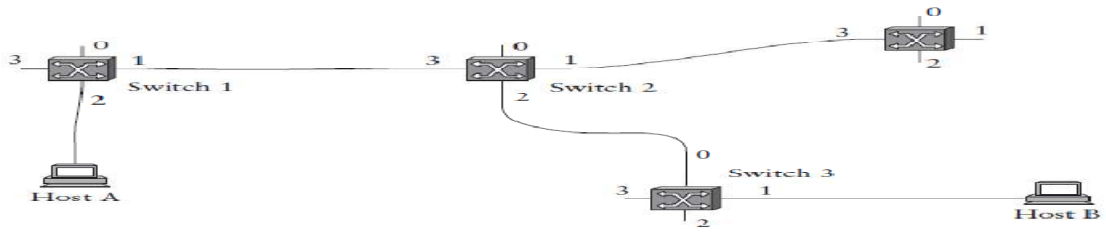
Virtual Circuit Switching

- It uses the concept of a *virtual circuit.*

- which is also called a connection-oriented model,

It requires that we first set up a virtual connection from the source host to the destination host before any data is sent.

An example of a virtual circuit network



- In this example where host A again wants to send packets to host B.

- We can think of this as a two-stage process.

- ❖ The first stage is ―connection setup.‖
- ❖ The second is data transfer.

- In the connection setup phase, it is necessary to establish ―connection state‖ in each of the switches between the source and destination hosts.

- The connection state for a single connection consists of an entry in a ―VC table‖ in each switch through which the connection passes.

VIRTUAL CIRCUIT TABLE CONTAINS

- a *virtual circuit identifier (VCI) that uniquely identifies the connection at this* switch and that will be carried inside the header of the packets that belong to this connection.

- an incoming interface on which packets for this VC arrive at the switch.

- an outgoing interface in which packets for this VC leave the switch.

a potentially different VCI that will be used for outgoing packets.

- If a packet arrives on the designated incoming interface and that packet contains the designated VCI value in its header, then that packet should be sent out the specified outgoing interface with the specified outgoing VCI value first having been placed in its header.
- Note that the combination of the VCI of packets as they are received at the switch *and the interface on which they are received uniquely identifies the virtual connection.*

- There may of course be many virtual connections established in the switch at one time.

- Thus, the VCI is not a globally significant identifier for the connection; rather, it has significance only on a given link—that is, it has *link local scope*.

- Whenever a new connection is created, we need to assign a new VCI for that connection on each link that the connection will traverse. We also need to ensure that

- the chosen VCI on a given link is not currently in use on that link by some existing connection.

permanent virtual circuit (PVC)

- There are two broad classes of approach to establishing connection state. One is to have a network administrator configure the state, in which case the virtual circuit is ―permanent.‖ Of course, it can also be deleted by the administrator, so a permanent virtual circuit (PVC) might best be thought of as a long-lived or administratively configured VC.

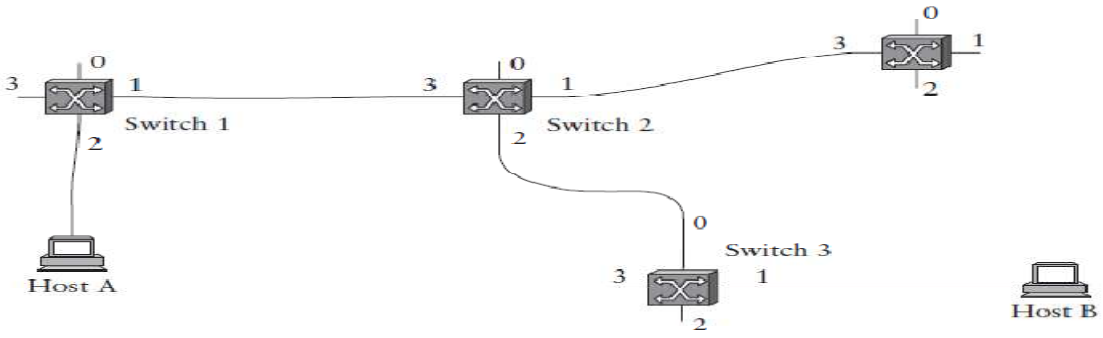
Switched Virtual Circuit (SVC)

- Alternatively, a host can send messages into the network to cause the state to be established. This is referred to as *signalling, and the resulting virtual circuits* are said to be *switched.*

Example PVC construction

- Let’s assume that a network administrator wants to manually create a new virtual connection from host A to host B.
- *The salient characteristic of a switched virtual circuit (SVC)* is that a host may set up and delete such a VC dynamically without the involvement of a network administrator.

- Note that an SVC should more accurately be called a ―signalled‖ VC, since it is the use of signalling (not switching) .



Incoming Interface	Incoming VCI	Outgoing Interface	Outgoing VCI
2	5	1	11

(a)

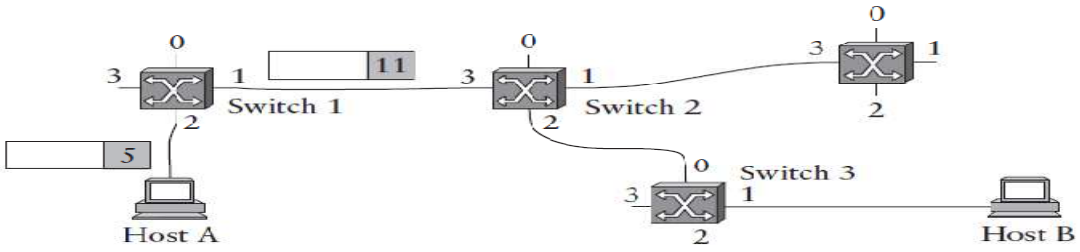
Incoming Interface	Incoming VCI	Outgoing Interface	Outgoing VCI
3	11	2	7

(b)

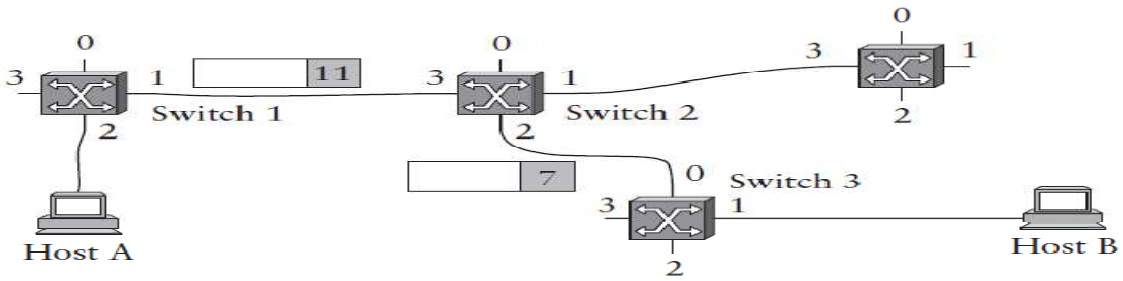
Incoming Interface	Incoming VCI	Outgoing Interface	Outgoing VCI
0	7	1	4

(c)

A packet is sent into a virtual circuit network



A packet makes its way through a virtual circuit network.



SVC CONSTRUCTION

- **To start the signalling process, host A sends a setup message into the network,that is, to switch 1.**

- **The setup message contains, among other things, the complete destination address of host B. The setup message needs to get all the way to B to create the necessary connection state in every switch along the way.**

We can see that getting the setup message to B is a lot like getting a datagram to B, in that the switches have to know which output to send the setup message to so that it eventually reaches B.For now, let's just assume that the switches know enough about the network topology to figure out how to do that, so that the setup message flows on to switches 2 and 3 before finally reaching host B.

When switch 1 receives the connection request, in addition to sending it on to switch 2, it creates a new entry in its virtual circuit table for this new connection. This entry is exactly the same as shown previously in virtual circuit Table.

The main difference is that now the task of assigning an unused VCI value on the interface is performed by the switch. In this example, the switch picks the value 5. The virtual circuit table now has the following information: ―When packets arrive on port 2 with identifier 5, send them out on port 1.‖ Another issue is that, somehow, host A will need to learn that it should put the VCI value of 5 in packets that it wants to send to B.

When switch 2 receives the setup message, it performs a similar process; in this example it picks the value 11 as the incoming VCI value. Similarly, switch 3 picks 7 as the value for its incoming VCI. Each switch can pick any number it likes, as long as that number is not currently in use for some other connection on that port of that switch. As noted above, VCIs have ―link local scope‖; that is, they have no global significance.

Finally, the setup message arrives at host B. Assuming that B is healthy and willing to accept a connection from host A, it too allocates an incoming VCI value, in this case 4. This VCI value can be used by B to identify all packets coming from host A.

When host A no longer wants to send data to host B, it tears down the connection by sending a teardown message to switch 1. The switch removes the relevant entry from its table and forwards the message on to the other switches in the path, which similarly delete the appropriate table entries. At this point, if host A were to send a packet with a VCI of 5 to switch 1, it would be dropped as if the connection had never existed.

There are several things to note about virtual circuit switching:

■ Since host A has to wait for the connection request to reach the far side of the network and return before it can send its first data packet, there is at least one RTT of delay before data is sent.

■ While the connection request contains the full address for host B (which mightbe quite large, being a global identifier on the network), each data packet contains only a small identifier, which is only unique on one link. Thus, the per-packet overhead caused by the header is reduced relative to the datagram model.

■ If a switch or a link in a connection fails, the connection is broken and a new one will need to be established. Also, the old one needs to be torn down to free up table storage space in the switches.

■ The issue of how a switch decides which link to forward the connection request on has been glossed over. In essence, this is the same problem as building up the forwarding table for datagram forwarding, which requires some sort of *routing algorithm*.

For example, an X.25 network—a packet-switched network that uses the connection-oriented model—employs the following three-part strategy:

1 Buffers are allocated to each virtual circuit when the circuit is initialized.

2 The sliding window protocol is run between each pair of nodes along the virtual circuit, and this protocol is augmented with flow control to keep the sending node from overrunning the buffers allocated at the receiving node.

3 The circuit is rejected by a given node if not enough buffers are available at that node when the connection request message is processed.

In doing these three things, each node is ensured of having the buffers it needs to queue the packets that arrive on that circuit. This basic strategy is usually called *hop-by-hop flow control.*

The most popular examples of virtual circuit technologies are Frame Relay and asynchronous transfer mode (ATM).

Frame Relay packet format.

PACKET SWITCHED NETWORKS:(IP,ARP,RARP,ICMP,DHCP)

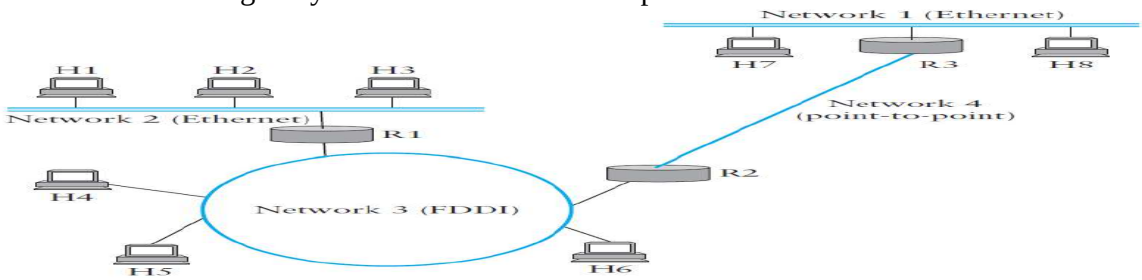
INTERNET PROTOCOL

- We use the term ―internetwork,‖ or sometimes just ―internet‖ refers to an arbitrary collection of networks interconnected to provide some sort of host to- host packet delivery service.

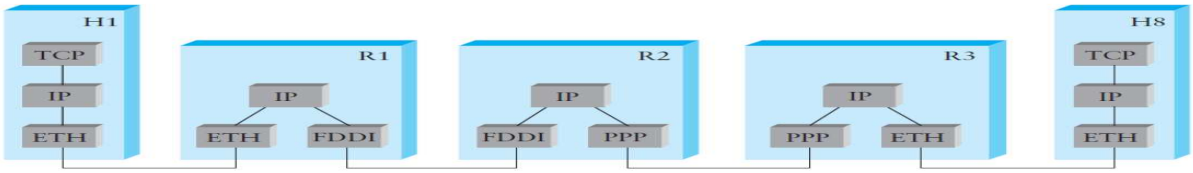
- An internet is a *logicAn internetwork is often referred to as a "network of networks" because it is made up of lots of smaller networks.al* network built out of a collection of physical networks.

- The nodes that interconnect the networks are called routers. They are also sometimes called gateways.

- The Internet Protocol is the key tool used today to build scalable, heterogeneous internetworks. It was originally known as the Kahn-Cerf protocol after its inventors.



A simple internetwork. H*n* = host; R*n* = router.



A simple internetwork, showing the protocol layers used to connect H1 to H8 in the previous figure.

Service Model

- Service model is to define its *service model*, that is, the host-to-host services you want to provide. The IP service model can be thought of as having two parts: an addressing scheme, which provides a way to identify all hosts in the internetwork.

- and a datagram (connectionless) model of data delivery. This service model is sometimes called *best effort* because, although IP makes every effort to deliver datagrams, it makes no guarantees.

Datagram Delivery

- The IP datagram is fundamental to the Internet Protocol.

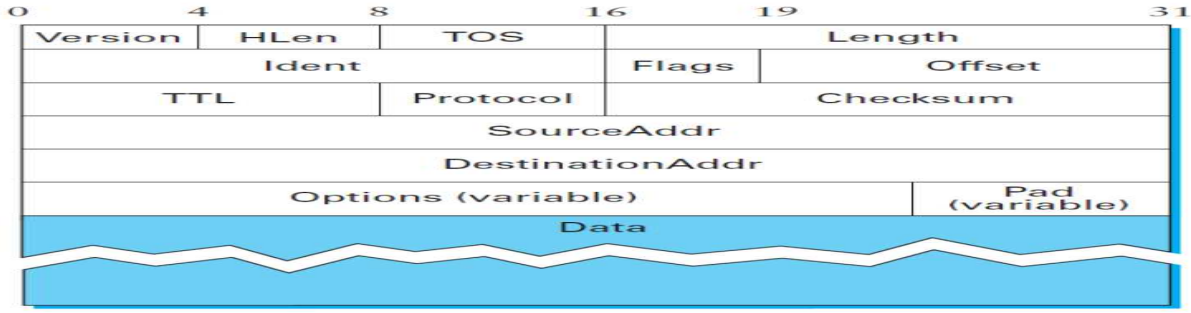
- Datagram is a type of packet that happens to be sent in a connectionless manner over a network. Every datagram carries enough information to let the network forward the packet to its correct destination; there is no need for any advance setup mechanism to tell the network what to do when the packet arrives.

- —The —best-effort‖ part means that if something goes wrong and the packet gets lost, corrupted, misdelivered, or in any way fails to reach its intended destination, the network does nothing—it made its best effort, and that is all it has to do. It does not make any attempt to recover from the failure. This is sometimes called an *unreliable* service‖.

- Best-effort, connectionless service is about the simplest service you could ask for from an internetwork, and this is a great strength.

- Best-effort delivery does not just mean that packets can get lost. Sometimes they can get delivered out of order, and sometimes the same packet can get delivered more than once. The higher-level protocols or applications that run above IP need to beaware of all these possible failure modes.

Packet Format



Version field-denotes version of IP. The current version of IP is 4, and it is sometimes called IPv4.

The next field, HLen,-specifies the length of the header in 32-bit words. When there are no options, which is most of the time, the header is 5 words (20 bytes) long.

TOS (8 bits-type of service) field - its basic function is to allow packets to be treated differently based on application needs. For example, the TOS value might determine whether or not a packet should be placed in a special queue that receives low delay.

Length (16 bits)-length of the datagram, including the header. Unlike the HLen field, the Length field counts bytes rather than words. Thus, the maximum size of an IP datagram is 65,535

bytes. The physical network overwhich IP is running, however, may not support such long packets. For this reason,IP supports a fragmentation and reassembly process.
The second word of the header contains information about fragmentation, and the details of its use are presented under ―Fragmentation and Reassembly‖ .

Ident field-to identify the fragments, all the fragments have same id ,it is same as original packet's id.

Flags field –it has three bits first bit is always 0(default),second bit is DF(don't fragment)-denote no need to fragment the packet, third bit is More Fragment-denote whether the current fragment has follow up fragments or not(if it is 1-more fragments,0-this is the last fragment).

Offset field-meaning that there are more fragments to follow, and it sets the Offset to 0, since this fragment contains the first part of the original datagram. The data carried in the second fragment starts with the 513th byte of the original data, so the Offset field in this header is set to 64, which is 512 ÷ 8. Why the division by 8? Because the designers of IP decided that fragmentation should always happen on 8-byte boundaries, which means that the Offset field counts 8-byte chunks, not bytes.The third fragment contains the last 376 bytes of data, and the offset is now 2 × 512 ÷ 8 = 128.
Moving on to the third word of the header contains

TTL field(time to live)- was set to a specific number of seconds that the packet would be allowed to live and routers along the path would decrement this field until it reached 0. However,since it was rare for a packet to sit for as long as 1 second in a router, and routers did not all have access to a common clock, most routers just decremented the TTL by 1 as they forwarded the packet.

The **Protocol** field- is simply a demultiplexing key that identifies the higher-level protocol to which this IP packet should be passed. There are values defined for TCP (6), UDP (17), and many other protocols that may sit above IP in the protocol graph.

The **Checksum** field(16 bits)-Error detection bits.

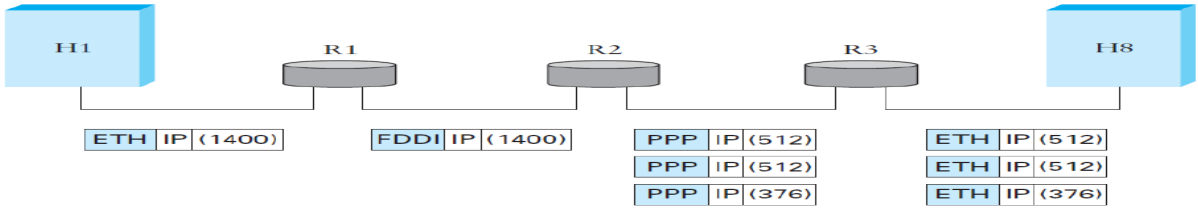
Source address-to identify the source.

Destination address-to identify the destination.
Finally, there may be a number of **options** at the end of the header. The presence or absence of options may be determined by examining the header length (HLen) field. While options are used fairly rarely, a complete IP implementation must handle them all.

- **Fragmentation and Reassembly**
- One of the problems of providing a uniform host-to-host service model over a heterogeneous collection of networks is that each network technology tends to have its own idea of how large a packet can be.
- For example, an Ethernet can accept packets up to 1500 bytes long, while FDDI packets may be 4500 bytes long. This leaves two choices for the IP service model: make sure that all IP datagrams are small enough to fit inside one packet on any network technology, or provide a means by which packets can be fragmented and reassembled when they are too big to go over a given network technology.
- For example, two hosts connected to FDDI networks that are interconnected by a point-to-point link would not need to send packets small enough to fit on an Ethernet. The central idea here is that every network type has a *maximum transmission unit* (MTU), which is the largest IP datagram that it can carry in a frame.
- Note that this value is smaller than the largest packet size on that network because the IP datagram needs to fit in the *payload* of the link-layer frame.
- When a host sends an IP datagram, therefore, it can choose any size that it wants. A reasonable choice is the MTU of the network to which the host is directly attached.

- Then fragmentation will only be necessary if the path to the destination includes a network with a smaller MTU. Should the transport protocol that sits on top of IP give IP a packet larger than the local MTU, however, then the source host must fragment it.
- — Fragmentation typically occurs in a router when it receives a datagram that it wants to forward over a network that has an MTU that is smaller than the received datagram‖.
- To enable these fragments to be reassembled at the receiving host, they all carry the same identifier in the Ident field. This identifier is chosen by the sending host and is intended to be unique among all the datagrams that might arrive at the destination from this source over some reasonable time period.
- Since all fragments of the original datagram contain this identifier, the reassembling host will be able to recognize those fragments that go together.
- Should all the fragments not arrive at the receiving host, the host gives up on the reassembly process and discards the fragments that did arrive. IP does not attempt to recover from missing fragments.

To see what this all means, consider what happens when host H1 sends a datagram to host H8 in the example internet shown in the following Figure 1500 bytes for the two Ethernets, 4500 bytes for the FDDI network, and 532 bytes for the point-to-point network, then a 1420-byte datagram (20-byte IP header plus 1400 bytes of data) sent from H1 makes it across the first Ethernet and the FDDI network without fragmentation but must be fragmented into three datagrams at router R2. These three fragments are then forwarded by router R3 across the second Ethernet to the destination host.



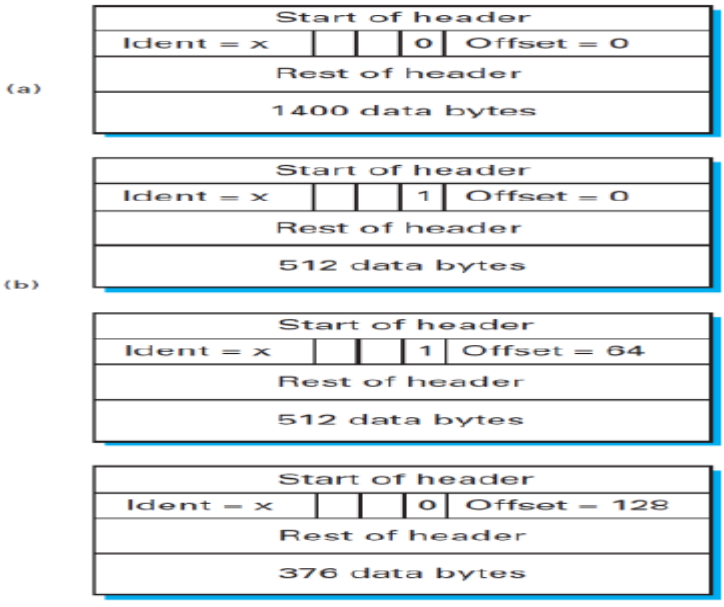
IP datagrams traversing the sequence of physical networks

Two important points regarding Fragmentation and Reassembly:

1 Each fragment is itself a self-contained IP datagram that is transmitted over a sequence of physical networks, independent of the other fragments.

2 Each IP datagram is reencapsulated for each physical network over which it travels.

Header fields used in IP fragmentation. (a) Unfragmented packet; (b) frag-mented packets.



Implementation

First, we define the key data structure (FragList) that is used to hold the individual fragments that arrive at the destination. Incoming fragments are saved in this data structure until all the fragments in the original datagram have arrived, at which time they are reassembled into a complete datagram and passed up to some higher-level protocol.

Note that each element in FragList contains either a fragment or a hole.

```
#define FRAGOFFMASK 0x1fff
#define FRAGOFFSET(fragflag) ((fragflag) & FRAGOFFMASK)
#define INFINITE_OFFSET 0xffff
/* structure to hold the fields that uniquely identify fragments  of the same IP datagram */
typedef struct fid {
IpHost source;
IpHost dest;
u_char prot;
u_char pad;
u_short ident;
} FragId;
typedef struct hole {
u_int first;
u_int last;
} Hole;
#define HOLE 1
#define FRAG 2
/* structure to hold a fragment or a hole */
typedef struct fragif {
u_char type;
union {
Hole hole;
Msg frag;
} u;
Struct fragif *next, *prev;
} FragInfo;

/* structure to hold all the fragments and holes for a
single IP datagram being reassembled */
typedef struct FragList {
u_short nholes;
FragInfo head; /* dummy header node */
Binding binding;
bool gcMark; /* garbage collection flag */
} FragList;
```

Global Addresses

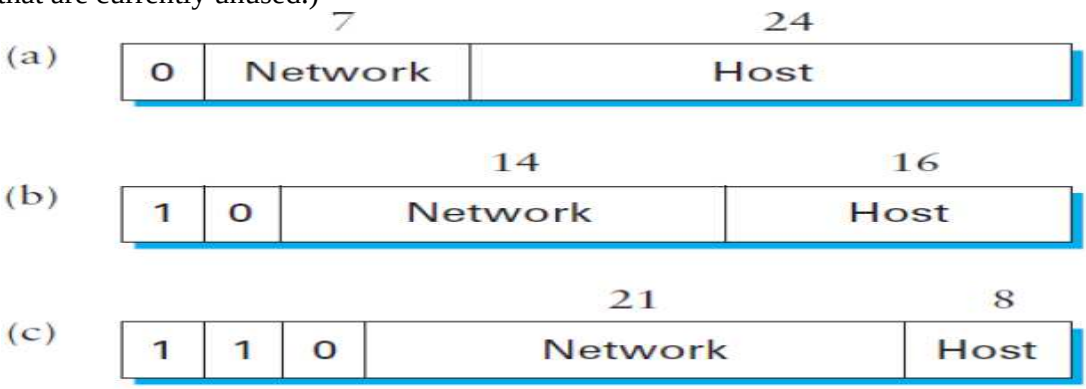
- Global uniqueness is the first property that should be provided in an addressing scheme.
- Ethernet addresses are globally unique, but that alone does not suffice for an addressing scheme in a large internetwork. Ethernet addresses are also *flat*, which means that they have no structure and provide very few clues to routing protocols.

- In contrast, IP addresses are *hierarchical*, by which we mean that they are made up of several parts that correspond to some sort of hierarchy in the internetwork.

- Specifically, IP addresses consist of two parts, a network part and a host part. This is a fairly logical structure for an internetwork, which is made up of many interconnected networks.
- The network part of an IP address identifies the network to which the host is attached; all hosts attached to the same network have the same network part in their IP address.

The host part then identifies each host uniquely on that particular network.

IP addresses are divided into three different classes.
(There are also class D addresses that specify a multicast group, and class E addresses that are currently unused.)



IP addresses : (a)class A;(b)class B;class C.

The class of an IP address is identified in the most significant few bits. If the first bit is 0, it is a class A address. If the first bit is 1 and the second is 0, it is a class B address.
If the first two bits are 1 and the third is 0, it is a class C address. Thus, of the approximately 4 billion possible IP addresses, half are class A, one quarter are class B, and one-eighth are class C.

Each class allocates a certain number of bits for the network part of the address and the rest for the host part. Class A networks have 7 bits for the network part and 24 bits for the host part, meaning that there can be only 126 class A networks (the values 0 and 127 are reserved), but each of them can accommodate up to 224 − 2 (about 16 million) hosts (again,there are two reserved values).

Class B addresses allocate 14 bits for the network and 16 bits for the host, meaning that each class B network has room for 65,534 hosts. Finally, class C addresses have only 8 bits for the host and 21 for the network part. Therefore, a class C network can have only 256 unique host identifiers, which means only 254 attached hosts (one host identifier, 255, is reserved for broadcast, and 0 is not a valid host number). However, the addressing scheme supports 221 class C networks.

By convention, IP addresses are written as four *decimal* integers separated by dots. Each integer represents the decimal value contained in 1 byte of the address, starting at the most significant. For example, the address of the computer on which this sentence was typed is 171.69.210.245.

Datagram Forwarding in IP

Forwarding is the process of taking a packet from an input and sending it out on the appropriate output, while *routing* is the process of building up the tables that allow the correct output for a packet to be determined.
The main points to bear in mind as we discuss the forwarding of IP datagrams are the following:

- Every IP datagram contains the IP address of the destination host.

- The ―network part‖ of an IP address uniquely identifies a single physical network that is part of the larger Internet.

- All hosts and routers that share the same network part of their address are connected to the same physical network and can thus communicate with each other by sending frames over that network.

- Every physical network that is part of the Internet has at least one router that, by definition, is also connected to at least one other physical network; this router can exchange packets with hosts or routers on either network.

Forwarding IP datagram:

A datagram is sent from a source host to a destination host, possibly passing through several routers along the way. Any node, whether it is a host or a router, first tries to establish whether it is connected to the same physical network as the destination.

To do this, it compares the network part of the destination address with the network part of the address of each of its network interfaces.

If the node is not connected to the same physical network as the destination node, then it needs to send the datagram to a router. In general, each node will have a choice of several routers, and so it needs to pick the best one, or at least one that has a reasonable chance of getting the datagram closer to its destination.

The router that it chooses is known as the *next hop* router. The router finds the correct next hop by consulting its forwarding table. The forwarding table is conceptually just a list of _NetworkNum, NextHop_ pairs. (As we will see below, forwarding tables in practice often contain some additional information related to the next hop.)
Normally, there is also a default router that is used if none of the entries in the table match the destination's network number.

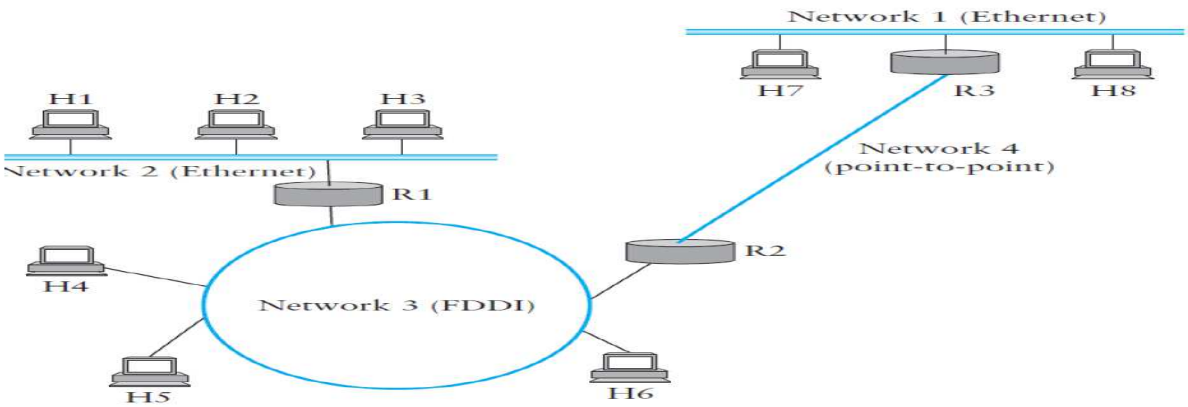
For a host, it may be quite acceptable to have a default router and nothing else—this means that all datagrams destined for hosts not on the physical network to which the sending host is attached will be sent out through the default router.

The datagram forwarding algorithm in the following way:

```
if (NetworkNum of destination = NetworkNum of one of my interfaces) then
deliver packet to destination over that interface
else
if (NetworkNum of destination is in my forwarding table) then
deliver packet to NextHop router
else
deliver packet to default router
```

For a host with only one interface and only a default router in its forwarding table, this simplifies to

```
if (NetworkNum of destination = my NetworkNum) then
deliver packet to destination directly
else
deliver packet to default router
```



Consider the above figure First, suppose that H1 wants to send a datagram to H2. Since they are on the same physical network, H1 and H2 have the same network number in their IP address. Thus, H1 deduces that it can deliver the datagram directly to H2 over the Ethernet.

The one issue that needs to be resolved is how H1 finds out the correct Ethernet address for H2—this is the address resolution mechanism. Now suppose H1 wants to send a datagram to H8. Since these hosts are on different physical networks, they have different network numbers, so H1 deduces that it needs to send the datagram to a router.

R1 is the only choice—the default router—so H1 sends the datagram over the Ethernet to R1. Similarly, R1 knows that it cannot deliver a datagram directly to H8 because neither of R1's interfaces is on the same network as H8.

Suppose R1's default router is R2; R1 then sends the datagram to R2 over the token ring network.

FORWARDING TABLE FOR ROUTER R2

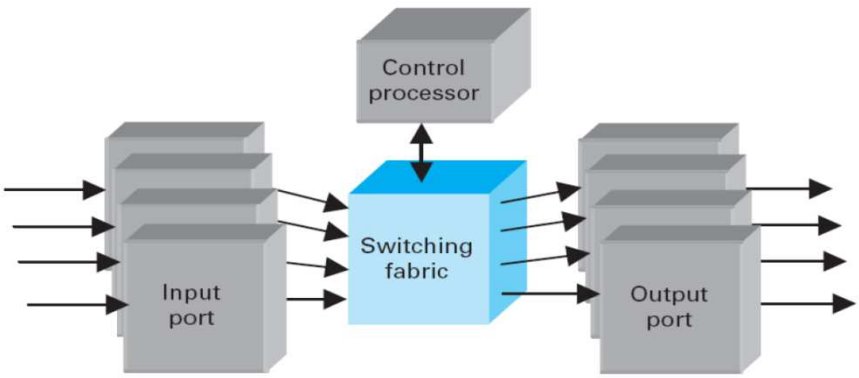
NetworkNum	NextHop
1	R3
2	R1

COMPLETE FORWARDING TABLE FOR R2

NetworkNum	NextHop
1	R3
2	R1
3	Interface 1
4	Interface 0

ROUTER IMPLEMENTATION

Block diagram of a router



The control processor is responsible for running the *routing protocols* and generally acts as the central point of control of the router. The switching fabric transfers packets from one port to another, just as in a switch, and the ports provide a range of functionality to allow the router to interface to links of various types (e.g., Ethernet, SONET, etc.).

A few points are worth noting about router design and how it differs from switch design. First, routers must be designed to handle variable-length packets, a constraint that does not apply to ATM switches but is certainly applicable to Ethernet or Frame Relay switches.
It turns out that many high-performance routers are designed using a switching fabric that is cell based.

In such cases the ports must be able to convert variable-length packets into cells and back again.

Another consequence of the variable length of IP datagrams is that it can be harder to characterize the performance of a router than a switch that forwards only cells. Routers can usually forward a certain number of packets per second, and this implies that the total throughput in *bits* per second depends on packet size.

Router designers generally have to make a choice as to what packet length they will support at *line rate*. That is, if pps (packets per second) is the rate at which packets arriving on a particular port can be forwarded, and linerate is the physical speed of the port in bits per second, then there will be some packetsize in bits such that

$$\text{packetsize} \times \text{pps} = \text{linerate}$$

When it comes to the task of forwarding IP packets, routers can be broadly characterized as having either a *centralized* or *distributed* forwarding model.

In the centralized model, the IP forwarding algorithm, outlined earlier in this section, is done in a single processing engine that handles the traffic from all ports.

In the distributed model, there are several processing engines, perhaps one per port, or more often one per line card, where a line card may serve one or more physical ports.

Each model has advantages and disadvantages. All things being equal, a distributed forwarding model should be able to forward more packets per second through the router as a whole because there is more processing power in total.

But a distributed model also complicates the software architecture because each forwarding engine typically needs its own copy of the forwarding table, and thus it is necessary for the control processor to ensure that the forwarding tables are updated consistently and in a timely manner.

ADDRESS RESOLUTION PROTOCOL(ARP)

The main issue is that IP datagrams contain IP addresses, but the physical interface hardware on the host or router to which you want to send the datagram only understands the addressing scheme of that particular network.

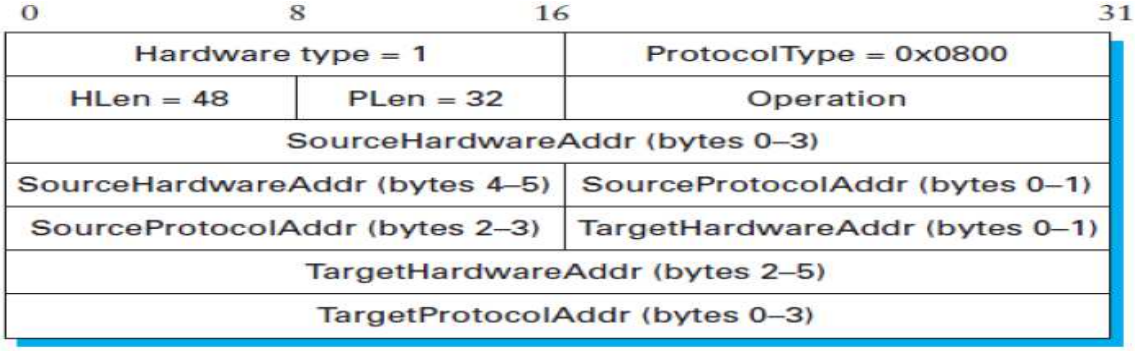
Thus, we need to translate the IP address to a link-level address that makes sense on this network.

We can then encapsulate the IP datagram inside a frame that contains that link-level address and send it either to the ultimate destination or to a router that promises to forward the datagram toward the ultimate destination.

One simple way to map an IP address into a physical network address is to encode a host's physical address in the host part of its IP address.

- For example, a host with physical address 00100001 01001001 (which has the decimal value 33 in the upper byte and 81 in the lower byte) might be given the IP address 128.96.33.81.
- While this solution has been used on some networks, it is limited in that the network's physical addresses can be no more than 16 bits long in this example; they can be only 8 bits long on a class C network. This clearly will not work for 48-bit Ethernet addresses.
- A more general solution would be for each host to maintain a table of address pairs; that is, the table would map IP addresses into physical addresses.

- While this table could be centrally managed by a system administrator and then copied to each host on the network, a better approach would be for each host to dynamically learn the contents of the table using the network. This can be accomplished using the Address Resolution Protocol (ARP).
- The goal of ARP is to enable each host on a network to build up a table of mappings between IP addresses and link-level addresses.
- Since these mappings may change over time (e.g., because an Ethernet card in a host breaks and is replaced by a new one with a new address), the entries are timed out periodically and removed. This happens on the order of every 15 minutes. The set of mappings currently stored in a host is known as the ARP cache or ARP table.
- ARP takes advantage of the fact that many link-level network technologies, such as Ethernet and token ring, support broadcast. If a host wants to send an IP datagram to a host (or router) that it knows to be on the same network (i.e., the sending and receiving node have the same IP network number), it first checks for a mapping in the cache.
- If no mapping is found, it needs to invoke the Address Resolution Protocol over the network. It does this by broadcasting an ARP query onto the network.
- This query contains the IP address in question (the ―target IP address‖). Each host receives the query and checks to see if it matches its IP address. If it does match, the host sends a response message that contains its link-layer address back to the originator of the query. The originator adds the information contained in this response to its ARP table.
- The query message also includes the IP address and link-layer address of the sending host. Thus, when a host broadcasts a query message, each host on the network can learn the sender's link-level and IP addresses and place that information in its ARP table.
- However, not every host adds this information to its ARP table. If the host already has an entry for that host in its table, it ―refreshes‖ this entry; that is, it resets the length of time until it discards the entry.
- If that host is the target of the query, then it adds the information about the sender to its table, even if it did not already have an entry for that host. This is because there is a good chance that the source host is about to send it an application-level message, and it may eventually have to send a response or ACK back to the source; it will need the source's physical address to do this.
- If a host is not the target and does not already have an entry for the source in its ARP table, then it does not add an entry for the source. This is because there is no reason to believe that this host will ever need the source's link-level address; there is no need to clutter its ARP table with this information.



ARP packet format for mapping IP addresses into Ethernet addresses.

The ARP packet contains

- a HardwareType field, which specifies the type of physical network (e.g.,Ethernet).
- a ProtocolType field, which specifies the higher-layer protocol (e.g., IP).
- HLen (―hardware‖ address length) and PLen (―protocol‖ address length) fields, which specify the length of the link-layer address and higher-layer protocol address, respectively.
- an Operation field, which specifies whether this is a request or a response.
- the source and target hardware (Ethernet) and protocol (IP) addresses.

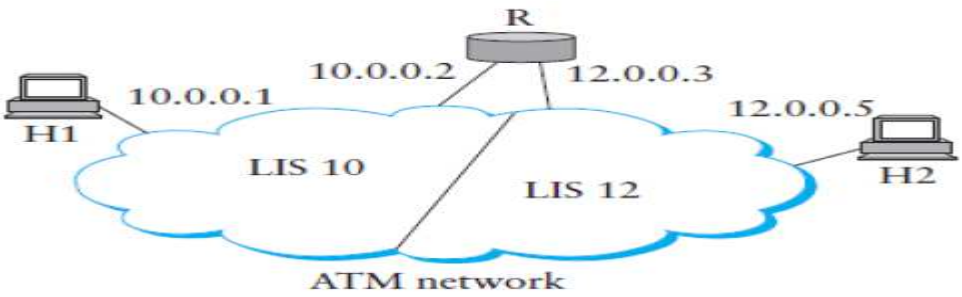
ATMARP

ATM network don't support broadcast messages,so it follows different ARP procedure that may be used in an ATM network and that does not depend on broadcast or LAN emulation.

This procedure is known as ATMARP and is part of the *Classical IP over ATM* model. The reason for calling the model ―classical‖ will become apparent shortly. Like LAN emulation, ATMARP relies on the use of a server to resolve addresses—in this case, it is called an ARP server, and its behavior is described below.

A key concept in the Classical IP over ATM model is the *logical IP subnet* (LIS).The LIS abstraction allows us to take one large ATM network and subdivide it into several smaller subnets.

All nodes on the same subnet have the same IP network number. And just as in -classical‖ IP, two nodes (hosts or routers) that are on the same subnet can communicate directly over the ATM network, whereas two nodes that are on different subnets will have to communicate via one or more routers.



Logical IP subnets.

in the above Figure Note that the IP address of host H1 has a network number of 10, as does the router interface that connects to the left-hand LIS, while H2 has a network number of 12, as does the right-hand interface on the router. That is, H1 and the router connect to the same LIS (LIS10) while H2 is on a different subnet (LIS 12) to which the router also connects.

An advantage of the LIS model is that we can connect a large number of hosts and routers to a big ATM network without necessarily giving them all addresses from the same IP network. This may make it easier to manage address assignment, for example, in the case where not all nodes connected to the ATM network are under the control of the same administrative entity.

The division of the ATM network into a number of LISs also improves scalability by limiting the number of nodes that must be supported by a single ARP server.

The basic job of an ARP server is to enable nodes on a LIS to resolve IP addresses to ATM addresses without using broadcast. Each node in the LIS must be configured with the ATM address of the ARP server, so that it can establish a VC to the server when it boots. Once it has a VC to the server, the node sends a registration message to the ARP server that contains both the IP and ATM addresses of the registering node.

Thus the ARP server builds up a complete database of all the _IP address, ATM address_ pairs. Once this is in place, any node that wants to send a packet to some IP address can ask the ARP server to provide the corresponding ATM address.

Once this is received, the sending node can use ATM signalling to set up a VC to that ATM address, and then send the packet. Just like conventional ARP, a cache of IP-to-ATM address mappings can be maintained. In addition, the node can keep a VC established to that ATM destination as long as there is enough traffic flowing to justify it, thus avoiding the delay of setting up the VC again when the next packet arrives.

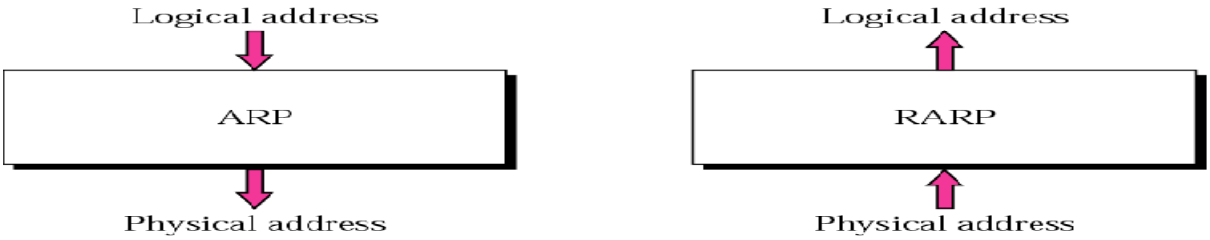
An interesting consequence of the Classical IP over ATM model is that two nodes on the same ATM network cannot establish a direct VC between themselves if they are on different subnet.

REVERSE ADDRESS RESOLUTION PROTOCOL(RARP)

RARP finds the logical address for a machine that only knows its physical address. This if often encountered on thin-client workstations. No disk, so when machine is booted, it needs to know its IP address (don't want to burn the IP address into the ROM).

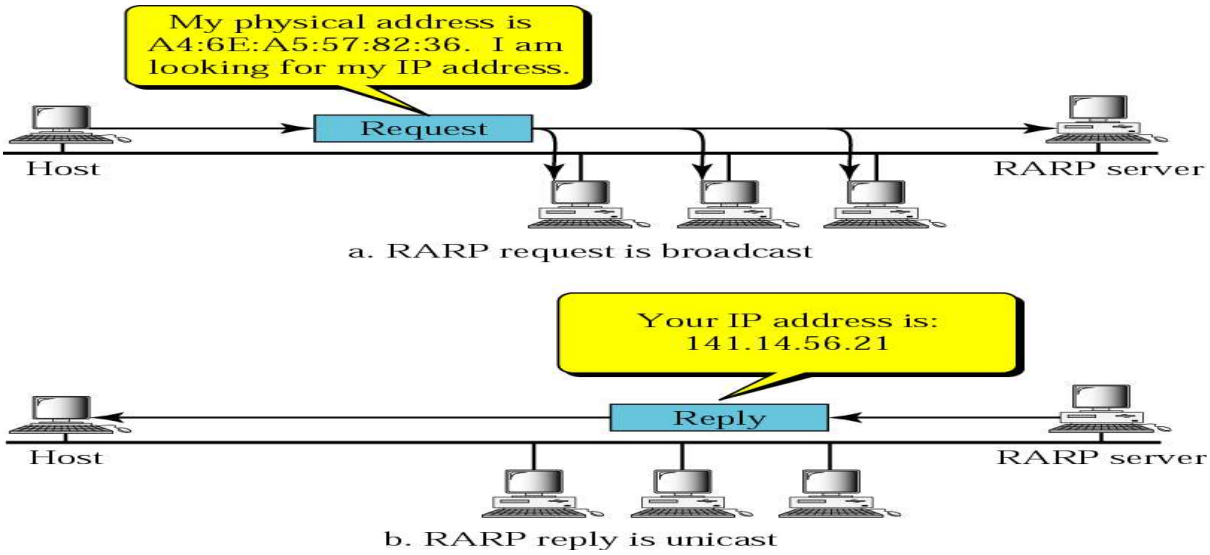
RARP requests are broadcast, RARP replies are unicast. If a thin-client workstation needs to know its IP address, it probably also needs to know its subnet mask, router address, DNS address, etc.

So we need something more than RARP. BOOTP, and now DHCP have replaced RARP.



Position of ARP and RARP in TCP/IP protocol suite

RARP operation



RARP packet

Hardware type		Protocol type
Hardware length	Protocol length	Operation Request 3, Reply 4
Sender hardware address (For example, 6 bytes for Ethernet)		
Sender protocol address (For example, 4 bytes for IP) (It is not filled for request)		
Target hardware address (For example, 6 bytes for Ethernet) (It is not filled for request)		
Target protocol address (For example, 4 bytes for IP) (It is not filled for request)		

The ARP packet contains

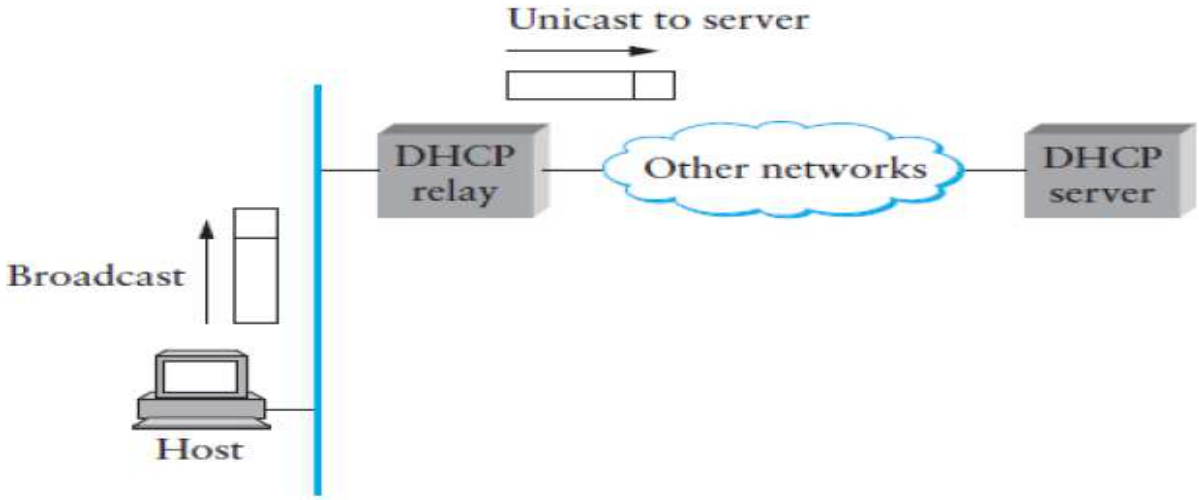
- a HardwareType field, which specifies the type of physical network (e.g.,Ethernet).
- a ProtocolType field, which specifies the higher-layer protocol (e.g., IP).
- HLen (―hardware‖ address length) and PLen (―protocol‖ address length) fields, which specify the length of the link-layer address and higher-layer protocol address, respectively.
- an Operation field, which specifies whether this is a request or a response.
- the source and target hardware (Ethernet) and protocol (IP) addresses.

DYNAMIC HOST CONTROL PROTOCOL(DHCP)

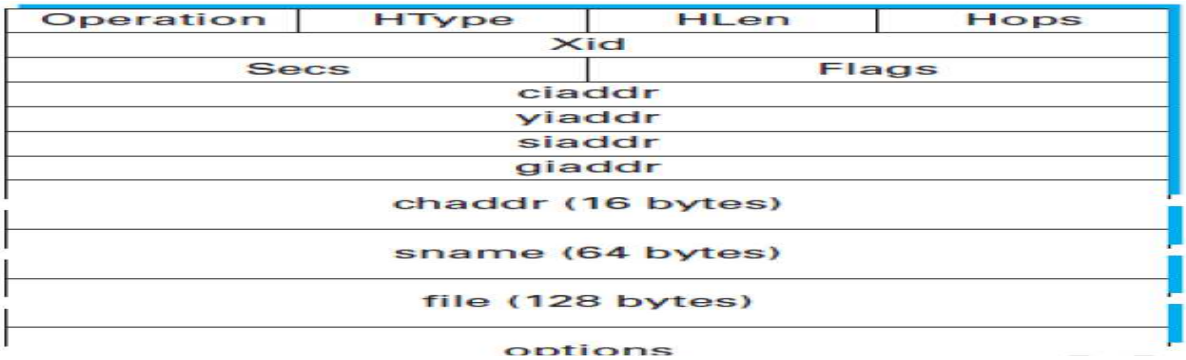
- Most host operating systems provide a way for a system administrator, or even a user, to manually configure the IP information needed by a host. However, there are some obvious drawbacks to such manual configuration.
- One is that it is simply a lot of work to configure all the hosts in a large network directly, especially when you consider that such hosts are not reachable over a network until they are configured.
- Even more importantly, the configuration process is very error-prone, since it is necessary to ensure that every host gets the correct network number and that no two hosts receive the same IP address.
- For these reasons, automated configuration methods are required. The primary method uses a protocol known as the Dynamic Host Configuration Protocol (DHCP).
- DHCP relies on the existence of a DHCP server that is responsible for providing configuration information to hosts. There is at least one DHCP server for an administrative domain.
- At the simplest level, the DHCP server can function just as a centralized repository for host configuration information. Consider, for example, the problem of administering addresses in the internetwork of a large company.
- DHCP saves the network administrators from having to walk around to every host in the company with a list of addresses and network map in hand and configuring each host manually.
- Instead, the configuration information for each host could be stored in the DHCP server and automatically retrieved by each host when it is booted or connected to the network.
- However, the administrator would still pick the address that each host is to receive; he would just store that in the server. In this model, the configuration information for reach host is stored in a table that is indexed by some form of unique client identifier typically the ―hardware address‖ (e.g., the Ethernet address of its network adaptor).
- A more sophisticated use of DHCP saves the network admininstrator from even having to assign addresses to individual hosts. In this model, the DHCP server maintains a pool of available addresses that it hands out to hosts on demand.
- This considerably reduces the amount of configuration an administrator must do, since now it is only necessary to allocate a range of IP addresses (all with the same network number) to each network.
- Since the goal of DHCP is to minimize the amount of manual configuration required for a host to function, it would rather defeat the purpose if each host had to be configured with the address of a DHCP server.
- Thus, the first problem faced by DHCP is that of server discovery. To contact a DHCP server, a newly booted or attached host sends a DHCPDISCOVER message to a special IP address (255.255.255.255) that is an IP broadcast address.
- This means it will be received by all hosts and routers on that network. (Routers do not forward such packets onto other networks, preventing broadcast to the entire Internet.) In the simplest case, one of these nodes is the DHCP server for the network.
- The server would then reply to the host that generated the discovery message (all the other nodes would ignore it). However, it is not really desirable to require one DHCP server on

every network because this still creates a potentially large number of servers that need to be correctly and consistently configured.

- Thus, DHCP uses the concept of a *relay agent*. There is at least one relay agent on each network, and it is configured with just one piece of information: the IP address of the DHCP server.
- When a relay agent receives a DHCPDISCOVER message, it unicasts it to the DHCP server and awaits the response, which it will then send back to the requesting client.



A DHCP relay agent receives a broadcast DHCPDISCOVER message from a host and sends a unicast DHCPDISCOVER message to the DHCP server.



DHCP PACKET FORMAT

INTERNET CONTROL MESSAGE PROTOCOL(ICMP)

IP is always configured with a companion protocol, known as the Internet Control Message Protocol (ICMP), that defines a collection of error messages that are sent back to the source host whenever a router or host is unable to process an IP datagram successfully. For example, ICMP defines error messages indicating that the destination host is unreachable (perhaps due to a link failure), that the reassembly process failed, that the TTL had reached 0, that the IP header checksum failed, and so on.

ICMP also defines a handful of control messages that a router can send back to a source host. One of the most useful control messages, called an ICMP-Redirect, tells the source host that there is a better route to the destination.

ICMP-Redirects are used in the following situation. Suppose a host is connected to a network that has two routers attached to it, called R1 and R2, where the host uses R1 as its default router. Should R1 ever receive a datagram from the host, where based on its forwarding table it knows that R2 would have been a better choice for a particular destination address, it sends an

ICMP-Redirect back to the host, instructing it to use R2 for all future datagrams addressed to that destination. The host then adds this new route to its forwarding table.

Queuing Disciplines

Each router must implement some queuing discipline that governs how packets are buffered while waiting to be transmitted.

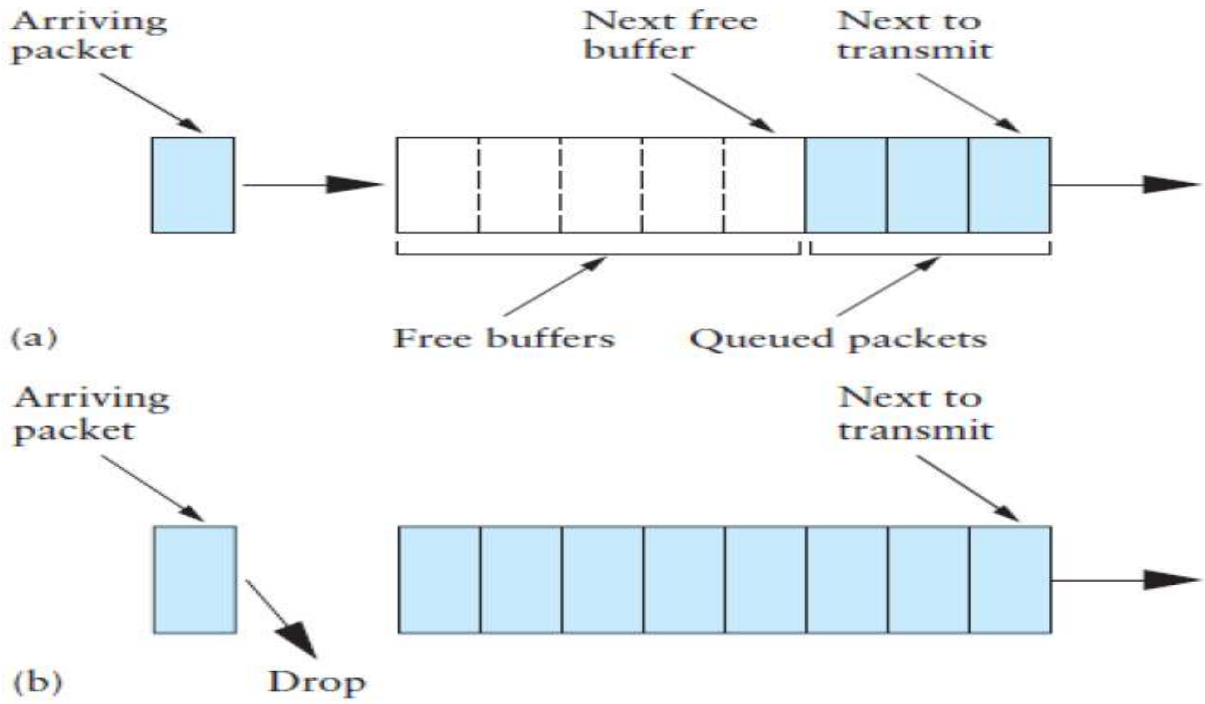
The queuing algorithm can be thought of as allocating both bandwidth (which packets get transmitted) and buffer pace (which packets get discarded). It also directly affects the latency experienced by a packet, by determining how long a packet waits to be transmitted.

Two common queuing algorithms:

- first-in-first-out (FIFO) and
- fair queuing (FQ)

FIFO

The idea of FIFO queuing, also called first-come-first-served (FCFS) queuing, is simple: The first packet that arrives at a router is the first packet to be transmitted.



This is illustrated in Figure (a), which shows a FIFO with ―slots‖ to hold up to eight packets. Given that the amount of buffer space at each router is finite, if a packet arrives and the queue (buffer space) is full, then the router discards that packet, as shown in Figure (b). This is done without regard to which flow the packet belongs to or how important the packet is. This is sometimes called *tail drop*, since packets that arrive at the tail end of the FIFO are dropped. Note that tail drop and FIFO are two separable ideas. FIFO is a *scheduling discipline*—it determines the order in which packets are transmitted. Tail drop is a *drop policy*—it determines which packets get dropped. Because FIFO and tail drop are the simplest instances of scheduling discipline and drop policy, respectively, they are sometimes viewed as a bundle—the vanilla queuing implementation. Unfortunately, the bundle is often referred to simply as ―FIFO queuing,‖ when it should more precisely be called ―FIFO with tail drop.‖ Section 6.4 provides an example of another drop policy, which uses a more complex algorithm than ―Is there a free buffer?‖ to decide when to drop packets. Such a drop policy may be used with FIFO, or with more complex scheduling disciplines. FIFO with tail drop, as the simplest of all queuing algorithms, is the most widely used in Internet routers at the time of writing. This simple approach to queuing

pushes all responsibility for congestion control and resource allocation out to the edges of the network.

A simple variation on basic FIFO queuing is priority queuing. The idea is to mark each packet with a priority; the mark could be carried, for example, in the IP Type of Service (TOS) field. The routers then implement multiple FIFO queues, one for each priority class.
The router always transmits packets out of the highest-priority queue if that queue is nonempty before moving on to the next priority queue. Within each priority, packets are still managed in a FIFO manner. This idea is a small departure from the best-effort delivery model, but it does not go so far as to make guarantees to any particular priority class. It just allows high-priority packets to cut to the front of the line.

The problem with priority queuing, of course, is that the high-priority queue can starve out all the other queues. That is, as long as there is at least one high-priority packet in the high priority queue, lower-priority queues do not get served. For this to be viable, there need to be hard limits on how much high-priority traffic is inserted in the queue.

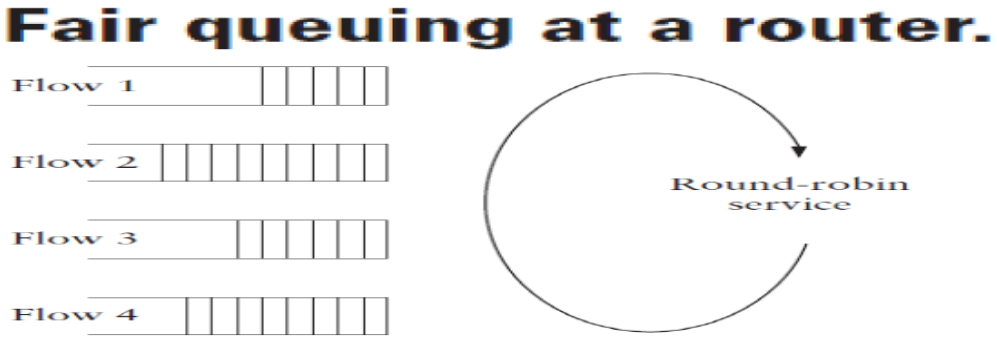
It should be immediately clear that we can't allow users to set their own doing this altogether or provide some form of ―pushback‖ on users. One obvious way to do this is to use economics—the network could charge more to deliver high priority packets than low-priority packets. However, there are significant challenges to implementing such a scheme in a decentralized environment such as the Internet.

Fair Queuing

Fair queuing (FQ) is an algorithm that has been proposed to address this problem. The idea of FQ is to maintain a separate queue for each flow currently being handled by the router. The router then services these queues in a round-robin manner, as illustrated in the following Figure . When a flow sends packets too quickly, then its queue fills up. When a queue reaches a particular length, additional packets belonging to that flow's queue are discarded. In this way, a given source cannot arbitrarily increase its share of the network's capacity at the expense of other flows.

Note that FQ does not involve the router telling the traffic sources anything about the state of the router or in any way limiting how quickly a given source sends packets. In other words, FQ is still designed to be used in conjunction with an endto- end congestion-control mechanism.

It simply segregates traffic so that ill-behaved traffic sources do not interfere with those that are faithfully implementing the end-toend algorithm. FQ also enforces fairness among a collection of flows managed by a well-behaved congestion-control algorithm.



ROUTING ALGORITHMS

The forwarding table is used when a packet is being forwarded and so must contain enough information to accomplish the forwarding function.

This means that a row in the forwarding table contains the mapping from a network number to an outgoing interface and some MAC information, such as the Ethernet address of the next hop.

The routing table, on the other hand, is the table that is built up by the routing algorithms as a precursor to building the forwarding table. It generally contains mappings from network numbers to next hops.

It may also contain information about how this information was learned, so that the router will be able to decide when it should discard some information.

Network Number	NextHop
10	171.69.245.10

(a)

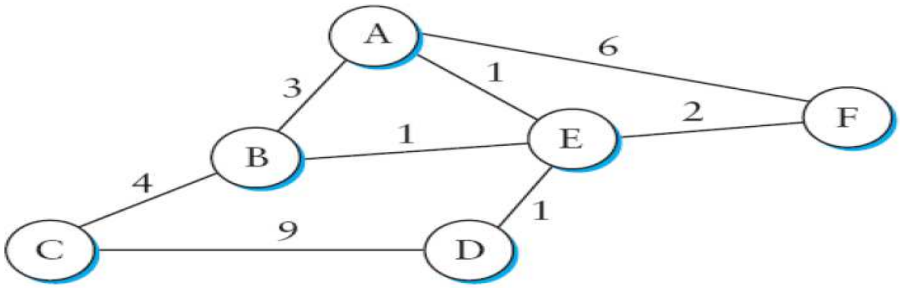
Network Number	Interface	MAC Address
10	if0	8:0:2b:e4:b:1:2

Example rows from (a) routing and (b) forwarding tables

Network as a Graph

Routing is, in essence, a problem of graph theory.

The nodes of the graph, labeled A through F, may be either hosts, switches,routers, or networks.



Network represented as a graph.

The edges of the graph correspond to the network links. Each edge has an associated *cost*, which gives some indication of the desirability of sending traffic over that link.

The basic problem of routing is to find the lowest-cost path between any two nodes, where the cost of a path equals the sum of the costs of all the edges that make up the path. Such a static approach has several shortcomings

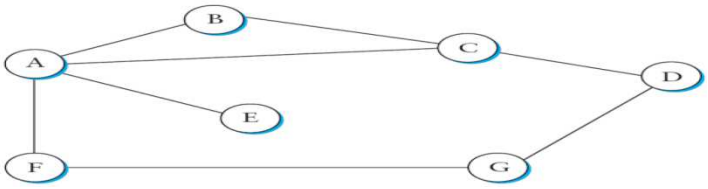
- It does not deal with node or link failures.
- It does not consider the addition of new nodes or links.
- It implies that edge costs cannot change, even though we might reasonably wish to temporarily assign a high cost to a link that is heavily loaded.

For these reasons, routing is achieved in most practical networks by running routing protocols among the nodes. These protocols provide a distributed, dynamic way to solve the problem of finding the lowest-cost path in the presence of link and node failures and changing edge costs.

Two main classes of routing protocols: *distance vector* and *link state*.

Distance Vector Routing

Each node constructs a one-dimensional array (a vector) containing the ―distances‖ (costs) to all other nodes and distributes that vector to its immediate neighbors. The starting assumption for distance-vector routing is that each node knows the cost of the link to each of its directly connected neighbors. A link that is down is assigned an infinite cost.



Distance-vector routing: an example network.

In this example, the cost of each link is set to 1, so that a least-cost path is simply the one with the fewest hops. (Since all edges have the same cost, we do not show the costs in the graph.) We can represent each node's knowledge about the distances to all other nodes as a table like the one given in the following Table.

Note that each node only knows the information in one row of the table (the one that bears its name in the left column). The global view that is presented here is not available at any single point in the network.

Initial routing table at node A.(Table1)

Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

Initial distances stored at each node (global view).(Table2)

Information Stored at Node	Distance to Reach 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 may consider each row in Table 2 as a list of distances from one node to all other nodes, representing the current beliefs of that node. Initially, each node sets a cost of 1 to its directly connected neighbors and ∞ to all other nodes.
- Thus, A initially believes that it can reach B in one hop and that D is unreachable. The routing table stored at A reflects this set of beliefs and includes the name of the next hop that A would use to reach any reachable node. Initially, then, A's routing table would look like Table 1.
- The next step in distance-vector routing is that every node sends a message to its directly connected neighbors containing its personal list of distances.

- For example node F tells node A that it can reach node G at a cost of 1; A also knows it can reach F at a cost of 1, so it adds these costs to get the cost of reaching G by means of F.
- This total cost of 2 is less than the current cost of infinity, so A records that it can reach G at a cost of 2 by going through F. Similarly, A learns from C that D can be reache from C at a cost of 1; it adds this to the cost of reaching C (1) and decides that D can be reached via C at a cost of 2, which is better than the old cost of infinity.
- At the same time, A learns from C that B can be reached from C at a cost of 1, so it concludes that the cost of reaching B via C is 2.
- Since this is worse than the current cost of reaching B (1), this new information is ignored.

Destination	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

- At this point, A can update its routing table with costs and next hops for all nodes in the network. The result is shown in Table 3.
- In the absence of any topology changes, it only takes a few exchanges of information between neighbors before each node has a complete routing table. The process of getting consistent routing information to all the nodes is called *convergence*.
- Table 4 shows the final set of costs from each node to all other nodes when routing has converged. We must stress that there is no one node in the network that has all the information in this table—each node only knows about the contents of its own routing table. The beauty of a distributed algorithm like this is that it enables all nodes to achieve a consistent view of the network in the absence of any centralized authority.

Final distances stored at each node (global view)(Table 4)

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

- First we note that there are two different circumstances under which a given node decides to send a routing update to its neighbors. One of these circumstances is the *periodic* update. In this case, each node automatically sends an update message every so often, even if nothing has changed.
- This serves to let the other nodes know that this node is still running. It also makes sure that they keep getting information that they may need if their current routes become unviable. The frequency of these periodic updates varies from protocol to protocol, but it is typically on the order of several seconds to several minutes.
- The second mechanism, sometimes called a *triggered* update, happens whenever a node receives an update from one of its neighbors that causes it to change one of the routes in its routing table. That is, whenever a node's routing table changes, it sends an update to

its neighbors, which may lead to a change in their tables, causing them to send an update to their neighbors.

The second mechanism, sometimes called a *triggered* update, happens whenever a node receives an update from one of its neighbors that causes it to changeone of the routes in its routing table. That is, whenever a node's routing table changes,it sends an update to its neighbors, which may lead to a change in their tables, causing them to send an update to their neighbors.

Disadvantages

The routing tables for the network do not stabilize. This situation is known as the *count-to-infinity* problem.

One technique to improve the time to stabilize routing is called *split horizon*.

The idea is that when a node sends a routing update to its neighbors, it does not send those routes it learned from each neighbor back to that neighbor. For example, if B has the route (E, 2, A) in its table, then it knows it must have learned this route from A, and so whenever B sends a routing update to A, it does not include the route (E, 2) in that update.

In a stronger variation of split horizon, called *split horizon with poison reverse*, B actually sends that route back to A, but it puts negative information in the route to ensure that A will not eventually use B to get to E. For example, B sends the route (E, ∞) to A.

IMPLEMENTATION

The code that implements this algorithm is very straightforward; we give only some of the basics here. Structure Route defines each entry in the routing table, and constant MAX TTL specifies how long an entry is kept in the table before it is discarded.

```
#define MAX_ROUTES 128 /* maximum size of routing table */
#define MAX_TTL 120 /* time (in seconds) until route expires */
typedef struct {
NodeAddr Destination; /* address of destination */
NodeAddr NextHop; /* address of next hop */
int Cost; /* distance metric */
u_short TTL; /* time to live */
} Route;
int numRoutes = 0;
Route routingTable[MAX_ROUTES];
```

The routine that updates the local node's routing table based on a new route is given by mergeRoute. Although not shown, a timer function periodically scans the list of routes in the node's routing table, decrements the TTL (time to live) field of each route, and discards any routes that have a time to live of 0. Notice, however, that the TTL field is reset to MAX TTL any time the route is reconfirmed by an update message from a neighboring node.

```
void mergeRoute (Route *new)
{
int i;
for (i = 0; i < numRoutes; ++i)
{
if (new->Destination == routingTable[i].Destination)
{
if (new->Cost + 1 < routingTable[i].Cost)
{
/* found a better route: */
break;
} else if (new->NextHop == routingTable[i].NextHop) {
/* metric for current next hop may have changed: */
break;
} else {
/* route is uninteresting---just ignore it */return;}
```

```
} }
if (i == numRoutes)
{
/*this is a completely new route; is there room for it?*/
if (numRoutes < MAXROUTES)
{
++numRoutes;
} else {
/* can't fit this route in table so give up */
return;
} }
routingTable[i] = *new;
/* reset TTL */
routingTable[i].TTL = MAX_TTL;
/* account for hop to get to next node */
++routingTable[i].Cost;
}
```

Finally, the procedure updateRoutingTable is the main routine that calls merge-Route to incorporate all the routes contained in a routing update that is received from a neighboring node.

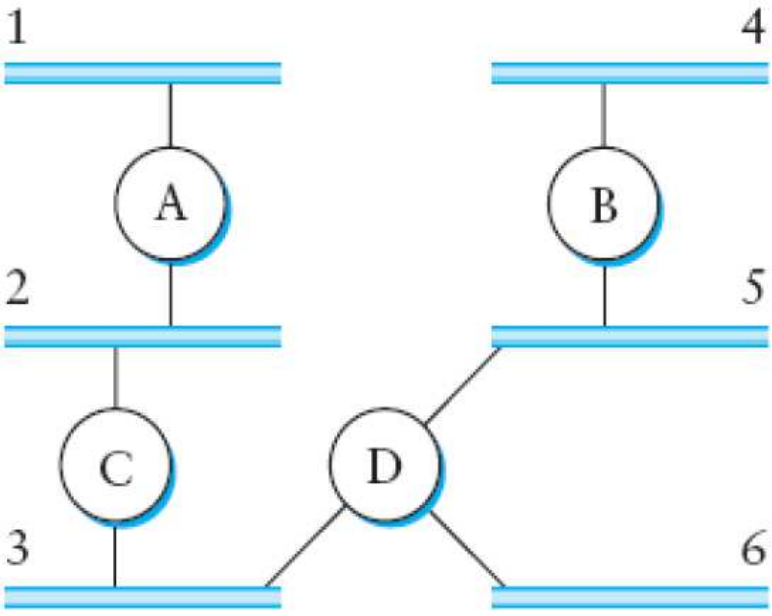
```
void
updateRoutingTable (Route *newRoute, int numNewRoutes)
{
int i;
for (i=0; i < numNewRoutes; ++i)
{
mergeRoute(&newRoute[i]);
} }
```

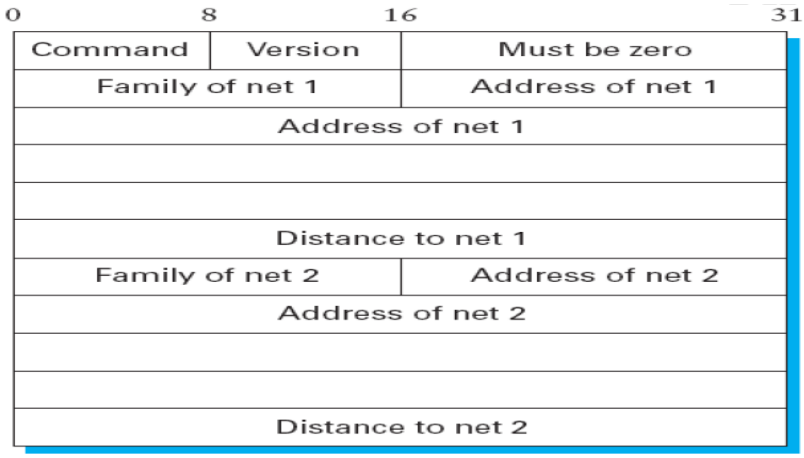
Routing Information Protocol (RIP)

RIP is the canonical example of a routing protocol built on the distance-vector algorithm.

In an internetwork, the goal of the routers is to learn how to forward packets to various *networks*. Thus, rather than advertising the cost of reaching other routers, the routers advertise the cost of reaching networks.

Example network running RIP





In the example network router C would advertise to router A the fact that it can reach networks 2 and 3 (to which it is directly connected) at a cost of 0; networks 5 and 6 at cost 1; and network 4 at cost 2.

Link State Routing

Link-state routing is the second major class of intradomain routing protocol. The starting assumptions for link-state routing are rather similar to those for distance vector routing**.**

Each node is assumed to be capable of finding out the state of the link to its neighbors (up or down) and the cost of each link. Again, we want to provide each node with enough information to enable it to find the least-cost path to any destination.

The basic idea behind link-state protocols is very simple: Every node knows how to reach its directly connected neighbors, and if we make sure that the totality of this knowledge is disseminated to every node, then every node will have enough knowledge of the network to build a complete map of the network.

Thus, link-state routing protocols rely on two mechanisms: reliable dissemination of link-state information, and the calculation of routes from the sum of all the accumulated link-state knowledge.

Reliable Flooding

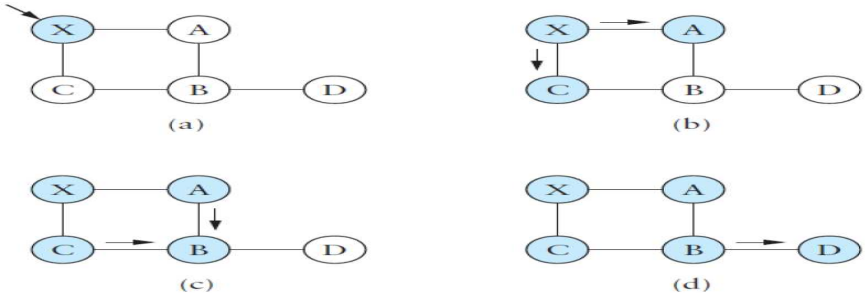
Reliable flooding is the process of making sure that all the nodes participating in the routing protocol get a copy of the link-state information from all the other nodes. As the term ―flooding‖ suggests, the basic idea is for a node to send its link-state information out on all of its directly connected links, with each node that receives this information forwarding it out on all of *its* links.

This process continues until the information has reached all the nodes in the network. More precisely, each node creates an update packet, also called a link-state packet (LSP), that contains the following information.

The ID of the node that created the LSP

- A list of directly connected neighbors of that node, with the cost of the link to each one
- A sequence number
- A time to live for this packet.

The first two items are needed to enable route calculation; the last two are used to make the process of flooding the packet to all nodes reliable. Reliability includes making sure that you have the most recent copy of the information, since there may be multiple, contradictory LSPs from one node traversing the network.



In the figure shows an LSP being flooded in a small network. Each node becomes shaded as it stores the new LSP. In Figure (a) the LSP arrives at node X, which sends it to neighbors A and in Figure 4.18.b) A and C do not send it back to X, but send it on to B. Since B receives two identical copies of the LSP, it will accept whichever arrived first and ignore the second as a duplicate. It then passes the LSP on to D, who has no neighbors to flood it to, and the process is complete.

Just as in RIP, each node generates LSPs under two circumstances. Either the expiry of a periodic timer or a change in topology can cause a node to generate a new LSP. However, the only topology-based reason for a node to generate an LSP is if one of its directly connected links or immediate neighbors has gone down.

The failure of a link can be detected in some cases by the link-layer protocol. The demise of a neighbor or loss of connectivity to that neighbor can be detected using periodic ―hello‖ packets.

Each node sends these to its immediate neighbors at defined intervals. If a sufficiently long time passes without receipt of a ―hello‖ from a neighbor, the link to that neighbor will be declared down, and a new LSP will be generated to reflect this fact.

One of the important design goals of a link-state protocol's flooding mechanism is that the newest information must be flooded to all nodes as quickly as possible, while old information must be removed from the network and not allowed to circulate. In addition, it is clearly desirable to minimize the total amount of routing traffic that is sent around the network; after all, this is just ―overhead‖ from the perspective of those who actually use the network for their applications. The next few paragraphs describe some of the ways that these goals are accomplished.

One easy way to reduce overhead is to avoid generating LSPs unless absolutely necessary. This can be done by using very long timers—often on the order of hours—for the periodic generation of LSPs. Given that the flooding protocol is truly reliable when topology changes, it is safe to assume that messages saying ―nothing has changed‖ do not need to be sent very often.

LSPs also carry a time to live. This is used to ensure that old link-state information is eventually removed from the network. A node always decrements the TTL of a newly received LSP before flooding it to its neighbors. It also ―ages‖ the LSP while it is stored in the node. When the TTL reaches 0, the node refloods the LSP with a TTL of 0, which is interpreted by all the nodes in the network as a signal to delete that LSP.

Route Calculation

In practice, each switch computes its routing table directly from the LSPs it has collected using a realization of Dijkstra's algorithm called the *forward search* algorithm. Specifically, each switch maintains two lists, known as Tentative and Confirmed. Each of these lists contains a set of entries of the form (Destination, Cost, NextHop).

The algorithm works as follows:

1 Initialize the Confirmed list with an entry for myself; this entry has a cost of 0.

2 For the node just added to the Confirmed list in the previous step, call it node Next, select its LSP.

3 For each neighbor (Neighbor) of Next, calculate the cost (Cost) to reach this Neighbor as the sum of the cost from myself to Next and from Next to Neighbor.

(a) If Neighbor is currently on neither the Confirmed nor the Tentative list, then add (Neighbor, Cost, NextHop) to the Tentative list, where NextHop is the direction I go to reach Next.

(b) If Neighbor is currently on the Tentative list, and the Cost is less than the currently listed cost for Neighbor, then replace the current entry with (Neighbor, Cost, NextHop), where NextHop is the direction I go to reach Next.

4 If the Tentative list is empty, stop. Otherwise, pick the entry from the Tentative list with the lowest cost, move it to the Confirmed list, and return to step 2.

Link-state routing: an example network.

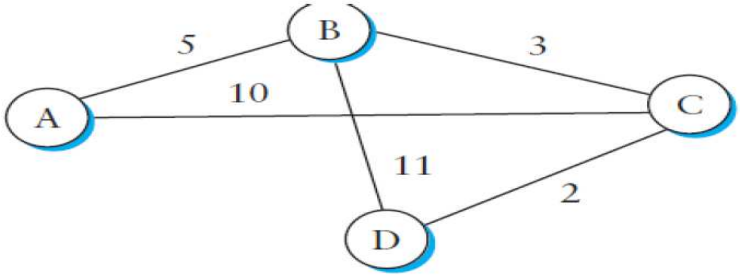


Table traces the steps for building the routing table for node D. We denote the two outputs of D by using the names of the nodes to which they connect, B and C. Note the way the algorithm seems to head off on false leads (like the 11-unit cost path to B that was the first addition to the Tentative list) but ends up with the least-cost paths to all nodes.

Step	Confirmed	Tentative	Comments
1	(D,0,-)		Since D is the only new member of the confirmed list, look at its LSP.
2	(D,0,-)	(B,11,B) (C,2,C)	D's LSP says we can reach B through B at cost 11, which is better than anything else on either list, so put it on Tentative list; same for C.
3	(D,0,-) (C,2,C)	(B,11,B)	Put lowest-cost member of Tentative (C) onto Confirmed list. Next, examine LSP of newly confirmed member (C).
4	(D,0,-) (C,2,C)	(B,5,C) (A,12,C)	Cost to reach B through C is 5, so replace (B,11,B). C's LSP tells us that we can reach A at cost 12.
5	(D,0,-) (C,2,C) (B,5,C)	(A,12,C)	Move lowest-cost member of Tentative (B) to Confirmed, then look at its LSP.
6	(D,0,-) (C,2,C) (B,5,C)	(A,10,C)	Since we can reach A at cost 5 through B, replace the Tentative entry.
7	(D,0,-) (C,2,C) (B,5,C) (A,10,C)		Move lowest-cost member of Tentative (A) to Confirmed, and we are all done.

The Open Shortest Path First Protocol (OSPF)

One of the most widely used link-state routing protocols is OSPF. The first word, ―Open,‖ refers to the fact that it is an open, nonproprietary standard, created under the auspices of the IETF. The ―SPF‖ part comes from an alternative name for linkstate routing. OSPF adds quite a number of features to the basic link-state algorithm described above, including the following:

Authentication of routing messages: This is a nice feature, since it is all too common for some misconfigured host to decide that it can reach every host in the universe at a cost of 0. When the host advertises this fact, every router in the surrounding neighborhood updates its forwarding tables to point to that host, and said host receives a vast amount of data that, in reality, it has no idea what to do with. This is not a strong enough form of authentication to prevent dedicated malicious users, but it alleviates many problems caused by misconfiguration.

Additional hierarchy: Hierarchy is one of the fundamental tools used to make systems more scalable. OSPF introduces another layer of hierarchy into routing by allowing a domain to be partitioned into *areas*. This means that a router within a domain does not necessarily need to know how to reach every network within that domain—it may be sufficient for it to know only how to get to the right area. Thus, there is a reduction in the amount of information that must be transmitted to and stored in each node.

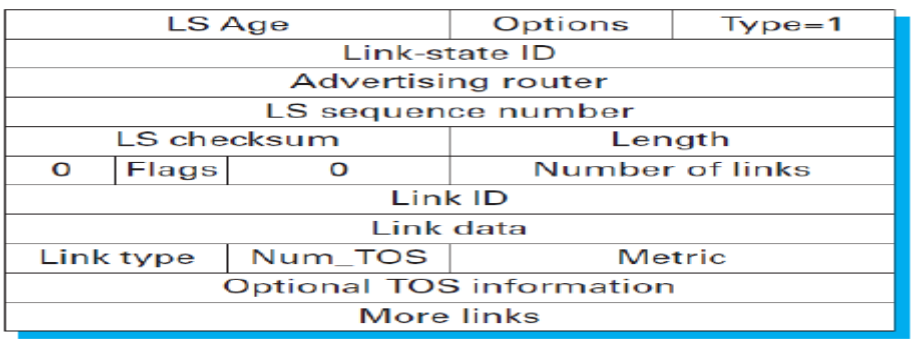
Load balancing: OSPF allows multiple routes to the same place to be assigned the same cost and will cause traffic to be distributed evenly over those route.

OSPF header format.

The Version field is currently set to 2, and the Type field may take the values 1 through 5. The SourceAddr identifies the sender of the message, and the AreaId is a 32-bit identifier of the area in which the node is located. The entire packet, except the authentication data, is protected by a 16-bit checksum using the same algorithm as the IP header. The Authentication type is 0 if no authentication is used; otherwise it may be 1, implying a simple password is used, or 2, which indicates that a cryptographic authentication checksum. In the latter cases the Authentication field carries the password or cryptographic checksum.

Of the five OSPF message types, type 1 is the ―hello‖ message, which a router sends to its peers to notify them that it is still alive and connected as described above. The remaining types are used to request, send, and acknowledge the receipt of linkstate messages. The basic building block of link-state messages in OSPF is known as the link-state advertisement (LSA).

OSPF link-state advertisement.



The above figure shows the packet format for a ―type 1‖ link-state advertisement.

Type 1 LSAs advertise the cost of links between routers. Type 2 LSAs are used to advertise networks to which the advertising router is connected, while other types are used to support additional hierarchy as described in the next section. Many fields in the LSA should be familiar from the preceding discussion.

The LS Age is the equivalent of a time to live, except that it counts up and the LSA expires when the age reaches a defined maximum value. The Type field tells us that this is a type 1 LSA. In a type 1 LSA, the Link-state ID and the Advertising router field are identical. Each carries a 32-bit identifier for the router that created this LSA. While a number of assignment strategies may be used to assign this ID, it is essential that it be unique in the routing domain and that a given router consistently uses the same router ID.

One way to pick a router ID that meets these requirements would be to pick the lowest IP address among all the IP addresses assigned to that router. (Recall that a router may have a different IP address on each of its interfaces.).The LS sequence number is used exactly as described above, to detect old or duplicate LSAs. The LS checksum it is of course used to verify that data has not been corrupted. It covers all fields in the packet except LS Age, so that it is not necessary to recomputed a checksum every time LS Age is incremented. Length is the length in bytes of the Now we get to the actual link-state information.

This is made a little complicated by the presence of TOS (type of service) information. Ignoring that for a moment, each link in the LSA is represented by a Link ID, some Link Data, and a metric. The first two of these fields identify the link; a common way to do this would be to use the router ID of the router at the far end of the link as the Link ID, and then use the Link Data to disambiguate among multiple parallel links if necessary. The metric is of course the cost of the link.

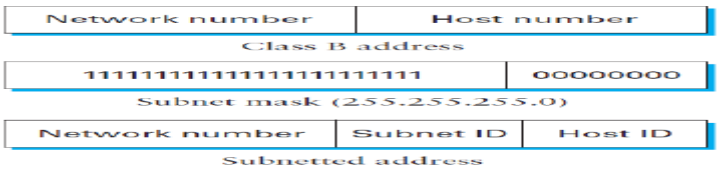
Type tells us something about the link, for example, if it is a point-to-point link. The TOS information is present to allow OSPF to choose different routes for IP packets based on the value in their TOS field. Instead of assigning a single metric to a link, it is possible to assign different metrics depending on the TOS value of the data.

SUBNETTING

- The original intent of IP addresses was that the network part would uniquely identify exactly one physical network.
- It has two drawbacks.
- Address assignment inefficiency.
- Assigning network number to every physical network, uses the IP address space potentially much faster.
- Assigning many network numbers has another drawback that becomes apparent when you think about routing.
- *Subnetting provides an elegantly simple way to reduce the total number of network* numbers that are assigned.
- The idea is to take a single IP network number and allocate the IP addresses with that network number to several physical networks, which are now referred to as *subnets.*
- First, the subnets should be close to each other.
- This is because at a distant point in the Internet, they will all look like a single network, having only one network number between them.
- This means that a router will only be able to select one route to reach any of the subnets, so they had better all be in the same general direction.
- A perfect situation in which to use subnetting is a large campus or corporation that has many physical networks.
- From outside the campus, all you need to know to reach any subnet inside the campus is where the campus connects to the rest of the Internet.

- The mechanism by which a single network number can be shared among multiple networks involves configuring all the nodes on each subnet with a *subnet mask.*
- With simple IP addresses, all hosts on the same network must have the same network number.
- The subnet mask enables us to introduce a *subnet number; all hosts on the same* physical network will have the same subnet number, which means that hosts may be on different physical networks but share a single network number.
- For example, suppose that we want to share a single class B address among several physical networks. We could use a subnet mask of 255.255.255.0.
- (Subnet masks are written down just like IP addresses; this mask is therefore all 1s in the upper 24 bits and 0s in the lower 8 bits.)
- In effect, this means that the top 24 bits(where the mask has 1s) are now defined to be the network number, and the lower 8 bits (where the mask has 0s) are the host number.
- Since the top 16 bits identify the network in a class B address, we may now think of the address as having not two parts but three: a network part, a subnet part, and a host part.

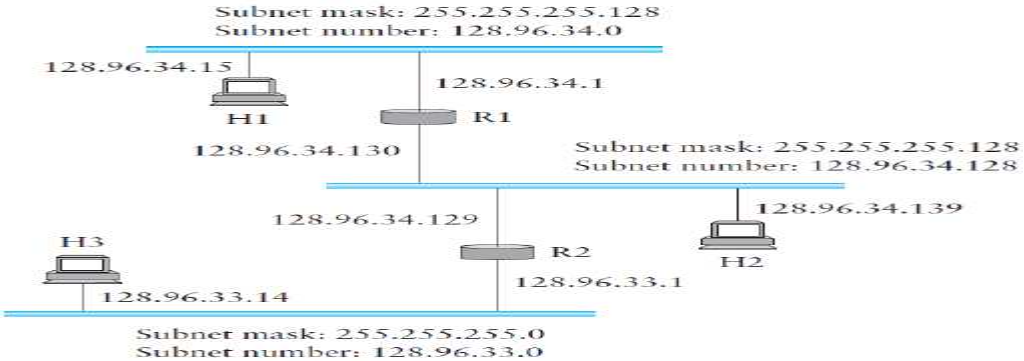
SUBNET ADDRESSING



- What subnetting means to a host is that it is now configured with both an IP address and a subnet mask for the subnet to which it is attached.
- For example, host H1 in Figure 4.26 is configured with an address of 128.96.34.15 and a subnet mask of 255.255.255.128.
- The bitwise AND of these two numbers defines the subnet number of the host and of all other hosts on the same subnet.
- In this case, 128.96.34.15 AND 255.255.255.128 equals 128.96.34.0, so this is the subnet number for the topmost subnet.

FORWARDING PACKET THROUGH SUBNET

- When the host wants to send a packet to a certain IP address, the first thing it does is to perform a bitwise AND between its own subnet mask and the destination IP address.
- If the results are not equal, the packet needs to be sent to a router to be forwarded to another subnet.
- For example, if H1 is sending to H2, then H1 ANDs its subnet mask (255.255.255.128) with the address for H2 (128.96.34.139) to obtain 128.96.34.128. This does not match the subnet number for H1 (128.96.34.0) so H1 knows that H2 is on a different subnet. Since H1 cannot deliver the packet toH2 directly over the subnet, it sends the packet to its default router R1.



- To support subnetting, the routing table must now hold entries of the form SubnetNumber, SubnetMask, NextHop. To find the right entry in the table,the router ANDs the packet's destination address with the SubnetMask for each entry in turn; if the result matches the SubnetNumber of the entry, then this is the right entry to use, and it forwards the packet to the next hop router indicated.

Example Subnetting Table

SubnetNumber	SubnetMask	NextHop
128.96.34.0	255.255.255.128	Interface 0
128.96.34.128	255.255.255.128	Interface 1
128.96.33.0	255.255.255.0	R2

We can now describe the datagram forwarding algorithm in the following way:

```
D = destination IP address
for each forwarding table entry (SubnetNumber, SubnetMask, NextHop)
   D1 = SubnetMask & D
   if D1 = SubnetNumber
      if NextHop is an interface
         deliver datagram directly to destination
      else
         deliver datagram to NextHop (a router)
```

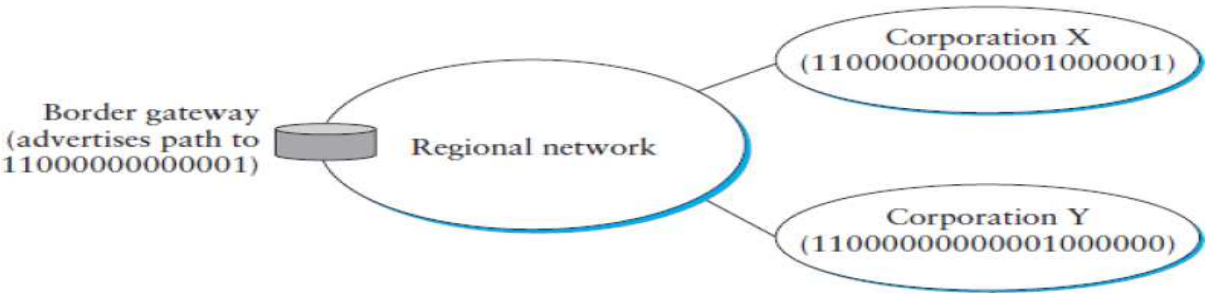
Classless Routing (CIDR)

- Classless interdomain routing (CIDR, pronounced ―cider‖) is a technique that addresses two scaling concerns in the Internet: the growth of backbone routing tables as more and more network numbers need to be stored in them, and the potential for the 32-bit IP address space to be exhausted well before the four-billionth host is attached to the Internet.
- This address space exhaustion is called address assignment inefficiency.
- The inefficiency arises because the IP address structure, with class A, B, and C addresses, forces us to hand out network address space in fixed-sized chunks of three very different sizes.
- A network with two hosts needs a class C address, giving an address assignment efficiency of 2/255 = 0.78%; a network with 256 hosts needs a class B address, for an efficiency of only 256/65,535 = 0.39%.
- Even though subnetting can help us to assign addresses carefully, it does not get around the fact that any autonomous system with more than 255 hosts, or an expectation of eventually having that many, wants a class B address.
- As it turns out, exhaustion of the IP address space centers on exhaustion of the class B network numbers.
- One way to deal with that would seem to be saying no to any AS that requests a class B address unless they can show a need for something close to 64K addresses, and instead giving them an appropriate number of class C addresses to cover the expected number of hosts. Since we would now be handing out address space in chunks of 256 addresses at a time, we could more accurately match the amount of address space consumed to the size of the AS.
- For any AS with at least 256 hosts (which means the majority of ASs), we can guarantee an address utilization of at least 50%, and typically much more.
- This solution, however, raises a problem that is at least as serious: excessive storage requirements at the routers. If a single AS has, say, 16 class C network numbers assigned to it,

that means every Internet backbone router needs 16 entries in its routing tables for that AS. This is true even if the path to every one of those networks is the same.

- If we had assigned a class B address to the AS, the same routing information could be stored in one table entry. However, our address assignment efficiency would then be only 16 × 255/65,536 = 6.2%.
- CIDR, therefore, tries to balance the desire to minimize the number of routes that a router needs to know against the need to hand out addresses efficiently.
- To do this, CIDR helps us to *aggregate* routes. That is, it lets us use a single entry in a forwarding table to tell us how to reach a lot of different networks.
- From the name, it does this by breaking the rigid boundaries between address classes.
- To understand how this works, consider our hypothetical AS with 16 class C network numbers. Instead of handing out 16 addresses at random, we can hand out a block of *contiguous* class C addresses.
- Suppose we assign the class C network numbers from 192.4.16 through 192.4.31. Observe that the top 20 bits of all the addresses in this range are the same (11000000 00000100 0001). Thus, what we have effectively created is a 20-bit network number—something that is between a class B network number and a class C number in terms of the number of hosts that it can support.
- In other words, we get both the high address efficiency of handing out addresses in chunks smaller than a class B network and a single network prefix that can be used in forwarding tables.
- Observe that for this scheme to work, we need to hand out blocks of class C addresses that share a common prefix, which means that each block must contain a number of class C networks that is a power of two.
- All we need now to make CIDR solve our problems is a routing protocol that can deal with these ―classless‖ addresses, which means that it must understand that a network number may be of any length.
- Modern routing protocols do exactly that. The network numbers that are carried in such a routing protocol are represented simply by _length, value_ pairs, where the length gives the number of bits in the network prefix—20 in the above example.
- Consider the example Figure The two corporations served by the provider network have been assigned adjacent 20- bit network prefixes.

Route Aggregate with CIDR

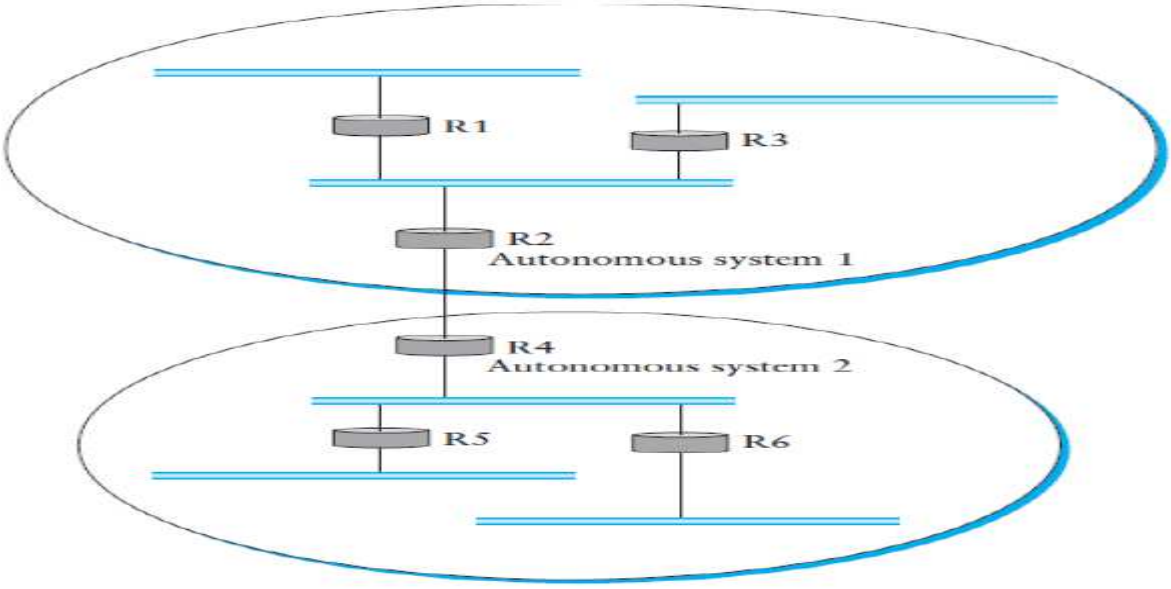


- Since both of the corporations are reachable through the same provider network, it can advertise a single route to both of them by just advertising the common 19-bit prefix they share.
- In general, it is possible to aggregate routes repeatedly if addresses are assigned carefully.
- This means that we need to pay attention to which provider a corporation is attached to before assigning it an address if this scheme is to work. One way to accomplish that is to assign a portion of address space to the provider and then to let the network provider assign addresses from that space to its customers.

Interdoain Routing (BGP)

- At the beginning of this section we introduced the notion that the Internet is organized as autonomous systems, each of which is under the control of a single administrative entity.

A network with two autonomous systems.



- A corporation's complex internal network might be a single AS, as may the network of a single Internet service provider
- The basic idea behind autonomous systems is to provide an additional way to hierarchically aggregate routing information in a large internet, thus improving scalability.
- We now divide the routing problem into two parts: routing within a single autonomous system and routing between autonomous systems. Since another name for autonomous systems in the Internet is routing *domains*, we refer to the two parts of the routing problem as interdomain routing and intradomain routing.
- In addition to improving scalability, the AS model decouples the intradomain routing that takes place in one AS from that taking place in another.
- Thus, each AS can run whatever intradomain routing protocols it chooses. It can even use static routes or multiple protocols if desired. The interdomain routing problem is then one of having different Ass share reachability information with each other.
- One feature of the autonomous system idea is that it enables some ASs to dramatically reduce the amount of routing information they need to care about by using *default routes*.

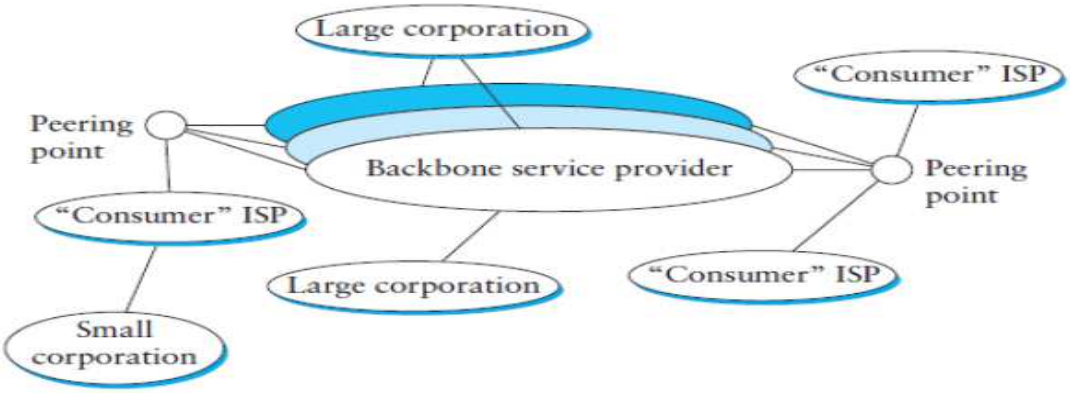
If a corporate network is connected to the rest of the Internet by a single router (this router is typically called a *border router* since it sits at the boundary between the AS and the rest of the Internet), then it is pretty easy for a host or router *inside* the autonomous system to figure out where it should send packets that are headed for a destination *outside* of this AS—they first go to the AS's border router.

This is the default route Similarly, a regional Internet service provider can keep track of how to reach the networks of all its directly connected customers and can have a default route to some other provider (typically a backbone provider) for everyone else.

- There have been two major interdomain routing protocols in the recent history of the Internet. The first was the Exterior Gateway Protocol (EGP). EGP had a number of limitations, perhaps the most severe of which was that it constrained the topology of the Internet rather significantly. EGP basically forced a treelike topology onto the Internet, or to be more precise, it was designed when the Internet had a treelike topology.

❖ The replacement for EGP is the Border Gateway Protocol (BGP), which is in its fourth version at the time of this writing (BGP-4). BGP is also known for being rather complex. This section presents the highlights of BGP-4. As a starting position, BGP assumes that the Internet is an arbitrarily interconnected set of ASs. This model is clearly general enough to accommodate non-treestructured internetworks, like the simplified picture of today's multibackbone Internet shown in the Figure.

Today's multibackbone Internet.



Today's Internet consists of an interconnection of multiple backbone networks (they are usually called *service provider networks*, and they are operated by private companies rather than the government), and sites are connected to each other in arbitrary ways.
Some large corporations connect directly to one or more of the backbones, while others connect to smaller, non backbone service providers. Many service providers exist mainly to provide service to ―consumers‖ (i.e., individuals with PCs in their homes), and these providers must also connect to the backbone providers.

Often many providers arrange to interconnect with each other at a single ―peering point.‖ Given this rough sketch of the Internet, if we define *local traffic* as traffic that originates at or terminates on nodes within an AS, and *transit traffic* as traffic that passes through an AS, we can classify ASs into three types:

■ *Stub AS:* an AS that has only a single connection to one other AS; such an AS will only carry local traffic. The small corporation is an example of a stub AS.

■ *Multihomed AS:* an AS that has connections to more than one other AS bu that refuses to carry transit traffic; for example, the large corporation.

■ *Transit AS:* an AS that has connections to more than one other AS and that is designed to carry both transit and local traffic, such as the backbone providers.

There are a few reasons why interdomain routing is hard.

The first is simply a matter of scale. An Internet backbone router must be able to forward any packet destined anywhere in the Internet

The second challenge in interdomain routing arises from the autonomous nature of the domains. Note that each domain may run its own interior routing protocols and use any scheme it chooses to assign metrics to paths. This means that it is impossible to calculate meaningful path costs for a path that crosses multiple Ass.

The third challenge involves the issue of trust. Provider A might be unwilling to believe certain advertisements from provider B for fear that provider B will advertise erroneous routing information.

When configuring BGP, the administrator of each AS picks at least one node to be a ―BGP speaker,‖ which is essentially a spokesperson for the entire AS. That BGP speaker establishes BGP sessions to other BGP speakers in other ASs.

These sessions are used to exchange reachability information among ASs. In addition to the BGP speakers, the AS has one or more border ―gateways,‖ which need not be the same as the speakers.

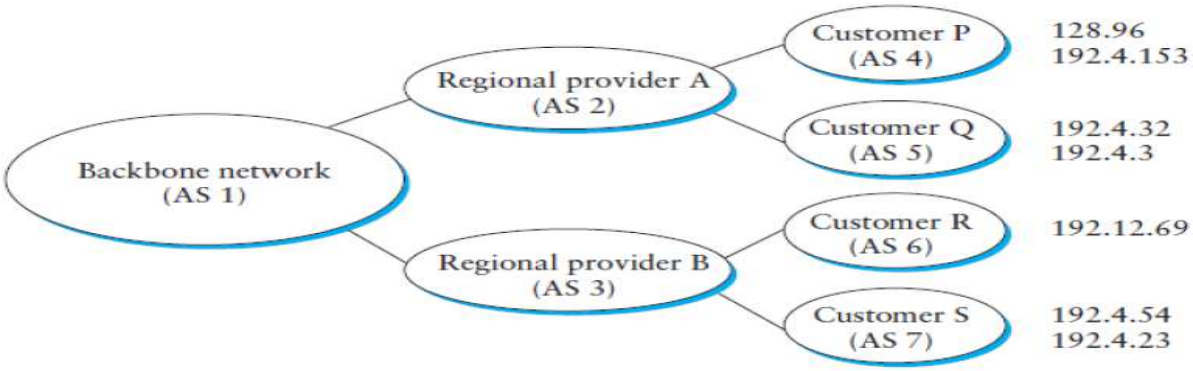
The border gateways are the routers through which packets enter and leave the AS.

In the example network R2 and R4 would be border gateways.

The important point to understand here is that, in the context of interdomain routing, a border gateway is simply an IP router that is charged with the task of forwarding packets between ASs.

Unlike link state and distance vector routing protocols BGP advertises *complete paths* as an enumerated list of ASs to reach a particular network.

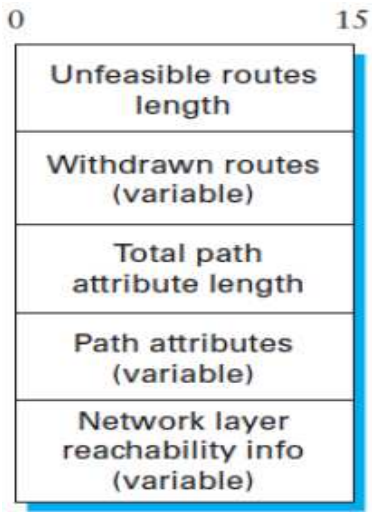
Example of a network running BGP.



Assume that the providers are transit networks, while the customer networks are stubs. A BGP speaker for the AS of provider A (AS 2) would be able to advertise reachability information for each of the network numbers assigned to customers P and Q. Thus, it would say, in effect, ―The networks 128.96, 192.4.153, 192.4.32, and 192.4.3 can be reached directly from AS 2.‖ The backbone network, on receiving this advertisement, can advertise, ―The networks 128.96, 192.4.153, 192.4.32, and 192.4.3 can be reached along the path (AS 1, AS 2).‖ Similarly, it could advertise, ―The networks 192.12.69, 192.4.54, and 192.4.23 can be reached along the path (AS 1, AS 3).‖

In addition to advertising paths, BGP speakers need to be able to cancel previously advertised paths if a critical link or node on a path goes down. This is done with a form of negative advertisement known as a *withdrawn route.*

BGP-4 update packet format.



IPV6

Accommodate scalable routing and adressing the ip address should be:

- support for real-time services
- security support
- autoconfiguration (i.e., the ability of hosts to automatically configure themselves with such information as their own IP address and domain name)
- enhanced routing functionality, including support for mobile hosts.

Addresses and Routing

- IPv6 provides a 128-bit address space.
- IPv6 can address 3.4×10^38 nodes.
- IPv6 address space is predicted to provide over 1500 addresses per square foot of the earth's surface

Address Space Allocation

- IPv6 addresses do not have classes.
- address space is subdivided in based on the leading bits.

Prefix	Use
0000 0000	Reserved
0000 0001	Unassigned
0000 001	Reserved for NSAP allocation
0000 010	Reserved for IPX allocation
0000 011 0000 1 0001	Unassigned Unassigned Unassigned
001	Aggregatable Global Unicast Addresses
010 011 100 101 110 1110 1111 0 1111 10 1111 110 1111 1110 0	Unassigned Unassigned Unassigned Unassigned Unassigned Unassigned Unassigned Unassigned Unassigned Unassigned
1111 1110 10	Link local use addresses
1111 1110 11	Site local use addresses
1111 1111	Multicast addresses

Address prefix assignments for IPv6

Address prefix assignments for IPv6

- large chunks of address space have been left unassigned to allow for future growth and new features.
- 0000 001-NSAP(Network Service Access Point) addresses are used by the ISO protocols.
- 0000 010-IPX addresses are used by Novell's network-layer protocol.
- —link local use‖-enable a host to construct an address that will work on the network to which it is connected without being concerned about global uniqueness of the address.
- —site local use‖-addresses are intended to allow valid addresses to be constructed on a site that is not connected to the larger Internet; again, global uniqueness need not be an issue.
- multicast address space- is for multicast, thereby serving the same role as class D addresses in IPv4.

Address Notation

- The standard representation is x:x:x:x:x:x:x:x, where each ―x‖ is a hexadecimal representation of a 16-bit piece of the address.
- Example IPV6 Address format 47CD:1234:4422:ACO2:0022:1234:A456:0124
- There are some special notations that may be helpful in certain circumstances. For example, an address with a large number of contiguous 0s can be written more compactly by omitting all the 0 fields.

Example47CD:0000:0000:0000:0000:0000:A456:0124 could be written 47CD::A456:0124

- This form of shorthand can only be used for one set of contiguous 0s in an address to avoid ambiguity.

Mapping from ipv4 to ipv6

- IPv4 address was 128.96.33.81 could be written as::FFFF:128.96.33.81.
- The last 32 bits are written in IPv4 notation, rather than as a pair of hexadecimal numbers separated by a colon. Note that the double colon at the front indicates the leading 0s.

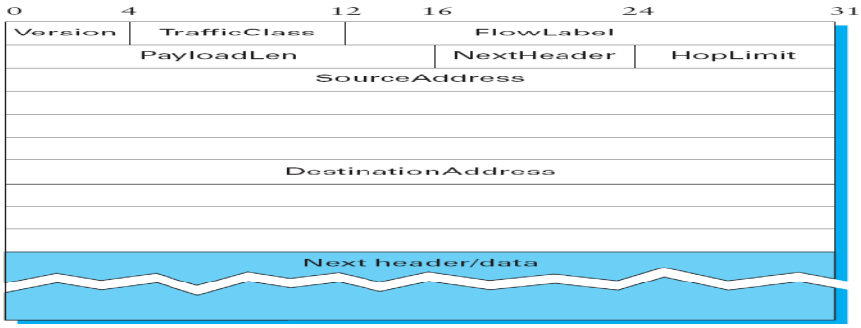
Aggregatable Global Unicast Addresses

- unicast address allocation plan that determines how addresses beginning with the 001 prefix will be assigned to service providers, autonomous systems, networks, hosts, and routers.
- consider a nontransit AS (i.e., a stub or multihomed AS) as a *subscriber*
- *consider* a transit AS as a *provider.*
- *Providers subdivided into direct and indirect.*
- *Direct providers- are directly connected to subscribers.*
- Indirect providers- are not connected directly to subscribers, and are often known as *backbone networks.*
- Similar to CIDR, the goal of the IPv6 address allocation plan is to provide aggregation of routing information to reduce the burden on intradomain routers.
- It uses address prefix—to aggregate reachability information to a large number of networks.

An IPv6 provider-based unicast address

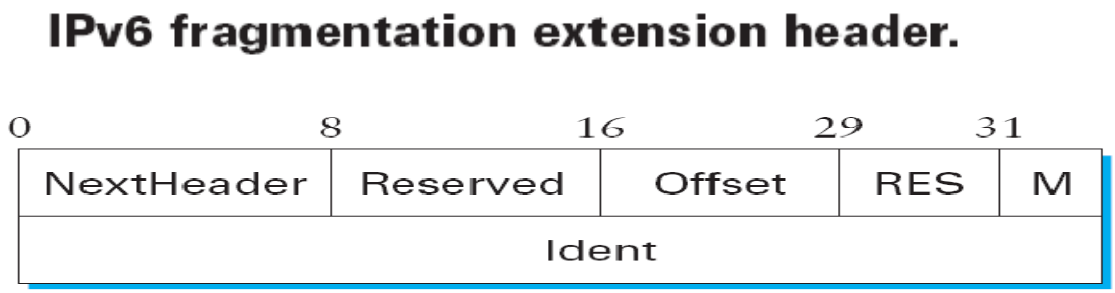
The RegistryID might be an identifier assigned to a European address registry, with different IDs assigned to other continents or countries**.**

Packet Format(IPV6 HEADER FORMAT)



- Version field-which is set to 6 for IPv6.
- The TrafficClass and FlowLabel fields- both relate to quality of service issues.
- The PayloadLen field -gives the length of the packet excluding the IPv6 header,measured in bytes.
- The NextHeader field cleverly replaces both the IP options and the Protocol field of IPv4. If options are required, then they are carried in one or more special headers following the IP header, and this is indicated by the value of the NextHeader field.
- If there are no special headers, the NextHeader field is the demuxkey identifying the higher-level protocol running over IP (e.g., TCP or UDP); that is, it serves the same purpose as the IPv4 Protocol field.
- The HopLimit field is simply the TTL of IPv4.
- Finally, the bulk of the header is taken up with the source and destination addresses,each of which is 16 bytes (128 bits) long.
- IPv6 addresses are four times longer than those of IPv4.

Ipv6-extension headers



- Assuming it is the only extension header present, then the NextHeader field of the IPv6 header would contain the value 44, which is the value assigned to indicate the fragmentation header.
- The NextHeader field of the fragmentation header itself contains a value describing the header that follows it. Again, assuming no other extension headers are present, then the next header might be the TCP header, which results in NextHeader containingthe value 6, just as the Protocol field would in IPv4.
- If the fragmentation header were followed by, say, an authentication header, then the fragmentation header's NextHeaderfield would contain the value 51.

Autoconfiguration

- One goal of IPv6, therefore, is to provide support for autoconfiguration, sometimes referred to as ―plug-and-play‖ operation.
- The longer address format in IPv6 helps provide a useful, new form
- of autoconfiguration called *stateless autoconfiguration*, which does not require a server.
- The autoconfiguration problem is subdivided into two parts:
- Obtain an interface ID that is unique on the link to which the host is attached.
- Obtain the correct address prefix for this subnet.
- The first part turns out to be rather easy, since every host on a
  - link must have a unique link-level address.
  - For example, all hosts on an Ethernet have a unique 48-bit Ethernet address.

  This can be turned into a valid link local use address by adding the appropriate prefix from Table 4.11 (1111 1110 10) followed by enough 0s to makeup 128 bits.

Advanced Routing Capabilities

- Another of IPv6's extension headers is the routing header. In the absence of this header, routing for IPv6 differs very little from that of IPv4 under CIDR.
- The routing header contains a list of IPv6 addresses that represent nodes or topological areas that the packet should visit en route to its destination.
- A topological area may be, for example, a backbone provider's network.
- To provide the ability to specify topological entities rather than individual nodes, IPv6 defines an *anycast address.*
- *An anycast* address is assigned to a set of interfaces, and packets sent to that address will go to the ―nearest‖ of those interfaces, with nearest being determined by the routing protocols.
- For example, all the routers of a backbone provider could be assigned a single anycast address, which would be used in the routing header.

Other Features

- The primary motivation behind the development of IPv6 was to support the continued growth of the Internet.

Primary features

- autoconfiguration and
- source-directed routing.

Additional features

- mobility
- Network security
- New service model

MULTICASTING

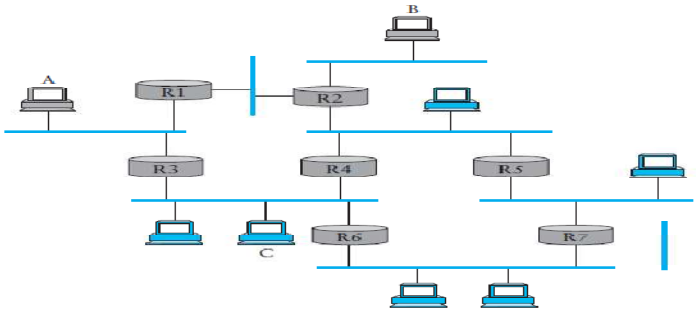
- The motivation for developing multicast is that there are applications that want to send a packet to more than one destination host.
- Instead of forcing the source host to send a separate packet to each of the destination hosts, we want the source to be able to send a single packet to a *multicast address, and for the network—or internet,* in this case—to deliver a copy of that packet to each of a group of hosts.
- Hosts can then choose to join or leave this group at will, without synchronizing or negotiating with other members of the group.
- Also, a host may belong to more than one group at a time.
- Hosts join multicast groups using a protocol called Internet Group Management Protocol (IGMP).
- Two types of multicast routing
  - Distance-vector Routing .
  - Link-state Routing.

Link-State Multicast

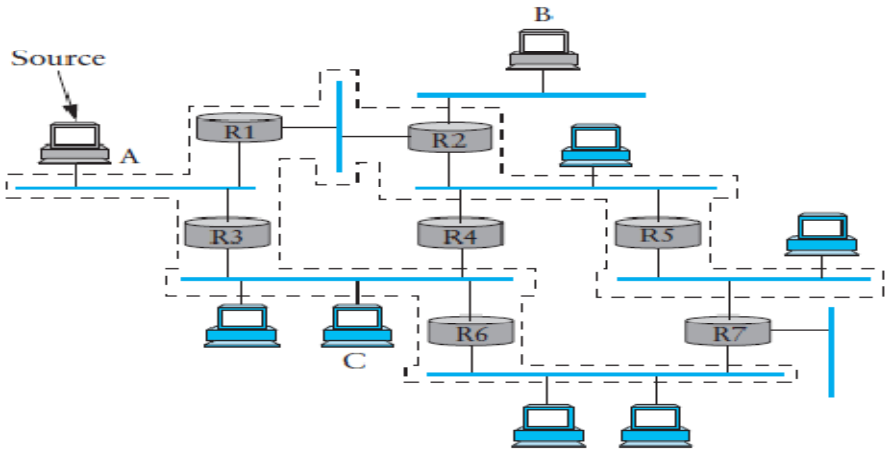
- Recall that in link-state routing, each router monitors the state of its
- directly connected links and sends an update message to all of the other routers whenever the state changes.
- Since each router receives enough information to reconstruct the entire topology of the network, it is able to use Dijkstra's algorithm to compute the shortest-path spanning tree rooted at itself and reaching all possible destinations.
- The router uses this tree to determine the best next hop for each packet it forwards.
- The only question is how each router determines which groups have members on which links.

- the solution is to have each host periodically announce to the LAN the groups to which it belongs.
- The router simply monitors the LAN for such announcements. Should such announcements stop arriving after a period of time, the router then assumes that the host has left the group.
- Given full knowledge of which groups have members on which links, each router is able to compute the *shortest-path multicast tree from any source to any group*, again using Dijkstra's algorithm.
- —Keep in mind that each router must potentially keep a separate shortest-path multicast tree from every source to every group. This is obviously very expensive.‖

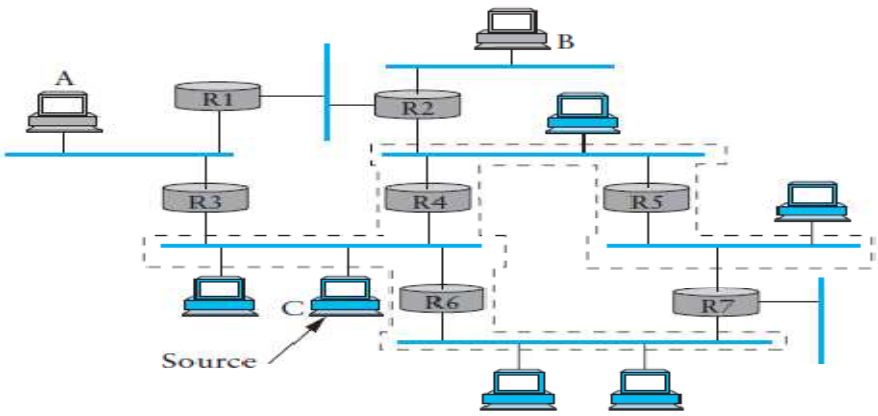
Example internet with members of group G in color.



Example shortest-path multicast tree (From source A to Group G)



Example shortest-path multicast tree (From source C to Group G)



Distance-Vector Multicast

- Adding multicast to the distance-vector algorithm is a bit trickier because the routers do not know the entire topology of the internet.
- Instead, recall that each router maintains a table of Destination, Cost, NextHop tuples, and exchanges a list of Destination, Cost pairs with its directly connected neighbors.
- Extending this algorithm to support multicast is a two-stage process. First, we need to design a broadcast mechanism that allows a packet to be forwarded to all the networks on the internet.
- Second, we need to refine this mechanism so that it prunes back networks that do not have hosts that belong to the multicast group.

Reverse-Path Broadcast (RPB)
Reverse-Path Multicast (RPM)

Reverse-Path Broadcast (RPB)

- Each router knows that the current shortest path to a given destination goes through NextHop.
- Thus, whenever it receives a multicast packet from source S, the router forwards the packet on all outgoing links (except the one on which the packet arrived) if and only if the packet arrived over the link that is on the shortest path to S.

(i.e., the packet came *from the NextHop associated with S in the routing table).*
This strategy effectively floods packets outward from S, but does not loop packets back toward S
There are two major shortcomings to this approach

- The first is that it truly floods the network; it has no provision for avoiding LANs that have no members in the multicast group.
- The second limitation is that a given packet will be forwarded over a LAN by each of the routers connected to that LAN. This is due to the forwarding strategy of flooding packets on all links other than the one on which the packet arrived, without regard to whether or not those links are part of the shortest-path tree rooted at the source.

(The solution to this second limitation is to eliminate the duplicate broadcast packets that are generated when more than one router is connected to a given LAN.)

RPB(Parent Router)

- One way to do this is to designate one router as the ―parent‖ router for each link, relative to the source, where only the parent router is allowed to forward multicast packets from that source over the LAN.
- The router that has the shortest path to source S is selected as the parent; a tie between two routers would be broken according to which router has the smallest address.
- Notice that this refinement requires that each router keep, for each source, a bit for each of its incident links indicating whether or not it is the parent for that source/link pair.
- Keep in mind that in an internet setting, a ―source‖ is a network, not a host, since an internet router is only interested in forwarding packets between networks.

Reverse-Path Multicast (RPM)

- RPB implements shortest-path broadcast.
- We now want to prune the set of networks that receives each packet addressed to group G to exclude those that have no hosts that are members of G.
- This can be accomplished in two stages.
- First, we need to recognize when a *leaf network has no group members. Determining that a network* is a leaf is easy—if the parent router as described in RPB is the only router on the network, then the network is a leaf.

- Determining if any group members reside on the network is accomplished by having each host that is a member of group G periodically announce this fact over the network, as described in our earlier description of link-state multicast.
- The router then uses this information to decide whether or not to forward a multicast packet addressed to G over this LAN.
- The second stage is to propagate this ―no members of G here‖ information up the shortest-path tree.
- This is done by having the router augment the Destination, Cost pairs it sends to its neighbors with the set of groups for which the leaf network is interested in receiving multicast packets.
- This information can then be propagated from router to router, so that for each of its links, a given router knows for what groups it should forward multicast packets.
- Note that including all of this information in the routing update is a fairly expensive thing to do.

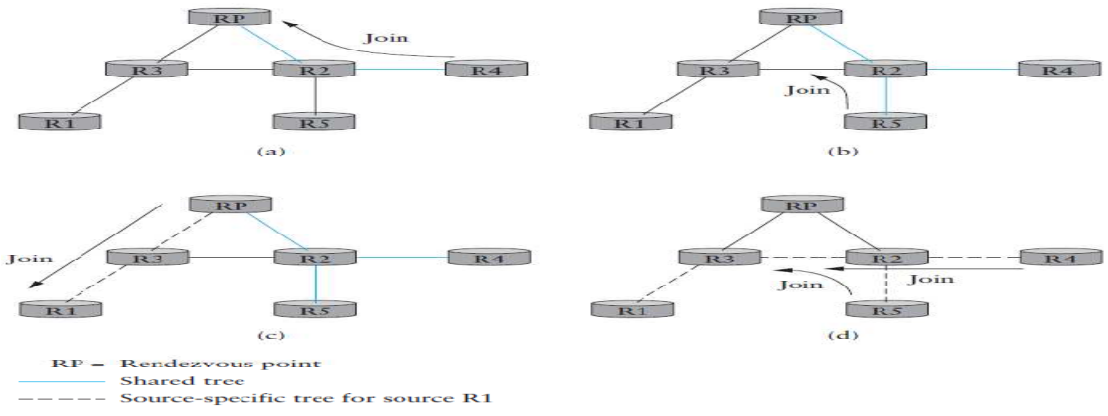
Protocol Independent Multicast (PIM)

- Scalability.(problem of previous protocols)
- This situation is sufficiently common that PIM divides the problem space into ―sparse mode‖ and ―dense mode.‖
- In PIM sparse mode (PIM-SM), routers explicitly join and leave the multicast group using PIM protocol messages known as Join and Prune messages.
- The question that arises is where to send those messages. To address this, PIM assigns a *rendezvous point (RP) to each group. In general, a number of routers in a domain are configured* to be candidate RPs, and PIM defines a set of procedures by which all the routers in a domain can agree on the router to use as the RP for a given group.
- These procedures are rather complex, as they must deal with a wide variety of scenarios, such as the failure of a candidate RP and the partitioning of a domain into two separate networks due to a number of link or node failures.
- For the rest of this discussion, we assume that all routers in a domain know the unicast IP address of the RP for a given group.
- A multicast forwarding tree is built as a result of routers sending Join messages to the RP. PIM-SM allows two types of trees to be constructed: a *shared tree, which* may be used by all senders, and a *source-specific tree, which may be used only by* a specific sending host.

The normal mode of operation creates the shared tree first, followed by one or more source-specific trees if there is enough traffic to warrant it.

Because building trees installs state in the routers along the tree, it is important that the default is to have only one tree for a group, not one for every sender to a group.

PIM operation: (a) R4 sends Join to RP and joins shared tree. (b) R5 joins shared tree. (c) RP builds source-specific tree to R1 by sending Join to R1. (d) R4 and R5 build source-specific tree to R1 by sending Joins to R1.



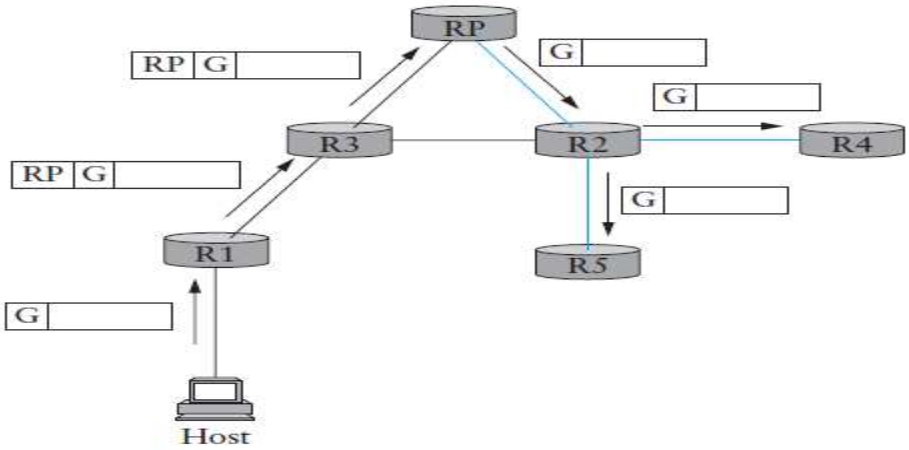
When a router sends a Join message toward the RP for a group G, it is sent using normal IP unicast transmission. This is illustrated in Figure 4.38(a), in which router R4 is sending a Join to the rendezvous point for some group.

The initial Join message is ―wildcarded‖; that is, it applies to all senders. A Join message clearly must pass through some sequence of routers before reaching the RP (e.g., R2). Each router along the path looks at the Join and creates a forwarding table entry for the shared tree, called a (*, G) entry (* meaning ―all senders‖).

As more routers send Joins toward the RP, they cause new branches to be addedto the tree, as illustrated in Figure 4.38(b). Note that in this case, the Join only needs to travel to R2, which can add the new branch to the tree simply by adding a new outgoing interface to the forwarding table entry created for this group. R2 need not forward the Join on to the RP. Note also that the end result of this process is to build a tree whose root is the RP.

At this point, suppose a host wishes to send a message to the group. To do so, it constructs a packet with the appropriate multicast group address as its destination and sends it to a router on its local network known as the *designated router* (DR).

Thus, in Figure 4.38(c), we see a source-specific route from R1 to the RP (indicated by the dashed line) and a tree that is valid for all senders from the RP to the receivers (indicated by the colored line).



Delivery of a packet along a shared tree. R1 tunnels the packet to the RP, which forwards it along the shared tree to R4 and R5.

CONGESTION AVOIDANCE IN NETWORK LAYER

• When one part of the subnet (e.g. one or more routers in an area) becomes overloaded, congestion results.

• Because routers are receiving packets faster than they can forward them, one of two things must happen:

– The subnet must prevent additional packets from entering the congested region until those already present can be processed.

– The congested routers can discard queued packets to make room for those that are arriving.

Factors that Cause Congestion

- Packet arrival rate exceeds the outgoing link capacity.
- Insufficient memory to store arriving packets
- Bursty traffic
- Slow processor

Traffic Shaping

• Another method of congestion control is to ―shape‖ the traffic before it enters the network.

• Traffic shaping controls the *rate* at which packets are sent (not just how many). Used in ATM and Integrated Services networks.

At connection set-up time, the sender and carrier negotiate a traffic pattern (shape).

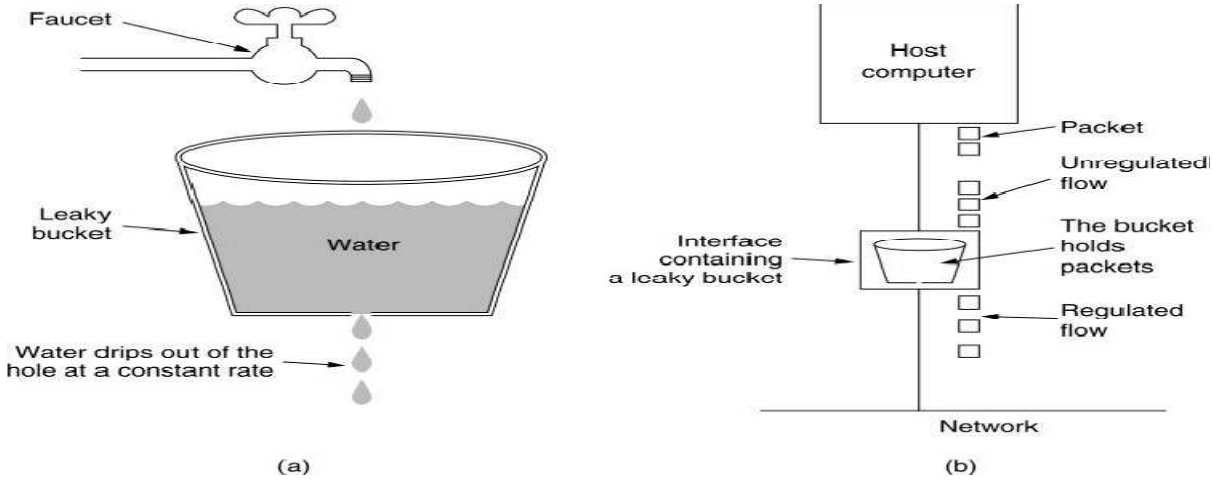
• Two traffic shaping algorithms are:

Leaky Bucket
Token Bucket

The Leaky Bucket Algorithm

• The Leaky Bucket Algorithm used to control rate in a network. It is implemented as a single-server queue with constant service time. If the bucket (buffer) overflows then packets are discarded**.**

The Leaky Bucket Algorithm



• The leaky bucket enforces a constant output rate (average rate) regardless of the burstiness of the input. Does nothing when input is idle.

• The host injects one packet per clock tick onto the network. This results in a uniform flow of packets, smoothing out bursts and reducing congestion.

• When packets are the same size (as in ATM cells), the one packet per tick is okay. For variable length packets though, it is better to allow a fixed number of bytes per tick. E.g. 1024 bytes per tick will allow one 1024-byte packet or two 512-byte packets or four 256-byte packets on 1 tick.

Token Bucket Algorithm

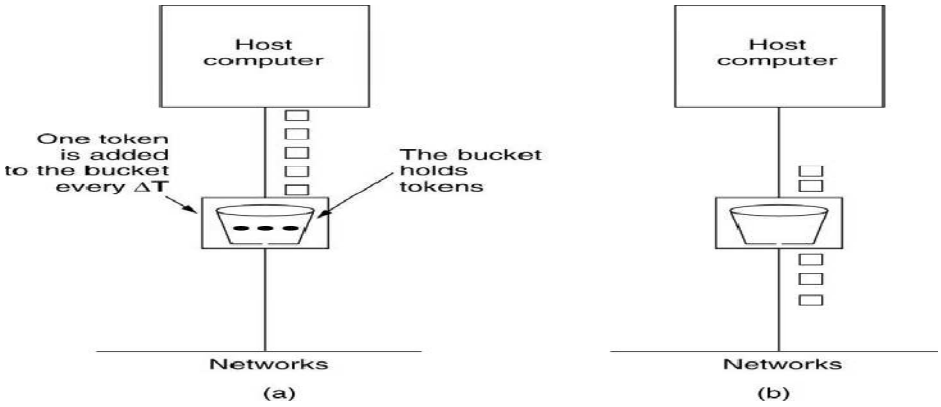
• In contrast to the LB, the Token Bucket Algorithm, allows the output rate to vary, depending on the size of the burst.

• In the TB algorithm, the bucket holds tokens. To transmit a packet, the host must capture and destroy one token.

• Tokens are generated by a clock at the rate of one token every t sec.

• Idle hosts can capture and save up tokens (up to the max. size of the bucket) in order to send larger bursts later.

The Token Bucket Algorithm



Leaky Bucket vs Token Bucket

• LB discards packets; TB does not. TB discards tokens.

• With TB, a packet can only be transmitted if there are enough tokens to cover its length in bytes.

• LB sends packets at an average rate. TB allows for large bursts to be sent faster by speeding up the output.

• TB allows saving up tokens (permissions) to send large bursts. LB does not allow saving.

**********ALL THE BEST**********