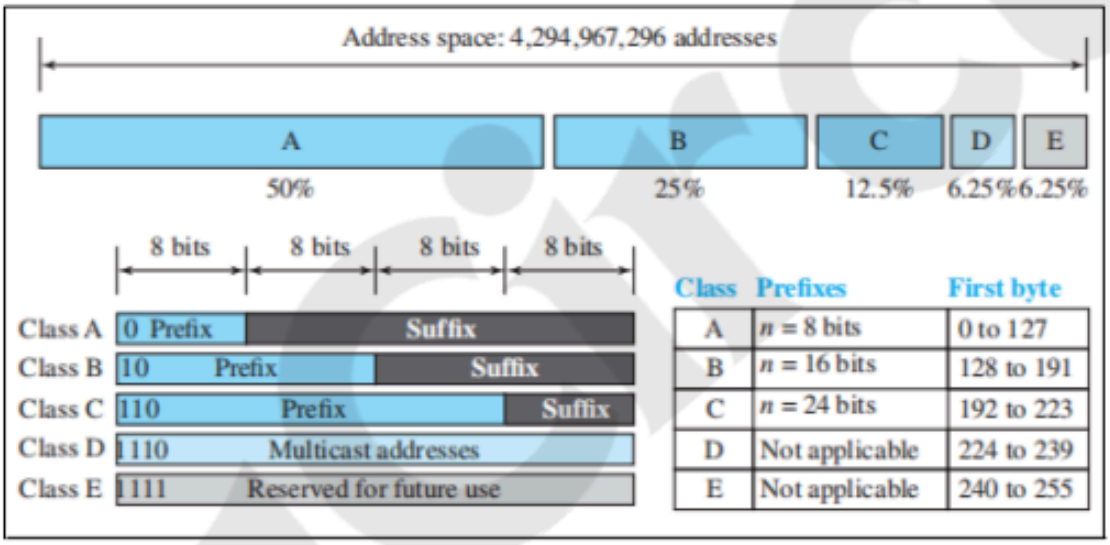
Classful Addressing

- When the Internet started, an IPv4 address was designed with a fixed-length prefix, but to accommodate both small and large networks, three fixed-length prefixes were designed instead of one (n = 8, n = 16, and n = 24). The whole address space was divided into five classes (class A, B, C, D, and E). This scheme is referred to as **classful addressing**.
- In class A, the network length is 8 bits, but since the first bit, which is 0, defines the class, we can have only seven bits as the network identifier. This means there are only $2^7 = 128$ networks in the world that can have a class A address.
- In class B, the network length is 16 bits, but since the first two bits, which are $(10)_2$, define the class, we can have only 14 bits as the network identifier. This means there are only 214 = 16,384 networks in the world that can have a class B address.
- All addresses that start with $(110)_2$ belong to class C. In class C, the network length is 24 bits, but since three bits define the class, we can have only 21 bits as the network identifier. This means there are $2^{21} = 2{,}097{,}152$ networks in the world that can have a class C address.



Class	Prefixes	First byte
A	*n* = 8 bits	0 to 127
B	*n* = 16 bits	128 to 191
C	*n* = 24 bits	192 to 223
D	Not applicable	224 to 239
E	Not applicable	240 to 255

Class D is not divided into prefix and suffix. It is used for multicast addresses. All addresses that start with 1111 in binary belong to class E. As in Class D, Class E is not divided into prefix and suffix and is used as reserve.

Address Depletion

Classful addressing became obsolete due to address depletion. The inefficient distribution of addresses led to a rapid shortage. For example, Class A, which could only be assigned to 128 organizations, provided more addresses (16.7 million per network) than most needed, resulting in wasted space. Class B, for midsize organizations, also had many unused addresses. Class C's flaw was offering too few addresses (256 per network), making it impractical for many companies. Class E was rarely used, leading to further waste.

Subnetting and Supernetting

Two strategies were proposed to address depletion: **subnetting** and **supernetting**. Subnetting divides large blocks (Class A or B) into smaller subnets with longer prefixes, but it failed because large organizations didn't want to share unused addresses. Supernetting combined multiple Class C blocks to create larger blocks for organizations needing more than 256 addresses, but this made packet routing more complex and also proved ineffective.

Advantage of Classful Addressing

Although classful addressing had several problems and became obsolete, it had one advantage: Given an address, we can easily find the class of the address and, since the prefix length for each class is fixed, we can find the prefix length immediately.

Classless Addressing

Subnetting and supernetting didn't fully solve address depletion. As the Internet grew, it became clear that a larger address space was needed, leading to the development of **IPv6** as a long-term solution. However, a short-term fix was **classless addressing**, which kept IPv4 but removed class boundaries to distribute addresses more fairly.

ISPs played a role in this change, providing Internet access by subdividing large address ranges and assigning smaller blocks (1, 2, 4, etc.) to customers. **Classless addressing**, introduced in 1996, allows for variable-length blocks of addresses (e.g., 1, 2, 4, or 128). The prefix of the address identifies the network, while the suffix identifies the device. The number of addresses in each block must be a power of 2, and the blocks are non-overlapping.

In classless addressing, the prefix length is variable, ranging from 0 to 32. A shorter prefix indicates a larger network, while a longer prefix means a smaller network. Classless addressing can be applied to classful addressing by treating Class A as having a prefix of 8, Class B with a prefix of 16, and so on.

Prefix Length and Slash Notation (CIDR)

In classless addressing, the prefix length (n) is added to the address, separated by a slash, known as slash notation or CIDR (Classless Inter-Domain Routing). For example, in 167.199.170.82/27, "27" is the prefix length. This notation is necessary since the address alone does not define the block.

To extract information from a classless address:

Number of addresses = 2^(32 - n)
First address: Keep the first n bits, set the rest to 0.
Last address: Keep the first n bits, set the rest to 1.
For 167.199.170.82/27, there are 32 addresses in the block.

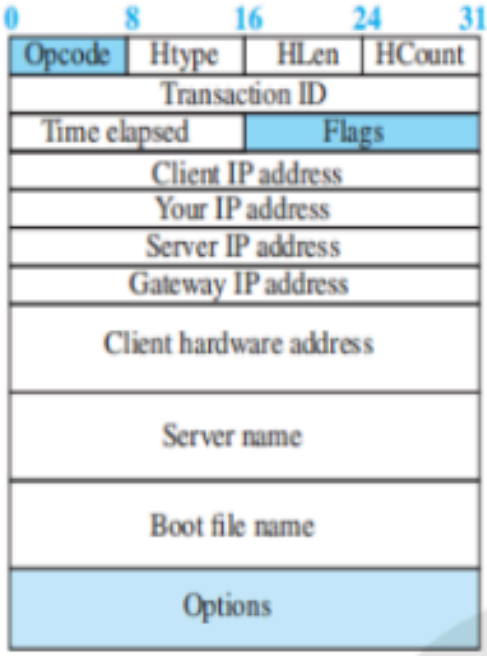
Dynamic Host Configuration Protocol (DHCP)

DHCP is an application-layer protocol that automates IP address assignment, using a client-server model. Widely used on the Internet, it is often referred to as a "plug-and-play" protocol.

- **Address Assignment:** DHCP can assign permanent or temporary IP addresses. For example, ISPs can use DHCP to provide temporary addresses to users, allowing limited IP resources to serve more customers.

- **Additional Information:** DHCP can also provide essential details like the network prefix, default router address, and name server address.

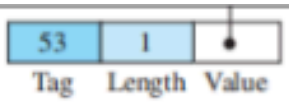
DHCP Message Format



Fields:
Opcode: Operation code, request (1) or reply (2)
Htype: Hardware type (Ethernet, ...)
HLen: Length of hardware address
HCount: Maximum number of hops the packet can travel
Transaction ID: An integer set by the client and repeated by the server
Time elapsed: The number of seconds since the client started to boot
Flags: First bit defines unicast (0) or multicast (1); other 15 bits not used
Client IP address: Set to 0 if the client does not know it
Your IP address: The client IP address sent by the server
Server IP address: A broadcast IP address if client does not know it
Gateway IP address: The address of default router
Server name: A 64-byte domain name of the server
Boot file name: A 128-byte file name holding extra information
Options: A 64-byte field with dual purpose described in text

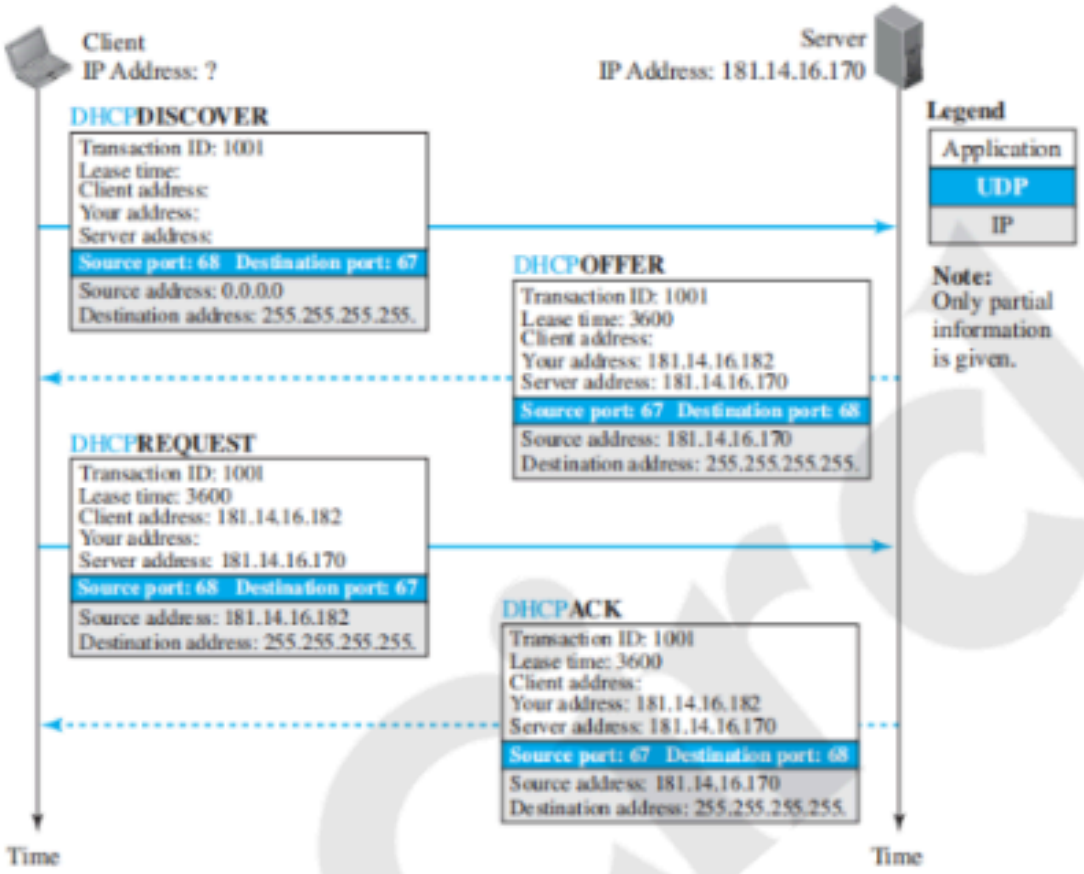
The 64-byte option field in DHCP serves two purposes: carrying additional or vendor-specific information. A special value, called a "magic cookie" (99.130.83.99), helps the client recognize options in the message. The next 60 bytes contain options, structured in three fields: a 1-byte tag, 1-byte length, and variable-length value. The tag field (e.g., 53) can indicate one of the 8 DHCP message types used by the protocol.

1 DHCPDISCOVER	5 DHCPACK
2 DHCPOFFER	6 DHCPNACK
3 DHCPREQUEST	7 DHCPRELEASE
4 DHCPDECLINE	8 DHCPINFORM


53
1
Tag
Length
Value

DHCP Operation

Figure shows a simple scenario.



1. The host creates a **DHCPDISCOVER** message with only a random transaction-ID, as it doesn't know its own IP address or the server's. The message is sent using UDP (source port 68, destination port 67) and broadcasted (source IP: 0.0.0.0, destination IP: 255.255.255.255).
2. The DHCP server responds with a **DHCPOFFER** message, containing the offered IP address, server address, and lease time. This message is sent with the same port numbers but reversed, using a broadcast address so other servers can also offer better options.
3. The host selects the best offer and sends a **DHCPREQUEST** to the server. The message includes the chosen IP address and is sent as a broadcast (source: new client IP, destination: 255.255.255.255) to notify other servers that their offers were declined.
4. The selected server responds with a **DHCPACK** if the IP address is valid, completing the process. If the IP address is unavailable, a **DHCPNACK** is sent, and the host must restart the process.

Two Well-Known Ports

DHCP uses well-known ports (68 for the client, 67 for the server) to avoid conflicts with other services using ephemeral ports. Since DHCP responses are broadcast, using a well-

known port (68) ensures the DHCP response is only delivered to the correct client, preventing confusion with other clients (e.g., DAYTIME) that might be using the same temporary port. If two DHCP clients are running simultaneously (e.g., after a power failure), they are distinguished by their unique transaction IDs.

Using FTP

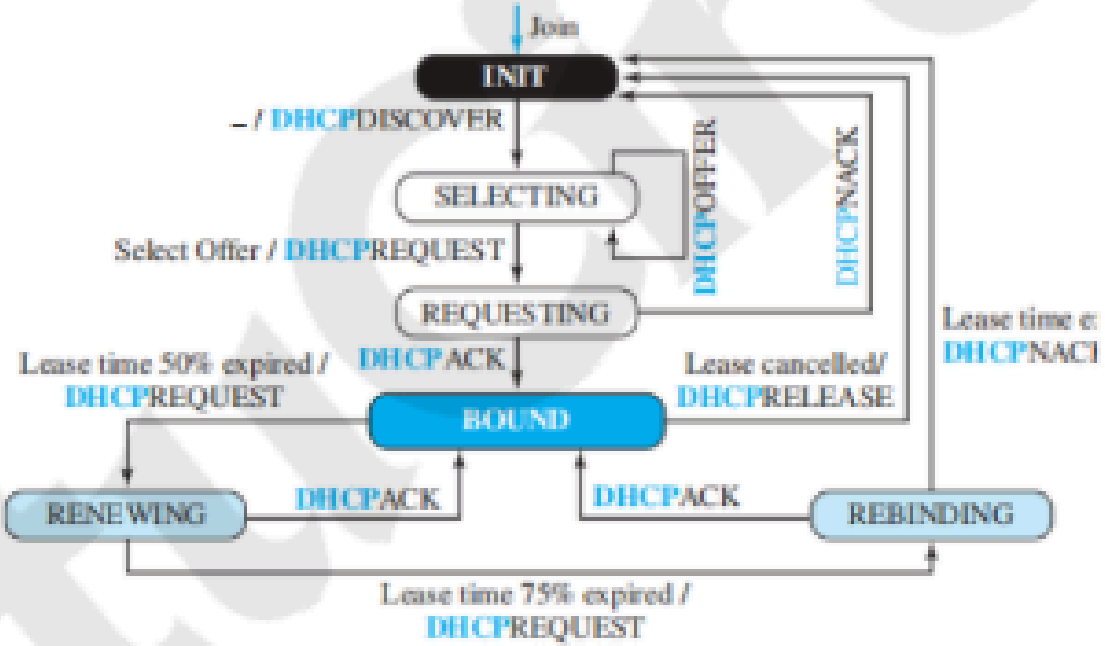
The DHCPACK message includes a pathname to a file with additional information (e.g., DNS server address). The client uses FTP to retrieve this information.

Error Control

Since DHCP relies on unreliable UDP, it ensures error control by requiring UDP checksums and using timers with a retransmission policy. To avoid traffic congestion (e.g., after a power failure), clients use random timers for retransmission.

Transition States

The operation of the DHCP were very simple. To provide dynamic address allocation, the DHCP client acts as a state machine that performs transitions from one state to another depending on the messages it receives or sends.



1. **INIT state**: The client starts here and sends a *Discover* message to find a DHCP server.
2. **SELECTING state**: After receiving one or more *Offer* messages, the client selects one offer.
3. **REQUESTING state**: The client sends a *Request* message to the selected server and waits.

4. **BOUND state**: If the server responds with an *ACK* message, the client uses the assigned IP address.
5. **RENEWING state**: When 50% of the lease time is expired, the client tries to renew the lease by contacting the server. If successful, it stays in the BOUND state.
6. **REBINDING state**: If the lease is 75% expired and no response is received, the client tries to contact any DHCP server. If the server responds, it stays BOUND; otherwise, it goes back to INIT to request a new IP.

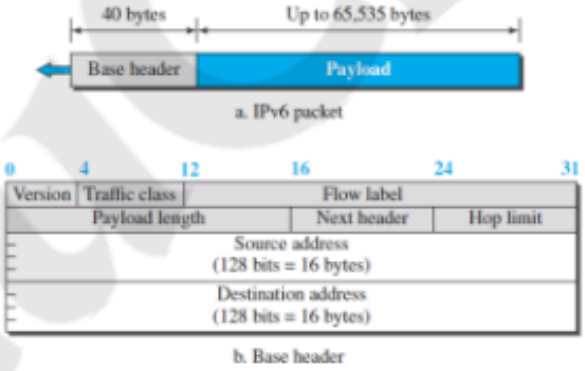
Timers:

- **Renewal timer**: Triggered at 50% lease expiration.
- **Rebinding timer**: Triggered at 75% lease expiration.
- **Expiration timer**: Triggered when the full lease expires.

confidentiality and integrity of the packet.

Packet Format

The IPv6 packet is shown in Figure. Each packet is composed of a base header followed by the payload. The base header occupies 40 bytes, whereas payload can be up to 65,535 bytes of information. The description of fields follows.



Version. The 4-bit version field defines the version number of the IP. For IPv6, the value is 6.

Traffic class. The 8-bit traffic class field is used to distinguish different payloads with different delivery requirements. It replaces the *type-of-service* field in IPv4.

Flow label. The flow label is a 20-bit field that is designed to provide special handling for a particular flow of data.

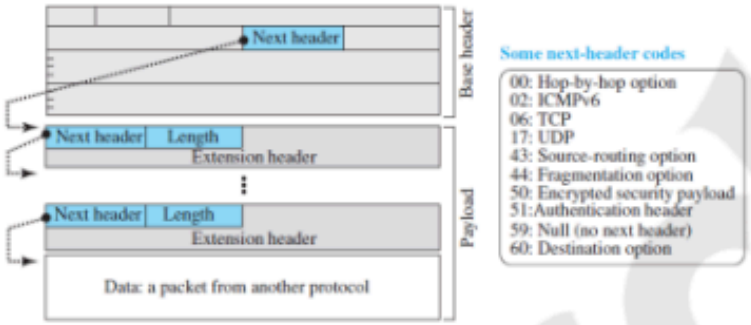
Payload length. The 2-byte payload length field defines the length of the IP datagram excluding the header.

Next header. The **next header** is an 8-bit field defining the type of the first extension header (if present) or the type of the data that follows the base header in the datagram..

Hop limit. The 8-bit hop limit field serves the same purpose as the TTL field in IPv4.

Source and destination addresses. The source address field is a 16-byte (128-bit) Internet address that identifies the original source of the datagram. The destination address field is a 16-byte (128-bit) Internet address that identifies the destination of the datagram.

Payload. Compared to IPv4, the payload field in IPv6 has a different format and meaning, as shown in Figure.



- ➔ The payload in IPv6 means a combination of zero or more extension headers (options) followed by the data from other protocols (UDP, TCP, and so on).
- ➔ In IPv6, options, which are part of the header in IPv4, are designed as extension headers. The payload can have as many extension headers as required by the situation.
- ➔ Each extension header has two mandatory fields, next header and the length, followed by information related to the particular option.
- ➔ Note that each next header field value (code) defines the type of the next header (hop-by-hop option, sourcerouting option, . . .); the last next header field defines the protocol (UDP, TCP, . . .) that is carried by the datagram.

Concept of Flow and Priority in IPv6

- ➔ In version 6, the flow label has been added to the format of the IPv6 datagram to allow us to use IPv6 as a connection-oriented protocol.
- ➔ To a router, a flow is a sequence of packets that share the same characteristics, such as traveling the same path, using the same resources, having the same kind of security, and so on.
- ➔ A router that supports the handling of flow labels has a flow label table. The table has an entry for each active flow label; each entry defines the services required by the corresponding flow label.
- ➔ In its simplest form, a flow label can be used to speed up the processing of a packet by a router.

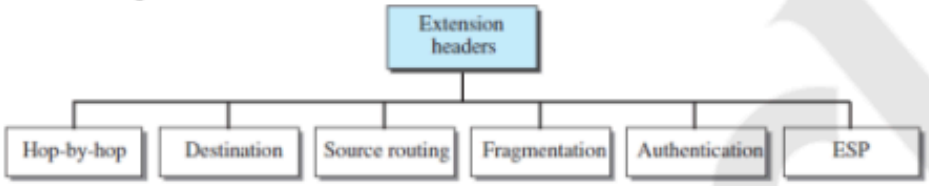
Fragmentation and Reassembly

- ➔ IPv6 datagrams can be fragmented only by the source, not by the routers; the reassembly takes place at the destination.

- ➔ The fragmentation of a packet in a router needs a lot of processing hence processing at router is not allowed.
- ➔ In IPv6, the source can check the size of the packet and make the decision to fragment the packet or not.
- ➔ When a router receives the packet, it can check the size of the packet and drop it if the size is larger than allowed by the MTU of the network ahead.
- ➔ The router then sends a packet-too-big ICMPv6 error message to inform the source.

Extension Header

An IPv6 packet is made of a base header and some extension headers. The length of the base header is fixed at 40 bytes. The base header can be followed by up to six **extension headers**, to give more functionality**.**



Hop-by-Hop Option

The *hop-by-hop option* is used when the source needs to pass information to all routers visited by the datagram. So far, only three hop by- hop options have been defined: Pad1, PadN, and jumbo payload.

Pad1. This option is 1 byte long and is designed for alignment purposes. Some options need to start at a specific bit of the 32-bit word. If an option falls short of this requirement by exactly one byte, Pad1 is added.

PadN. PadN is similar in concept to Pad1. The difference is that PadN is used when 2 or more bytes are needed for alignment.

Jumbo payload. If for any reason a longer payload(>65,535 bytes) is required, the jumbo payload option is used to define this longer length.

Destination Option

The **destination option** is used when the source needs to pass information to the destination only. Intermediate routers are not permitted access to this information.

Source Routing

The source routing extension header combines the concepts of the strict source route and the loose source route options of IPv4.

Fragmentation

In IPv6, only the original source can fragment. A source must use a **Path MTU Discovery technique** to find the smallest MTU supported by any network on the path. The source then fragments using this knowledge.

Authentication

The **authentication** extension header has a dual purpose: it validates the message sender and ensures the integrity of data.

Encrypted Security Payload

The **encrypted security payload (ESP)** is an extension that provides confidentiality and guards against eavesdropping.

The Distance-Vector (DV) Routing Algorithm

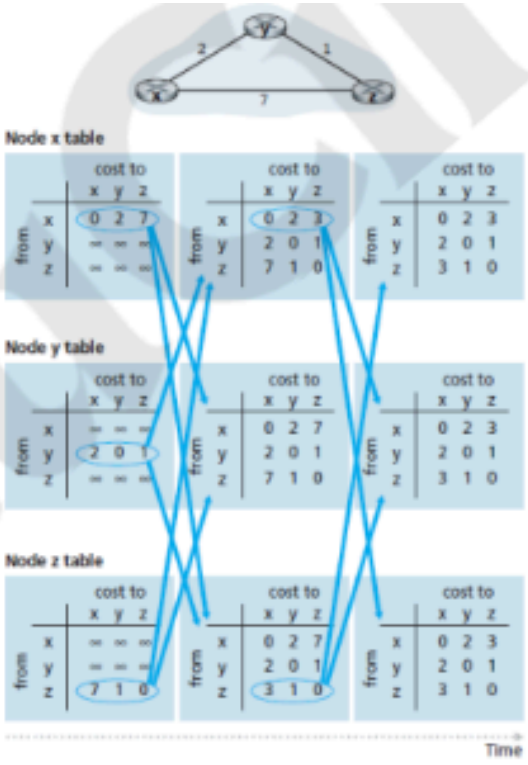
- ➔ The distance vector (DV) algorithm is iterative, asynchronous, and distributed.
- ➔ It is iterative that the process continues on until no more information is exchanged between neighbours.
- ➔ The algorithm is asynchronous in that it does not require all of the nodes to operate in lockstep with each other.
- ➔ Let dx(y) be the cost of the least-cost path from node x to node y. Then the least costs are related by the celebrated Bellman-Ford equation, namely,

$$\mathbf{dx(y) = minv\{c(x,v) + dv(y)\}}$$

- ➔ With the DV algorithm, each node x maintains the following routing information:
  - For each neighbour v, the cost c(x,v) from x to directly attached neighbour, v
  - Node x's distance vector, that is, Dx = [Dx(y): y in N], containing x's estimate of its cost to all destinations, y, in N
  - The distance vectors of each of its neighbours, that is, Dv = [Dv(y): y in N] for each neighbour v of x


```
1  Initialization:
2     for all destinations y in N:
3        Dx(y) = c(x,y)    /* if y is not a neighbor then c(x,y) = ∞ */
4     for each neighbor w
5        Dw(y) = ? for all destinations y in N
6     for each neighbor w
7        send distance vector Dx = [Dx(y): y in N] to w
8
9  loop
10    wait (until I see a link cost change to some neighbor w or
11          until I receive a distance vector from some neighbor w)
12
13    for each y in N:
14       Dx(y) = minv(c(x,v) + Dv(y))
15
16    if Dx(y) changed for any destination y
17       send distance vector Dx = [Dx(y): y in N] to all neighbors
18
19 forever
```

Figure illustrates the operation of the DV algorithm for the simple three node network shown at the top of the figure.



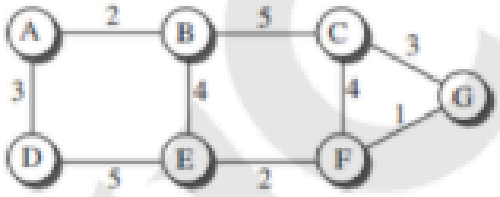
Distance-vector (DV) algorithm

Count to Infinity

A problem with distance-vector routing is that any decrease in cost (good news) propagates quickly, but any increase in cost (bad news) will propagate slowly. For a routing protocol to work properly, if a link is broken (cost becomes infinity), every other router should be aware of it immediately, but in distance-vector routing, this takes some time. The problem is referred to as count to infinity. It sometimes takes several updates before the cost for a broken link is recorded as infinity by all routers.

Link-State Routing

- ➔ Link-State (LS) routing is a method used by routers to determine the best, or least-cost, path through a network.
- ➔ In this method, each connection between routers is referred to as a **link**, and the **link-state** describes the cost of using that connection. Lower costs are preferred, while a link with an infinite cost means it is unavailable or broken.
- ➔ To create efficient routing paths, each router must have a complete map of the network, which includes the state (cost) of every link. This information is stored in the **Link-State Database (LSDB)**, which is shared across all routers.
- ➔ The LSDB is often represented as a matrix, where each value shows the cost of a link between two routers.
- ➔ With this database, each router can build the least-cost tree, ensuring it knows the shortest routes to every other router.



a. The weighted graph

	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

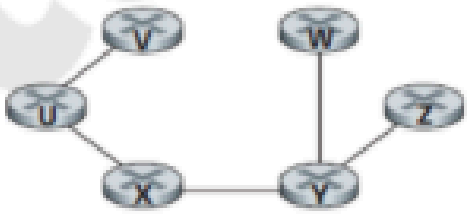
b. Link state database

- To create the **Link-State Database (LSDB)**, routers use a process called **flooding**. Each router sends a greeting to its direct neighbors to gather two pieces of information: the neighbor's identity and the cost of the link.
- This data is packaged into a **Link-State Packet (LSP)** and sent to all neighboring routers. When a router receives an LSP, it checks if the information is new by comparing the sequence number.
- If the LSP is new, the router keeps it and sends copies to other neighbors, except the one from which it was received.
- This process repeats until all routers have the latest LSPs, allowing each router to create an identical **LSDB**, which provides a complete map of the network. This LSDB helps routers calculate the least-cost paths to all other nodes.

- The link-state routing algorithm is known as **Dijkstra's algorithm**. Dijkstra's algorithm computes the least-cost path from one node to all other nodes in the network.
- The following notations are used:
  - D(v): cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.
  - p(v): previous node (neighbor of v) along the current least-cost path from the source to v.
  - N' : subset of nodes; v is in N ' if the least-cost path from the source to v is definitively known.

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v is a neighbor of u
5        then D(v) = c(u,v)
6      else D(v) = ∞
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for each neighbor v of w and not in N':
12         D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14    least path cost to w plus cost from w to v */
15 until N'= N
```

step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Destination	Link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

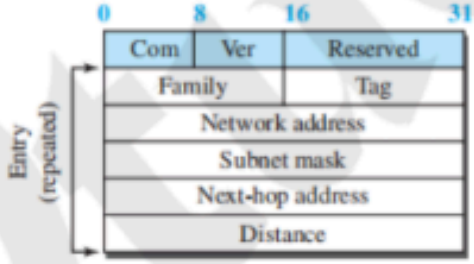
Least cost path and forwarding table for nodule u

Routing Information Protocol (RIP)

The Routing Information Protocol (RIP) is one of the most widely used intradomain routing protocols based on the distance-vector routing algorithm we described earlier. RIP was started as part of the Xerox Network System (XNS), but it was the Berkeley Software Distribution (BSD) version of UNIX that helped make the use of RIP widespread.

RIP Messages

Two RIP processes, a client and a server, like any other processes, need to exchange messages. RIP defines the format of the message. Part of the message, which we call *entry*, can be repeated as needed in a message. Each entry carries the information related to one line in the forwarding table of the router that sends the message.



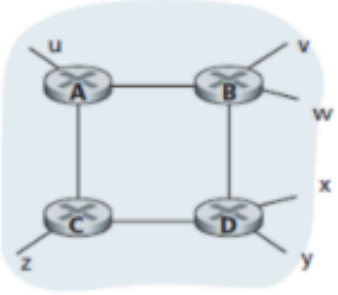
Fields

Com: Command, request (1), response (2)
Ver: Version, current version is 2
Family: Family of protocol, for TCP/IP value is 2
Tag: Information about autonomous system
Network address: Destination address
Subnet mask: Prefix length
Next-hop address: Address length
Distance: Number of hops to the destination

RIP has two types of messages: request and response. A request message is sent by a router that has just come up or by a router that has some time-out entries. A request message can ask about specific entries or all entries. A response (or update) message can be either solicited or unsolicited. A solicited response message is sent only in answer to a request message. It contains information about the destination specified in the corresponding request message. An unsolicited response message, on the other hand, is sent periodically, every 30 seconds or when there is a change in the forwarding table.

RIP Algorithm

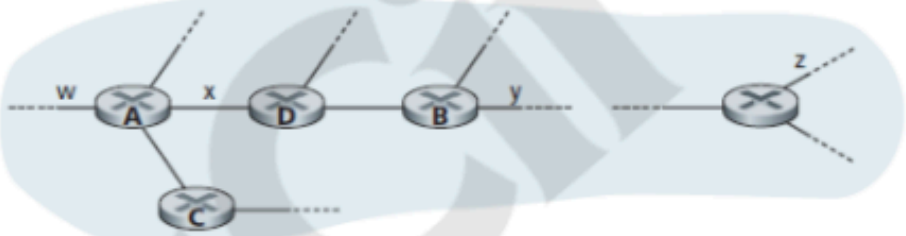
- ➔ RIP is a distance-vector protocol. Figure illustrates an AS with six leaf subnets. The table in the figure indicates the number of hops from the source A to each of the leaf subnets.
- ➔ In RIP, routing updates are exchanged between neighbours approximately every 30 seconds using a RIP response message.



Destination	Hops
u	1
v	2
w	2
x	3
y	3
z	2

Number of hops from source router A to various subnets

➔ Response messages are also known as RIP advertisements. Consider the portion of an AS shown in Figure.



A portion of an autonomous system

➔ Figure shows the routing table for router D. suppose that 30 seconds later, router D receives from router A the advertisement shown in next Figure.

Destination Subnet	Next Router	Number of Hops to Destination
w	A	2
y	B	2
z	B	7
x	--	1
....	....	....

Routing table in router *D* before receiving advertisement from router *A*

Destination Subnet	Next Router	Number of Hops to Destination
z	C	4
w	--	1
x	--	1
....	....	....

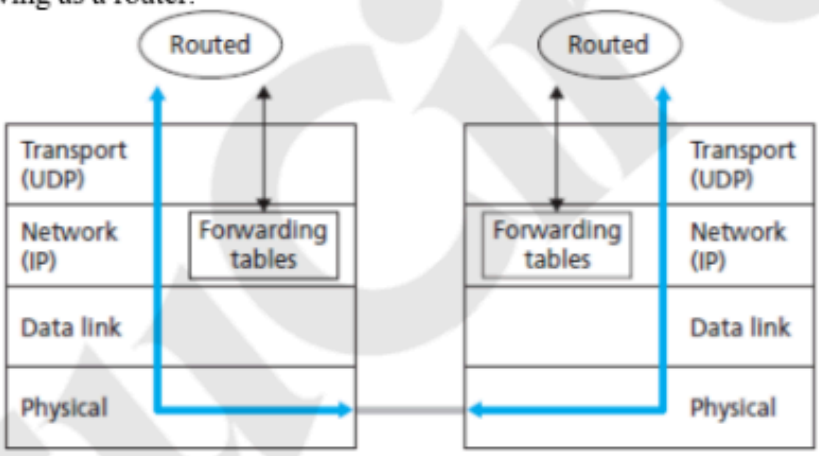
Advertisement from router A

- Subnet z is only four hops away from router A. Router D, upon receiving the advertisement, merges the advertisement with the old routing table.
- Router D learns that there is a path through router A to subnet z that is shorter than the path through router B.
- Thus, router D updates its routing table to account for the shorter shortest path, as shown in Figure.

Destination Subnet	Next Router	Number of Hops to Destination
w	A	2
y	B	2
z	A	5
....	....	....

Routing table in router *D* after receiving advertisement from router *A*

Figure below shows how RIP is implemented in a UNIX system, for example, a UNIX workstation serving as a router.

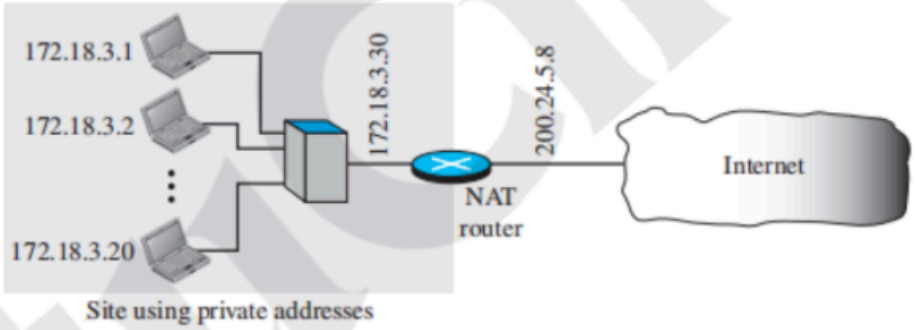


Implementation of RIP as the *routed* daemon

- A process called **routed** executes RIP, that is, maintains routing information and exchanges messages with routed processes running in neighbouring routers.
- Because RIP is implemented as an application-layer process, it can send and receive messages over a standard socket and use a standard transport protocol.
- As shown, RIP is implemented as an application-layer protocol running over UDP.

Network Address Resolution (NAT)

ISPs allocate limited IP address ranges to small businesses or households, but expanding these ranges can be difficult due to neighboring allocations. Often, not all devices in a network need simultaneous Internet access. For example, a small business with 20 computers may only need 4 to access the Internet at once, while the rest handle internal tasks. Using private IP addresses for internal communication and a few global IP addresses from the ISP for external communication solves this. Network Address Translation (NAT) allows a network to use private addresses internally and maps them to global addresses for Internet access via a NAT-enabled router.



The private network uses private addresses. The router that connects the network to the global address uses one private address and one global address. The private network is invisible to the rest of the Internet; the rest of the Internet sees only the NAT router with the address 200.24.5.8.

- ➔ In Network Address Translation (NAT), all outgoing packets pass through a NAT router, which replaces the source address with the global NAT address.
- ➔ Incoming packets also pass through the router, where the destination (global) address is replaced with the corresponding private address. To manage this, the NAT router uses a translation table that links private addresses to external addresses.
- ➔ When an outgoing packet is sent, the router records the destination address.

➔ When a response arrives, the router uses the source (external) address to find the corresponding private address, allowing proper routing within the local network.