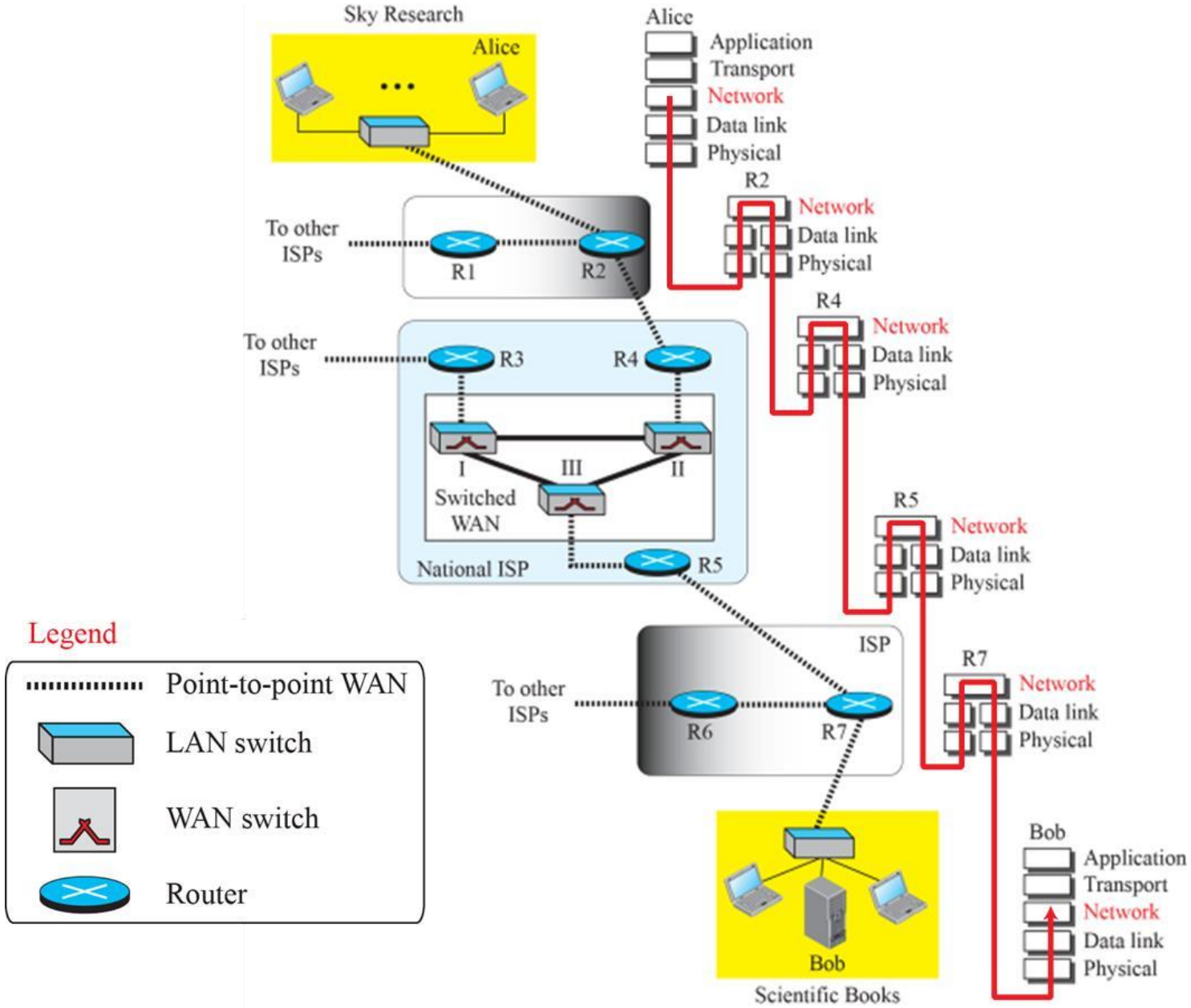
Chapter 4
Network layer

Objective

- We first discuss services that can be provided at the network layer: packetizing, routing, and forwarding.
- We then discuss the network layer at the TCP/IP suite: IPv4 and ICMPv4. We also discuss IPv4 addressing and related issues.
- We then concentrate on the unicast routing and unicast routing protocols.
- We then move to multicasting and multicast routing protocols protocol.


Sky Research
Alice
Alice
Application
Transport
Network
Data link
Physical
R2
Network
Data link
Physical
To other ISPs
R1
R2
R4
Network
Data link
Physical
To other ISPs
R3
R4
I
III
II
Switched WAN
National ISP
R5
R5
Network
Data link
Physical
ISP
R7
Network
Data link
Physical
To other ISPs
R6
R7
Bob
Application
Transport
Network
Data link
Physical
Bob
Scientific Books
Legend
Point-to-point WAN
LAN switch
WAN switch
Router

Network-Layer Services

Before discussing the network layer in the Internet today, let's briefly discuss the network-layer services that, in general, are expected from a network-layer protocol.

- Packetizing
- Routing
- Forwarding

Packetizing

The network layer has an important job. It takes data from one place and sends it to another without altering or using the data. It's like a postal service that delivers packages without changing their contents.

Here's how it works:

- The source computer gets data from a higher-level program and puts it in a network packet. **This packet includes sender and receiver information** and other details needed for the network.
- The source computer then sends the packet to the data-link layer for delivery. **It can't change the data unless it's too big**, and it needs to split it into smaller parts.
- The destination computer gets the network packet from its data-link layer. It takes out the data and gives it to the right program. **If the packet was split into parts, the network layer waits until all parts arrive, puts them together**, and then gives the data to the program.
- Routers in between don't open the packets unless they need to **split them into smaller parts**. They also don't change the sender and receiver info. They only look at the info to know where to send the packet next. If a packet is split, they copy the header to all parts and make some changes as needed.

Routing

The network layer has another important job called "routing." This means it figures out **the best path for data to travel** from where it starts to where it needs to go. Think of it like finding the fastest way to get somewhere on a road trip.

In a computer network, there are different paths data can take, like different roads on a map. The network layer's job is to pick the best road for the data to travel on.

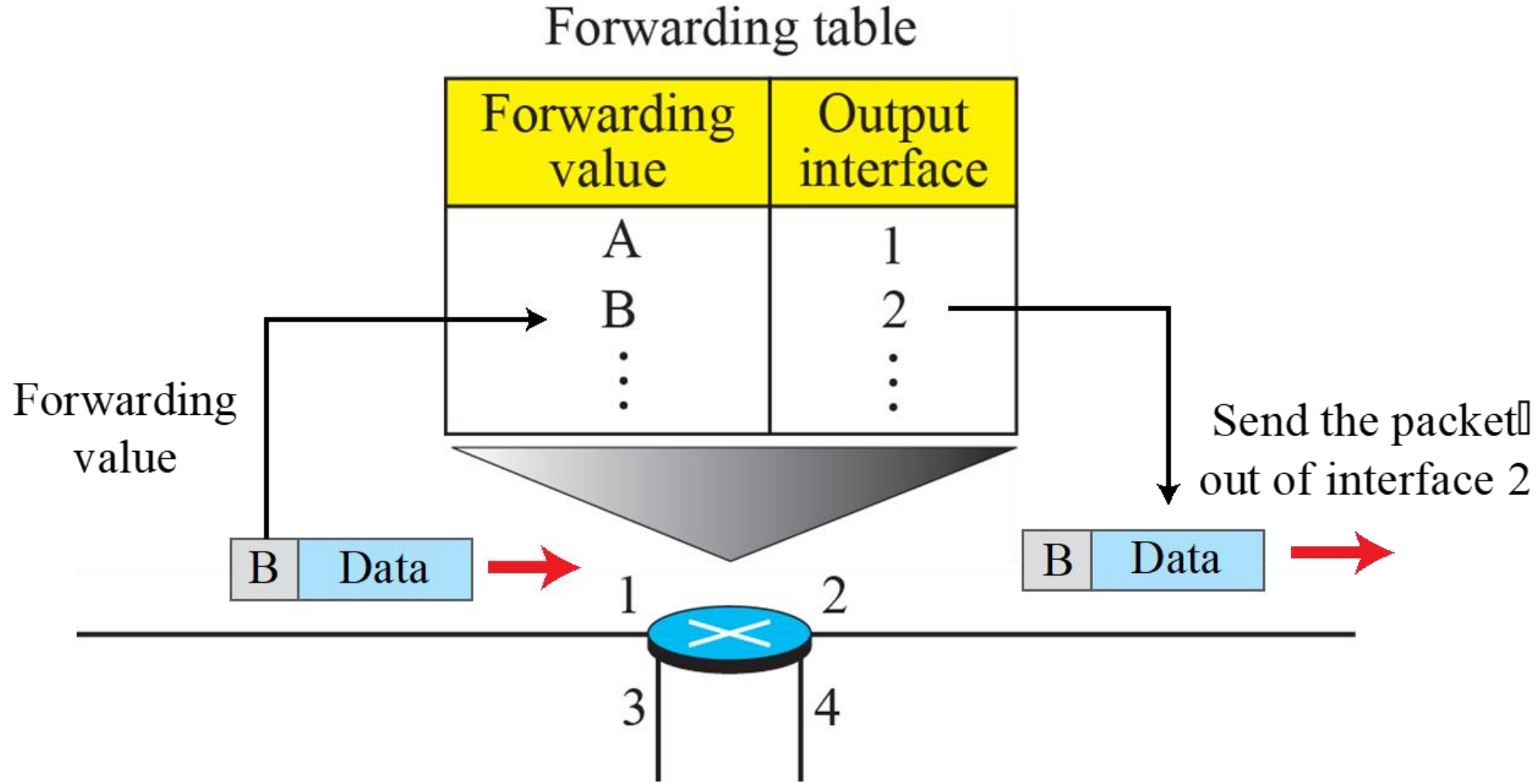
To do this, the network layer has special plans, like GPS for data. **It uses something called "routing protocols" to help routers (like traffic cops for data) know which path to choose**. These plans are set up before any data starts moving.

Forwarding

Think of routing like making a plan and using special rules to decide how data should travel through a network. Now, **forwarding is what each router does when data actually arrives at its doorstep**.

When data arrives at a router, it needs to know **where to send it next**. Imagine it's like a traffic cop directing cars at an intersection. **The router looks at the data's address (or a special tag) and checks a list (sometimes called a forwarding table, or routing table) to figure out which way to send it**.

So, routing is the plan-making part, and forwarding is the action part where the router decides where data goes based on that plan. It's like the router is following a map to guide the data to its destination.


Forwarding table
Forwarding value
Output interface
A
B
1
2
Forwarding value
Send the packet out of interface 2
B
Data
B
Data
1
2
3
4

Error control

In Chapter 3, we talked about how we handle mistakes in data transfer at the transport layer. In Chapter 5, we'll dive into how we deal with errors at the data-link layer. Now, even though we can also handle errors at the network layer, **the folks who designed the network layer for the Internet didn't focus on it much**.

One big reason is that data in the network layer can get broken up into pieces at different points along its journey, like **splitting a puzzle into parts. Checking for errors in this situation is not very efficient.**

However, these designers **did add a special "checksum" field** to the header of the data packet. It's like a security seal that makes sure the header isn't tampered with during its journey from one place to another.

It's important to note that even though the Internet's network layer doesn't directly fix errors, it has a helper **called ICMP. This protocol helps out if a data packet is thrown away or has weird stuff in its header.** We'll get into more detail about ICMP later in this chapter.

Flow control

Flow control is like managing how much data a sender computer can send to a receiver computer without causing a traffic jam. If the sender's computer is making data faster than the receiver's computer can handle, it's like pouring too much water into a glass—it overflows. To avoid this, the receiver needs to let the sender know when it's getting overwhelmed with data.

Interestingly, the network layer in the Internet doesn't directly handle this flow control job. Instead, it lets the sender send data whenever it's ready, without worrying about whether the receiver can keep up.

There are a few reasons for this approach.

First, **the receiver's job at the network layer is pretty simple** because it doesn't have to deal with error checking, so it's less likely to get overwhelmed.

Second, the higher-level programs using the network layer can **manage the incoming data by storing it temporarily and don't need to consume it all at once.**

Third, most of these higher-level programs **have their own flow control**, so adding more control at the network layer would make things more complex and less efficient.

Congestion control

Another thing to consider in network-layer protocols is congestion control. Congestion happens when there are too many data packets in one part of the Internet. It's like a traffic jam on the information highway. This can occur if computers send more data than the network or routers can handle. When this occurs, some packets might get dropped by routers to ease the traffic.

But here's the tricky part: **if too many packets get dropped, it can make things worse. That's because higher-level error checks might make the sender resend the lost packets, causing even more congestion. In extreme cases, everything can grind to a halt, and no data can get through.**

We'll go into more detail about how to deal with congestion in the network layer later in this chapter, even though the Internet doesn't directly handle it.

Quality of service

With the Internet, we can now do cool things like video chats and streaming music. But for these things to work smoothly, we need to make sure the quality of our communication is really good. The Internet has gotten better at this by improving the quality of service (QoS), which is like making sure your online experience is top-notch.

However, **most of these quality improvements happen in the higher layers of the Internet, not the network layer.** The network layer remains mostly the same.

Security

Another thing to think about in network communication is safety. When the Internet first started, it wasn't a big deal because only a few people at universities used it for research, and outsiders couldn't access it. So, they didn't worry much about security in the network layer.

But now, security is a big deal**. To make the network layer more secure, they've created something called IPSec**, which adds a layer of protection to the network.

Packet switching

how data moves around in a network. Think of it like a switch that connects things, like how an electrical switch connects power. In networking, there are two ways to do this: **circuit switching and packet switching.** We **use packet switching in the network layer because data is sent in packets here.** Circuit switching is used at a lower level.

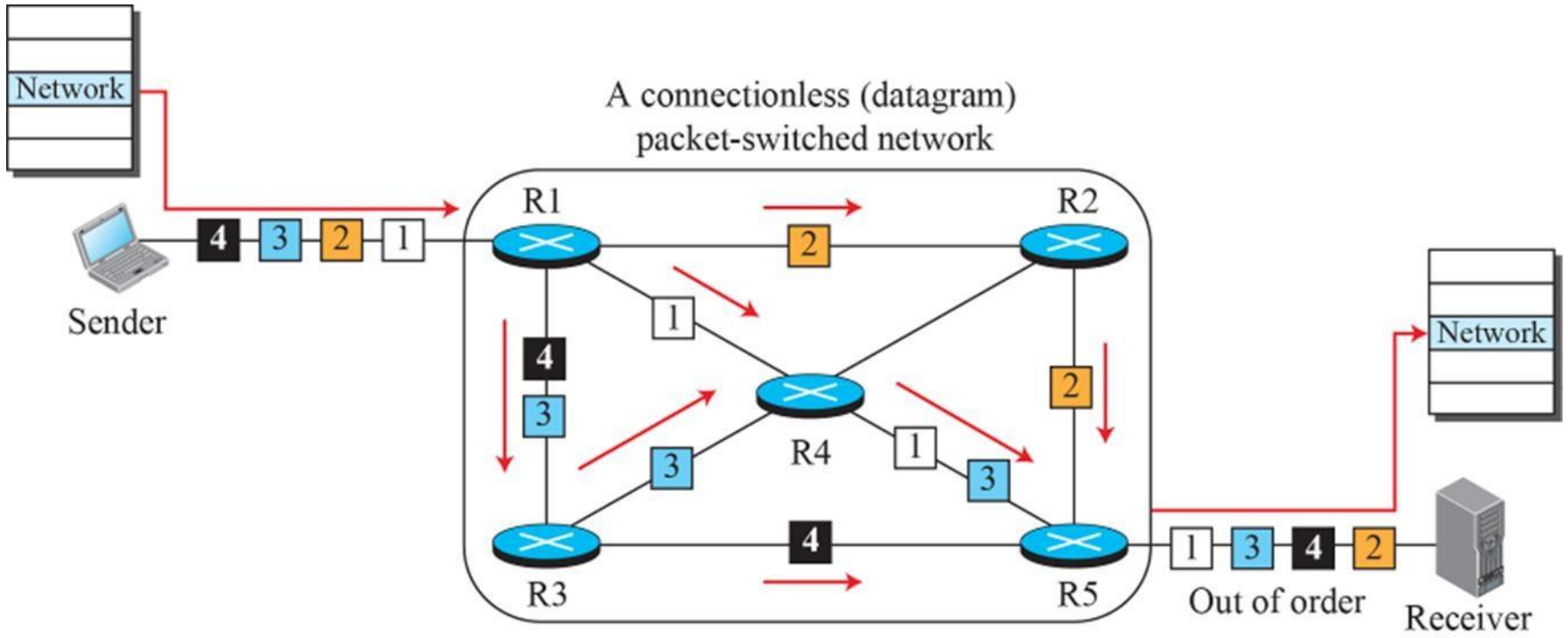
In the network layer, a message is split into packets and sent through the network. The sender sends them one by one, and the receiver gets them one by one. **The receiver waits for all the packets from the same message before passing it to the next level.** In a packet-switched network, devices still need to figure out how to send the packets to the right place. There are two ways to do this:

- the datagram way
- the virtual circuit way

Datagram Approach: Connectionless Service

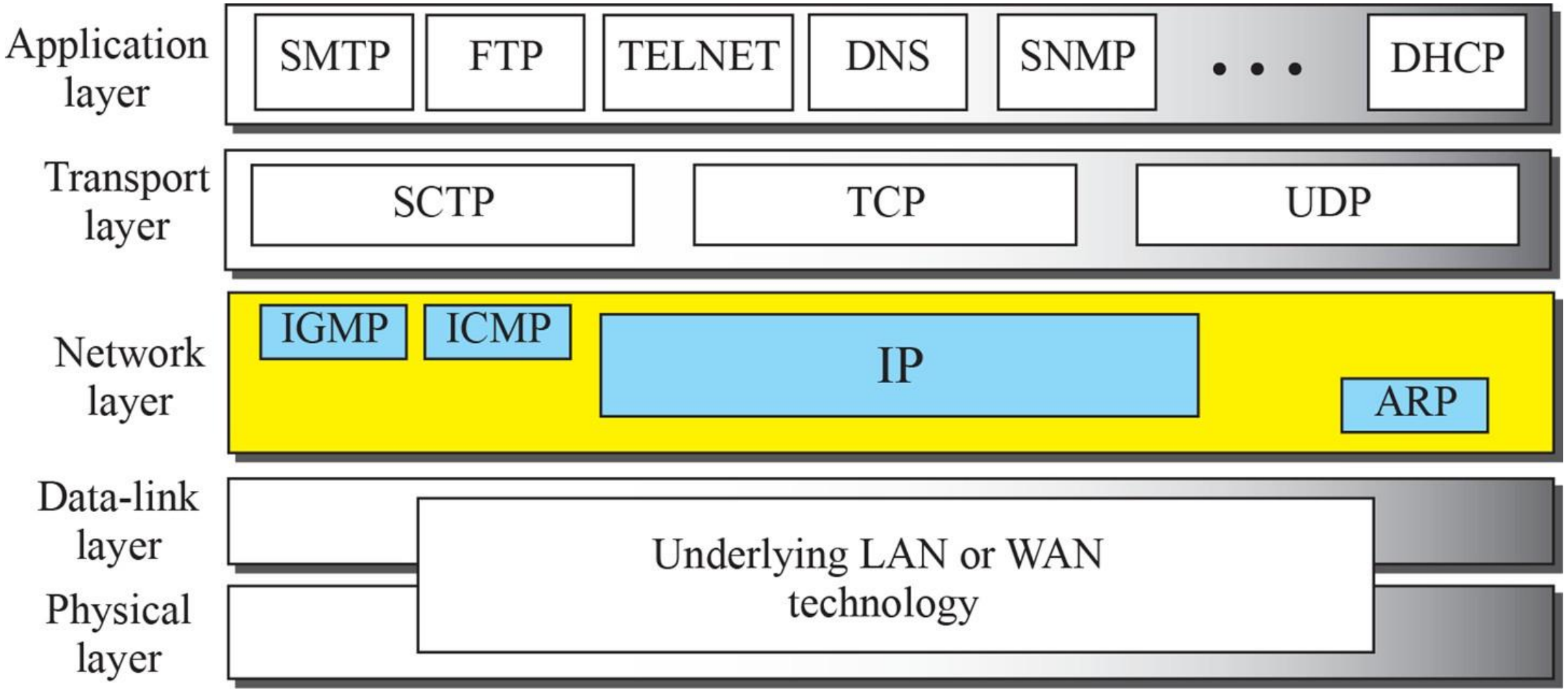
- When the Internet first started, the network layer was designed to keep things simple. It offered a service where each packet of data was treated separately, with no connection or relationship between them. **The network layer's job was just to get packets from one place to another**, and they could take different paths.
- Imagine each packet as its own independent traveler in the Internet journey. **They don't know each other, and they don't have to follow the same route**. Routers, which are like traffic cops for Internet data, help these packets find their way. They decide where to send each packet based on the destination address written on it.

- **The source address is there to help send an error message back to the sender if something goes wrong.** So, it's like each packet has a destination it's trying to reach, and routers help them get there. Think of it like sending letters to different places



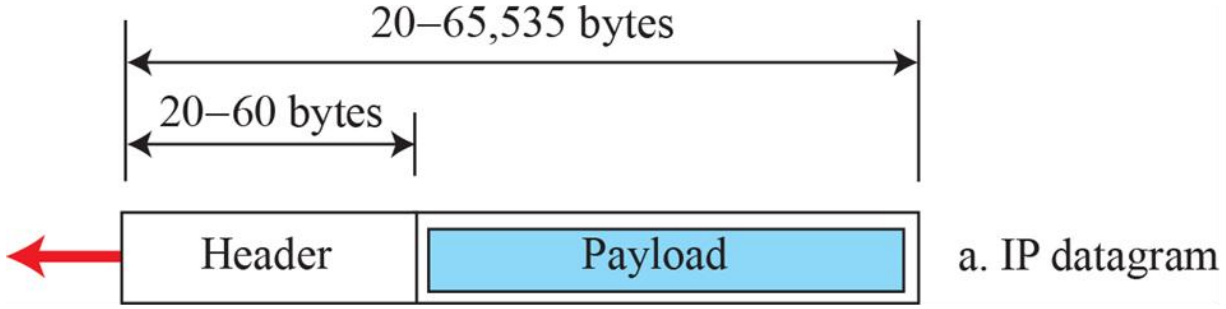
NETWORK-LAYER PROTOCOLS

- In this section, we show how the network layer is implemented in the TCP/IP protocol suite. The protocols in the network layer have gone through several versions; in this section, we concentrate on the current version (4), in the last section of this chapter, we briefly discuss version 6, which is on the horizon.


Application layer
SMTP
FTP
TELNET
DNS
SNMP
. . .
DHCP
Transport layer
SCTP
TCP
UDP
Network layer
IGMP
ICMP
IP
ARP
Data-link layer
Underlying LAN or WAN technology
Physical layer

IPv4 Datagram Format

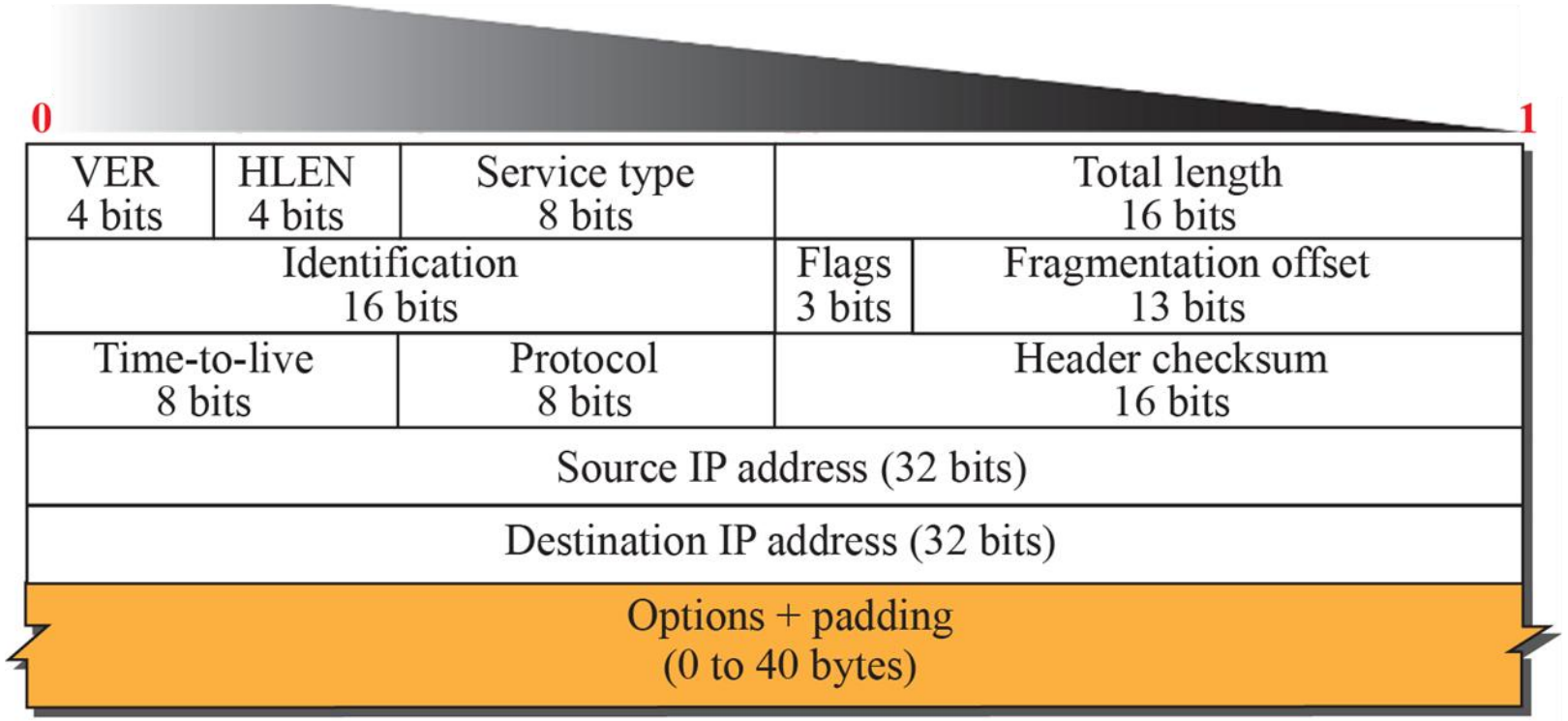
- the IPv4 datagram format. A datagram is a variable-length packet consisting of two parts: **header and payload (data**). The header is 20 to 60 bytes in length and contains information essential to routing and delivery. It is customary in TCP/IP to show the header in 4-byte sections



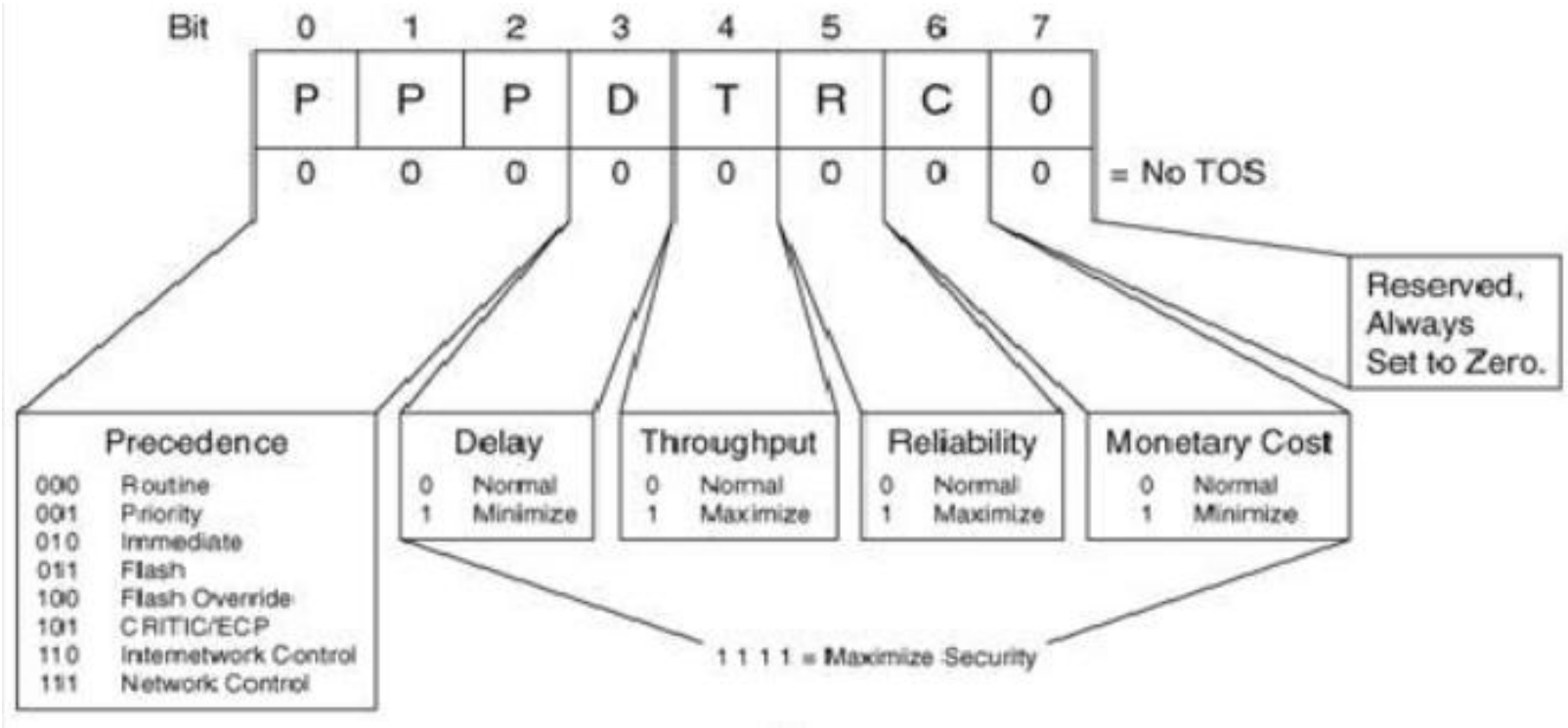
a. IP datagram

Legend

VER: version number
HLEN: header length
byte: 8 bits



b. Header format


Bit 0 1 2 3 4 5 6 7
P P P D T R C 0
0 0 0 0 0 0 0 0
= No TOS
Reserved, Always Set to Zero.
Precedence
000 Routine
001 Priority
010 Immediate
011 Flash
100 Flash Override
101 CRITIC/ECP
110 Internetwork Control
111 Network Control
Delay
0 Normal
1 Minimize
Throughput
0 Normal
1 Maximize
Reliability
0 Normal
1 Maximize
Monetary Cost
0 Normal
1 Minimize
1 1 1 1 = Maximize Security

- **Version Number**

The 4-bit version number (VER) field defines the version of the IPv4 protocol, which, obviously, has the value of 4.

- **Header Length (HLEN)**

In IPv4 is a 4-bit field that tells us how long the header of a datagram is in terms of 4-byte words. **Since the header's length can vary, this field helps devices figure out where the header ends and the actual data begins.** To fit this length into 4 bits, we calculate it in 4-byte words. To get the actual length in bytes, you need to multiply the value in this field by 4.

- **Service Type**

Used to be called Type of Service (TOS) in the IP header, and **it determined how to treat the datagram**. In the late 1990s, the IETF changed it to provide Differentiated Services (DiffServ). (PPPDTRCO)

- **Total Length**:

This 16-bit field tells us the size of the entire IP datagram in bytes, including both the header and data. It can go up to 65,535 bytes, but most datagrams are smaller. It helps the receiving device know when the entire packet has arrived.

- **Identification, Flags, and Fragmentation** Offset:

These three fields are connected to breaking up large IP datagrams into smaller pieces when the network can't handle their size. We'll explain these fields in more detail in the next section when we talk about fragmentation.

- **Time-to-live**: To prevent data packets from endlessly traveling through the internet due to routing issues, the Time-to-live (TTL) field is used. It limits the number of routers a packet can pass through. The source sets an initial value, typically twice the maximum number of hops between hosts. Each router reduces this value by one. If it reaches zero, the router drops the packet.

- The "**Protocol**" field in TCP/IP identifies which type of data is carried in a packet. It uses an 8-bit number assigned to each protocol, helping devices know how to handle the packet's content. This field works like a label, ensuring the right protocol deals with the packet, similar to how port numbers operate at the transport layer.
- **Header Checksum** in IP ensures the accuracy of the header but not the payload. Errors in the IP header can lead to misdirection or fragmentation issues. This 16-bit checksum is recalculated at each router to account for changes in certain fields like TTL. Checksum calculation is discussed in detail in Chapter 5 for error detection.

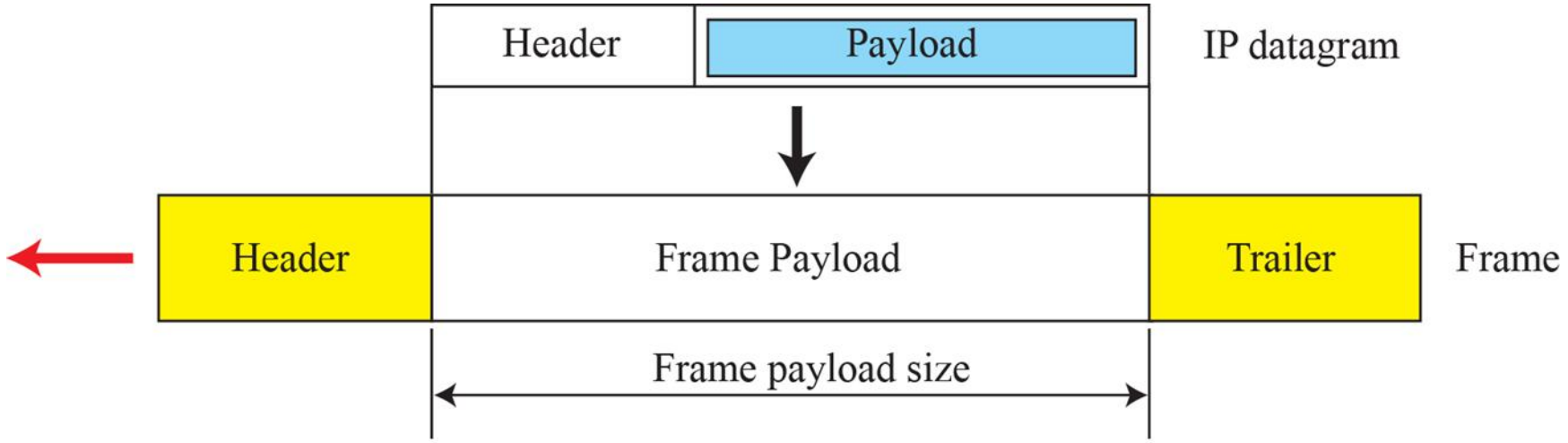
- **Source and Destination Addresses**: These 32-bit fields in an IP header specify the source and destination IP addresses. The source host knows its IP address, while the destination address is provided by the protocol using IP or through DNS. These addresses must stay unchanged during the datagram's journey. We'll dive deeper into IP addresses later in this chapter.
- **Options**: An IP datagram header can hold up to 40 bytes of optional information, useful for network testing and debugging. While not mandatory, all IP implementations must handle options if present. Having options can affect datagram handling and may require routers to recalculate the header checksum. We discuss one-byte and multi-byte options in more detail on the book's website.
- **Payload**: The payload, or data, is the primary content of a datagram. It's the information sent by other protocols using IP. Think of a datagram like a package: the payload is the package's content, and the header is like the label on the package.

Fragmentation

- A datagram can move through various networks. When a router gets a datagram, it takes out the IP data and works on it. Then, it puts it into a new frame. The frame's look and size are different based on the network it's on. If a router links a local network (LAN) to a wide-area network (WAN), it gets a LAN-style frame and sends out a WAN-style frame.

Maximum Transfer Unit (MTU)

- Different networks use their own rules for packaging data. Each network type has a limit on how big a piece of data, called a payload, can be put into its package. This limit is called the Maximum Transmission Unit (MTU), and it varies depending on the type of network. For example, a local network can handle a payload of up to 1500 bytes, but a wide-area network might allow more or less.



MTU: Maximum size of frame payload

- The Internet Protocol (IP) allows data packets to be as large as 65,535 bytes, which is good for future networks. However, in current networks with smaller limits (MTUs), these large packets must be broken into smaller parts, called "fragmentation." If a network with an even smaller limit is encountered, further splitting may occur. This means one data packet can be split multiple times before reaching its destination.
- This splitting can be done by the source computer or any router along the way. However, the task of reassembling these pieces into the original packet is done only by the destination computer. Each piece acts as an independent packet as it travels through the network.
- Even though fragmented packets can take different paths, they are designed to eventually reach the destination. It's more efficient to reassemble them at the end, rather than during the journey, to avoid unnecessary complexity.
- Fragmentation mainly involves breaking up the payload of an IP data packet. Most header information, except for a few options, must be copied in each fragment. The device doing the fragmentation needs to adjust three fields: flags, fragmentation offset, and total length. Everything else in the header remains the same. Also, the checksum value must be recalculated, regardless of how many times the data is split.

Fields Related to Fragmentation

- **Identification field**

The 16-bit ID field in a datagram helps identify where it comes from. To ensure each datagram is unique, the IP protocol uses a counter. This counter starts at a positive number and is increased by one for each datagram sent. As long as this counter stays in the computer's memory, uniqueness is guaranteed.

When a datagram gets split into smaller parts (fragments), all of them get the same ID number as the original datagram. This ID number helps the destination computer put the fragments back together. It knows that all fragments with the same ID should be assembled into one datagram.

- **Flag field**

The 3-bit flags field has three flags. The first one isn't used. The second flag, called the "**do not fragment**" bit (D bit), when set to 1, means the datagram shouldn't be broken into pieces. If it can't fit through a network, it's discarded, and an error message is sent back. When the D bit is set to 0, the datagram can be split into smaller parts if needed.

The third flag, called the "**more fragment**" bit (M bit), is set to 1 when there are more fragments to come after this one. If it's set to 0, it means this is the last part or the only part of the datagram.

Offset

The 13-bit **fragmentation offset field tells us where this piece fits in the whole datagram**, counting in chunks of 8 bytes. Imagine we have a datagram with 4000 bytes split into three parts. Here's how it works:

- The first piece covers bytes 0 to 1399 and has an offset of 0.
- The second piece covers bytes 1400 to 2799 and has an offset of 175.
- The third piece covers bytes 2800 to 3999 and has an offset of 350.

Imagine the fragments are like pieces of a puzzle sent from one place to another. Even if they arrive in a different order, the receiving end can still put the puzzle together correctly. This happens because:

- The identification and flags fields are the same in all fragments, except for the last one.
- The offset field in each fragment shows its position relative to the original data.

If a fragment itself gets split into smaller pieces, the offset is still relative to the original data. So, even if pieces take different paths and arrive out of order, they can be correctly reassembled.

To reassemble, follow these steps:

- a. The first fragment has an offset of 0.
- b. Divide the length of the first fragment by 8 to get the offset of the second fragment.
- c. Divide the total length of the first and second fragments by 8 to get the offset of the third fragment.
- d. Continue this process. The last fragment has its "more" bit set to 0.

- Remember that the value of the offset is measured in units of 8 bytes. **This is done because the length of the offset field is only 13 bits long and cannot represent a sequence of bytes greater than 8191.** This forces hosts or routers that fragment datagrams to **choose the size of each fragment so that the first byte number is divisible by 8.**

Q - A datagram of 3000 B (20B of IP header + 2980B IP payload) reached at router and must be forward to link with MTU of 500B. How many fragments will be generated and also write: MF, Offset, totle length value for all.

Security of IPv4 Datagrams

The IPv4 protocol, as well as the whole Internet, was started when the Internet users trusted each other. No security was provided for the IPv4 protocol. Today, however, the situation is different; the Internet is not secure anymore. There are three security issues that are particularly applicable to the IP protocol:

- packet sniffing
- packet modification
- IP spoofing.

Packet Sniffing

- Packet Sniffing **An intruder may intercept an IP packet and make a copy of it.** Packet sniffing is a passive attack, in which the attacker **does not change the contents of the packet**. This type of attack is very difficult to detect because the sender and the receiver may never know that the packet has been copied. Although packet sniffing cannot be stopped**, encryption of the packet can make the attacker's effort useless.** The attacker may still sniff the packet, but the content is not detectable

Packet Modification

- The 2nd type of attack is to modify the packet. The **attacker intercepts the packet, changes its contents, and sends the new packet to the receiver**. The receiver believes that the packet is coming from the original sender. This type of attack can be detected using a data integrity mechanism. The receiver, before opening and using the contents of the message, can use this mechanism to make sure that the packet has not been changed during the transmission.

IP Spoofing

- **A person with malicious intent can pretend to be someone else and send an IP packet that looks like it's from a different computer.** For example, they can send a packet to a bank, making it seem like it's from one of the bank's customers.

IPSec

The IP packets today can be protected from the previously mentioned attacks using a protocol called IPSec (IP Security). This protocol, which is used in conjunction with the IP protocol, creates a connection-oriented service between two entities in which they can exchange IP packets without worrying about the three attacks discussed above.

- **Defining Algorithms and Keys**. The two entities that want to create a secure channel between themselves can agree on some available algorithms and keys to be used for security purposes.
- **Packet Encryption.** The packets exchanged between two parties can be encrypted for privacy using one of the encryption algorithms and a shared key agreed upon in the first step. This makes the packet sniffing attack useless.

- **Data Integrity.** Data integrity guarantees that the packet is not modified during the transmission. If the received packet does not pass the data integrity test, it is discarded. This prevents the second attack, packet modification, described above.
- **Origin Authentication.** IPSec can authenticate the origin of the packet to be sure that the packet is not created by an imposter. This can prevent IP spoofing attacks as described above.

IPv4 Addresses

- The Internet address, known as an IP address, is like a label for devices on the Internet. It's a 32-bit code that tells the Internet where a device is. It's important to note **that this address is for the connection, not the device itself. If a device switches networks, its IP address can change.** Each IP address is unique and corresponds to one Internet connection. So**, if a device has two Internet connections, it has two IP addresses.** These addresses are universal, meaning any Internet-connected device must recognize them.

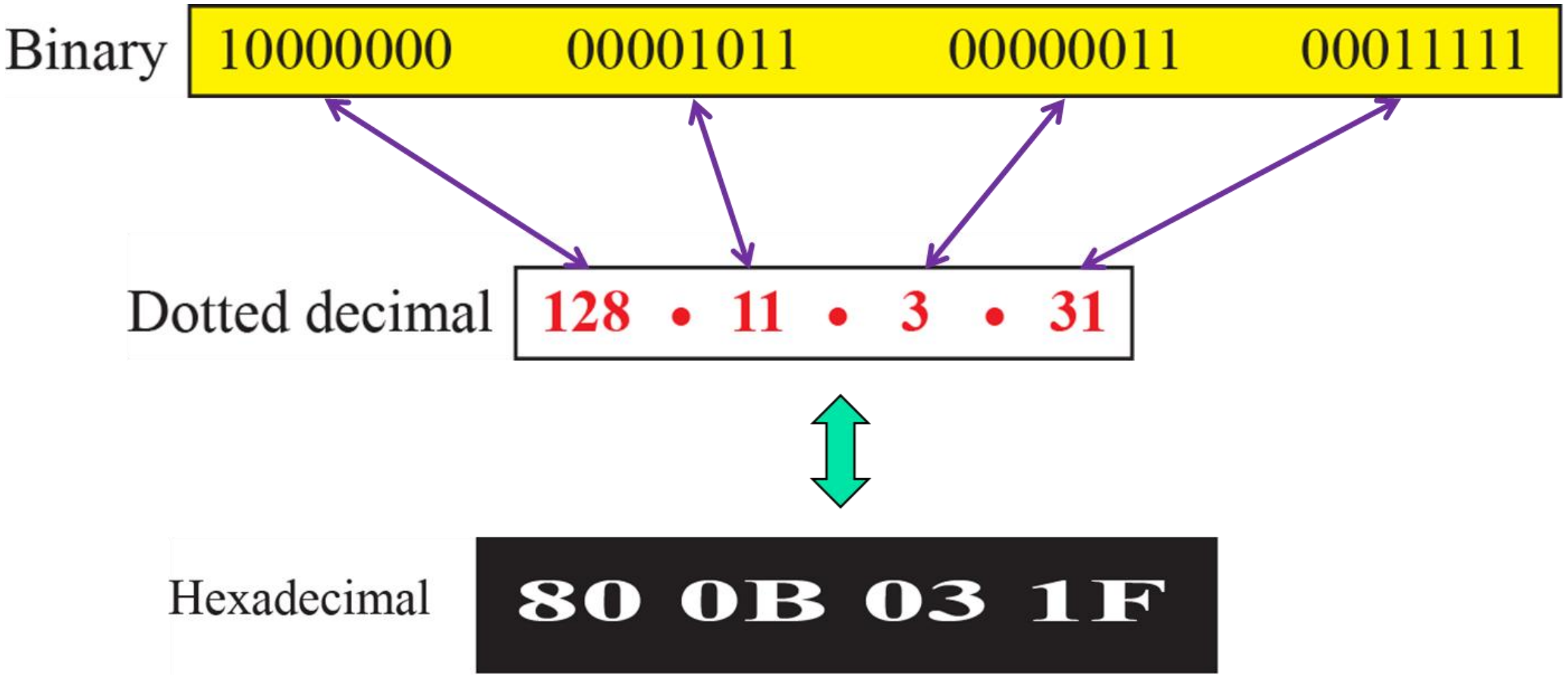
Address Space

- A protocol like IPv4 that defines addresses has an address space. An address space is the total number of addresses used by the protocol. If a protocol uses b bits to define an address, the address space is 2^b because each bit can have two different values (0 or 1). IPv4 uses 32-bit addresses, which means that the address space is 2^(32) or 4,294,967,296 (more than four billion). If there were no restrictions, more than 4 billion devices could be connected to the Internet.

Notation

IPv4 addresses can be written in three common ways: binary (0s and 1s), dotted-decimal (numbers separated by dots, base 256), and hexadecimal (numbers and letters, base 16).

- In binary, it's 32 bits with spaces to make it readable usually between each octet (8 bits).
- Dotted-decimal is the usual way, with dots separating numbers from 0 to 255.
- Hexadecimal uses numbers and letters and is common in network programming. Figure displays an IP address in these three ways.


Binary
10000000 00001011 00000011 00011111
Dotted decimal
128 • 11 • 3 • 31
Hexadecimal
80 0B 03 1F

There are two prevalent notations to show an IPv4 address: binary notation and dotted decimal notation.

Binary Notation: 01110101 10010101 00011101 00000010

Dotted-Decimal Notation: 117.149.29.2

Notation of IPv4 address: A.B.C.D (Only 4 octets)

$0 \leq A,B,C,D \leq 255$

0.0.0.0 to 255.255.255.255

Valid IP Addresses

10.10.56.80

240.230.220.89

1.2.3.4

99.88.67.89

100.200.89.90

Invalid IP Address

56.89.1.2.5

10.065.34.56

200.28.256.8

Question
111.56.045.78
221.34.7.8.20
75.45.301.14
11100010.23.14.67

Conversion D-B, B-D

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

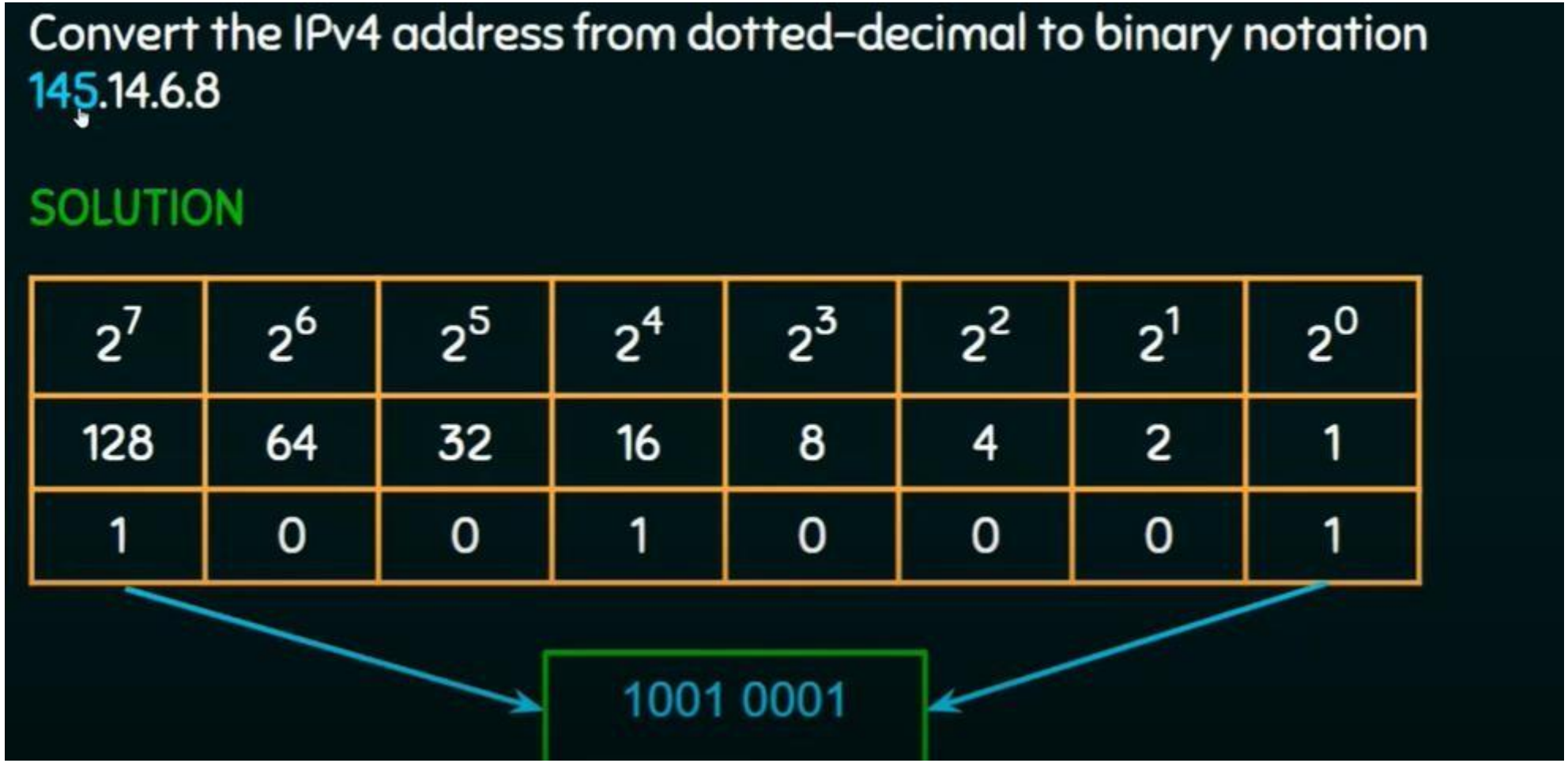
Example 1

Convert the IPv4 address from binary to dotted-decimal notation.
10000001 00001011 01001011 11101111

SOLUTION

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	1

Example 2

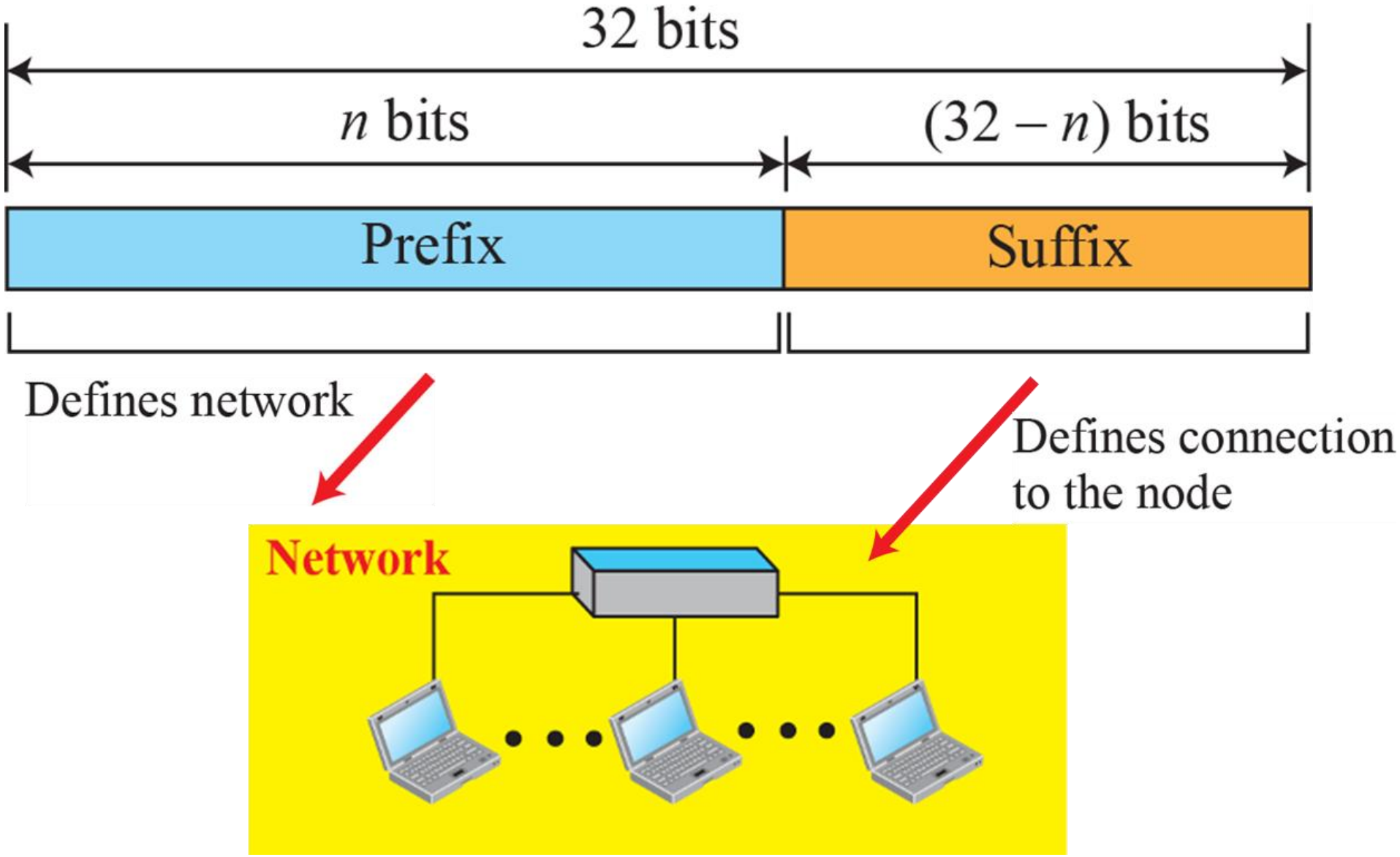

Convert the IPv4 address from dotted-decimal to binary notation
145.14.6.8
SOLUTION
2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
128 64 32 16 8 4 2 1
1 0 0 1 0 0 0 1
1001 0001

Hierarchy in Addressing

In networks like telephone or postal systems, the addressing works in a structured way. For example, a postal address includes country, state, city, street, house number, and recipient's name. Similarly, a phone number has country code, area code, local exchange, and the specific connection.

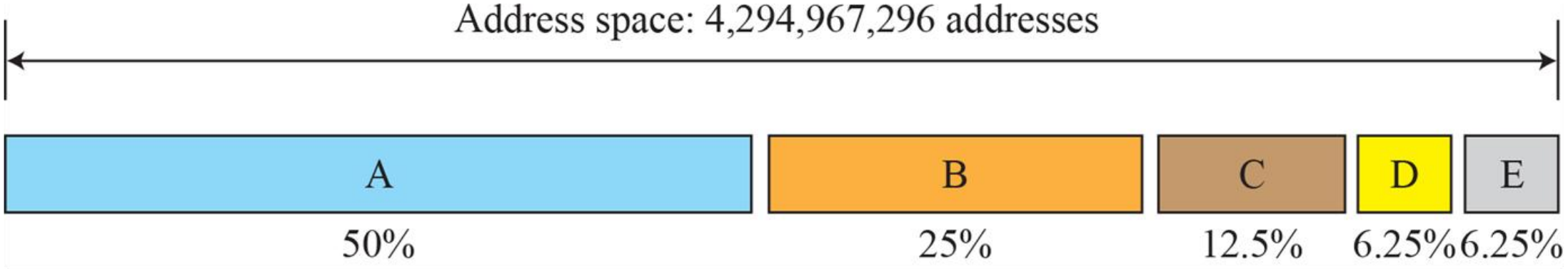
Now, in the case of a 32-bit IPv4 address, it's also structured but in a simpler way. It's divided into two parts: the first part (prefix) defines the network, and the second part (suffix) defines the device on that network. The prefix can be either a fixed or variable length.

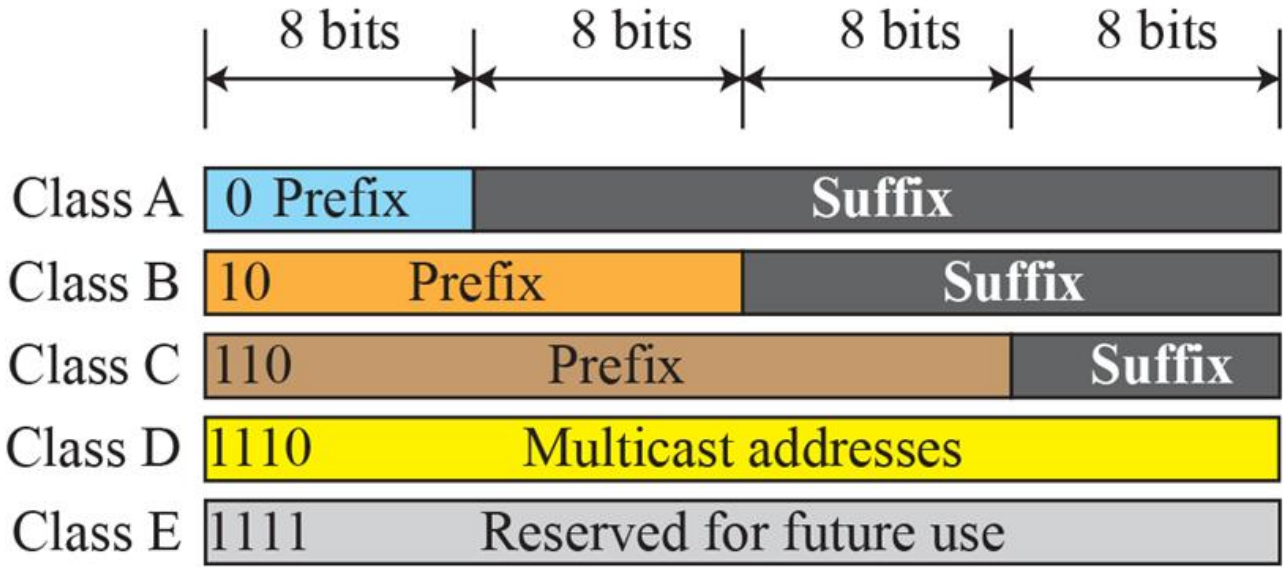
In the past, **IPv4 used fixed-length prefixes (classful addressing), but that's outdated. Nowadays, it uses variable-length prefixes (classless addressing).** We'll first touch on the old method and then focus on the new one.


32 bits
n bits
(32 – n) bits
Prefix
Suffix
Defines network
Defines connection to the node
Network

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), as shown in Figure. This scheme is referred to as classful addressing.





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 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.

- **Class B**

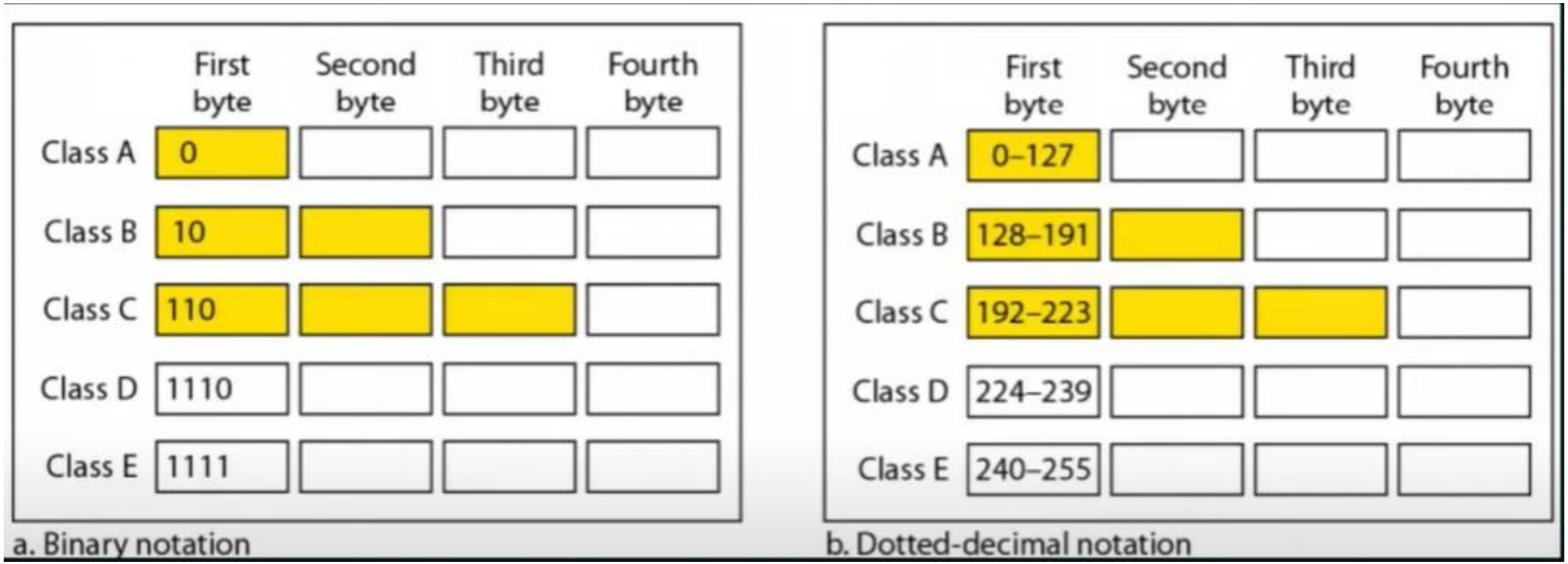
the network length is 16 bits, but since the first two bits, which are (10)_2, define the class, we can have only 14 bits as the network identifier. This means there are only 2^14 = 16,384 networks in the world that can have a class B address.

- **Class C**

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 D and Class E**

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.


First byte
Second byte
Third byte
Fourth byte
Class A
0
Class B
10
Class C
110
Class D
1110
Class E
1111
a. Binary notation
First byte
Second byte
Third byte
Fourth byte
Class A
0–127
Class B
128–191
Class C
192–223
Class D
224–239
Class E
240–255
b. Dotted-decimal notation

Classes of IPv4 address

Address Class	1st Octet range in decimal	1st Octet bits (Blue Dots do not change)	Network (N) and Host (H) Portion	Default mask (Decimal)	Number of possible networks and hosts per network
A	0–127	00000000 - 01111111	N.H.H.H	255.0.0.0	128 Nets (2^7) 16,777,214 hosts (2^{24}–2)
B	128–191	10000000 - 10111111	N.N.H.H	255.255.0.0	16,384 Nets (2^{14}) 65,534 hosts (2^{16}–2)
C	192–223	11000000 - 11011111	N.N.N.H	255.255.255.0	2,09,150 Nets (2^{21}) 254 hosts (2^8–2)
D	224–239	11100000 - 11101111	NA (Multicast)	-	-
E	240–255	11110000 - 11111111	NA (Experimental)	-	-

Address Depletion

- Classful addressing became outdated because it led to address shortage. Internet addresses were not handed out efficiently, so we ran out of them fast. For instance, class A addresses were given to only 128 organizations, but each one got a massive chunk of addresses, which most didn't need. Class B addresses were for midsize organizations, but many went unused. Class C addresses had too few usable addresses for companies. Class E addresses were hardly used at all, wasting the entire class. This inefficiency in address distribution caused problems, which is why we moved to a different system.

Subnetting and Supernetting

- To deal with the problem of running out of addresses, two strategies were suggested: subnetting and supernetting.
- Subnetting **involves breaking up a big address block (like class A or class B) into smaller parts, creating more subnetworks with shorter prefixes.** This also allowed unused addresses to be shared among different organizations. However, many **large organizations didn't like sharing their unused addresses**, so this idea didn't work well.
- Supernetting, on the other hand, aimed to **combine multiple class C blocks into a larger one to provide more addresses to organizations.** But this made it harder to route data packets effectively, so it wasn't a successful solution either.

Advantage of Classful Addressing

- Classful addressing, despite its issues and eventual obsolescence, had one clear benefit:

it made it easy to determine the class of an address and, because each class had a fixed prefix length, you could instantly know the prefix length from the address itself. In simple terms, the address itself contained all the information needed to figure out its prefix and suffix; no extra details were required.

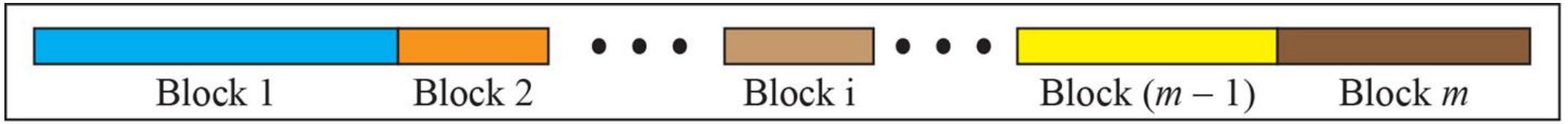
Classless Addressing

Address Deletion and the need for change

- Subnetting and supernetting in classful addressing didn't effectively address address depletion.
- The growth of the Internet highlighted the need for a larger address space.
- IPv6 was developed as a long-term solution, but a short-term solution called classless addressing was also introduced.
- Classless addressing removed the class distinctions in address distribution to address depletion.
- Internet Service Providers (ISPs) played a role in the adoption of classless addressing.

Classless Addressing: A flexible solution

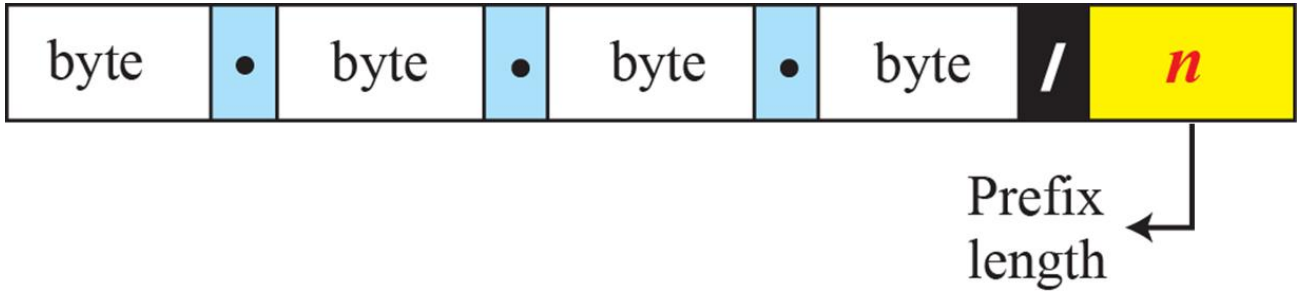
- Classless addressing introduced variable-length address blocks without class distinctions.
- Address space is divided into variable-length blocks, where the prefix defines the network, and the suffix defines the device.
- Blocks can vary in size from 2^0 to 2^32 addresses.
- Prefix length ranges from 0 to 32, with smaller prefixes indicating larger networks and vice versa.
- Classless addressing can be applied to classful addressing, making classful addressing a special case of classless addressing.



Address space

Prefix Length: Slash Notation

- In classless addressing, determining the prefix length is crucial.
- Unlike classful addressing, the prefix length is not inherent in the address.
- To specify the prefix length, it's added to the address using slash notation or CIDR (pronounced cider) strategy.
- The format is like this: address/prefix_length.
- This means that in classless addressing, an address alone doesn't fully define its network; we also need to provide the prefix length.



Examples:

12.24.76.8/**8**

23.14.67.92/**12**

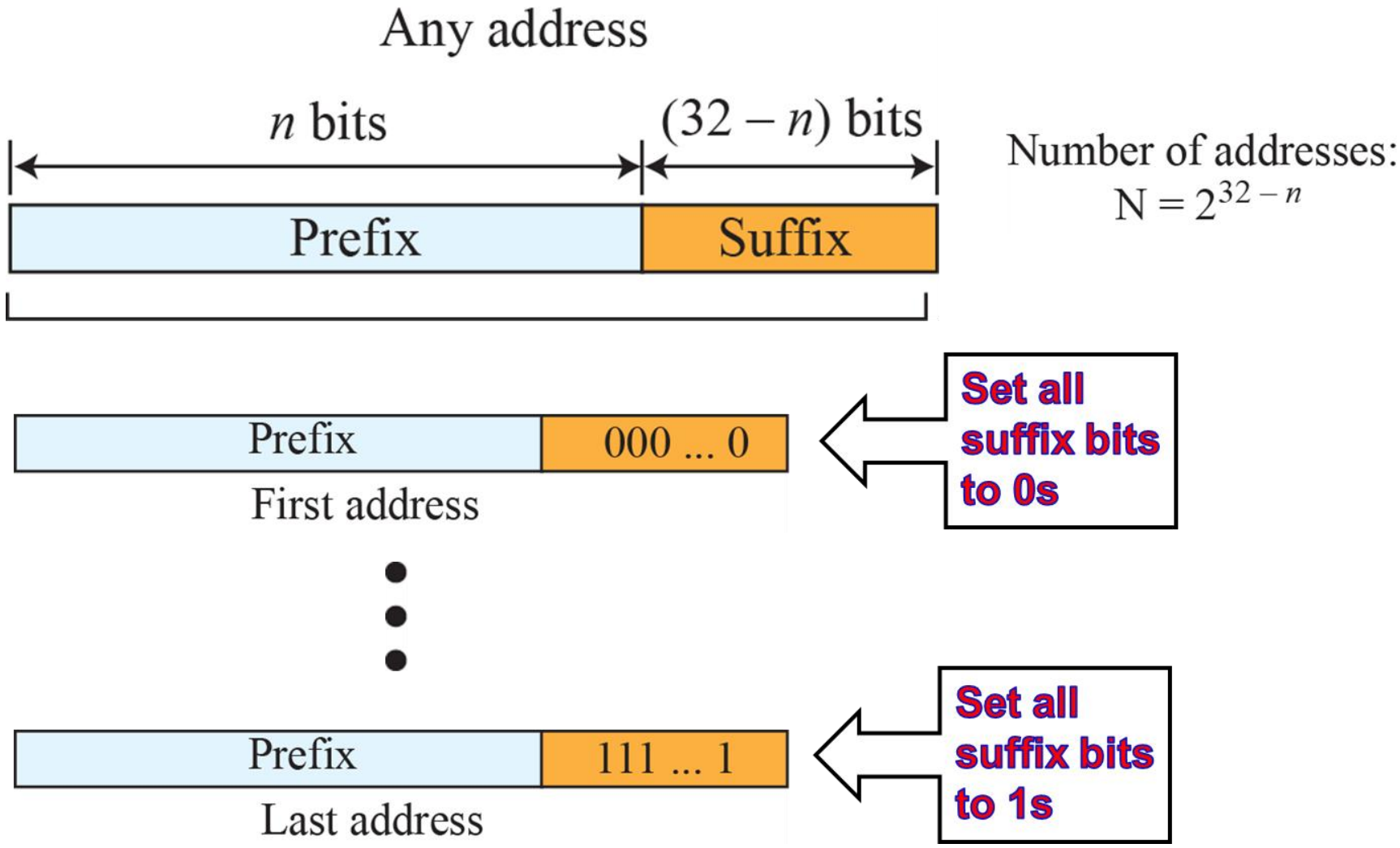
220.8.24.255/**25**

Extracting Information from an Address

Given any address in the block, we normally like to know three pieces of information about the block to which the address belongs:

- the number of addresses
- the first address in the block
- the last address.

If the value of prefix length, n, is given, we can easily find these three pieces of information


Any address
n bits
(32 – n) bits
Prefix
Suffix
Number of addresses:
N = 2^(32 – n)
Prefix
000 ... 0
First address
Set all suffix bits to 0s
Prefix
111 ... 1
Last address
Set all suffix bits to 1s

Method for extracting information from an address

- The number of addresses in the block is found as N = 2^(32−n).
- To find the first address, we keep the n leftmost bits and set the (32 − n) rightmost bits all to 0s.
- To find the last address, we keep the n leftmost bits and set the (32 − n) rightmost bits all to 1s.

Example

A classless address is given as 167.199.170.82/27. We can find the above three pieces of information as follows. The number of addresses in the network is $2^{32-n} = 2^5 = 32$ addresses. The first address can be found by keeping the first 27 bits and changing the rest of the bits to 0s.

Address: 167.199.170.82/27	10100111	11000111	10101010	01010010
First address: 167.199.170.64/27	10100111	11000111	10101010	01000000

The last address can be found by keeping the first 27 bits and changing the rest of the bits to 1s.

Address: 167.199.170.82/27	10100111	11000111	10101010	01011111
Last address: 167.199.170.95/27	10100111	11000111	10101010	01011111

Address Mask

Another way to find the first and last addresses in the block is to use the address mask. The address mask is a 32-bit number in which the n leftmost bits are set to 1s and the rest of the bits (32 − n) are set to 0s. A computer can easily find the address mask because it is the complement of (2^(32 − n) − 1). The reason for defining a mask in this way is that it can be used by a computer program to extract the information in a block, using the three bit-wise operations NOT, AND, and OR.

1. The number of addresses in the block N = NOT (Mask) + 1.
2. The first address in the block = (Any address in the block) AND (Mask).
3. The last address in the block = (Any address in the block) OR [(NOT (Mask)].

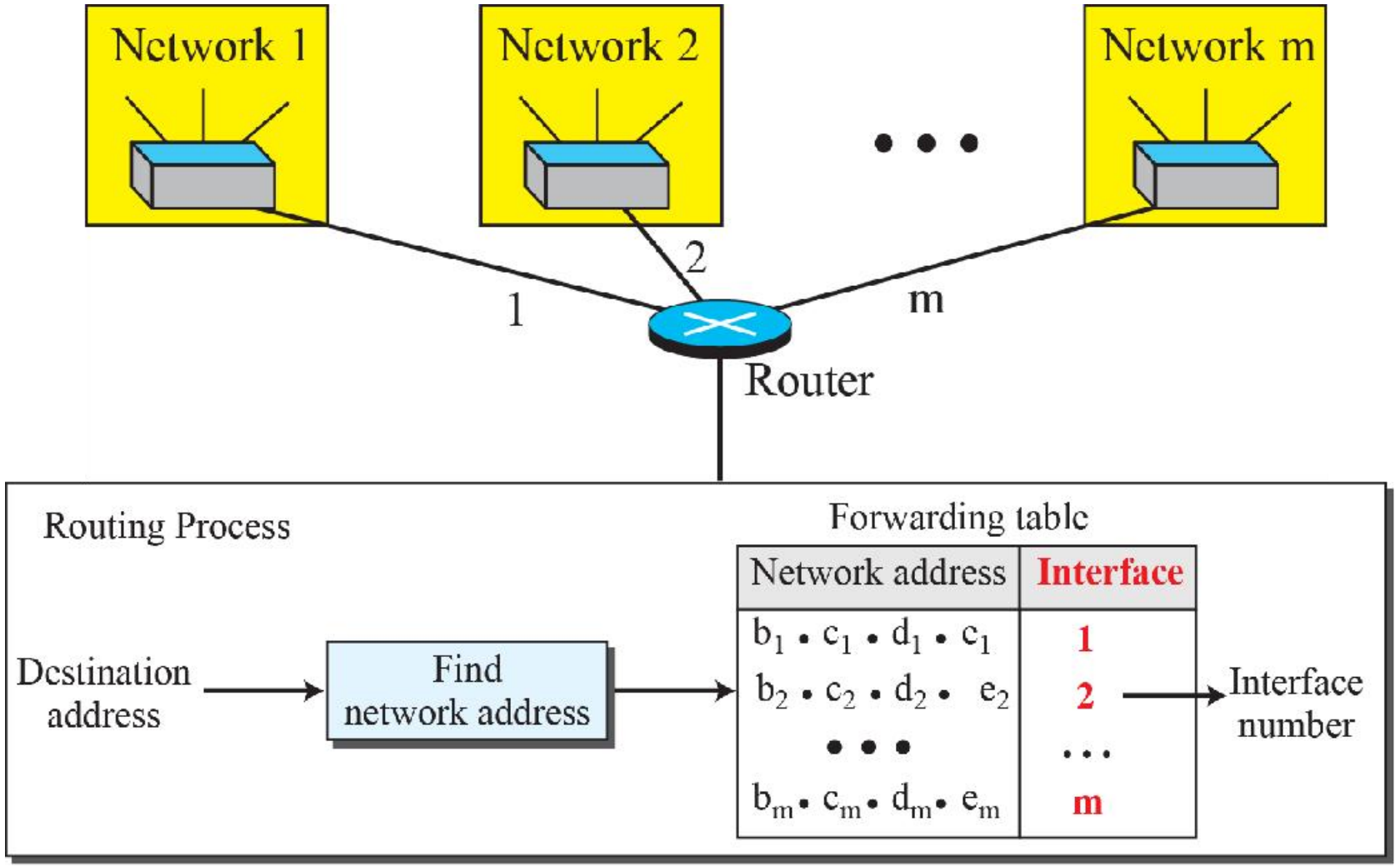
Example

The mask in dotted-decimal notation is 255.255.255.224
The AND, OR, and NOT operations can be applied to individual bytes using calculators.

Number of addresses in the block: N = NOT (mask) + 1= 0.0.0.31 + 1 = 32 addresses
First address: First = (address) AND (mask) = 167.199.170. 82
Last address: Last = (address) OR (NOT mask) = 167.199.170. 255

Network Address

- The provided examples demonstrate our ability to retrieve block-related information from any given address.
- The network address, which is the first address, is vital for routing packets to their destination networks.
- Let's assume there are m networks in an internet, each connected to a router with m interfaces.
- When a packet arrives at the router from any source host, the router must determine which network to send the packet to and which interface to use.
- The packet employs a different strategy to reach its final destination host once it reaches the network.
- Figure visually represents this process.
- **After identifying the network address, the router checks its forwarding table to find the correct interface for sending out the packet.**
- Each network is uniquely identified by its network address.


Network 1
Network 2
Network m
1
2
m
Router
Routing Process
Forwarding table
Network address
Interface
Destination address
Find network address
Interface number
1
2
m

Block Allocation

The next concern with classless addressing is how blocks are handed out. The job of giving out these blocks falls on a global authority named ICANN. But, ICANN doesn't usually hand out addresses to regular internet users. Instead, it gives a big chunk of addresses to an ISP (or a similar big organization that acts like an ISP). To make CIDR work well, there are two important rules that need to be followed for these allocated blocks.

1. The number of requested addresses, N, needs to be a power of 2. The reason is that $N = 2^{(32 - n)}$ or $n = 32 - \log_2(N)$. If N is not a power of 2, we cannot have an integer value for n.
2. The requested block needs to be allocated where there are a contiguous number of available addresses in the address space. However, there is a restriction on choosing the first address in the block. The first address needs to be divisible by the number of addresses in the block. The reason is that the first address needs to be the prefix followed by (32 − n) number of 0s. The decimal value of the first address is then

$$\text{first address} = (\text{prefix in decimal}) \times 2^{(32 - n)} = (\text{prefix in decimal}) \times N$$

Example

An ISP has requested a block of 1000 addresses.

Since 1000 is not a power of 2, 1024 addresses are granted. The prefix length is calculated as $n = 32 - \log_2(1024) = 22$. An available block, 18.14.12.0/22, is granted to the ISP. It can be seen that the first address in decimal is 302,910,464, which is divisible by 1024.

Subnetting

Subnetting allows for the creation of additional layers in the network hierarchy. When an organization or ISP is given a block of addresses, **they can split it into smaller sub-blocks and allocate each of these to separate subnetworks**. Importantly, there's no limit to how deep this hierarchy can go. A subnetwork can be further divided into sub-subnetworks, and the process can continue with sub-sub-subnetworks, and so forth.

Designing Subnets The subnetworks in a network should be carefully designed to enable the routing of packets. We assume the total number of addresses granted to the organization is N, the prefix length is n, the assigned number of addresses to each subnetwork is N_{sub}, and the prefix length for each subnetwork is n_{sub}. Then the following steps need to be carefully followed to guarantee the proper operation of the subnetworks.

- The number of addresses in each subnetwork should be a power of 2.
- The prefix length for each subnetwork should be found using the following formula:

$$n_{sub} = 32 - \log_2 N_{sub}$$

- The starting address in each subnetwork should be divisible by the number of addresses in that subnetwork. This can be achieved if we first assign addresses to larger subnetworks.

Finding Information about Each Subnetwork

After designing the subnetworks, the information about each subnetwork, such as first and last address, can be found using the process we described to find the information about each network in the Internet.

EXAMPLE: An organization is granted a block of addresses with the beginning address 14.24.74.0/24. The organization needs to have 3 subblocks of addresses to use in its three subnets: one subblock of 10 addresses, one subblock of 60 addresses, and one subblock of 120 addresses. Design the subblocks.

Solution

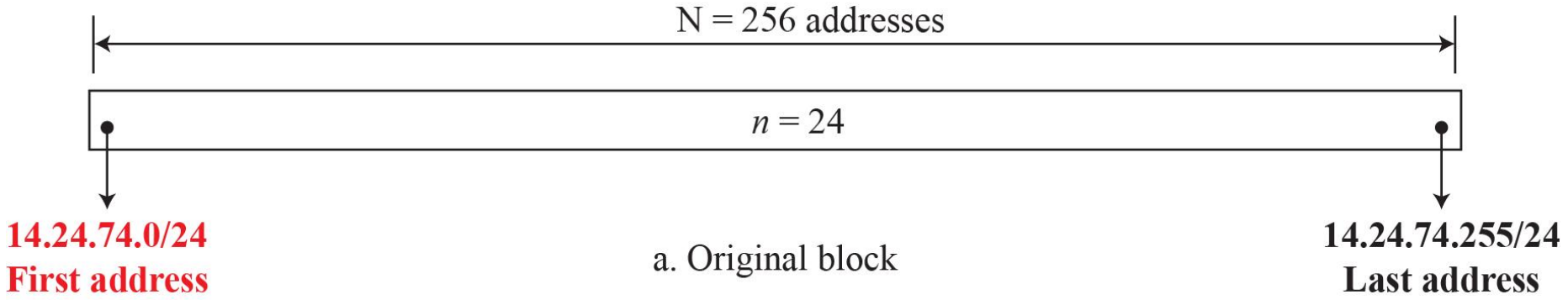
There are $2^{32-24} = 256$ addresses in this block. The first address is 14.24.74.0/24; the last address is 14.24.74.255/24. To satisfy the third requirement, we assign addresses to subblocks, starting with the largest and ending with the smallest one.

a. The number of addresses in the largest subblock, which requires 120 addresses, is not a power of 2. We allocate 128 addresses. The subnet mask for this subnet can be found as $n_1 = 32 - \log_2 128 = 25$. The first address in this block is 14.24.74.0/25; the last address is 14.24.74.127/25.

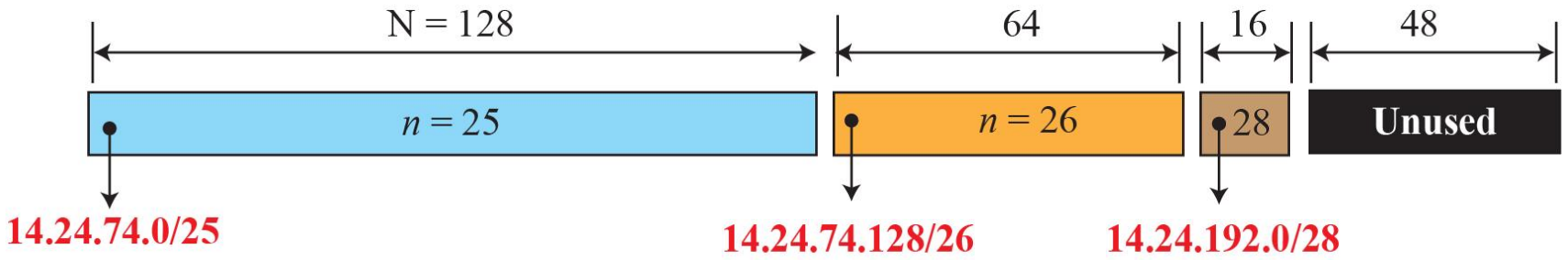
b. The number of addresses in the second largest subblock, which requires 60 addresses, is not a power of 2 either. We allocate 64 addresses. The subnet mask for this subnet can be found as $n_2 = 32 - \log_2 64 = 26$. The first address in this block is 14.24.74.128/26; the last address is 14.24.74.191/26.

c. The number of addresses in the smallest subblock, which requires 10 addresses, is not a power of 2. We allocate 16 addresses. The subnet mask for this subnet can be found as $n_1 = 32 - \log_2 16 = 28$. The first address in this block is 14.24.74.207/28; the last address is 14.24.74.207/28.

If we add all addresses in the previous subblocks, the result is 208 addresses, which means 48 addresses are left in reserve. The first address in this range is 14.24.74.208. The last address is 14.24.74.255. We don't know about the prefix length yet. Figure 4.36 shows the configuration of blocks. We have shown the first address in each block.



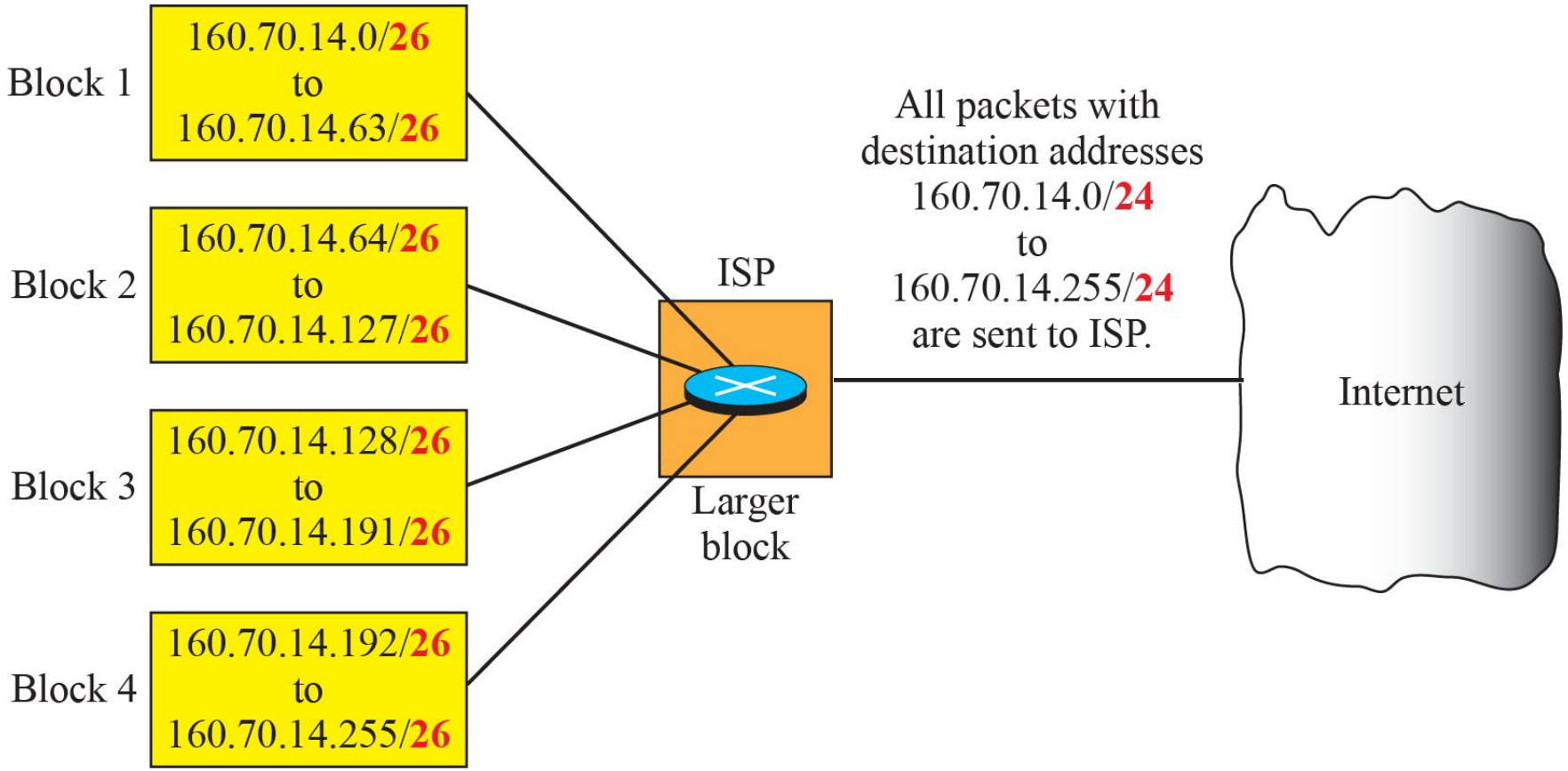
a. Original block



b. Subblocks

Address Aggregation

One of the advantages of the CIDR strategy is address aggregation (sometimes called **address summarization or route summarization**). **When blocks of addresses are combined to create a larger block, routing can be done based on the prefix of the larger block.** ICANN assigns a large block of addresses to an ISP. Each ISP in turn divides its assigned block into smaller subblocks and grants the subblocks to its customers.


Block 1
160.70.14.0/26
to
160.70.14.63/26
Block 2
160.70.14.64/26
to
160.70.14.127/26
Block 3
160.70.14.128/26
to
160.70.14.191/26
Block 4
160.70.14.192/26
to
160.70.14.255/26
ISP
Larger
block
All packets with
destination addresses
160.70.14.0/24
to
160.70.14.255/24
are sent to ISP.
Internet

Special Addresses

Before finishing the topic of addresses in IPv4, we need to mention five special addresses that are used for special purposes: this-host address, limited-broadcast address, loopback address, private addresses, and multicast addresses.

This-host Address

The only address in the block 0.0.0.0/32 is called the this-host address. It is used **whenever a host needs to send an IP datagram but it does not know its own address** to use as the source address.

Limited-broadcast Address.

The only address in the block 255.255.255.255/32 is called the limited-broadcast address. It is used **whenever a router or a host needs to send a datagram to all devices in a network**. The routers in the network, however, block the packet having this address as the destination; **the packet cannot travel outside the network.**

Loopback Address.

The block 127.0.0.0/8 is called the loopback address. **A packet with one of the addresses in this block as the destination address never leaves the host; it will remain in the host**. Any address in the block is used to test a piece of software in the machine. For example, we can write a client and a server program in which one of the addresses in the block is used as the server address. We can test the programs using the same host to see if they work before running them on different computers.

Private addresses

Are a specific range of IP addresses reserved for use within private networks. These addresses are not routable on the public Internet, meaning routers on the Internet will not forward packets with private IP addresses. Instead, they are intended for use within closed, local networks, such as home networks, corporate intranets, or other internal networks.

There are three main blocks of private IP addresses defined in the IPv4 address space:

10.0.0.0 to 10.255.255.255
172.16.0.0 to 172.31.255.255
192.168.0.0 to 192.168.255.255

Multicast addresses

The block 224.0.0.0/4 is reserved for multicast addresses. Multicast addresses in networking refer to a specific range of IP addresses designated for multicast communication. Multicast allows a host to send a s**ingle message to a selected group of hosts**, rather than sending it to every host on the network. It's an efficient way to distribute information to multiple recipients simultaneously

In IPv4, multicast addresses are within the range of 224.0.0.0 to 239.255.255.255. The addresses in this range are reserved for various purposes, and some are well-known for specific applications or protocols.

For example:

224.0.0.1 is the "All Hosts" multicast group, meaning all hosts on the network.
224.0.0.2 is the "All Routers" multicast group, used for communication with all routers on the network.

EXAMPLE: in Figure shows how four small blocks of addresses are assigned to four organizations by an ISP. The ISP combines these four blocks into one single block and advertises the larger block to the rest of the world. **Any packet destined for this larger block should be sent to this ISP**. It is the responsibility of **the ISP to forward the packet to the appropriate organization**. This is similar to routing we can find in a postal network. All packages coming from outside a country are sent first to the capital and then distributed to the corresponding destination.

Dynamic Host Configuration Protocol (DHCP)

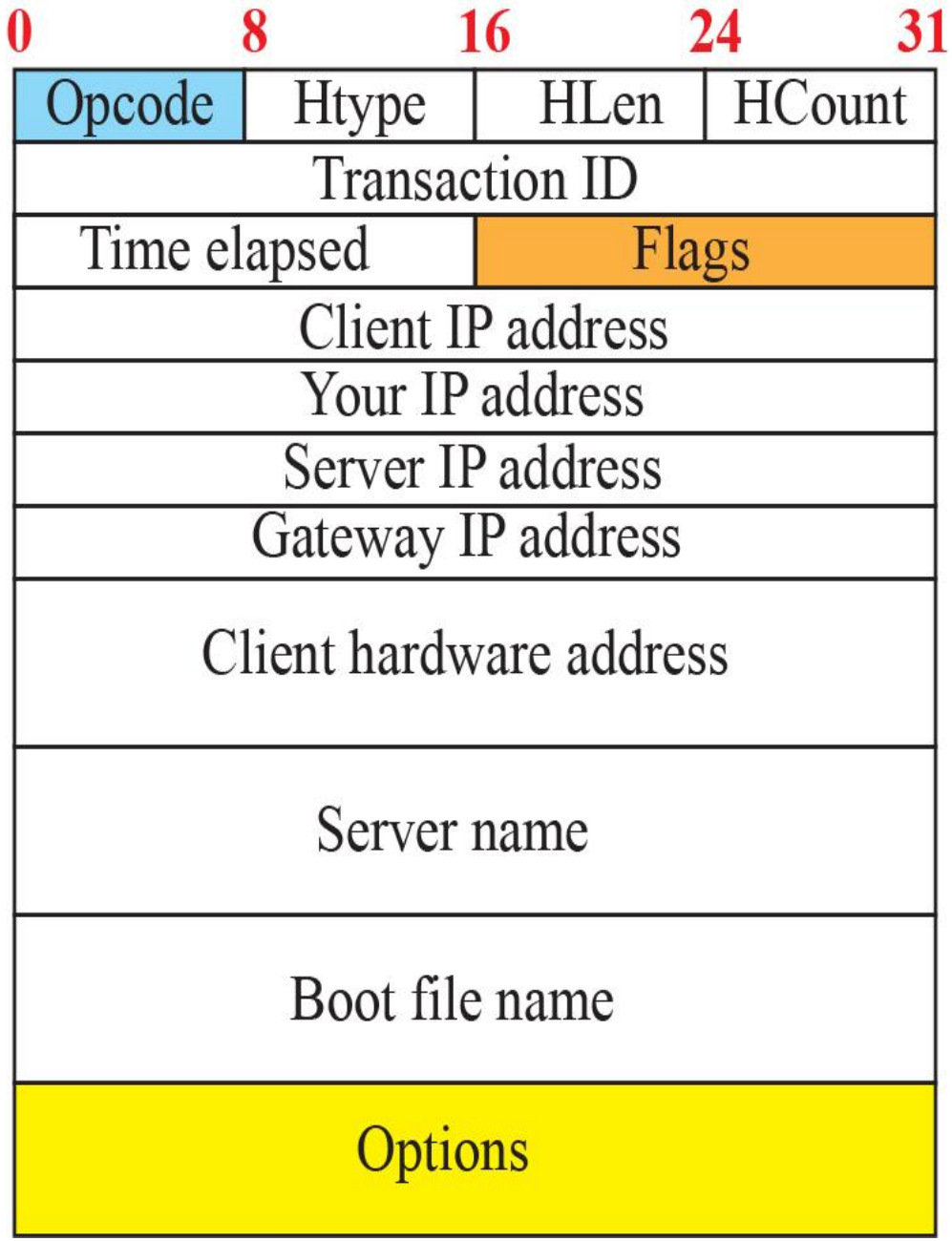
Big organizations and ISPs can get a bunch of addresses from ICANN, while smaller ones get them from ISPs. Once an organization gets its address block, the network admin can either manually assign addresses to devices or use DHCP, a handy protocol that makes things easy. **DHCP is like a plug-and-play system and can be set up to give devices permanent or temporary addresses.**

For instance, if you're at a hotel, DHCP can quickly give your laptop a temporary internet address. It's also handy for ISPs, letting them serve more households than they have addresses for.

Besides the IP address, **computers need to know a few more things like the network prefix, the address of a router to talk to other networks, and the address of a name server for using names instead of numbers.** DHCP can handle all these details for your devices.

DHCP Message Format

DHCP is a client-server protocol in which the client **sends a request message and the server returns a response message**. Before we discuss the operation of DHCP, let us show the general format of the DHCP message in Figure. Most of the fields are explained in the figure, but we need to discuss the option field, which plays a very important role in DHCP.



Fields:

Opcode: Operation code, request (1) or reply (2)

Htype: Hardware type (Ethernet, ...)

HLen: Lengh of hardware address

HCount: Maximum number of hops the packet can travel

Transaction ID: An integer set by 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

Imagine you're passing notes between two friends, a client, and a server. The client wants to know some information, and the server wants to provide it.

1. **64-byte Option Field**: Think of it as a space on the note where you can add extra details or secret messages. It's like a bonus section.

2. **Magic Cookie**: This is like a special code (99.130.83.99) hidden in the note. When the client reads the note, it checks for this code. If it finds it, the next part is important info.

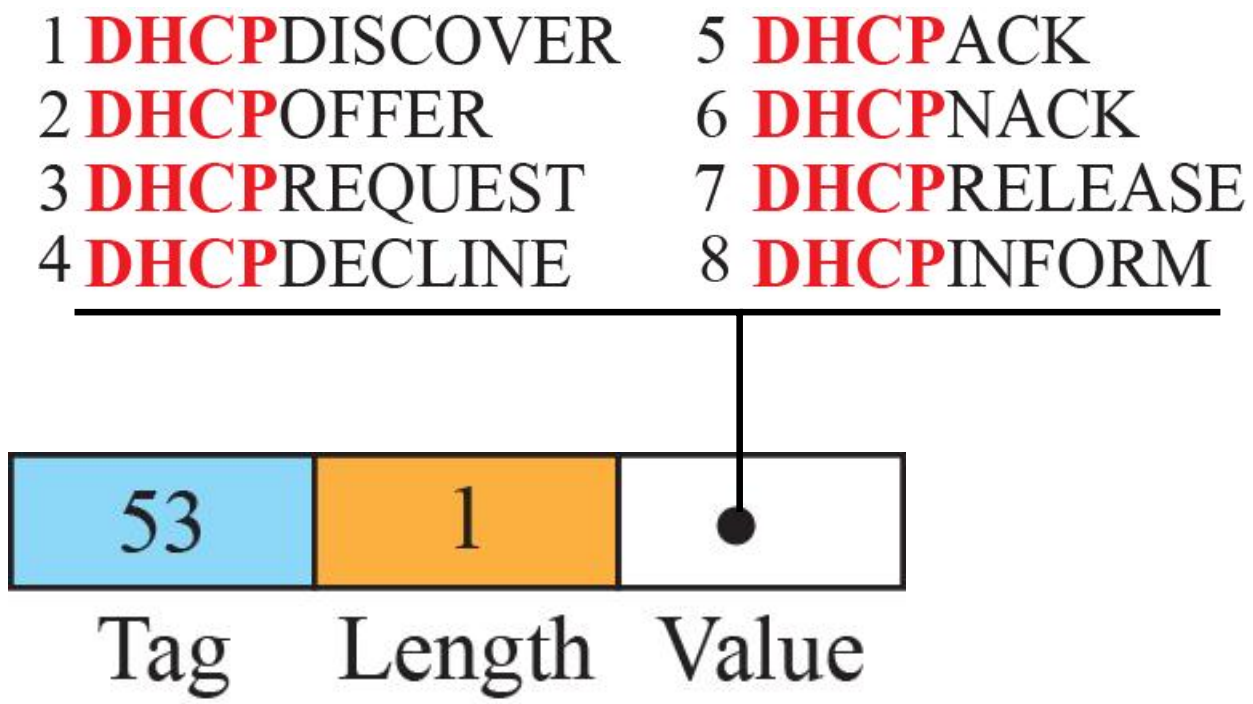
3. **Options (Next 60 Bytes)**: Following the magic code, there are 60 characters of info. Each piece of info has three parts: a tag, a length, and a value.

    - **Tag Field (1 byte)**: It's like a label telling you what the info is about.

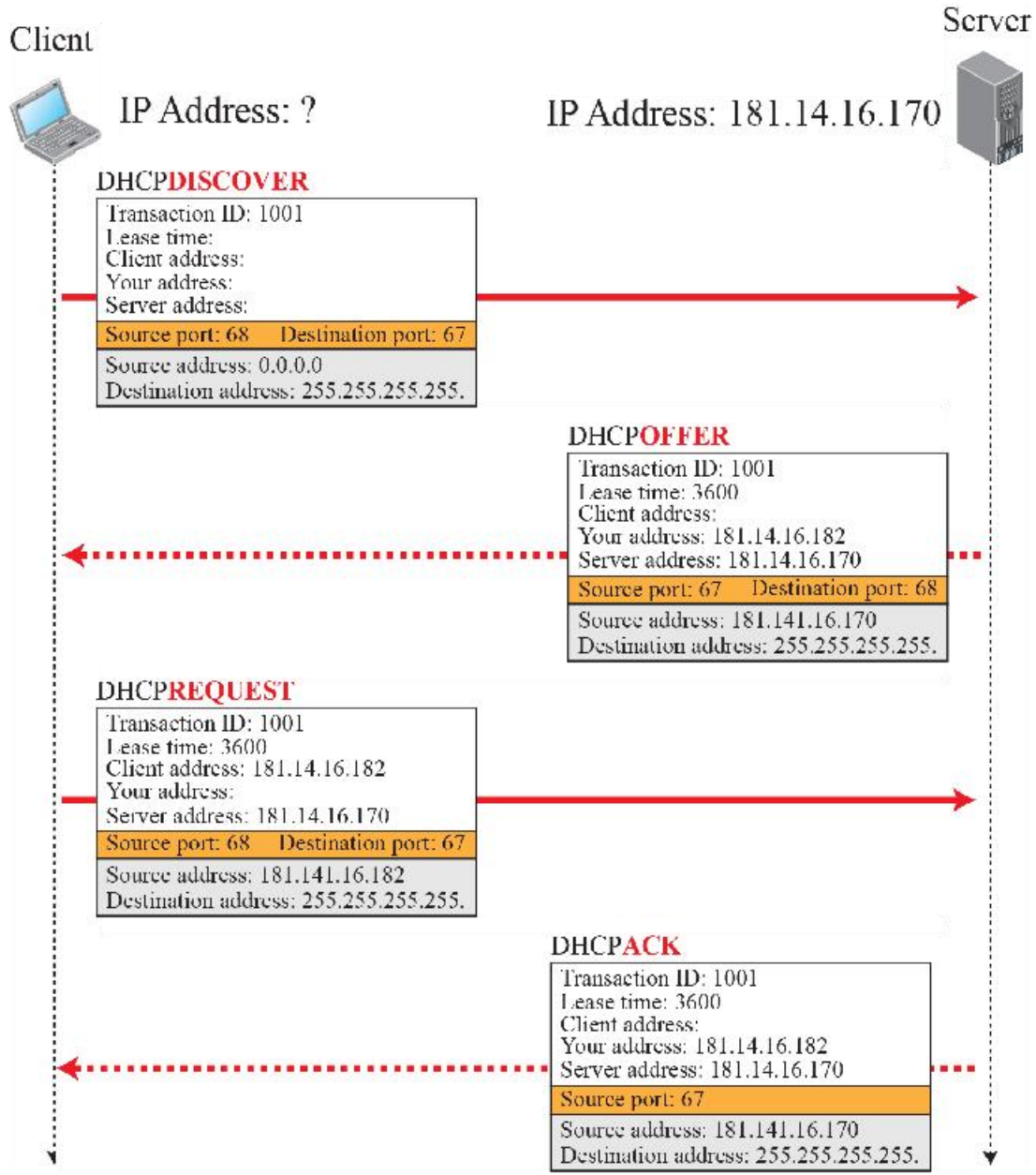
    - **Length Field (1 byte)**: It says how long the info is.

    - **Value Field (Variable Length)**: The actual info.

4. **Tag Field 53**: If the label says 53, it means the info is about one of 8 specific message types. The note has a guide (like a manual) on the next page explaining what each type means.


1 DHCPDISCOVER
2 DHCPOFFER
3 DHCPREQUEST
4 DHCPDECLINE
5 DHCPACK
6 DHCPNACK
7 DHCPRELEASE
8 DHCPINFORM
53
1
Tag
Length
Value

DHCP Operation



Legend

Application
UDP
IP

Note:
Only partial information is given.

1. The joining host creates a **DHCPDISCOVER** message in which only the transaction ID field is set to a random number. **No other field can be set because the host has no knowledge to do so.** This message is **encapsulated in a UDP user datagram** with the source port set to 68 and the destination port set to 67. We will discuss the reason for using two well-known port numbers later. The user datagram is encapsulated in an IP datagram with the source address set to 0.0.0.0 (“this host”) and the destination address set to 255.255.255.255 (broadcast address). **The reason is that the joining host knows neither its own address nor the server address.**

2. The DHCP server or servers (if more than one) responds with a **DHCPOFFER** message in which the your-IP-address field defines the offered IP address for the joining host and the server-IP-address includes the IP address of the server. **The message also includes the lease time for which the host can keep the IP address.** This message is **encapsulated in a user datagram with the same port numbers**, but in the reverse order. The user datagram in turn is encapsulated in a datagram with the server address as the source IP address, **but the destination address is a broad cast address, in which the server allows other DHCP servers to receive the offer and give a better offer if they can.**

3. The joining host receives **one or more offers and selects the best of them**. The joining host then sends a **DHCPREQUEST** message to the server that has given the best offer. The fields with known value are set. The message is encapsulated in a user datagram with port numbers as the first message. The user datagram is encapsulated in an IP datagram with the source address set to the new client address, but **the destination address still is set to the broadcast address to let the other servers know that their offer was not accepted.**

4. Finally, the selected server responds with an a **DHCPACK** message to the client if the offered IP address is valid. If the server cannot keep its offer (for example, if the address is offered to another host in between), the server sends a **DHCPNACK** message and the client needs to repeat the process. **This message is also broadcast to let other servers know that the request is accepted or rejected.**

Two well-known port

DHCP uses two well-known ports (68 and 67) to avoid confusion. **Port 68 is chosen for the client because the server's response is broadcasted**, and using a well-known port **prevents mix-ups**. If both DHCP and another client use the same temporary port, it could lead to errors. Well-known ports help avoid this by using a different range. **If two DHCP clients are active simultaneously, their messages can be distinguished by the transaction ID.**

Using FTP

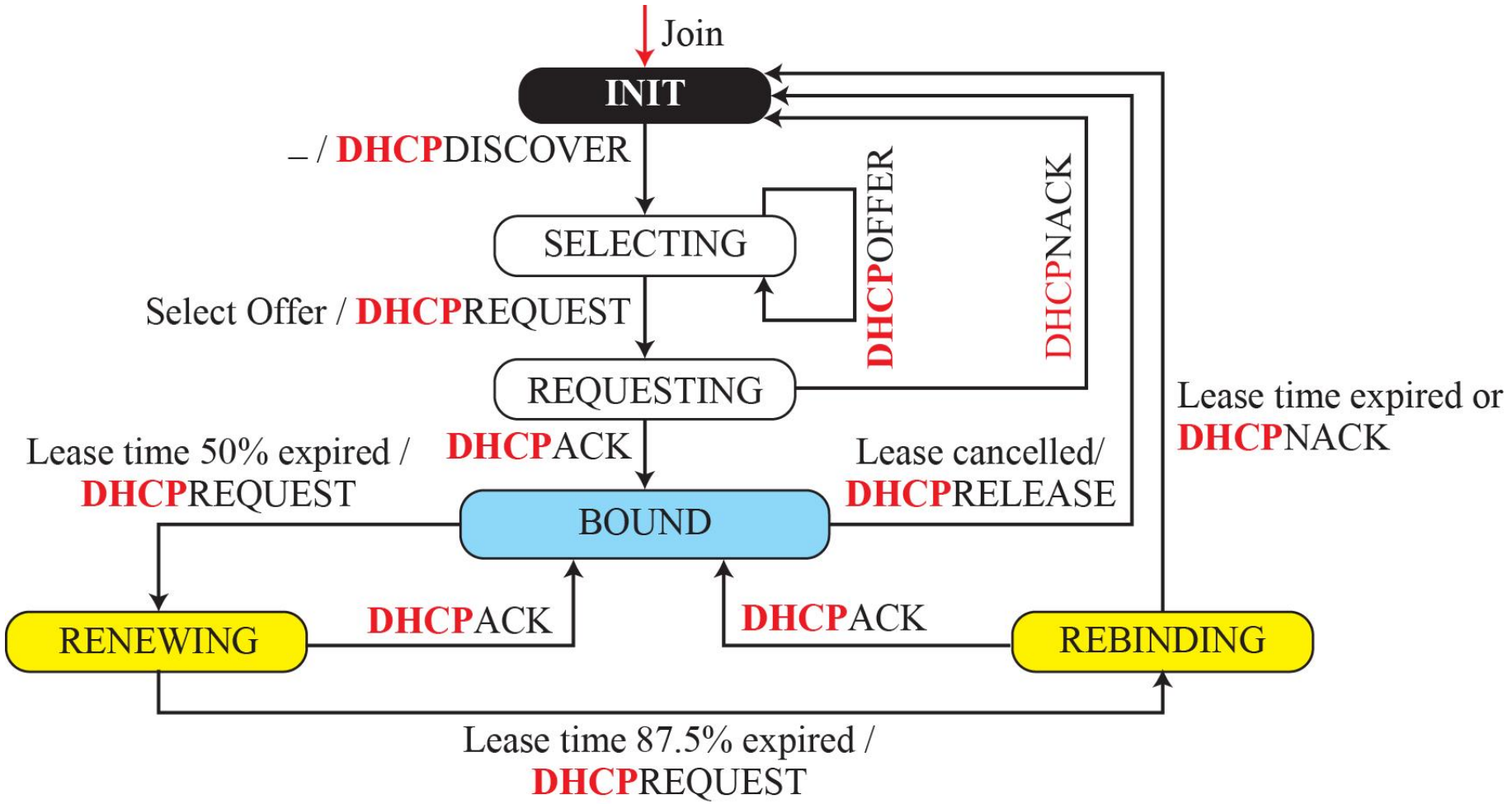
The server does not send all of the information that a client may need for joining the network. **In the DHCPACK message, the server defines the pathname of a file in which the client can find complete information such as the address of the DNS server.** The client can then **use a file transfer protocol to obtain the rest of the needed information**.

Error Control

DHCP uses the service of UDP, which is not reliable. To provide error control, **DHCP uses two strategies. First, DHCP requires that UDP uses the checksum**. As we saw in Chapter 3, the use of the checksum in UDP is optional. Second, **the DHCP client uses timers and a retransmission policy if it does not receive the DHCP reply to a request**. However, to prevent a traffic jam when several hosts need to retransmit a request (for example, after a power failure), **DHCP forces the client to use a random number to set its timers.**

Transition States

The previous scenarios we discussed for 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. Figure (next page) shows the transition diagram with the main states.


Join
INIT
– / DHCPDISCOVER
SELECTING
DHCPOFFER
DHCPNACK
Select Offer / DHCPREQUEST
REQUESTING
Lease time 50% expired / DHCPREQUEST
DHCPACK
Lease cancelled/ DHCPRELEASE
Lease time expired or DHCPNACK
BOUND
RENEWING
DHCPACK
DHCPACK
REBINDING
Lease time 87.5% expired / DHCPREQUEST

When the DHCP client first starts, it is in the INIT state (initializing state). The client broadcasts a discover message. When it receives an offer, the client goes to the SELECTING state. While it is there, it may receive more offers. After it selects an offer, it sends a request message and goes to the REQUESTING state. If an ACK arrives while the client is in this state, it goes to the BOUND state and uses the IP address. When the lease is 50 percent expired, the client tries to renew it by moving to the RENEWING state. If the server renews the lease, the client moves to the BOUND state again. If the lease is not renewed and the lease time is 75 percent expired, the client moves to the REBINDING state. If the server agrees with the lease (ACK message arrives), the client moves to the BOUND state and continues using the IP address; otherwise, the client moves to the INIT state and requests another IP address. Note that the client can use the IP address only when it is in the BOUND, RENEWING, or REBINDING state. The above procedure requires that the client uses three timers: renewal timer (set to 50 percent of the lease time), rebinding timer (set to 75 percent of the lease time), and expiration timer (set to the lease time)

NAT

Network Address Translation

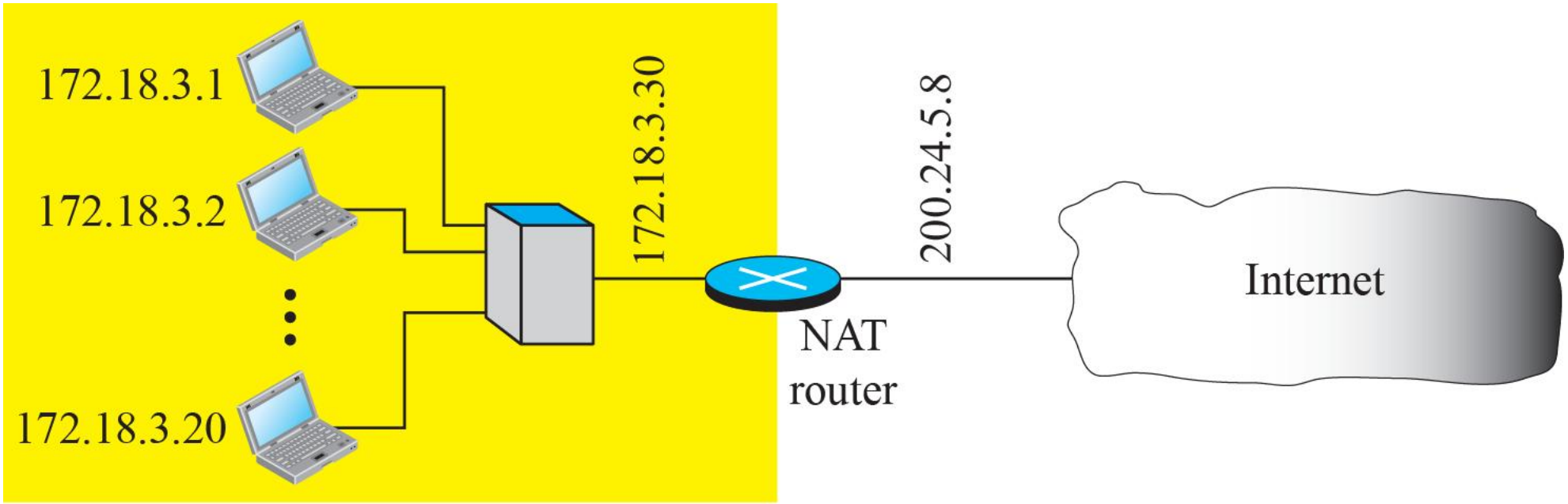
When Internet Service Providers (ISPs) hand out addresses, it can lead to a hitch. Picture this: an ISP gives a small business or a household a handful of addresses. Now, if the business grows or the home needs more addresses. The ISP might struggle to fulfill the request because addresses around the given range could already be taken by other networks.

But here's the thing: not all computers in a small network need internet access at the same time. Let's say a small business has 20 computers, but only 4 of them need internet access simultaneously. The rest might be busy with tasks that don't require internet access or are communicating internally. So, even if the business has 20 computers, it doesn't need 20 internet addresses.

So, what's the workaround? The business can use the TCP/IP protocol for both internal and external communication. They can reserve, let's say, 20 addresses from a private block for internal stuff and get 5 addresses from the ISP for internet communication.

Now, there's this cool technology called Network Address Translation (NAT). **It acts like a middleman, connecting private addresses used inside a place to global internet addresses.** Imagine it like this: a small business has its own set of **private addresses for internal communication, and then it has one or more global internet addresses for talking to the outside world.**

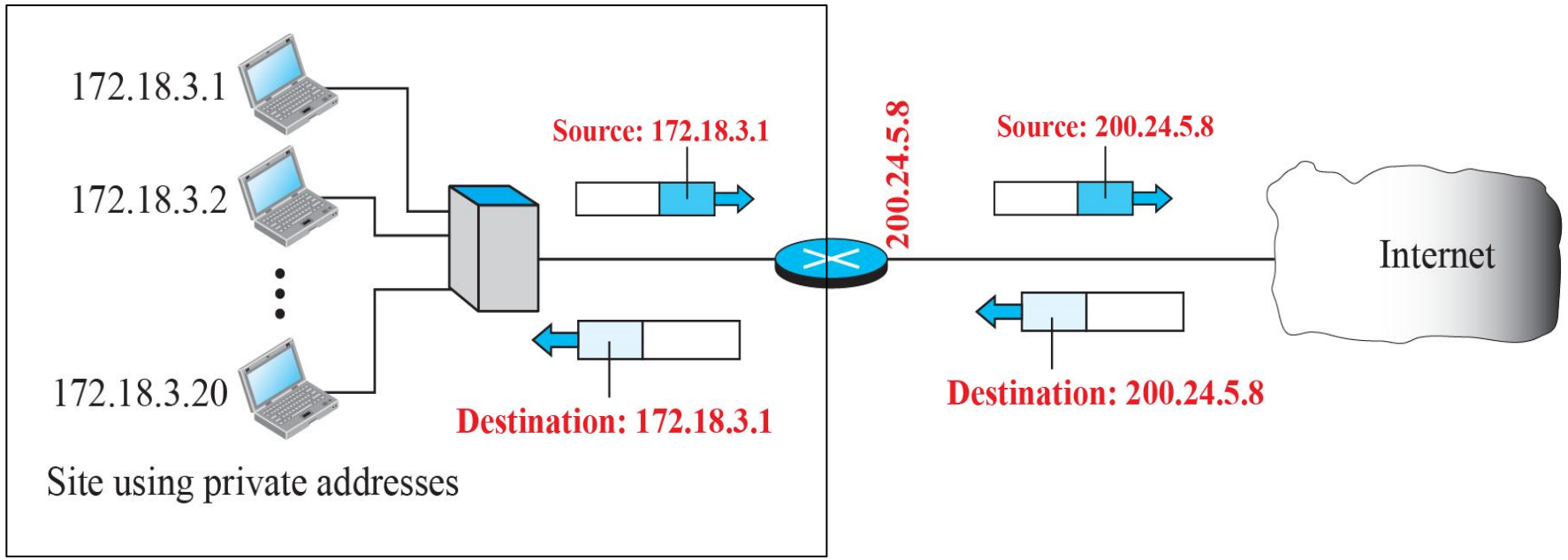
This is where the NAT router comes in. It's like a wizard that **translates between the private and global addresses**. So, **even though the private network is doing its thing with its private addresses, the rest of the internet sees only the NAT router, with its own global address.** It's like having a secret identity on the internet!



Site using private addresses

Address Translation

All of the outgoing packets go through the NAT router, which replaces the source address in the packet with the global NAT address. All incoming packets also pass through the NAT router, which replaces the destination address in the packet (the NAT router global address) **with the appropriate private address**. Figure shows an example of address translation.



Translation Table

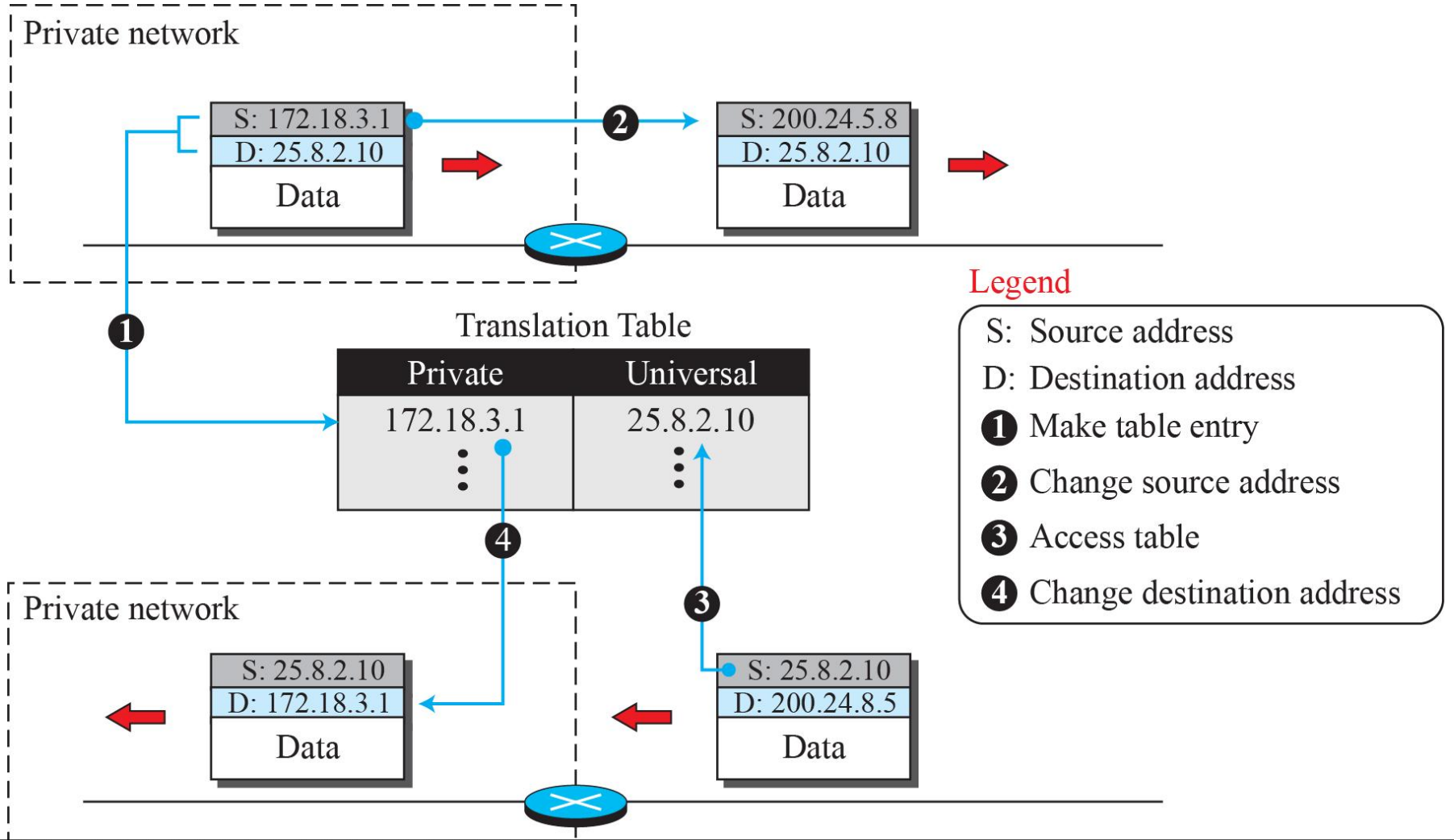
You may noticed that translating the source addresses for an outgoing packet is straightforward. But **how does the NAT router know the destination address for a packet coming from the Internet?** There may be tens or hundreds of private IP addresses, each belonging to one specific host. The problem is solved if the NAT router has a translation table.

Using One IP Address

In its simplest form, **a translation table is like a pair of columns: one for private addresses and the other for external addresses** (where the packet is headed). So, when the router changes the source address of an outgoing packet, it also takes note of where the packet is going—the destination address. When the response comes back, the router uses the packet's source address (now the external address) to find its private address. Think of it like keeping tabs on a conversation.

Now, here's the catch: in this setup, the private network always has to kick off the communication. It's like saying, "Hey, I want to talk!" The NAT mechanism we just talked about is usually used by ISPs that give one single address to a customer. But here's the twist: the customer might actually be part of a private network with lots of private addresses.

So, when this customer wants to chat with the internet, it's always the customer site that says, "Let's talk!" They use programs like HTTP, TELNET, or FTP to connect with corresponding servers. For instance, if an email from outside the network arrives at the ISP's email server, it's kept in the customer's mailbox until they fetch it using a protocol like POP. It's like the customer site is the one ringing the internet's doorbell!



Using a Pool of IP Addresses

When a NAT router uses just one global address, only one computer from the private network can talk to an external computer at a time. To fix this, the router can have a bunch of global addresses, like a pool of them—let's say four addresses (200.24.5.8, 200.24.5.9, 200.24.5.10, and 200.24.5.11). Now, four computers from the private network can chat with the same external computer simultaneously because each pair of addresses makes a separate connection.

But, there are catches. Only four connections can be made to the same place, and a private computer can't talk to two different external programs (like HTTP and TELNET) at once. Also, two private computers can't both talk to the same external program (like both doing HTTP or TELNET) at the same time. It's like having a few more phone lines, but there are still some limits on who can call who and when.

Using Both IP Addresses and Port Addresses

To let a bunch of computers in a private network talk to different external programs, we need more details in the translation table. Imagine two computers in a private network, with addresses 172.18.3.1 and 172.18.3.2, wanting to connect to the HTTP server on an external host, say 25.8.3.2.

Now, instead of just two columns in the table, we use five. These include info like the source and destination port addresses and the type of transport-layer protocol. This extra info removes any confusion. Check out Table(on next page) for an example. When the response from the HTTP server comes back, the combo of the source address (25.8.3.2) and destination port address (1401) tells us which computer in the private network should get the response. And, to make this work, the port addresses (1400 and 1401) have to be unique. It's like giving each conversation its own special code to keep things clear.

Private address	*Private port*	*External address*	*External port*	*Transport protocol*
172.18.3.1	1400	25.8.3.2	80	TCP
172.18.3.2	1401	25.8.3.2	80	TCP
⋮	⋮	⋮	⋮	⋮

ICMPv4

Internet Control Message Protocol version 4

I**Pv4 doesn't have a way to report or fix errors.** So, what if something goes wrong? Like when a router has to get **rid of data because it can't find a way to the destination, or if the time-to-live is zero?** What if the destination host has to toss out fragments of data because it didn't get them all in time? In these cases, there's an error, but **IPv4 can't tell the original host about it.**

Also, **IPv4 doesn't have a way for hosts to check if a router or another host is alive.** Sometimes, a network manager needs info from another host or router. That's where ICMPv4 comes in. It's designed to fix these issues. **ICMP is a friend to the IP protocol, but its messages don't go directly to the data-link layer. Instead, they're put inside IP datagrams before going to the lower layer.** When an IP datagram has an ICMP message, the protocol field is set to 1 to show it's an ICMP message.

Messages

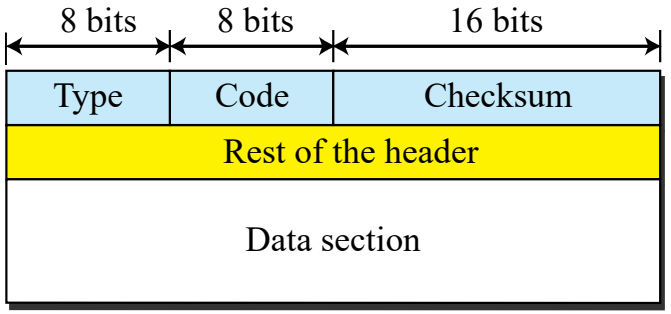
ICMPv4 messages are divided into two broad categories:

- error-reporting messages
- query messages

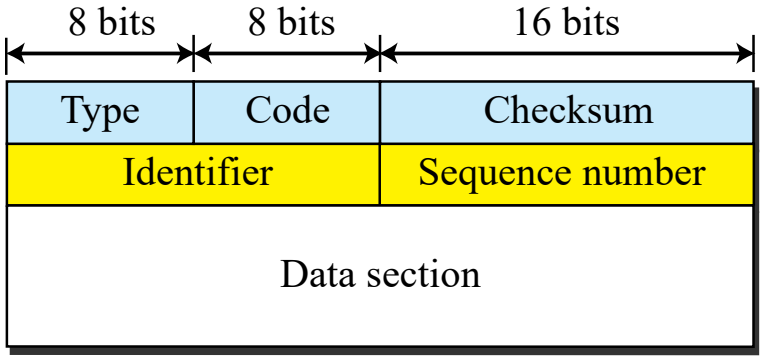
The error-reporting messages report problems that a router or a host (destination) may encounter when it processes an IP packet. The query messages, **which occur in pairs, help a host or a network manager get specific information from a router or another host.** For example, hosts can discover their neighbor hosts or routers on their network and routers can help a node redirect its messages.

Message Format

An ICMPv4 message has an 8-byte header and a variable-size data section. Although the general format of the **header is different for each message type**, the **first 4 bytes are common to all**. As Figure(next page) shows, the first field, **ICMP type, defines the type of the message**. The **code field specifies the reason for the particular message type**. The last
common field is the checksum field. The rest of the header is specific to each message type. **The data section in error messages carries information for finding the original packet that had the error**. In query messages, **the data section carries extra information based on the type of query.**



Error-reporting messages



Query messages

Type and code values

Error-reporting messages

03: Destination unreachable (codes 0 to 15)
04: Source quench (only code 0)
05: Redirection (codes 0 to 3)
11: Time exceeded (codes 0 and 1)
12: Parameter problem (codes 0 and 1)

Query messages

08 and 00: Echo request and reply (only code 0)
13 and 14: Timestamp request and reply (only code 0)

Note: See the book website for more explanation about the code values.

Error Reporting Messages

ICMP is like the messenger of the Internet, especially when things go wrong with the IP protocol. Since IP isn't perfect, ICMP steps in to report errors but doesn't fix them—higher-level protocols handle that. When there's an error, ICMP shoots a message back to the original sender because the only route info in the datagram is the source and destination IP addresses.

ICMP follows some rules for simplicity. No error messages pop up for multicast or special addresses, and if a datagram carries an ICMP error, no additional error message is created. Also, **fragmented datagrams that aren't the first fragment won't trigger an ICMP error message.**

All error messages have a data section, including the IP header of the original datagram and the first 8 bytes of data. This helps the original sender understand what went wrong. The 8 bytes include info about port numbers and sequence numbers, crucial for informing TCP or UDP about the error.

The most common error message is **"destination unreachable" (type 3)**. It uses different codes to say why the datagram didn't reach its final stop. For instance, code 0 signals that a host is unreachable, like when the server is down while using the HTTP protocol. The message "destination host is not reachable" goes back to the sender.

Another error message, **"source quench" (type 4)**, **tells the sender that the network is too busy**, and it dropped the datagram. The sender needs to chill and not send more datagrams for a bit. So, ICMP adds a sort of traffic control to the IP protocol with this message.

The **"redirection" message (type 5) comes into play when the source sends its message through the wrong router.** The router redirects the message to the right router but tells the source to use a different default router next time. It even includes the IP address of the suggested router.

Remember **the time-to-live (TTL)** field in the IP datagram? It prevents datagrams from endlessly circling the Internet. When the TTL hits 0, the router drops the datagram and sends a "time exceeded" message (type 11, code 0) to the sender. This says, "Hey, your datagram ran out of time." It can also happen when not all fragments of a datagram arrive in time.

Lastly, there's a **"parameter problem" message (type 12)** for **when there's an issue in the datagram header (code 0) or some options are missing or can't be understood (code 1)**. It's like saying, "Hey, your datagram has a hiccup."

Query Messages

It's cool that ICMP's query messages can work on their own without tying to an IP datagram. These queries, wrapped up in a datagram, help check if hosts or routers in the internet are alive, measure the time it takes for an IP datagram to travel between devices, or see if the clocks in two devices are in sync. These queries come in pairs: a request and a reply.

Take the **echo request (type 8) and the echo reply (type 0) messages—they're like a conversation starter. One device asks another if it's alive (echo request)**, and if it is, it responds with an "I'm here" message (echo reply). We'll see how handy this pair is in tools like ping and traceroute.

Then there's the **timestamp request (type 13) and the timestamp reply (type 14) messages. They help figure out how long it takes for data to go back and forth between devices or check if their clocks match.** The timestamp request sends a time-stamp, and the reply sends it back, plus when it got the request and when it sent the reply. If all these times are in Universal time, the sender can calculate how long it took for the data to travel one way and round-trip.

Example

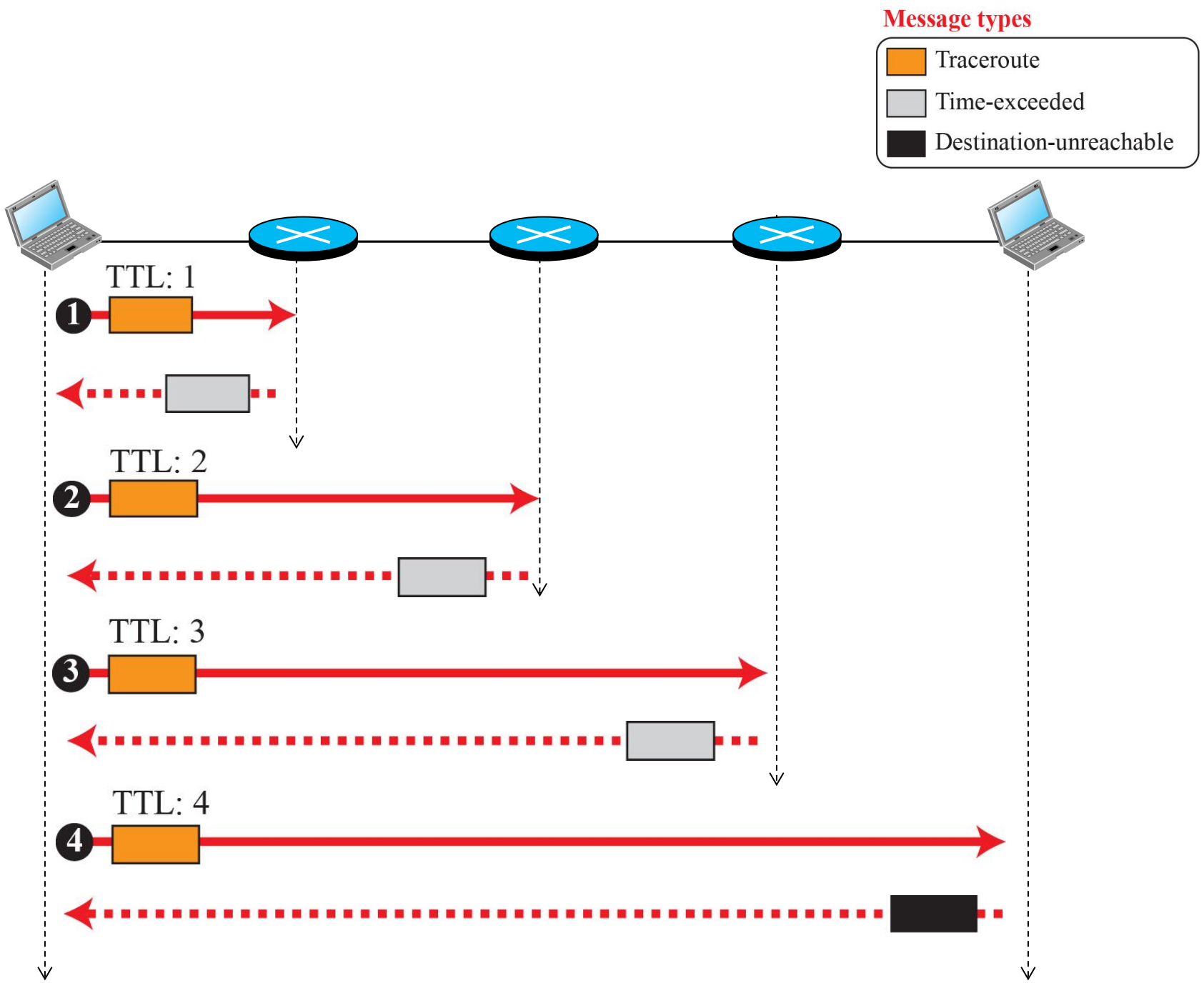
One of the tools that a host can use to test the liveliness of another host is the ping program. The ping program takes advantage of the ICMP echo request and echo reply messages. A host can send an echo request (type 8, code 0) message to another host, which, if alive, can send back an echo reply (type 0, code 0) message. To some extent, the ping program can also measure the reliability and congestion of the router between the two hosts by sending a set of request-reply messages.

```
$ ping auniversity.edu
PING auniversity.edu (152.181.8.3)   56 (84)  bytes of data.
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=0     ttl=62     time=1.91 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=1     ttl=62     time=2.04 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=2     ttl=62     time=1.90 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=3     ttl=62     time=1.97 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=4     ttl=62     time=1.93 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=5     ttl=62     time=2.00 ms
--- auniversity.edu statistics ---
6 packets transmitted, 6 received, 0% packet loss
rtt min/avg/max = 1.90/1.95/2.04 ms
```

Example

The traceroute program in UNIX or tracert in Windows can be used to trace the path of a packet from a source to the destination. It can find the IP addresses of all the routers that are visited along the path. The program is usually set to check for the maximum of 30 hops (routers) to be visited. The number of hops in the Internet is normally less than this. The traceroute program is different from the ping program. The ping program gets help from two query messages; the traceroute program gets help from two error-reporting messages: time-exceeded and destination-unreachable

Figure (next page) shows an example in which n = 3.


Message types
Traceroute
Time-exceeded
Destination-unreachable
TTL: 1
1
TTL: 2
2
TTL: 3
3
TTL: 4
4

The traceroute program also sets a timer to find the round-trip time for each router and the destination. Most traceroute programs send three messages to each device, with the same TTL value, to be able to find a better estimate for the round-trip time. The following shows an example of a traceroute program, which uses three probes for each device and gets three RTTs.

```
$ traceroute printers.com
traceroute to printers.com (13.1.69.93), 30 hops max, 38 byte packets
1  route.front.edu      (153.18.31.254)   0.622 ms   0.891 ms   0.875 ms
2  ceneric.net          (137.164.32.140)  3.069 ms   2.875 ms   2.930 ms
3  satire.net           (132.16.132.20)   3.071 ms   2.876 ms   2.929 ms
4  alpha.printers.com   (13.1.69.93)      5.922 ms   5.048 ms   4.922 ms
```

UNICAST ROUTING

Imagine the internet as a giant delivery system. The network layer's job is to get information (let's call it a "datagram") from **where it starts to where it needs to go. If it's going to one place, it's like a direct delivery (unicast). If it's going to several places, it's multicast. And if it's going to everyone, it's broadcast.**

Now, let's focus on the one-to-one delivery (unicast) in this chat. With so many routers and tons of devices on the internet, it's like organizing a huge delivery network. We can't just send things in one go. We need a plan, right? So, we use hierarchical routing—like a step-by-step guide. First, we understand the basics of getting stuff from A to B using routers. Once we've got that down, we can talk about the big picture of how this works on the entire internet using hierarchical routing. Cool, huh?

General Idea

Think of unicast routing like sending a message from one friend to another through a bunch of postal relay stations.

So, your message (packet) goes from your friend (source host) to the first relay station (default router in your local network). Then, **it hops from station to station until it reaches your other friend (destination host)**, again through their default router.

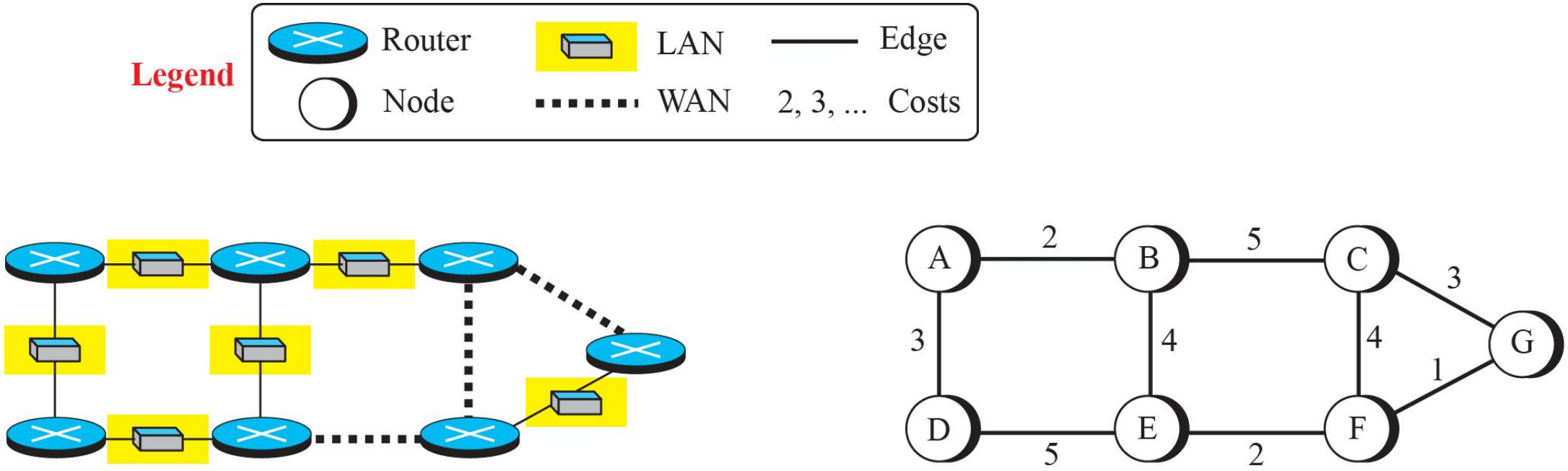
Now, only **these relay stations (routers connecting networks in the internet) need a list (forwarding table) to know where to send things.** Your friends don't need a list because they just hand stuff to their nearest relay station. So, the real question is, out of all the possible routes, which relay stations should your message visit to get from your friend to the other? **It's like finding the best way for your message to travel!**

An Internet as a Graph

Imagine the internet as a big map with cities (routers) and roads (networks between routers). We're trying to find the best way to travel through this map. So, we turn it into a graph—a bunch of points (nodes) connected by lines (edges).

Each city (router) is a point, and the roads (networks) between them are the lines. Now, this map is special; it's like a game where each road has a cost. Maybe it's the time it takes or the distance. If there's no direct road between cities, we say the cost is infinity.

It's kind of like planning a trip: we want the route with the lowest total cost. This helps us figure out the best path for our internet data. Later on, we'll dive into the different meanings of these costs in routing, but for now, just imagine finding the best, most efficient way on this internet map. Cool, right?



a. An internet

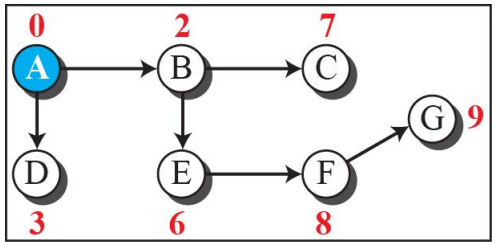
b. The weighted graph

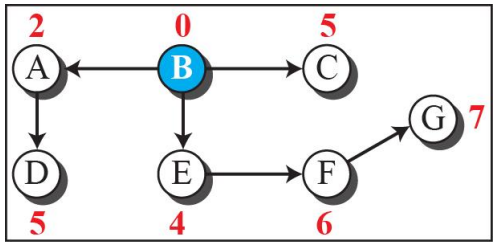
Least-Cost Routing

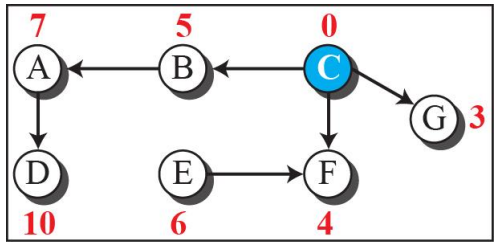
When an internet is modeled as a weighted graph, **one of the ways to interpret the best route from the source router to the destination router is to find the least cost between the two**. In other words, the source router chooses a route to the destination router in such a way that the total cost for the route is the least cost among all possible routes. In Figure (pre. page), **the best route between A and E is A-B-E, with the cost of 6.** This means that each router needs to find the least-cost route between itself and the all other routers to be able to route a packet using this criteria.

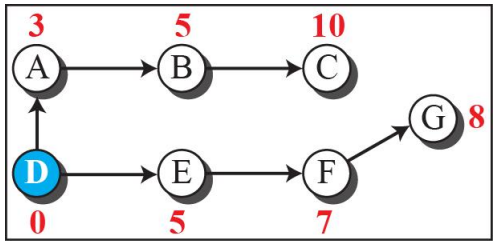
Least-Cost Trees

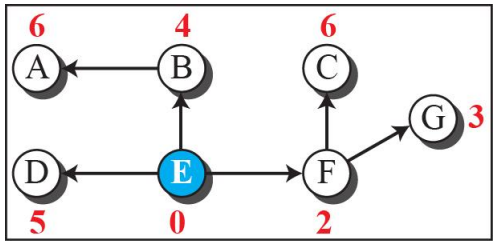
If there are **N routers in an internet, there are (N – 1) least-cost paths from each router to any other router.** This **means we need N × (N – 1) least-cost paths for the whole internet.** If we have only 10 routers in an internet, we need 90 least-cost paths. A better way to see all of these paths is to combine them in a least-cost tree. **A least-cost tree is a tree with the source router as the root that spans the whole graph (visits all other nodes) and in which the path between the root and any other node is the shortest.** In this way, we can have only one shortest-path tree for each node; we have N least-cost path trees for the whole internet. We show how to create a least-cost tree for each node later in this section; for the moment, Figure 4.57 shows the seven least-cost trees for the internet in Figure 4.56.

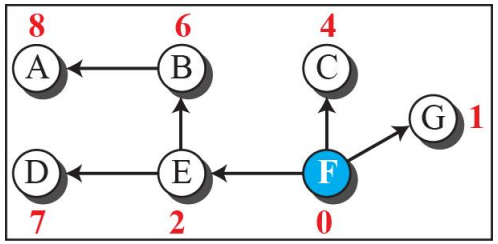






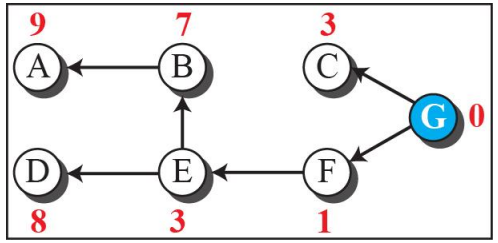






Legend

	Root of the tree
	Intermediate or end node
1, 2, ...	Total cost from the root



The least-cost trees for a weighted graph can have several properties if they are created using consistent criteria.

1. The least-cost route from X to Y in X's tree is the inverse of the least cost route from Y to X in Y's tree; the cost in both directions is the same. For example, in Figure 4.57, the route from A to F in A's tree is (A → B → E → F), but the route from F to A in F's tree is (F → E → B → A), which is the inverse of the first route. The cost is 8 in each case.
2. Instead of travelling from X to Z using X's tree, we can travel from X to Y using X's tree and continue from Y to Z using Y's tree. For example, in Figure 4.57, we can go from A to G in A's tree using the route (A → B → E → F → G). We can also go from A to E in A's tree (A → B → E) and then continue in E's tree using the route (E → F → G). The combination of the two routes in the second case is the same route as in the first case. The cost in the first case is 9; the cost in the second case is also 9 (6 + 3).

Routing Algorithms

Several routing algorithms have been designed in the past. The differences between these methods are in the way they interpret the least cost and the way they create the least-cost tree for each node. In this section, we discuss the common algorithm; later we show how a routing protocol in the Internet implements one of these algorithms.

Distance-Vector Routing

The distance-vector (DV) routing uses the goal we discussed in the introduction to find the best route. In distance-vector routing, **the first thing each node creates is its own least-cost tree** with **the rudimentary information it has about its immediate neighbors.** The incomplete trees are exchanged between immediate neighbors to make the trees more and more complete and to represent the whole internet. We can say that in **distance-vector routing, a router continuously tells all of its neighbors about what it knows about the whole internet** (although the knowledge can be incomplete).

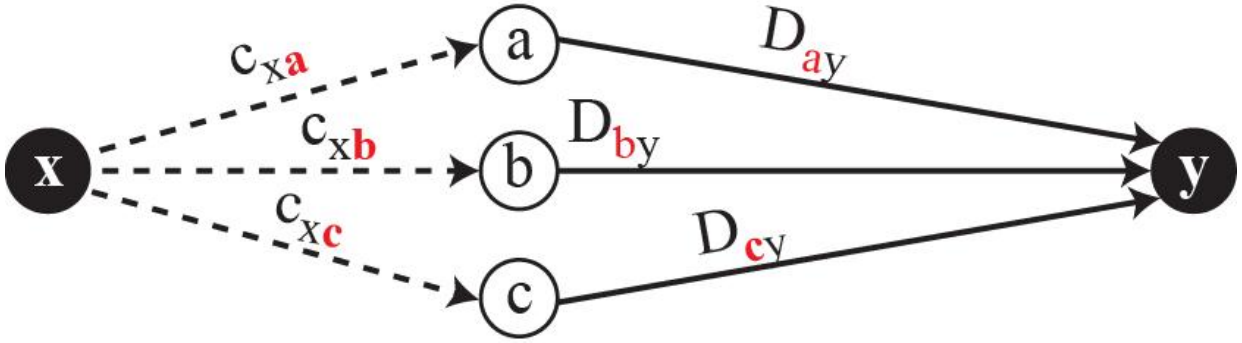
Bellman-Ford Equation

This equation
is used to find the **least cost (shortest distance) between a source node, x, and a destination node, y, through some intermediary nodes (a, b, c, . . .)** when the costs between the source and the intermediary nodes and the least costs between the intermediary nodes and the destination are given. The following shows the general case in which D_(ij) is the shortest distance and c_(ij) is the cost between node i and j.

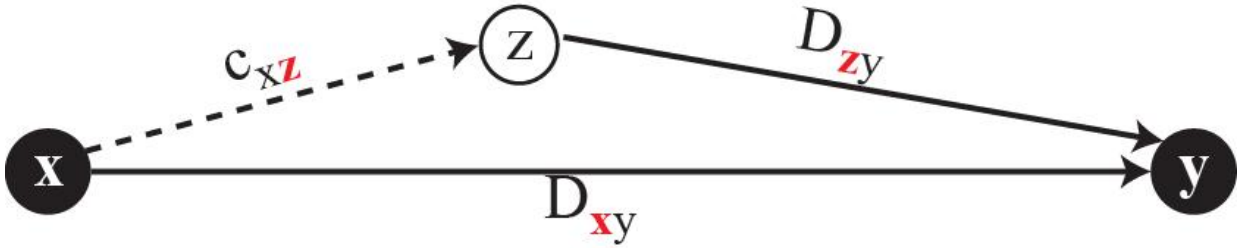
$$D_{xy} = \min\{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \ldots\}$$

In distance-vector routing, normally we want to update an existing least cost with a least cost through an intermediary node, such as z, if the latter is shorter. In this case, the equation becomes simpler, as shown below:

$$D_{xy} = \min\{D_{xy}, (c_{xz} + D_{zy})\}$$



a. General case with three intermediate nodes



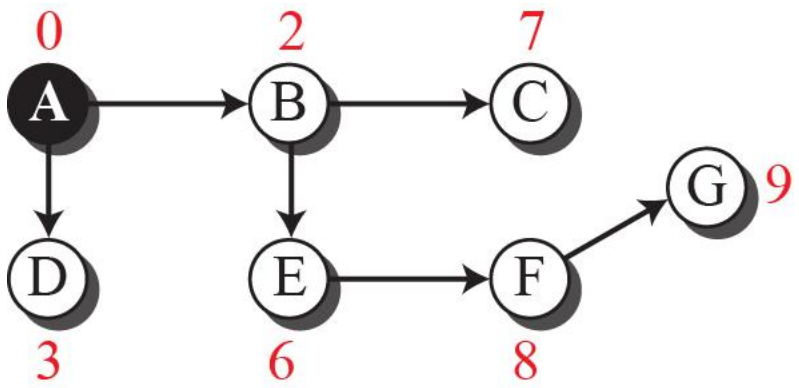
b. Updating a path with a new route

we can think of (a→y),(b→y), and (c→y) as previously-established least-cost paths and (x→y) as the new least-cost path. We can even think of this equation as the builder of a new least-cost tree from previously established least-cost trees if we use the equation repeatedly

Distance Vectors

Distance-vector routing uses a concept called a distance vector, which is like a simplified map showing the shortest paths from one point (the root) to all destinations. **Imagine it as a one-dimensional list where each entry represents the cost from the root to a destination.** Nodes in a network create this vector by exchanging basic information about their neighbors and distances.

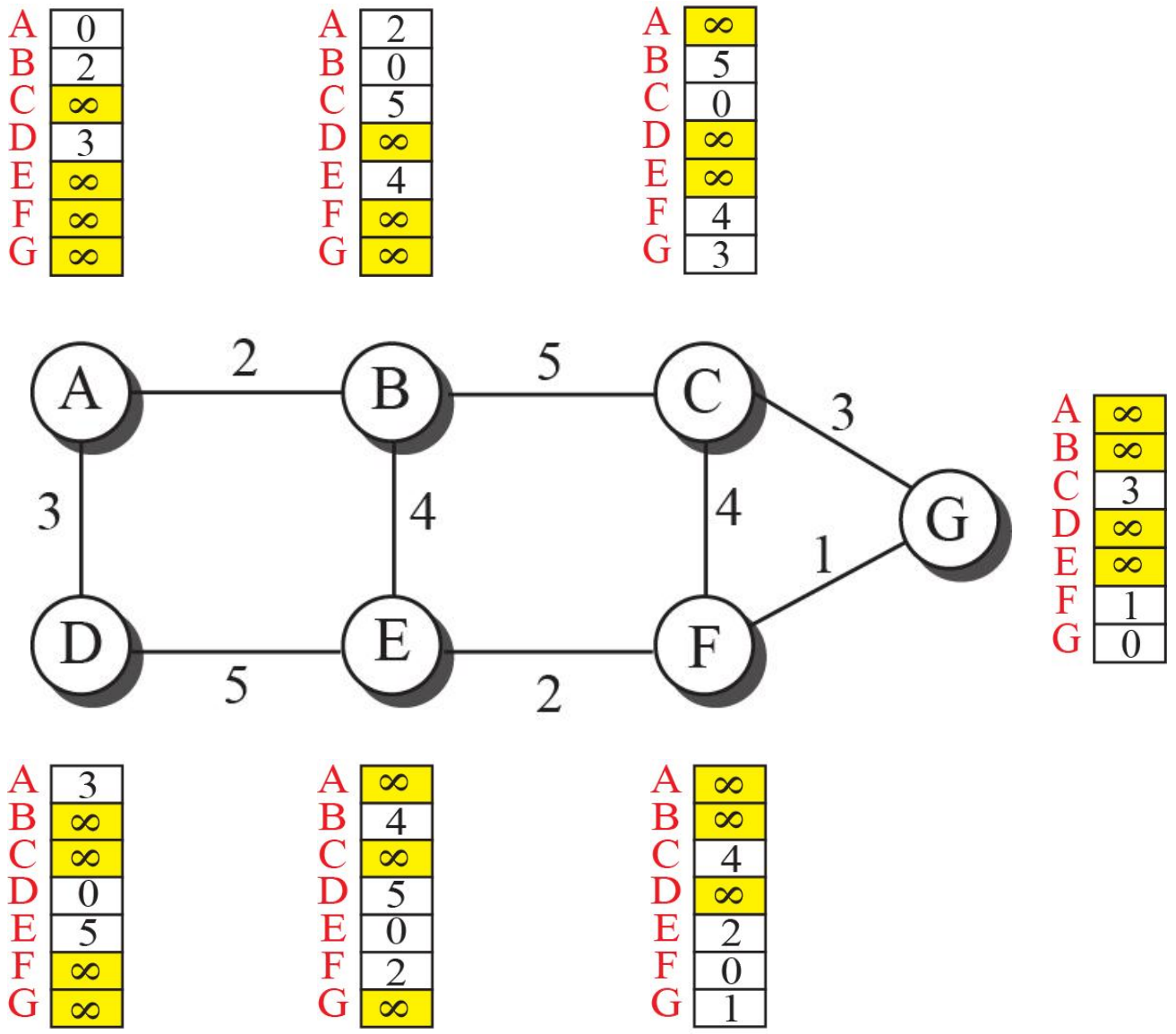
Unlike a detailed map (least-cost tree), **the vector only shows costs, not the actual paths. Each node starts with a basic vector, gradually improving it by learning distances to neighbors.** This process happens independently for each node, not all at once. The goal is to eventually have accurate distance vectors for the entire network, which can then be used to find the most efficient paths.



a. Tree for node A

	A
A	0
B	2
C	7
D	3
E	6
F	8
G	9

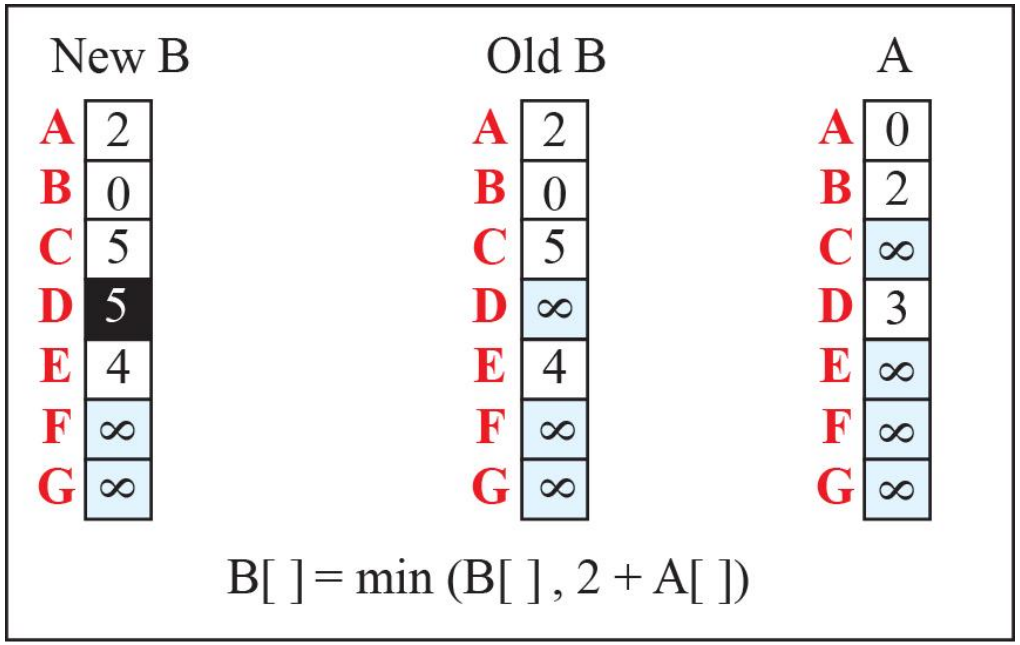
b. Distance vector for node A


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

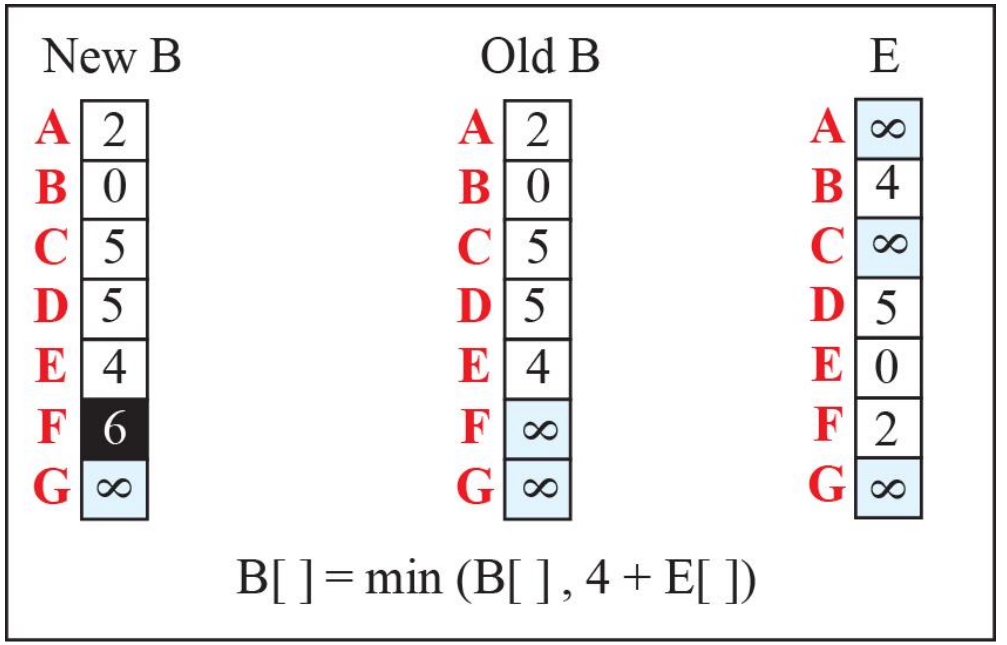
To make the internet smart in forwarding packets, nodes share their basic distance vectors with neighbors. Initially, these vectors aren't very helpful. For instance, node A might think it's not connected to node G because the cost is marked as infinity.

To improve, nodes exchange their vectors and update them using the Bellman-Ford equation. This equation considers all possible paths (N of them, where N is the number of nodes). This updating process happens in a loop or visually as a complete vector, not as separate equations for each node.

Imagine it like sharing notes: Node A tells B about its vector, and B updates its own. This process repeats as nodes exchange information. The goal is to stabilize the system, allowing all nodes to figure out the best routes. Even if a node has received an update, getting a newer one can still be beneficial.



a. First event: B receives a copy of A's vector.



b. Second event: B receives a copy of E's vector.

Note:
X[ ]: the whole vector

Distance-Vector Routing Algorithm

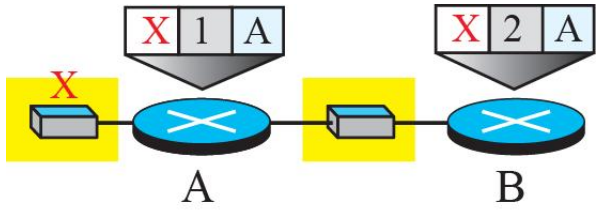
```
1   Distance_Vector_Routing ( )
2   {
3       // Initialize (create initial vectors for the node)
4       D[myself] = 0
5       for (y = 1 to N)
6       {
7           if (y is a neighbor)
8               D[y] = c[myself][y]
9           else
10              D[y] = ∞
11      }
12      send vector {D[1], D[2], …, D[N]} to all neighbors
13      // Update (improve the vector with the vector received from a neighbor)
14      repeat (forever)
15      {
16          wait (for a vector Dw from a neighbor w or any change in the link)
17          for (y = 1 to N)
18          {
19              D[y] = min [D[y], (c[myself][w] + Dw[y])]        // Bellman-Ford equation
20          }
21          if (any change in the vector)
22              send vector {D[1], D[2], …, D[N]} to all neighbors
23      }
24  }  // End of Distance Vector
```

Count to Infinity

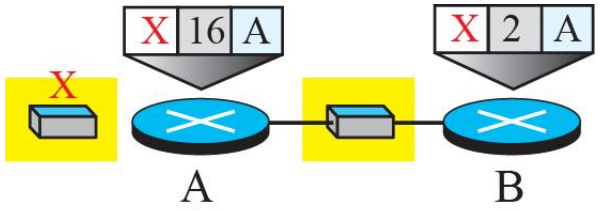
A problem with distance-vector routing is that any decrease in cost (good news) propagates quickly, but any increase in cost (bad news) will propagate slowly. **For a routing protocol to work properly, if a link is broken (cost becomes infinity)**, every other router should be aware of it immediately, 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.

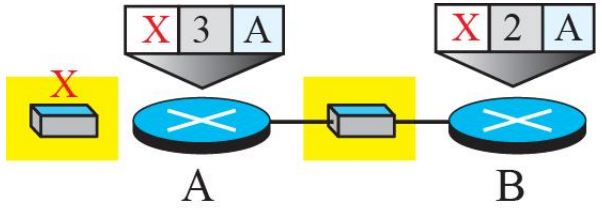
Two-Node Loop



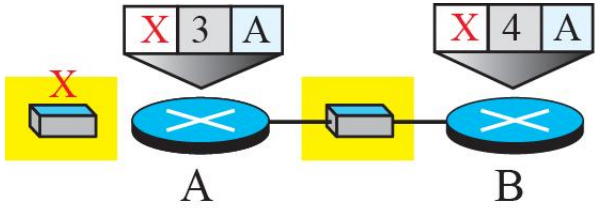
a. Before failure



b. After link failure

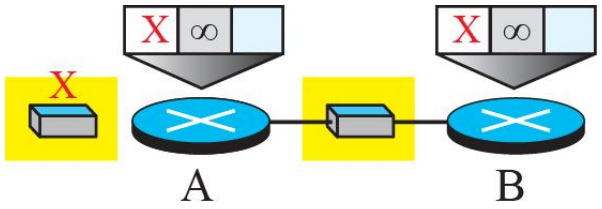


c. After A is updated by B



d. After B is updated by A

• • •



e. Finally

Imagine a system with nodes A, B, and X. At first, A and B know how to reach X. But if the link between A and X fails, trouble arises. A updates its table, but if B sends its update to A first, things get messy. A receives B's update, assumes B found a way to X, and updates its table. A then sends the update to B. Now, B thinks something changed around A and updates its table, gradually increasing the cost to X until it's infinite.

During this unstable time, A thinks the route to X is via B, and B thinks it's via A. If a packet for X arrives at A, it goes to B and comes back, creating a loop. The same happens if the packet arrives at B. This bouncing back and forth causes a two-node loop problem. People have suggested solutions to fix this kind of instability.

Split Horizon

One solution to instability is called split horizon. In this strategy,
instead of flooding the table through each interface, each node sends only part of its table through each interface. If, according to its table, node B thinks that the optimum route to reach X is via A, it does not need to advertise this piece of information to A; the information has come from A (A already knows). Taking information from node A, modifying it, and sending it back to node A is what creates the confusion. In our scenario, node B eliminates the last line of its forwarding table before it sends it to A. In this case, node A keeps the value of infinity as the distance to X. Later, when node A sends its forwarding table to B, node B also corrects its forwarding table. The system becomes stable after the first update: both node A and B know that X is not reachable.

Poisoned Reverse

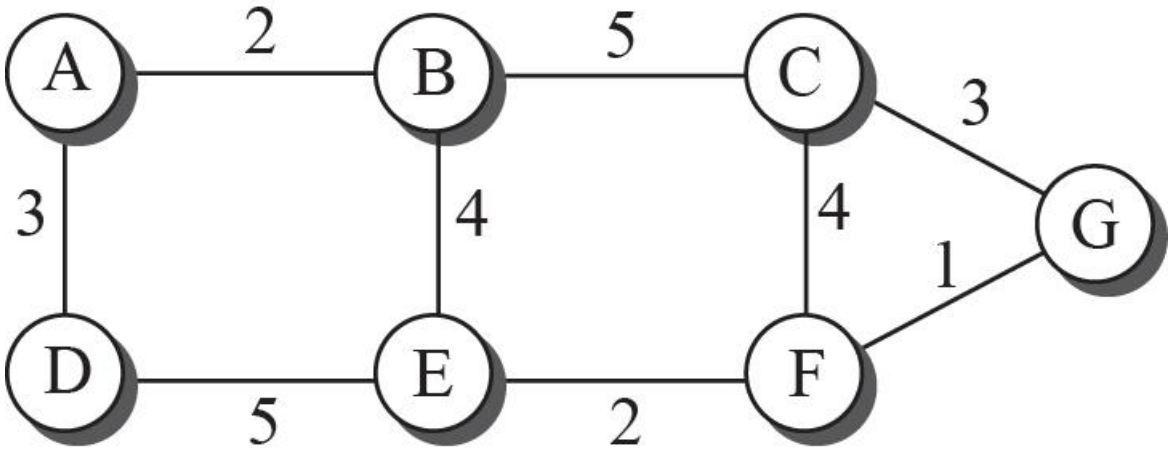
Using the split-horizon strategy has one drawback. Normally, the corresponding protocol uses a timer, and if there is no news about a route, the node deletes the route from its table. **When node B in the previous scenario eliminates the route to X from its advertisement to A, node A cannot guess whether this is due to the split-horizon strategy (the source of information was A) or because B has not received any news about X recently.** In the poisoned reverse strategy B can still advertise the value for X, but if the source of information is A, it can replace the distance with infinity as a warning: “Do not use this value; what I know about this route comes from you.”

Three-Node Instability

The two-node instability can be avoided using split horizon combined with poisoned reverse. However, if the instability is between three nodes, stability cannot be guaranteed.

Link-State Routing

A routing algorithm that directly follows our discussion for creating least-cost trees and forwarding tables is link-state (LS) routing. This method uses the term link-state to define the characteristic of a link (an edge) that represents a network in the internet. In this algorithm the cost associated with an edge defines the state of the link. Links with lower costs are preferred to links with higher costs; if the cost of a link is infinity, it means that the link does not exist or has been broken.



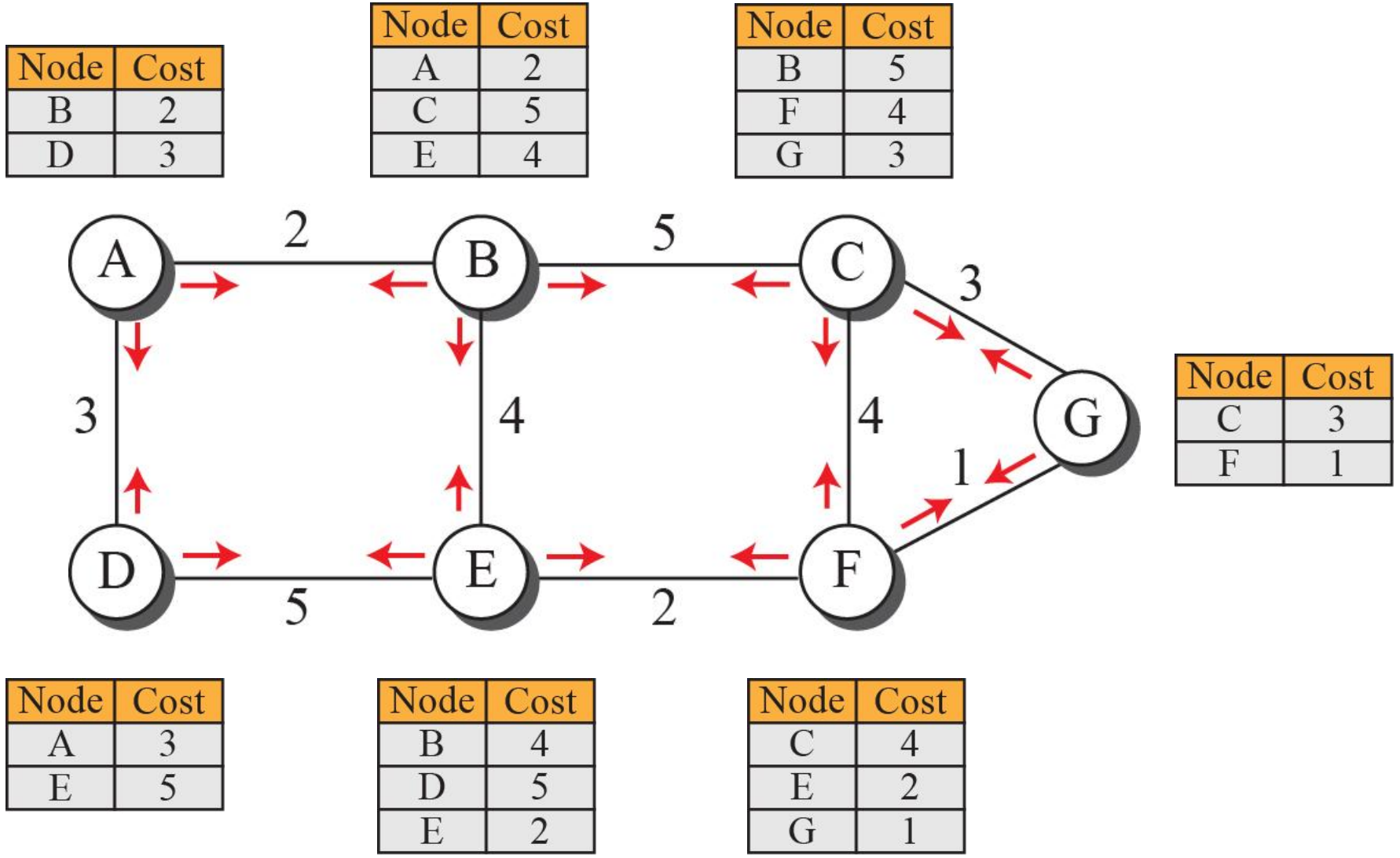
a. The weighted graph

	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

Link-State Database (LSDB)

To make the cheapest route map, each point in the network needs a full map of it. This map, called the link-state database (LSDB), has all the info about every connection's status. To create this, each point shares greetings with its nearby points, learning who they are and the link costs. This info, bundled as LS packets (LSP), is sent to all connected points. When a point gets an LSP, it checks if it's newer and keeps the latest. This process, called flooding, ensures all points eventually have the same LSDB, creating a full map of the network for everyone.



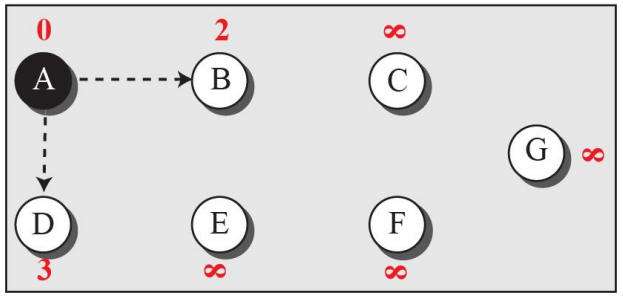
We can compare the link-state routing algorithm with the distance-vector routing algorithm. In the distance-vector routing algorithm, each router tells its neighbors what it knows about the whole internet; in the link-state routing algorithm, each router tells the whole internet what it knows about its neighbors.

Formation of Least-Cost Trees

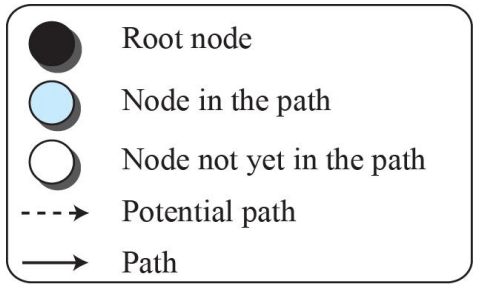
To create a least-cost tree for itself, using the shared LSDB, each node needs to run the famous Dijkstra Algorithm. This iterative algorithm uses the following steps:

1. The node chooses itself as the root of the tree, creating a tree with a single node,and sets the total cost of each node based on the information in the LSDB.
2. The node selects one node, among all nodes not in the tree, which is closest to the root, and adds this to the tree. After this node is added to the tree, the cost of all other nodes not in the tree needs to be updated because the paths may have been changed.
3. The node repeats step 2 until all nodes are added to the tree.

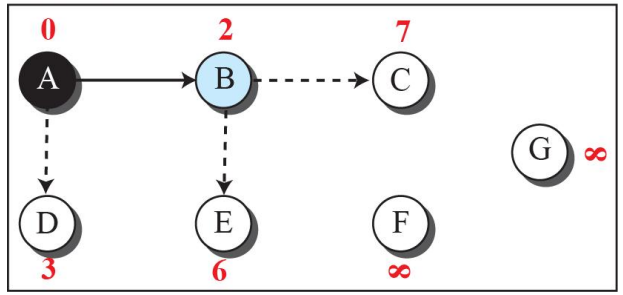
Initialization



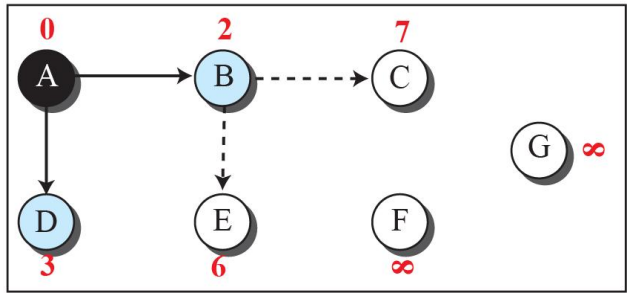
Legend



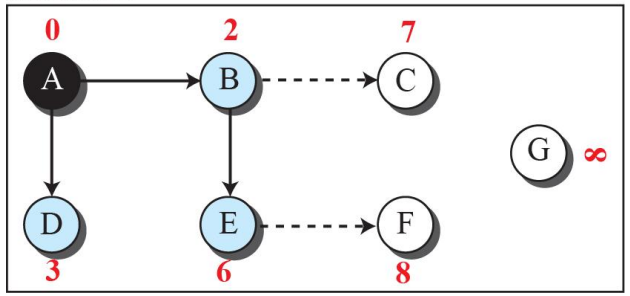
Iteration 1



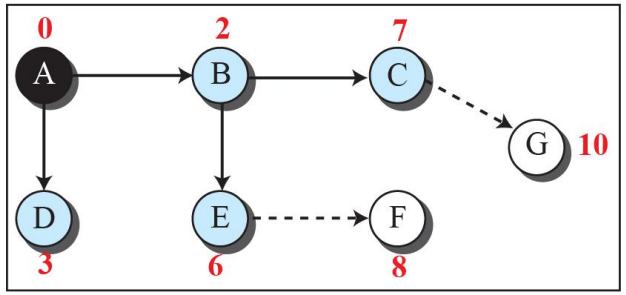
Iteration 2



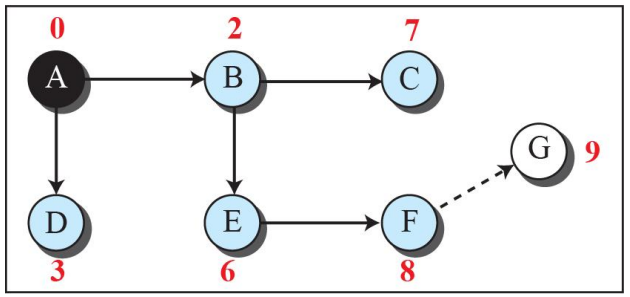
Iteration 3



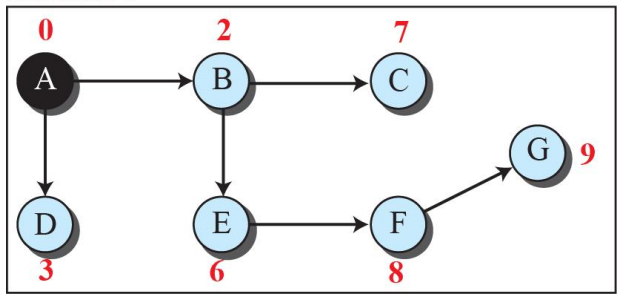
Iteration 4



Iteration 5



Iteration 6



```
1   Dijkstra's Algorithm ( )
2   {
3       // Initialization
4       Tree = {root}                    // Tree is made only of the root
5       for (y = 1 to N)                 // N is the number of nodes
6       {
7           if (y is the root)
8               D[y] = 0                 // D[y] is shortest distance from root to node y
9           else if (y is a neighbor)
10              D[y] = c[root][y]        // c[x][y] is cost between nodes x and y in LSDB
11          else
12              D[y] = ∞
13      }
14      // Calculation
15      repeat
16      {
17          find a node w, with D[w] minimum among all nodes not in the Tree
18          Tree = Tree ∪ {w}            // Add w to tree
19          // Update distances for all neighbor of w
20          for (every node x, which is neighbor of w and not in the Tree)
21          {
22              D[x] = min{D[x], (D[w] + c[w][x])}
23          }
24      } until (all nodes included in the Tree)
25  } // End of Dijkstra
```

Lines 4 to 13 implement step 1 in the algorithm. Lines 16 to 23 implement step 2 in the algorithm. Step 2 is repeated until all nodes are added to the tree.

Path-Vector Routing

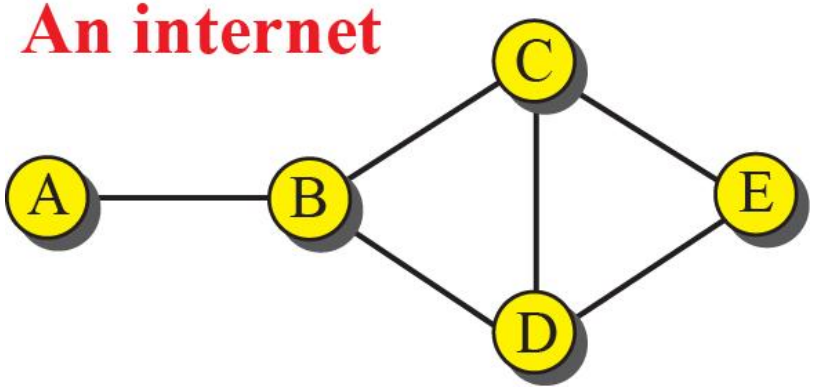
There are two common ways to decide the best route for data on the internet: link-state (LS) and distance-vector (DV) routing. They both focus on finding the cheapest path. But sometimes, you might not want the cheapest path. **For instance, you might want to avoid routers in a specific organization or a competitor's network.**

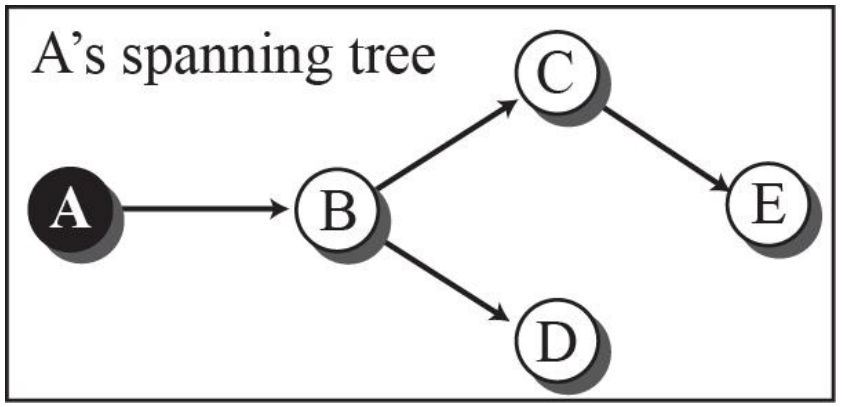
Here's where a third method, called path-vector (PV) routing, comes in. Unlike LS and DV, PV doesn't insist on the cheapest path. Instead, **it lets the sender decide based on its own rules or policies.** This means you can control the route your data takes. PV is not widely used on the internet, but it's designed to let big networks, like those of internet service providers (ISPs), choose their own routes based on their needs.

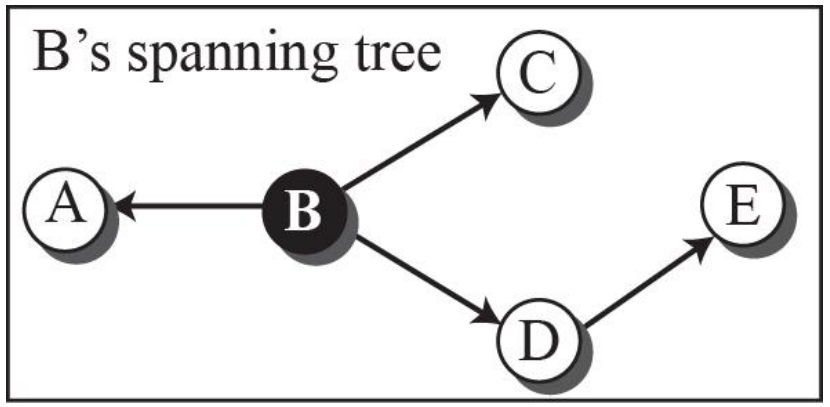
Spanning Trees

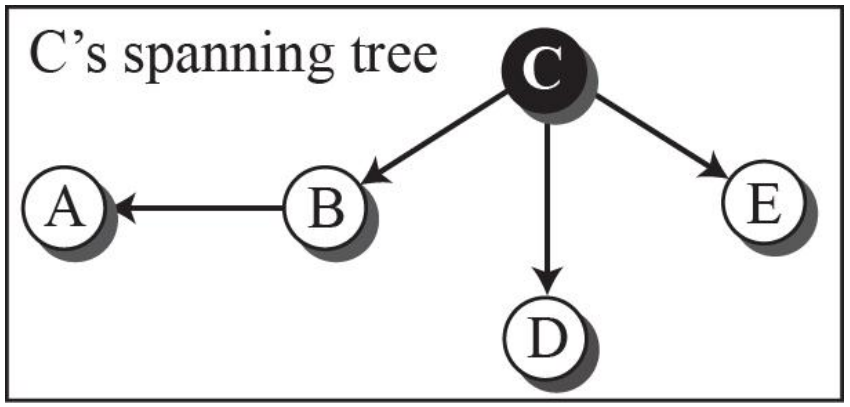
In path-vector routing, the route from a starting point to all destinations is decided by the best spanning tree. Unlike the least-cost tree, the best spanning tree is determined by the starting point based on its own rules. If there are multiple routes to a destination, the starting point can choose the one that fits its rules best. **These rules can include visiting the fewest nodes or avoiding specific nodes in the middle of the route.**

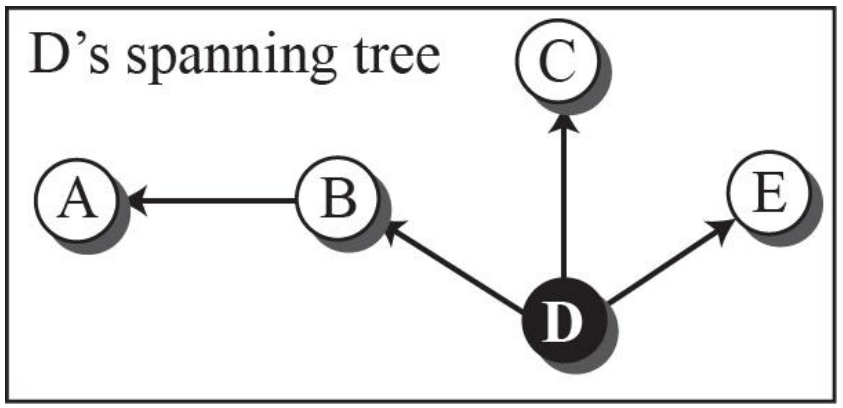
For example, imagine a small network with five nodes shown in Figure (next page). Each starting point creates its own spanning tree based on its rules. Some rules might aim for the shortest route (similar to least-cost), while others might avoid certain nodes in the middle of the route. So, each starting point customizes its route to follow its specific policies.

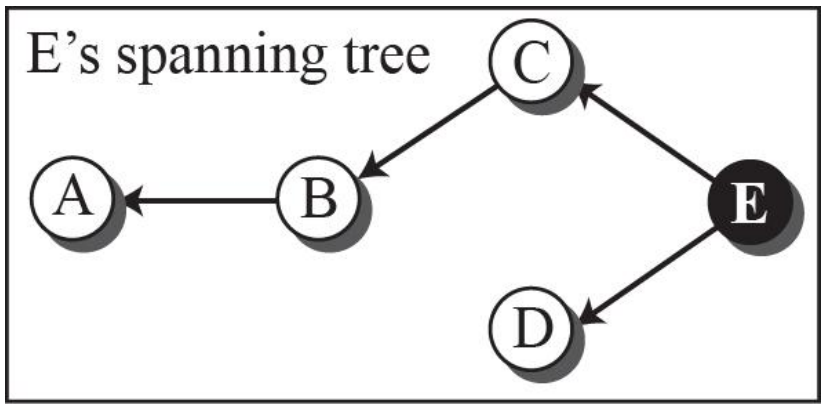

An internet
A
B
C
D
E


A's spanning tree
A
B
C
D
E


B's spanning tree
A
B
C
D
E


C's spanning tree
A
B
C
D
E


D's spanning tree
A
B
C
D
E


E's spanning tree
A
B
C
D
E

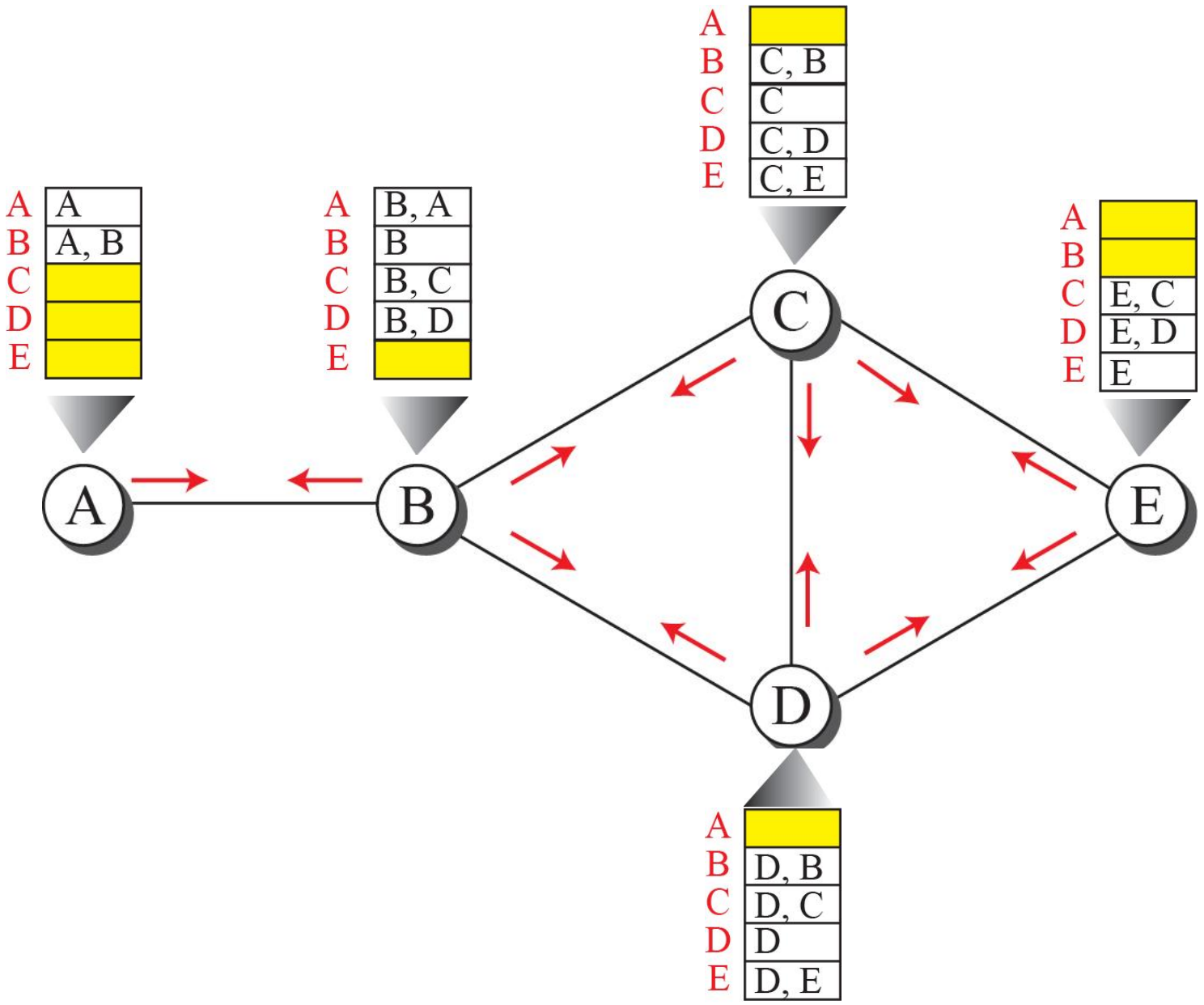
Creation of Spanning Trees

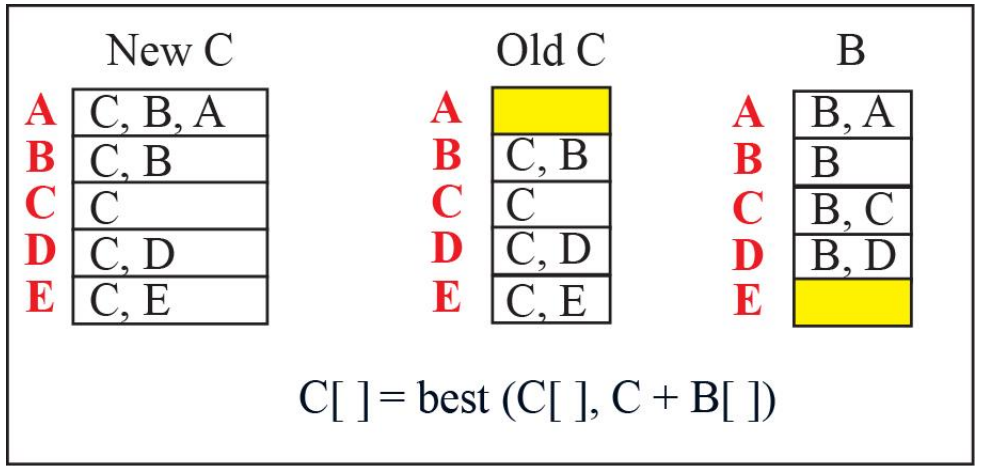
In path-vector routing, which is similar to distance-vector routing, the route decisions happen gradually and independently for each node. When a node starts up, it creates a path vector based on the info it gets from its nearby neighbors. This info is collected by sending greeting messages. **Each node then shares its path vector with all immediate neighbors.**

This process doesn't happen all at once; each node does it when it boots up. The path vectors are then updated using an equation similar to Bellman-Ford, but with the node's own rules instead of aiming for the least cost. The equation is like this:

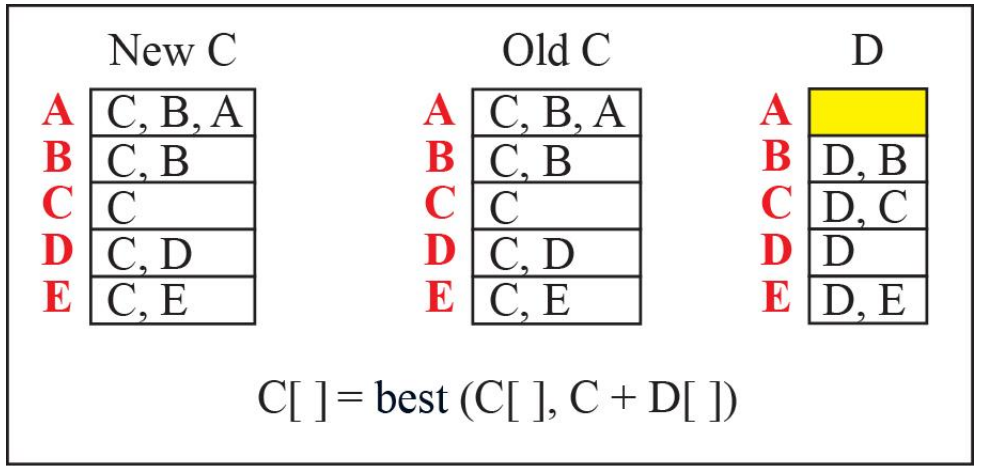
$$\textbf{Path}(x, y) = \textbf{best}\ \{\textbf{Path}(x, y), [(x + \textbf{Path}(v, y)]\}$$

for all v's in the network. This means adding x to the beginning of the path. But be careful not to add a node to an empty path, as an empty path means it doesn't exist. The policy involves choosing the best path among many. Path-vector routing also adds a condition: if the path includes x, it's discarded to avoid looping back to itself.


A
A
A, B
B
B, A
B
B, C
B, D
C
C, B
C
C, D
C, E
D
D, B
D, C
D
D, E
E
E, C
E, D
E
A
B
C
D
E



Event 1: C receives a copy of B's vector



Event 2: C receives a copy of D's vector

Note:
X [ ]: vector X
Y: node Y

In Figure , you can see the path vector of node C after two events. In the first event, node C gets B's vector, improving its own: now it knows how to reach node A. In the second event, node C gets D's vector, but it doesn't change its own vector. After the first event, node C's vector stabilizes and becomes its forwarding table.

Path-Vector Algorithm

```
1   Path_Vector_Routing ( )
2   {
3       // Initialization
4       for (y = 1 to N)
5       {
6           if (y is myself)
7               Path[y] = myself
8           else if (y is a neighbor)
9               Path[y] = myself + neighbor node
10          else
11              Path[y] = empty
12      }
13      Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
14      // Update
15      repeat (forever)
16      {
17          wait (for a vector Path_w from a neighbor w)
18          for (y = 1 to N)
19          {
20              if (Path_w includes myself)
21                  discard the path                    // Avoid any loop
22              else
23                  Path[y] = best {Path[y], (myself + Path_w[y])}
24          }
25          If (there is a change in the vector)
26              Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
27      }
28  } // End of Path Vector
```

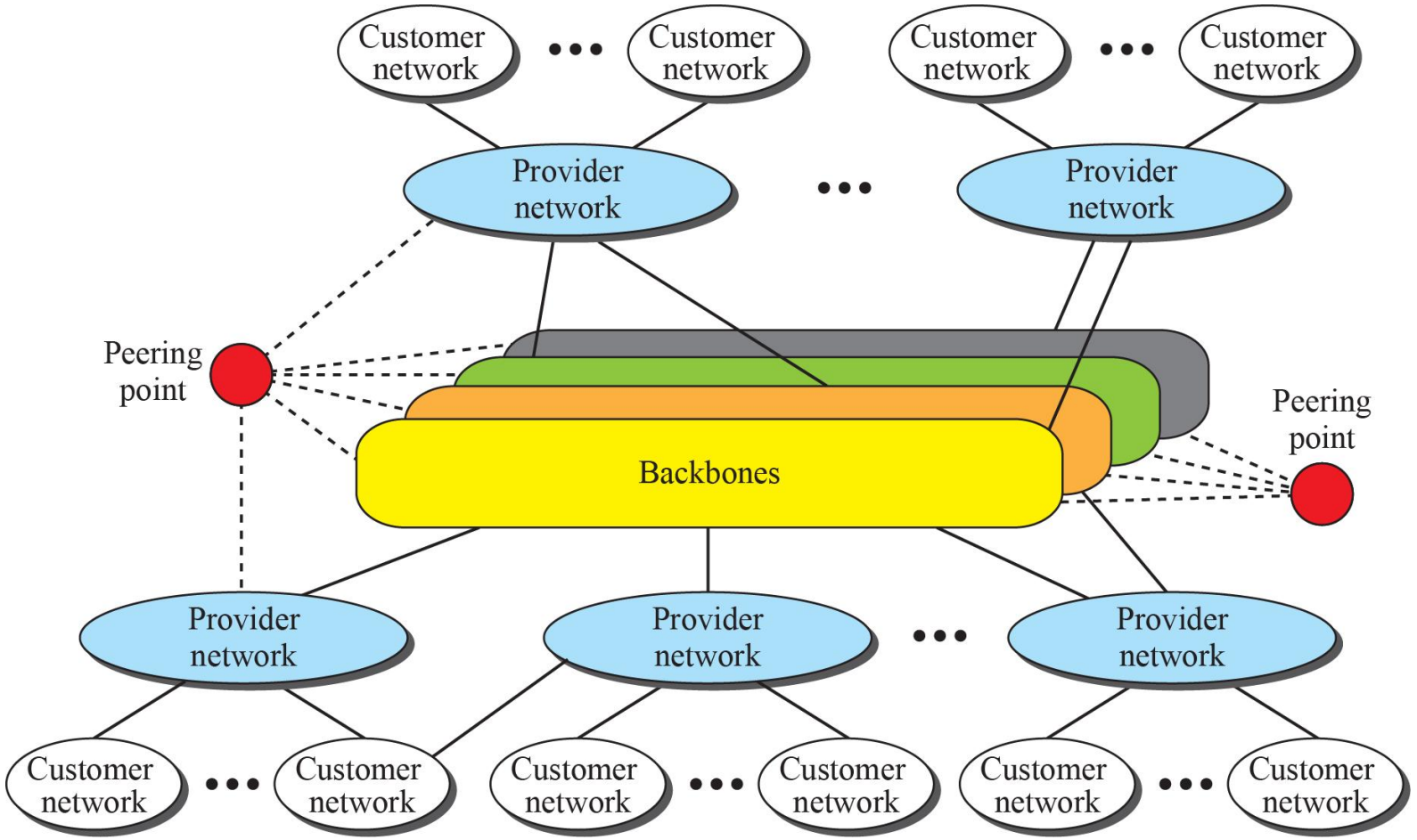
Lines 4 to 12 show the initialization for the node. Lines 17 to 24 show how the node updates its vector after receiving a vector from the neighbor. The update process
is repeated forever. We can see the similarities between this algorithm and the DV algorithm.

Unicast Routing Protocols

We talked about how to find paths for data in a network using unicast routing algorithms. Now, let's look at the specific methods used on the Internet. **There are three main protocols: RIP, which follows the distance-vector algorithm; OSPF, which uses the link-state algorithm, and BGP, based on the path-vector algorithm.** These protocols not only use the algorithms we discussed earlier but also have their own rules for communication between routers, defining their scope and how they interact with other protocols.

Internet Structure

There are several backbones run by private communication companies that provide global connectivity. These backbones are connected by some peering points that allow connectivity between backbones. At a lower level, there are some provider networks that use the backbones for global connectivity but provide services to Internet customers. Finally, there are some consumer networks that use the services provided by the provider networks. Any of these three entities (backbone, provider network, or customer network) can be called an Internet Service Provider or ISP. They provide services, but at different levels.


Customer network
Customer network
Customer network
Customer network
Provider network
Provider network
Peering point
Backbones
Peering point
Provider network
Provider network
Provider network
Customer network
Customer network
Customer network
Customer network
Customer network
Customer network

Hierarchical Routing

The Internet is like a massive web of networks and routers, and using just one routing method for the entire Internet would create problems. There are two main issues:

1. **Scalability Problem:** If we use a single method, the tables used to figure out where to send data become too large, making it slow and causing a lot of network traffic.

2. **Administrative Issue:** The Internet is divided into parts, each managed by a different authority (like Internet Service Providers or ISPs). **Each of these authorities wants control over its own part. They might want to use specific equipment, choose their own routing method, or set rules for how data should travel through their part.**

To solve this, we use a system called hierarchical routing. Think of each ISP as its own little kingdom. **Inside each kingdom, they can use whatever routing method they like (intradomain routing protocol). But to connect all these kingdoms, we need a global method (interdomain routing protocol).**

Right now, the common intradomain methods are RIP and OSPF, and the only interdomain method is BGP. This might change with IPv6, but for now, this system helps keep the Internet running smoothly.

Autonomous Systems

Each Internet Service Provider (ISP) is like its own little kingdom, and they are each given a unique number called an Autonomous System Number (ASN) by the ICANN. These ASNs are like ID numbers for each kingdom.

Now, these kingdoms are not classified by their size but by how they connect to other kingdoms. There are three types:

1. Stub AS: Imagine a kingdom with only one bridge to another kingdom. This kingdom is a stub AS. It's like a customer network that either starts or ends data traffic but doesn't let it pass through.

2. Multihomed AS: This kingdom has more than one bridge to other kingdoms, but again, it doesn't let data pass through. It's like a customer using services from multiple providers but not allowing their data to travel through.

3. Transient AS: This kingdom is connected to multiple other kingdoms and lets data traffic pass through. Provider networks and the backbone of the Internet are examples of transient ASs.

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.

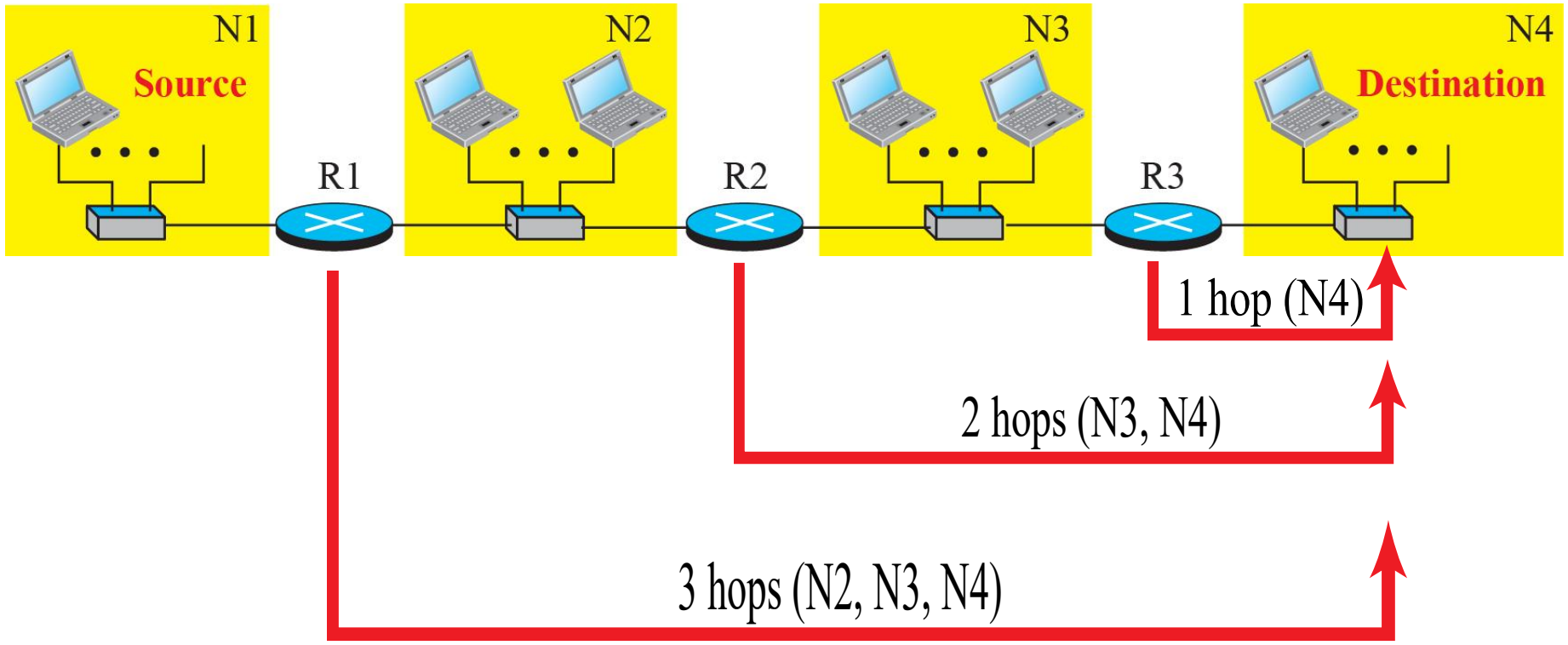
Hop Count

In this routing protocol called RIP, routers use a method called the distance-vector algorithm to figure out the best path to send data within a network. Now, they've **tweaked this algorithm in a couple of ways**:

1. **Advertised Costs**: Instead of telling each other about the distance to other routers in a theoretical graph, **RIP routers share the cost of reaching different networks (subnets) within their own kingdom** (Autonomous System or AS).

2. **Simplified Cost Calculation**: The cost is made simple. They **count the number of "hops"** or networks a packet needs to travel through, from where it starts to where it needs to go. They don't consider fancy things like delay or bandwidth. It's just a straightforward count.

3. **Hop Count**: The **maximum number of hops or networks a packet can go through is 15**. If it needs more than that, it's considered impossible (infinity). So, RIP works best in smaller kingdoms where the whole journey doesn't exceed 15 hops.


N1
Source
N2
N3
N4
Destination
R1
R2
R3
1 hop (N4)
2 hops (N3, N4)
3 hops (N2, N3, N4)

Forwarding Tables

Imagine you have a network of routers in a kingdom, and they need to decide how to send data packets to different places. In RIP, each router keeps something called a forwarding table. This table has three parts:

1. **Destination Network**: This is where the data packet is supposed to go.
2. **Next Router**: The router that the packet should go to next on its journey.
3. **Cost (Hops)**: How many networks or "hops" the packet needs to go through to reach its destination.

So, for example, if Router 1 wants to send a packet to Network 4, its forwarding table says to send it to Router 2 first, then Router 2's table says to send it to Router 3, and finally, Router 3 says there's no more hopping, you've reached Network 4.

Now, the third column, the cost or number of hops, isn't crucial for sending the packet, but it's super important when things change. If there's a shift in the route, the routers need that cost information to update their forwarding tables. It's like having a roadmap that tells you not just where to go, but also how far each step is. Handy, right?

Forwarding table for R1

Destination network	Next router	Cost in hops
N1	——	1
N2	——	1
N3	R2	2
N4	R2	3

Forwarding table for R2

Destination network	Next router	Cost in hops
N1	R1	2
N2	——	1
N3	——	1
N4	R3	2

Forwarding table for R3

Destination network	Next router	Cost in hops
N1	R2	3
N2	R2	2
N3	——	1
N4	——	1

RIP Implementation

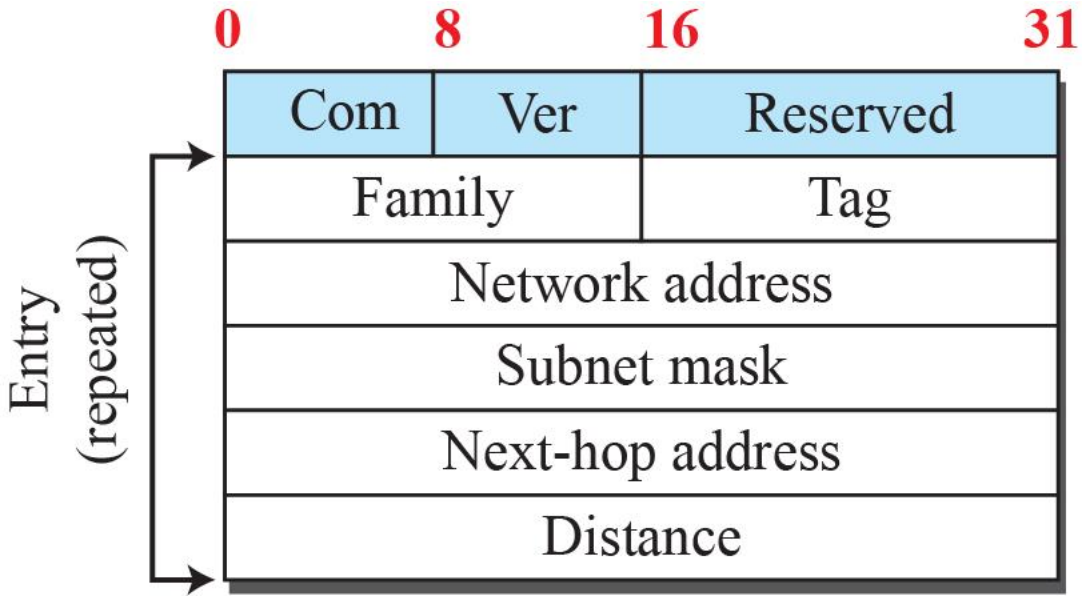
RIP, the Routing Information Protocol, is like a backstage helper for routers on the Internet. It uses a process called "routed," speaking a special language (UDP) on port 520. Even though it works in the background, it's crucial for routing data (IP) on the network.

There are two versions, RIP-1 and RIP-2,
RIP-2 being the upgraded and more versatile one. It wraps its messages in layers (UDP and IP), making it work like an app creating instructions for the network.

RIP Messages

Imagine two friends, a client, and a server, who need to chat and share information. In the world of RIP-2 (Routing Information Protocol), these friends, or processes, exchange messages.

Now, RIP-2 messages have a specific structure, shown in a figure. Think of these messages like letters. Each letter has parts called "entries," and these entries are like lines in a router's forwarding table.



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-2 has two types of messages:

1. **Request Message**: It's like a friend asking questions. A router sends this when it starts up or when it needs information about certain entries. It's saying, "Hey, tell me about these lines in your forwarding table."

2. **Response (or Update) Message**: This is the friend responding. There are two kinds of responses:

- Solicited Response: It's like answering a friend's specific question. The router sends this in response to a request message, giving information about the requested entries.
- Unsolicited Response: This is like a periodic check-in. The router sends this every 30 seconds or when there's a change in its forwarding table. It's like saying, "Just updating you on what's going on in my world."

RIP Algorithm

Imagine routers are like travelers sharing their journey plans. In RIP, they use the same algorithm we talked about earlier, but with a few tweaks for updating their travel plans (forwarding tables):

1. **Sharing the Whole Plan**: Instead of just sharing parts of their plans, routers share their entire forwarding table. It's like sharing the whole travel itinerary.

2. **Making Updates**: When a router receives this info, it makes some changes:
   - It adds one more hop to each cost (like saying it's one more stop on the journey).
   - It changes the next router field to the address of the router sending the info.

3. **Choosing the Best Routes**: The receiving router decides which routes to keep:
   - If a route is new, it adds it.
   - If the received route is cheaper, it replaces the old one.
   - If the received route is more expensive but has the same next router, it replaces the old one. This is like saying, "Even though it's more expensive, it's still from the same source, so I'll take it."

4. **Handling Special Cases**: There are special cases, like when **a route that used to have a path suddenly doesn't**. Even if it's more expensive, the router considers it because the situation has changed.

5. **Sorting the Plans**: The router organizes its new forwarding table by destination, putting the most specific routes first. It's like sorting your travel plans by the most detailed information.

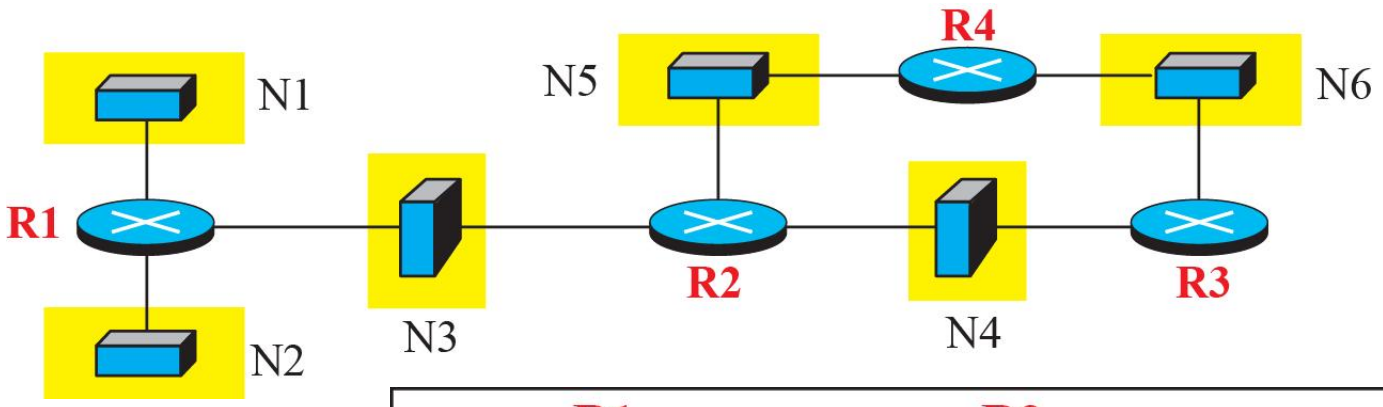
Timers in RIP

RIP has three timers to keep things running smoothly:

Periodic Timer: This timer controls when routers send regular updates. **Each router has its own timer set randomly between 25 and 35 seconds. When the timer hits zero, the router sends an update message, and the timer gets set randomly again.** This randomness prevents all routers from bombarding the network with updates at the same time.

Expiration Timer: This timer keeps track of how long a route's info is valid. When a router gets an update about a route, the expiration timer is set to 180 seconds for that route. **Every time a new update for the route comes in, the timer resets. If no updates arrive within 180 seconds, the route is considered expired,** and the destination is marked unreachable with a hop count of 16.

Garbage Collection Timer: This timer handles cleaning up old route info. When a route becomes invalid, the router doesn't immediately throw it out. Instead, it keeps advertising the route with a "unreachable" metric of 16. At the same time, **a garbage collection timer is set to 120 seconds for that route. When this timer hits zero, the route is finally removed.** This helps neighboring routers know about the invalid route before it's completely removed.



Legend

Des.: Destination network
N. R.: Next router
Cost: Cost in hops

R1

Des.	N. R.	Cost
N1	———	1
N2	———	1
N3	———	1

R2

Des.	N. R.	Cost
N3	———	1
N4	———	1
N5	———	1

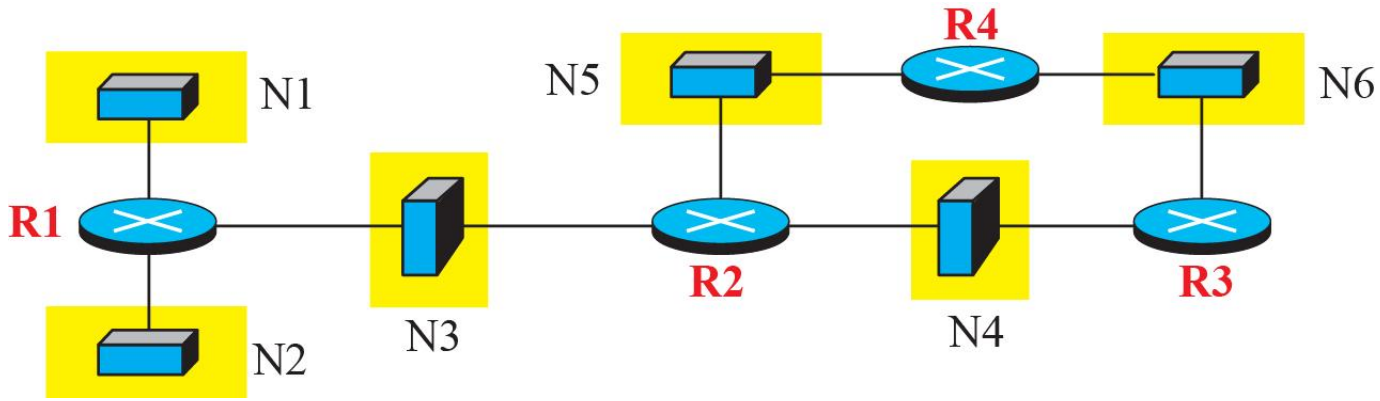
R3

Des.	N. R.	Cost
N4	———	1
N6	———	1

R4

Des.	N. R.	Cost
N5	———	1
N6	———	1

Forwarding tables after all routers booted



Legend

Des.: Destination network
N. R.: Next router
Cost: Cost in hops
: New route
: Old route

New R1

Des.	N. R.	Cost
N1	——	1
N2	——	1
N3	——	1
N4	R2	2
N5	R2	2

Old R1

Des.	N. R.	Cost
N1	——	1
N2	——	1
N3	——	1

R2 Seen by R1

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	R2	2

New R3

Des.	N. R.	Cost
N3	R2	2
N4	——	1
N5	R2	2
N6	——	1

Old R3

Des.	N. R.	Cost
N4	——	1
N6	——	1

R2 Seen by R3

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	R2	2

New R4

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	——	1
N6	——	1

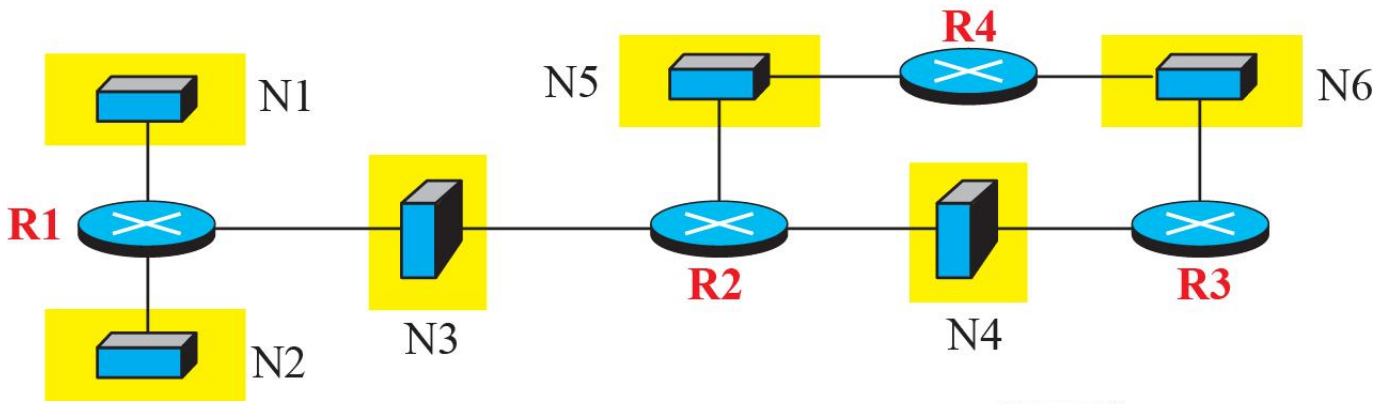
Old R4

Des.	N. R.	Cost
N5	——	1
N6	——	1

R2 Seen by R4

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	R2	2

Changes in the forwarding tables of R1, R3, and R4 after they receive a copy of R2's table



Legend

Des.: Destination network
N. R.: Next router
Cost: Cost in hops

Forwarding tables for all routers
after they have been stablized

Final R1

Des.	N. R.	Cost
N1	——	1
N2	——	1
N3	——	1
N4	R2	2
N5	R2	2
N6	R2	3

Final R2

Des.	N. R.	Cost
N1	R1	2
N2	R1	2
N3	——	1
N4	——	1
N5	——	1
N6	R3	2

Final R3

Des.	N. R.	Cost
N1	R2	3
N2	R2	3
N3	R2	2
N4	——	1
N5	R2	2
N6	——	1

Final R4

Des.	N. R.	Cost
N1	R2	3
N2	R2	3
N3	R2	2
N4	R2	2
N5	——	1
N6	——	1

Performance

1. **Update Messages**: RIP's update messages are simple and **sent only to neighbors, keeping the information local.** They usually don't create traffic issues because routers try not to send them all at the same time.

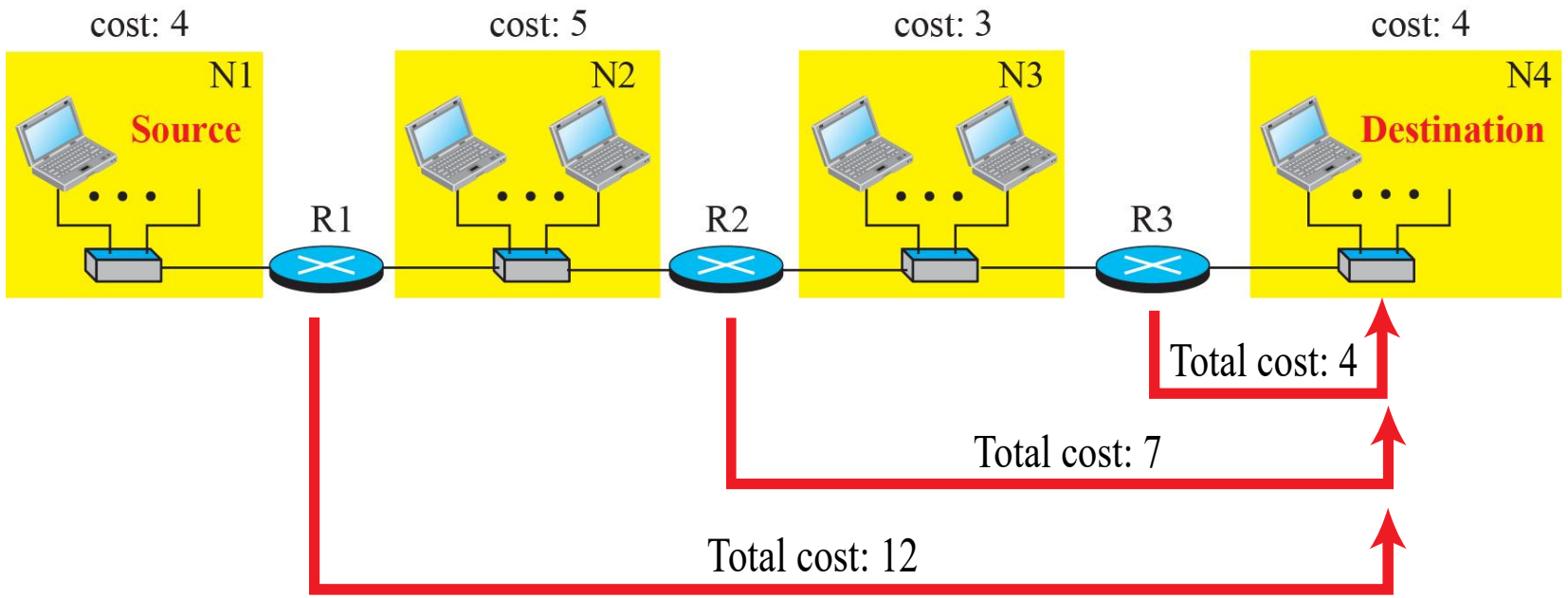
2. **Convergence of Forwarding Tables**: RIP might be a bit slow in large networks due to its distance-vector algorithm. However, since RIP limits the hops to 15 (considering 16 as infinity), convergence is generally not a big problem. **The only issues that could slow it down are count-to-infinity and loops in the network,** but strategies like poison-reverse and split-horizon can help.

3. **Robustness**: **RIP's strength lies in routers sharing what they know with their neighbors.** However, if one router fails or gets corrupted, it can cause problems for all routers in the network. This interconnectedness means issues can spread across the network, affecting forwarding in each router.

Open Shortest Path First (OSPF)

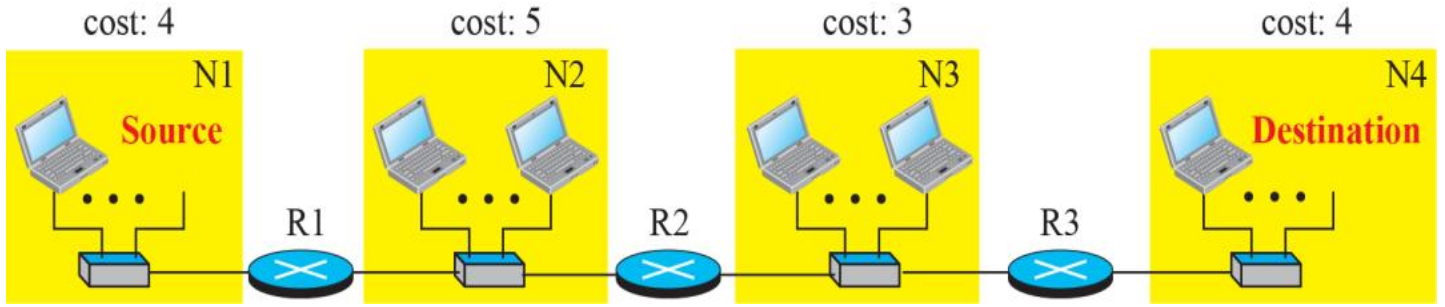
Open Shortest Path First (OSPF) is also an intradomain routing protocol like RIP, but it is based on the link-state routing protocol we described earlier in the chapter. OSPF is an open protocol, which means that the specification is a public document.

Metric - In OSPF, the cost of reaching a destination is calculated from the source router to the destination network. Unlike RIP, **OSPF allows assigning weights to links based on factors like throughput and reliability.** Administrators can also choose to use hop count as the cost. What's unique is that OSPF considers different service types, each with its own weight. Check out Figure (next page) to visualize the cost concept, and compare it with Figure (pg. 174) in RIP.


cost: 4
cost: 5
cost: 3
cost: 4
N1
Source
N2
N3
N4
Destination
R1
R2
R3
Total cost: 4
Total cost: 7
Total cost: 12

Forwarding Tables

Each OSPF router figures out the best path to a destination by creating a forwarding table using Dijkstra's algorithm. In Figure (on next page), you can see these tables for a simple AS. When **comparing OSPF and RIP forwarding tables in the same AS, the only difference is the cost values.** Essentially, **if we use hop count for OSPF, the tables become identical.** This consistency arises because both protocols use shortest-path trees to determine the best route from source to destination.



The internet from previous figure

Forwarding table for R1

Destination network	Next router	Cost
N1	——	
N2	——	
N3	R2	
N4	R2	12

Forwarding table for R2

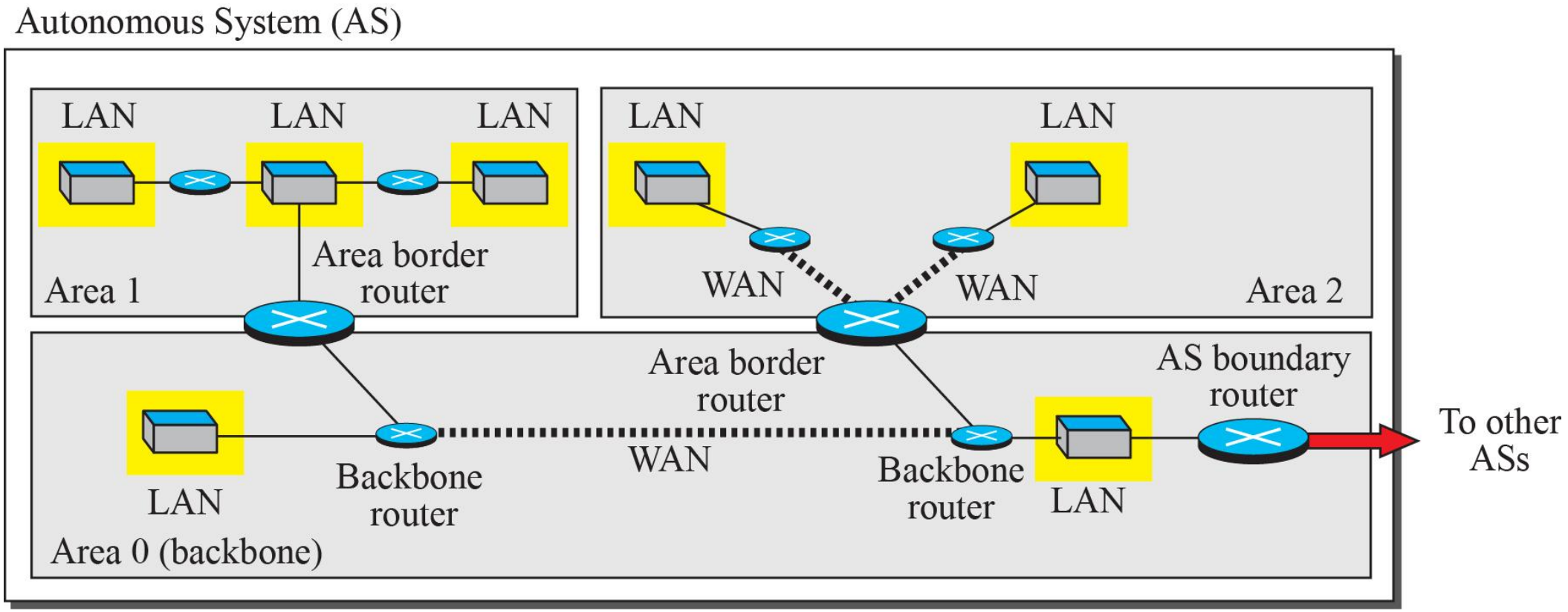
Destination network	Next router	Cost
N1	R1	9
N2	——	5
N3	——	3
N4	R3	7

Forwarding table for R3

Destination network	Next router	Cost
N1	R2	12
N2	R2	8
N3	——	3
N4	——	4

Areas

OSPF is designed for both small and large autonomous systems, unlike RIP, which is more suited for small ones. However, OSPF's creation of shortest-path trees involves flooding the entire AS with Link State Packets (LSPs), **causing potential traffic issues in large ASs**. To manage this, OSPF divides the AS into smaller sections called areas, **introducing a hierarchy. The first level is the autonomous system, and the second is the area.** Every router in an area knows about link states within its area and others. **A special area, the backbone, acts as the glue connecting all areas.** Routers in the backbone share information between areas, ensuring comprehensive communication. Each area has an identification, with the backbone's ID as zero. Check out Figure (next page) for a visual representation.


Autonomous System (AS)
LAN
LAN
LAN
Area border
router
Area 1
LAN
LAN
WAN
WAN
Area 2
Area border
router
AS boundary
router
To other
ASs
LAN
Backbone
router
WAN
Backbone
router
LAN
Area 0 (backbone)

Link-State Advertisement

OSPF works using the link-state routing algorithm, **where routers share the state of each link with neighbors to form the LSDB.** In the real world, it's not just routers and simple links; there are various entities, link types, and associated costs. To represent this complexity, **OSPF uses five types of advertisements: router link, network link, summary link to network, summary link to AS border router, and external link.** These advertisements convey different information about the network structure. Check out Figure (next page) for a visual guide to these advertisements and their purposes.

Imagine you're in a big city, and there are different ways to get around—roads, highways, and maybe even some secret shortcuts. Now, think of routers as guides telling you about these paths. They use different types of advertisements to let everyone know what's out there.

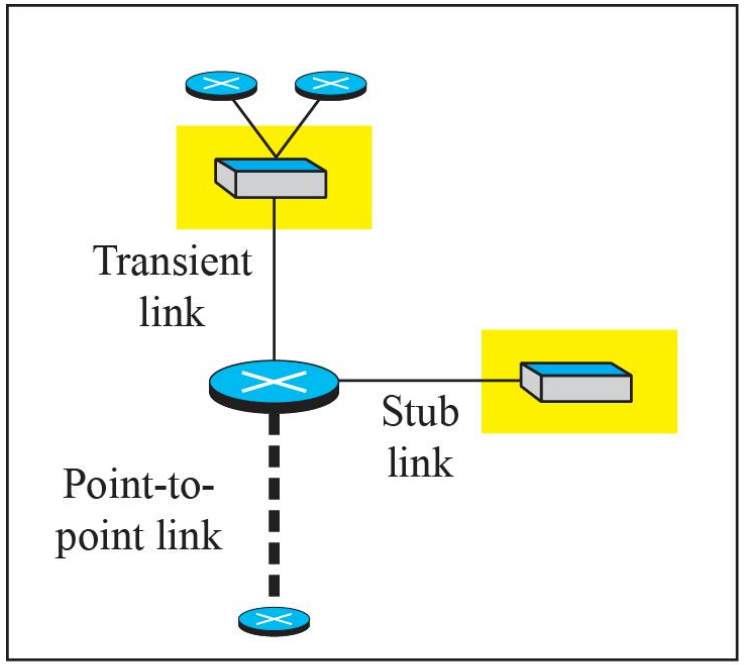
1. **Router Link**: It's like a router raising its hand and saying, "Hey, I'm here!" **It shares its address and can talk about different paths it knows.** There are three types:
   - **Transient Link**: A route to a temporary area connected by routers. It tells you how to reach that area and the cost.
   - **Stub Link**: Points to a dead-end network. Shares the network's address and the cost to reach it.
   - **Point-to-Point Link**: Direct route to another router. It gives the address and the cost to reach that specific router.

2. **Network Link**: This is like a network shouting out its existence, but it's too shy to do it directly. So, **one router becomes the spokesperson and shares the addresses of all routers in the network.** The cost isn't mentioned because each router will share that when it talks about its own link.

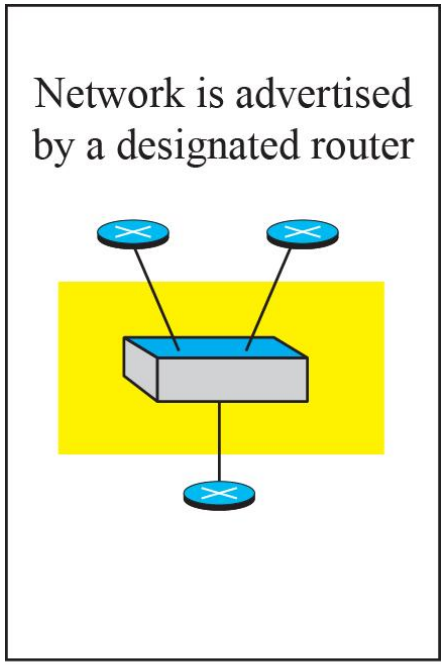
3. **Summary Link to Network**: Picture an important messenger (area border router) collecting news about all the roads in the city. It then **shares the summarized info with a specific area to keep everyone in the loop.**

4. **Summary Link to AS**: Now, think of a super messenger (AS router) collecting news not just from the city but from other cities (ASs). **It shares the summarized news with the main city area, which then spreads the word to other areas.**

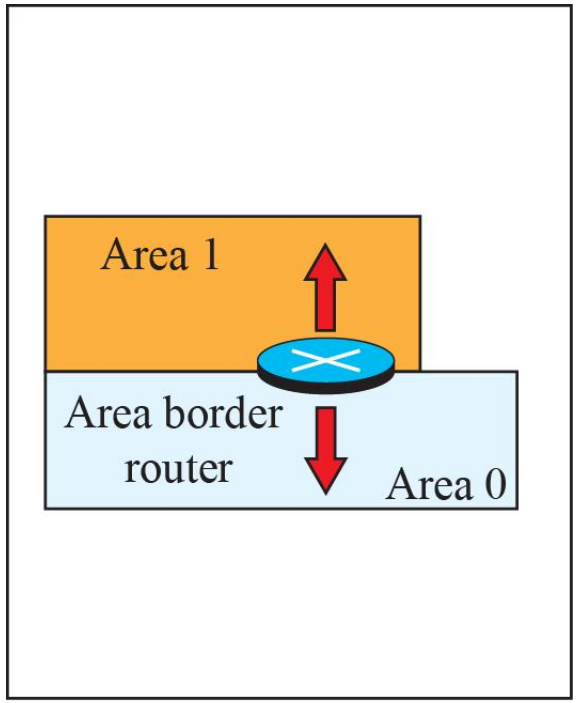
5. **External Link**: Imagine an ambassador (AS router again) announcing news about a city outside its own. It **shares this info with the main city area, which then tells everyone else about the new city.**



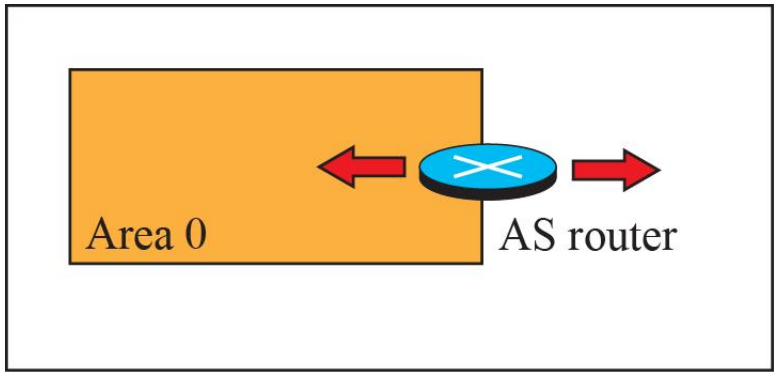
a. Router link



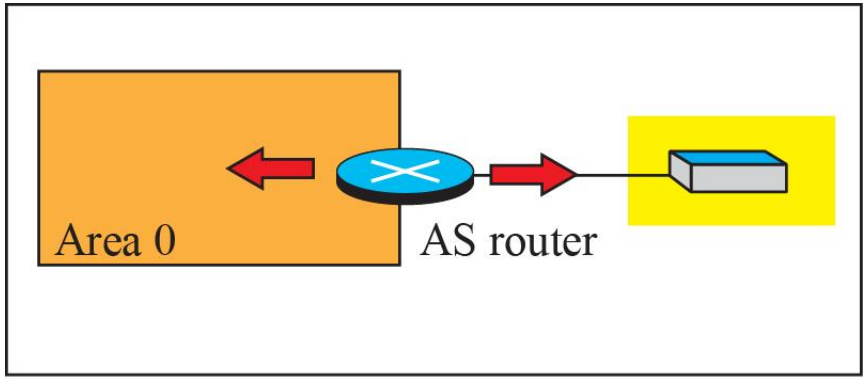
b. Network link



c. Summary link to network



d. Summary link to AS



e. External link

OSPF Messages

Imagine OSPF as a language routers use to chat with each other. They have five types of messages to make sure everyone's on the same page.

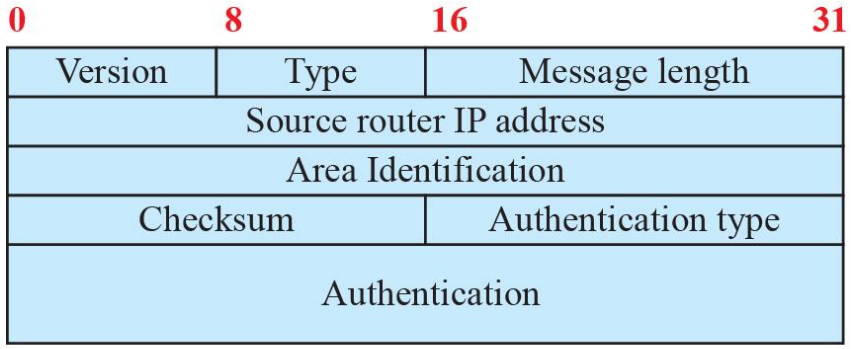
Hello Message (Type 1): It's like a friendly wave between routers. One router says, "Hey, I'm here!" and tells about its neighbors. A bit like introducing yourself at a party.

Database Description Message (Type 2): Imagine you just joined the party, and someone hands you a big booklet about everything happening. This message is like that booklet. **It helps a new router catch up with all the info in the network.**

Link-State Request Message (Type 3): **Sometimes, a router needs more details about a specific part of the network.** It's like asking for the juicy details about a certain topic at the party.

Link-State Update Message (Type 4): This is the **main message for sharing news about the network.** It has different versions, like sharing info about a router, a network, a summary of links, and more. It's like the routers exchanging detailed stories about what's going on.

Link-State Acknowledgment Message (Type 5): **To make sure everyone got the memo, routers send a quick "Got it!"** This helps in ensuring that the information shared in the Link-State Update message is received and understood.

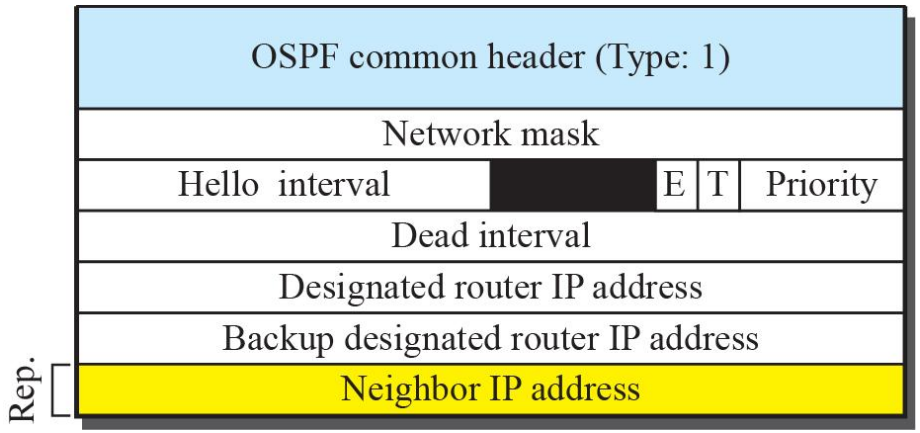


OSPF common header

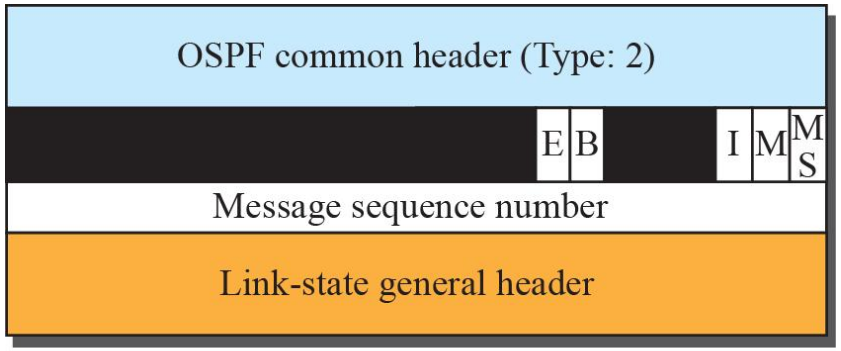
Legend

E, T, B, I, M, MS: flags used by OSPF
Priority: used to define the designated router
Rep.: Repeated as required

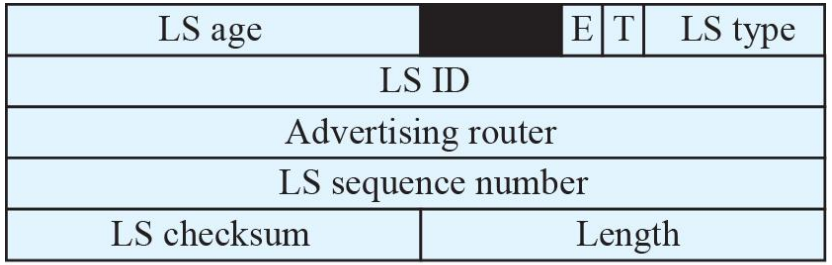
Attention



hello message



Database description

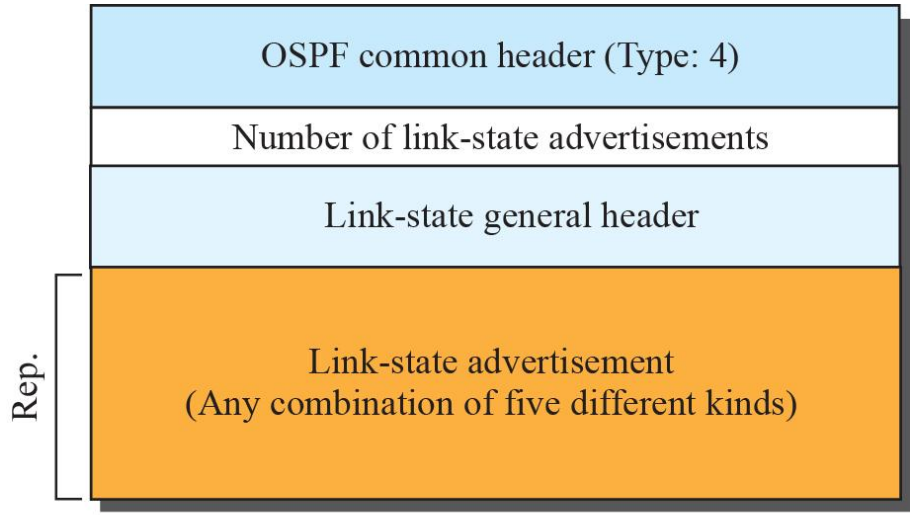


Link-state general header

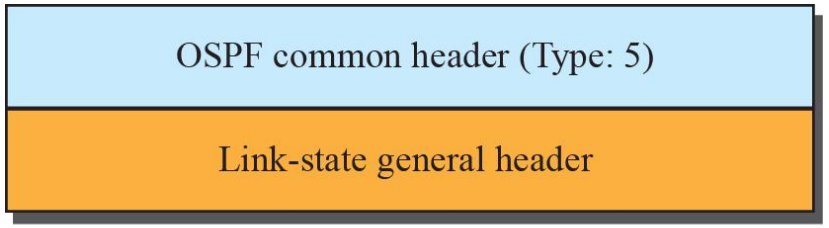
Legend

E, T, B, I, M, MS: flags used by OSPF
Priority: used to define the designated router
Rep.: Repeated as required

Attention



Link-state update



Link-state acknowledgment

Performance

- **Update Messages.** The link-state messages in OSPF have a somewhat complex format. They also are **flooded** to the whole area. **If the area is large, these messages may create heavy traffic and use a lot of bandwidth.**
- **Convergence of Forwarding Tables.** When the flooding of LSPs is completed, each router can create its own shortest-path tree and forwarding table; convergence is fairly quick. However, each **router needs to run the Dijkstra's algorithm**, which may take some time.
- **Robustness.** The OSPF protocol is more robust than RIP because, after receiving the completed LSDB, **each router is independent and does not depend on other routers in the area.** Corruption or failure in one router does not affect other routers as seriously as in RIP.

Border Gateway Protocol Version 4 (BGP4)

Imagine the internet as a big neighborhood with different sections (autonomous systems or AS). Each section has its own rules (routing protocols like RIP or OSPF). Now, to make sure everyone can talk to each other, they use a special language called BGP.

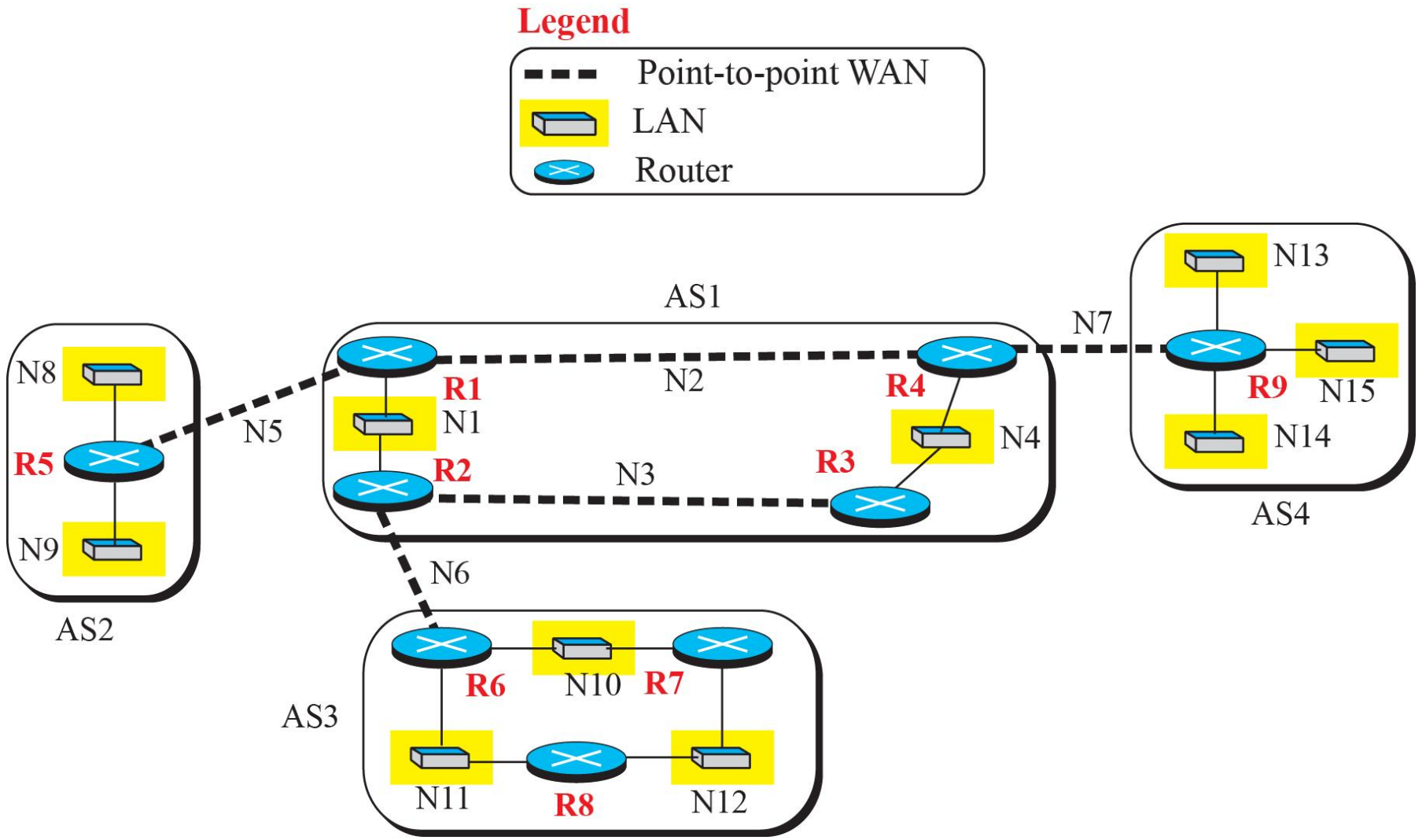
1. **AS Setup**: In our example, there are four sections (AS1, AS2, AS3, AS4). AS1 is like a transit area, and the others are like neighborhoods.

2. **Local Knowledge**: Each neighborhood's routers know how to reach places inside their own neighborhood but have no clue about the other neighborhoods.

3. **Making Friends**: To fix this, each neighborhood's border router (the one at the edge) learns a special version of BGP called eBGP. It's like having a language to talk to the neighbors in other neighborhoods.

4. **Sharing the News**: Now, all the routers in each neighborhood learn another version of BGP called iBGP. This helps them share information within their own neighborhood.

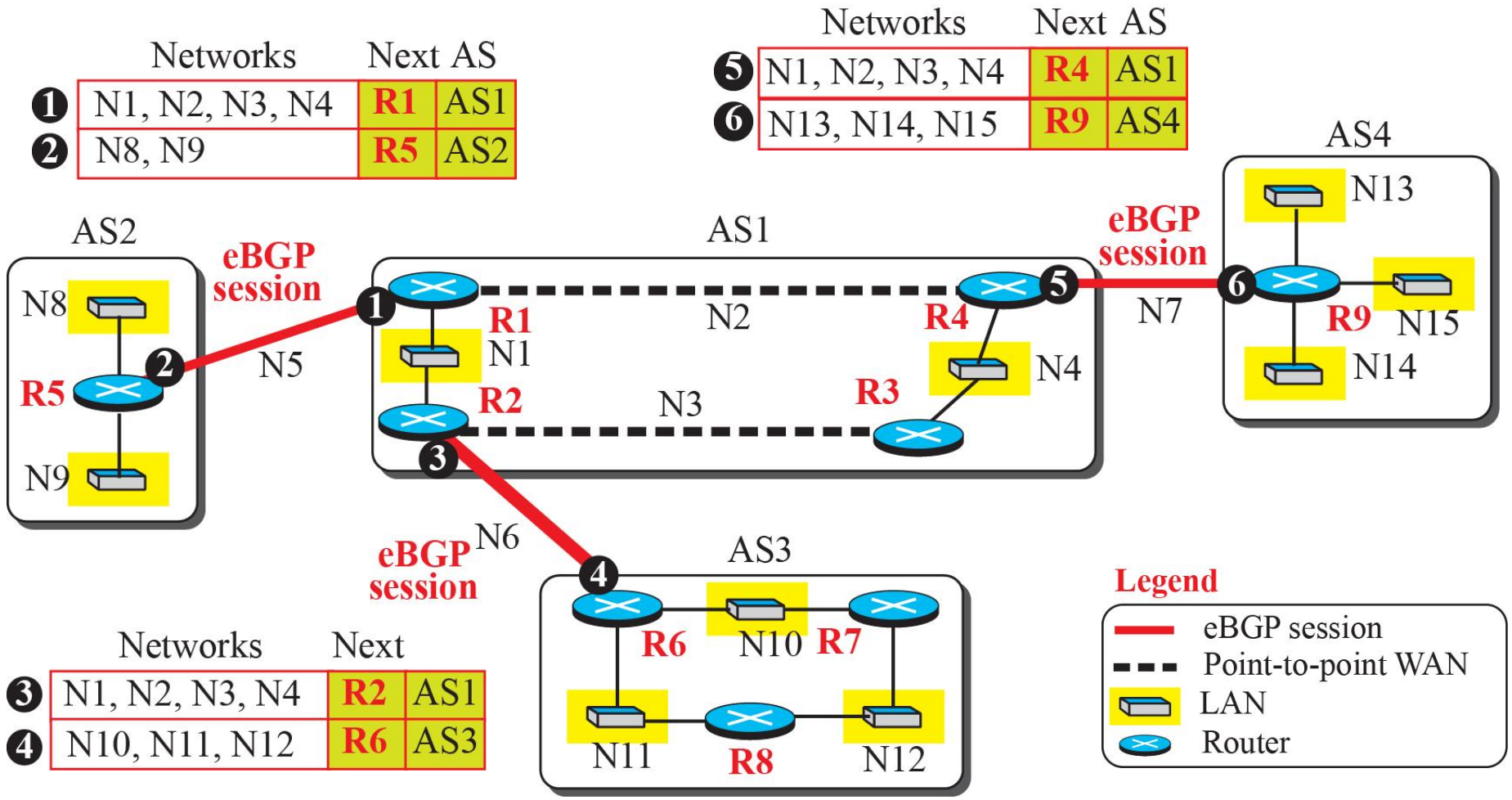
5.**Router Jobs**: So, the border routers are like multilingual experts, **knowing intradomain (inside the neighborhood), eBGP (talking to other neighborhoods), and iBGP (talking within their own neighborhood).** Other routers just need to know intradomain and iBGP.


Legend
Point-to-point WAN
LAN
Router
AS1
AS2
AS3
AS4
N1
N2
N3
N4
N5
N6
N7
N8
N9
N10
N11
N12
N13
N14
N15
R1
R2
R3
R4
R5
R6
R7
R8
R9

Operation of External BGP (eBGP)

BGP, or Border Gateway Protocol, is like a private chat between routers. When two routers want to talk, they create a connection using TCP port 179. These routers, called BGP peers, share messages to announce reachable networks. In eBGP (external BGP), routers from different **ASs connect and exchange info over physical connections, forming sessions.** For effective communication, logical TCP connections, or sessions, are needed.

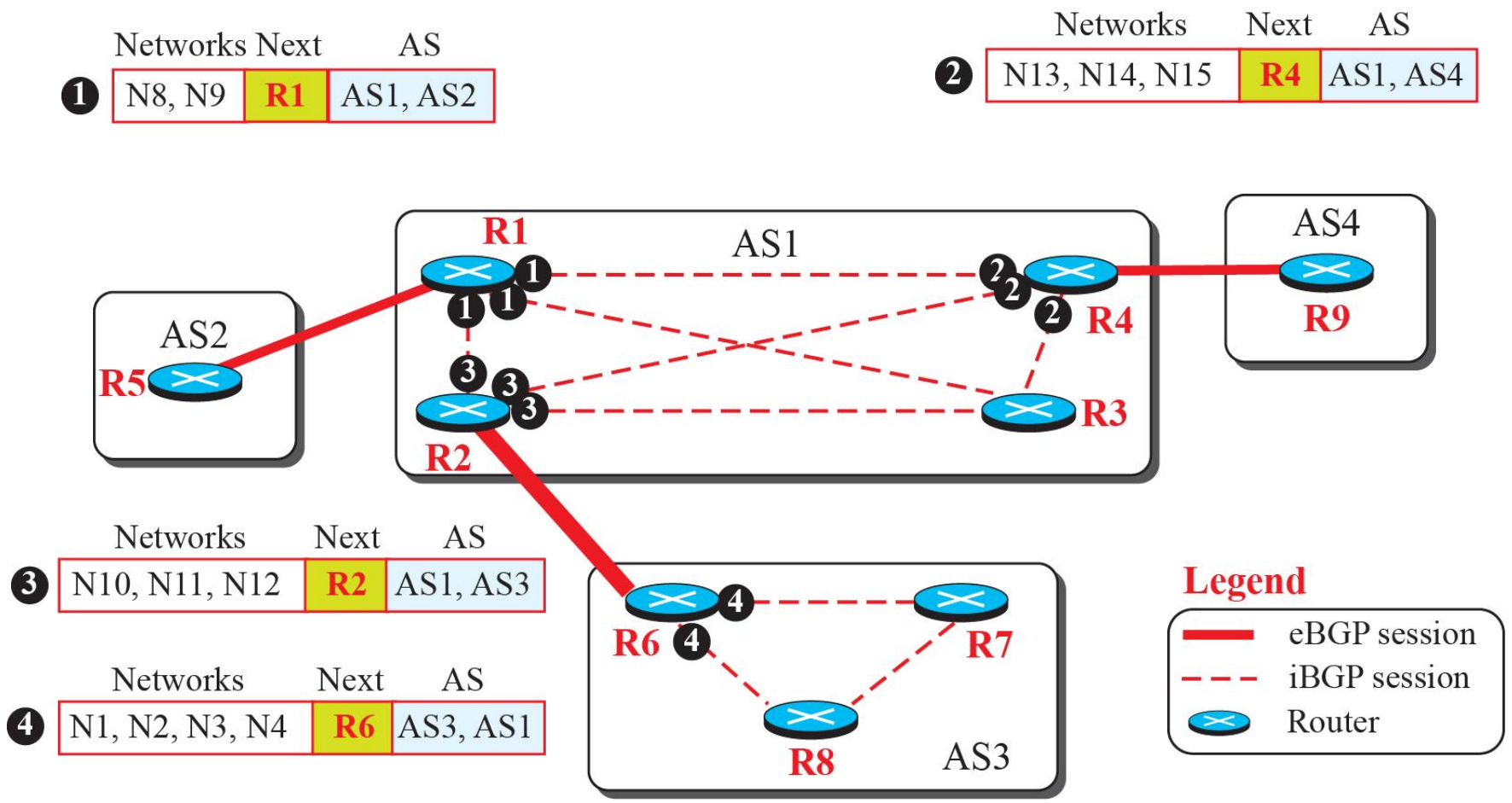
The update messages during eBGP sessions help routers know how to route packets to certain networks. However, **there are two issues: some routers don't know how to route packets to non-neighbor ASs, and non-border routers lack knowledge about networks in other ASs. To solve this, routers, both border and non-border, run iBGP (internal BGP) to exchange complete reachability information.**


Networks
Next AS
1 N1, N2, N3, N4 R1 AS1
2 N8, N9 R5 AS2
Networks
Next AS
5 N1, N2, N3, N4 R4 AS1
6 N13, N14, N15 R9 AS4
AS2
AS1
AS4
AS3
eBGP session
eBGP session
eBGP session
N5
N6
N7
N8
N9
R5
R1
N1
N2
R4
N4
R2
N3
R3
N13
R9
N15
N14
R6
N10
R7
N11
R8
N12
Networks
Next
3 N1, N2, N3, N4 R2 AS1
4 N10, N11, N12 R6 AS3
Legend
eBGP session
Point-to-point WAN
LAN
Router

Operation of Internal BGP (iBGP)

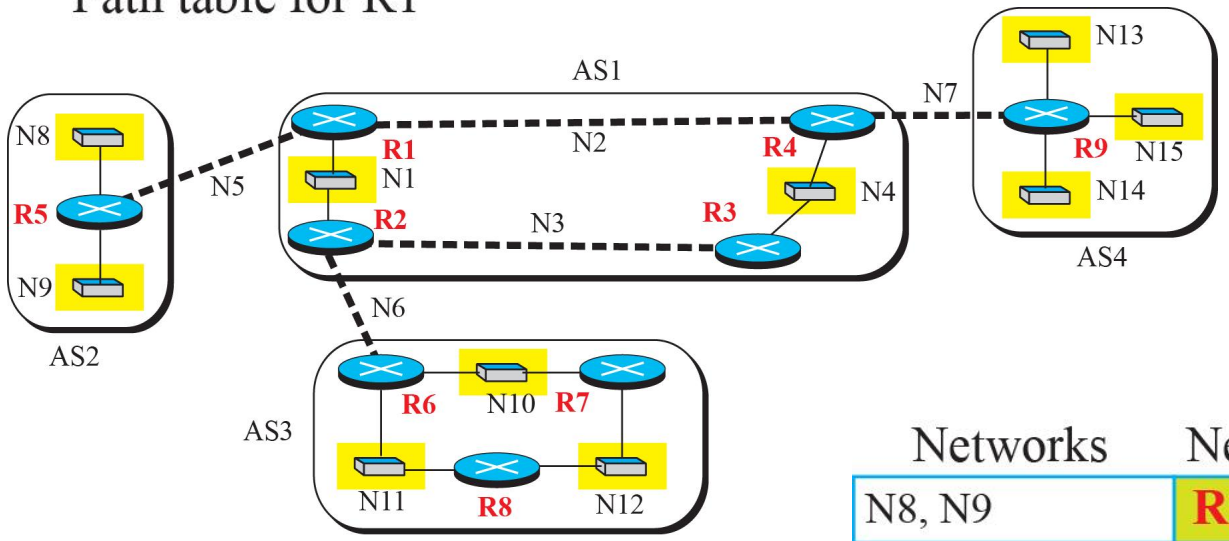
iBGP is like a private chat for routers within the same AS, **using TCP port 179.** Each router creates sessions with every other router in the AS, forming a fully connected mesh to avoid loops. However, an AS with only one router can't have iBGP sessions.

In an AS with n routers, there should be [n × (n – 1) / 2] iBGP sessions. These routers exchange messages to announce reachable networks within the AS. The process ensures that routers know how to route packets to various networks. Updates are continuous, with routers combining information from eBGP and iBGP to create path tables, guiding packet delivery through the best routes.


Networks Next AS
1 N8, N9 R1 AS1, AS2
Networks Next AS
2 N13, N14, N15 R4 AS1, AS4
AS1
R1
R2
R3
R4
AS2
R5
AS4
R9
Networks Next AS
3 N10, N11, N12 R2 AS1, AS3
Networks Next AS
4 N1, N2, N3, N4 R6 AS3, AS1
R6
R7
R8
AS3
Legend
eBGP session
iBGP session
Router

Networks	Next	Path
N8, N9	R5	AS1, AS2
N10, N11, N12	R2	AS1, AS3
N13, N14, N15	R4	AS1, AS4

Path table for R1



Networks	Next	Path
N8, N9	R1	AS1, AS2
N10, N11, N12	R6	AS1, AS3
N13, N14, N15	R1	AS1, AS4

Path table for R2

Networks	Next	Path
N8, N9	R2	AS1, AS2
N10, N11, N12	R2	AS1, AS3
N13, N14, N15	R4	AS1, AS4

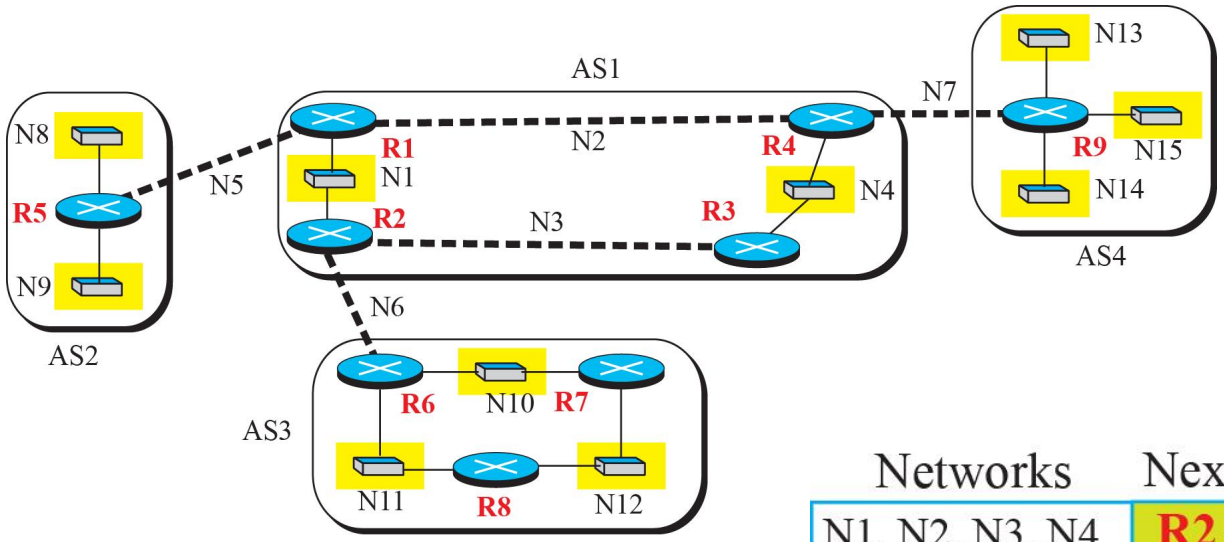
Path table for R3

Networks	Next	Path
N1, N2, N3, N4	R1	AS2, AS1
N10, N11, N12	R1	AS2, AS1, AS3
N13, N14, N15	R1	AS2, AS1, AS4

Path table for R5

Networks	Next	Path
N8, N9	R1	AS1, AS2
N10, N11, N12	R1	AS1, AS3
N13, N14, N15	R9	AS1, AS4

Path table for R4

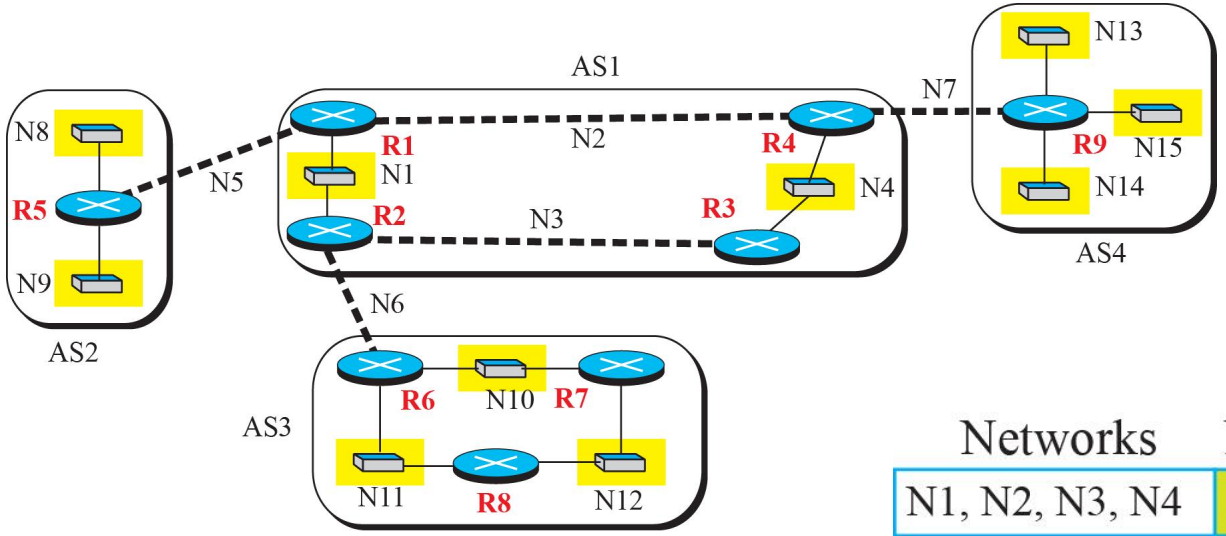


Networks	Next	Path
N1, N2, N3, N4	R2	AS3, AS1
N8, N9	R2	AS3, AS1, AS2
N13, N14, N15	R2	AS3, AS1, AS4

Path table for R6

Networks	Next	Path
N1, N2, N3, N4	R4	AS4, AS1
N8, N9	R4	AS4, AS1, AS2
N10, N11, N12	R4	AS4, AS1, AS3

Path table for R9



Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R8

Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R7

Injection of Information into Intradomain Routing

BGP's job is to enhance routing information inside an AS. The path tables it creates aren't directly used for routing; instead, they're added to intradomain forwarding tables (like RIP or OSPF) to guide packet routing.

In a stub AS, the border router adds a default entry in its forwarding table, pointing to the eBGP-connected router. For example, R5 in AS2 sets R1 as the default router. Transient ASes face a more complex task; routers like R1 in AS1 need to inject the entire path table into their intradomain forwarding tables.

Cost values are a challenge since RIP and OSPF use different metrics. A common solution is to set the cost to foreign networks equal to the cost of reaching the first AS in the path. For instance, R5's cost to other AS networks equals the cost to reach N5.

In the interdomain forwarding tables (next page), shaded areas represent BGP-injected enhancements, and default destinations are marked as zero.

Des.	Next	Cost
N1	—	1
N4	R4	2
N8	R5	1
N9	R5	1
N10	R2	2
N11	R2	2
N12	R2	2
N13	R4	2
N14	R4	2
N15	R4	2

Table for R1

Des.	Next	Cost
N1	—	1
N4	R3	2
N8	R1	2
N9	R1	2
N10	R6	1
N11	R6	1
N12	R6	1
N13	R3	3
N14	R3	3
N15	R3	3

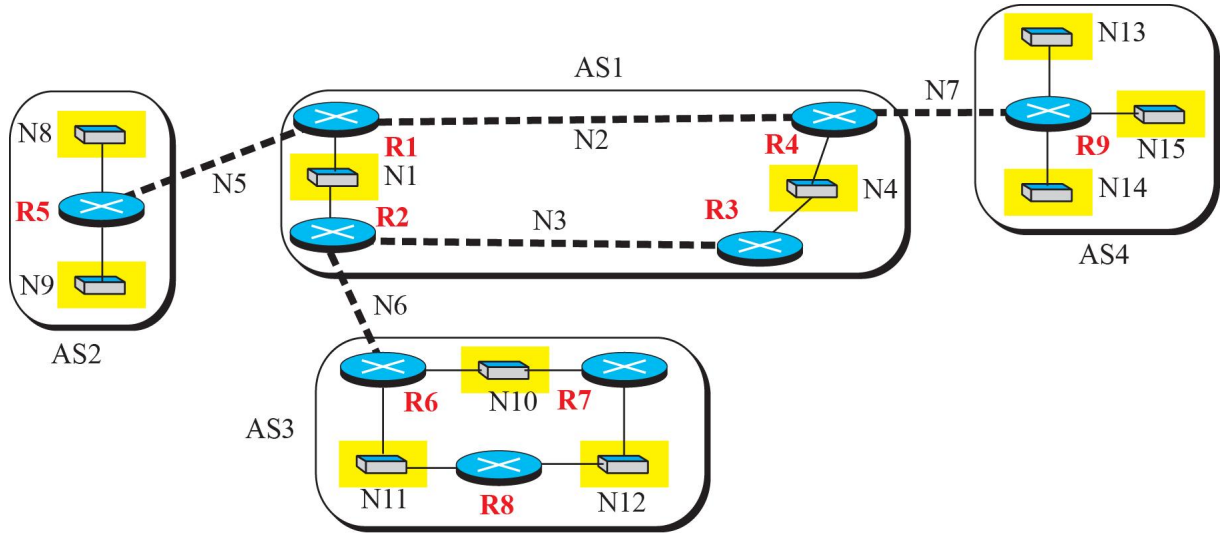
Table for R2

Des.	Next	Cost
N1	R2	2
N4	—	1
N8	R2	3
N9	R2	3
N10	R2	2
N11	R2	2
N12	R2	2
N13	R4	2
N14	R4	2
N15	R4	2

Table for R3

Des.	Next	Cost
N1	R1	2
N4	—	1
N8	R1	2
N9	R1	2
N10	R3	3
N11	R3	3
N12	R3	3
N13	R9	1
N14	R9	1
N15	R9	1

Table for R4

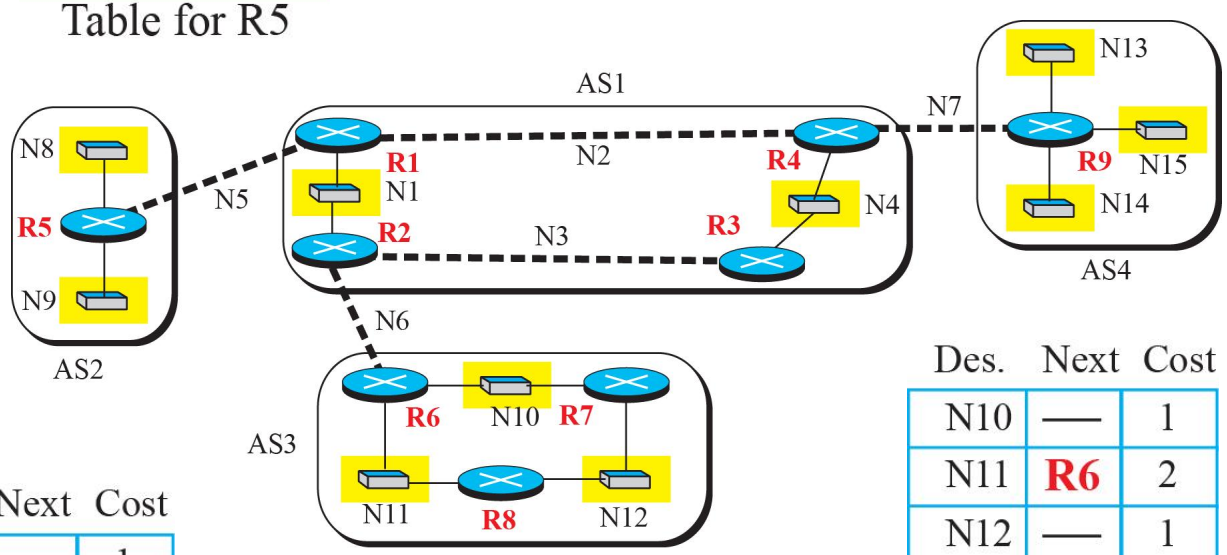


Des.	Next	Cost
N8	—	1
N9	—	1
0	**R1**	1

Table for R5

Des.	Next	Cost
N13	—	1
N14	—	1
N15	—	1
0	**R4**	1

Table for R9



Des.	Next	Cost
N10	—	1
N11	—	1
N12	**R7**	2
0	**R2**	1

Table for R6

Des.	Next	Cost
N10	**R6**	2
N11	—	1
N12	—	1
0	**R6**	2

Table for R8

Des.	Next	Cost
N10	—	1
N11	**R6**	2
N12	—	1
0	**R6**	2

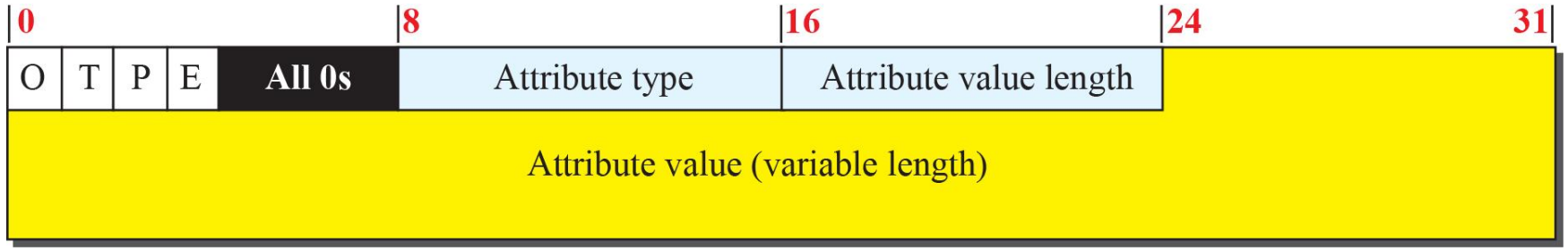
Table for R7

Path Attributes

In intradomain routing (like RIP or OSPF), reaching a destination involves knowing the next hop's address and the cost. Interdomain routing, especially in BGP, requires more details. BGP uses "path attributes" for this purpose, allowing up to seven attributes for each destination.

Path attributes fall into two main categories: **well-known and optional**. Well-known attributes must be recognized by all routers, and they can be either mandatory (always present) or discretionary (optional). Optional attributes can be transitive (can pass to the next AS) or intransitive (cannot pass). These attributes are added after the destination prefix in a BGP update message, as shown in Figure (next page).

O: Optional bit (set if attribute is optional)
P: Partial bit (set if an optional attribute in lost in transit)
T: Transitive bit (set if attribute is transitive)
E: Extended bit (set if attribute length is two bytes)



The first byte in each attribute defines the four attribute flags (as shown). The next byte defines the type of attributes assigned by ICANN (only seven types have been assigned, as explained next). The attribute value length defines the length of the attribute value field (not the length of the whole attributes section).

The following gives a brief description of each attribute.

Origin(type 1) This attribute, a well-known and mandatory one in BGP, **identifies the source of routing information.** It can be set to three values: **1 for information from an intradomain protocol (like RIP or OSPF), 2 for information from BGP itself, and 3 for information from an unknown source.**

AS-PATH(type 2) This well-known mandatory attribute, called AS-PATH, **specifies the list of autonomous systems through which a destination can be reached. It plays a crucial role in preventing routing loops.** If a router receives an update message listing its own AS in the path, it rejects that path to avoid loops. Additionally, AS-PATH is used in selecting the best route.

NEXT-HOP(type 3) This well-known mandatory attribute **specifies the next router to which a data packet should be forwarded.** It's used to inject path information gathered through eBGP and iBGP into intradomain routing protocols like RIP or OSPF.

MULTI-EXIT-DISC (type 4) The multiple-exit discriminator (MED) is an optional, non-transitive attribute in BGP. **It helps differentiate among multiple exit paths to a destination, typically based on the metric in the intradomain protocol.** The path with the **lowest MED value is chosen.** Importantly, MED is not passed from one AS to another.

LOCAL-PREF (type 5). The local preference attribute is a well-known discretionary attribute in BGP, set by administrators based on organizational policy. It assigns higher values (four-byte unsigned integers) to preferred routes. For example, **in a network with five ASs, an administrator may set a local preference of 400 to the path AS1-AS2-AS5, 300 to AS1-AS3, AS5, and 50 to AS1-AS4-AS5. This reflects the administrator's preference for more secure paths, with the highest local preference chosen when multiple routes are available.**

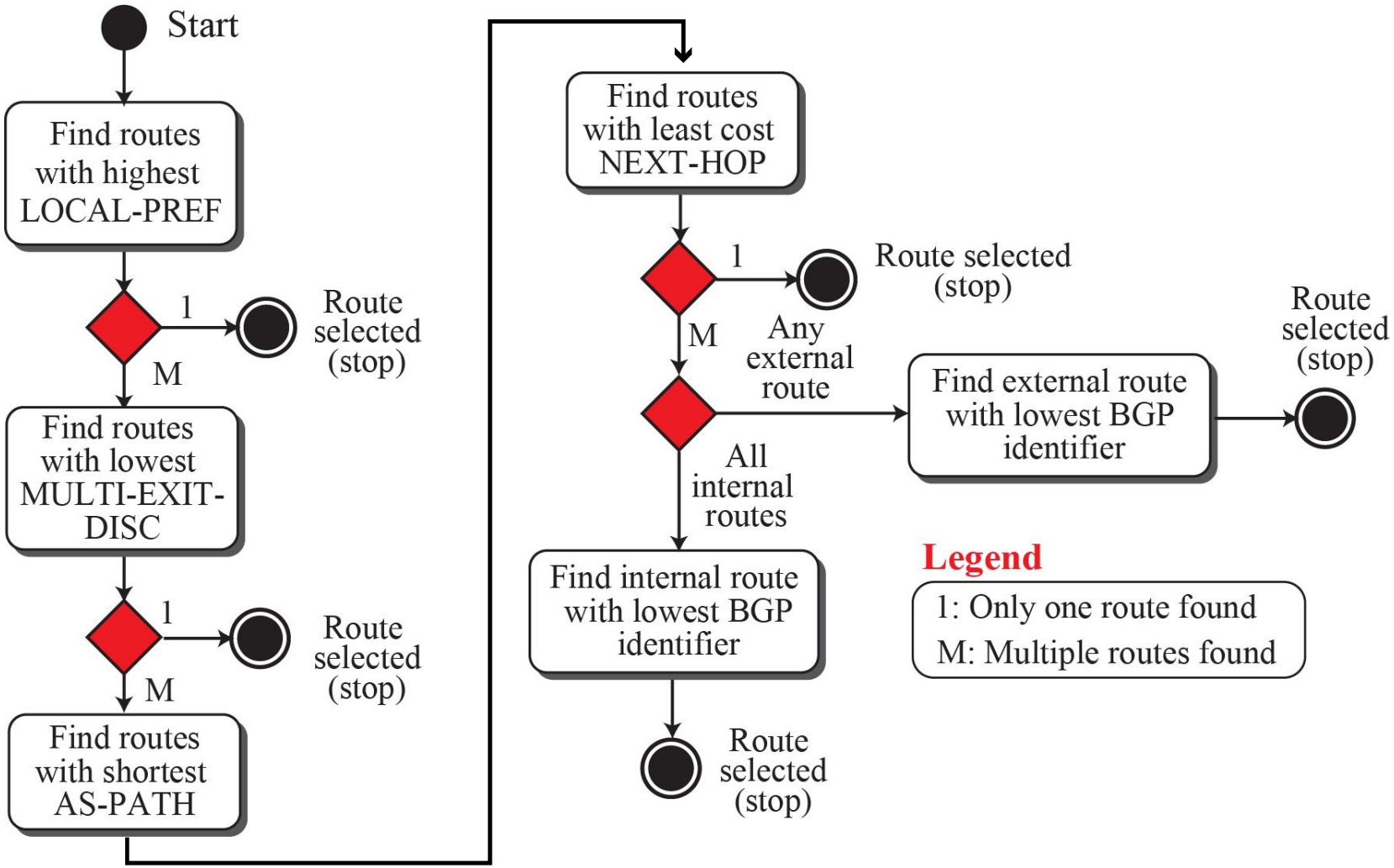
ATOMIC-AGGREGATE (type 6). This is a well-known discretionary attribute, **which defines the destination prefix as not aggregate; it only defines a single destination network.** This attribute has no value field, which means the value of the length field is zero.

AGGREGATOR (type 7). This is an optional transitive attribute, which emphasizes that the destination prefix is an aggregate. **The attribute value gives the number of the last AS that did the aggregation followed by the IP address of the router that did so.**

Route Selection

When a BGP router receives multiple routes to a destination, it goes through a selection process. Unlike intradomain routing based on the shortest-path tree, BGP's route selection is more complex due to attached attributes and different origins (eBGP or iBGP).

The router follows a flow diagram (as in next page), extracting routes based on specific criteria in each step. If only one route meets the criteria, it's selected, and the process ends. If multiple routes remain, the process continues. The first choice often relates to the LOCAL-PREF attribute, reflecting the administration's policy on the route.

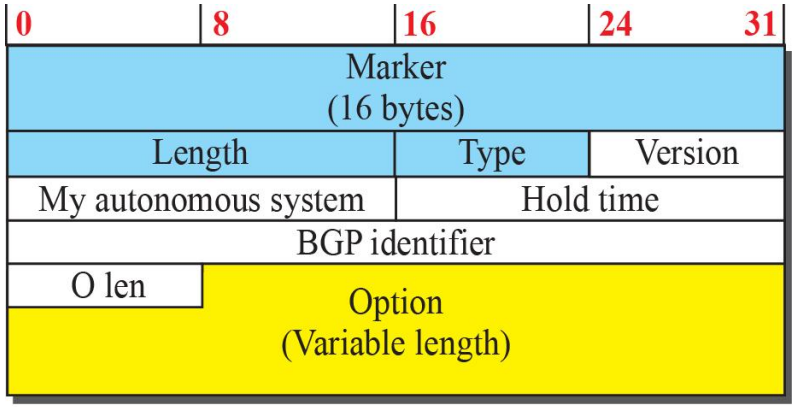

Start
Find routes with highest LOCAL-PREF
1
Route selected (stop)
M
Find routes with lowest MULTI-EXIT-DISC
1
Route selected (stop)
M
Find routes with shortest AS-PATH
Find routes with least cost NEXT-HOP
1
Route selected (stop)
M
Any external route
Find external route with lowest BGP identifier
Route selected (stop)
All internal routes
Find internal route with lowest BGP identifier
Route selected (stop)
Legend
1: Only one route found
M: Multiple routes found

Messages

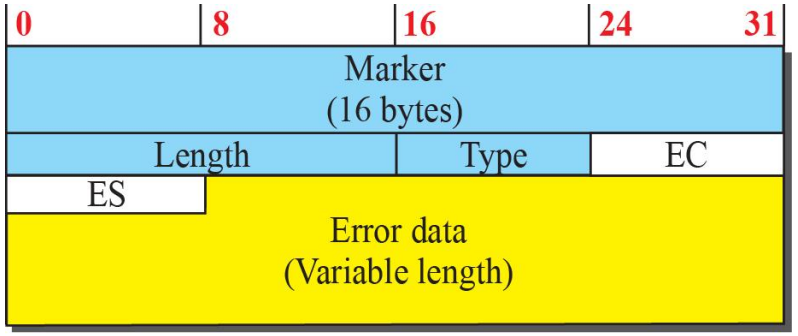
BGP uses four types of messages for communication between the BGP speakers across the ASs and inside an AS: open, update, keepalive, and notification (see Figure next page). All BGP packets share the same common header.

Open Message. To **create a neighborhood relationship**, a router running BGP opens a TCP connection with a neighbor and sends an open message.

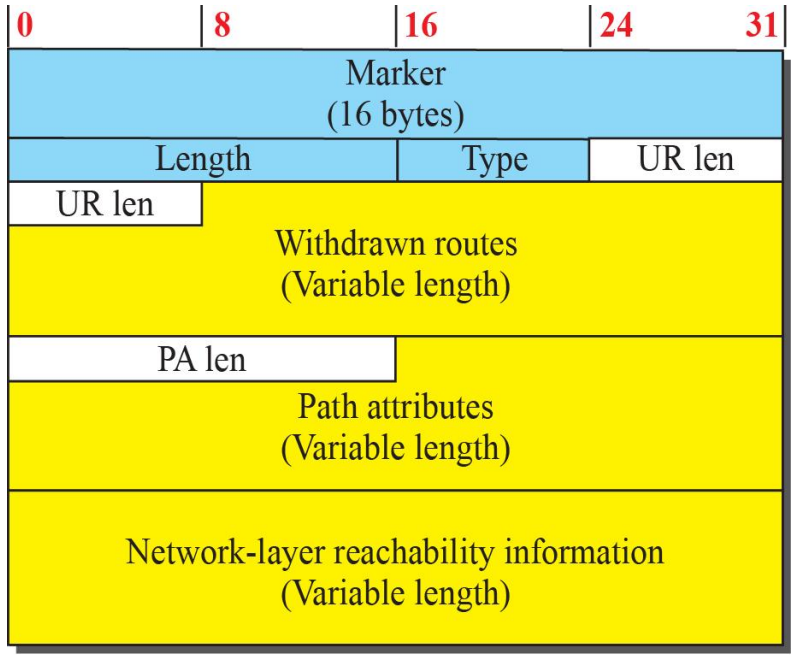
Notification. A notification message is sent by a router **whenever an error condition is detected or a router wants to close the session.**



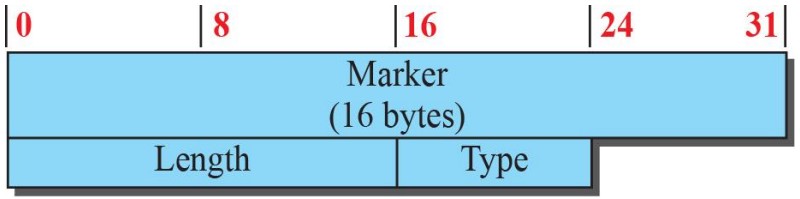
Open message (type 1)



Notification message (type 3)



Update message (type 2)



Keepalive message (type 4)

Fields in common header

Marker: Reserved for authentication
Length: Length of total message in bytes
Type: Type of message (1 to 4)

Abbreviations

O len: Option length
EC: Error code
ES: Error subcode
UR len: Unfeasible route length
PA len: Path attribute length

Update message The update message is crucial in BGP, allowing a router to **retract previously advertised destinations, announce a route to a new destination, or both.** While BGP can withdraw multiple destinations, it can only introduce one new destination (or multiple destinations with the same path attributes) in a single update message.

Keepalive Message. The BGP peers that are running exchange keepalive messages regularly (before their hold time expires) to **tell each other that they are alive.**

Performance

BGP performance can be compared with RIP. **BGP speakers exchange a lot of messages to create forwarding tables, but BGP is free from loops and count-to-infinity.** The same weakness we mention for RIP about propagation of failure and corruptness also exists in BGP.