

CS232 Final - Network layer



need help in cs232 final exam study.

2.5 - DNS: services, overview, hierarchical architecture.

explain from the texts

2.5.1 Services Provided by DNS

We have just seen that there are two ways to identify a host—by a hostname and by an IP address. People prefer the more mnemonic hostname identifier, while routers prefer fixed-length, hierarchically structured IP addresses. In order to reconcile these preferences, we need a directory service that translates hostnames to IP addresses. This is the main task of the Internet’s domain name system (DNS). The DNS is (1) a distributed database implemented in a hierarchy of DNS servers, and (2) an application-layer protocol that allows hosts to query the distributed database. The DNS servers are often UNIX machines running the Berkeley Internet Name Domain (BIND) software [BIND 2012]. The DNS protocol runs over UDP and uses port 53.

DNS is commonly employed by other application-layer protocols—including HTTP, SMTP, and FTP—to translate user-supplied hostnames to IP addresses. As an example, consider what happens when a browser (that is, an HTTP client), running on some user’s host, requests the URL `www.someschool.edu/index.html`. In order for the user’s host to be able to send an HTTP request message to the Web server `www.someschool.edu`, the user’s host must first obtain the IP address of `www.someschool.edu`. This is done as follows.

1. The same user machine runs the client side of the DNS application.
2. The browser extracts the hostname, `www.someschool.edu`, from the URL and passes the hostname to the client side of the DNS application.
3. The DNS client sends a query containing the hostname to a DNS server.
4. The DNS client eventually receives a reply, which includes the IP address for the hostname.

5. Once the browser receives the IP address from DNS, it can initiate a TCP connection to the HTTP server process located at port 80 at that IP address.

We see from this example that DNS adds an additional delay—sometimes substantial—to the Internet applications that use it. Fortunately, as we discuss below, the desired IP address is often cached in a “nearby” DNS server, which helps to reduce DNS network traffic as well as the average DNS delay.

DNS provides a few other important services in addition to translating hostnames to IP addresses:

- Host aliasing. A host with a complicated hostname can have one or more alias names. For example, a hostname such as `relay1.west-coast.enterprise.com` could have, say, two aliases such as `enterprise.com` and `www.enterprise.com`. In this case, the hostname `relay1.west-coast.enterprise.com` is said to be a canonical hostname. Alias hostnames, when present, are typically more mnemonic than canonical hostnames. DNS can be invoked by an application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.
- Mail server aliasing. For obvious reasons, it is highly desirable that e-mail

addresses be mnemonic. For example, if Bob has an account with Hotmail, Bob's e-mail address might be as simple as bob@hotmail.com. However, the host^[STX]name of the Hotmail mail server is more complicated and much less mnemonic than simply hotmail.com (for example, the canonical hostname might be something like relay1.west-coast.hotmail.com). DNS can be invoked by a mail application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host. In fact, the MX record (see below) permits a company's mail server and Web server to have identical (aliased) hostnames; for example, a company's Web server and mail server can both be called enterprise.com.

- Load distribution. DNS is also used to perform load distribution among replicated servers, such as replicated Web servers. Busy sites, such as cnn.com, are replicated over multiple servers, with each server running on a different end system and each having a different IP address. For replicated Web servers, a set of IP addresses is thus associated with one canonical hostname. The DNS database contains this set of IP addresses. When clients make a DNS query for a name mapped to a set of addresses, the server responds with the entire set of IP addresses, but rotates the ordering of the addresses within each reply. Because a client typically sends its HTTP request message to the IP address that is listed first in the set, DNS rotation distributes the traffic among the replicated servers. DNS rotation is also used for e-mail so that multiple mail servers can have the same alias name. Also, content distribution companies such as Akamai have used DNS in more sophisticated ways [Dilley 2002] to provide Web content distribution (see Chapter 7).

The DNS is specified in RFC 1034 and RFC 1035, and updated in several additional RFCs. It is a complex system, and we only touch upon key aspects of its operation here. The interested reader is referred to these RFCs and the book by Albitz and Liu [Albitz 1993]; see also the retrospective paper [Mockapetris 1988], which provides a nice description of the what and why of DNS, and [Mockapetris 2005].

2.5.2 Overview of How DNS Works

We now present a high-level overview of how DNS works. Our discussion will focus on the hostname-to-IP-address translation service.

Suppose that some application (such as a Web browser or a mail reader) running in a user's host needs to translate a hostname to an IP address. The application will invoke the client side of DNS, specifying the hostname that needs to be translated. (On many UNIX-based machines, `gethostbyname()` is the function call that an application calls in order to perform the translation.) DNS in the user's host then takes over, sending a query message into the network. All DNS query and reply messages are sent within UDP datagrams to port 53. After a delay, ranging from milliseconds to seconds, DNS in the user's host receives a DNS reply message that provides the desired mapping. This mapping is then passed to the invoking application. Thus, from the perspective of the invoking application in the user's host, DNS is a black box providing a simple, straightforward translation service. But in fact, the black box that implements the service is complex, consisting of a large number of DNS servers distributed around the globe, as well as an application-layer protocol that specifies how the DNS servers and querying hosts communicate.

A simple design for DNS would have one DNS server that contains all the mappings. In this centralized design, clients simply direct all queries to the single DNS server, and the DNS server responds directly to the querying clients. Although the

simplicity of this design is attractive, it is inappropriate for today's Internet, with its vast (and growing) number of hosts. The problems with a centralized design include:

- A single point of failure. If the DNS server crashes, so does the entire Internet!
- Traffic volume. A single DNS server would have to handle all DNS queries (for all the HTTP requests and e-mail messages generated from hundreds of millions of hosts) Distant centralized database. A single DNS server cannot be "close to" all the querying clients. If we put the single DNS server in New York City, then all queries from Australia must travel to the other side of the globe, perhaps over slow and congested links. This can lead to significant delays.
- Maintenance. The single DNS server would have to keep records for all Internet hosts. Not only would this centralized database be huge, but it would have to be updated frequently to account for every new host.

In summary, a centralized database in a single DNS server simply doesn't scale. Consequently, the DNS is distributed by design. In fact, the DNS is a wonderful example of how a distributed database can be implemented in the Internet.

A Distributed, Hierarchical Database

In order to deal with the issue of scale, the DNS uses a large number of servers, organized in a hierarchical fashion and distributed around the world. No single DNS server has all of the mappings for all of the hosts in the Internet. Instead, the map**ings** are distributed across the DNS servers. To a first approximation, there are three classes of DNS servers—root DNS servers, top-level domain (TLD) DNS servers, and authoritative DNS servers—organized in a hierarchy as shown in Fig**ure** 2.19. To understand how these three classes of servers interact, suppose a DNS client wants to determine the IP address for the hostname www.amazon.com. To a first approximation, the following events will take place. The client first contacts one of the root servers, which returns IP addresses for TLD servers for the top-level domain com. The client then contacts one of these TLD servers, which returns the IP address of an authoritative server for amazon.com. Finally, the client contacts one of the authoritative servers for amazon.com, which returns the IP address for the hostname www.amazon.com. We'll soon examine this DNS lookup process in more detail. But let's first take a closer look at these three classes of DNS servers:

- Root DNS servers. In the Internet there are 13 root DNS servers (labeled A through M), most of which are located in North America. An October 2006 map of the root DNS servers is shown in Figure 2.20; a list of the current root DNS servers is available via [Root-servers 2012]. Although we have referred to each of the 13 root DNS servers as if it were a single server, each "server" is actually a network of replicated servers, for both security and reliability purposes. All together, there are 247 root servers as of fall 2011.
- Top-level domain (TLD) servers. These servers are responsible for top-level domains such as com, org, net, edu, and gov, and all of the country top-level domains such as uk, fr, ca, and jp. The company Verisign Global Registry Services maintains the TLD servers for the com top-level domain, and the company Educause maintains the TLD servers for the edu top-level domain. See [IANA TLD 2012] for a list of all top-level domains.
- Authoritative DNS servers. Every organization with publicly accessible hosts (such as Web servers and mail servers) on the Internet must provide publicly acces**sible** DNS records that map the names of those hosts to IP addresses. An organization's authoritative DNS server houses these DNS records. An organization can

- c.
- d.
- h.
- j.
- Cogent, Herndon, VA (5 other sites)
- U Maryland College Park, MD
- ARL Aberdeen, MD
- Verisign, Dulles VA (69 other sites)
- i. Netnod, Stockholm
(37 other sites)
- k. RIPE London
(17 other sites)
- m. WIDE Tokyo
(5 other sites)
- g. US DoD Columbus, OH
(5 other sites)

e.
f.
NASA Mt View, CA
Internet Software C.
Palo Alto, CA
(and 48 other sites)

- a.
b.
l.
- Verisign, Los Angeles CA
(5 other sites)
- USC-ISI Marina del Rey, CA
- ICANN Los Angeles, CA
(41 other sites)

Figure 2.20 [STX] DNS root servers in 2012 (name, organization, location) choose to implement its own authoritative DNS server to hold these records; alter[STX]natively, the organization can pay to have these records stored in an authoritative DNS server of some service provider. Most universities and large companies implement and maintain their own primary and secondary (backup) authoritative DNS server.

The root, TLD, and authoritative DNS servers all belong to the hierarchy of DNS servers, as shown in Figure 2.19. There is another important type of DNS server called the local DNS server. A local DNS server does not strictly belong to the hierarchy of servers but is nevertheless central to the DNS architecture. Each ISP—such as a university, an academic department, an employee's company, or a residential ISP—has a local DNS server (also called a default name server). When a host connects to an ISP, the ISP provides the host with the IP addresses of one or more of its local DNS servers (typically through DHCP, which is discussed in Chap[STX]ter 4). You can easily determine the IP address of your local DNS server by access[STX]ing network status windows in Windows or UNIX. A host's local DNS server is typically "close to" the host. For an institutional ISP, the local DNS server may be on the same LAN as the host; for a residential ISP, it is typically separated from the host by no more than a few routers. When a host makes a DNS query, the query is sent to the local DNS server, which acts a proxy, forwarding the query into the DNS server hierarchy, as we'll discuss in more detail below.

Let's take a look at a simple example. Suppose the host cis.poly.edu desires the IP address of gaia.cs.umass.edu. Also suppose that Polytechnic's local DNS server is called dns.poly.edu and that an authoritative DNS server for gaia.cs.umass.edu is called dns.umass.edu. As shown in Figure 2.21, the host cis.poly.edu first sends a DNS query message to its local DNS server, dns.poly.edu. The query message contains the hostname to be translated, namely, gaia.cs.umass.edu. The local DNS server forwards the query message to a root DNS server. The root DNS server takes note of the edu suffix and returns to the local DNS server a list of IP addresses for TLD servers responsible for edu. The local DNS server then resends the query message to one of these TLD servers. The TLD server takes note of the umass.edu suffix and responds with the IP address of the authoritative DNS server for the University of Massachusetts, namely, dns.umass.edu. Finally, the local DNS server resends the query message directly to dns.umass.edu, which responds with the IP address of gaia.cs.umass.edu. Note that in this example, in order to obtain the mapping for one hostname, eight DNS messages were sent: four query messages and four reply messages! We'll soon see how DNS caching reduces this query traffic.

Our previous example assumed that the TLD server knows the authoritative DNS server for the hostname. In general this is not always true. Instead, the TLD server may know only of an intermediate DNS server, which in turn knows the authoritative DNS server for the hostname. For example, suppose again that the University of Massachusetts has a DNS server for the university, called dns.umass.edu. Also suppose that each of the departments at the University of Massachusetts has its own DNS server, and that each departmental DNS server is authoritative for all hosts in the department. In this case, when the intermediate DNS server, dns.umass.edu, receives a query for a host with a hostname ending with cs.umass.edu, it returns to dns.poly.edu the IP address of dns.cs.umass.edu, which is authoritative for all hostnames ending with cs.umass.edu. The local DNS server dns.poly.edu then sends the query to the authoritative DNS server, which returns the desired mapping to the local DNS server, which in turn returns the mapping to the requesting host. In this case, a total of 10 DNS messages are sent!

The example shown in Figure 2.21 makes use of both recursive queries and iterative queries. The query sent from cis.poly.edu to dns.poly.edu is a recursive query, since the query asks dns.poly.edu to obtain the mapping on its behalf. But the subsequent three queries are iterative since all of the replies are directly returned to dns.poly.edu. In theory, any DNS query can be iterative or recursive. For example, Figure 2.22 shows a DNS query chain for which all of the queries are recursive. In practice, the queries typically follow the pattern in Figure 2.21: The query from the requesting host to the local DNS server is recursive, and the remaining queries are iterative.

DNS Caching

Our discussion thus far has ignored DNS caching, a critically important feature of the DNS system. In truth, DNS extensively exploits DNS caching in order to improve the delay performance and to reduce the number of DNS messages ricocheting around

138 CHAPTER 2 • APPLICATION LAYER

Requesting host

cis.poly.edu

Local DNS server TLD DNS server

dns.poly.edu

Root DNS server

1

8
5
4
2
7

Authoritative DNS server

dns.umass.edu

gaia.cs.umass.edu

6 3

Figure 2.22  Recursive queries in DNS

the Internet. The idea behind DNS caching is very simple. In a query chain, when a DNS server receives a DNS reply (containing, for example, a mapping from a host  name to an IP address), it can cache the mapping in its local memory. For example, in Figure 2.21, each time the local DNS server dns.poly.edu receives a reply from some DNS server, it can cache any of the information contained in the reply. If a hostname/IP address pair is cached in a DNS server and another query arrives to the DNS server for the same hostname, the DNS server can provide the desired IP address, even if it is not authoritative for the hostname. Because hosts and mappings between hostnames and IP addresses are by no means permanent, DNS servers discard cached information after a period of time (often set to two days).

As an example, suppose that a host apricot.poly.edu queries

dns.poly.edu for the IP address for the hostname cnn.com. Furthermore, suppose that a few hours later, another Polytechnic University host, say, kiwi.poly.fr,

also queries dns.poly.edu with the same hostname. Because of caching, the local

DNS server will be able to immediately return the IP address of cnn.com to this second requesting host without having to query any other DNS servers. A local DNS

server can also cache the IP addresses of TLD servers, thereby allowing the local DNS server to bypass the root DNS servers in a query chain (this often happens)

Perfect — this is **core CS232 exam material**. I'll explain this **directly from your textbook**, in a clean, exam-ready way. I'll structure it as:

1. **What services DNS provides (2.5.1)**
2. **How DNS works at a high level (2.5.2)**
3. **Why DNS must be distributed (centralized vs distributed)**
4. **Hierarchical architecture (root → TLD → authoritative → local)**
5. **Recursive vs Iterative queries**
6. **DNS caching (why DNS is fast in practice)**

I'll also flag **exam-favorite phrases** in **bold** so you know what to memorize.

1. DNS: Services Provided (Section 2.5.1)

- ◆ **Core Purpose of DNS**

DNS is:

A distributed database + an application-layer protocol

that translates:

hostname → IP address

Humans use:

```
www.google.com
```

Routers use:

```
142.250.72.78
```

DNS bridges that gap.

It uses:

- **UDP**
- **Port 53**
- Software: **BIND**

Main Services DNS Provides

(1) Hostname → IP Translation (MOST IMPORTANT)

Example:

```
www.someschool.edu → 128.119.40.186
```

This happens **before HTTP, SMTP, FTP can even work.**

EXAM PHRASE:

DNS adds **an extra delay before application-layer protocols run**
(but caching reduces it).

(2) Host Aliasing

One complicated real name → many simple aliases:

Canonical name:

```
relay1.west-coast.enterprise.com
```

Aliases:

enterprise.com
www.enterprise.com

DNS can return:

- The **real hostname**
- The **IP address**

✓ (3) Mail Server Aliasing (MX Records)

Email address:

```
css  
bob@hotmail.com
```

But real server:

```
relay1.west-coast.hotmail.com
```

DNS:

- Resolves **email domain → mail server**
- Uses **MX records**
- Allows **Web + Mail to share same alias**

✓ (4) Load Distribution (Very Important)

Big sites like:

```
cnn.com
```

have:

```
nginx  
Multiple IP addresses
```

DNS:

- Returns **all IPs**
- **Rotates the order**
- Client uses **first IP**

- This automatically spreads traffic across servers
 - Used by Akamai & CDNs
-

✓ 2. High-Level DNS Operation (2.5.2)

When an app needs an IP:

Example:

```
nginx
Browser → www.amazon.com
```

Steps:

1. Application calls DNS (e.g., `gethostbyname()`)
2. DNS client sends **UDP query to port 53**
3. After delay → reply arrives with IP
4. Browser now opens **TCP connection to port 80**

From the app's view:

- DNS is a black box that gives IP addresses

But internally:

- DNS is distributed, global, hierarchical
-

✓ 3. Why DNS Cannot Be Centralized

If ONE DNS server existed:

✗ Single Point of Failure

→ If it crashes → Internet breaks

✗ Huge Traffic Load

→ Every HTTP + Email needs DNS

✗ Far Away Delays

→ Australia → NYC queries would be slow

✗ Impossible Maintenance

→ Must store & update records for EVERY host

- Therefore:

DNS MUST be distributed

✓ 4. DNS Hierarchical Architecture (MOST TESTED)

There are **4 important types of DNS servers**:

✓ (1) Root DNS Servers (Top of the Hierarchy)

- Only **13 logical servers (A-M)**
- Actually ~**250 replicated worldwide**
- They **don't know IPs**
- They only say:

"Ask the .com TLD server"

✓ (2) TLD Servers (Top-Level Domain)

Handle:

```
.com, .org, .edu, .gov  
.uk, .fr, .jp, etc.
```

Example:

- **.com** → Verisign
- **.edu** → Educause

They tell you:

"Here is the **authoritative server** for amazon.com"

✓ (3) Authoritative DNS Servers (Final Answer Source)

Every organization **MUST** have one.

They store:

```
www.amazon.com → 142.250.x.x  
mail.amazon.com → 98.137.x.x
```

They give the **actual IP address**.

✓ (4) Local DNS Server (Your ISP's DNS)

Does NOT belong to hierarchy but is **critical**.

Your computer always contacts:

Local DNS server FIRST

It:

- Acts as a **proxy**
 - Talks to root, TLD, authoritative for you
 - Is usually **1-2 routers away**
-

5. Full DNS Lookup Example (Exam Gold)

Host:

```
cis.poly.edu
```

Wants IP of:

```
gaia.cs.umass.edu
```

Steps:

1. Host → Local DNS (recursive)
2. Local DNS → Root
3. Root → Local DNS → gives TLD for .edu
4. Local DNS → .edu TLD server
5. TLD → Local DNS → gives dns.umass.edu
6. Local DNS → dns.umass.edu
7. Authoritative → Local DNS → gives IP
8. Local DNS → Host → gives IP

-  **8 messages total (4 query + 4 reply)**
 If departmental DNS exists → **10 messages**
-

6. Recursive vs Iterative Queries (EXAM FAVORITE)

Recursive Query

"You complete the lookup for me."

Used by:

```
sql
```

```
Host → Local DNS
```

✓ Iterative Query

"If you don't know, tell me who to ask next."

Used by:

```
sql
Local DNS → Root → TLD → Authoritative
```

✓ Typical DNS uses:

- 1 recursive
- Rest iterative

✓ 7. DNS Caching (Why DNS is Fast)

Each DNS server caches:

```
nginx
hostname → IP
```

If another host asks for same name:

- ✓ DNS responds **immediately**
- ✓ No root/TLD/authoritative queries needed

Cached data:

- Expires after **TTL (~2 days)**

✓ Local DNS can also cache:

- TLD IPs
- Authoritative server IPs
→ This often **skips the root server entirely**

✓ FINAL EXAM SUMMARY (MEMORIZE THIS)

✓ DNS is:

- A **distributed hierarchical database**
- An **application-layer protocol**
- Uses **UDP port 53**

✓ DNS provides:

1. Hostname → IP resolution

2. Host aliasing
3. Mail server aliasing (MX)
4. Load distribution

DNS Hierarchy:

```
sql
```

Host → Local DNS → Root → TLD → Authoritative

Query types:

- Host → Local = **Recursive**
- Other queries = **Iterative**

DNS caching:

- Reduces delay
- Reduces traffic
- Allows skipping higher servers

If you want, I can now:

- Give you **2-3 exam-style questions**
- Draw the full hierarchy lookup with arrows
- Or summarize this into a **1-page cheat sheet** for your final.



5.4.1 - Link layer addressing: MAC addresses

5.4.1 Link-Layer Addressing and ARP

Hosts and routers have link-layer addresses. Now you might find this surprising, recalling from Chapter 4 that hosts and routers have network-layer addresses as well. You might be asking, why in the world do we need to have addresses at both the network and link layers? In addition to describing the syntax and function of the link-layer addresses, in this section we hope to shed some light on why the two layers

462 CHAPTER 5 • THE LINK LAYER: LINKS, ACCESS NETWORKS, AND LANS

Mail

server

To external

internet

1 Gbps

1

2 3

4

6 5

1 Gbps

1 Gbps
 Electrical Engineering Computer Science
 100 Mbps
 (fiber)
 100 Mbps
 (fiber)
 100 Mbps
 (fiber)
 Mixture of 10 Mbps,
 100 Mbps, 1 Gbps,
 Cat 5 cable
 Web
 server
 Computer Engineering

Figure 5.15 An institutional network connected together by four switches of addresses are useful and, in fact, indispensable. We'll also cover the Address Resolution Protocol (ARP), which provides a mechanism to translate IP addresses to link-layer addresses.

MAC Addresses

In truth, it is not hosts and routers that have link-layer addresses but rather their adapters (that is, network interfaces) that have link-layer addresses. A host or router with multiple network interfaces will thus have multiple link-layer addresses associated with it, just as it would also have multiple IP addresses associated with it. It's important to note, however, that link-layer switches do not have

link-layer addresses associated with their interfaces that connect to hosts and routers. This is because the job of the link-layer switch is to carry datagrams between hosts and routers; a switch does this job transparently, that is, without the host or router having to explicitly address the frame to the intervening switch.

This is illustrated in Figure 5.16. A link-layer address is variously called a LAN address, a physical address, or a MAC address. Because MAC address seems to be the most popular term, we'll henceforth refer to link-layer addresses as MAC addresses. For most LANs (including Ethernet and 802.11 wireless LANs), the MAC address is 6 bytes long, giving 248 possible MAC addresses. As shown in Figure 5.16, these 6-byte addresses are typically expressed in hexadecimal notation, with each byte of the address expressed as a pair of hexadecimal numbers.

Although MAC addresses were designed to be permanent, it is now possible to change an adapter's MAC address via software. For the rest of this section, however, we'll assume that an adapter's MAC address is fixed.

One interesting property of MAC addresses is that no two adapters have the same address. This might seem surprising given that adapters are manufactured in many countries by many companies. How does a company manufacturing adapters in Taiwan make sure that it is using different addresses from a company manufacturing adapters in Belgium? The answer is that the IEEE manages the MAC address space. In particular, when a company wants to manufacture adapters, it purchases a chunk of the address space consisting of 224 addresses for a nominal fee. IEEE allocates the chunk of 224 addresses by fixing the first 24 bits of a MAC address and letting the company create unique combinations of the last 24 bits for each adapter.

An adapter's MAC address has a flat structure (as opposed to a hierarchical structure) and doesn't change no matter where the adapter goes. A laptop with an Ethernet interface always has the same MAC address, no matter where the computer goes. A

smartphone with an 802.11 interface always has the same MAC address, no matter where the smartphone goes. Recall that, in contrast, IP addresses have a hierarchical structure (that is, a network part and a host part), and a host's IP addresses needs to be changed when the host moves, i.e., changes the network to which it is attached. An adapter's MAC address is analogous to a person's social security number, which also has a flat addressing structure and which doesn't change no matter where the person goes. An IP address is analogous to a person's postal address, which is hierarchical and which must be changed whenever a person moves. Just as a person may find it useful to have both a postal address and a social security number, it is useful for a host and router interfaces to have both a network-layer address and a MAC address.

When an adapter wants to send a frame to some destination adapter, the sending adapter inserts the destination adapter's MAC address into the frame and then

sends the frame into the LAN. As we will soon see, a switch occasionally broadcasts an incoming frame onto all of its interfaces. We'll see in Chapter 6 that

802.11 also broadcasts frames. Thus, an adapter may receive a frame that isn't addressed to it. Thus, when an adapter receives a frame, it will check to see whether the destination MAC address in the frame matches its own MAC address.

If there is a match, the adapter extracts the enclosed datagram and passes the datagram up the protocol stack. If there isn't a match, the adapter discards the frame, without passing the network-layer datagram up. Thus, the destination only will be interrupted when the frame is received.

However, sometimes a sending adapter does want all the other adapters on the LAN to receive and process the frame it is about to send. In this case, the sending adapter inserts a special MAC broadcast address into the destination address field of the frame. For LANs that use 6-byte addresses (such as Ethernet and 802.11), the broadcast address is a string of 48 consecutive 1s (that is, FF-FF-FF-FF-FF-FF in hexadecimal notation). Address Resolution Protocol (ARP)

Because there are both network-layer addresses (for example, Internet IP addresses) and link-layer addresses (that is, MAC addresses), there is a need to translate between them. For the Internet, this is the job of the Address Resolution Protocol (ARP) [RFC 826].

To understand the need for a protocol such as ARP, consider the network shown in Figure 5.17. In this simple example, each host and router has a single IP address and single MAC address. As usual, IP addresses are shown in dotted-decimal notation and MAC addresses are shown in hexadecimal notation. For the purposes of this discussion, we will assume in this section that the switch broadcasts all frames; that is, whenever a switch receives a frame on one interface, it forwards the frame on all of its other interfaces. In the next section, we will provide a more accurate explanation of how switches operate.

Great — this is another **high-yield CS232 final exam topic**. I'll explain **MAC addresses and why they matter**, then connect it cleanly to **ARP** (since your text immediately transitions there). I'll stick tightly to your textbook wording + exam logic.

I'll break this into:

1. **Why do we need BOTH IP and MAC addresses?**
2. **What exactly is a MAC address?**

3. Key properties of MAC addresses (flat, permanent, uniqueness)
 4. Who has MAC addresses (hosts, routers, switches?)
 5. How MAC addresses are used to send frames
 6. Broadcast MAC address
 7. Why ARP is needed (bridge between IP and MAC)
-

1. Why Do We Need BOTH IP and MAC Addresses?

This is **the main conceptual question** your professor expects you to answer.

We have:

- Network-layer address → IP
- Link-layer address → MAC

Why not just one?

Because:

Layer	Address Type	Purpose
Network Layer	IP Address	Used for routing across the Internet
Link Layer	MAC Address	Used for delivery within a local network (LAN)

-  IP = end-to-end delivery across multiple networks
 -  MAC = one-hop delivery inside a single LAN
-

Exam Analogy (From Your Book)

- MAC address = Social Security Number
 - Flat
 - Permanent
 - Globally unique
- IP address = Postal address
 - Hierarchical
 - Changes when you move networks

 Just like a person needs **both**, a network interface needs **both IP and MAC**.

2. What Is a MAC Address?

A **MAC address** is:

The **link-layer (physical) address of a network interface (adapter)**

Important wording:

- NOT the host
- NOT the router
- It is the ADAPTER (network interface) that has a MAC address

So:

- Laptop with:
 - Ethernet → 1 MAC
 - WiFi → 1 MAC
 - Total = **2 MAC addresses**
-

Size & Format

For Ethernet & WiFi:

- **6 bytes = 48 bits**
- Written in **hexadecimal**

Example:

```
css
AA-BB-CC-DD-EE-FF
```

Total possible addresses:

```
2^48
```

Which is **huge**.

3. Key Properties of MAC Addresses

(1) Flat Structure

- No hierarchy
- No network prefix
- Unlike IP:

```
128.119.40.17 → has network + host part
```

MAC:

```
ruby
```

```
AA:BB:CC:DD:EE:FF → no structure
```

✓ (2) Permanent (Traditionally)

Originally:

- Burned into hardware
- Never changed

Now:

- Can be spoofed by software

But for theory:

✓ Assume MAC is fixed

✓ (3) Globally Unique

No two adapters in the world have the same MAC.

How is that guaranteed?

✓ IEEE controls MAC allocation

- Company buys **first 24 bits**
- Company assigns **last 24 bits**

So:

```
csharp
```

```
[Company ID | Unique Device ID]
[24 bits    | 24 bits]
```

✓ 4. Who Has MAC Addresses?

Device	Has MAC?	Why
Host (Laptop, Phone)	✓ Yes	Has network interface
Router Interface	✓ Yes	Each interface is on a LAN
Switch Ports	✗ No	Works transparently

! Very Important:

✓ Switch interfaces DO NOT have MAC addresses

Why?

Because switches:

- Forward frames **without being explicitly addressed**
 - Operate **transparently**
-

✓ 5. How MAC Addresses Are Used to Send Frames

When a device wants to send data inside a LAN:

1. It creates a **frame**
 2. Inserts:
 - ✓ Destination **MAC**
 - ✓ Source **MAC**
 3. Sends frame into LAN
-

Receiving Side Behavior

Every adapter:

- Checks:


```
perl
Is destination MAC == my MAC?
```
- If ✓ yes → extract IP datagram → pass up to network layer
- If ✗ no → **discard silently**

✓ This ensures:

Only the intended device is interrupted.

✓ 6. MAC Broadcast Address (VERY EXAM-IMPORTANT)

Sometimes you want:

✓ **Everyone on the LAN to receive the frame**

Then use:

```
FF-FF-FF-FF-FF-FF
```

(48 ones)

This is called:

✓ MAC Broadcast Address

Used by:

- ARP
 - DHCP
 - Some routing protocols
-

✓ 7. Why Do We Need ARP?

Now we connect **IP + MAC**.

You said it perfectly:

We have **network-layer addresses (IP)**
 and **link-layer addresses (MAC)**
 → so we need a way to **translate between them**

✓ That translator is:

ARP — Address Resolution Protocol (RFC 826)

✓ The Big ARP Problem

Suppose Host A wants to send to:

128.119.40.17

It knows:

- ✓ IP address (from DNS)
 ✗ MAC address (needed to build frame)

So it asks:

? "Who has IP 128.119.40.17? Tell me your MAC."

✓ ARP Basic Operation (Conceptual)

1. Sender broadcasts ARP request:

css

Who has IP X.X.X.X?

2. All nodes receive it

3. Only the node with that IP replies:

```
ruby
```

```
My MAC is AA:BB:CC:DD:EE:FF
```

4. Sender caches the result
5. Sender now sends real data frame

This is why the book **immediately transitions from MAC → ARP**

FINAL EXAM SUMMARY (MAC + ARP)

MAC Address

- 48 bits (6 bytes)
- Hexadecimal
- Globally unique
- Flat structure
- Assigned to **network interfaces (adapters)**

IP vs MAC

- IP → routing across networks
- MAC → delivery inside LAN

Switches

- Do NOT have MAC addresses
- Operate transparently

Broadcast MAC

```
FF-FF-FF-FF-FF-FF
```

ARP

- Translates **IP → MAC**
- Uses **broadcast**
- Essential for frame delivery on LAN

If you want, next I can:

- Walk you through an **exact ARP request/reply example with real numbers**
- Or explain **why routers need MAC addresses on each interface**
- Or give you **final-exam ARP vs DNS comparison (very common confusion)**



explain only these : 4.2 - transport layer vs network layer services, datagram networks (forwarding tables, forwarding rules).

4.2 Virtual Circuit and Datagram Networks

Recall from Chapter 3 that a transport layer can offer applications connectionless service or connection-oriented service between two processes. For example, the Internet's transport layer provides each application a choice between two services: UDP, a connectionless service; or TCP, a connection-oriented service. In a similar manner, a network layer can provide connectionless service or connection service between two hosts. Network-layer connection and connectionless services in many ways parallel transport-layer connection-oriented and connectionless services. For example, a network-layer connection service begins with handshaking between the source and destination hosts; and a network-layer connectionless service does not have any handshaking preliminaries.

Although the network-layer connection and connectionless services have some parallels with transport-layer connection-oriented and connectionless services, there are crucial differences:

- In the network layer, these services are host-to-host services provided by the network layer for the transport layer. In the transport layer these services are process-to-process services provided by the transport layer for the application layer.
- In all major computer network architectures to date (Internet, ATM, frame relay, and so on), the network layer provides either a host-to-host connectionless service or a host-to-host connection service, but not both. Computer networks that provide only a connection service at the network layer are called virtual-circuit (VC) networks; computer networks that provide only a connectionless service at the network layer are called datagram networks.
- The implementations of connection-oriented service in the transport layer and the connection service in the network layer are fundamentally different. We saw in the previous chapter that the transport-layer connection-oriented service is

4.2 • VIRTUAL CIRCUIT AND DATAGRAM NETWORKS 313

implemented at the edge of the network in the end systems; we'll see shortly that the network-layer connection service is implemented in the routers in the network core as well as in the end systems.

Virtual-circuit and datagram networks are two fundamental classes of computer networks. They use very different information in making their forwarding decisions.

Let's now take a closer look at their implementations.

4.2.1 Virtual-Circuit Networks

While the Internet is a datagram network, many alternative network architectures—including those of ATM and frame relay—are virtual-circuit networks and, therefore, use connections at the network layer. These network-layer connections are called virtual circuits (VCs). Let's now consider how a VC service can be implemented in a computer network.

A VC consists of (1) a path (that is, a series of links and routers) between the source and destination hosts, (2) VC numbers, one number for each link along the path, and (3) entries in the forwarding table in each router along the path. A packet belonging to a virtual circuit will carry a VC number in its header. Because a virtual circuit may have a different VC number on each link, each intervening router must replace the VC number of each traversing packet with a new VC number. The new

VC number is obtained from the forwarding table.

To illustrate the concept, consider the network shown in Figure 4.3. The numbers next to the links of R1 in Figure 4.3 are the link interface numbers. Suppose now that Host A requests that the network establish a VC between itself and Host B. Suppose also that the network chooses the path A-R1-R2-B and assigns VC numbers 12, 22, and 32 to the three links in this path for this virtual circuit. In this case, when a packet in this VC leaves Host A, the value in the VC number field in the packet header is 12; when it leaves R1, the value is 22; and when it leaves R2, the value is 32.

How does the router determine the replacement VC number for a packet traversing the router? For a VC network, each router's forwarding table includes VC. Whenever a new VC is established across a router, an entry is added to the forwarding table. Similarly, whenever a VC terminates, the appropriate entries in each table along its path are removed.

You might be wondering why a packet doesn't just keep the same VC number on each of the links along its route. The answer is twofold. First, replacing the number from link to link reduces the length of the VC field in the packet header. Second, and more importantly, VC setup is considerably simplified by permitting a different VC number at each link along the path of the VC. Specifically, with multiple VC numbers, each link in the path can choose a VC number independently of the VC numbers chosen at other links along the path. If a common VC number were required for all links along the path, the routers would have to exchange and process a substantial number of messages to agree on a common VC number (e.g., one that is not being used by any other existing VC at these routers) to be used for a connection.

In a VC network, the network's routers must maintain connection state information for the ongoing connections. Specifically, each time a new connection is established across a router, a new connection entry must be added to the router's forwarding table; and each time a connection is released, an entry must be removed from the table. Note that even if there is no VC-number translation, it is still necessary to maintain connection state information that associates VC numbers with output interface numbers. The issue of whether or not a router maintains connection state information for each ongoing connection is a crucial one—one that we'll return to repeatedly in this book.

There are three identifiable phases in a virtual circuit:

- VC setup. During the setup phase, the sending transport layer contacts the network layer, specifies the receiver's address, and waits for the network to set up the VC. The network layer determines the path between sender and receiver, that is, the series of links and routers through which all packets of the VC will travel. The network layer also determines the VC number for each link along the path. Finally, the network layer adds an entry in the forwarding table in each router

Incoming Interface	Incoming VC #	Outgoing Interface	Outgoing VC #
1	12	2	22
2	63	1	18
3	7	2	17
1	97	3	87
...

inform the end system on the other side of the network of the call termination and update the forwarding tables in each of the packet routers on the path to indicate that the VC no longer exists.

There is a subtle but important distinction between VC setup at the network layer and connection setup at the transport layer (for example, the TCP three-way handshake we studied in Chapter 3). Connection setup at the transport layer involves only the two end systems. During transport-layer connection setup, the two end systems alone determine the parameters (for example, initial sequence number and flow-control window size) of their transport-layer connection.

Although the two end systems are aware of the transport-layer connection, the routers within the network are completely oblivious to it. On the other hand, with a VC network layer, routers along the path between the two end systems are involved in VC setup, and each router is fully aware of all the VCs passing through it.

The messages that the end systems send into the network to initiate or terminate a VC, and the messages passed between the routers to set up the VC (that is, to modify connection state in router tables) are known as signaling messages, and the protocol used to exchange these messages are often referred to as signaling protocols. VC setup is shown pictorially in Figure 4.4. We'll not cover VC signaling protocols in this book; see [Black 1997] for a general discussion of signaling in connection-oriented networks and [ITU-T Q.2931 1995] for the specification of ATM's Q.2931 signaling protocol.

4.2.2 Datagram Networks

In a datagram network, each time an end system wants to send a packet, it stamps the packet with the address of the destination end system and then pops the packet into the network. As shown in Figure 4.5, there is no VC setup and routers do not maintain any VC state information (because there are no VCs!).

As a packet is transmitted from source to destination, it passes through a series of routers. Each of these routers uses the packet's destination address to forward the packet. Specifically, each router has a forwarding table that maps destination addresses to link interfaces; when a packet arrives at the router, the router uses the packet's destination address to look up the appropriate output link interface in the forwarding table. The router then intentionally forwards the packet to that output link interface.

To get some further insight into the lookup operation, let's look at a specific example. Suppose that all destination addresses are 32 bits (which just happens to be the length of the destination address in an IP datagram). A brute-force implementation of the forwarding table would have one entry for every possible destination address. Since there are more than 4 billion possible addresses, this option is totally out of the question. Now let's further suppose that our router has four links, numbered 0 through 3,

and that packets are to be forwarded to the link interfaces as follows:

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000	through 0
11001000 00010111 00010111 11111111	
11001000 00010111 00011000 00000000	through 1
11001000 00010111 00011000 11111111	
11001000 00010111 00011001 00000000	through 2
11001000 00010111 00011111 11111111	

otherwise 3

Clearly, for this example, it is not necessary to have 4 billion entries in the router's forwarding table. We could, for example, have the following forwarding table with just four entries:

Prefix Match	Link Interface
11001000 00010111 00010 0	
11001000 00010111 00011000 1	
11001000 00010111 00011 2	

otherwise 3

With this style of forwarding table, the router matches a prefix of the packet's destination address with the entries in the table; if there's a match, the router forwards the packet to a link associated with the match. For example, suppose the packet's destination address is 11001000 00010111 00010110 10100001; because the 21-bit prefix of this address matches the first entry in the table, the router forwards the packet to link interface 0. If a prefix doesn't match any of the first three entries, then the router forwards the packet to interface 3. Although this sounds simple enough, there's an important subtlety here. You may have noticed that it is possible for a destination address to match more than one entry. For example, the first 24 bits of the address 11001000 00010111 00011000 10101010 match the second entry in the table, and the first 21 bits of the address match the third entry in the table. When there are multiple matches, the router uses the longest prefix matching rule; that is, it finds the longest matching entry in the table and forwards the packet to the link

318 CHAPTER 4 • THE NETWORK LAYER

interface associated with the longest prefix match. We'll see exactly why this longest prefix-matching rule is used when we study Internet addressing in more detail in Section 4.4.

Although routers in datagram networks maintain no connection state information, they nevertheless maintain forwarding state information in their forwarding tables. However, the time scale at which this forwarding state information changes is relatively slow. Indeed, in a datagram network the forwarding tables are modified by the routing algorithms, which typically update a forwarding table every one-to-five minutes or so. In a VC network, a forwarding table in a router is modified whenever a new connection is set up through the router or whenever an existing connection through the router is torn down. This could easily happen at a microsecond timescale in a backbone, tier-1 router.

Because forwarding tables in datagram networks can be modified at any time, a series of packets sent from one end system to another may follow different paths through the network and may arrive out of order. [Paxson 1997] and [Jaiswal 2003] present interesting measurement studies of packet reordering and other phenomena in the public Internet 4.2.3 Origins of VC and Datagram Networks

The evolution of datagram and VC networks reflects their origins. The notion of a virtual circuit as a central organizing principle has its roots in the telephony world, which uses real circuits. With call setup and per-call state being maintained at the routers within the network, a VC network is arguably more complex than a datagram network (although see [Molinero-Fernandez 2002] for an interesting comparison of the complexity of circuit- versus packet-switched networks). This, too, is in keeping with its telephony heritage. Telephone networks, by necessity, had their complexity within the network, since they were connecting dumb end-system devices such as rotary telephones. (For those too young to know, a rotary phone is an analog telephone with no buttons—only a dial.)

The Internet as a datagram network, on the other hand, grew out of the need to

connect computers together. Given more sophisticated end-system devices, the Internet architects chose to make the network-layer service model as simple as possible. As we have already seen in Chapters 2 and 3, additional functionality (for example, in-order delivery, reliable data transfer, congestion control, and DNS name resolution) is then implemented at a higher layer, in the end systems. This inverts the model of the telephone network, with some interesting consequences:

- Since the resulting Internet network-layer service model makes minimal (no!) service guarantees, it imposes minimal requirements on the network layer. This makes it easier to interconnect networks that use very different link-layer technologies (for example, satellite, Ethernet, fiber, or radio) that have very different transmission rates and loss characteristics. We will address the interconnection of IP networks in detail in Section 4.4.

4.2 • VIRTUAL CIRCUIT AND DATAGRAM NETWORKS 319

- As we saw in Chapter 2, applications such as e-mail, the Web, and even some network infrastructure services such as the DNS are implemented in hosts (servers) at the network edge. The ability to add a new service simply by attaching a host to the network and defining a new application-layer protocol (such as HTTP) has allowed new Internet applications such as the Web to be deployed in a remarkably short period of time

Got it — I'll explain **only** what you asked for, tightly aligned to the textbook and in **exam-ready form**:

✓ 1. Transport Layer vs Network Layer Services

(From Section 4.2 opening)

✓ Transport Layer Services (Chapter 3)

- **Provided between PROCESSES**
- Runs **only at the end systems**
- Two service types:
 - **Connectionless** → UDP
 - **Connection-oriented** → TCP
- Routers are **NOT involved**
- Example services:
 - Reliability
 - Flow control
 - Congestion control
 - In-order delivery

✓ Key exam phrase:

Transport-layer services are **process-to-process** and are implemented **at the edge of the network**.

✓ Network Layer Services (Chapter 4)

- **Provided between HOSTS**
- Implemented in:
 - **End systems**
 - **AND routers (core of the network)**
- Two service types:
 - **Connection-oriented** → Virtual Circuit (VC)
 - **Connectionless** → Datagram

✓ Key exam phrase:

Network-layer services are **host-to-host** and are implemented **inside the network core (routers)**.

✓ Crucial Differences (Directly from your text)

Feature	Transport Layer	Network Layer
Endpoints	Process ↔ Process	Host ↔ Host
Where implemented	End systems only	End systems + routers
Service choice	UDP or TCP	Only ONE per network
Internet provides	Both UDP & TCP	Only Datagram

✓ Important:

The Internet is a **datagram network**, NOT a virtual-circuit network.

✓ 2. Datagram Networks (Your Main Focus)

In a **datagram network** (like the Internet):

- **No connection setup**
- **No VC state in routers**
- **Every packet is treated independently**
- **Each packet carries full destination IP address**

✓ How Forwarding Works in a Datagram Network

✓ What Each Router Has:

A **forwarding table** that maps:

```
bash
```

Destination address prefix → Output **link** interface

✓ What Each Router Does:

1. Packet arrives
2. Router looks at:

```
css
```

Destination IP **address**

3. Router finds **best match in forwarding table**
4. Router sends packet to the selected **output interface**

✓ This happens at **every hop**, independently.

✓ 3. Forwarding Tables (How Routers Avoid 4 Billion Entries)

IP addresses are **32 bits**, so naïvely you'd need:

```
2^32 = 4 billion entries
```

→ This is impossible.

So routers use:

✓ **Prefix-based forwarding tables**

✓ Example from Your Text

Instead of listing every address, the router stores **ranges using prefixes**:

Prefix Match	Link
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
Otherwise	3

Router checks:

"Which prefix matches the destination IP?"

✓ 4. Longest Prefix Matching Rule (VERY EXAM-IMPORTANT)

Sometimes:

- A destination IP matches **multiple prefixes**

In that case:

Router chooses the LONGEST matching prefix

Meaning:

- The most **specific match**
- The prefix with the **most bits matched**

This is what allows:

- Subnetting
 - Efficient routing
 - Scalable Internet forwarding
-

5. Forwarding State vs Connection State

Datagram Network (Internet)

- Routers maintain:
 -  **Forwarding state** (routing tables)
- Routers do NOT maintain:
 -  Connection state
- Forwarding tables are updated:
 - Every **1-5 minutes**
 - By **routing algorithms**
- Consequence:
 -  Packets may take **different paths**
 -  Packets may arrive **out of order**

Virtual-Circuit Networks (For Comparison Only)

- Routers maintain:
 -  **Connection state per VC**
- Tables updated:
 - At **microsecond time scales**
- Packets in a VC:
 - Always follow **same path**

- Always arrive **in order**
-

✓ Final Exam Summary (What You Must Memorize)

✓ Transport Layer

- Process-to-process
- Implemented at edge
- UDP (connectionless), TCP (connection-oriented)

✓ Network Layer

- Host-to-host
- Implemented in routers + hosts
- Internet uses **datagram service only**

✓ Datagram Network

- No setup
- No per-connection state
- Each packet independently routed
- Uses **destination IP address**

✓ Forwarding Table

- Uses **prefix matching**
- Uses **longest prefix match rule**
- Maps destination → output link

✓ Datagram Network Behavior

- Routing tables change slowly (minutes)
 - Packets may take different paths
 - Packets may arrive out of order
-

If you want, I can now:

- ✓ Compare **Datagram vs Virtual Circuit in one clean exam table**, or
- ✓ Walk you through **one full packet forwarding example using prefix matching**, step by step.



So how do I know if a router is using the datagram or virtual circuit? Because I think if a router is supposed to provide internet service, then it should use datagram. Otherwise, I mean, it might use

virtual circuit network. But I mean, our home routers, shouldn't it use datagram? Then which routers use virtual circuit network? I'm not sure.

Great question — this confusion is **very common**, and your intuition is actually **almost 100% correct**. Let's cleanly separate what's *theory*, what's *real Internet*, and what's *special carrier networks*.

I'll answer in this exact order:

1. **How to know if a router is using datagram vs virtual circuit**
 2. **What YOUR home router uses**
 3. **Who actually uses virtual circuits**
 4. **What about MPLS? (modern confusion point)**
 5. **One-sentence exam answer**
-

1. How Do You Know If a Router Uses Datagram or Virtual Circuit?

You don't decide this **router-by-router**.

You decide this based on:

- What type of NETWORK the router belongs to.**

If the network is:

- **Internet (IP-based) → DATAGRAM**
- **ATM / Frame Relay (telecom networks) → VIRTUAL CIRCUIT**

So the key rule is:

- Routers don't "choose" VC vs datagram. The NETWORK ARCHITECTURE determines it.**
-

2. What Does Your HOME Router Use?

Your home router:

- Uses **IP**
- Connects to:
 - Your ISP
 - The Internet
- Forwards packets using:
 - Destination IP
 - Forwarding tables

- ✓ Longest prefix match
- Has:
 - ✗ NO per-connection state
 - ✗ NO virtual circuit numbers
 - ✗ NO setup/teardown

✓ Therefore:

- ✓ Your home router is a **DATAGRAM** router.
- ✓ All Internet routers are **DATAGRAM** routers.

There is **no such thing as a “virtual-circuit Internet router”** in the public Internet.

✓ 3. Then Who Uses Virtual Circuit Networks?

Virtual circuits come from **TELECOMMUNICATION (carrier) networks**, not the open Internet.

✓ Classic Virtual-Circuit Networks:

Technology	Uses Virtual Circuits?
ATM	✓ YES
Frame Relay	✓ YES
X.25 (old)	✓ YES
PSTN (telephone network)	✓ YES (real circuits)

These networks:

- Do **connection setup**
- Maintain **per-flow state in routers**
- Everything follows the **same reserved path**

These are used by:

- Phone companies
- Backbone carriers
- Private enterprise WANs (historically)

✓ These are **NOT what your home connects to directly** with IP routing.

✓ 4. Important Modern Confusion: MPLS (Feels Like VC, But Isn't IP-VC)

Modern ISPs use something called:

✓ MPLS = Multi-Protocol Label Switching

MPLS:

- Looks like a **virtual circuit**
- Uses:
 - Labels instead of IP
 - Predefined paths
- BUT:
 - It runs **UNDER IP**
 - It is used **INSIDE ISP backbones**
 - End users **STILL** see normal **datagram IP networking**

So even when MPLS is present:

- ✓ Your traffic is still **datagram at the network layer**
- ✓ MPLS is a **traffic-engineering trick**, not a public VC network

For your exam:

✗ You do NOT need to treat MPLS as a virtual-circuit network.

✓ 5. Final One-Sentence Exam Answer (MEMORIZE THIS)

✓ All routers in the public Internet are datagram routers. Virtual-circuit routers exist only in special carrier networks such as ATM and Frame Relay, not in home or ISP Internet routing.

✓ Ultra-Clear Decision Rule

If you see this in a question:

If the problem mentions...	Then the network is...
IP, Internet, packets, forwarding tables, prefix match	✓ DATAGRAM
VC setup, VC numbers, teardown, per-connection state	✓ VIRTUAL CIRCUIT
TCP handshake	✗ Transport layer only
ATM, Frame Relay	✓ Virtual Circuit
Home WiFi router	✓ Datagram
ISP router	✓ Datagram

✓ Why Your Earlier Thought Was Actually Right

You said:

"If a router is supposed to provide internet service, then it should use datagram."

This is exactly correct.

That's how the Internet was designed.

If you want, next I can:

- Give you a **Datagram vs Virtual Circuit comparison table for memorization**, or
- Walk you through a **real packet path from your laptop → ISP → Google using datagram forwarding**, or
- Give you **2 final-exam trick questions based on this topic**.



explain these - 4.3 - Routers: architecture and components, input processing, switching, output processing, queueing.

4.3 What's Inside a Router?

Now that we've overviewed the network layer's services and functions, let's turn our attention to its forwarding function—the actual transfer of packets from a router's incoming links to the appropriate outgoing links at that router. We already took a brief look at a few aspects of forwarding in Section 4.2, namely, addressing and longest prefix matching. We mention here in passing that the terms forwarding and switching are often used interchangeably by computer-networking researchers and practitioners; we'll use both terms interchangeably in this textbook as well.

A high-level view of a generic router architecture is shown in Figure 4.6. Four router components can be identified:

- Input ports. An input port performs several key functions. It performs the physical layer function of terminating an incoming physical link at a router; this is shown in the leftmost box of the input port and the rightmost box of the output port in Figure 4.6. An input port also performs link-layer functions needed to interoperate with the link layer at the other side of the incoming link; this is represented by the middle boxes in the input and output ports. Perhaps most crucially, the lookup function is also performed at the input port; this will occur in the rightmost box of the input port. It is here that the forwarding table is consulted to determine the router output port to which an arriving packet will be forwarded via the switching fabric. Control packets (for example, packets carrying routing protocol information) are forwarded from an input port to the routing processor. Note that the term port here—referring to the physical input and output router interfaces—is distinctly different from the software ports associated with network applications and sockets discussed in Chapters 2 and 3.
- Switching fabric. The switching fabric connects the router's input ports to its output ports. This switching fabric is completely contained within the router—a network inside of a network router!
- Output ports. An output port stores packets received from the switching fabric and transmits these packets on the outgoing link by performing the necessary

link-layer and physical-layer functions. When a link is bidirectional (that is,

320 CHAPTER 4 • THE NETWORK LAYER

carries traffic in both directions), an output port will typically be paired with the input port for that link on the same line card (a printed circuit board containing one or more input ports, which is connected to the switching fabric).

- Routing processor. The routing processor executes the routing protocols (which we'll study in Section 4.6), maintains routing tables and attached link state information, and computes the forwarding table for the router. It also performs the network management functions that we'll study in Chapter 9.

Recall that in Section 4.1.1 we distinguished between a router's forwarding and routing functions. A router's input ports, output ports, and switching fabric together implement the forwarding function and are almost always implemented in hardware, as shown in Figure 4.6. These forwarding functions are sometimes collectively referred to as the router forwarding plane. To appreciate why a hardware implementation is needed, consider that with a 10 Gbps input link and a 64-byte IP datagram, the input port has only 51.2 ns to process the datagram before another datagram may arrive. If N ports are combined on a line card (as is often done in practice), the datagram-processing pipeline must operate N times faster—far too fast for software implementation. Forwarding plane hardware can be implemented either using a router vendor's own hardware designs, or constructed using purchased merchant-silicon chips (e.g., as sold by companies such as Intel and Broadcom).

While the forwarding plane operates at the nanosecond time scale, a router's control functions—executing the routing protocols, responding to attached links that

4.3 • WHAT'S INSIDE A ROUTER? 321

Input port Output port

Input port Output port

Routing

processor Routing, management

control plane (software)

Forwarding

data plane (hardware)

Switch

fabric

Figure 4.6 Router architecture

go up or down, and performing management functions such as those we'll study in Chapter 9—operate at the millisecond or second timescale. These router control plane functions are usually implemented in software and execute on the routing processor (typically a traditional CPU).

Before delving into the details of a router's control and data plane, let's return to our analogy of Section 4.1.1, where packet forwarding was compared to cars entering and leaving an interchange. Let's suppose that the interchange is a roundabout, and that before a car enters the roundabout, a bit of processing is required—the car stops at an entry station and indicates its final destination (not at the local roundabout, but the ultimate destination of its journey). An attendant at the entry station looks up the final destination, determines the roundabout exit that leads to that final destination, and tells the driver which roundabout exit to take. The car enters the roundabout (which may be filled with other cars entering from other input roads and heading to other roundabout exits) and eventually leaves at the prescribed roundabout exit ramp, where it may encounter other cars leaving the roundabout at that exit.

We can recognize the principal router components in Figure 4.6 in this analogy—the entry road

and entry station correspond to the input port (with a lookup function to determine to local outgoing port); the roundabout corresponds to the switch fabric; and the roundabout exit road corresponds to the output port. With this analogy, it's instructive to consider where bottlenecks might occur. What happens if cars arrive blazingly fast (for example, the roundabout is in Germany or Italy!) but the station attendant is slow? How fast must the attendant work to ensure there's no backup on an entry road? Even with a blazingly fast attendant, what happens if cars traverse the roundabout slowly—can backups still occur? And what happens if most of the entering cars all want to leave the roundabout at the same exit ramp—can backups occur at the exit ramp or elsewhere? How should the roundabout operate if we want to assign priorities to different cars, or block certain cars from entering the roundabout in the first place? These are all analogous to critical questions faced by router and switch designers.

In the following subsections, we'll look at router functions in more detail. [Iyer 2008, Chao 2001; Chuang 2005; Turner 1988; McKeown 1997a; Partridge 1998] provide a discussion of specific router architectures. For concreteness, the ensuing discussion assumes a datagram network in which forwarding decisions are based on the packet's destination address (rather than a VC number in a virtual-circuit network). However, the concepts and techniques are quite similar for a virtual-circuit network.

4.3.1 Input Processing

A more detailed view of input processing is given in Figure 4.7. As discussed above, the input port's line termination function and link-layer processing implement the physical and link layers for that individual input link. The lookup performed in the input port is central to the router's operation—it is here that the router uses the forwarding table to look up the output port to which an arriving packet will be forwarded via the switching fabric. The forwarding table is computed and updated by the routing processor, with a shadow copy typically stored at each input port. The forwarding table is copied from the routing processor to the line cards over a separate bus (e.g., a PCI bus) indicated by the dashed line from the routing processor to the input line cards in Figure 4.6. With a shadow copy, forwarding decisions can be made locally, at each input port, without invoking the centralized routing processor on a per-packet basis and thus avoiding a centralized processing bottleneck. Given the existence of a forwarding table, lookup is conceptually simple—we just search through the forwarding table looking for the longest prefix match, as described

4.3 • WHAT'S INSIDE A ROUTER? 323

Line

termination

Data link

processing

(protocol,

decapsulation)

Lookup, forwarding,

queuing

Switch

fabric

Figure 4.7 Input port processing

CISCO SYSTEMS: DOMINATING THE NETWORK CORE

As of this writing 2012, Cisco employs more than 65,000 people. How did this gorilla of a networking company come to be? It all started in 1984 in the living room of a Silicon Valley apartment.

Len Bosak and his wife Sandy Lerner were working at Stanford University when they had the idea to build and sell Internet routers to research and academic institutions, the

primary adopters of the Internet at that time. Sandy Lerner came up with the name Cisco (an abbreviation for San Francisco), and she also designed the company's bridge logo. Corporate headquarters was their living room, and they financed the project with credit cards and moonlighting consulting jobs. At the end of 1986, Cisco's revenues reached \$250,000 a month. At the end of 1987, Cisco succeeded in attracting venture capital—\$2 million from Sequoia Capital in exchange for one-third of the company. Over the next few years, Cisco continued to grow and grab more and more market share. At the same time, relations between Bosak/Lerner and Cisco management became strained. Cisco went public in 1990; in the same year Lerner and Bosak left the company.

Over the years, Cisco has expanded well beyond the router market, selling security, wireless caching, Ethernet switch, datacenter infrastructure, video conferencing, and voice-over IP products and services. However, Cisco is facing increased international competition, including from Huawei, a rapidly growing Chinese network-gear company. Other sources of competition for Cisco in the router and switched Ethernet space include Alcatel-Lucent and Juniper.

CASE HISTORY

in Section 4.2.2. But at Gigabit transmission rates, this lookup must be performed in nanoseconds (recall our earlier example of a 10 Gbps link and a 64-byte IP datagram). Thus, not only must lookup be performed in hardware, but techniques beyond a simple linear search through a large table are needed; surveys of fast lookup algorithms can be found in [Gupta 2001, Ruiz-Sanchez 2001]. Special attention must also be paid to memory access times, resulting in designs with embedded on-chip DRAM and faster SRAM (used as a DRAM cache) memories. Ternary Content Address Memories (TCAMs) are also often used for lookup. With a TCAM, a 32-bit IP address is presented to the memory, which returns the content of the forwarding table entry for that address in essentially constant time. The Cisco 8500 has a 64K CAM for each input port.

Once a packet's output port has been determined via the lookup, the packet can be sent into the switching fabric. In some designs, a packet may be temporarily blocked from entering the switching fabric if packets from other input ports are currently using the fabric. A blocked packet will be queued at the input port and then scheduled to cross the fabric at a later point in time. We'll take a closer look at the blocking, queuing, and scheduling of packets (at both input ports and output ports) in Section 4.3.4. Although "lookup" is arguably the most important action in input port processing, many other actions must be taken: (1) physical- and link-layer processing must occur, as discussed above; (2) the packet's version number, checksum and time-to-live field—all of which we'll study in Section 4.4.1—must be checked and the latter two fields rewritten; and (3) counters used for network management (such as the number of IP datagrams received) must be updated.

Let's close our discussion of input port processing by noting that the input port steps of looking up an IP address ("match") then sending the packet into the switching fabric ("action") is a specific case of a more general "match plus action" abstraction that is performed in many networked devices, not just routers. In link-layer switches (covered in Chapter 5), link-layer destination addresses are looked up and several actions may be taken in addition to sending the frame into the switching fabric towards the output port. In firewalls (covered in Chapter 8)—devices that filter out selected incoming packets—an incoming packet whose header matches a given criteria (e.g., a combination of source/destination IP addresses and transport-layer port numbers) may be prevented from being forwarded (action). In a network address translator (NAT, covered in Section 4.4), an incoming packet whose transport-layer port number matches a given value will have its port number rewritten before forwarding (action). Thus, the "match plus action" abstraction is both powerful and prevalent in network devices.

4.3.2 Switching

The switching fabric is at the very heart of a router, as it is through this fabric that the packets are actually switched (that is, forwarded) from an input port to an output port. Switching can be accomplished in a number of ways, as shown in Figure 4.8:

- Switching via memory. The simplest, earliest routers were traditional computers, with switching between input and output ports being done under direct control of

324 CHAPTER 4 • THE NETWORK LAYER

the CPU (routing processor). Input and output ports functioned as traditional I/O devices in a traditional operating system. An input port with an arriving packet first signaled the routing processor via an interrupt. The packet was then copied from the input port into processor memory. The routing processor then extracted the destination address from the header, looked up the appropriate output port in the forwarding table, and copied the packet to the output port's buffers. In this scenario, if the memory bandwidth is such that B packets per second can be written into, or read from, memory, then the overall forwarding throughput (the total rate at which packets are transferred from input ports to output ports) must be less than $B/2$. Note also that two packets cannot be forwarded at the same time, even if they have different destination ports, since only one memory read/write over the shared system bus can be done at a time.

Many modern routers switch via memory. A major difference from early routers, however, is that the lookup of the destination address and the storing of the packet into the appropriate memory location are performed by processing on the input line cards. In some ways, routers that switch via memory look very much like shared-memory multiprocessors, with the processing on a line card switching (writing) packets into the memory of the appropriate output port. Cisco's Catalyst 8500 series switches [Cisco 8500 2012] forward packets via a shared memory.

4.3 • WHAT'S INSIDE A ROUTER? 325

Memory

A

B

C

X

Y

Z

Memory

Key:

Input port Output port

A

XYZ

B

C

Crossbar

A

B

C

X

Y

Z

Bus

Figure 4.8 Three switching techniques

- Switching via a bus. In this approach, an input port transfers a packet directly to the

output port over a shared bus, without intervention by the routing processor. This is typically done by having the input port pre-pend a switch-internal label (header) to the packet indicating the local output port to which this packet is being transferred and transmitting the packet onto the bus. The packet is received by all output ports, but only the port that matches the label will keep the packet. The label is then removed at the output port, as this label is only used within the switch to cross the bus. If multiple packets arrive to the router at the same time, each at a different input port, all but one must wait since only one packet can cross the bus at a time. Because every packet must cross the single bus, the switching speed of the router is limited to the bus speed; in our roundabout analogy, this is as if the roundabout could only contain one car at a time. Nonetheless, switching via a bus is often sufficient for routers that operate in small local area and enterprise networks. The Cisco 5600 [Cisco Switches 2012] switches packets over a 32 Gbps backplane bus.

- Switching via an interconnection network. One way to overcome the bandwidth limitation of a single, shared bus is to use a more sophisticated interconnection network, such as those that have been used in the past to interconnect processors in a multiprocessor computer architecture. A crossbar switch is an interconnection network consisting of $2N$ buses that connect N input ports to N output ports, as shown in Figure 4.8. Each vertical bus intersects each horizontal bus at a crosspoint, which can be opened or closed at any time by the switch fabric controller (whose logic is part of the switching fabric itself). When a packet arrives from port A and needs to be forwarded to port Y, the switch controller closes the crosspoint at the intersection of busses A and Y, and port A then sends the packet onto its bus, which is picked up (only) by bus Y. Note that a packet from port B can be forwarded to port X at the same time, since the A-to-Y and B-to-X packets use different input and output busses. Thus, unlike the previous two switching approaches, crossbar networks are capable of forwarding multiple packets in parallel. However, if two packets from two different input ports are destined to the same output port, then one will have to wait at the input, since only one packet can be sent over any given bus at a time. More sophisticated interconnection networks use multiple stages of switching elements to allow packets from different input ports to proceed towards the same output port at the same time through the switching fabric. See [Tobagi 1990] for a survey of switch architectures. Cisco 12000 family switches [Cisco 12000 2012] use an interconnection network.

4.3.3 Output Processing

Output port processing, shown in Figure 4.9, takes packets that have been stored in the output port's memory and transmits them over the output link. This includes selecting and de-queueing packets for transmission, and performing the needed link layer and physical-layer transmission functions 4.3.4 Where Does Queueing Occur? If we consider input and output port functionality and the configurations shown in Figure 4.8, it's clear that packet queues may form at both the input ports and the output ports, just as we identified cases where cars may wait at the inputs and outputs of the traffic intersection in our roundabout analogy. The location and extent of queueing (either at the input port queues or the output port queues) will depend on the traffic load, the relative speed of the switching fabric, and the line speed. Let's now consider these queues in a bit more detail, since as these queues grow large, the router's memory can eventually be exhausted and packet loss will occur when no memory is available to store arriving packets. Recall that in our earlier discussions, we said that packets were "lost within the network" or "dropped at a router." It is here, at these queues within a router, where such packets are actually dropped and lost. Suppose that the input and output line speeds (transmission rates) all have an

identical transmission rate of Rline packets per second, and that there are N input ports and N output ports. To further simplify the discussion, let's assume that all packets have the same fixed length, and the packets arrive to input ports in a synchronous manner. That is, the time to send a packet on any link is equal to the time to receive a packet on any link, and during such an interval of time, either zero or one packet can arrive on an input link. Define the switching fabric transfer rate Rswitch as the rate at which packets can be moved from input port to output port. If Rswitch is N times faster than Rline, then only negligible queuing will occur at the input ports. This is because even in the worst case, where all N input lines are receiving packets, and all packets are to be forwarded to the same output port, each batch of N packets (one packet per input port) can be cleared through the switch fabric before the next batch arrives.

But what can happen at the output ports? Let's suppose that Rswitch is still N times faster than Rline. Once again, packets arriving at each of the N input ports are destined to the same output port. In this case, in the time it takes to send a single packet onto the outgoing link, N new packets will arrive at this output port. Since the output port can transmit only a single packet in a unit of time (the packet transmission time), the N arriving packets will have to queue (wait) for transmission over the outgoing link. Then N more packets can possibly arrive in the time it takes to

4.3 • WHAT'S INSIDE A ROUTER? 327

Line

termination

Data link

processing

(protocol,

encapsulation)

Queuing (buffer

Switch management)

fabric

Figure 4.9 Output port processing

transmit just one of the N packets that had just previously been queued. And so on.

Eventually, the number of queued packets can grow large enough to exhaust available memory at the output port, in which case packets are dropped.

Output port queuing is illustrated in Figure 4.10. At time t, a packet has arrived at each of the incoming input ports, each destined for the uppermost outgoing port. Assuming identical line speeds and a switch operating at three times the line speed, one time unit later (that is, in the time needed to receive or send a packet), all three original packets have been transferred to the outgoing port and are queued awaiting transmission. In the next time unit, one of these three packets will have been transmitted over the outgoing link. In our example, two new packets have arrived at the incoming side of the switch; one of these packets is destined for this uppermost output port.

Given that router buffers are needed to absorb the fluctuations in traffic load, the natural question to ask is how much buffering is required. For many years, the rule of thumb [RFC 3439] for buffer sizing was that the amount of buffering (B) should be equal to an average round-trip time (RTT, say 250 msec) times the link capacity (C).

This result is based on an analysis of the queueing dynamics of a relatively small number of TCP flows [Villamizar 1994]. Thus, a 10 Gbps link with an RTT of 250 msec

would need an amount of buffering equal to $B = RTT \cdot C = 2.5$ Gbits of buffers. Recent

328 CHAPTER 4 • THE NETWORK LAYER

Switch

fabric

Output port contention at time t

One packet time later

Switch

fabric

Figure 4.10 [STX] Output port queuing

theoretical and experimental efforts [Appenzeller 2004], however, suggest that when there are a large number of TCP flows (N) passing through a link, the amount of buffering needed is $B = RTT \cdot C / \sqrt{N}$

—. With a large number of flows typically passing through large backbone router links (see, e.g., [Fraleigh 2003]), the value of N can be large, with the decrease in needed buffer size becoming quite significant. [Appenzellar 2004; Wischik 2005; Beheshti 2008] provide very readable discussions of the buffer sizing problem from a theoretical, implementation, and operational standpoint.

A consequence of output port queuing is that a packet scheduler at the output port must choose one packet among those queued for transmission. This selection might be done on a simple basis, such as first-come-first-served (FCFS) scheduling, or a more sophisticated scheduling discipline such as weighted fair queuing (WFQ), which shares the outgoing link fairly among the different end-to-end connections that have packets queued for transmission. Packet scheduling plays a crucial role in providing quality-of-service guarantees. We'll thus cover packet scheduling extensively in Chapter 7. A discussion of output port packet scheduling disciplines is [Cisco Queue 2012].

Similarly, if there is not enough memory to buffer an incoming packet, a decision must be made to either drop the arriving packet (a policy known as drop-tail) or remove one or more already-queued packets to make room for the newly arrived packet. In some cases, it may be advantageous to drop (or mark the header of) a packet before the buffer is full in order to provide a congestion signal to the sender. A number of packet-dropping and -marking policies (which collectively have become known as active queue management (AQM) algorithms) have been proposed and analyzed [Labrador 1999, Hollot 2002]. One of the most widely studied and implemented AQM algorithms is the Random Early Detection (RED) algorithm. Under RED, a weighted average is maintained for the length of the output queue. If the average queue length is less than a minimum threshold, minth, when a packet arrives, the packet is admitted to the queue. Conversely, if the queue is full or the average queue length is greater than a maximum threshold, maxth, when a packet arrives, the packet is marked or dropped. Finally, if the packet arrives to find an average queue length in the interval [minth, maxth], the packet is marked or dropped with a probability that is typically some function of the average queue length, minth, and maxth. A number of probabilistic marking/dropping functions have been proposed, and various versions of RED have been analytically modeled, simulated, and/or implemented. [Christiansen 2001] and [Floyd 2012] provide overviews and pointers to additional reading.

4.3 • WHAT'S INSIDE A ROUTER? 329

If the switch fabric is not fast enough (relative to the input line speeds) to transfer all arriving packets through the fabric without delay, then packet queuing can also occur at the input ports, as packets must join input port queues to wait their turn to be transferred through the switching fabric to the output port. To illustrate an important consequence of this queuing, consider a crossbar switching fabric and suppose that (1) all link speeds are identical, (2) that one packet can be transferred from any one input port to a given output port in the same amount of time it takes for a packet to be received on an input link, and (3) packets are moved from a given input queue to their desired output queue in an FCFS manner. Multiple packets can be transferred in parallel, as long

as their output ports are different. However, if two packets at the front of two input queues are destined for the same output queue, then one of the packets will be blocked and must wait at the input queue—the switching fabric can transfer only one packet to a given output port at a time.

Figure 4.11 shows an example in which two packets (darkly shaded) at the front of their input queues are destined for the same upper-right output port. Suppose that the switch fabric chooses to transfer the packet from the front of the upper-left queue. In this case, the darkly shaded packet in the lower-left queue must wait. But not only must this darkly shaded packet wait, so too must the lightly shaded packet that is queued behind that packet in the lower-left queue, even though there is no contention for the middle-right output port (the destination for the lightly shaded packet). This phenomenon is known as head-of-the-line (HOL) blocking in an input-queued switch—a queued packet in an input queue must wait for transfer through the fabric (even though its output port is free) because it is blocked by another packet at the head of the line. [Karol 1987] shows that due to HOL blocking, the input queue will grow to unbounded length (informally, this is equivalent to saying that significant packet loss will occur) under certain assumptions as soon as the packet arrival rate on the input links reaches only 58 percent of their capacity. A number of solutions to HOL blocking are discussed in [McKeown 1997b].

Nice, this is a big but very mechanical chunk of Chapter 4. I'll walk through exactly what you asked:

1. Router **architecture & components**
2. **Input processing**
3. **Switching (switch fabric types)**
4. **Output processing**
5. **Queueing (where and why)**

I'll keep everything aligned with the text but in "exam brain" language.

1 Router Architecture & Components (Figure 4.6)

Think of a router as having **two planes**:

- **Forwarding (data) plane – hardware, nanoseconds**
- **Control plane – software, milliseconds/seconds**

The main components:

- ◆ (a) **Input Ports**

Each input port:

- **Terminates the physical link** (physical layer)
- **Runs the link-layer protocol** for that link (Ethernet, etc.)
- **Does lookup**: uses the **forwarding table** to decide:

"To which output port should I send this packet?"

Input ports are where **most of the fast work happens**.

◆ (b) Switching Fabric

- Internal "network inside the router"
 - Connects **input ports** → **output ports**
 - Does the actual **switching/forwarding** of packets between them.
-

◆ (c) Output Ports

Each output port:

- **Buffers packets** coming from the switching fabric
 - Runs link-layer + physical-layer to **transmit onto the outgoing link**
 - May have **queues** if many packets are waiting to be sent.
-

◆ (d) Routing Processor

- Runs **routing protocols** (RIP, OSPF, BGP)
 - Builds and updates the **routing table**
 - From routing table, generates the **forwarding table** used by input ports
 - Handles **network management** tasks
-

⌚ Time-scale Difference

- **Forwarding plane (input + fabric + output)** → hardware → **nanoseconds**
- **Control plane (routing processor)** → software → **milliseconds/seconds**

Exam phrase:

Input ports, switching fabric, and output ports implement **forwarding** (hardware); the routing processor implements **routing & control** (software).

2 Input Processing (Figure 4.7)

When a packet arrives at an input port, several things happen:

1. Line termination

- Physical-layer stuff: receiving bits, decoding signal.

2. Data link processing

- Receive the **frame** from the link (e.g., Ethernet)
- Error checking, decapsulation to get the **IP packet**.

3. Lookup & forwarding decision (most important)

- Use **destination IP address** to:
 - Perform **longest-prefix match** in the forwarding table
 - Decide **which output port** to send to.
- Typically:
 - A **shadow copy** of the forwarding table is stored at each input port.
 - So input port can look up locally without bothering the routing processor.

4. Other per-packet operations

- Check and update:
 - **TTL (time-to-live)** field
 - **Header checksum**
- Update **counters** for management (e.g., # packets received).

5. Possible queuing at input

- If the switching fabric is busy, the packet might **have to wait** in an **input queue** until it can cross the fabric.

“Match + Action” Abstraction

Input processing is an example of a general pattern used in many devices:

- **Match**: use some header fields (IP dest, ports, etc.)
- **Action**: forward, drop, modify, etc.

Used in:

- Routers (forwarding)
- Switches (forward based on MAC)
- Firewalls (drop/block matching packets)
- NAT (rewrite headers)

3 Switching (Switch Fabric Types) – Figure 4.8

The **switching fabric** is how packets move **from input ports to output ports**. Three main designs:

◆ (1) Switching via Memory

Old-school, simple:

- Router acts like a **normal computer**:

1. Packet arrives at input port
2. Input port interrupts **CPU**
3. CPU:
 - Copies packet into **memory**
 - Does lookup
 - Copies packet to **output port buffer**
- Limitations:
 - Every packet must be **read and written through memory**
 - If memory can process **B packets/sec**, forwarding is at most $\approx B/2$
 - Only **one packet at a time** → bottleneck.

Modern versions still use shared memory but with **line card processors** doing the lookup.

♦ (2) Switching via a Bus

- All ports share a **single internal bus**
- Input port:
 1. Prepends an **internal label** (which output port)
 2. Broadcasts the packet onto the bus
- All output ports see it; only the **right one keeps it** (strips label)

Limitations:

- Only **one packet can be on the bus at a time**
 - Switch speed is limited by **bus speed**
 - Okay for **small / enterprise** routers, not huge backbone.
-

♦ (3) Switching via an Interconnection Network (Crossbar)

- Use a fabric like a **crossbar switch**, with:
 - N input ports
 - N output ports
 - $2N$ internal buses
- Each input can be connected to **any output**, with crosspoints controlled by the switch controller.
- Allows **multiple packets in parallel**:
 - Example: A → Y and B → X simultaneously.

Limitations:

- Still: only **one packet per output port** at a time.
- If two inputs want the **same output**, one waits.

Used in **high-performance routers** (backbone).

4 Output Processing (Figure 4.9)

Once a packet reaches the output port:

1. Queueing / buffering

- Packets from the fabric are stored in the **output queue**.

2. Scheduling

- Decide **which packet to send next** on the link:
 - Simple: **FIFO / FCFS**
 - Advanced: **WFQ**, priority queues (for QoS, fairness, etc.)

3. Data link encapsulation + physical

- Add link-layer header/trailer
- Send bits on physical medium.

So output processing = **select packet + prepare + transmit**.

5 Queueing: Where and Why?

♦ Where can queues form?

Queues can form at:

1. **Output ports** (most common case)
2. **Input ports** (if switch fabric is too slow or contention issues like HOL blocking)

Why queueing?

Because:

Packets can arrive **faster** than the router can forward/transmit them.

♦ Output Queueing (Figure 4.10)

Assume:

- N input ports
- All sending traffic to **the same output port**
- Switch fabric is fast enough to move packets quickly to the output port.

Problem:

- Output link can send at rate **R_{line}**
- But up to **N packets** may arrive in the time we can send only **1**.

- So a **queue builds up at the output port**.

If the buffer fills up:

- Router has to **drop packets** → this is what we mean by:

“Packet dropped in the network” or “congestion”.

Dropping policies:

- **Drop-tail**: drop arriving packets when buffer is full
- **RED / AQM**: proactively drop/mark packets early to signal congestion.

♦ **Input Queueing & HOL (Head-of-the-Line) Blocking (Figure 4.11)**

If the **switch fabric is not fast enough**, packets can line up at **input ports**:

- Even if the output link is free, a packet may be **stuck behind another packet** in the input queue that is waiting for a busy output → **HOL blocking**.

Example:

- Two input queues:
 - Front packets both want the **same output port** → only one can go this time.
 - A packet behind them in the queue might want a **different, free output**, but must wait, because it's behind the first packet.
- Result:
 - Input queues can **grow large**
 - Performance can be bad even at **~58% utilization** (theoretical result).

♦ **Buffer Sizing (just what you need conceptually)**

Key idea:

- Buffers are needed to **absorb burstiness** of traffic.
- Old rule of thumb:

$$B = RTT \times C$$

(Buffer size = round-trip time × link capacity)

- Newer work:

With many flows, need much **less buffering**.

You don't need details for the final; just know:

Buffers exist at router ports to handle traffic bursts; if too small → more drops; if too large → can cause large delays.

Quick Exam-Style Summary

- Router has **input ports, switching fabric, output ports, routing processor**.
- **Forwarding plane (hardware)**: input → fabric → output, very fast.
- **Control plane (software)**: routing protocols, forwarding table, slower.
- **Input processing**: physical/link-layer work + longest-prefix lookup + possible queue.
- **Switch fabric**:
 - Via memory
 - Via bus
 - Via interconnection network (crossbar)
- **Output processing**: buffering, scheduling, link-layer + physical sending.
- **Queueing**:
 - At **output** (most common: many inputs → one output)
 - At **input** (if fabric is slow; can cause **HOL blocking**)
 - Queue overflow → **packet drops** → congestion.

If you want, next I can give you a **1-page cheatsheet** that compares:

- Memory / Bus / Crossbar switching
- Input vs output queueing
- HOL blocking explanation in 2 lines for exam recall.



explain 4.3.4 Where Does Queueing Occur? in very details

Love this section—this is where “packets get dropped in the network” actually becomes real. Let’s go slowly and in depth through **4.3.4 Where Does Queueing Occur?**

I’ll break it into:

1. Why queues appear in routers at all
2. Output queueing: how and why packets pile up at outputs
3. How much buffering is “needed” (old rule vs modern view)
4. Input queueing: when the switch fabric is the bottleneck
5. Head-of-the-line (HOL) blocking and why it’s nasty

Why Do Routers Need Queues At All?

Very simple principle:

Packets can arrive faster than they can be sent out.

Whenever **incoming rate > outgoing capacity** (even temporarily), packets must:

- **Wait in memory** (buffer/queue), or
- Be **dropped** if memory is full.

In a router, this “waiting room” is the **queue**, and it can appear:

- At **output ports**
- At **input ports**
depending on where the bottleneck is.

So 4.3.4 is basically answering:

“Given the architecture (input ports, switch fabric, output ports), *where* do packets pile up and *why*?“

2 Output Queueing (Most Intuitive Case)

Assume this setup:

- Every input and output link sends at the same rate:
 - **R_line packets/sec** (line speed)
- There are **N input ports** and **N output ports**
- The switch fabric can move packets from inputs to outputs at rate:
 - **R_switch packets/sec**

Now suppose:

- The switch fabric is **fast enough**:
 - $R_{\text{switch}} = N \cdot R_{\text{line}}$
 - So it can move **N packets per line-time** (one per input) to outputs.

And also suppose:

- At some point in time, **every input port** gets **1 packet** that is **destined for the same output port**.

This is worst-case contention for a single output.

Step-by-step what happens (Figure 4.10 scenario):

1 At time t:

- N packets arrive at N input ports
- All are destined to **one output port** (say, the top output)

Because the switch is fast enough ($R_{switch} = N \cdot R_{line}$):

- In one packet transmission time, all N packets get transferred through the fabric to the output port's queue.

So at the output port, **N packets are now queued**.

⌚ Next time unit (one packet transmission time later):

- The output link can only transmit **1 packet** (since line sends at R_{line})
- So:
 - 1 packet leaves the queue (transmitted)
 - **$N - 1$** remain in the queue
- Meanwhile, **new packets may arrive** from the input ports (some of them maybe to this same output port again).

If this pattern continues, output queue length can grow like:

- After 1 time unit: N packets queued
- After 2 units: roughly $2N - 1$ (if more keep arriving)
- Eventually: **buffer fills up**

When the buffer is full:

✓ New arriving packets must be dropped.

This is exactly what we mean when we say:

"Packets are lost/dropped within the network."

It's literally:

- Arrive at router
- **No space in output queue**
- Router discards them

3 How Much Buffering Do We Need?

We know we **need buffers** to absorb bursts, but **how big** should they be?

Historically, there was a simple rule of thumb:

📌 Old Rule: $B = RTT \times C$

- B = buffer size (in bits)
- RTT = average round-trip time on the path (seconds)
- C = link capacity (bits per second)

So if:

- $C = 10 \text{ Gbps}$
- $\text{RTT} = 0.25 \text{ sec}$

Then:

- $B = 0.25 \times 10^{10} = 2.5 \times 10^9 \text{ bits} = \mathbf{2.5 \text{ Gbits of buffering.}}$

Intuition:

- With TCP flows, when packets are dropped, TCP slows down and then speeds back up.
- Buffers help absorb the temporary bursts during this increase.
- The rule $B = \text{RTT} \cdot C$ comes from analysis of a small number of TCP flows sharing a link.

📌 **Newer View:** $B = \frac{\text{RTT} \cdot C}{\sqrt{N}}$

More recent research showed:

- If there are **many flows (N)** sharing the link,
- They “desynchronize” in time — not all burst at once.
- So you actually need **much less buffering.**

New suggested formula:

- $B = \text{RTT} \cdot C / \sqrt{N}$

Where:

- N = number of **independent TCP flows** sharing the link.

For large backbone routers:

- N is often **very large** (100s or 1000s)
- So \sqrt{N} is big
- So **buffer size can be much smaller** than $\text{RTT} \cdot C$.

You don't have to derive these for the exam—just know:

Buffers are needed to absorb bursts. Old rule said big buffers ($\text{RTT} \cdot C$); newer work says with many flows we can use much smaller buffers.

4 Output Queueing → Need Scheduling and Dropping Policies

Because multiple packets may be waiting at an output:

- The router needs a **scheduler** to choose which packet to send next:

- Simple: **First-Come-First-Served (FCFS)**
- Fancy: **Weighted Fair Queueing (WFQ)**, priorities, etc.
- Scheduling is important for:
 - **Quality of Service (QoS)**
 - **Fairness** among flows

Also:

- If the output buffer is full:
 - Router must **decide whether to drop the new packet or some existing one.**

Two main families:

1. Drop-tail:

- The simplest policy:
 - If buffer is full → **drop incoming packet**
- This is what many basic routers do by default.

2. Active Queue Management (AQM):

- Start **dropping or marking** packets **before** queue is totally full
- Goal: give **early congestion signals** to TCP senders.
- Example: **RED (Random Early Detection)**

♦ RED (just high level)

- Keep a **weighted average queue length**
- If average < min_th → **accept all packets**
- If average > max_th → **drop/mark all packets**
- If between min_th and max_th → **drop with some probability**

This helps:

- Avoid huge queue build-up (which causes big delays)
- Smoothly signal congestion to TCP so it backs off.

5 Input Queueing and Head-of-the-Line (HOL) Blocking (Figure 4.11)

So far we focused on **output queues**.

But queueing can also happen at **input ports**, especially if:

- The **switch fabric is not fast enough**, or
- There is **contention for outputs**.

◆ Setup for HOL Blocking Example

Assume:

1. All link speeds are **identical**.
2. It takes **one time unit** to:
 - Receive 1 packet on an input link
 - Transmit 1 packet on an output link
 - Switch 1 packet across the fabric (per output).
3. The router is **input-queued**:
 - Each input port has a **queue of packets** waiting to go through the switch.
4. Switching fabric can transfer **multiple packets in parallel**, *as long as they have different output ports*.

Now consider this situation (like Figure 4.11):

- At the **front** of two input queues are **two packets**, both destined to the **same output port** (say top-right output).
- Behind one of them (the dark packet) is another (light) packet destined to **a different output port**.

In that time unit:

- The switch fabric can only send **one packet to that output port**.
- It chooses one of the two (say the top-left input).

What happens?

- The packet at the head of the **lower-left input queue** cannot go (because that output is busy).
- But because queue is FIFO:
 - The **second packet (light-colored)** behind it also **cannot move**, even though *its desired output port is free*.

This is:

Head-of-the-Line (HOL) blocking

A packet at the front of the input queue blocks packets behind it, even if those could go to free outputs.

Result:

- Packets that *could* be forwarded (to free outputs) are **stuck behind others**.
- Overall throughput is reduced.

In fact, theory shows:

- With simple input-queued switches and FIFO queues:

- Due to HOL blocking, queues become unstable ("unbounded") when average load is **only about 58%** of link capacity.

Interpretation:

Even before you hit 100% utilization, HOL blocking can cause serious queue build-up and packet loss.

Various more advanced architectures (e.g., virtual output queues, better scheduling) are designed to **reduce HOL blocking**, but that's beyond what you need for the exam.

Big Picture Summary of 4.3.4

1. **Queues form where there is contention / mismatch** between arrival rate and service rate.

2. At **output ports**:

- If many inputs send to same output → output queues build up.
- Buffers absorb bursts; when full → packet loss.
- Scheduling decides which queued packet is sent next.

3. **Buffer sizing**:

- Old: $B = RTT \cdot C$
- New: $B = RTT \cdot C/\sqrt{N}$ when many flows.

4. At **input ports**:

- If switch fabric isn't super-fast or if packets compete for same output → input queues form.
- HOL blocking: a packet at front of queue can block others behind it, even if they want different outputs.
- This can cause poor throughput and long queues even at moderate load.

If you want, next I can:

- Turn this into a **1-page LaTeX summary sheet** you can plug into your notes, or
- Draw an ASCII time-step example showing **exact queue lengths over time** at input and output under different R_switch vs R_line ratios.

Sure — here is **an extremely clear, detailed, exam-friendly explanation of ONLY Section 4.3.4 ("Where Does Queueing Occur?")**.

I will rewrite it in a way that is *super intuitive* while keeping the technical accuracy.

4.3.4 — Where Does Queueing Occur? (Very Detailed Explanation)

Routers move packets:

Input port → Switching fabric → Output port → Link

Packets can pile up (queue) at the **input ports**, at the **output ports**, or both.

Queueing happens whenever:

Packets arrive faster than they can be moved forward

(into the switching fabric or onto the output link).

Let's analyze each part carefully.

■ 1. Basic Setup Used in the Book

To understand where queues form, we assume:

- All input links have rate **R_line** (packets per second)
- All output links have rate **R_line**
- The switch fabric moves packets at rate **R_switch**
- We consider **N input ports** and **N output ports**

The goal is to examine:

- When input ports overflow
 - When output ports overflow
 - Why packets get dropped
 - Why HOL (Head-of-the-Line) blocking happens
-

■ 2. Output Queueing (MOST COMMON case)

Output queueing happens when **many inputs send packets to the same output port**.

★ Example Scenario:

- N input ports
- Each receives **1 packet** in the same time slot
- **All packets are destined for the same output port**

Suppose switching fabric is fast enough:

If $R_{switch} = N \times R_{line}$

→ the switch can move N packets in the time it takes to receive 1 packet.

What happens?

At time **t**:

- N new packets arrive (one per input)

- Fabric moves all N to the output port's buffer

But...

! Output link can only send 1 packet per time slot

Why?

Because its transmission rate = R_{line} (one packet per time slot).

Therefore:

- At time t: N packets moved to output queue
- At time $t + 1$: only 1 can be transmitted
- In that same time, **N more could arrive**

So queue length grows:

```
mathematica

After 1 time unit:  N packets
After 2 time units:  N + (N - 1) ≈ 2N
After 3 time units:  ≈ 3N
...
Eventually: buffer fills → packet drops
```

📌 This is where packet loss happens in the Internet.

This is exactly Figure 4.10 in the textbook.

■ 3. How Much Buffering Do We Need?

Buffers exist to absorb bursts.

The traditional rule:



◆ Old Buffer Rule

Buffer size $B = RTT \times C$

- RTT = round trip time (e.g., 250 ms)
- C = link capacity (e.g., 10 Gbps)

Thus:

- $B = 0.25 \times 10,000,000,000 \text{ bits}$
- **B = 2.5 Gbits of buffering**

This came from early studies with a **small number of TCP flows**.

✓ Larger buffers absorb burstiness

- ✓ Prevent packet drops
 - ✓ Keep high throughput
-



◆ New Modern Rule (for large backbone networks)

If many TCP flows share the link (N flows):

$$\text{Buffer} = (\text{RTT} \times C) / \sqrt{N}$$

When N is large (hundreds or thousands), buffering needs drop dramatically.

Example:

- N = 10,000 flows
- $\sqrt{N} = 100$
- So buffer can be **100x smaller**

📌 This is very important in modern routers where full RTT·C buffering is impractical.

■ 4. Input Queueing (when switch fabric is the bottleneck)

Queueing can also happen **before the switching fabric**, at input ports.

This occurs when:

- $R_{\text{switch}} < N \times R_{\text{line}}$
(fabric isn't fast enough to move all packets instantly)
- Multiple packets want to use the **same output port** in the same time slot

So input queues form.

■ 5. Head-of-the-Line (HOL) Blocking — KEY CONCEPT

This happens in **input-queued switches**.

Let's visualize Figure 4.11.

Consider:

Input Port A queue:

```
csharp
[Packet X (dest: Output 1)]
[Packet Y (dest: Output 2)]
```

Input Port B queue:

```
csharp
```

```
[Packet Z (dest: Output 1)]
```

At this time slot:

- Two packets at the front of two queues want the **same output** (Output 1)
- Only ONE packet can go to that output

Suppose Packet X goes first.

Problem:

Packet Z must wait — this is fine.

But bigger problem:

! HOL Blocking:

Packet Y cannot move either, even though its output port (Output 2) is free.

Why?

Because input queues are **FIFO**.

So:

- Packet at the front blocks packets behind it
- Even if those packets could go to free outputs

❗ This can severely reduce throughput

Mathematically, Karol (1987) showed:

Even with perfectly uniform traffic, input queues saturate when arrival rate exceeds **58%** of link capacity.

This is a stunning result —

- HOL blocking prevents using the full switch bandwidth
- Even if switching fabric is fast

6. When Packet Loss Occurs

Packet loss happens when:

◆ At output queues:

- Buffer is full → new packets cannot be stored → **drop-tail** or AQM (RED)

◆ At input queues:

- If input buffers are finite → overflow → packets dropped
 - Inside the switching fabric:
 - Rare, but possible if fabric can't accept new packets fast enough
-

7. Active Queue Management (AQM)

To prevent full queues, routers can:

- **Mark** packets early (ECN)
- **Drop** packets probabilistically (RED)

This signals TCP senders to slow down early to avoid buffer overflow.

FINAL SUMMARY YOU CAN MEMORIZE

Where do queues occur?

- **Output ports** (most common)
- **Input ports** (when switch fabric is slow)
- **Anywhere packets temporarily wait due to contention**

Why do they occur?

- Multiple packets competing for same output
- Switching fabric bottleneck
- Traffic burstiness

Key Phenomena

- **Output queue buildup** → packet loss
 - **HOL blocking** → reduces throughput to ~58%
 - **Buffers need sizing** → Old: $RTT \cdot C$, New: $RTT \cdot C / \sqrt{N}$
-

If you want, I can make:

- A diagram summary
- A one-page exam cheat sheet
- A multiple-choice test for this exact section
- A very short 5-line version for fast revision

Just tell me!



explain this part

4.4.3 Internet Control Message Protocol (ICMP)

Recall that the network layer of the Internet has three main components: the IP protocol, discussed in the previous section; the Internet routing protocols (including RIP, OSPF, and BGP), which are covered in Section 4.6; and ICMP, which is the subject of this section.

ICMP, specified in [RFC 792], is used by hosts and routers to communicate network-layer information to each other. The most typical use of ICMP is for error reporting. For example, when running a Telnet, FTP, or HTTP session, you may have encountered an error message such as "Destination network unreachable." This message had its origins in ICMP. At some point, an IP router was unable to find a path to the host specified in your Telnet, FTP, or HTTP application. That router created and sent a type-3 ICMP message to your host indicating the error.

ICMP is often considered part of IP but architecturally it lies just above IP, as ICMP messages are carried inside IP datagrams. That is, ICMP messages are carried as IP payload, just as TCP or UDP segments are carried as IP payload. Similarly, when a host receives an IP datagram with ICMP specified as the upper-layer protocol, it demultiplexes the datagram's contents to ICMP, just as it would demultiplex a datagram's content to TCP or UDP.

ICMP messages have a type and a code field, and contain the header and the first 8 bytes of the IP datagram that caused the ICMP message to be generated in the first place (so that the sender can determine the datagram that caused the error). Selected ICMP message types are shown in Figure 4.23. Note that ICMP messages are used not only for signaling error conditions.

The well-known ping program sends an ICMP type 8 code 0 message to the specified host. The destination host, seeing the echo request, sends back a type 0 code 0 ICMP echo reply. Most TCP/IP implementations support the ping server directly in the operating system; that is, the server is not a process. Chapter 11 of [Stevens 1990] provides the source code for the ping client program. Note that the client program needs to be able to instruct the operating system to generate an ICMP message of type 8 code 0.

Another interesting ICMP message is the source quench message. This message is seldom used in practice. Its original purpose was to perform congestion control—to allow a congested router to send an ICMP source quench message to a host to force that host to reduce its transmission rate. We have seen in Chapter 3 that TCP has its own congestion-control mechanism that operates at the transport layer, without the use of network-layer feedback such as the ICMP source quench message.

In Chapter 1 we introduced the Traceroute program, which allows us to trace a route from a host to any other host in the world. Interestingly, Traceroute is implemented with ICMP messages. To determine the names and addresses of the routers between source and destination, Traceroute in the source sends a series of ordinary IP datagrams to the destination. Each of these datagrams carries a UDP segment with an unlikely UDP port number. The first of these datagrams has a TTL of 1, the second of 2, the third of 3, and so on. The source also starts timers for each of the datagrams. When the nth datagram arrives at the nth router, the nth router observes that the TTL of the datagram has just expired. According to the rules of the IP protocol, the

router discards the datagram and sends an ICMP warning message to the source (type 11 code 0). This warning message includes the name of the router and its IP address. When this ICMP message arrives back at the source, the source obtains the round-trip time from the timer and the name and IP address of the nth router from the ICMP message.

How does a Traceroute source know when to stop sending UDP segments?

Recall that the source increments the TTL field for each datagram it sends. Thus, one of the datagrams will eventually make it all the way to the destination host. Because this datagram contains a UDP segment with an unlikely port number, the destination host sends a port unreachable ICMP message (type 3 code 3) back to the source. When the source host receives this particular ICMP message, it knows it does not need to send additional probe packets. (The standard Traceroute program actually sends sets of three packets with the same TTL; thus the Traceroute output provides three results for each TTL.)

354 CHAPTER 4 • THE NETWORK LAYER

ICMP Type Code Description

0 0 echo reply (to ping)

3 0 destination network unreachable

3 1 destination host unreachable

3 2 destination protocol unreachable

3 3 destination port unreachable

3 6 destination network unknown

3 7 destination host unknown

4 0 source quench (congestion control)

8 0 echo request

9 0 router advertisement

10 0 router discovery

11 0 TTL expired

12 0 IP header bad

Figure 4.23  ICMP message types

In this manner, the source host learns the number and the identities of routers that lie between it and the destination host and the round-trip time between the two hosts. Note that the Traceroute client program must be able to instruct the operating system to generate UDP datagrams with specific TTL values and must also be able to be notified by its operating system when ICMP messages arrive. Now that you understand how Traceroute works, you may want to go back and play with it some more

Perfect — this is a **high-yield IP control concept** and your excerpt contains everything you need for exams. I'll explain **only this ICMP section (4.4.3)** in a structured, very clear way.

I'll organize it as:

1. **What ICMP is and where it lives**
2. **What ICMP is used for**
3. **ICMP message format (type + code + IP header)**
4. **Ping and ICMP (echo request/reply)**
5. **Source quench (why it exists and why it's mostly obsolete)**
6. **Traceroute and ICMP (most important applied use)**

7. ICMP message types summary

✓ 1. What Is ICMP and Where Does It Live?

The Internet network layer has **three major components**:

1. **IP** → moves packets
2. **Routing protocols** (RIP, OSPF, BGP) → build routes
3. **ICMP** → reports network-layer problems and diagnostics ✓

📌 Key Definition:

ICMP (Internet Control Message Protocol) is used by **hosts and routers** to communicate **network-layer control and error information**.

Even though ICMP is often *informally* called “part of IP”:

✓ Architecturally:

- ICMP sits **just above IP**
- ICMP messages are **carried inside IP datagrams**
- Just like:
 - TCP segments are inside IP
 - UDP segments are inside IP
 - ✓ ICMP messages are also inside IP

So:

ICMP = IP payload carrying control information

✓ 2. What Is ICMP Mainly Used For?

◆ Primary Purpose: Error Reporting

Example error messages you've seen:

- “Destination network unreachable”
- “Destination host unreachable”
- “Port unreachable”
- “TTL expired”

All of these come from **ICMP**.

◆ Who generates ICMP?

- **Routers** → when they can't forward packets properly
- **Hosts** → when services/ports are unreachable

So ICMP is how the network layer says:

"Something went wrong with this IP packet."

✓ 3. ICMP Message Format (Very Important)

Every ICMP message contains:

✓ (1) Type Field

Tells the **general category** of the message
(e.g., error, echo, TTL expired)

✓ (2) Code Field

Gives **specific details** within that type
(e.g., which kind of unreachable)

✓ (3) The Original IP Header + First 8 Bytes of Payload

This is **critical**:

It allows the **sender to identify exactly which packet caused the error**.

So the sender can match:

- "Oh, this error refers to *this* specific TCP/UDP/IP packet."
-

✓ 4. Ping and ICMP (Echo Request / Reply)

The **ping program** uses ICMP directly.

◆ Step-by-step:

1. Sender sends:

- **ICMP Type 8, Code 0** → *Echo Request*

2. Destination replies with:

- **ICMP Type 0, Code 0** → *Echo Reply*

3. Sender measures:

- **Round-trip time (RTT)**

◆ Important Implementation Detail:

- Ping is usually implemented **inside the operating system**

- It is **not a normal user application process**
 - The OS must allow:
 - Generation of ICMP type-8 messages
 - Reception of ICMP replies
-

✓ 5. Source Quench (ICMP Type 4) — Why It Exists but Is Rarely Used

ICMP Source Quench:

- **Type 4, Code 0**
- Originally intended for **congestion control**
- ◆ **Original Idea:**

If a router becomes congested, it sends:

"Slow down!" → ICMP source quench → to sender

- ◆ **Why It's Rarely Used Today:**

Because:

- **TCP already has its own congestion control**
- TCP reacts to:
 - Packet loss
 - Timeouts
 - Duplicate ACKs
- No need for routers to explicitly tell senders to slow down

So today:

✓ ICMP source quench is **mostly obsolete in practice**

✓ 6. Traceroute and ICMP (THIS IS EXAM GOLD)

This is the most important **applied use of ICMP**.

- ◆ **Goal of Traceroute:**

To discover:

- The **routers on the path**
- The **round-trip time (RTT) to each router**

How Traceroute Works (Step by Step)

Traceroute sends **ordinary IP packets**, but cleverly manipulates the **TTL field**.

Step 1 — TTL = 1

- Source sends a UDP packet with **TTL = 1**
- First router receives it:
 - TTL decreases from 1 → 0
 - TTL expired → router must:
 - Discard the packet**
 - Send back **ICMP Type 11, Code 0 (TTL expired)**

That ICMP reply contains:

- Router's **name**
- Router's **IP address**

Source now knows:

"Router #1 on the path is X, RTT = Y ms"

Step 2 — TTL = 2

- Source sends next UDP packet with **TTL = 2**
- First router: TTL 2 → 1 → forwards
- Second router: TTL 1 → 0 → drops
- Second router sends:
 - ICMP TTL expired**

Now source learns:

"Router #2 is Z"

Step 3 — TTL = 3, 4, 5, ...

This continues until a packet finally reaches the **destination host**.

How Does Traceroute Know It Reached the Destination?

Traceroute uses a **deliberately invalid UDP port number**.

When the destination host finally receives the packet:

- The UDP port doesn't exist

- So destination sends back:
- ICMP Type 3, Code 3 → Port unreachable**

This is Traceroute's **stop signal**.

So:

When the source receives **ICMP port unreachable**, it knows:

"We reached the destination — stop probing."

7. ICMP Message Types You Must Recognize

From your table (Figure 4.23):

Type	Code	Meaning
0	0	Echo reply (ping response)
3	0	Destination network unreachable
3	1	Destination host unreachable
3	2	Destination protocol unreachable
3	3	Destination port unreachable <input checked="" type="checkbox"/> (Traceroute stop signal)
3	6	Destination network unknown
3	7	Destination host unknown
4	0	Source quench (congestion — obsolete)
8	0	Echo request (ping)
11	0	TTL expired (Traceroute core message)
12	0	IP header bad

Final High-Yield Summary (Memorize This)

- **ICMP is a network-layer control protocol used for error reporting and diagnostics.**
- ICMP messages are **carried inside IP datagrams**.
- Every ICMP message has:
 - Type
 - Code
 - Original IP header + 8 bytes of payload
- Ping uses **ICMP Echo Request (Type 8) and Echo Reply (Type 0)**.
- Traceroute uses **ICMP TTL Expired (Type 11) and Port Unreachable (Type 3, Code 3)**.
- **Source quench (Type 4)** was for congestion control but is mostly obsolete.

If you'd like, I can now:

- Give you a **1-minute ICMP revision version**, or
- Create a **Traceroute TTL → ICMP message mapping chart**, or
- Give you **final-exam style ICMP questions** with answers.



explain these things from following texts

4.5 - graph model, shortest path, global vs decentralized routing algorithms, Link State protocol, Distance

Vector, comparison LS vs DV, Hierarchical routing.

4.5 Routing Algorithms

So far in this chapter, we've mostly explored the network layer's forwarding function. We learned that when a packet arrives to a router, the router indexes a forwarding table and determines the link interface to which the packet is to be directed. We

also learned that routing algorithms, operating in network routers, exchange and

4.5 • ROUTING ALGORITHMS 363

compute the information that is used to configure these forwarding tables. The interplay between routing algorithms and forwarding tables was shown in Figure 4.2.

Having explored forwarding in some depth we now turn our attention to the other major topic of this chapter, namely, the network layer's critical routing function.

Whether the network layer provides a datagram service (in which case different packets between a given source-destination pair may take different routes) or a VC service (in which case all packets between a given source and destination will take the same path), the network layer must nonetheless determine the path that packets take from senders to receivers. We'll see that the job of routing is to determine good paths (equivalently, routes), from senders to receivers, through the network of routers.

Typically a host is attached directly to one router, the default router for the host (also called the first-hop router for the host). Whenever a host sends a packet, the packet is transferred to its default router. We refer to the default router of the source host as the source router and the default router of the destination host as the destination router. The problem of routing a packet from source host to destination host clearly boils down to the problem of routing the packet from source router to destination router, which is the focus of this section.

The purpose of a routing algorithm is then simple: given a set of routers, with links connecting the routers, a routing algorithm finds a "good" path from source router to destination router. Typically, a good path is one that has the least cost. We'll see, however, that in practice, real-world concerns such as policy issues (for example, a rule such as "router x, belonging to organization Y, should not forward any packets originating from the network owned by organization Z") also come into play to complicate the conceptually simple and elegant algorithms whose theory underlies the practice of routing in today's networks.

A graph is used to formulate routing problems. Recall that a graph $G = (N, E)$ is a set N of nodes and a collection E of edges, where each edge is a pair of nodes from N . In the context of network-layer routing, the nodes in the graph represent routers—the points at which packet-forwarding decisions are made—and the edges connecting these nodes represent the physical links between these routers. Such a graph abstraction of a computer network is shown in Figure 4.27. To view some

graphs representing real network maps, see [Dodge 2012, Cheswick 2000]; for a discussion of how well different graph-based models model the Internet, see [Zegura 1997, Faloutsos 1999, Li 2004].

As shown in Figure 4.27, an edge also has a value representing its cost. Typically, an edge's cost may reflect the physical length of the corresponding link (for example, a transoceanic link might have a higher cost than a short-haul terrestrial link), the link speed, or the monetary cost associated with a link. For our purposes, we'll simply take the edge costs as a given and won't worry about how they are determined. For any edge (x,y) in E , we denote $c(x,y)$ as the cost of the edge between nodes x and y . If the pair (x,y) does not belong to E , we set $c(x,y) = \infty$. Also, throughout we consider only undirected graphs (i.e., graphs whose edges do not have a direction), so that edge (x,y) is the same as edge (y,x) and that $c(x,y) = c(y,x)$. Also, a node y is said to be a neighbor of node x if (x,y) belongs to E .

364 CHAPTER 4 • THE NETWORK LAYER

Given that costs are assigned to the various edges in the graph abstraction, a natural goal of a routing algorithm is to identify the least costly paths between sources and destinations. To make this problem more precise, recall that a path in a graph $G = (N,E)$ is a sequence of nodes (x_1, x_2, \dots, x_p) such that each of the pairs $(x_1, x_2), (x_2, x_3), \dots, (x_{p-1}, x_p)$ are edges in E . The cost of a path (x_1, x_2, \dots, x_p) is simply the sum of all the edge costs along the path, that is, $c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$. Given any

two nodes x and y , there are typically many paths between the two nodes, with each path having a cost. One or more of these paths is a least-cost path. The least-cost problem is therefore clear: Find a path between the source and destination that has least cost. In Figure 4.27, for example, the least-cost path between source node u and destination node w is (u, x, y, w) with a path cost of 3. Note that if all edges in the graph have the same cost, the least-cost path is also the shortest path (that is, the path with the smallest number of links between the source and the destination).

As a simple exercise, try finding the least-cost path from node u to z in Figure 4.27 and reflect for a moment on how you calculated that path. If you are like most people, you found the path from u to z by examining Figure 4.27, tracing a few routes from u to z , and somehow convincing yourself that the path you had chosen had the least cost among all possible paths. (Did you check all of the 17 possible paths between u and z ? Probably not!) Such a calculation is an example of a centralized routing algorithm—the routing algorithm was run in one location, your brain, with complete information about the network. Broadly, one way in which we can classify routing algorithms is according to whether they are global or decentralized.

- A global routing algorithm computes the least-cost path between a source and destination using complete, global knowledge about the network. That is, the algorithm takes the connectivity between all nodes and all link costs as inputs. This then requires that the algorithm somehow obtain this information before actually performing the calculation. The calculation itself can be run at one site

4.5 • ROUTING ALGORITHMS 365

x y
v
3

5
2 5
2 3 1
1 2
1
u z
w

Figure 4.27 Abstract graph model of a computer network

(a centralized global routing algorithm) or replicated at multiple sites. The key distinguishing feature here, however, is that a global algorithm has complete information about connectivity and link costs. In practice, algorithms with global state information are often referred to as link-state (LS) algorithms, since the algorithm must be aware of the cost of each link in the network. We'll study LS algorithms in Section 4.5.1.

- In a decentralized routing algorithm, the calculation of the least-cost path is carried out in an iterative, distributed manner. No node has complete information about the costs of all network links. Instead, each node begins with only the knowledge of the costs of its own directly attached links. Then, through an iterative process of calculation and exchange of information with its neighboring nodes (that is, nodes that are at the other end of links to which it itself is attached), a node gradually calculates the least-cost path to a destination or set of destinations. The decentralized routing algorithm we'll study below in Section 4.5.2 is called a distance-vector (DV) algorithm, because each node maintains a vector of estimates of the costs (distances) to all other nodes in the network.

A second broad way to classify routing algorithms is according to whether they are static or dynamic. In static routing algorithms, routes change very slowly over time, often as a result of human intervention (for example, a human manually editing a router's forwarding table). Dynamic routing algorithms change the routing paths as the network traffic loads or topology change. A dynamic algorithm can be run either periodically or in direct response to topology or link cost changes. While dynamic algorithms are more responsive to network changes, they are also more susceptible to problems such as routing loops and oscillation in routes.

A third way to classify routing algorithms is according to whether they are load-sensitive or load-insensitive. In a load-sensitive algorithm, link costs vary dynamically to reflect the current level of congestion in the underlying link. If a high cost is associated with a link that is currently congested, a routing algorithm will tend to choose routes around such a congested link. While early ARPAnet routing algorithms were load-sensitive [McQuillan 1980], a number of difficulties were encountered [Huitema 1998]. Today's Internet routing algorithms (such as RIP, OSPF, and BGP) are load-insensitive, as a link's cost does not explicitly reflect its current (or recent past) level of congestion.

4.5.1 The Link-State (LS) Routing Algorithm

Recall that in a link-state algorithm, the network topology and all link costs are known, that is, available as input to the LS algorithm. In practice this is accomplished by having each node broadcast link-state packets to all other nodes in the network, with each link-state packet containing the identities and costs of its attached links. In practice (for example, with the Internet's OSPF routing protocol, discussed in Section 4.6.1) this is often accomplished by a link-state broadcast

366 CHAPTER 4 • THE NETWORK LAYER

algorithm [Perlman 1999]. We'll cover broadcast algorithms in Section 4.7. The result of the nodes' broadcast is that all nodes have an identical and complete view

of the network. Each node can then run the LS algorithm and compute the same set of least-cost paths as every other node.

The link-state routing algorithm we present below is known as Dijkstra's algorithm, named after its inventor. A closely related algorithm is Prim's algorithm; see

[Cormen 2001] for a general discussion of graph algorithms. Dijkstra's algorithm computes the least-cost path from one node (the source, which we will refer to as u) to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the k th iteration of the algorithm, the least-cost paths are known to k destination nodes, and among the least-cost paths to all destination nodes, these k paths will have the k smallest costs. Let us define the following notation:

- $D(v)$: cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.
- $p(v)$: previous node (neighbor of v) along the current least-cost path from the source to v .
- N' : subset of nodes; v is in N' if the least-cost path from the source to v is definitely known.

The global routing algorithm consists of an initialization step followed by a loop. The number of times the loop is executed is equal to the number of nodes in the network. Upon termination, the algorithm will have calculated the shortest paths from the source node u to every other node in the network.

Link-State (LS) Algorithm for Source Node u

1 Initialization:

2 $N' = \{u\}$

3 for all nodes v

4 if v is a neighbor of u

5 then $D(v) = c(u,v)$

6 else $D(v) = \infty$

7

8 Loop

9 find w not in N' such that $D(w)$ is a minimum

10 add w to N'

11 update $D(v)$ for each neighbor v of w and not in N' :

12 $D(v) = \min(D(v), D(w) + c(w,v))$

13 /* new cost to v is either old cost to v or known

14 least path cost to w plus cost from w to v */

15 until $N' = N$

4.5 • ROUTING ALGORITHMS 367

As an example, let's consider the network in Figure 4.27 and compute the least-cost paths from u to all possible destinations. A tabular summary of the algorithm's computation is shown in Table 4.3, where each line in the table gives the values of the algorithm's variables at the end of the iteration. Let's consider the few first steps in detail.

- In the initialization step, the currently known least-cost paths from u to its directly attached neighbors, v , x , and w , are initialized to 2, 1, and 5, respectively. Note in particular that the cost to w is set to 5 (even though we will soon see that a lesser-cost path does indeed exist) since this is the cost of the direct (one hop) link from u to w . The costs to y and z are set to infinity because they are not directly connected to u .
- In the first iteration, we look among those nodes not yet added to the set N' and find that node with the least cost as of the end of the previous iteration. That node is x , with a cost of 1, and thus x is added to the set N' . Line 12 of the LS algorithm is then

performed to update $D(v)$ for all nodes v , yielding the results shown in the second line (Step 1) in Table 4.3. The cost of the path to v is unchanged. The cost of the path to w (which was 5 at the end of the initialization) through node x is found to have a cost of 4. Hence this lower-cost path is selected and w 's predecessor along the shortest path from u is set to x . Similarly, the cost to y (through x) is computed to be 2, and the table is updated accordingly.

- In the second iteration, nodes v and y are found to have the least-cost paths (2), and we break the tie arbitrarily and add y to the set N_{EDT} so that N_{EDT} now contains u , x , and y . The cost to the remaining nodes not yet in N_{EDT} , that is, nodes v , w , and z , are updated via line 12 of the LS algorithm, yielding the results shown in the third row in the Table 4.3.

- And so on. . . .

When the LS algorithm terminates, we have, for each node, its predecessor along the least-cost path from the source node. For each predecessor, we also

368 CHAPTER 4 • THE NETWORK LAYER

step N' $D(v), p(v)$ $D(w), p(w)$ $D(x), p(x)$ $D(y), p(y)$ $D(z), p(z)$

0 u 2, u 5, u 1, u ∞ ∞

1 ux 2, u 4, x 2, x ∞

2 uxy 2, u 3, y 4, y

3 $uxyv$ 3, y 4, y

4 $uxyvw$ 4, y

5 $uxyvwz$

Table 4.3 Running the link-state algorithm on the network in Figure 4.27

VideoNote

Dijkstra's algorithm:

discussion and example

have its predecessor, and so in this manner we can construct the entire path from the source to all destinations. The forwarding table in a node, say node u , can then be constructed from this information by storing, for each destination, the next-hop node on the least-cost path from u to the destination. Figure 4.28 shows the resulting least-cost paths and forwarding table in u for the network in Figure 4.27.

What is the computational complexity of this algorithm? That is, given n nodes (not counting the source), how much computation must be done in the worst case to find the least-cost paths from the source to all destinations? In the first iteration, we need to search through all n nodes to determine the node, w , not in N_{EDT} that has the minimum cost. In the second iteration, we need to check $n - 1$ nodes to determine the minimum cost; in the third iteration $n - 2$ nodes, and so on. Overall, the total number of nodes we need to search through over all the iterations is $n(n + 1)/2$, and thus we say that the preceding implementation of the LS algorithm has worst-case complexity of order n squared: $O(n^2)$.

(A more sophisticated implementation of this algorithm, using a data structure known as a heap,

can find the minimum in line 9 in logarithmic rather than linear time, thus reducing the complexity.)

Before completing our discussion of the LS algorithm, let us consider a pathology that can arise.

Figure 4.29 shows a simple network topology where link costs are equal to the load carried on the link, for example, reflecting the delay that would be experienced. In this example, link costs are not symmetric; that is, $c(u,v)$ equals $c(v,u)$ only if the load carried on both directions on the link (u,v) is the same. In this example, node z originates a unit of traffic destined for w , node x also originates a

unit of traffic destined for w, and node y injects an amount of traffic equal to e, also destined for w. The initial routing is shown in Figure 4.29(a) with the link costs corresponding to the amount of traffic carried.

When the LS algorithm is next run, node y determines (based on the link costs shown in Figure 4.29(a)) that the clockwise path to w has a cost of 1, while the counterclockwise path to w (which it had been using) has a cost of $1 + e$. Hence y's

4.5 • ROUTING ALGORITHMS 369

Destination Link

```
v
w
x
y
z
(u, v)
(u, x)
(u, x)
(u, x)
X Y (u, x)
V
U Z
W
```

Figure 4.28 Least cost path and forwarding table for node u
least-cost path to w is now clockwise. Similarly, x determines that its new least-cost path to w is also clockwise, resulting in costs shown in Figure 4.29(b). When the LS algorithm is run next, nodes x, y, and z all detect a zero-cost path to w in the counterclockwise direction, and all route their traffic to the counterclockwise routes. The next time the LS algorithm is run, x, y, and z all then route their traffic to the clockwise routes.

What can be done to prevent such oscillations (which can occur in any algorithm, not just an LS algorithm, that uses a congestion or delay-based link metric)? One solution would be to mandate that link costs not depend on the amount

of traffic carried—an unacceptable solution since one goal of routing is to avoid that not all routers run the LS algorithm at the same time. This seems a more reasonable solution, since we would hope that even if routers ran the LS algorithm with the same periodicity, the execution instance of the algorithm would not be the same at each node. Interestingly, researchers have found that routers in the Internet can self-synchronize among themselves [Floyd Synchronization 1994].

That is, even though they initially execute the algorithm with the same period but at different instants of time, the algorithm execution instance can eventually become, and remain, synchronized at the routers. One way to avoid such self-synchronization is for each router to randomize the time it sends out a link advertisement.

Having studied the LS algorithm, let's consider the other major routing algorithm that is used in practice today—the distance-vector routing algorithm.

4.5.2 The Distance-Vector (DV) Routing Algorithm

Whereas the LS algorithm is an algorithm using global information, the distance-vector (DV) algorithm is iterative, asynchronous, and distributed. It is distributed in that each node receives some information from one or more of its directly attached neighbors, performs a calculation, and then distributes the results of its calculation back to its neighbors. It is iterative in that this process continues on until no more information is exchanged between neighbors. (Interestingly, the

algorithm is also self-terminating—there is no signal that the computation should stop; it just stops.) The algorithm is asynchronous in that it does not require all of the nodes to operate in lockstep with each other. We'll see that an asynchronous, iterative, self-terminating, distributed algorithm is much more interesting and fun than a centralized algorithm!

Before we present the DV algorithm, it will prove beneficial to discuss an important relationship that exists among the costs of the least-cost paths. Let $d_x(y)$

be the cost of the least-cost path from node x to node y . Then the least costs are related by the celebrated Bellman-Ford equation, namely,

d_x

$(y) = \min_v$

$\{c(x,v) + d_v$

$(y)\}$, (4.1)

where the \min_v in the equation is taken over all of x 's neighbors. The Bellman-Ford equation is rather intuitive. Indeed, after traveling from x to v , if we then take the least-cost path from v to y , the path cost will be $c(x,v) + d_v(y)$.

Since we must begin

by traveling to some neighbor v , the least cost from x to y is the minimum of $c(x,v) + d_v(y)$ taken over all neighbors v .

But for those who might be skeptical about the validity of the equation, let's check it for source node u and destination node z in Figure 4.27. The source node u has three neighbors: nodes v , x , and w . By walking along various paths in the graph, it is easy to see that $d_v(z) = 5$, $d_x(z) = 3$, and $d_w(z) = 3$. Plugging these values into Equation 4.1, along with the costs $c(u,v) = 2$, $c(u,x) = 1$, and $c(u,w) = 5$, gives $d_u(z) = \min\{2 + 5, 5 + 3, 1 + 3\} = 4$, which is obviously true and which is exactly what the Dijkstra algorithm gave us for the same network. This quick verification should help relieve any skepticism you may have.

The Bellman-Ford equation is not just an intellectual curiosity. It actually has significant practical importance. In particular, the solution to the Bellman-Ford equation provides the entries in node x 's forwarding table. To see this, let v^* be any neighboring node that achieves the minimum in Equation 4.1. Then, if node x wants to send a packet to node y along a least-cost path, it should first forward the packet to node v^* . Thus, node x 's forwarding table would specify node v^* as the next-hop router for the ultimate destination y . Another important practical contribution of the Bellman-Ford equation is that it suggests the form of the neighbor-to-neighbor communication that will take place in the DV algorithm.

The basic idea is as follows. Each node x begins with $D_x(y)$, an estimate of the

cost of the least-cost path from itself to node y , for all nodes in N . Let $D_x = [D_x(y)]_{y \in N}$

be node x 's distance vector, which is the vector of cost estimates from x to all other nodes, y , in N . With the DV algorithm, each node x maintains the following routing information:

- For each neighbor v , the cost $c(x,v)$ from x to directly attached neighbor, v

- Node x 's distance vector, that is, $D_x = [D_x(y)]_{y \in N}$

- (y) in N , containing x 's estimate of

its cost to all destinations, y , in N

- The distance vectors of each of its neighbors, that is, $D_v = [D_{v(y)}: y \in N]$ for each neighbor v of x

In the distributed, asynchronous algorithm, from time to time, each node sends a copy of its distance vector to each of its neighbors. When a node x receives a new distance vector from any of its neighbors v , it saves v 's distance vector, and then uses the Bellman-Ford equation to update its own distance vector as follows:

D_x

$(y) \leftarrow \min\{c(x,y) + D_v(y)\}$

{ $c(x,v)$ + $D_v(y)$ }

(y) for each node y in N

If node x 's distance vector has changed as a result of this update step, node x will then send its updated distance vector to each of its neighbors, which can in turn update their own distance vectors. Miraculously enough, as long as all the nodes continue to exchange their distance vectors in an asynchronous fashion, each cost estimate D_x

(y) converges to d_x

(y), the actual cost of the least-cost path from node x to node y [Bertsekas 1991]!

372 CHAPTER 4 • THE NETWORK LAYER

Distance-Vector (DV) Algorithm

At each node, x :

4.5 • ROUTING ALGORITHMS 373

1 Initialization:

2 for all destinations y in N :

3 $D_x(y) = c(x,y)$ /* if y is not a neighbor then $c(x,y) = \infty$ */

4 for each neighbor w

5 $D_w(y) = ?$ for all destinations y in N

6 for each neighbor w

7 send distance vector $D_x = [D_x(y): y \in N]$ to w

8

9 loop

10 wait (until I see a link cost change to some neighbor w or

11 until I receive a distance vector from some neighbor w)

12

13 for each y in N :

14 $D_x(y) = \min\{c(x,y) + D_w(y)\}$

15

16 if $D_x(y)$ changed for any destination y

17 send distance vector $D_x = [D_x(y): y \in N]$ to all neighbors

18

19 forever

In the DV algorithm, a node x updates its distance-vector estimate when it either sees a cost change in one of its directly attached links or receives a distance vector update from some neighbor. But to update its own forwarding table for a given destination y , what node x really needs to know is not the shortest-path distance to y but instead the neighboring node $v^*(y)$ that is the next-hop router along the shortest path to y . As you might expect, the next-hop router $v^*(y)$ is the neighbor v that achieves the minimum in Line 14 of the DV algorithm. (If there are multiple neighbors v that achieve the minimum, then $v^*(y)$ can be any of the minimizing

neighbors.) Thus, in Lines 13–14, for each destination y , node x also determines $v^*(y)$ and updates its forwarding table for destination y .

Recall that the LS algorithm is a global algorithm in the sense that it requires each node to first obtain a complete map of the network before running the Dijkstra algorithm. The DV algorithm is decentralized and does not use such global information. Indeed, the only information a node will have is the costs of the links to its directly attached neighbors and information it receives from these neighbors. Each node waits for an update from any neighbor (Lines 10–11), calculates its new distance vector when receiving an update (Line 14), and distributes its new distance vector to its neighbors (Lines 16–17). DV-like algorithms are used in many routing protocols in practice, including the Internet's RIP and BGP, ISO IDRP, Novell IPX, and the original ARPAnet.

Figure 4.30 illustrates the operation of the DV algorithm for the simple three-node network shown at the top of the figure. The operation of the algorithm is illustrated in a synchronous manner, where all nodes simultaneously receive distance vectors from their neighbors, compute their new distance vectors, and inform their neighbors if their distance vectors have changed. After studying this example, you

374 CHAPTER 4 • THE NETWORK LAYER

Node y table

Node x table

0 2 7

x y z

$\infty \infty \infty$

$\infty \infty \infty$

Time

7

2 1

y

x z

Node z table from

cost to

x

y

z

0 2 3

x y z

2 0 1

7 1 0

from

cost to

x

y

z

0 2 3

x y z

2 0 1

3 1 0

from

cost to

x

y

z
201
xyz
 $\infty \infty \infty$
 $\infty \infty \infty$
from
cost to
x
y
z
027
xyz
201
710
from
cost to
x
y
z
023
xyz
201
310
from
cost to
x
y
z
710
xyz
 $\infty \infty \infty$
 $\infty \infty \infty$
from
cost to
x
y
z
027
xyz
201
310
from
cost to
x
y
z
023
xyz
201
310
from

cost to

x
y
z

Figure 4.30 [STX] Distance-vector (DV) algorithm

should convince yourself that the algorithm operates correctly in an asynchronous manner as well, with node computations and update generation/reception occurring at any time.

The leftmost column of the figure displays three initial routing tables for each of the three nodes. For example, the table in the upper-left corner is node x's initial routing table. Within a specific routing table, each row is a distance vector—specifically, each node's routing table includes its own distance vector and that of each of its neighbors. Thus, the first row in node x's initial routing table is $D_x = [D_x(x), D_x(y), D_x(z)] = [0, 2, 7]$.

(x) , D_x

(y) , D_x

$(z)] = [0, 2, 7]$. The second and third rows in this table are the most recently received distance vectors from nodes y and z, respectively. Because at initialization node x has not received anything from node y or z, the entries in the second and third rows are initialized to infinity.

After initialization, each node sends its distance vector to each of its two neighbors. This is illustrated in Figure 4.30 by the arrows from the first column of tables

to the second column of tables. For example, node x sends its distance vector $D_x = [0, 2, 7]$ to both nodes y and z. After receiving the updates, each node recomputes its own distance vector. For example, node x computes

D_x

$(x) = 0$

D_x

$(y) = \min\{c(x,y) + D_x(y), c(x,z) + D_x(z)\} = \min\{2 + 0, 7 + 1\} = 2$

y

$(y), c(x,z) + D_x(z)$

$(y)\} = \min\{2 + 1, 7 + 0\} = 3$

D_x

$(z) = \min\{c(x,y) + D_x(y), c(x,z) + D_x(z)\} = \min\{2 + 1, 7 + 0\} = 3$

y

$(z), c(x,z) + D_x(z)$

$(z)\} = \min\{2 + 1, 7 + 0\} = 3$

The second column therefore displays, for each node, the node's new distance vector along with distance vectors just received from its neighbors. Note, for example, that node x's estimate for the least cost to node z, $D_x(z)$, has changed from 7 to 3.

Also note that for node x, neighboring node y achieves the minimum in line 14 of the DV algorithm; thus at this stage of the algorithm, we have at node x that $v^*(y) = y$ and $v^*(z) = y$.

After the nodes recompute their distance vectors, they again send their updated distance vectors to their neighbors (if there has been a change). This is illustrated in Figure 4.30 by the arrows from the second column of tables to the third column of tables. Note that only nodes x and z send updates: node y's distance vector didn't change so node y doesn't send an update. After receiving the updates, the nodes then recompute their distance vectors and update their routing tables, which are shown in the third column.

The process of receiving updated distance vectors from neighbors, recomputing routing table entries, and informing neighbors of changed costs of the least-cost path to a destination continues until no update messages are sent. At this point, since no update messages are sent, no further routing table calculations will occur and the algorithm will enter a quiescent state; that is, all nodes will be performing the wait in Lines 10–11 of the DV algorithm. The algorithm remains in the quiescent state until a link cost changes, as discussed next.

4.5 • ROUTING ALGORITHMS 375

Distance-Vector Algorithm: Link-Cost Changes and Link Failure

When a node running the DV algorithm detects a change in the link cost from itself to a neighbor (Lines 10–11), it updates its distance vector (Lines 13–14) and, if there's a change in the cost of the least-cost path, informs its neighbors (Lines 16–17) of its new distance vector. Figure 4.31(a) illustrates a scenario where the link cost from y to x changes from 4 to 1. We focus here only on y' and z's distance table entries to destination x. The DV algorithm causes the following sequence of events to occur:

- At time t

0, y detects the link-cost change (the cost has changed from 4 to 1), updates its distance vector, and informs its neighbors of this change since its distance vector has changed.

- At time t

1, z receives the update from y and updates its table. It computes a new least cost to x (it has decreased from a cost of 5 to a cost of 2) and sends its new distance vector to its neighbors.

- At time t

2, y receives z's update and updates its distance table. y's least costs do not change and hence y does not send any message to z. The algorithm comes to a quiescent state.

Thus, only two iterations are required for the DV algorithm to reach a quiescent state. The good news about the decreased cost between x and y has propagated quickly through the network.

Let's now consider what can happen when a link cost increases. Suppose that the link cost between x and y increases from 4 to 60, as shown in Figure 4.31(b).

1. Before the link cost changes, D

y
(x) = 4, D

y
(z) = 1, Dz
(y) = 1, and Dz
(x) = 5. At

time t

0, y detects the link-cost change (the cost has changed from 4 to 60). y computes its new minimum-cost path to x to have a cost of

D

y
(x) = min{c(y,x) + Dx
(x), c(y,z) + Dz
(x)} = min{60 + 0, 1 + 5} = 6

376 CHAPTER 4 • THE NETWORK LAYER

50

4

1 60

```

1
y
x
a. b.
z 50
4 1
y
x z

```

Figure 4.31 [STX] Changes in link cost

Of course, with our global view of the network, we can see that this new cost via z is wrong. But the only information node y has is that its direct cost to x is 60 and that z has last told y that z could get to x with a cost of 5. So in order to get to x, y would now route through z, fully expecting that z will be able to get to x with a cost of 5. As of t

1 we have a routing loop—in order to get to x, y routes through z, and z routes through y. A routing loop is like a black hole—a packet destined for x arriving at y or z as of t

1 will bounce back and forth between these two nodes forever (or until the forwarding tables are changed).
2. Since node y has computed a new minimum cost to x, it informs z of its new distance vector at time t

1.

3. Sometime after t

1, z receives y's new distance vector, which indicates that y's minimum cost to x is 6. z knows it can get to y with a cost of 1 and hence computes a new least cost to x of Dz

$(x) = \min\{50 + 0, 1 + 6\} = 7$. Since z's least cost to x has increased, it then informs y of its new distance vector at t

2.

4. In a similar manner, after receiving z's new distance vector, y determines

D

y

$(x) = 8$ and sends z its distance vector. z then determines Dz

$(x) = 9$ and

sends y its distance vector, and so on.

How long will the process continue? You should convince yourself that the loop will persist for 44 iterations (message exchanges between y and z)—until z eventually computes the cost of its path via y to be greater than 50. At this point, z will (finally!) determine that its least-cost path to x is via its direct connection to x. y will then route to x via z. The result of the bad news about the increase in link cost has indeed traveled slowly! What would have happened if the link cost $c(y, x)$ had changed from 4 to 10,000 and the cost $c(z, x)$ had been 9,999? Because of such scenarios, the problem we have seen is sometimes referred to as the count-to-infinity problem.

Distance-Vector Algorithm: Adding Poisoned Reverse

The specific looping scenario just described can be avoided using a technique known as poisoned reverse. The idea is simple—if z routes through y to get to destination x, then z will advertise to y that its distance to x is infinity, that is, z will advertise to y that Dz

$(x) = \infty$ (even though z knows Dz)

$(x) = 5$ in truth). z will continue telling this little white lie to y as long as it routes to x via y. Since y

believes

that z has no path to x, y will never attempt to route to x via z, as long as z continues to route to x via y (and lies about doing so).

Let's now see how poisoned reverse solves the particular looping problem we encountered before in Figure 4.31(b). As a result of the poisoned reverse, y's distance table indicates D_z

(x) = ∞ . When the cost of the (x, y) link changes from 4 to 60 at time t

0, y updates its table and continues to route directly to x, albeit at a higher cost of 60, and informs z of its new cost to x, that is, D_z

y

(x) = 60. After receiving the

4.5 • ROUTING ALGORITHMS 377

update at t

1, z immediately shifts its route to x to be via the direct (z, x) link at a cost of 50. Since this is a new least-cost path to x, and since the path no longer passes through y, z now informs y that D_z

(x) = 50 at t

2. After receiving the update from z, y updates its distance table with D_z

y

(x) = 51. Also, since z is now on y's least-cost path

to x, y poisons the reverse path from z to x by informing z at time t

3 that D_z

y

(x) = ∞

(even though y knows that D_z)

y

(x) = 51 in truth).

Does poisoned reverse solve the general count-to-infinity problem? It does not.

You should convince yourself that loops involving three or more nodes (rather than simply two immediately neighboring nodes) will not be detected by the poisoned reverse technique.

A Comparison of LS and DV Routing Algorithms

The DV and LS algorithms take complementary approaches towards computing routing. In the DV algorithm, each node talks to only its directly connected neighbors, but it

provides its neighbors with least-cost estimates from itself to all the

nodes (that it knows about) in the network. In the LS algorithm, each node talks with

all other nodes (via broadcast), but it tells them only the costs of its directly connected links. Let's conclude our study of LS and DV algorithms with a quick comparison of some of their attributes.

Recall that N is the set of nodes (routers) and E

is the set of edges (links).

- Message complexity. We have seen that LS requires each node to know the cost of each link in the network. This requires $O(|N| \cdot |E|)$ messages to be sent.

Also, whenever a link cost changes, the new link cost must be sent to all

nodes. The DV algorithm requires message exchanges between directly connected neighbors at each iteration. We have seen that the time needed for the

algorithm to converge can depend on many factors. When link costs change, the DV algorithm will propagate the results of the changed link cost only if the new link cost results in a changed least-cost path for one of the nodes attached to that link.

- Speed of convergence. We have seen that our implementation of LS is an $O(|N|^2)$ algorithm requiring $O(|N| |E|)$ messages. The DV algorithm can converge slowly and can have routing loops while the algorithm is converging. DV also suffers from the count-to-infinity problem.
- Robustness. What can happen if a router fails, misbehaves, or is sabotaged? Under LS, a router could broadcast an incorrect cost for one of its attached links (but no others). A node could also corrupt or drop any packets it received as part of an LS broadcast. But an LS node is computing only its own forwarding tables; other nodes are performing similar calculations for themselves. This means route calculations are somewhat separated under LS, providing a degree of robustness. Under DV, a node can advertise incorrect least-cost paths to any or all destinations. (Indeed, in 1997, a malfunctioning router in a small ISP

378 CHAPTER 4 • THE NETWORK LAYER

provided national backbone routers with erroneous routing information. This caused other routers to flood the malfunctioning router with traffic and caused large portions of the Internet to become disconnected for up to several hours [Neumann 1997].) More generally, we note that, at each iteration, a node's calculation in DV is passed on to its neighbor and then indirectly to its neighbor's neighbor on the next iteration. In this sense, an incorrect node calculation can be diffused through the entire network under DV.

In the end, neither algorithm is an obvious winner over the other; indeed, both algorithms are used in the Internet.

Other Routing Algorithm

The LS and DV algorithms we have studied are not only widely used in practice, they are essentially the only routing algorithms used in practice today in the Internet. Nonetheless, many routing algorithms have been proposed by researchers over the past 30 years, ranging from the extremely simple to the very sophisticated and complex. A broad class of routing algorithms is based on viewing packet traffic as flows between sources and destinations in a network. In this approach, the routing problem can be formulated mathematically as a constrained optimization problem known as a network flow problem [Bertsekas 1991]. Yet another set of routing algorithms we mention here are those derived from the telephony world. These circuit-switched routing algorithms are of interest to packet-switched data networking in cases where per-link resources (for example, buffers, or a fraction of the link bandwidth) are to be reserved for each connection that is routed over the link. While the formulation of the routing problem might appear quite different from the least-cost routing formulation we have seen in this chapter, there are a number of similarities, at least as far as the path-finding algorithm (routing algorithm) is concerned. See [Ash 1998; Ross 1995; Girard 1990] for a detailed discussion of this research area.

4.5.3 Hierarchical Routing

In our study of LS and DV algorithms, we've viewed the network simply as a collection of interconnected routers. One router was indistinguishable from another in the sense that all routers executed the same routing algorithm to compute routing paths through the entire network. In practice, this model and its view of a homogeneous set of routers all executing the same routing algorithm is a bit simplistic for at least two important reasons:

- Scale. As the number of routers becomes large, the overhead involved in computing, storing, and communicating routing information (for example,

4.5 • ROUTING ALGORITHMS 379

LS updates or least-cost path changes) becomes prohibitive. Today's public

Internet consists of hundreds of millions of hosts. Storing routing information at each of these hosts would clearly require enormous amounts of memory. The overhead required to broadcast LS updates among all of the routers in the public Internet would leave no bandwidth left for sending data packets! A distance-vector algorithm that iterated among such a large number of routers would surely never converge. Clearly, something must be done to reduce the complexity of route computation in networks as large as the public Internet.

- Administrative autonomy. Although researchers tend to ignore issues such as a company's desire to run its routers as it pleases (for example, to run whatever routing algorithm it chooses) or to hide aspects of its network's internal organization from the outside, these are important considerations. Ideally, an organization should be able to run and administer its network as it wishes, while still being able to connect its network to other outside networks.

Both of these problems can be solved by organizing routers into autonomous systems (ASes), with each AS consisting of a group of routers that are typically under the same administrative control (e.g., operated by the same ISP or belonging to the same company network). Routers within the same AS all run the same routing algorithm (for example, an LS or DV algorithm) and have information about each other—exactly as was the case in our idealized model in the preceding section. The routing algorithm running within an autonomous system is called an intra-autonomous system routing protocol. It will be necessary, of course, to connect ASes to each other, and thus one or more of the routers in an AS will have the added task of being responsible for forwarding packets to destinations outside the AS; these routers are called gateway routers.

Figure 4.32 provides a simple example with three ASes: AS1, AS2, and AS3. In this figure, the heavy lines represent direct link connections between pairs of routers. The thinner lines hanging from the routers represent subnets that are directly connected to the routers. AS1 has four routers—1a, 1b, 1c, and 1d—which run the intra-AS routing protocol used within AS1. Thus, each of these four routers knows how to forward packets along the optimal path to any destination within AS1. Similarly, autonomous systems AS2 and AS3 each have three routers. Note that the intra-AS routing protocols running in AS1, AS2, and AS3 need not be the same. Also note that the routers 1b, 1c, 2a, and 3a are all gateway routers.

It should now be clear how the routers in an AS determine routing paths for source-destination pairs that are internal to the AS. But there is still a big missing piece to the end-to-end routing puzzle. How does a router, within some AS, know how to route a packet to a destination that is outside the AS? It's easy to answer this question if the AS has only one gateway router that connects to only one other AS. In this case, because the AS's intra-AS routing algorithm has determined the least-cost path from each internal router to the gateway router, each

380 CHAPTER 4 • THE NETWORK LAYER

internal router knows how it should forward the packet. The gateway router, upon receiving the packet, forwards the packet on the one link that leads outside the AS. The AS on the other side of the link then takes over the responsibility of routing the packet to its ultimate destination. As an example, suppose router 2b in Figure 4.32 receives a packet whose destination is outside of AS2. Router 2b will then forward the packet to either router 2a or 2c, as specified by router 2b's forwarding table, which was configured by AS2's intra-AS routing protocol. The packet will eventually arrive to the gateway router 2a, which will forward the packet to 1b. Once the packet has left 2a, AS2's job is done with this one

packet.

So the problem is easy when the source AS has only one link that leads outside the AS. But what if the source AS has two or more links (through two or more gateway routers) that lead outside the AS? Then the problem of knowing where to forward the packet becomes significantly more challenging. For example, consider a router in AS1 and suppose it receives a packet whose destination is outside the AS. The router should clearly forward the packet to one of its two gateway routers, 1b or 1c, but which one? To solve this problem, AS1 needs (1) to learn which destinations are reachable via AS2 and which destinations are reachable via AS3, and (2) to propagate this reachability information to all the routers within AS1, so that each router can configure its forwarding table to handle external-AS destinations. These

4.5 • ROUTING ALGORITHMS 381

AS1

AS3

3b

3c

3a

1a

1c

1b

1d

AS2

2a

2c

2b

Intra-AS routing
algorithm
Forwarding
table

Inter-AS routing
algorithm

Figure 4.32 An example of interconnected autonomous systems
two tasks—obtaining reachability information from neighboring ASs and propagating the reachability information to all routers internal to the AS—are handled by the inter-AS routing protocol. Since the inter-AS routing protocol involves communication between two ASs, the two communicating ASs must run the same inter-AS routing protocol. In fact, in the Internet all ASs run the same inter-AS routing protocol, called BGP4, which is discussed in the next section. As shown in Figure 4.32, each router receives information from an intra-AS routing protocol and an inter-AS routing protocol, and uses the information from both protocols to configure its forwarding table. As an example, consider a subnet x (identified by its CIDRized address), and suppose that AS1 learns from the inter-AS routing protocol that subnet x is reachable from AS3 but is not reachable from AS2. AS1 then propagates this information to all of its routers. When router 1d learns that subnet x is reachable from AS3, and hence from gateway 1c, it then determines, from the information provided by the intra-AS routing protocol, the router interface that is on the least-cost path from router 1d to gateway router 1c. Say this is interface I. The router 1d can then put the entry (x, I) into its forwarding table. (This example, and others presented in this section, gets the general ideas across but is a simplification of what really happens in the Internet. In the next section we'll provide a more detailed description, albeit more complicated, when we discuss BGP.)

Following up on the previous example, now suppose that AS2 and AS3 connect to other ASs, which are not shown in the diagram. Also suppose that AS1 learns from the inter-AS routing protocol that subnet x is reachable both from AS2, via gateway 1b, and from AS3, via gateway 1c. AS1 would then propagate this information to all its routers, including router 1d. In order to configure its forwarding table, router 1d would have to determine to which gateway router, 1b or 1c, it should direct packets that are destined for subnet x. One approach, which is often employed in practice, is to use hot-potato routing. In hot-potato routing, the AS gets rid of the packet (the hot potato) as quickly as possible (more precisely, as inexpensively as possible). This is done by having a router send the packet to the gateway router that has the smallest router-to-gateway cost among all gateways with a path to the destination. In the context of the current example, hot-potato routing, running in 1d, would use information from the intra-AS routing protocol to determine the path costs to 1b and 1c, and then choose the path with the least cost. Once this path is chosen, router 1d adds an entry for subnet x in its forwarding table. Figure 4.33 summarizes the actions taken at router 1d for adding the new entry for x to the forwarding table.

When an AS learns about a destination from a neighboring AS, the AS can advertise this routing information to some of its other neighboring ASs. For example, suppose AS1 learns from AS2 that subnet x is reachable via AS2. AS1 could then tell AS3 that x is reachable via AS1. In this manner, if AS3 needs to route a packet destined to x, AS3 would forward the packet to AS1, which would in turn forward the packet to AS2. As we'll see in our discussion of BGP, an AS has quite a bit of flexibility in deciding which destinations it advertises to its neighboring ASs. This is a policy decision, typically depending more on economic issues than on technical issues.

Recall from Section 1.5 that the Internet consists of a hierarchy of interconnected ISPs. So what is the relationship between ISPs and ASs? You might think that the routers in an ISP, and the links that interconnect them, constitute a single AS. Although this is often the case, many ISPs partition their network into multiple ASs. For example, some tier-1 ISPs use one AS for their entire network; others break up their ISP into tens of interconnected ASs.

In summary, the problems of scale and administrative authority are solved by defining autonomous systems. Within an AS, all routers run the same intra-AS routing protocol. Among themselves, the ASs run the same inter-AS routing protocol.

The problem of scale is solved because an intra-AS router need only know about routers within its AS. The problem of administrative authority is solved since an organization can run whatever intra-AS routing protocol it chooses; however, each pair of connected ASs needs to run the same inter-AS routing protocol to exchange reachability information.

In the following section, we'll examine two intra-AS routing protocols (RIP and OSPF) and the inter-AS routing protocol (BGP) that are used in today's Internet. These case studies will nicely round out our study of hierarchical routing.

Nice chunk of routing theory here 😊 Let's go piece by piece and keep it clean + intuitive. I'll follow exactly the items you listed:

graph model, shortest path, global vs decentralized routing algorithms, Link State protocol, Distance Vector, comparison LS vs DV, Hierarchical routing.

1. Graph model & shortest path

We model the network as a **graph**:

- **Nodes (N)** = routers
- **Edges (E)** = physical links between routers
- Each edge (x, y) has a **cost** $c(x, y)$ (delay, distance, monetary cost, etc.)

If (x, y) is **not** a link $\rightarrow c(x, y) = \infty$.

A **path** from x_1 to x_p is a sequence of nodes:

$(x_1, x_2, x_3, \dots, x_p)$
where each consecutive pair is connected by a link.

Path cost = sum of edge costs along the path:

$$c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$$

Routing goal:

Find a **least-cost path** between source router and destination router.

If all link costs are equal (say cost = 1 per link),

- “least-cost path” = “fewest hops” (shortest in number of links).
-

2. Global vs decentralized routing algorithms

This is about **who knows what** when computing routes.

2.1 Global routing algorithms (Link-State style)

- Each router has **complete knowledge** of:
 - All nodes
 - All links
 - All link costs
- You can think of it like: every router has the same full map of the network.
- The routing algorithm runs with this full map and computes least-cost paths.

In practice, these are called **Link-State (LS)** algorithms:

- Because each node must know the “state” (cost) of **every** link.
-

2.2 Decentralized routing algorithms (Distance-Vector style)

- No node has global information.
- Each node knows only:
 - Costs of **its directly connected links** (to neighbors).
- Nodes **exchange information with neighbors**, iteratively.
- Over time, everyone's estimates converge to correct shortest paths.

This is called a **Distance-Vector (DV)** algorithm because:

- Each node maintains a **vector** of distances (cost estimates) to all destinations.
-

2.3 Other classifications

1. Static vs Dynamic

- **Static**: routes change rarely, usually by manual configuration.
- **Dynamic**: routes adapt to changes (link failures, topology changes).
Can be periodic or triggered by changes.

2. Load-sensitive vs Load-insensitive

- **Load-sensitive**: link cost depends on current congestion (e.g., delay).
 - **Load-insensitive**: link cost is fixed (e.g., 1/hop or configured weight).
Modern Internet protocols (RIP, OSPF, BGP) are mostly **load-insensitive**.
-

3. Link-State (LS) routing: high-level idea

Core idea:

1. Every router **learns the full topology + link costs**.
2. Every router runs the **same shortest-path algorithm (Dijkstra)** on that topology.
3. Everyone computes the same set of shortest paths, only from their own perspective.

3.1 Step 1 – Get full topology: link-state advertisements

Each router:

- Creates a **link-state packet (LSP)** containing:
 - Who I am
 - Who my neighbors are
 - The cost to each neighbor

Then:

- It **floods** this LSP to all other routers (broadcast).
- so eventually **every router has a copy** of everyone's LSP.

Now every router has the **same complete network map**.

3.2 Step 2 – Run Dijkstra's algorithm

We run **Dijkstra** from the source router u :

Notations:

- $D(v)$ = current best known distance from u to v
- $p(v)$ = predecessor of v on the shortest path from u
- N' = set of nodes whose shortest path from u is **finalized**

Algorithm sketch:

1. Initialization

- $N' = \{u\}$
- For each node v :
 - If v is neighbor of u : $D(v) = c(u, v)$
 - Else: $D(v) = \infty$

2. Loop

- Find node w not in N' with smallest $D(w)$
- Add w to N'
- For each neighbor v of w not in N' :
 - Update:

```
text
D(v) = min( D(v), D(w) + c(w, v) )
```

(Either keep old path, or use path via w if cheaper)

3. Repeat until N' contains all nodes.

When done:

- For each node v , we know:
 - $D(v)$ = cost of shortest path
 - $p(v)$ = predecessor → we can reconstruct entire path.
- To build **forwarding table** at u , for each destination:
 - Look at the path and pick the **first hop**.

3.3 Complexity and issues

- Simple implementation: **$O(N^2)$** time per router (N = number of routers).
- Requires **broadcast** of link-state info → **$O(N \cdot E)$** messages.

Oscillation issue:

- If link cost depends on load (e.g., higher cost when more traffic), routers may change routes in a way that makes traffic “swing” back and forth. That’s why modern protocols usually use **fixed link costs**, not dynamic congestion-based ones.
-

4. Distance-Vector (DV) routing: high-level idea

Where LS uses global info, DV uses **local neighbor info + the Bellman–Ford equation**.

4.1 Bellman–Ford equation

Let:

- $d_x(y)$ = cost of the shortest path from node x to node y
- $c(x, v)$ = cost of direct link $x \rightarrow v$ (∞ if no link)

Then:

Bellman–Ford:

$$d_x(y) = \min \text{ over neighbors } v \text{ of } [c(x, v) + d_v(y)]$$

Meaning:

- To go from x to y , you must first pick a neighbor v .
- Total cost = cost to get to v + best cost from v to y .
- So you choose the **neighbor v that gives minimum total cost**.

This equation both:

- Defines the correct shortest path distances.
 - Suggests how nodes should **exchange information**.
-

4.2 What each node stores

Node x keeps:

1. $c(x, v)$ for each neighbor v
 2. Its **own distance vector $d_x(y)$** for all destinations y
 3. The **distance vectors of each neighbor: $d_v(y)$** for each neighbor v
-

4.3 DV algorithm (behavior)

Each node x periodically or upon change:

1. **Sends** its current distance vector d_x to all neighbors.

2. **Receives** neighbors' distance vectors Dv .

3. **Updates** its own estimates:

text

$$Dx(y) = \min \text{ over neighbors } v [c(x, v) + Dv(y)]$$

4. If any $Dx(y)$ value changes:

- Send updated Dx to neighbors again.

This repeats **until nothing changes** → algorithm converges.

Important: it's **distributed, iterative, asynchronous, self-terminating**.

4.4 Link-cost changes & count-to-infinity

- If a link **cost decreases** → good news spreads quickly.
- If a link **cost increases** → can cause **slow convergence** and **loops**.

Example problem:

- Two nodes think they can reach destination x via each other.
- They keep increasing their estimates ($5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow \dots$) until they realize the direct path is cheaper.
- This is the **count-to-infinity** problem.

Poisoned reverse:

- If node z routes to x via neighbor y , it tells y :
“My distance to x is ∞ ” (a lie to prevent loops through me)
 - This prevents simple 2-node loops, but **not more complex loops** with 3+ nodes.
-

5. LS vs DV: comparison

Message complexity

- **LS:**
 - Each node floods its link-state to all others.
 - About **$O(N \cdot E)$** messages.
 - On link-cost change: new LSP must again be flooded.
- **DV:**
 - Messages exchanged only between **neighbors**.
 - Number of iterations until convergence depends on:

- Topology
 - Link-cost changes
 - Could be fast or painfully slow (count-to-infinity).
-

Speed of convergence

- **LS:**
 - Usually **fast**, predictable:
 - Flood updated LSPs
 - Run Dijkstra ($O(N^2)$)
 - **DV:**
 - Can be **slow** on bad news (link cost increase).
 - Can have **routing loops** while converging.
 - Count-to-infinity is the main problem.
-

Robustness

- **LS:**
 - Each router computes **its own paths** using global map.
 - If one router lies or misbehaves:
 - It mainly affects **its own view** and the link costs it advertises.
 - Some robustness because everyone has the full map and computes locally.
- **DV:**
 - Each node's **results depend on neighbors' distance vectors**.
 - If a node advertises wrong distances, that error can **propagate** through the network.
 - One bad node can poison many others.

In practice: **both** LS and DV are used in the Internet:

- **OSPF / IS-IS** → Link-State (intra-AS)
 - **RIP, BGP (conceptually)** → Distance-Vector style
-

6. Hierarchical routing

Why we can't just run one big LS or DV across the whole Internet:

1. Scale:

- Millions of routers/hosts
- Huge routing tables

- Massive overhead for flooding or DV convergence

2. Administrative autonomy:

- Different organizations (ISPs, companies) want:
 - Their own routing policies
 - Their own internal structure
- They don't want to expose all internal details.

6.1 Autonomous Systems (AS)

Solution: **group routers into Autonomous Systems (ASs)**.

- **AS** = a collection of routers under one administrative control:
 - e.g., one ISP
 - one company's internal network

Within an AS:

- All routers run the **same intra-AS routing protocol** (e.g., OSPF, RIP).
- Each router knows how to reach all internal subnets.

Some routers are **gateway routers**:

- They connect this AS to **other ASs**.
-

6.2 Intra-AS vs Inter-AS routing

- **Intra-AS routing protocol**:
 - Runs **inside** an AS
 - Example: **RIP, OSPF, IS-IS**
 - Computes routes to all internal destinations
- **Inter-AS routing protocol**:
 - Runs **between ASs**
 - All ASs must run the **same inter-AS protocol**
 - In the Internet: **BGP-4** is the standard inter-AS protocol

Each router:

- Uses both:
 - Intra-AS info (inside topology)
 - Inter-AS info (which AS can reach which prefixes)
 - To build its forwarding table.
-

6.3 Example behavior

Imagine AS1, AS2, AS3:

- AS1 learns (via BGP) that subnet x is reachable via AS3.
- AS1 spreads this info internally using its intra-AS protocol:
 - Each router in AS1 knows:
"To reach subnet x, go toward gateway 1c (which connects to AS3)."

Now router 1d (internal) decides:

- How to reach gateway 1c (using OSPF/RIP inside AS1)
- Adds forwarding entry:
 - `dest = x → next hop = interface towards 1c.`

If AS1 learns x is reachable both via AS2 and AS3:

- It might use **hot potato routing**:
 - Choose the gateway (1b vs 1c) that is **closest in cost inside AS1**.
 - "Get rid of the packet as quickly/inexpensively as possible."

6.4 Why hierarchical routing helps

- **Scale:**
 - A router in AS1 only needs:
 - Full info for **its own AS**
 - Summary "which prefixes are reachable through which neighboring ASs," not full global topology.
- **Autonomy:**
 - Each AS chooses its own intra-AS protocol and policies.
 - Only the inter-AS protocol (BGP) must be shared between ASs.

If you want, next we can:

- Do a **small worked example** of Dijkstra (LS) vs Bellman–Ford (DV) on a tiny graph, or
- Make a tiny "cheat sheet" summarizing LS vs DV vs hierarchical routing in a table for last-minute exam revision.



explain poisoned reverse

Poisoned reverse is a little hack on top of Distance-Vector (DV) routing to prevent the simplest kind of routing loop (a 2-node loop) and the “count-to-infinity” behavior between **two neighbors**.

Let's build it up slowly.

1. Recap: where the problem comes from

In Distance-Vector (DV) routing:

- Each router x keeps a **distance vector** $D_x(y)$ = best-known cost from x to every destination y .
- Each router periodically sends its D_x to **all neighbors**.
- To update its own table, x uses the Bellman–Ford equation:

$$D_x(y) = \min_v \{c(x, v) + D_v(y)\}$$

where v ranges over neighbors of x .

This works fine, but there's a classic problem when a **link cost increases or a link fails**.

Example: 2-node loop between y and z

Imagine three nodes: x , y , z .

- z can reach x directly with cost 5.
- y can reach z with cost 1.
- There *was* a direct link $y-x$ with cost 4, but that link suddenly becomes **very expensive** (say 60) or fails.

Before the change:

- z thinks:
 - $D_z(x) = 5$ (direct link to x)
- y thinks:
 - $D_y(x) = 4$ (direct link to x)
 - or maybe $D_y(x) = \min(4, 1 + 5) = 4$

After the link cost $c(y, x)$ becomes huge (60):

1. y recomputes:

```
text


$$D_y(x) = \min( c(y, x) + D_x(x), c(y, z) + D_z(x) )$$


$$= \min(60 + 0, 1 + 5)$$


$$= 6$$

```

So y now thinks: “Best path to x is via z with cost 6.”

2. But **z** still thinks: "I reach **x** directly with cost 5."

So **z** recomputes:

```
text

Dz(x) = min( c(z, x) + Dx(x), c(z, y) + Dy(x) )
        = min(5 + 0, 1 + 6)
        = 5
```

So **z** keeps using the **direct route** to **x**.

3. Now **y** sends $Dy(x) = 6$ to **z**, **z** sends $Dz(x) = 5$ to **y**, and they keep updating.

If the direct link from **z** to **x** is also broken or very expensive, they can end up in a **loop**:

- **y** says: "I can get to **x** via **z** with cost 7."
- **z** says: "I can get to **x** via **y** with cost 8."
- **y**: "Oh, via **z** is now 9."
- **z**: "Via **y** is now 10."
- And so on → **count-to-infinity**.

This happens because:

- Each node **trusts** the other's distance to **x**.
- They don't realize they are trying to reach **x** *through each other*.

2. Intuition of poisoned reverse

Key idea:

If router **z** is using neighbor **y** as the **next hop** to reach destination **x**, then when **z** advertises its distance to **x** **to y**, it will lie and say:

"My distance to **x** is ∞ ."

This is the "poison":

- To *everyone else*, **z** advertises the real cost to **x**.
- But **back to the neighbor along the chosen path**, it advertises ∞ .

Effect:

- **y** will *never* think "I can reach **x** via **z**" when **z** itself is using **y** to reach **x**.
- So you can't get that simple 2-node loop:
"I go to **x** via you" ↔ "I go to **x** via you."

3. How poisoned reverse is applied

Normal DV behavior (without poison)

Router **z** computes:

```
text
Dz(x) = 5 (best path is via y)
```

Then **z** sends:

```
text
To all neighbors (including y): Dz(x) = 5
```

So **y** might think:

```
text
Dy(x) = c(y, z) + Dz(x) = 1 + 5 = 6
```

and route to **x** via **z**.

If at the same time **z** is routing to **x** via **y**, they loop.

With poisoned reverse

Router **z** still computes:

```
text
Dz(x) = 5 (best path is via y)
```

But when **z** **sends its distance vector**:

- To neighbors *other than y*:
 - It advertises $Dz(x) = 5$ (real value).
- To neighbor **y**:
 - It advertises $Dz(x) = \infty$.

So from **y**'s perspective:

- “**z** says cost to **x** is ∞ , so I will **never** pick a path to **x** via **z**.”

Thus:

- As long as **z** routes to **x** via **y**,
- **y** will **never** route to **x** via **z**,
- So a 2-node loop **y** ↔ **z** cannot form.

Whenever **z** stops using **y** as next hop to **x**, it stops poisoning and starts advertising the real cost to **x** again.

4. What poisoned reverse fixes and what it doesn't

Poisoned reverse does fix:

- The classic 2-node loop:
 - $y \rightarrow x \text{ via } z$
 - $z \rightarrow x \text{ via } y$
- So it solves **count-to-infinity between two neighbors** for that destination.

Poisoned reverse does NOT fix:

- Loops that involve **three or more routers** in a cycle, e.g.:


```
A → x via B
B → x via C
C → x via A
```
- In such multi-node loops:
 - Each node is poisoning reverse **only to the neighbor it uses as next hop**.
 - There can still be a longer cycle of "I reach x via someone else" that poison doesn't break.

So poisoned reverse **reduces** the problem but doesn't eliminate all possible DV loops or all forms of count-to-infinity.

5. One-sentence summary

Poisoned reverse is a Distance-Vector trick where a router advertises a distance of ∞ for destination x to the neighbor it uses to reach x , so that neighbor will never try to route to x through it, preventing simple 2-node loops and speeding up convergence—but it cannot prevent loops involving 3 or more routers.



4.6 - intra-AS routing, RIP and OSPF (overview), inter-AS routing, BGP basics, BGP route selection, routing policy, why inter and intra-AS routing

from book :

4.6 Routing in the Internet

Having studied Internet addressing and the IP protocol, we now turn our attention to the Internet's routing protocols; their job is to determine the path taken by a data~~stx~~gram between source and destination. We'll see that the Internet's routing protocols embody many of the principles we learned earlier in this chapter. The link-state and distance-vector approaches studied in Sections 4.5.1 and 4.5.2 and the notion of an autonomous system considered in Section 4.5.3 are all central to how routing is done in today's Internet.

4.6 • ROUTING IN THE INTERNET 383

Learn from inter-AS protocol that subnet x is reachable via multiple gateways. Use routing info from intra-AS protocol to determine costs of least-cost paths to each of the gateways. Hot potato routing: Choose the gateway that has the smallest least cost. Determine from forwarding table the interface I that leads to least-cost gateway. Enter (x,I) in forwarding table.

Figure 4.33 [STX] Steps in adding an outside-AS destination in a router's forwarding table

Recall from Section 4.5.3 that an autonomous system (AS) is a collection of routers under the same administrative and technical control, and that all run the same routing protocol among themselves. Each AS, in turn, typically contains multiple subnets (where we use the term subnet in the precise, addressing sense in Section 4.4.2).

4.6.1 Intra-AS Routing in the Internet: RIP

An intra-AS routing protocol is used to determine how routing is performed within an autonomous system (AS). Intra-AS routing protocols are also known as interior gateway protocols. Historically, two routing protocols have been used extensively for routing within an autonomous system in the Internet: the Routing Information Protocol (RIP) and Open Shortest Path First (OSPF). A routing protocol closely related to OSPF is the IS-IS protocol [RFC 1142, Perlman 1999]. We first discuss RIP and then consider OSPF.

RIP was one of the earliest intra-AS Internet routing protocols and is still in widespread use today. It traces its origins and its name to the Xerox Network Systems (XNS) architecture. The widespread deployment of RIP was due in great part to its inclusion in 1982 in the Berkeley Software Distribution (BSD) version of UNIX supporting TCP/IP. RIP version 1 is defined in [RFC 1058], with a backward-compatible version 2 defined in [RFC 2453].

RIP is a distance-vector protocol that operates in a manner very close to the idealized DV protocol we examined in Section 4.5.2. The version of RIP specified in RFC 1058 uses hop count as a cost metric; that is, each link has a cost of 1. In the DV algorithm in Section 4.5.2, for simplicity, costs were defined between pairs of routers. In RIP (and also in OSPF), costs are actually from source router to a destination subnet. RIP uses the term hop, which is the number of subnets traversed along the shortest path from source router to destination subnet, including the destination subnet. Figure 4.34 illustrates an AS with six leaf subnets. The table in the figure indicates the number of hops from the source A to each of the leaf subnets. The maximum cost of a path is limited to 15, thus limiting the use of RIP to autonomous systems that are fewer than 15 hops in diameter. Recall that in DV protocols, neighboring routers exchange distance vectors with each other. The distance vector for any one router is the current estimate of the shortest path

distances from that router to the subnets in the AS. In RIP, routing updates are exchanged between neighbors approximately every 30 seconds using a RIP response message. The response message sent by a router or host contains a list of up to 25 destination subnets within the AS, as well as the sender's distance to each of those subnets. Response messages are also known as RIP advertisements.

Let's take a look at a simple example of how RIP advertisements work. Consider the portion of an AS shown in Figure 4.35. In this figure, lines connecting the routers denote subnets. Only selected routers (A, B, C, and D) and subnets (w, x, y, z) are labeled. Dotted lines indicate that the AS continues on; thus this autonomous system has many more routers and links than are shown.

Each router maintains a RIP table known as a routing table. A router's routing table includes both the router's distance vector and the router's forwarding table. Figure 4.36 shows the routing table for router D. Note that the routing table has three columns. The first column is for the destination subnet, the second column indicates the identity of the next router along the shortest path to the destination subnet, and the third column indicates the number of hops (that is, the number of subnets that have to be traversed, including the destination subnet) to get to the

destination subnet along the shortest path. For this example, the table indicates that to send a datagram from router D to destination subnet w, the datagram should first be forwarded to neighboring router A; the table also indicates that destination subnet w is two hops away along the shortest path. Similarly, the table indicates that subnet z is seven hops away via router B. In principle, a routing table will have one row for each subnet in the AS, although RIP version 2 allows subnet entries to be aggregated using route aggregation techniques similar to those we examined in Destination Subnet Next Router Number of Hops to Destination

wA 2

yB 2

zB 7

x— 1

.....

Figure 4.36 Routing table in router D before receiving advertisement from router A

Destination Subnet Next Router Number of Hops to Destination

zC 4

w— 1

x— 1

.....

Figure 4.37 Advertisement from router A

connecting link has gone down. When this happens, RIP modifies the local routing table and then propagates this information by sending advertisements to its neighboring routers (the ones that are still reachable). A router can also request information about its neighbor's cost to a given destination using RIP's request message.

Routers send RIP request and response messages to each other over UDP using port number 520. The UDP segment is carried between routers in a standard IP datagram. The fact that RIP uses a transport-layer protocol (UDP) on top of a network-layer protocol (IP) to implement network-layer functionality (a routing algorithm) may seem rather convoluted (it is!). Looking a little deeper at how RIP is implemented will clear this up.

Figure 4.39 sketches how RIP is typically implemented in a UNIX system, for

example, a UNIX workstation serving as a router. A process called `routed` (pronounced “route dee”) executes RIP, that is, maintains routing information and exchanges messages with routed processes running in neighboring routers. Because RIP is implemented as an application-layer process (albeit a very special one that is able to manipulate the routing tables within the UNIX kernel), it can send and receive messages over a standard socket and use a standard transport protocol. As shown, RIP is implemented as an application-layer protocol (see Chapter 2) running over UDP. If you’re interested in looking at an implementation of RIP (or the OSPF and BGP protocols that we will study shortly), see [Quagga 2012].

4.6.2 Intra-AS Routing in the Internet: OSPF

Like RIP, OSPF routing is widely used for intra-AS routing in the Internet. OSPF and its closely related cousin, IS-IS, are typically deployed in upper-tier ISPs whereas RIP is deployed in lower-tier ISPs and enterprise networks. The Open in OSPF indicates that the routing protocol specification is publicly available (for example, as opposed to Cisco’s EIGRP protocol). The most recent version of OSPF, version 2, is defined in RFC 2328, a public document.

OSPF was conceived as the successor to RIP and as such has a number of advanced features. At its heart, however, OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra least-cost path algorithm. With OSPF, a router constructs a complete topological map (that is, a graph) of the entire autonomous system. The router then locally runs Dijkstra’s shortest-path algorithm to determine a shortest-path tree to all subnets, with itself as the root node. Individual link costs are configured by the network administrator (see Principles and Practice: Setting OSPF Weights). The administrator might choose to set all link costs to

1, thus achieving minimum-hop routing, or might choose to set the link weights to be inversely proportional to link capacity in order to discourage traffic from using low-bandwidth links. OSPF does not mandate a policy for how link weights are set (that is the job of the network administrator), but instead provides the mechanisms (protocol) for determining least-cost path routing for the given set of link weights.

With OSPF, a router broadcasts routing information to all other routers in the autonomous system, not just to its neighboring routers. A router broadcasts link-state information whenever there is a change in a link’s state (for example, a change in cost or a change in up/down status). It also broadcasts a link’s state periodically (at least once every 30 minutes), even if the link’s state has not changed. RFC 2328 notes that “this periodic updating of link state advertisements adds robustness to the link state algorithm.” OSPF advertisements are contained in OSPF messages that are carried directly by IP, with an upper-layer protocol of 89 for OSPF. Thus, the OSPF protocol must itself implement functionality such as reliable message transfer and link-state broadcast. The OSPF protocol also checks that links are operational (via a HELLO message that is sent to an attached neighbor) and allows an OSPF router to obtain a neighboring router’s database of network-wide link state.

Some of the advances embodied in OSPF include the following:

- Security. Exchanges between OSPF routers (for example, link-state updates) can be authenticated. With authentication, only trusted routers can participate

388 CHAPTER 4 • THE NETWORK LAYER

in the OSPF protocol within an AS, thus preventing malicious intruders (or networking students taking their newfound knowledge out for a joyride) from injecting incorrect information into router tables. By default, OSPF packets between routers are not authenticated and could be forged. Two types of authentication can be configured—simple and MD5 (see Chapter 8 for a discussion on MD5 and authentication in general). With simple authentication, the

same password is configured on each router. When a router sends an OSPF packet, it includes the password in plaintext. Clearly, simple authentication is not very secure. MD5 authentication is based on shared secret keys that are configured in all the routers. For each OSPF packet that it sends, the router computes the MD5 hash of the content of the OSPF packet appended with the secret key. (See the discussion of message authentication codes in Chapter 7.) Then the router includes the resulting hash value in the OSPF packet. The receiving router, using the preconfigured secret key, will compute an MD5 hash of the packet and compare it with the hash value that the packet carries, thus verifying the packet's authenticity. Sequence numbers are also used with MD5 authentication to protect against replay attacks.

- Multiple same-cost paths. When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used (that is, a single path need not be chosen for carrying all traffic when multiple equal-cost paths exist).
- Integrated support for unicast and multicast routing. Multicast OSPF (MOSPF) [RFC 1584] provides simple extensions to OSPF to provide for multicast routing (a topic we cover in more depth in Section 4.7.2). MOSPF uses the existing OSPF link database and adds a new type of link-state advertisement to the existing OSPF link-state broadcast mechanism.
- Support for hierarchy within a single routing domain. Perhaps the most significant advance in OSPF is the ability to structure an autonomous system hierarchically. Section 4.5.3 has already looked at the many advantages of hierarchical routing structures. We cover the implementation of OSPF hierarchical routing in the remainder of this section.

An OSPF autonomous system can be configured hierarchically into areas. Each area runs its own OSPF link-state routing algorithm, with each router in an area broadcasting its link state to all other routers in that area. Within each area, one or more area border routers are responsible for routing packets outside the area. Lastly, exactly one OSPF area in the AS is configured to be the backbone area. The primary role of the backbone area is to route traffic between the other areas in the AS. The backbone always contains all area border routers in the AS and may contain nonborder routers as well. Inter-area routing within the AS requires that the packet be first routed to an area border router (intra-area routing), then routed through the backbone to the area border router that is in the destination area, and then routed to the final destination.

4.6.3 Inter-AS Routing: BGP

We just learned how ISPs use RIP and OSPF to determine optimal paths for source-destination pairs that are internal to the same AS. Let's now examine how paths are determined for source-destination pairs that span multiple ASs. The Border Gateway Protocol version 4, specified in RFC 4271 (see also [RFC 4274]), is the de facto standard inter-AS routing protocol in today's Internet. It is commonly referred to as BGP4 or simply as BGP. As an inter-AS routing protocol (see Section 4.5.3), BGP provides each AS a means to

1. Obtain subnet reachability information from neighboring ASs.
2. Propagate the reachability information to all routers internal to the AS.
3. Determine "good" routes to subnets based on the reachability information and on AS policy.

390 CHAPTER 4 • THE NETWORK LAYER

SETTING OSPF LINK WEIGHTS

Our discussion of link-state routing has implicitly assumed that link weights are set, a routing algorithm such as OSPF is run, and traffic flows according to the routing tables computed by the

LS algorithm. In terms of cause and effect, the link weights are given (i.e., they come first) and result (via Dijkstra's algorithm) in routing paths that minimize overall cost. In this viewpoint, link weights reflect the cost of using a link (e.g., if link weights are inversely proportional to capacity, then the use of high-capacity links would have smaller weights and thus be more attractive from a routing standpoint) and Dijkstra's algorithm serves to minimize overall cost.

In practice, the cause and effect relationship between link weights and routing paths may be reversed, with network operators configuring link weights in order to obtain routing paths that achieve certain traffic engineering goals [Fortz 2000, Fortz 2002]. For example, suppose a network operator has an estimate of traffic flow entering the network at each ingress point and destined for each egress point. The operator may then want to put in place a specific routing of ingress-to-egress flows that minimizes the maximum utilization over all of the network's links. But with a routing algorithm such as OSPF, the operator's main "knobs" for tuning the routing of flows through the network are the link weights. Thus, in order to achieve the goal of minimizing the maximum link utilization, the operator must find the set of link weights that achieves this goal. This is a reversal of the cause and effect relationship—the desired routing of flows is known, and the OSPF link weights must be found such that the OSPF routing algorithm results in this desired routing of flows.

PRINCIPLES IN PRACTICE

Most importantly, BGP allows each subnet to advertise its existence to the rest of the Internet. A subnet screams "I exist and I am here," and BGP makes sure that all the ASes in the Internet know about the subnet and how to get there. If it weren't for BGP, each subnet would be isolated—alone and unknown by the rest of the Internet.

BGP Basics

BGP is extremely complex; entire books have been devoted to the subject and many issues are still not well understood [Yannuzzi 2005]. Furthermore, even after having read the books and RFCs, you may find it difficult to fully master BGP without having practiced BGP for many months (if not years) as a designer or administrator of an upper-tier ISP. Nevertheless, because BGP is an absolutely critical protocol for the Internet—in essence, it is the protocol that glues the whole thing together—we need to acquire at least a rudimentary understanding of how it works. We begin by describing how BGP might work in the context of the simple example network we studied earlier in Figure 4.32. In this description, we build on our discussion of hierarchical routing in Section 4.5.3; we encourage you to review that material.

In BGP, pairs of routers exchange routing information over semipermanent TCP connections using port 179. The semi-permanent TCP connections for the network in Figure 4.32 are shown in Figure 4.40. There is typically one such BGP TCP connection for each link that directly connects two routers in two different ASes; thus, in Figure 4.40, there is a TCP connection between gateway routers 3a and 1c and another TCP connection between gateway routers 1b and 2a. There are also semipermanent BGP TCP connections between routers within an AS. In particular, Figure 4.40 displays a common configuration of one TCP connection for each pair of routers internal to an AS, creating a mesh of TCP connections within each AS. For each TCP connection, the two routers at the end of the connection are called BGP peers, and the TCP connection along with all the BGP messages sent over the

4.6 • ROUTING IN THE INTERNET 391

AS connection is called a BGP session. Furthermore, a BGP session that spans two ASes is called an external BGP (eBGP) session, and a BGP session between routers in the same AS is called an internal BGP (iBGP) session. In Figure 4.40, the eBGP sessions are shown with the long dashes; the iBGP sessions are shown with the short dashes. Note that BGP session lines in Figure 4.40 do not always correspond to the

physical links in Figure 4.32.

BGP allows each AS to learn which destinations are reachable via its neighboring ASs. In BGP, destinations are not hosts but instead are CIDRized prefixes, with each prefix representing a subnet or a collection of subnets. Thus, for example, suppose there are four subnets attached to AS2: 138.16.64/24, 138.16.65/24, 138.16.66/24, and 138.16.67/24. Then AS2 could aggregate the prefixes for these four subnets and use BGP to advertise the single prefix to 138.16.64/22 to AS1. As another example, suppose that only the first three of those four subnets are in AS2 and the fourth subnet, 138.16.67/24, is in AS3. Then, as described in the Principles and Practice in Section 4.4.2, because routers use longest-prefix matching for forwarding datagrams, AS3 could advertise to AS1 the more specific prefix 138.16.67/24 and AS2 could still advertise to AS1 the aggregated prefix 138.16.64/22.

Let's now examine how BGP would distribute prefix reachability information over the BGP sessions shown in Figure 4.40. As you might expect, using the eBGP session between the gateway routers 3a and 1c, AS3 sends AS1 the list of prefixes that are reachable from AS3; and AS1 sends AS3 the list of prefixes that are reachable from AS1. Similarly, AS1 and AS2 exchange prefix reachability information through their gateway routers 1b and 2a. Also as you may expect, when a gateway router (in any AS) receives eBGP-learned prefixes, the gateway router uses its iBGP sessions to distribute the prefixes to the other routers in the AS. Thus, all the routers in AS1 learn about AS3 prefixes, including the gateway router 1b. The gateway router 1b (in AS1) can therefore re-advertise AS3's prefixes to AS2. When a router (gateway or not) learns about a new prefix, it creates an entry for the prefix in its forwarding table, as described in Section 4.5.3.

4.6 • ROUTING IN THE INTERNET 393

any one of the routers receives the datagram, it is going to look for an entry in its forwarding table to determine on which outgoing port it should forward the datagram. Therefore, each of the routers needs to know about the existence of your company's /24 prefix (or some aggregate entry). How does a router become aware of your company's prefix? As we have just seen, it becomes aware of it from BGP! Specifically, when your company contracts with a local ISP and gets assigned a prefix (i.e., an address range), your local ISP will use BGP to advertise this prefix to the ISPs to which it connects. Those ISPs will then, in turn, use BGP to propagate the advertisement. Eventually, all Internet routers will know about your prefix (or about some aggregate that includes your prefix) and thus be able to appropriately forward datagrams destined to your Web and mail servers.

Path Attributes and BGP Routes

Having now a preliminary understanding of BGP, let's get a little deeper into it (while still brushing some of the less important details under the rug!). In BGP, an autonomous system is identified by its globally unique autonomous system number (ASN) [RFC 1930]. (Technically, not every AS has an ASN. In particular, a so-called stub AS that carries only traffic for which it is a source or destination will not typically have an ASN; we ignore this technicality in our discussion in order to better see the forest for the trees.) AS numbers, like IP addresses, are assigned by ICANN regional registries [ICANN 2012].

When a router advertises a prefix across a BGP session, it includes with the prefix a number of BGP attributes. In BGP jargon, a prefix along with its attributes is called a route. Thus, BGP peers advertise routes to each other. Two of the more important attributes are AS-PATH and NEXT-HOP:

- AS-PATH. This attribute contains the ASs through which the advertisement for the prefix has passed. When a prefix is passed into an AS, the AS adds its ASN to the AS-PATH

attribute. For example, consider Figure 4.40 and suppose that prefix 138.16.64/24 is first advertised from AS2 to AS1; if AS1 then advertises the prefix to AS3, AS-PATH would be AS2 AS1. Routers use the AS-PATH attribute to detect and prevent looping advertisements; specifically, if a router sees that its AS is contained in the path list, it will reject the advertisement. As we'll soon discuss, routers also use the AS-PATH attribute in choosing among multiple paths to the same prefix.

- Providing the critical link between the inter-AS and intra-AS routing protocols, the NEXT-HOP attribute has a subtle but important use. The NEXT-HOP is the router interface that begins the AS-PATH. To gain insight into this attribute, let's again refer to Figure 4.40. Consider what happens when the gateway router 3a in AS3 advertises a route to gateway router 1c in AS1 using eBGP. The route includes the advertised prefix, which we'll call x , and an AS-PATH to the prefix. This advertisement also includes the NEXT-HOP, which is the IP address of the router 3a interface that leads to 1c. (Recall that a router has multiple IP addresses, one for each of its interfaces.) Now consider what happens when router 1d learns about this route from iBGP. After learning about this route to x , router 1d may want to forward packets to x along the route, that is, router 1d may want to include the entry (x, l) in its forwarding table, where l is its interface that begins the least-cost path from 1d towards the gateway router 1c. To determine l , 1d provides the IP address in the NEXT-HOP attribute to its intra-AS routing module. Note that the intra-AS routing algorithm has determined the least-cost path to all subnets attached to the routers in AS1, including to the sub~~ISTX~~net for the link between 1c and 3a. From this least-cost path from 1d to the 1c-3a sub~~ISTX~~net, 1d determines its router interface l that begins this path and then adds the entry (x, l) to its forwarding table. Whew! In summary, the NEXT-HOP attribute is used by routers to properly configure their forwarding tables.

- Figure 4.41 illustrates another situation where the NEXT-HOP is needed. In this figure, AS1 and AS2 are connected by two peering links. A router in AS1 could learn

394 CHAPTER 4 • THE NETWORK LAYER

about two different routes to the same prefix x . These two routes could have the same AS-PATH to x , but could have different NEXT-HOP values corresponding to the different peering links. Using the NEXT-HOP values and the intra-AS routing algorithm, the router can determine the cost of the path to each peering link, and then

apply hot-potato routing (see Section 4.5.3) to determine the appropriate interface.

BGP also includes attributes that allow routers to assign preference metrics to the routes, and an attribute that indicates how the prefix was inserted into BGP at the origin AS. For a full discussion of route attributes, see [Griffin 2012; Stewart 1999; Halabi 2000; Feamster 2004; RFC 4271].

When a gateway router receives a route advertisement, it uses its import policy to decide whether to accept or filter the route and whether to set certain attributes such as the router preference metrics. The import policy may filter a route because the AS may not want to send traffic over one of the ASs in the route's AS-PATH.

The gateway router may also filter a route because it already knows of a preferable route to the same prefix.

BGP Route Selection

As described earlier in this section, BGP uses eBGP and iBGP to distribute routes to all the routers within ASs. From this distribution, a router may learn about more than one route to any one prefix, in which case the router must select one of the

4.6 • ROUTING IN THE INTERNET 395

Two peering possible routes. The input into this route selection process is the set of all routes that have been learned and accepted by the router. If there are two or more routes to the same prefix, then BGP sequentially invokes the following elimination rules until one

route remains:

- Routes are assigned a local preference value as one of their attributes. The local preference of a route could have been set by the router or could have been learned by another router in the same AS. This is a policy decision that is left up to the AS's network administrator. (We will shortly discuss BGP policy issues in some detail.) The routes with the highest local preference values are selected.
- From the remaining routes (all with the same local preference value), the route with the shortest AS-PATH is selected. If this rule were the only rule for route selection, then BGP would be using a DV algorithm for path determination, where the distance metric uses the number of AS hops rather than the number of router hops.
- From the remaining routes (all with the same local preference value and the same AS-PATH length), the route with the closest NEXT-HOP router is selected. Here, closest means the router for which the cost of the least-cost path, determined by the intra-AS algorithm, is the smallest. As discussed in Section 4.5.3, this process is called hot-potato routing.
- If more than one route still remains, the router uses BGP identifiers to select the route; see [Stewart 1999].

The elimination rules are even more complicated than described above. To avoid nightmares about BGP, it's best to learn about BGP selection rules in small doses Routing Policy Let's illustrate some of the basic concepts of BGP routing policy with a simple example. Figure 4.42 shows six interconnected autonomous systems: A, B, C, W, X, and Y.

It is important to note that A, B, C, W, X, and Y are ASs, not routers. Let's assume that autonomous systems W, X, and Y are stub networks and that A, B, and C are backbone provider networks. We'll also assume that A, B, and C, all peer with each other, and provide full BGP information to their customer networks. All traffic entering a stub network must be destined for that network, and all traffic leaving a stub network must have originated in that network. W and Y are clearly stub networks. X is a multi-homed stub network, since it is connected to the rest of the network via two different providers (a scenario that is becoming increasingly common in practice). However, like W and Y, X itself must be the source/destination of all traffic leaving/entering X. But how will this stub network behavior be implemented and enforced? How will X be prevented from forwarding traffic between B and C? This can easily be accomplished by controlling the manner in which BGP routes are advertised. In particular, X will function as a stub network if it advertises (to its neighbors B and C) that it has no paths to any other destinations except itself. That is, even though X may know of a path, say XCY, that reaches network Y, it will not advertise this path to B. Since B is unaware that X has a path to Y, B would never forward traffic destined to Y (or C) via X. This simple example illustrates how a selective route advertisement policy can be used to implement customer/provider routing relationships.

Let's next focus on a provider network, say AS B. Suppose that B has learned (from A) that A has a path AW to W. B can thus install the route BAW into its routing information base. Clearly, B also wants to advertise the path BAW to its customer, X, so that X knows that it can route to W via B. But should B advertise the path BAW to C? If it does so, then C could route traffic to W via CBAW. If A, B, and C are all backbone providers, than B might rightly feel that it should not have to shoulder the burden (and cost!) of carrying transit traffic between A and C. B might rightly feel that it is A's and C's job (and cost!) to make sure that C can route to/from A's customers via a direct connection between A and C. There are currently no official standards that govern how backbone ISPs route among themselves. However, a rule of thumb followed by commercial ISPs is that any traffic flowing across an ISP's backbone network must have either a source or a destination (or both) in a

network that is a customer of that ISP; otherwise the traffic would be getting a free ride on the ISP's network. Individual peering agreements (that would govern questions such as those raised above) are typically negotiated between pairs of ISPs and are often confidential; [Huston 1999a] provides an interesting discussion of peering agreements. For a detailed description of how routing policy reflects commercial relationships among ISPs, see [Gao 2001; Dimitriopoulos 2007]. For a discussion of BGP routing policies from an ISP standpoint, see [Caesar 2005b].

As noted above, BGP is the de facto standard for inter-AS routing for the public Internet. To see the contents of various BGP routing tables (large!) extracted from routers in tier-1 ISPs, see <http://www.routeviews.org>. BGP routing tables often contain tens of thousands of prefixes and corresponding attributes. Statistics about the size and characteristics of BGP routing tables are presented in [Potaroo 2012].

This completes our brief introduction to BGP. Understanding BGP is important because it plays a central role in the Internet. We encourage you to see the references [Griffin 2012; Stewart 1999; Labovitz 1997; Halabi 2000; Huitema 1998; Gao 2001; Feamster 2004; Caesar 2005b; Li 2007] to learn more about BGP

Nice, this is one of the most "whole-picture" sections in the chapter. I'll go through it in the same order as the book, but more story-like:

1. Big picture: what 4.6 is about

We already learned:

- **How routing works in theory:** LS vs DV, shortest paths, AS, etc.
- **How packets are forwarded:** forwarding tables, longest-prefix match.

Now 4.6 says: "Okay, how does the *actual Internet* do routing with millions of routers and many different organizations?"

Key idea:

- The Internet is split into many **Autonomous Systems (ASes)** – each AS is a network under one administrative control (an ISP, a university, a company, etc.).
- We need:
 - **Intra-AS routing** (inside one AS).
 - **Inter-AS routing** (between ASes).

Different goals → different protocols.

2. Intra-AS routing (inside one network)

Also called **interior gateway protocols (IGPs)**. Two main ones in the book:

- **RIP** – distance vector, simple, older.
- **OSPF** – link state, more powerful, widely used in big networks.

2.1 RIP (Routing Information Protocol) – overview

Type: Distance-Vector (DV) protocol, very close to the Bellman-Ford style you saw in 4.5.2.

- **Metric:** hop count
 - Every link = cost 1.
 - Distance = number of subnets (hops) to reach a destination subnet (including the destination subnet).
 - Max path cost = 15 → if >15, considered unreachable → RIP can only be used for **small-ish** ASes.
- **Destination:** not individual routers, but **destination subnets**.
- **What routers store:**
Each router has a **RIP routing table**:
 - Destination subnet
 - Next hop router
 - Number of hops to reach that subnet
- **How routers communicate:**
 - Every ~30 seconds, neighbors exchange **RIP Response messages** (a DV “advertisement”).
 - Each advertisement lists up to 25 subnets and the sender’s hop count to each.
 - Protocol runs over **UDP port 520** (RIP is implemented as an application – **routed** process – talking via UDP).
- **When things change (link fails, cost changes):**
 - Router updates its table.
 - Sends new advertisements to neighbors (DV behavior).
 - RIP inherits DV issues: slow convergence, count-to-infinity (mitigated but not eliminated).

Where RIP is used:

Smaller enterprise networks, small ISPs. Not ideal for huge, complex backbones.

2.2 OSPF (Open Shortest Path First) – overview

Type: Link-State (LS) protocol.

Core idea: Every router in the AS:

1. Learns the full **topology** and **link costs** inside the AS via link-state flooding.
2. Locally runs **Dijkstra's algorithm** to compute the shortest-path tree.
3. Builds its forwarding table from that.

Key points:

- **Metric:** configurable **link weights**.
 - Admin can set all links to cost 1 → shortest-hop routing.
 - Or set lower costs for high-capacity links → traffic prefers high-bandwidth paths.
 - OSPF doesn't dictate policy; it just uses whatever costs are configured.
- **How link-state is spread:**
 - Routers send **LSAs (Link-State Advertisements)** to all other routers in the AS.
 - They flood LSAs whenever:
 - A link's state changes (up/down, cost change), or
 - Periodically (e.g., at least every 30 minutes) even if no change → robustness.
 - OSPF packets are carried directly over **IP (protocol number 89)** (not over UDP/TCP), so OSPF must implement reliability and flooding itself.
- **Nice features in OSPF:**
 1. **Security:**
 - Optional authentication of OSPF packets:
 - Simple password (weak, in plaintext).
 - MD5-based authentication (stronger): hash(packet + secret key), used to verify that only legitimate routers send LSAs.
 2. **Multiple equal-cost paths (ECMP):**
 - If there are several paths with the same total cost, OSPF can use **all** of them (load balancing).
 3. **Integrated unicast + multicast:**
 - MOSPF (Multicast OSPF) extends OSPF to support multicast routing using the same topology database.
 4. **Hierarchy inside an AS:**
 - The AS can be divided into **areas**.
 - Within an area:
 - Each router runs OSPF LS, builds full map of that area only.
 - Special routers:
 - **Area Border Routers (ABRs):** connect areas, route traffic in/out of an area.
 - **Backbone area (Area 0):** central area that connects all other areas; ABRs connect into the backbone.
 - Routing:
 - Intra-area: route within an area using its topology.
 - Inter-area: from router → ABR → backbone → ABR of destination area → destination.

Where OSPF is used:

Larger enterprise networks and upper-tier ISPs.

3. Inter-AS routing (between networks): BGP

When traffic goes from one AS to another (e.g., from your ISP to Google's AS), we need **inter-AS routing**. That's what **BGP (Border Gateway Protocol)** does.

BGP is the glue of the global Internet.

3.1 BGP basics: sessions, peers, prefixes

- BGP runs between routers called **BGP peers**, over **TCP port 179**.
- A BGP TCP connection + the BGP messages over it = a **BGP session**.
- Two types of sessions:
 - **eBGP** (external BGP): between routers in *different* ASes (e.g., ISP A ↔ ISP B).
 - **iBGP** (internal BGP): between routers within the *same* AS (to propagate routes learned from outside).
- **Destinations in BGP:**
Not individual IPs, but **prefixes** (CIDR blocks), e.g., **138.16.64/22**.
 - A prefix ≈ one or more subnets.
- **What BGP does:**
 1. ASes tell their neighbors what prefixes they can reach (reachability).
 2. That information is propagated inside the AS via iBGP.
 3. Routers use this info + policies to choose "**good" routes** to each prefix.

Without BGP, your /24 network at some company would not be announced anywhere, and no one could reach you.

3.2 BGP attributes: AS-PATH and NEXT-HOP

When a router advertises a prefix, it includes **attributes**. A *route* in BGP = prefix + attributes. Two key ones:

1. AS-PATH

- A list of AS numbers that the advertisement has passed through.
- Example: AS2 → AS1 → AS3 carrying prefix P
 - At AS3, AS-PATH for P might look like: **AS2 AS1**.
- Uses:
 - Loop detection: if an AS sees itself already in AS-PATH, it rejects the route.
 - Routing choice: shorter AS-PATH is often preferred.

2. NEXT-HOP

- IP address of the router interface that begins the AS-PATH.
- Used to glue BGP (inter-AS) to the **intra-AS** routing protocol.
- Example:
 - AS3's gateway 3a advertises prefix x to AS1's gateway 1c via eBGP.
 - NEXT-HOP = IP of 3a's interface towards 1c.
 - Inside AS1, iBGP spreads this route to routers like 1d.
 - Router 1d asks its intra-AS protocol (e.g., OSPF): "What is the least-cost path to that NEXT-HOP address?"
 - That gives 1d the output interface I.
 - 1d installs forwarding entry: (**prefix x → interface I**).

This is where **hot-potato routing** appears: when there are multiple possible NEXT-HOPS to reach the same prefix, the router chooses the one with **smallest internal cost** (get rid of the "hot potato" as quickly/cheaply as possible).

4. BGP route selection: how it picks **one** route

A router may learn multiple routes to the same prefix (multiple AS-PATHs, different neighbors). It then has to choose **one** to use.

Rough steps (simplified):

1. Apply import policy

- Router may **filter** some routes (e.g., "never use AS Z"; "don't use this peer for transit").
- It may also **set attributes** (like local preference) based on business rules.

2. Among remaining routes, choose:

1. Highest local preference (LOCAL_PREF)

- Admin-configured: e.g., "prefer customer routes > peer > provider".

2. If tie, shortest AS-PATH

- Fewer AS hops.

3. If tie, closest NEXT-HOP

- Use intra-AS metric (OSPF cost) → this is **hot-potato routing**.

4. If tie, use tie-breakers (like router ID, etc.).

So BGP routing is not "pure shortest path"; it's **policy + economics + topology**.

5. Routing policy: why BGP is about business, not just distance

Example from the book: ASes A, B, C (backbone providers), and stub ASes W, X, Y.

- **Stub AS (W, Y):**
Only carries traffic that **starts or ends** in it. Not transit for others.
- **Multi-homed stub (X):**
Connected to two providers (B and C), but still doesn't want to carry traffic between B and C.

How to enforce this? **By controlling what routes you advertise.**

- X can:
 - Learn from C that there's a path $X \rightarrow C \rightarrow Y$.
 - But X **does NOT advertise** "I can reach Y" to B.
 - So B doesn't know it can go to Y via X; therefore B will never send $B \rightarrow X \rightarrow C \rightarrow Y$ traffic.
 - This keeps X as a stub, not a transit network.

For provider AS B:

- B learns from A that A has a path $A \rightarrow W$.
- B advertises route $B \rightarrow A \rightarrow W$ to its **customer X**, so X can reach W via B.
- But should B advertise $B \rightarrow A \rightarrow W$ to C?
If it does, C could send traffic to W via $C \rightarrow B \rightarrow A \rightarrow W$, making B a transit for $C \leftrightarrow A$ traffic.

ISPs usually follow a "no free ride" rule:

- Traffic that crosses an ISP's backbone should have source or destination in a customer network.
- They don't want to carry transit traffic for other big ISPs for free.

BGP routing policy = **selective advertising + attribute tweaking** to match business contracts (customer, peer, provider relationships).

6. Why do we need *both* intra-AS and inter-AS routing?

Why not run a single global LS or DV algorithm?

(1) Scale

- The Internet has **hundreds of thousands of routers** and **many millions of prefixes**.
- One global LS or DV:
 - LS: every router would have to know **full global topology**, flood LSAs globally → insane overhead.
 - DV: would have to iteratively converge over a huge graph → extremely slow, unstable.
- Splitting into ASes limits complexity:
 - Intra-AS protocols (RIP/OSPF) only see **inside their AS**.
 - BGP only deals with **prefix-level reachability and AS-paths**, not every router-level detail.

(2) Administrative autonomy

- Each organization wants to run its network **its own way**:
 - Choose its own IGP (RIP, OSPF, IS-IS, EIGRP, etc.).
 - Configure link weights, LS parameters, RIP timers, and internal policies.
- Inter-AS must be **standardized** so different organizations can interoperate → that's BGP.

(3) Different goals

- **Intra-AS (RIP/OSPF):**
 - Goal: performance, simplicity, fast convergence inside one admin domain.
 - Mostly shortest paths (or TE-tuned paths) and reliability.
- **Inter-AS (BGP):**
 - Goal: enforce **policy and economics** across organizations:
 - Who can transit through whom?
 - Prefer customer over peer over provider.
 - Avoid certain ASes for political/economic reasons.
 - Shortest path is not always desirable; business policy wins.

So:

IGPs (RIP/OSPF) = "How do *I* move packets within my own network efficiently?"

BGP = "How do different networks talk to each other while respecting contracts and policies?"

If you want, next we can do a quick "cheat sheet" style summary you can memorize for exam: 3–4 bullets per protocol (RIP, OSPF, BGP) + one diagram for intra vs inter-AS.



4.7 - Broadcast routing, uncontrolled and controlled flooding, spanning tree, multicast routing, multicast trees

from book:

4.7 Broadcast and Multicast Routing

Thus far in this chapter, our focus has been on routing protocols that support unicast (i.e., point-to-point) communication, in which a single source node sends a packet to a single destination node. In this section, we turn our attention to broadcast and multicast routing protocols. In broadcast routing, the network layer provides a service of delivering a packet sent from a source node to all other nodes in the network; multicast routing enables a single source node to send a copy of a packet to a subset of the other network nodes. In Section 4.7.1 we'll consider broadcast routing algorithms and their embodiment in routing protocols. We'll examine multi~~cast~~^{cast} routing in Section 4.7.2.

4.7 • BROADCAST AND MULTICAST ROUTING 399

4.7.1 Broadcast Routing Algorithms

Perhaps the most straightforward way to accomplish broadcast communication is

for the sending node to send a separate copy of the packet to each destination, as shown in Figure 4.43(a). Given N destination nodes, the source node simply makes N copies of the packet, addresses each copy to a different destination, and then transmits the N copies to the N destinations using unicast routing. This N -way unicast approach to broadcasting is simple—no new network-layer routing protocol, packet-duplication, or forwarding functionality is needed. There are, however, several drawbacks to this approach. The first drawback is its inefficiency. If the source node is connected to the rest of the network via a single link, then N separate copies of the (same) packet will traverse this single link. It would clearly be more efficient to send only a single copy of a packet over this first hop and then have the node at the other end of the first hop make and forward any additional needed copies. That is, it would be more efficient for the network nodes themselves (rather than just the source node) to create duplicate copies of a packet. For example, in Figure 4.43(b), only a single copy of a packet traverses the R1-R2 link. That packet is then duplicated at R2, with a single copy being sent over links R2-R3 and R2-R4. The additional drawbacks of N -way-unicast are perhaps more subtle, but no less important. An implicit assumption of N -way-unicast is that broadcast recipients, and their addresses, are known to the sender. But how is this information obtained? Most likely, additional protocol mechanisms (such as a broadcast membership or destination-registration protocol) would be required. This would add more overhead and, importantly, additional complexity to a protocol that had initially seemed quite simple. A final drawback of N -way-unicast relates to the purposes for which broadcast is to be used. In Section 4.5, we learned that link-state routing protocols use broadcast to disseminate the link-state information that is used to compute unicast routes. Clearly, in situations where broadcast is used to create and update unicast routes, it would be unwise (at best!) to rely on the unicast routing infrastructure to achieve broadcast.

400 CHAPTER 4 • THE NETWORK LAYER

R2

R3 R4

a. b.

Duplicate

Duplicate creation/transmission

R1

R3 R4

Duplicate

R1

R2

Figure 4.43 Source-duplication versus in-network duplication

Given the several drawbacks of N -way-unicast broadcast, approaches in which the network nodes themselves play an active role in packet duplication, packet forwarding, and computation of the broadcast routes are clearly of interest. We'll examine several such approaches below and again adopt the graph notation introduced in Section 4.5. We again model the network as a graph, $G = (N, E)$, where N is a set of nodes and a collection E of edges, where each edge is a pair of nodes from N . We'll be a bit sloppy with our notation and use N to refer to both the set of nodes, as well as the cardinality ($|N|$) or size of that set when there is no confusion.

Uncontrolled Flooding

The most obvious technique for achieving broadcast is a flooding approach in which the source node sends a copy of the packet to all of its neighbors. When a node receives a broadcast packet, it duplicates the packet and forwards it to all of its

neighbors (except the neighbor from which it received the packet). Clearly, if the graph is connected, this scheme will eventually deliver a copy of the broadcast packet to all nodes in the graph. Although this scheme is simple and elegant, it has a fatal flaw (before you read on, see if you can figure out this fatal flaw): If the graph has cycles, then one or more copies of each broadcast packet will cycle indefinitely. For example, in Figure 4.43, R2 will flood to R3, R3 will flood to R4, R4 will flood to R2, and R2 will flood (again!) to R3, and so on. This simple scenario results in the endless cycling of two broadcast packets, one clockwise, and one counterclockwise. But there can be an even more calamitous fatal flaw: When a node is connected to more than two other nodes, it will create and forward multiple copies of the broadcast packet, each of which will create multiple copies of itself (at other nodes with more than two neighbors), and so on. This broadcast storm, resulting from the endless multiplication of broadcast packets, would eventually result in so many broadcast packets being created that the network would be rendered useless. (See the homework questions at the end of the chapter for a problem analyzing the rate at which such a broadcast storm grows.)

Controlled Flooding

The key to avoiding a broadcast storm is for a node to judiciously choose when to flood a packet and (e.g., if it has already received and flooded an earlier copy of a packet) when not to flood a packet. In practice, this can be done in one of several ways.

In sequence-number-controlled flooding, a source node puts its address (or other unique identifier) as well as a broadcast sequence number into a broadcast packet, then sends the packet to all of its neighbors. Each node maintains a list of the source address and sequence number of each broadcast packet it has already received, duplicated, and forwarded. When a node receives a broadcast packet, it first checks whether the packet is in this list. If so, the packet is dropped; if not, the

4.7 • BROADCAST AND MULTICAST ROUTING 401

packet is duplicated and forwarded to all the node's neighbors (except the node from which the packet has just been received). The Gnutella protocol, discussed in Chapter 2, uses sequence-number-controlled flooding to broadcast queries in its overlay network. (In Gnutella, message duplication and forwarding is performed at the application layer rather than at the network layer.)

A second approach to controlled flooding is known as reverse path forwarding (RPF) [Dalal 1978], also sometimes referred to as reverse path broadcast (RPB). The idea behind RPF is simple, yet elegant. When a router receives a broadcast packet with a given source address, it transmits the packet on all of its outgoing links (except the one on which it was received) only if the packet arrived on the link that is on its own shortest unicast path back to the source. Otherwise, the router simply discards the incoming packet without forwarding it on any of its outgoing links. Such a packet can be dropped because the router knows it either will receive or has already received a copy of this packet on the link that is on its own shortest path back to the sender. (You might want to convince yourself that this will, in fact, happen and that looping and broadcast storms will not occur.) Note that RPF does not use unicast routing to actually deliver a packet to a destination, nor does it require that a router know the complete shortest path from itself to the source. RPF need only know the next neighbor on its unicast shortest path to the sender; it uses this neighbor's identity only to determine whether or not to flood a received broadcast packet.

Figure 4.44 illustrates RPF. Suppose that the links drawn with thick lines represent the least-cost paths from the receivers to the source (A). Node A initially broadcasts a source-A packet to nodes C and B. Node B will forward the source-A packet

it has received from A (since A is on its least-cost path to A) to both C and D. B will ignore (drop, without forwarding) any source-A packets it receives from any other nodes (for example, from routers C or D). Let us now consider node C, which will receive a source-A packet directly from A as well as from B. Since B is not on C's own shortest path back to A, C will ignore any source-A packets it receives from B. On the other hand, when C receives a source-A packet directly from A, it will forward the packet to nodes B, E, and F.

Spanning-Tree Broadcast

While sequence-number-controlled flooding and RPF avoid broadcast storms, they do not completely avoid the transmission of redundant broadcast packets. For example, in Figure 4.44, nodes B, C, D, E, and F receive either one or two redundant packets. Ideally, every node should receive only one copy of the broadcast packet.

Examining the tree consisting of the nodes connected by thick lines in Figure 4.45(a), you can see that if broadcast packets were forwarded only along links within this tree, each and every network node would receive exactly one copy of the broadcast packet—exactly the solution we were looking for! This tree is an example of a spanning tree—a tree that contains each and every node in a graph. More formally, a spanning tree of a graph $G = (N, E)$ is a graph $G^{(ST)} = (N, E^{(ST)})$ such that $E^{(ST)}$ is a subset of E , $G^{(ST)}$ is connected, $G^{(ST)}$ contains no cycles, and $G^{(ST)}$ contains all the original nodes in G . If each link has an associated cost and the cost of a tree is the sum of the link costs, then a spanning tree whose cost is the minimum of all of the graph's spanning trees is called (not surprisingly) a minimum spanning tree.

Thus, another approach to providing broadcast is for the network nodes to first construct a spanning tree. When a source node wants to send a broadcast packet, it sends the packet out on all of the incident links that belong to the spanning tree. A node receiving a broadcast packet then forwards the packet to all its neighbors in the spanning tree (except the neighbor from which it received the packet). Not only does spanning tree eliminate redundant broadcast packets, but once in place, the spanning tree can be used by any node to begin a broadcast, as shown in Figures 4.45(a) and 4.45(b). Note that a node need not be aware of the entire tree; it simply needs to know which of its neighbors in G are spanning-tree neighbors.

The main complexity associated with the spanning-tree approach is the creation and maintenance of the spanning tree. Numerous distributed spanning-tree algorithms have been developed [Gallager 1983, Gartner 2003]. We consider only one simple algorithm here. In the center-based approach to building a spanning tree, a center node (also known as a rendezvous point or a core) is defined. Nodes then unicast tree-join messages addressed to the center node. A tree-join message is forwarded using unicast routing toward the center until it either arrives at a node that already belongs to the spanning tree or arrives at the center. In either case, the path that the tree-join message has followed defines the branch of the spanning tree between the edge node that initiated the tree-join message and the center. One can think of this new path as being grafted onto the existing spanning tree.

Figure 4.46 illustrates the construction of a center-based spanning tree. Suppose that node E is selected as the center of the tree. Suppose that node F first joins the tree and forwards a tree-join message to E. The single link EF becomes the initial spanning tree. Node B then joins the spanning tree by sending its tree-join message to E.

Suppose that the unicast path route to E from B is via D. In this case, the tree-join message results in the path BDE being grafted onto the spanning tree. Node A next joins the spanning group by forwarding its tree-join message towards E. If A's unicast path to E is through B, then since B has already joined the spanning tree, the arrival of A's tree-join message at B will result in the AB link being immediately

grafted onto the spanning tree. Node C joins the spanning tree next by forwarding its tree-join message directly to E. Finally, because the unicast routing from G to E
 404 CHAPTER 4 • THE NETWORK LAYER

3
2
4
1
5

a. Stepwise construction of spanning tree b. Constructed spanning tree

A
B
C
D
G
F E
A
B
C
D
G
F E

Figure 4.46 [STX] Center-based construction of a spanning tree must be via node D, when G sends its tree-join message to E, the GD link is grafted onto the spanning tree at node D.

Broadcast Algorithms in Practice

Broadcast protocols are used in practice at both the application and network layers. Gnutella [Gnutella 2009] uses application-level broadcast in order to broadcast queries for content among Gnutella peers. Here, a link between two distributed application-level peer processes in the Gnutella network is actually a TCP connection. Gnutella uses a form of sequence-number-controlled flooding in which a 16-bit identifier and a 16-bit payload descriptor (which identifies the Gnutella message type) are used to detect whether a received broadcast query has been previously received, duplicated, and forwarded. Gnutella also uses a time-to-live (TTL) field to limit the number of hops over which a flooded query will be forwarded. When a Gnutella process receives and duplicates a query, it decrements the TTL field before forwarding the query. Thus, a flooded Gnutella query will only reach peers that are within a given number (the initial value of TTL) of application-level hops from the query initiator. Gnutella's flooding mechanism is thus sometimes referred to as limited-scope flooding.

A form of sequence-number-controlled flooding is also used to broadcast link-state advertisements (LSAs) in the OSPF [RFC 2328, Perlman 1999] routing algorithm, and in the Intermediate-System-to-Intermediate-System (IS-IS) routing algorithm [RFC 1142, Perlman 1999]. OSPF uses a 32-bit sequence number, as well as a 16-bit age field to identify LSAs. Recall that an OSPF node broadcasts LSAs for its attached links periodically, when a link cost to a neighbor changes, or when a link goes up/down. LSA sequence numbers are used to detect duplicate LSAs, but also serve a second important function in OSPF. With flooding, it is possible for an LSA generated by the source at time t to arrive after a newer LSA that was generated by the same source at time $t + d$. The sequence numbers used by the source node allow an older LSA to be distinguished from a newer LSA. The age field serves a purpose similar to that of a TTL value. The initial age field value is set to zero and is incremented at each hop as it is flooded, and is also

incremented as it sits in a router's memory waiting to be flooded. Although we have only briefly described the LSA flooding algorithm here, we note that designing LSA broadcast protocols can be very tricky business indeed. [RFC 789; Perlman 1999] describe an incident in which incorrectly transmitted LSAs by two malfunctioning routers caused an early version of an LSA flooding algorithm to take down the entire ARPAnet!

4.7.2 Multicast

We've seen in the previous section that with broadcast service, packets are delivered to each and every node in the network. In this section we turn our attention to multicast service, in which a multicast packet is delivered to only a subset of network nodes. A number of emerging network applications require the delivery of packets from one or more senders to a group of receivers. These applications include

4.7 • BROADCAST AND MULTICAST ROUTING 405

bulk data transfer (for example, the transfer of a software upgrade from the software developer to users needing the upgrade), streaming continuous media (for example, the transfer of the audio, video, and text of a live lecture to a set of distributed lecture participants), shared data applications (for example, a whiteboard or teleconferencing application that is shared among many distributed participants), data feeds (for example, stock quotes), Web cache updating, and interactive gaming (for example, distributed interactive virtual environments or multiplayer games).

In multicast communication, we are immediately faced with two problems—how to identify the receivers of a multicast packet and how to address a packet sent to these receivers. In the case of unicast communication, the IP address of the receiver (destination) is carried in each IP unicast datagram and identifies the single recipient; in the case of broadcast, all nodes need to receive the broadcast packet, so no destination addresses are needed. But in the case of multicast, we now have multiple receivers. Does it make sense for each multicast packet to carry the IP addresses of all of the multiple recipients? While this approach might be workable with a small number of recipients, it would not scale well to the case of hundreds or thousands of receivers; the amount of addressing information in the datagram would swamp the amount of data actually carried in the packet's payload field. Explicit identification of the receivers by the sender also requires that the sender know the identities and addresses of all of the receivers. We will see shortly that there are cases where this requirement might be undesirable.

For these reasons, in the Internet architecture (and other network architectures such as ATM [Black 1995]), a multicast packet is addressed using address indirection. That is, a single identifier is used for the group of receivers, and a copy of the packet that is addressed to the group using this single identifier is delivered to all of the multicast receivers associated with that group. In the Internet, the single identifier that represents a group of receivers is a class D multicast IP address. The group of receivers associated with a class D address is referred to as a multicast group. The multicast group abstraction is illustrated in Figure 4.47. Here, four hosts (shown in shaded color) are associated with the multicast group address of 226.17.30.197 and will receive all datagrams addressed to that multicast address. The difficulty that we must still address is the fact that each host has a unique IP unicast address that is completely independent of the address of the multicast group in which it is participating.

While the multicast group abstraction is simple, it raises a host (pun intended) of questions. How does a group get started and how does it terminate? How is the group address chosen? How are new hosts added to the group (either as senders or receivers)? Can anyone join a group (and send to, or receive from, that group) or is group membership restricted and, if so, by whom? Do group members know the identities of the other group members as part of the network-layer protocol? How

do the network nodes interoperate with each other to deliver a multicast datagram to all group members? For the Internet, the answers to all of these questions involve the Internet Group Management Protocol [RFC 3376]. So, let us next briefly consider IGMP and then return to these broader questions Internet Group Management Protocol. The IGMP protocol version 3 [RFC 3376] operates between a host and its directly attached router (informally, we can think of the directly attached router as the first-hop router that a host would see on a path to any other host outside its own local network, or the last-hop router on any path to that host), as shown in Figure 4.48. Figure 4.48 shows three first-hop multicast routers, each connected to its attached hosts via one outgoing local interface. This local interface is attached to a LAN in this example, and while each LAN has multiple attached hosts, at most a few of these hosts will typically belong to a given multicast group at any given time.

IGMP provides the means for a host to inform its attached router that an application running on the host wants to join a specific multicast group. Given that the scope of IGMP interaction is limited to a host and its attached router, another protocol is clearly required to coordinate the multicast routers (including the attached routers) throughout

4.7 • BROADCAST AND MULTICAST ROUTING 407

128.119.40.186

Key:

Router with attached

group member

Router with no attached

group member

128.34.108.63

128.34.108.60

128.59.16.20

mcast group

226.17.30.197

Figure 4.47 The multicast group: A datagram addressed to the group is delivered to all members of the multicast group

the Internet, so that multicast datagrams are routed to their final destinations. This latter functionality is accomplished by network-layer multicast routing algorithms, such as those we will consider shortly. Network-layer multicast in the Internet thus consists of two complementary components: IGMP and multicast routing protocols.

IGMP has only three message types. Like ICMP, IGMP messages are carried

(encapsulated) within an IP datagram, with an IP protocol number of 2. The membership_query message is sent by a router to all hosts on an attached interface

(for example, to all hosts on a local area network) to determine the set of all multicast groups that have been joined by the hosts on that interface. Hosts respond to a membership_query message with an IGMP membership_report message.

membership_report messages can also be generated by a host when an application first joins a multicast group without waiting for a membership_query message from the router. The final type of IGMP message is the leave_group message. Interestingly, this message is optional. But if it is optional, how does a router detect when a host leaves the multicast group? The answer to this question is

that the router infers that a host is no longer in the multicast group if it no longer responds to a membership_query message with the given group address. This is an example of what is sometimes called soft state in an Internet protocol. In a soft state protocol, the state (in this case of IGMP, the fact that there are hosts joined to a given multicast group) is removed via a timeout event (in this case, via a periodic membership_query message from the router) if it is not explicitly refreshed (in

this case, by a membership_report message from an attached host).

The term soft state was coined by Clark [Clark 1988], who described the notion of periodic state refresh messages being sent by an end system, and suggested that

408 CHAPTER 4 • THE NETWORK LAYER

Wide-area

multicast

routing

IGMP

IGMP

IGMP

IGMP

Figure 4.48 The two components of network-layer multicast in the Internet: IGMP and multicast routing protocols

with such refresh messages, state could be lost in a crash and then automatically restored by subsequent refresh messages—all transparently to the end system and without invoking any explicit crash-recovery procedures:

“...the state information would not be critical in maintaining the desired type of service associated with the flow. Instead, that type of service would be enforced by the end points, which would periodically send messages to ensure that the proper type of service was being associated with the flow. In this way, the state information associated with the flow could be lost in a crash without permanent disruption of the service features being used. I call this concept “soft state,” and it may very well permit us to achieve our primary goals of survivability and flexibility. . .”

It has been argued that soft-state protocols result in simpler control than hard-state protocols, which not only require state to be explicitly added and removed, but also require mechanisms to recover from the situation where the entity responsible for removing state has terminated prematurely or failed. Interesting discussions of soft state can be found in [Raman 1999; Ji 2003; Lui 2004].

Multicast Routing Algorithms

The multicast routing problem is illustrated in Figure 4.49. Hosts joined to the multicast group are shaded in color; their immediately attached router is also shaded in color. As shown in Figure 4.49, only a subset of routers (those with attached hosts that are joined to the multicast group) actually needs to receive the multicast traffic. In Figure 4.49, only routers A, B, E, and F need to receive the multicast traffic. Since none of the hosts attached to router D are joined to the multicast group and since router C has no attached hosts, neither C nor D needs to receive the multicast group traffic. The goal of multicast routing, then, is to find a tree of links that connects all of the routers that have attached hosts belonging to the multicast group. Multicast packets will then be routed along this tree from the sender to all of the hosts belonging to the multicast tree. Of course, the tree may contain routers that do not have attached hosts belonging to the multicast group (for example, in Figure 4.49, it is impossible to connect routers A, B, E, and F in a tree without involving either router C or D).

In practice, two approaches have been adopted for determining the multicast routing tree, both of which we have already studied in the context of broadcast routing, and so we will only mention them in passing here. The two approaches differ according to whether a single group-shared tree is used to distribute the traffic for all senders in the group, or whether a source-specific routing tree is constructed for each individual sender.

- Multicast routing using a group-shared tree. As in the case of spanning-tree broadcast, multicast routing over a group-shared tree is based on building a tree that

includes all edge routers with attached hosts belonging to the multicast group.

In practice, a center-based approach is used to construct the multicast routing tree, with edge routers with attached hosts belonging to the multicast group sending

4.7 • BROADCAST AND MULTICAST ROUTING 409

(via unicast) join messages addressed to the center node. As in the broadcast case, a join message is forwarded using unicast routing toward the center until it either arrives at a router that already belongs to the multicast tree or arrives at the center. All routers along the path that the join message follows will then forward received multicast packets to the edge router that initiated the multicast join. A critical question for center-based tree multicast routing is the process used to select the center.

Center-selection algorithms are discussed in [Wall 1980; Thaler 1997; Estrin 1997].

- Multicast routing using a source-based tree. While group-shared tree multicast routing constructs a single, shared routing tree to route packets from all senders, the second approach constructs a multicast routing tree for each source in the multicast group. In practice, an RPF algorithm (with source node x) is used to construct a multicast forwarding tree for multicast datagrams originating at source x. The RPF broadcast algorithm we studied earlier requires a bit of tweaking for use in multicast. To see why, consider router D in Figure 4.50. Under broadcast RPF, it would forward packets to router G, even though router G has no attached hosts that are joined to the multicast group. While this is not so bad for this case where D has only a single downstream router, G, imagine what would happen if there were thousands of routers downstream from D! Each of these thousands of routers would receive unwanted multicast packets.

410 CHAPTER 4 • THE NETWORK LAYER

Figure 4.49 Multicast hosts, their attached routers, and other routers

(This scenario is not as far-fetched as it might seem. The initial MBone [Casner 1992; Macedonia 1994], the first global multicast network, suffered from precisely this problem at first.). The solution to the problem of receiving unwanted multicast packets under RPF is known as pruning. A multicast router that receives multicast packets and has no attached hosts joined to that group will send a prune message to its upstream router. If a router receives prune messages from each of its downstream routers, then it can forward a prune message upstream.

Multicast Routing in the Internet

The first multicast routing protocol used in the Internet was the Distance-Vector Multicast Routing Protocol (DVMRP) [RFC 1075]. DVMRP implements source-based trees with reverse path forwarding and pruning. DVMRP uses an RPF algorithm with pruning, as discussed above. Perhaps the most widely used Internet multicast routing protocol is the Protocol-Independent Multicast (PIM) routing protocol, which explicitly recognizes two multicast distribution scenarios. In dense mode [RFC 3973], multicast group members are densely located; that is, many or most of the routers in the area need to be involved in routing multicast datagrams. PIM dense mode is a flood-and-prune reverse path forwarding technique similar in spirit to DVMRP.

4.7 • BROADCAST AND MULTICAST ROUTING 411

pkt not forwarded beyond receiving router

Figure 4.50 Reverse path forwarding, the multicast case

In sparse mode [RFC 4601], the number of routers with attached hosts

In sparse mode [RFC 4601], the number of routers with attached group members is small with respect to the total number of routers; group members are widely dispersed. PIM sparse mode uses rendezvous points to set up the multicast distribution tree. In

source-specific multicast (SSM) [RFC 3569, RFC 4607], only a single sender is allowed to send traffic into the multicast tree, considerably simplifying tree construction and maintenance.

When PIM and DVMP are used within a domain, the network operator can configure IP multicast routers within the domain, in much the same way that intra-domain unicast routing protocols such as RIP, IS-IS, and OSPF can be configured.

But what happens when multicast routes are needed between different domains? Is there a multicast equivalent of the inter-domain BGP protocol? The answer is (literally) yes. [RFC 4271] defines multiprotocol extensions to BGP to allow it to carry routing information for other protocols, including multicast information. The Multicast Source Discovery Protocol (MSDP) [RFC 3618, RFC 4611] can be used to connect together rendezvous points in different PIM sparse mode domains. An excellent overview of the current state of multicast routing in the Internet is [RFC 5110].

Let us close our discussion of IP multicast by noting that IP multicast has yet to

take off in a big way. For interesting discussions of the Internet multicast service

model and deployment issues, see [Diot 2000, Sharma 2003]. Nonetheless, in spite

of the lack of widespread deployment, network-level multicast is far from “dead.”

Multicast traffic has been carried for many years on Internet 2, and the networks

with which it peers [Internet2 Multicast 2012]. In the United Kingdom, the BBC is

engaged in trials of content distribution via IP multicast [BBC Multicast 2012]. At

the same time, application-level multicast, as we saw with PPLive in Chapter 2 and

in other peer-to-peer systems such as End System Multicast [Chu 2002], provides

multicast distribution of content among peers using application-layer (rather than

network-layer) multicast protocols. Will future multicast services be primarily

implemented in the network layer (in the network core) or in the application layer (at

the network’s edge)? While the current craze for content distribution via peer-to-peer

approaches tips the balance in favor of application-layer multicast at least in the near-term

future, progress continues to be made in IP multicast, and sometimes the race

ultimately goes to the slow and steady.

I'll go topic by topic and keep tying them back to simple pictures you can hold in your head.

1. Broadcast vs Multicast (quick recap)

- **Broadcast routing:**

One source → **all** nodes in the network.

- **Multicast routing:**

One (or more) source(s) → **a selected subset** of nodes (a group), not everyone.

Everything in 4.7 is about: “How do we efficiently send copies of a packet to many nodes without wasting bandwidth or looping forever?”

2. Broadcast routing

2.1 Naive “N-way unicast” (just for contrast)

Source sends **N separate unicast packets**, one to each destination.

Problems:

1. Inefficient on shared links

- If the source has only one outgoing link, that same link carries N copies of the same packet.

2. Sender needs a list of all receivers

- You'd need some extra protocol to know who all the receivers are.

3. You rely on unicast routing to *build the broadcast*, but unicast routing itself often *depends on broadcast* (e.g., LS flooding). That's circular.

So better idea: let the **network nodes** (routers) duplicate packets.

3. Uncontrolled flooding

Algorithm:

- Source sends the broadcast packet to **all neighbors**.
- Each node that receives a broadcast packet:
 - **Duplicates** it and
 - Forwards it to **all neighbors except the one it came from**.

If the graph is connected, every node eventually gets the packet.

Fatal problems:

- If the network has **cycles**, packets just keep circulating.
- When nodes have multiple neighbors, the number of packets explodes → **broadcast storm**.

So pure flooding is not usable in real networks.

4. Controlled flooding

Goal: still "simple flooding idea", but avoid infinite loops / storms.

4.1 Sequence-number-controlled flooding

Idea: "Forward each broadcast packet only once."

- Source **tags** each broadcast packet with:
 - Source ID (its address)
 - A **sequence number** (e.g., 1, 2, 3, ... for that source)
- Each router keeps a **list of (source, sequence_number)** it has already seen.
- When a router receives a broadcast packet:
 - If (source, seq) **already in list** → **drop** it (do not forward).

- If it's **new** → forward it to all neighbors (except incoming link) and **add to list**.

Effect:

No loops, no infinite storms. Each node forwards each broadcast exactly once.

Used e.g. in:

- **Gnutella** application-level broadcast (for queries).
 - **OSPF LSAs** (with some extra mechanisms).
-

4.2 Reverse Path Forwarding (RPF)

Idea: "Flood only along links that are consistent with shortest paths back to the source."

Each router already has **unicast routing** info that tells it the **next hop** to reach the source S.

RPF rule at router R for a packet from source S:

- Packet arrives on interface I.
- If I is the **same interface** that R would use to send a unicast packet **back to S**
 - This means packet came on R's **reverse shortest path** from S
 - **ACCEPT and forward** the broadcast packet to all other interfaces.
- If I is **not** on the reverse shortest path to S
 - **DROP** the packet (assume a "better" copy came or will come via correct interface).

Why this works:

- From the source's perspective, packets only flow "**outward**" along trees rooted at S.
- Any packet that tries to come in from a "wrong" direction is discarded.

Result:

- No infinite loops.
 - Still some redundant packets (nodes near branch points might see duplicates), but far fewer than uncontrolled flooding.
-

5. Spanning-tree broadcast

Even controlled flooding (sequence-number / RPF) can send **duplicate copies** to some routers.

Ideal goal: **Each router gets exactly one copy**. That's what a **spanning tree** gives us.

5.1 What is a spanning tree?

Given a graph $G = (N, E)$:

- A **spanning tree** is a subgraph $G' = (N, E')$ such that:
 - $E' \subseteq E$,
 - G' is **connected**,

- G' has **no cycles**,
- It contains **all nodes** in N .

So it's like: "Choose a subset of links so that all routers stay connected, but there are no loops."

If links have costs, the tree with minimum total cost is a **minimum spanning tree**, but for pure broadcast we just need *some* spanning tree.

5.2 How broadcast works on a spanning tree

Once a spanning tree is built:

- Source sends a broadcast packet on **all tree links** incident to it.
- Any node receiving the packet:
 - Forwards it only on **tree links** to its other tree neighbors (not to all neighbors).
- Because the tree is loop-free and connects all nodes:
 - Every node gets exactly **one** copy,
 - No redundant paths.

5.3 Center-based construction of a spanning tree

Main challenge: **build** and **Maintain** the tree in a distributed network.

One simple method = **center-based spanning tree**:

1. Choose a **center node** (also called **rendezvous point/core**).
2. When a router wants to be part of the tree, it sends a **tree-join message** (unicast) towards the center.
3. The join follows the normal **unicast route** to the center.
4. As the join packet travels, every router it passes through:
 - Marks the incoming interface as belonging to the tree.
5. The path from joining router to the first already-in-tree router (or the center) becomes a **branch** of the spanning tree.

Step by step:

- First joining router → path to center becomes initial tree.
- Next router joins → its path to center is "grafted" onto the existing tree.
- After enough joins, you have a full spanning tree (or at least a group-spanning tree for those who joined).

Once this tree exists, any node can start a broadcast simply by sending the packet along the **tree** (like above).

6. Multicast routing & multicast trees

6.1 Multicast vs broadcast

- **Broadcast:** deliver to **all routers/hosts**.
- **Multicast:** deliver only to routers/hosts that are part of a **multicast group**.

Key concept: **address indirection** → use a **group address** rather than listing all receivers.

- In IP: group = **Class D multicast address**.
- Many hosts can **join** the same group address → datagrams to that address go to all group members.

6.2 Who tells the router that a host has joined the group?

Between host and its **first-hop router**, this is handled by **IGMP** (Internet Group Management Protocol):

- Host says: "I want to join group G" → sends **membership_report** to router.
- Router periodically sends **membership_query** to learn which groups are still joined.
- If a host stops replying, router eventually assumes it left (this is **soft state**).

But IGMP is **only local** (host ↔ first-hop router).

We still need a **multicast routing protocol** among routers to build trees.

6.3 Multicast routing problem

Given a source and a multicast group:

- Only routers that have attached hosts in the group need to receive the traffic.
- Multicast routing must build a **multicast tree**:
 - Connects all routers with group members.
 - May pass through routers with no local members (transit routers).

The multicast tree defines how packets flow from sender(s) to all group members.

Two main approaches (similar to broadcast techniques you already saw):

7. Multicast trees: group-shared vs source-based

7.1 Group-shared tree

"**One shared tree per group**, used by all sources in that group."

- Typically built using a **center-based** method, similar to the spanning-tree construction:
 1. Choose a **rendezvous point (RP)** / center for the group.
 2. Routers with local group members send **join messages** towards the RP.
 3. Joins are forwarded by unicast routing until they hit someone already on the tree or the RP.

4. Those paths are grafted on → forming a **shared multicast tree** connecting all group members.
- All sources send to the RP, which then **distributes** the packets down the same tree to all members.

Pros:

- Only one tree per group → simpler.

Cons:

- Tree may not be optimal for each source (some may have long detours via the RP).

Used in **PIM sparse mode** (rendezvous-point based multicast).

7.2 Source-based trees (one tree per source)

"One multicast tree per source" in the group."

- For each source S, build a tree rooted at S.
- Use **Reverse Path Forwarding (RPF)** with S as the source:
 - Routers forward packets only if they arrive on the interface that lies on the shortest path **back to S**.
 - This creates a **source-specific forwarding tree**.

Problem: in pure broadcast RPF, packets go everywhere (to routers with no group members too).

For multicast, those routers would receive **unwanted traffic**.

Solution: **Pruning**.

RPF + Pruning

- If a router receives multicast packets for group G but:
 - It has **no local group members** and
 - It has **no downstream neighbors** needing G,
- It sends a **prune message** upstream to its parent (towards the source).
- Once the parent has prune messages from all downstream interfaces for that group:
 - It stops forwarding that group's traffic downstream,
 - And might send its own prune upstream.

Effect:

- Start with RPF flooding (like broadcast).
- Then prune away branches where no one wants the group.
- What remains is a **source-based multicast tree** only to routers that need the traffic.

Used in protocols like **DVMRP** and **PIM dense mode**.

7.3 Summary of multicast tree styles

- **Group-shared tree:**

- Single tree per group, typically **center-based** (rendezvous point).
- All senders share the same tree.

- **Source-based tree:**

- Different tree per source.
- Built using **RPF**, with **pruning** to remove unwanted branches.

Both reuse the ideas you already saw in broadcast routing (spanning tree, RPF, flooding + pruning).

If you want, next we can do a super-short exam-style cheat sheet like:

- “Define uncontrolled flooding / controlled flooding / RPF / spanning tree / group-shared tree / source-based tree in 1–2 lines each” so you can memorize them quickly.



Q13 In Distance Vector-based routing algorithms

- a) the routers send broadcast messages to all the routers in the autonomous system
- b) the routers send broadcast messages to all their directly connected routers
- c) the routers do not need to exchange messages.

Q14 In broadcast routing based on reverse path forwarding, the router forwards the packet

- a) in any case
- b) if the packet comes from their shortest path link to the source
- c) If the packet comes from a link part of the spanning tree

Q13

In Distance Vector (DV) routing, each router exchanges its distance vector **only with its directly connected neighbors**, not with every router in the AS.

Answer: (b) – *the routers send broadcast messages to all their directly connected routers*

Q14

In Reverse Path Forwarding (RPF) broadcast, a router **forwards** a received packet **only if** it arrived on the interface that lies on its **shortest path back to the source**. Otherwise it drops it.

Answer: (b) – *if the packet comes from their shortest path link to the source*