Atria Institute of Technology

Department of Information Science and Engineering

Bengaluru-560024

ACADEMIC YEAR: 2021-2022

ODD SEMESTER NOTES

Semester : 5th Semester

Subject Name : Computer Science and Engineering

Subject Code : 18CS52

Faculty Name : Ms. Uzma Sulthana

Module-1 Application Layer

- Numerous useful and entertaining applications have indeed been created due to the Internet's inception.
- The applications have been the driving force behind the Internet's success, motivating people in homes, schools, governments and businesses to make the Internet an integral part of their daily activities.
- Internet applications include the classic text-based applications that became popular in the 1970s and 1980s.
- Some of the applications are: **text email, remote access to computers, file transfers and newsgroups.**
- The World Wide Web (mid of 1990), encompassing Web surfing, search, and electronic commerce are popular applications.
- Instant messaging and P2P file sharing, the two applications were introduced at the end of the millennium.
- Since 2000, popular voice and video applications, including voice-over-IP (VoIP) and video conferencing over IP such as **Skype**, user-generated video distribution such as YouTube and movies on demand such as **Netflix are developed**.
- Most recently, a new generation of social networking applications, such as Facebook and Twitter have been emerged which have created engaging human networks on top of the Internet's network of routers and communication links.

1. Principles of Network Applications:

a. The core of network application development is writing programs that run on different end systems and communicate with each other over the network.

b. For example, in the Web application there are two distinct programs that communicate with each other: the browser program running in the user's host (desktop, laptop, smartphone etc.) and the Web server program running in the Web server host.

c. As another example, in a P2P file-sharing system there is a program in each host that participates in the file-sharing community. Here, the programs in the various hosts may be similar or identical.

d. Thus, when developing new application, we need to write software that will run on multiple end systems.

e. This software could be written, for example, in C, Java, or Python.

f. No need to write software that runs on network core devices, such as routers or link-layer switches.

g. Basic design—namely, confining application software to the end systems—as shown in Figure 2.1, has facilitated the rapid development and deployment of a vast array of network applications.

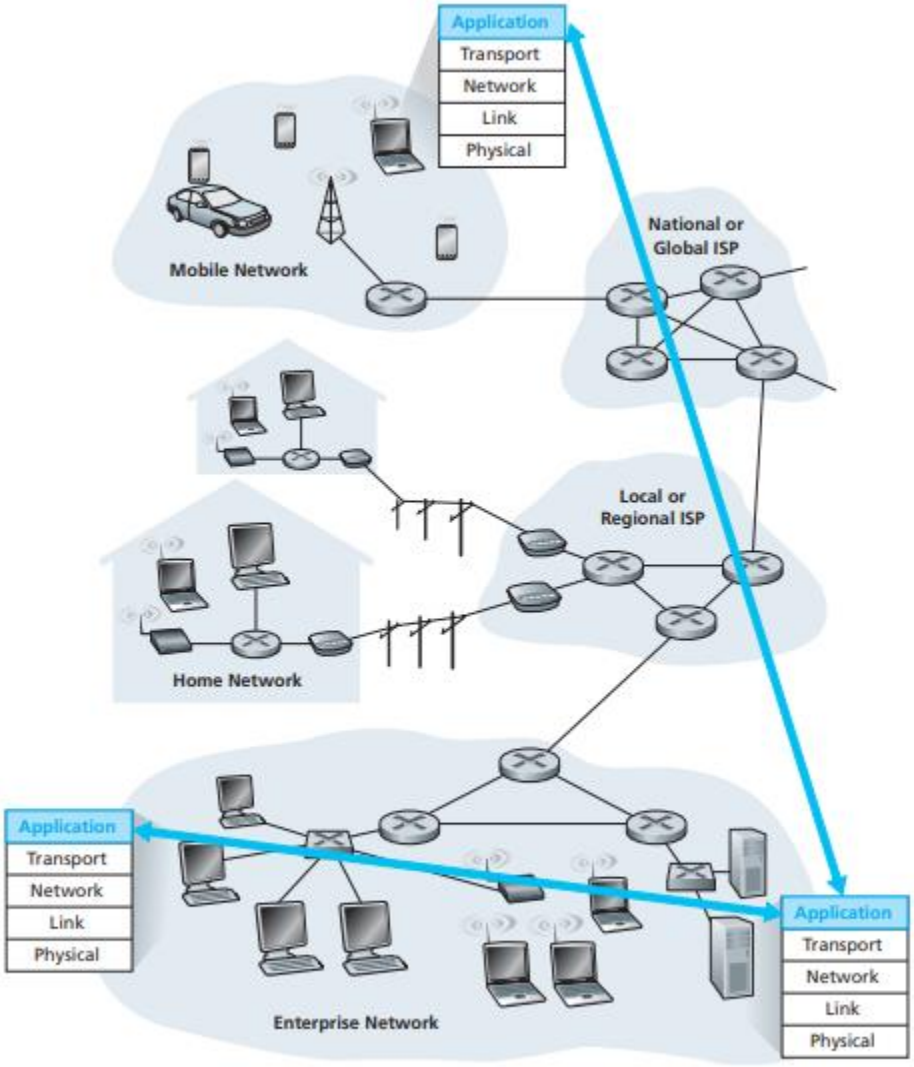


Figure 2.1 ♦ Communication for a network application takes place between end systems at the application layer

1.1 Network Application Architecture:

The application architecture is designed by the application developer and describes the structure of application over the various end systems.

There are two different network application architecture, they are

1. The client-server architecture.
2. The peer-to-peer (P2P) architecture.

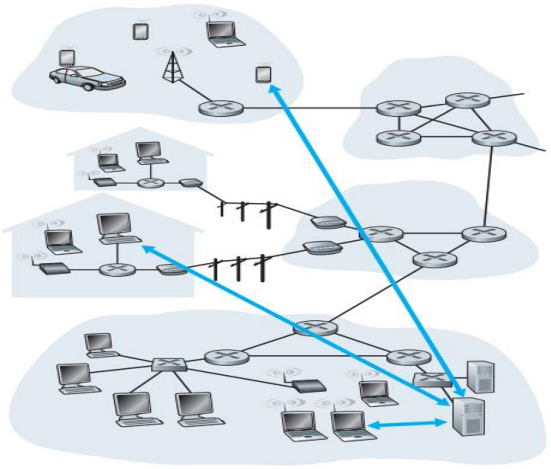
The client-server architecture.

- In client-server architecture, there is an always-on host, called the **server**, which provides services when it receives requests from many other hosts, called **clients**.
- Example: In Web application Web server services requests from browsers running on client hosts. When a Web server receives a request for an object from a client host, it responds by sending the requested object to the client host.

Characteristics of Client Server architecture:

- In client-server architecture, **clients do not directly communicate with each other**. For example, in the Web application, two browsers do not directly communicate.
- The server has a fixed, well-known address, called an **IP address**. Because the server has a fixed, well-known address, and because the server is always on, a client can always contact the server by sending a packet to the server's IP address.

Some of the better-known applications with a client-server architecture include the Web, FTP, Telnet, and e-mail.



a. Client-server architecture

The client-server architecture is shown in Figure (a) Above.

- In a client-server application, a single-server host is incapable of keeping up with all the requests from clients. For example, a popular social-networking site can quickly become overwhelmed if it has only one server handling all of its requests.
- For this reason, a **data center**, housing a large number of hosts, is often used to create a powerful virtual server.
- The most popular Internet services—such as **search engines** (e.g., Google and Bing), **Internet commerce** (e.g., Amazon and e-Bay), **Web-based email** (e.g., Gmail and Yahoo Mail), **social networking** (e.g., Facebook and Twitter)—employ one or more data centers.
- Eg. Google has **30 to 50 data centers** distributed around the world, which collectively handle search, YouTube, Gmail, and other services.
- A data center can have hundreds of thousands of servers, which must be powered and maintained.

Peer-to-Peer Architecture:

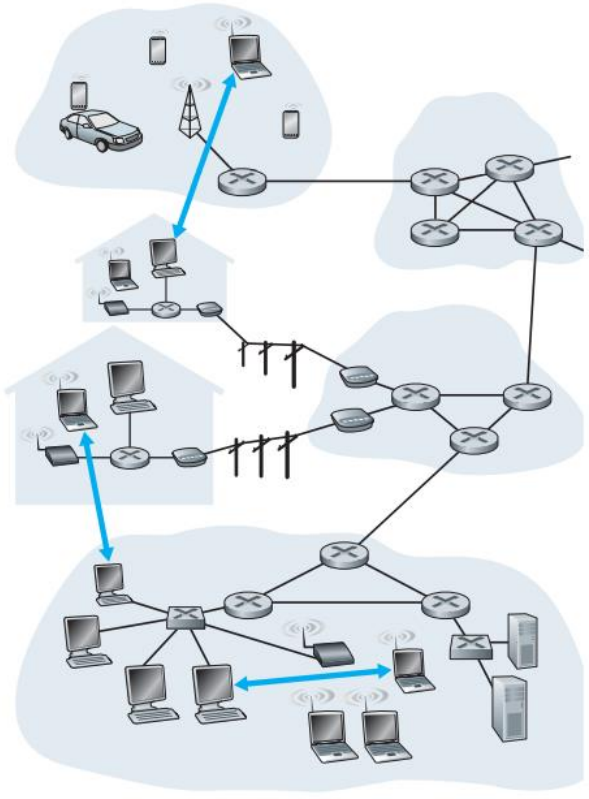
- In a P2P architecture, there is minimal dependence on dedicated servers in data centers.
- The application employs direct communication between pairs of intermittently connected hosts, called **peers.**
- The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices.
- Because the peers communicate without passing through a dedicated server, the architecture is called **peer-to-peer**.
- Most popular and traffic-intensive applications are based on P2P architectures.

- These applications include file sharing (e.g., BitTorrent), Internet Telephony (e.g., Skype), and IPTV (e.g., Kankan and PPstream).

Features:

- **Self-scalability:**
  - For example, in a P2P file-sharing application, although each peer generates workload by requesting files, each peer also adds service capacity to the system by distributing files to other peers.
- **Cost effective:**
  - P2P architectures are also cost effective, since they normally don't require significant server infrastructure and server bandwidth

The P2P architecture is illustrated in Figure (b) below



b. Peer-to-peer architecture

Future P2P applications face three major challenges:

1. **ISP Friendly**:

- Most residential ISPs have been dimensioned for "asymmetrical" bandwidth usage, that is, for much more downstream than upstream traffic.
- But P2P video streaming and file distribution applications shift upstream traffic from servers to residential ISPs, thereby putting significant stress on the ISPs.
- Future P2P applications need to be designed so that they are friendly to ISPs

2. **Security**.

   Because of their highly distributed and open nature, P2P applications can be a challenge to secure.

3. **Incentives**.

   The success of future P2P applications also depends on convincing users to volunteer bandwidth, storage, and computation resources to the applications, which is the challenge of incentive design.

1.2 Processes Communicating:

Process:

- A Process is a program or application under execution within a host system.
- When processes are running on the **1. same** or **2. different end system**, they can communicate with each other with inter process communication, using rules that are governed by the end system's operating system.
- Processes on two different end systems communicate with each other by exchanging **messages** across the computer network.
  - A sending process creates and sends messages into the network;
  - A receiving process receives these messages and possibly responds by sending messages back.

1. **Client and Server Process:**
   - A network-application consists of pairs of processes that send messages to other over a network.

     1) The process that initiates the communication is labelled as the **client.**

     2) The process that waits to be contacted to begin the session is labelled as the **server.**

     For example:

     1) In Web application, a client-browser process communicates with a Web-server-process.

2) In P2P file system, a file is transferred from a process in one peer to a process in another peer.

3) In the Web, a browser process initializes contact with a Web server process; hence the browser process is the client and the Web server process is the server.

4) In P2P file sharing, when Peer A asks Peer B to send a specific file, Peer A is the client and Peer B is the server in the context of this specific communication session.

2. Interface between the Process and the Computer Network Socket:

- Any message sent from one process to another must go through the underlying-network.
- A process sends messages into, and receives messages from, the network through a software interface called a **socket.**
- **Figure 2.3** illustrates socket communication between two processes that communicate over the Internet.
- As shown in this figure, a socket is the interface between the application layer and the transport layer within a host.
- It is also referred to as the **Application Programming Interface (API)** between the application and the network, since the socket is the programming interface with which network applications are built.
- The application developer has control of everything on the application-layer side of the socket but has little control of the transport-layer side of the socket.
- **For ex:** The application-developer can control:
  1) The choice of transport-protocol: TCP or UDP. (API Application Programming Interface)
  2) The ability to fix a few transport-layer parameters such as maximum-buffer & maximum-segment-sizes.

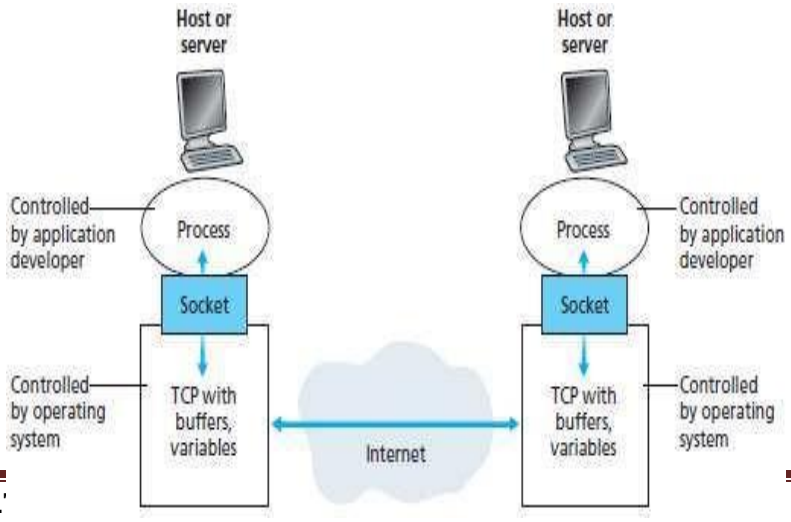


Figure 2.3: Application processes, sockets, and transport-protocol

3. **Addressing Process:**

For a process running on one host to send packets to a process running on another host, the receiving process needs to have an address.

- To identify the receiving process, two pieces of information need to be specified:

  (1) The address of the host

  (2) An identifier that specifies the receiving process in the destination host.

- In the Internet, the host is identified by its **IP address**.
- An IP address is a **32-bit** that uniquely identify the host.
- In addition to knowing the address of the host to which a message is destined, the sending process must also identify the receiving process running in the host.
- A destination port number serves this purpose.
- Popular applications have been assigned specific port numbers.

- **For example,**
  - A **Web server** is identified by port number **80**.
  - A **mail server process** (using the SMTP protocol) is identified by port number **25.**

1.3 Transport Services Available to Applications

- Many Networks, including the Internet, provide more than one transport-layer protocols for different applications.
- An application-developer should choose certain protocol according to the type of applications.
- Different protocols may provide different services.

1. Reliable Data Transfer:

- Packets can get lost within a computer network.
- **For example,** a packet can overflow a buffer in a router, or can be discarded by a host or router after having some of its bits corrupted.
- **For many applications**—such as electronic mail, file transfer, remote host access, Web document transfers, and financial applications—data loss can

have devastating consequences.

- Thus, to support these applications, something has to be done to **guarantee** that the data sent by one end of the application is delivered correctly and completely to the other end of the application.
- **Reliable means guaranteeing the data from the sender to the receiver is delivered correctly. For example: TCP provides reliable service to an application.**
- **Unreliable means the data from the sender to the receiver may never arrive. For example: UDP provides unreliable service to an application.**
- This may be acceptable for **loss-tolerant applications,** most notably **multimedia applications** such as conversational **audio/video** that can tolerate some amount of data loss.

2. Throughput

- Throughput is the rate at which the sending-process can deliver r bits/sec, to the receiving-process.
- Since other hosts are using the network, the throughput can fluctuate with time.
- Two types of applications:
  1. **Bandwidth Sensitive Applications**
     - These applications need a guarantee throughput.
       - For Example: Multimedia Applications.
     - Some transport-protocol provides guaranteed throughput at some specified rate **(r bits/sec)**

  2. **Elastic Applications**
- These applications may not need a guarantee throughput.
  - For Example: Electronic mail, File transfer & Web transfers.

3. Timinig

- A transport-layer protocol can also provide timing guarantees.
- For ex: guaranteeing every bit arrives at the receiver in less than 100 msec.
- Timing constraints are useful for real-time applications such as
  - Internet telephony
  - Virtual environments
  - Teleconferencing and
  - Multiplayer games

4. Security

- A transport-protocol can provide one or more security services.
-
- For example,

  1) In the sending host, a transport-protocol can encrypt all the transmitted-data.
  2) In the receiving host, the transport-protocol can decrypt the received-data.

- A transport protocol can provide security services like confidentiality, data integrity and endpoint authentication.

1.4 Transport Services Provided by the Internet:

- The Internet makes two transport-protocols available to applications, UDP and TCP.
- An application-developer who creates a new network-application must use either: UDP or TCP.
- Both UDP & TCP offers a different set of services to the invoking applications.
- Figure 2.4 shows the service requirements for some selected applications.

Application	Data Loss	Throughput Time	Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet-telephony/ Video-conferencing	Loss-tolerant	Audio: few kbps–1 Mbps Video: 10 kbps–5 Mbps	Yes: 100s of ms
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of ms
Instant messaging	No loss	Elastic	Yes and no

1. TCP Services

- An application using transport-protocol TCP, receives following 2 services.

1) Connection-Oriented Service

- Before the start of communication, client & server need to exchange control-information.
- This phase is called handshaking phase.
- Then, the two processes can send messages to each other over the connection.
- After the end of communication, the applications must tear down the connection.

2) Reliable Data Transfer Service

- The communicating processes must deliver all data sent without error & in the proper order.

- TCP also includes a congestion-control.

- The congestion-control throttles a sending-process when the network is congested.

2. UDP Services

- UDP is a lightweight transport-protocol, providing minimal services.
- UDP is connectionless, so there is no handshaking before the 2 processes start to communicate.
- UDP provides an unreliable data transfer service.
- Unreliable means providing no guarantee that the message will reach the receiving-process.
- Furthermore, messages that do arrive at the receiving-process may arrive out-of-order.
- UDP does not include a congestion-control.
- UDP can pump data into the network-layer at any rate.

3. Services Not Provided by Internet Transport Protocols

- Services not provided by today's Internet transport protocols.
- Does this mean that time sensitive applications such as Internet telephony cannot run in today's Internet?
- The answer is clearly no—the Internet has been hosting time-sensitive applications for many years.
- These applications often work fairly well because they have been designed to cope, to the greatest extent possible, with this lack of guarantee.
- Figure 2.5 indicates the transport protocols used by some popular Internet applications.

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	TCP

Internet-telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

Figure 2.5 Popular Internet applications, their application-layer protocols, and their underlying transport protocols.

- The e-mail, remote terminal access, the Web, and file transfer all use TCP.
- These applications have chosen TCP primarily because TCP provides reliable data transfer, guaranteeing that all data will eventually get to its destination.
- Because Internet telephony applications (such as Skype) can often tolerate some loss but require a minimal rate to be effective, developers of Internet telephony applications usually prefer to run their applications over UDP, thereby circumventing TCP's congestion control mechanism and packet overheads.
- But because many firewalls are configured to block (most types of) UDP traffic, Internet telephony applications often are designed to use TCP as a backup if UDP communication fails

1.5 Application-Layer Protocols

An application-layer protocol defines:

- The types of messages exchanged, for example, request messages and response messages
- The syntax of the various message types, such as the fields in the message and how the fields are delineated
- The semantics of the fields, that is, the meaning of the information in the fields
- Rules for determining when and how a process sends messages and responds to messages.

2. Web and HTTP

- The appearance of Web dramatically changed the Internet.
- Web has many advantages for a lot of applications.
- It operates on demand so that the users receive what they want when they want it.
- It provides an easy way for everyone make information available over the world.
- Hyperlinks and search engines help us navigate through an ocean of Web-sites.
- Forms, JavaScript, Java applets, and many other devices enable us to interact with pages and sites.

- The Web serves as a platform for many killer applications including YouTube, Gmail, and Facebook.

1. Overview of HTTP:

1. Web

- A web-page consists of objects (HTML -Hyper Text Markup Language).
- An object is a file such as an HTML file, a JPEG image, a Java applet, a video chip.
- The object is addressable by a single URL (URL- Uniform Resource Locator).
- Most Web-pages consist of a base HTML file & several referenced objects.
- For example:

  If a Web-page contains HTML text and five JPEG images; then the Web-page has six objects:
  1) Base HTML file and
  2) Five images.
- The base HTML file references the other objects in the page with the object's URLs.
- URL has 2 components:
  1) The hostname of the server that houses the object and
  2) The object's path name.
- **For example**:

  "http://www.someSchool.edu/someDepartment/picture.gif"

  In above URL,
  1) Hostname = "www.someSchool.edu "
  2) Path name = "/someDepartment/picture.gif".
- The web browsers implement the client-side of HTTP. For ex: Google Chrome, Internet Explorer
- The web-servers implement the server-side of HTTP. For ex: Apache

2. HTTP

- HTTP is Web's application-layer protocol (Figure 1.3) (HTTP -HyperText Transfer Protocol) is at the heart of Web.
- HTTP defines
  → how clients request Web-pages from servers and
  → how servers transfer Web-pages to clients.
- HTTP is implemented in **two programs**:
  a. client program and
  b. server program.
- The client program and server program, executing on different end systems, talk to each other by exchanging HTTP messages.
- HTTP defines the structure of these messages and how the client and server exchange the messages.
- The interaction between client and server in detail later, but the general idea is illustrated in Figure 2.6.
- When a user requests a Web-page, the browser sends HTTP request to the server.
- Then, the server responds with HTTP response that contains the requested-objects.
- HTTP uses TCP as its underlying transport-protocol.
- The HTTP client first initiates a TCP connection with the server.

- After connection setup, the browser and the server-processes access TCP through their sockets
- HTTP is a **stateless protocol.**
- Stateless means the server sends requested-object to client w/o storing state-info about the client.
- HTTP uses the client-server architecture:
  1) **Client**
     - Browser that requests receive and displays Web objects.
  2) **Server**
     - Web-server sends objects in response to requests.

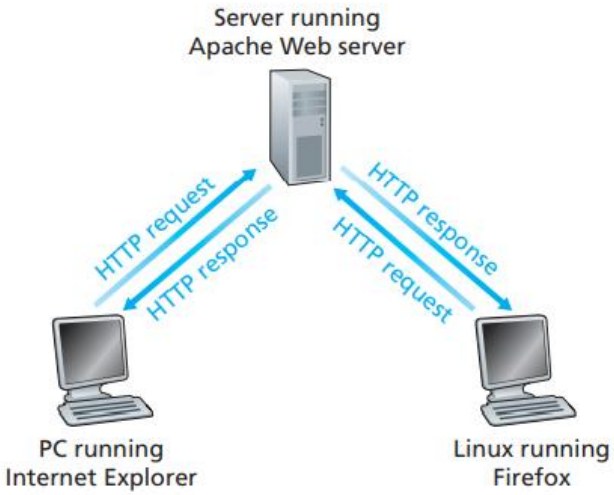


Figure 2.6 ♦ HTTP request-response behavior

2. Non-Persistent and Persistent Connections:

- In many internet applications, the client and server communicate for an extended period of time.
- When this client-server interaction takes place over TCP, a decision should be made:
  1) Should each request/response pair be sent over a separate TCP connection or
  2) Should all requests and their corresponding responses be sent over same TCP connection?
- These different connections are called **non-persistent connections** (1) or **persistent connections (2).**
- Default mode: HTTP uses persistent connections

1. HTTP with Non-Persistent Connections:

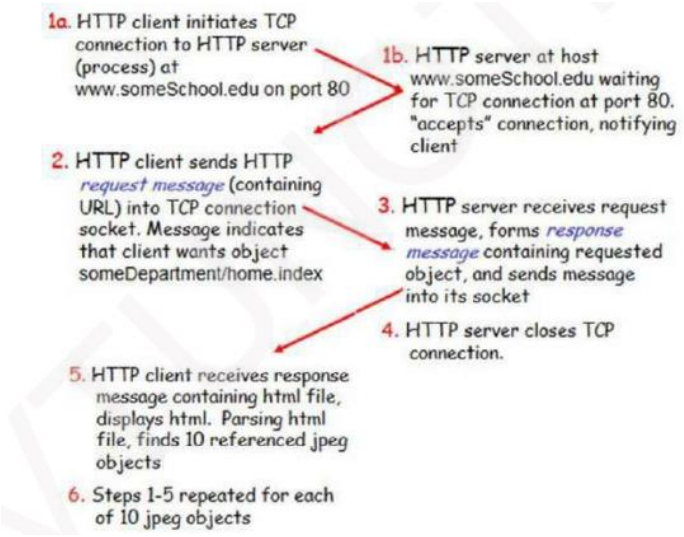
- A non-persistent connection is closed after the server sends the requested-object to the client.
- In other words, the connection is used exactly for one request and one response.

- For downloading multiple objects, multiple connections must be used.
- Suppose user enters URL: "http://www.someSchool.edu/someDepartment/home.index" • Assume above link contains **text and references to 10 jpeg images.**

Here is what happens:

1. The HTTP client process initiates a TCP connection to the server www.someSchool.edu on port number 80, which is the default port number for HTTP. Associated with the TCP connection, there will be a socket at the client and a socket at the server.
2. The HTTP client sends an HTTP request message to the server via its socket. The request message includes the path name /someDepartment/home.index. (We will discuss HTTP messages in some detail below.)
3. The HTTP server process receives the request message via its socket, retrieves the object /someDepartment/home.index from its storage (RAM or disk), encapsulates the object in an HTTP response message, and sends the response message to the client via its socket.
4. The HTTP server process tells TCP to close the TCP connection. (But TCP doesn't actually terminate the connection until it knows for sure that the client has received the response message intact.)
5. The HTTP client receives the response message. The TCP connection terminates. The message indicates that the encapsulated object is an HTML file. The client extracts the file from the response message, examines the HTML file, and finds references to the 10 JPEG objects.
6. The first four steps are then repeated for each of the referenced JPEG objects

Pictorial Representation of above steps:



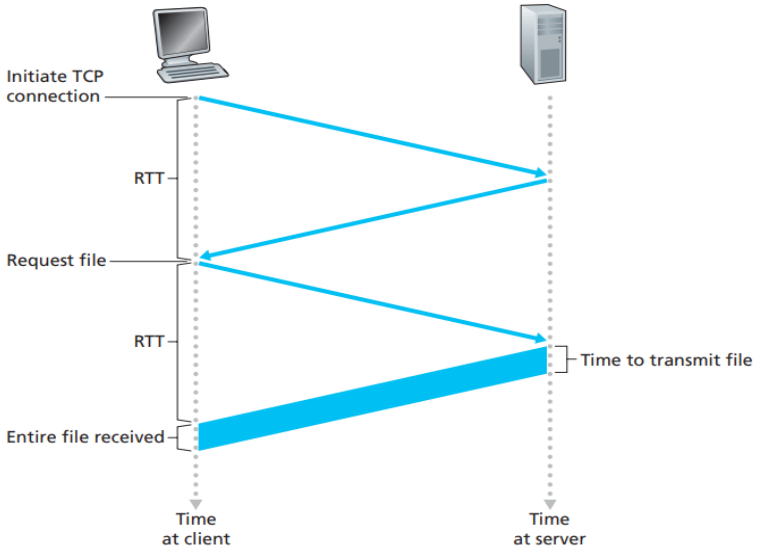


Figure 2.7 ♦ Back-of-the-envelope calculation for the time needed to request and receive an HTML file

- As shown in Figure 2.7, this causes the browser to initiate a TCP connection between the browser and the Web server; this involves a **"three-way handshake"**—the client sends a small TCP segment to the server, the server acknowledges and responds with a small TCP segment, and, finally, the client acknowledges back to the server.
- **RTT(Round Trip Time)** is the time taken for a packet to travel from client to server and then back to the client.
- The first two parts of the three-way handshake take one **RTT**.
- After completing the first two parts of the handshake, the client sends the HTTP request message combined with the third part of the three-way handshake (the acknowledgment) into the TCP connection.

- Once the request message arrives at the server, the server sends the HTML file into the TCP connection.
- This HTTP request/response eats up another RTT. Thus, roughly, the total response time is two RTTs plus the transmission time at the server of the HTML file.
- **The total response time is sum of following (Figure 2.7):**
  i) One RTT to initiate TCP connection (RTT -> Round Trip Time).
  ii) One RTT for HTTP request and first few bytes of HTTP response to return
  iii) File transmission time.
- i.e. **Total response time** = (i) + (ii) + (iii) = 1 RTT+ 1 RTT+ File transmission time **= 2(RTT) + File transmission time**

2. HTTP with Persistent Connections:

- **Problem with Non-Persistent Connections:**
  1. A new connection must be established and maintained for each requested-object.
    i) Hence, buffers must be allocated and state info must be kept in both the client and server.
    ii) This results in a significant burden on the server.
  2. Each object suffers a delivery delay of two RTTs:
    i) One RTT to establish the TCP connection and
    ii) One RTT to request and receive an object.
- **Solution:** Use persistent connections.
- With persistent connections, the server leaves the TCP connection open after sending responses.
- Hence, subsequent requests & responses b/w same client & server can be sent over same connection
- The server closes the connection only when the connection is not used for a certain amount of time.
- Default mode of HTTP: Persistent connections with pipelining.
- **Advantages:**
  1. This method requires only one RTT for all the referenced-objects.
  2. The performance is improved by 20%.

3. HTTP Message Format

Two types of HTTP messages: 1) Request-message and 2) Response-message.

1. HTTP Request Message:

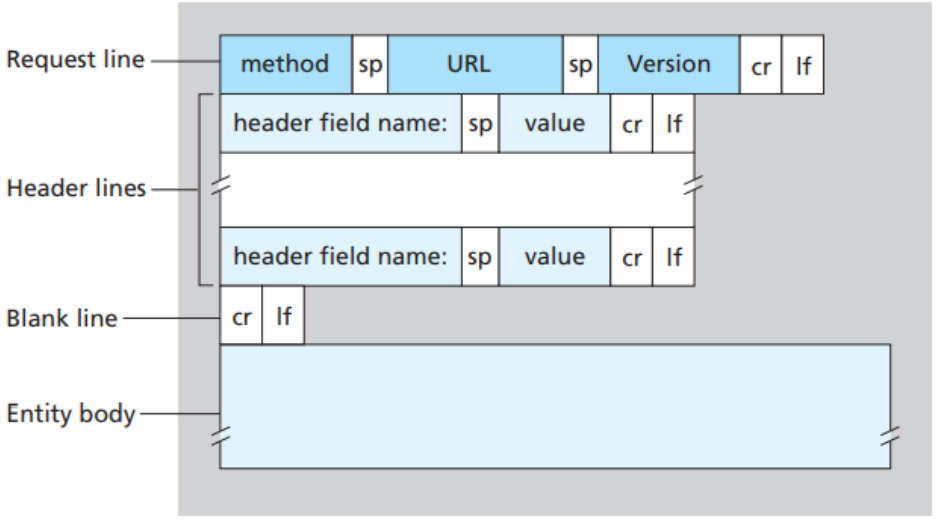


Figure 2.8 ♦ General format of an HTTP request message

- An example of request-message is as follows:

```
GET   /somedir/page.html    HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: eng
```

- The request-message contains 3 sections (Figure 2.8)
  1. Request-line
  2. Header-line and
  3. Carriage return.
- The first line of message is called the **request-line**.
- The subsequent lines are called the **header-lines**.
- The request-line contains 3 fields. The meaning of the fields is as follows:
  1. **Method**
     - "**GET**": This method is used when the browser requests an object from the server.
  2. **URL**
     - "/somedir/page.html": This is the object requested by the browser.
  3. **Version**
     - "HTTP/1.1": This is version used by the browser.

- The request-message contains 4 header-lines. The meaning of the header-lines is as follows:
    1. "**Hos**t: www.someschool.edu" specifies the host on which the object resides.
    2. "**Connection**: close" means requesting a non-persistent connection.
    3. "**User-agent**:Mozilla/5.0" means the browser used is the Firefox.
    4. "**Accept-language**:eng" means English is the preferred language.
- The method field can take following values: GET, POST, HEAD, PUT and DELETE.
    1. **GET** is used when the browser requests an object from the server.
    2. **POST** is used when the user fills out a form & sends to the server.
    3. **HEAD** is identical to GET except the server must not return a message-body in the response.
    4. **PUT** is used to upload objects to servers.
    5. **DELETE** allows an application to delete an object on a Web server.

2. HTTP Response Message:

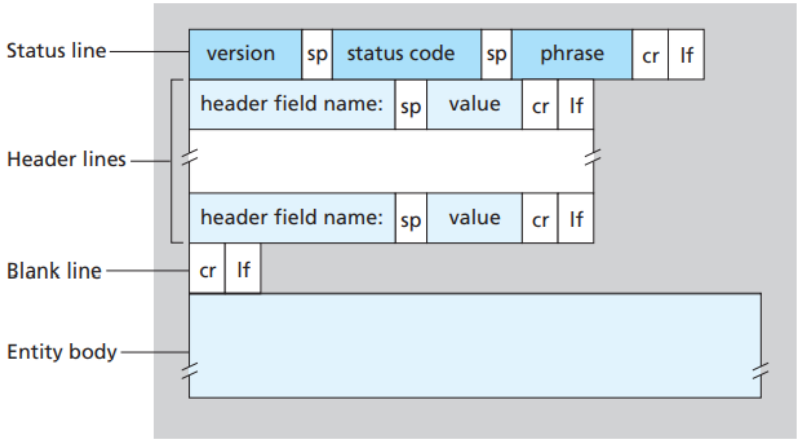


Figure 2.9 ♦ General format of an HTTP response message

- An example of response-message is as follows:

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

- The response-message contains 3 sections (Figure 2.9):
    1. Status line
    2. Header-lines and
    3. Data (Entity body).
- The status line contains 3 fields:
    1. Protocol version
    2. Status-code and
    3. Status message.
- Some common status-codes and associated messages include:
    1. **200 OK**: Standard response for successful HTTP requests.
    2. **400** Bad Request: The server cannot process the request due to a client error.
    3. **404** Not Found: The requested resource cannot be found.
    4. **505 HTTP Version Not Supported**: The requested HTTP protocol version is not supported by the server.
- The meaning of the Status line is as follows:

  "HTTP/1.1 200 OK": This line indicates the server is using HTTP/1.1 & that everything is OK.
- The response-message contains **6 header-lines**. The meaning of the header-lines is as follows:
    1. **Connection:** This line indicates browser requesting a non-persistent connection.
    2. **Date:** This line indicates the time & date when the response was sent by the server.
    3. **Server:** This line indicates that the message was generated by an Apache Web-server.
    4. **Last-Modified:** This line indicates the time & date when the object was last modified.
    5. **Content-Length:** This line indicates the number of bytes in the sent-object.
    6. **Content-Type:** This line indicates that the object in the entity body is HTML text.

3. User-Server Interaction: Cookies

- It is often desirable for a Web site to identify users, either because the server wishes to restrict user access or because it wants to serve content as a function of the user identity.
- For these purposes, HTTP uses **cookies.**
- Cookies refer to a small text file created by a Web-site that is stored in the user's computer.
- Cookies are stored either temporarily for that session only or

permanently on the hard disk.

- Cookies allow Web-sites to keep track of users.

- Cookie technology has **four components**:
  1. A cookie header-line in the HTTP response-message.
  2. A cookie header-line in the HTTP request-message.
  3. A cookie file kept on the user's end-system and managed by the user's browser.
  4. A back-end database at the Web-site.
- **Example:**

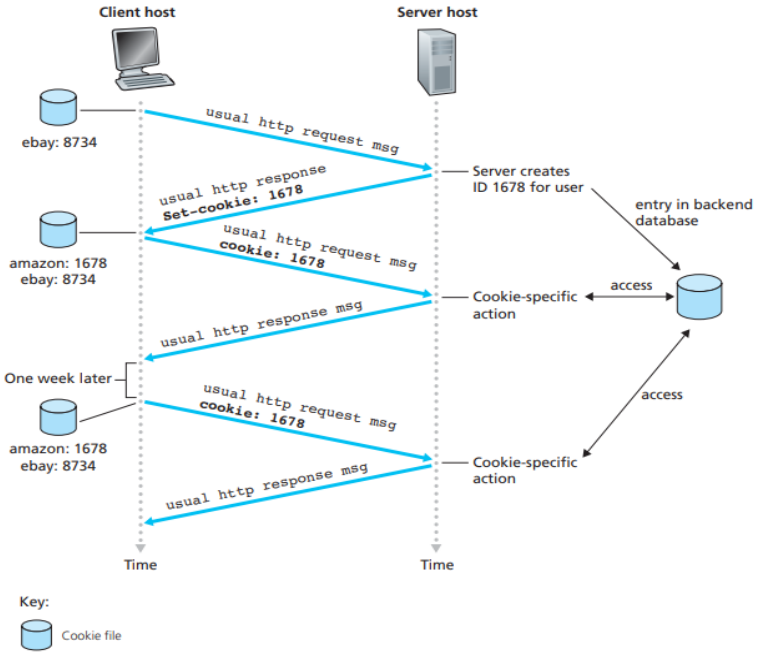


Figure 2.10 ♦ Keeping user state with cookies

Here is how it works (Figure 2.10)

1. Suppose a user, who always accesses the Web using Internet Explorer from her home PC, contacts Amazon.com for the first time.
2. Let us suppose that in the past he has already visited the eBay site.
3. When the request comes into the Amazon Web server, first time visits a site, the server
   - creates a unique identification number (**1678**) and
   - creates an entry in its back-end database by the identification number.
4. The server then responds to user's browser.
5. HTTP response includes **Set-cookie:** header which contains the identification number (1678)
6. The browser then stores the identification number into the cookie-file.

7. Each time the user requests a Web-page, the browser
   - extracts the identification number from the cookie file, and
   - puts the identification number in the HTTP request.
8. In this manner, the server is able to track user's activity at the web-site

4. Web Caching

- A Web cache—also called a proxy server—is a network entity that satisfies HTTP requests on the behalf of an origin Web server
- The Web-cache has disk-storage.
- The disk-storage contains copies of recently requested-objects.
- **As shown in Figure 2.11**, a user's browser can be configured so that all of the user's HTTP requests are first directed to the Web cache.
- Once a browser is configured, each browser request for an object is first directed to the Web cache.
- **As an example,** suppose a browser is requesting the object http://www.someschool.edu/campus.gif.

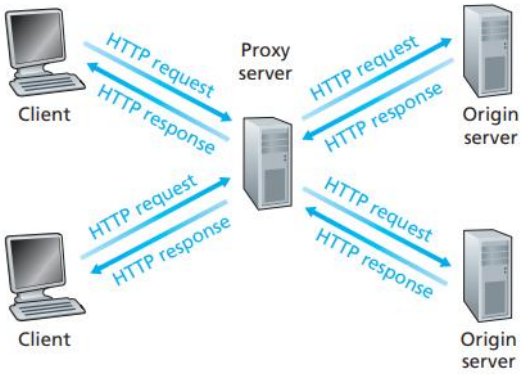


Figure 2.11 ♦ Clients requesting objects through a Web cache

- **Here is how it works (Figure 2.11):**
  1) The user's HTTP requests are first directed to the web-cache.
  2) If the cache has the object requested, the cache returns the requested-object to the client.
  3) If the cache does not have the requested-object, then the cache
     → connects to the original server and
     → asks for the object.
  4) When the cache receives the object, the cache
     → stores a copy of the object in local-storage and
     → sends a copy of the object to the client browser (over the existing TCP connection between the client browser and the Web cache).
- **A cache acts as both a server and a client at the same time.**
  1) The cache acts as a server when the cache
     → receives requests from a browser and

→ sends responses to the browser.

2) The cache acts as a client when the cache
   → requests to an original server and
   → receives responses from the origin server.

- **Advantages of caching**:
  1) To reduce response-time for client-request.
  2) To reduce traffic on an institution's access-link to the Internet.
  3) To reduce Web-traffic in the Internet.
- Typically, a Web cache is purchased and installed by an ISP.
- **For example**, a university might install a cache on its campus network and configure all of the campus browsers to point to the cache.
- Or a major residential ISP (such as AOL) might install one or more caches in its network and pre configure its shipped browsers to point to the installed caches.

5. The Conditional GET

- Conditional GET refers a mechanism that allows a cache to verify that the objects are up to date.
- An HTTP request-message is called conditional GET if
  1) Request-message uses the GET method and
  2) Request-message includes an If-Modified-Since: header-line.

Ex: First, on the behalf of a requesting browser, a proxy cache sends a request message to a Web server:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

Second, the Web server sends a response message with the requested object to the cache:

```
HTTP/1.1 200 OK
Date: Sat, 8 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 7 Sep 2011 09:23:24
Content-Type: image/gif
(data data data data data ...)
```

- The cache forwards the object to the requesting browser but also caches the object locally. Importantly, the cache also stores the last-modified date along with the object.
- **Third,** one week later, another browser requests the same object via the cache, and the object is still in the cache.
- Since this object may have been modified at the Web server in the past week, the cache performs an up-to-date check by issuing a conditional GET.
- Specifically, the cache sends:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 7 Sep 2011 09:23:24
```

- This conditional GET is telling the server to send the object only if the object has been modified since the specified date.
- Suppose the object has not been modified since 7 Sep 2011 09:23:24. Then, fourth, the Web server sends a response message to the cache:

```
HTTP/1.1 304 Not Modified
Date: Sat, 15 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
(empty entity body)
```

We see that in response to the conditional GET, the Web server still sends a response message but does not include the requested object in the response message.

3. File Transfer: FTP

- FTP is used by the local host to transfer files to or from a remote-host over the network.
- FTP is used for transferring file from one host to another host.
- In order for the user to access the remote account, the user must provide user identification and a password. After providing this authorization information, the user can transfer files from the local file system to the remote file system and vice versa.
- The user first provides the hostname of the remote host, causing the FTP client process in the local host to establish a TCP connection with the FTP server process in the remote host.
- The user then provides the user identification and password, which are sent over the TCP connection as part of FTP commands
- Once the server has authorized the user, the user copies one or more files stored in the local file system into the remote file system (or vice versa).
- As shown in Figure 2.14, the user interacts with FTP through an FTP user agent.

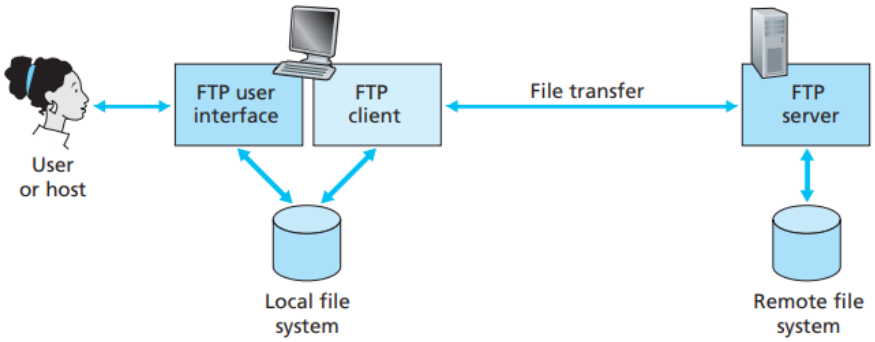


Figure 2.14 ♦ FTP moves files between local and remote file systems

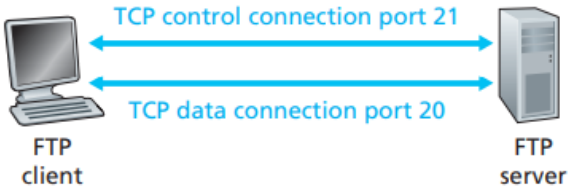


Figure 2.15 ♦ Control and data connections

- HTTP and FTP are both file transfer protocols and have many common characteristics; for example, they both run on top of TCP. However, the two application-layer protocols have some important differences.
- The most striking difference is that FTP uses two parallel TCP connections to transfer a file, a control connection and a data connection.
- **The control connection** is used for sending control information between the two hosts—information such as user identification, password, commands to change remote directory, and commands to "put" and "get" files.
- **The data connection** is used to actually send a file. Because FTP uses a separate control connection, FTP is said to send its control information out-of-band. HTTP, as you recall, sends request and response header lines into the same TCP connection that carries the transferred file itself.
- For this reason, HTTP is said to send its control information in-band.
- The FTP control and data connections are illustrated in Figure 2.15
- **Here is how it works:**
  1) When session starts, the client initiates a control-connection with the server on port 21.
  2) The client sends user-identity and password over the control-connection.
  3) Then, the server initiates data-connection to the client on port 20.
  4) FTP sends exactly one file over the data-connection and then closes the data-connection.

5) Usually, the control-connection remains open throughout the duration of the user-session.
6) But, a new data-connection is created for each file transferred within a session.

- During a session, the server must maintain the state-information about the user.
- **For example:**
  The server must keep track of the user's current directory.
- **Disadvantage:**
  Keeping track of state-info limits the no. of sessions maintained simultaneously by a server.

1. FTP Commands and Replies:

- The commands are sent from client to server.
- The replies are sent from server to client.
- The commands and replies are sent across the control-connection in **7-bit ASCII** format.
- Each command consists of **4-uppercase** ASCII characters followed by optional arguments.
- For example:
  1) **USER** username
     - Used to send the user identification to the server.
  2) **PASS** password
     - Used to send the user password to the server.
  3) **LIST**
     - Used to ask the server to send back a list of all the files in the current remote directory.
  4) **RETR** filename
     - Used to retrieve a file from the current directory of the remote-host.
  5) **STOR** filename
     - Used to store a file into the current directory of the remote-host.
- There is typically a one-to-one correspondence between the command that the user issues and the FTP command sent across the control connection.
- Each command is followed by a reply, sent from server to client.
- The replies **are three-digit numbers**, with an optional message following the number. This is similar in structure to the status code and phrase in the status line of the HTTP response message.
- Some typical replies, along with their possible messages, are as follows:
- For example:
  1) 331 Username OK, password required
  2) 125 Data-connection already open; transfer starting
  3) 425 Can't open data-connection
  4) 452 Error writing file

4. Electronic Mail in the Internet

- e-mail is an asynchronous communication medium in which people send and read messages.

- e-mail is fast, easy to distribute, and inexpensive.
- e-mail has features such as
  → messages with attachments
  → hyperlinks
  → HTML-formatted text and
  → embedded photos.

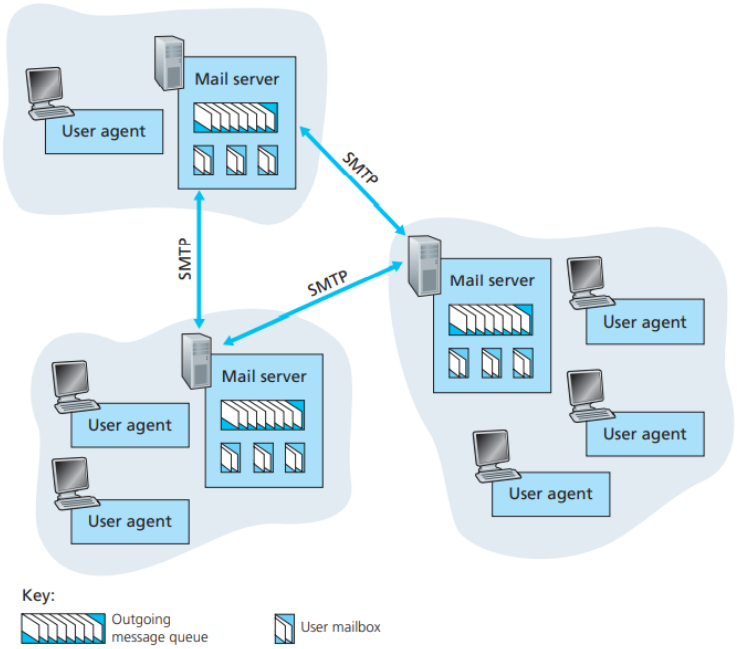


Figure 2.16 ♦ A high-level view of the Internet e-mail system

- Three major components of an e-mail system (Figure 2.16):

1) User Agents
- User-agents allow users to read, reply to, forward, save and compose messages.
- For example: Microsoft Outlook and Apple Mail

2) Mail Servers
- Mail-servers contain mailboxes for users.
- A message is first sent to the sender's mail-server.
- Then, the sender's mail-server sends the message to the receiver's mail-server.
- If the sender's server cannot deliver mail to receiver's server, the sender's server
  → holds the message in a message queue and
  → attempts to transfer the message later.

3) SMTP (Simple Mail Transfer Protocol)
- SMTP is an application-layer protocol used for email.
- SMTP uses TCP to transfer mail from the sender's mail-server to the recipient's mail-server.
- SMTP has two sides:
  1) A client-side, which executes on the sender's mail-server.
  2) A server-side, which executes on the recipient's mail-server.

- Both the client and server-sides of SMTP run on every mail-server.
- When a mail-server receives mail from other mail-servers, the mail-server acts as a server. When a mail-server sends mail to other mail-servers, the mail-server acts as a client.

1. SMTP:

- SMTP is the most important protocol of the email system.
- Three characteristics of SMTP (that differs from other applications):
  1. Message body uses 7-bit ASCII code only.
  2. Normally, no intermediate mail-servers used for sending mail.
  3. Mail transmissions across multiple networks through mail relaying.

To illustrate the basic operation of SMTP, let's walk through a common scenario. Suppose Alice wants to send Bob a simple ASCII message.

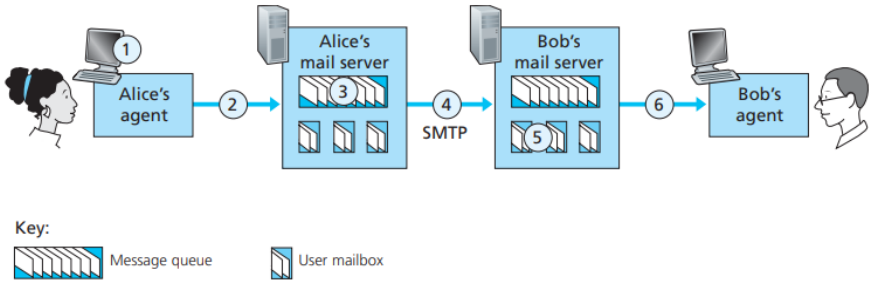


Figure 2.17 ♦ Alice sends a message to Bob

1. Alice invokes her user agent for e-mail, provides Bob's e-mail address (for example, bob@someschool.edu), composes a message, and instructs the user agent to send the message.
2. Alice's user agent sends the message to her mail server, where it is placed in a message queue.
3. The client side of SMTP, running on Alice's mail server, sees the message in the message queue. It opens a TCP connection to an SMTP server, running on Bob's mail server.
4. After some initial SMTP handshaking, the SMTP client sends Alice's message into the TCP connection.
5. At Bob's mail server, the server side of SMTP receives the message. Bob's mail server then places the message in Bob's mailbox.
6. Bob invokes his user agent to read the message at his convenience.

The scenario is summarized in Figure 2.17.
An example transcript of messages exchanged between an **SMTP client (C)** and an **SMTP server (S).**

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 HIello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

- In the example above, the client sends a message ("Do you like ketchup? How about pickles?") from mail server crepes.fr to mail server hamburger.edu.
- As part of the dialogue, the client issued five commands: HELO (an abbreviation for HELLO), MAIL FROM, RCPT TO, DATA, and QUIT.
- These commands are self-explanatory.
- The client also sends a line consisting of a single period, which indicates the end of the message to the server. (In ASCII jargon, each message ends with CRLF.CRLF, where CR and LF stand for carriage return and line feed, respectively.)
- The server issues reply to each command, with each reply having a reply code and some (optional) English-language explanation.
- We mention here that SMTP uses persistent connections: If the sending mail server has several messages to send to the same receiving mail server, it can send all of the messages over the same TCP connection.
- For each message, the client begins the process with a new MAIL FROM: crepes.fr, designates the end of message with an isolated period, and issues QUIT only after all messages have been sent.
- It is highly recommended that you use Telnet to carry out a direct dialogue with an SMTP server.
- To do this, issue telnet **serverName 25** where serverName is the name of a local mail server. When you do this, you are simply establishing a TCP connection between your local host and the mail server.
- After typing this line, you should immediately receive the 220 reply from the server.
- Then issue the SMTP commands HELO, MAIL FROM, RCPT TO, DATA, CRLF.CRLF, and QUIT at the appropriate times. It is also highly recommended.

2. Comparison of SMTP with HTTP

HTTP	SMTP
Pull Protocol- someone loads information on a Web server and users use HTTP to pull the information from the server at their convenience.	Push Protocol- the sending mail server pushes the file to the receiving mail server.
HTTP does not mandates data to be in 7-bit ASCII format.	SMTP requires each message, including the body of each message, to be in 7-bit ASCII format.
HTTP encapsulates each object in its own HTTP response message.	Internet mail places all of the message's objects into one message.

3. Mail Message Formats

- When an e-mail message is sent from one person to another, a header containing peripheral information precedes the body of the message.
- The header lines and the body of the message are separated by a blank line.
- Every header must have a **From:** header line and a **To:** header line; a header may include a **Subject:** header line as well as other optional header lines.

A typical message header looks like this

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.
```

4. Mail Access Protocols

- It is not realistic to run the mail-servers on PC & laptop. This is because
  → mail-servers must be always-on and
  → mail-servers must have fixed IP addresses
- **Problem:** How a person can access the email using PC or laptop?
- **Solution:** Use mail access protocols.

- Three mail access protocols:
  1. Post Office Protocol (POP)
  2. Internet Mail Access Protocol (IMAP) and
  3. HTTP.

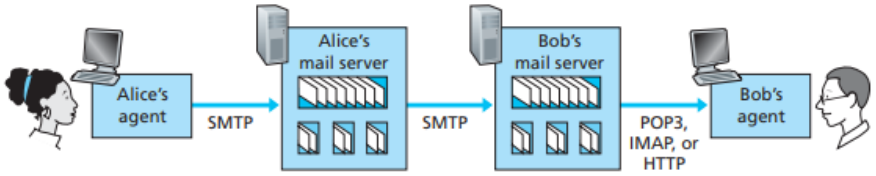


Figure 2.18 ♦ E-mail protocols and their communicating entities

- From the above **Figure 2.18**, provides a summary of the protocols that are used for Internet mail: SMTP is used to transfer mail from the sender's mail server to the recipient's mail server; SMTP is also used to transfer mail from the sender's user agent to the sender's mail server.
- A mail access protocol, such as POP3, is used to transfer mail from the recipient's mail server to the recipient's user agent.

1. Post Office Protocol (POP3)

- POP3 is an extremely simple mail access protocol.
- POP3 begins when the user agent (the client) opens a TCP connection to the mail server (the server) on port 110.
- **With the TCP connection established, POP3 progresses through three phases: authorization, transaction, and update.**
- During the **authorization phase**, the user agent sends a username and a password to authenticate the user.
- During the **transaction phase**, the user agent retrieves messages; also during this phase, the user agent can mark messages for deletion, remove deletion marks, and obtain mail statistics.
- The **update phase** occurs after the client has issued the quit command, ending the POP3 session; at this time, the mail server deletes the messages that were marked for deletion.
- In a POP3 transaction, the user agent issues commands, and the server responds to each command with a reply. There are two possible responses: +OK used by the server to indicate that the previous command was fine; and -ERR, used by the server to indicate that something was wrong with the previous command.
- The authorization phase has two principal commands: user <username> and pass <password>.

```
user bob
+OK
pass hungry
+OK user successfully logged on
```

- A user agent using POP3 can often be configured (by the user) to "**download and delete**" or to "**download and keep**."
- In the download-and-delete mode, the user agent will issue the list, retr, and delecommands.
- **Example:**

```
C: list
S: 1 498
S: 2 912
S: .
```

```
C: retr 1
S: (blah blah ...
S: .....
S .....blah)
S: .
C: dele 1
C: retr 2
S: (blah blah ...
S: .....
S .....blah)
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

- A problem with this download-and-delete mode is that the recipient cannot access mail messages from multiple machines.
- In the download-and keep mode, the user agent leaves the messages on the mail server after downloading them. In this case, user can reread messages from different machines.
- **Disadvantage:**

The user cannot manage the mails at remote mail-server. For ex: user cannot delete messages.

2. IMAP

- IMAP is another mail access protocol, which has more features than POP.
- An IMAP server will associate each message with a folder.
- When a message first arrives at server, the message is associated with recipient's **INBOX** folder
- Then, the recipient can
  - → move the message into a new, user-created folder
  - → read the message
  - → delete the message and
  - → search remote folders for messages matching specific criteria.
- An IMAP server maintains user state-information across IMAP sessions.
- IMAP permits a user-agent to obtain components of messages.
- For example, a user-agent can obtain just the message header of a message.

- With **POP3 access**, once user has downloaded his messages to the local machine, he can create mail folders and move the downloaded messages into the folders.
- User can then delete messages, move messages across folders, and search for messages (by sender name or subject).
- But this paradigm—namely, folders and messages in the local machine—poses a problem for the nomadic user, who would prefer to maintain a folder hierarchy on a remote server that can be accessed from any computer. This is not possible with POP3—the POP3 protocol does not provide any means for a user to create remote folders and assign messages to folders.
- To solve this and other problems, the IMAP protocol was invented. Like POP3, IMAP is a mail access protocol. It has many more features than POP3, but it is also significantly more complex.
- An IMAP server will associate each message with a folder; when a message first arrives at the server, it is associated with the recipient's INBOX folder.
- The recipient can then move the message into a new, user-created folder, read the message, delete the message, and so on.
- The IMAP protocol provides commands to allow users to create folders and move messages from one folder to another.
- IMAP also provides commands that allow users to search remote folders for messages matching specific criteria.
- Another important feature of IMAP is that it has commands that permit a user agent to obtain components of messages.
- **For example**, a user agent can obtain just the message header of a message or just one part of a multipart MIME message. This feature is useful when there is a low-bandwidth connection (for example, a slow-speed modem link) between the user agent and its mail server. With a low bandwidth connection, the user may not want to download all of the messages in its mailbox, particularly avoiding long messages that might contain, for example, an audio or video clip

3. Web-Based E-Mail

- HTTPs are now used for Web-based email accessing.
- The user-agent is an ordinary Web browser.
- The user communicates with its remote-server via HTTP.
- Now, Web-based emails are provided by many companies including Google, Yahoo etc.

5. DNS-The Internet's Directory Service

- DNS is an internet service that translates **domain-names into IP addresses**. For ex: the domain-name "www.google.com" might translate to IP address "198.105.232.4".
- Because domain-names are alphabetic, they are easier to remember for human being.
- But the Internet is really based on IP addresses (DNS -Domain Name System).

1. Services Provided by DNS:

- The DNS is
  1) A distributed database implemented in a hierarchy of DNS servers.
  2) An application-layer protocol that allows hosts to query the distributed database.
- DNS servers are often UNIX machines running the BIND software.
- The DNS protocol runs over UDP and uses port 53. (BIND -Berkeley Internet Name Domain)
- DNS is used by application-layer protocols such as HTTP, SMTP, and FTP.
- Assume a browser requests the URL www.someschool.edu/index.html.
- Next, the user's host must first obtain the IP address of www.someschool.edu

Example:

Consider what happens when a browser 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.

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

1. **Host aliasing:**

a. 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.

b. In this case, the hostname **relay1.westcoast.enterprise.com** is said to be a canonical hostname.
c. Alias hostnames, when present, are typically more mnemonic than canonical hostnames.
d. 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.

2. **Mail server aliasing:**

a. For obvious reasons, it is highly desirable that e-mail addresses be mnemonic.
b. **For example**, if Bob has an account with Hotmail, Bob's e-mail address might be as simple as **bob@hotmail.com**.
c. However, the hostname 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).
d. 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.

3. **Load distribution:**

a. DNS is also used to perform load distribution among replicated servers, such as replicated Web servers.

b. 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.
c. For replicated Web servers, a set of IP addresses is thus associated with one canonical hostname.

d. The DNS database contains this set of IP addresses.
e. 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.
f. 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.

2. Overview of How DNS Works

- Suppose that some application 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.
- 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.
- 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:**
  1. **A single point of failure.** If the DNS server crashes, so does the entire Internet!
  2. **Traffic volume.** A single DNS server would have to handle all DNS queries.
  3. **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.
  4. **Maintenance.**
     - The single DNS server would have to keep records for all Internet hosts.
     - This centralized database be huge, but it would have to be updated frequently to account for every new host.

2.1 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.
- There are **three classes of DNS servers**—root DNS servers, top-level domain (TLD) DNS servers, and authoritative DNS servers—organized in a hierarchy.

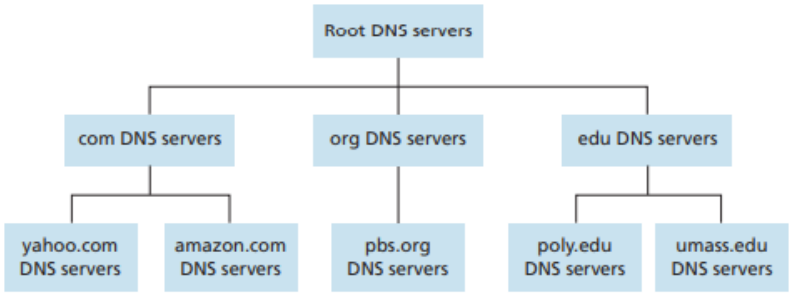


Figure 2.19 ♦ Portion of the hierarchy of DNS servers

1. **Root DNS servers**.

- In the Internet there are 13 root DNS servers (labeled A through M), most of which are located in North America.

- 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.
- Altogether, there are 247 root servers.

2. 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 in,uk, fr, ca.

3. Authoritative DNS servers:
- Every organization with publicly accessible hosts on the Internet must provide publicly accessible DNS records that map the names of those hosts to IP addresses.
- An organization's authoritative DNS server houses these DNS records.
- 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.

Two Types of Interaction:

1. Recursive Queries:

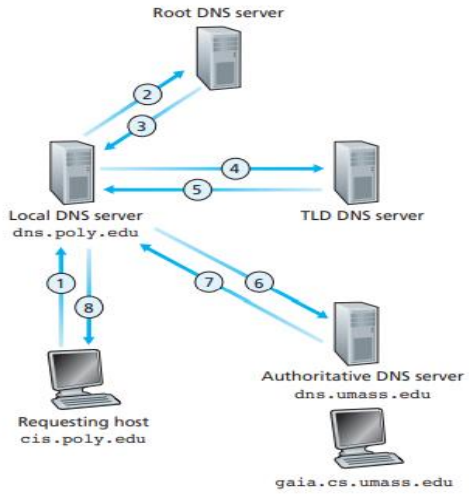


Figure 2.21 ♦ Interaction of the various DNS servers

- 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.

2. Iterative Queries:

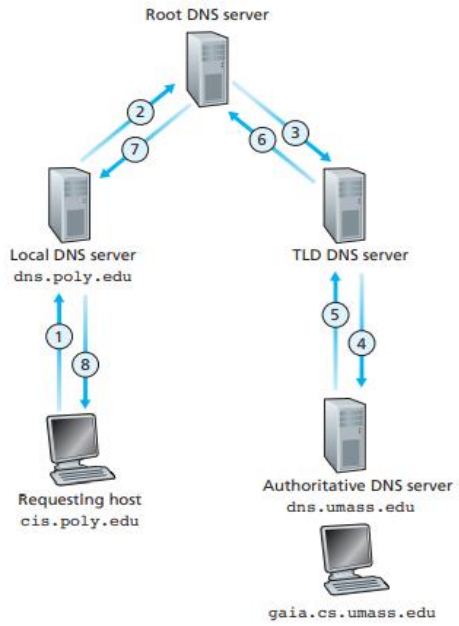


Figure 2.22 ♦ Recursive queries in DNS

- Figure 2.21: The query from the requesting host to the local DNS server is recursive, and the remaining queries are iterative.
- Here DNS query will be sent to Local DNS server, then to root server. Root server sends the IP address of TLD server.
- Now local DNS server sends query to TLD DNS server.
- TLD DNS server sends the IP address of authoritative DNS server to local DNS server.
- Now Local DNS server sends query to authoritative DNS server.
- Authoritative DNS server sends the IP address of host to local DNS server.
- Local DNS server sends it to the host.

2.2 DNS Caching

- In a query chain, when a DNS server receives a DNS reply it can cache the mapping in its local memory.

- 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).

2.3 DNS Records and Messages

- The DNS server stores resource-records (RRs).
- RRs provide hostname-to-IP address mappings.
- Each DNS reply message carries one or more resource-records.
- A resource-record is a 4-tuple that contains the following fields: **(Name, Value, Type, TTL)**
- TTL (time to live) determines when a resource should be removed from a cache.
- **The meaning of Name and Value depend on Type:**

1) If **Type=A,** then Name is a hostname and Value is the IP address for the hostname.

- Thus, a Type A record provides the standard hostname-to-IP address mapping.
- **For ex:** (relay1.bar.foo.com, 145.37.93.126, A)

2) If **Type=NS**, then

I. Name is a domain (such as foo.com) and
II. Value is the hostname of an authoritative DNS server.

- This record is used to route DNS queries further along in the query chain.
- **For ex:** (foo.com, dns.foo.com, NS) is a Type NS record.

3) If **Type=CNAME**, then Value is a canonical hostname for the alias hostname Name.

- This record can provide querying hosts the canonical name for a hostname.
- **For ex:** (foo.com, relay1.bar.foo.com, CNAME) is a CNAME record.

4) If **Type=MX**, Value is the canonical name of a mail-server that has an alias hostname Name.

- MX records allow the hostnames of mail-servers to have simple aliases.
- **For ex:** (foo.com, mail.bar.foo.com, MX) is an MX record

2.4 DNS Messages

- Two types of DNS messages: **1) query and 2) reply**.
- Both query and reply messages have the same format.

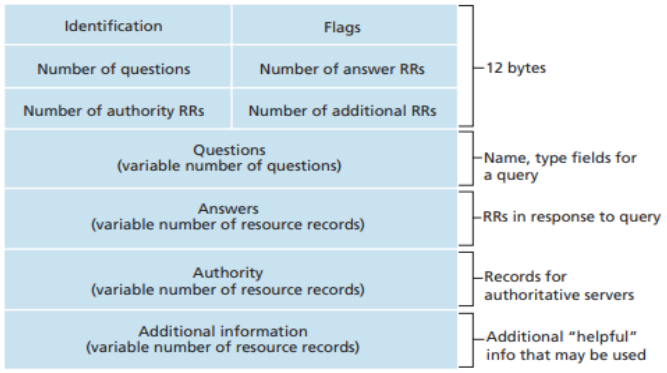


Figure 2.23 ♦ DNS message format

The various fields in a DNS message are as follows (Figure 2.23):

1) **Header Section**

- The first 12 bytes is the header-section.
- **This section has following fields:**

1. Identification:

- This field identifies the query.
- This identifier is copied into the reply message to a query.
- This identifier allows the client to match received replies with sent queries.

2. Flag:

- This field has following 3 flag-bits:

a) Query/Reply

This flag-bit indicates whether the message is a query (0) or a reply (1).

b) Authoritative

This flag-bit is set in a reply message when a DNS server is an authoritative-server.

c) Recursion Desired

This flag-bit is set when a client desires that the DNS server perform recursion.

3. Four Number-of-Fields:

These fields indicate the no. of occurrences of 4 types of data sections that follow the header.

2) **Question Section**

- This section contains information about the query that is being made.
- This section has following fields:

I. Name

This field contains the domain-name that is being queried.

II. Type

This field indicates the type of question being asked about the domain-name.

3) **Answer Section**

- This section contains a reply from a DNS server.
- This section contains the resource-records for the name that was originally queried.
- A reply can return multiple RRs in the answer, since a hostname can have multiple IP addresses.

4) **Authority Section**

This section contains records of other authoritative-servers.

5) **Additional Section**

This section contains other helpful records.

2.5 Inserting Records into the DNS

- Suppose you have just created an exciting new startup company called Network Utopia.
- The first thing you'll surely want to do is register the domain name networkutopia.com at a registrar.
- A registrar is a commercial entity that verifies the uniqueness of the domain name, enters the domain name into the DNS database (as discussed below), and collects a small fee from you for its services.
- For the primary authoritative server for networkutopia.com, the registrar would insert the following two resource records into the DNS system:

(networkutopia.com, dns1.networkutopia.com, NS)

(dns1.networkutopia.com, 212.212.212.1, A)

6. Peer-to-Peer Applications

- Peer-to-peer architecture is different from client-server architecture.
- In P2P, each node (called peers) acts as a client and server at the same time.
- The peers are not owned by a service-provider.
- The peers not supposed to be always listening on the Internet.
- The peers are dynamic, i.e., some peers will join some peers will leave from time to time.

1. P2P File Distribution

- In P2P file distribution, each peer can redistribute any portion of the file it has

received to any other peers, thereby assisting the server in the distribution process.

- The most popular P2P file distribution protocol is BitTorrent

Scalability of P2P Architectures

- Consider the following scenarios:

Suppose a server has a large file and 'N' computers want to download the file (Figure 2.24).

1) In client-server architecture, each of the N computers will
   → connect to the server &
   → download a copy of the file to local-host.
2) In P2P architecture, a peer need not necessarily download a copy from the server. Rather, the peer may download from other peers

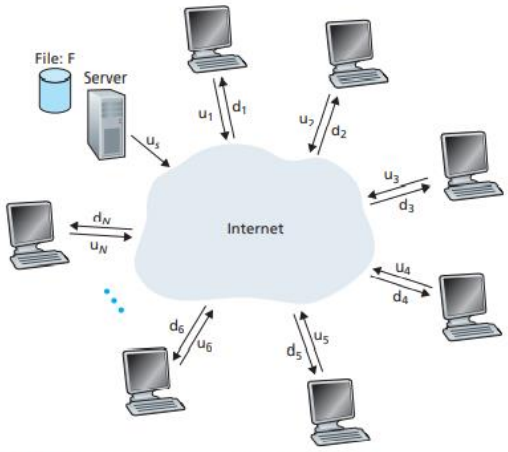


Figure 2.24 ♦ An illustrative file distribution problem

Let us define the following:
Length of the file = F
Upload rate for the server = u_s
Upload rate for ith computer = u_i
Download-rate for ith computer = d_i
Distribution-time for the client-server architecture = D_{cs}
Server needs transmit = N_F bits.
Lowest download-rate of peer = d_{min}
Distribution-time for the P2P architecture = D_{P2P}

Case 1: Client-Server Architecture

- We have 2 observations:

1) The server must transmit one copy of the file to each of the N peers.
   - Since the server's upload rate is u_s, the distribution-time is at least N_F/u_s.
2) The peer with the lowest download-rate cannot obtain all F bits of the file in less than F/d_{min}.
   - Thus, the minimum distribution-time is at least F/ d_{min}.

Putting these two observations together, we obtain

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}.$$

For N large file, the client-server distribution time is given by NF/us. Thus, the distribution time increases linearly with the number of peers N.

$$D_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}$$

Case 2: P2P Architecture

- ❖ Each peer can assist the server in distributing the file. In particular, when a peer receives some file data, it can use its own upload capacity to redistribute the data to other peers.
- We have 2 observations:

1) At the beginning of the distribution, only the server has the file.
   - ➢ So, the minimum distribution-time is at least F/u_s.
2) The peer with the lowest download-rate cannot obtain all F bits of the file in less than F/d_{min}.
   - ➢ Thus, the minimum distribution-time is at least F/ d_{min}.

3) The total upload capacity of the system as a whole is $u_{total} = u_s + u_1 + u_2 \ldots + u_N$.
   - ➢ The system must deliver F bits to each of the N peers.
   - ➢ Thus, the minimum distribution-time is at least $NF/(u_s + u_1 + u_2 \ldots + u_N)$.
   - ➢ Putting above 3 observations together, we have

$$D_{P2P} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\}$$

- ➢ Above Equations provides a lower bound for the minimum distribution time for the P2P architecture.
- ➢ It turns out that if we imagine that each peer can redistribute a bit as soon as it receives the bit, then there is a redistribution scheme that actually achieves this lower bound [Kumar 2006].
- ➢ In reality, where chunks of the file are redistributed rather than individual bits, Above Equation serves as a good approximation of the actual minimum distribution time.
- ➢ Thus, let's take the lower bound provided by Above Equation as the actual minimum distribution time, that is,

$$D_{P2P} = \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\}$$

- ➢ **Figure 2.23:** compares the minimum distribution-time for the client-server and P2P architectures.

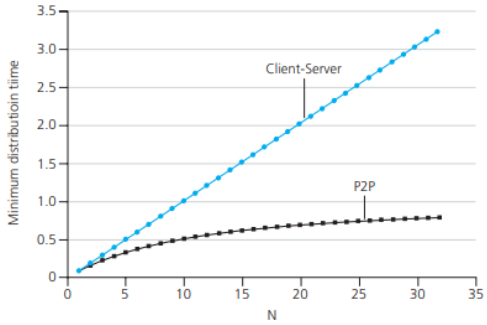


Figure 2.25 ♦ Distribution time for P2P and client-server architectures

- The above comparison shows that when N is large,
  - P2P architecture is consuming less distribution-time.
  - P2P architecture is self-scaling.

BitTorrent

- In BitTorrent, the collection of all peers participating in the distribution of a particular file is called a **torrent.**
- Peers in a torrent download equal-size chunks of the file from one another. Chunk size = 256 KBytes.
- When a peer first joins a torrent, it has no chunks. Over time it accumulates more and more chunks. While it downloads chunks it also uploads chunks to other peers.
- Once a peer has acquired the entire file, the peer may leave the torrent or remain in the torrent and continue to upload chunks to other peers.
- Also, any peer may leave the torrent at any time with only a subset of chunks, and later rejoin the torrent.
- Each torrent has an infrastructure node called **tracker.**
- Here is how it works (Figure):

1) When a peer joins a torrent, the peer
   → registers itself with the tracker and
   → periodically informs the tracker that it is in the torrent.
2) When a new peer joins the torrent, the tracker
   → randomly selects a subset of peers (for concreteness, say 50) from the set of participating peers and
   → sends the IP addresses of these 50 peers to the new peer.
3) Then, the new peer tries to establish concurrent TCP connections with all peers on this list.

- All peers on the list are called **neighboring-peers**.

4) Periodically, the new peer will ask each of the neighboring-peers(Over the TCP connections) for the set of chunks.
   - To choose the chunks to download, the peer uses a technique called rarest-first.
   - Main idea of rarest-first:

→ Determine the chunks that are the rarest among the neighbors and
→ Request then those rarest chunks first.

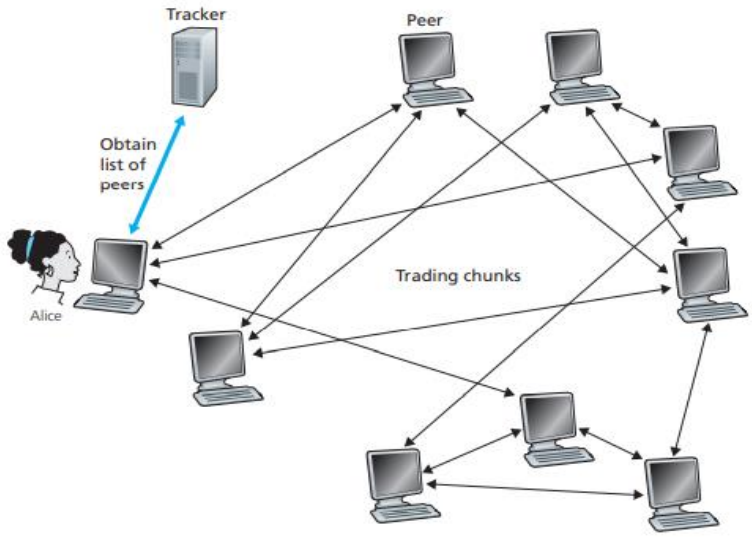


Figure 2.26 ♦ File distribution with BitTorrent

- To determine which requests peer responds to, BitTorrent uses a clever trading algorithm.
- The basic idea is that peer gives priority to the neighbors that are currently supplying data to it at the highest rate. Specifically, for each of its neighbors, peer continually measures the rate at which it receives bits and determines the four peers that are feeding bits at the highest rate.
- Peer then reciprocates by sending chunks to these same four peers.
- Every 10 seconds, peer recalculates the rates and possibly modifies the set of four peers.
- In BitTorrent lingo, these four peers are said to be **unchoked.**
- Importantly, every 30 seconds, peer also picks one additional neighbor at random and sends it chunks.
- In BitTorrent lingo, this randomly selected peer is said to **be optimistically unchoked.**

- The random neighbor selection also allows new peers to get chunks, so that they can have something to trade.
- The incentive mechanism for trading just described is often referred to as **tit-for-tat**.

2. Distributed Hash Table:

- Centralized version of this simple database will simply contain **(key, value)** pairs. We query the database with a key.
- If there are one or more key-value pairs in the database that match the query key, the database returns the corresponding values.
- Building such a database is straightforward with client-server architecture that stores all the **(key, value)** pairs in one central server.

- P2P version of this database will store the (key, value) pairs over millions of peers.
  - In the P2P system, each peer will only hold a small subset of the totality of the (key, value) pairs. We'll allow any peer to query the distributed database with a particular key. The distributed database will then locate the peers that have the corresponding **(key, value) pairs** and return the key-value pairs to the querying peer.
  - Any peer will also be allowed to **insert new key-value pairs into the database**. Such a distributed database is referred to as a distributed hash table (DHT).
  - One naïve approach to building a DHT is to randomly scatter the (key, value) pairs across all the peers and have each peer maintain a list of the IP addresses of all participating peers. In this design, the **querying peer sends its query** to all other peers, and the peers containing the (key, value) pairs that match the key can respond with their matching pairs.
  - Such an approach is **completely unscalable** as it would require each peer to know about all other peers and have each query sent to all peers.
  - An elegant approach to designing a DHT is to first assign an identifier to each peer, where each identifier is an integer in the range [0, 2n -1] for some fixed n.
  - This also require each key to be an integer in the same range.
  - To create integers out of such keys, we will use a hash function that maps each key (e.g., social security number) to an integer in the range **[0, 2 n -1].**

Problem of storing the (key, value) pairs in the DHT:

- The central issue here is defining a rule for assigning keys to peers. Given that each peer has an integer identifier and that each key is also an integer in the same range, a natural approach is to assign each (key, value) pair to the peer whose identifier is the closest to the key.
- To implement such a scheme, **let's define the closest peer as the closest successor of the key.**
- Now suppose a peer, Alice, wants to insert a (key, value) pair into the DHT. Conceptually, this is straightforward: She first determines the peer whose identifier is closest to the key; she then sends a message to that peer, instructing it to store the (key, value) pair.
- If Alice were to keep track of all the peers in the system (peer IDs and corresponding IP addresses), she could locally determine the closest peer. But such an approach requires each peer to keep track of all other peers in the DHT—which is completely impractical for a large scale system with millions of peers.

3. **Circular DHT**

- To address this problem of scale, let's now consider organizing the peers into a circle. In this circular arrangement, each peer only keeps track of its immediate successor and immediate predecessor (modulo 2n).

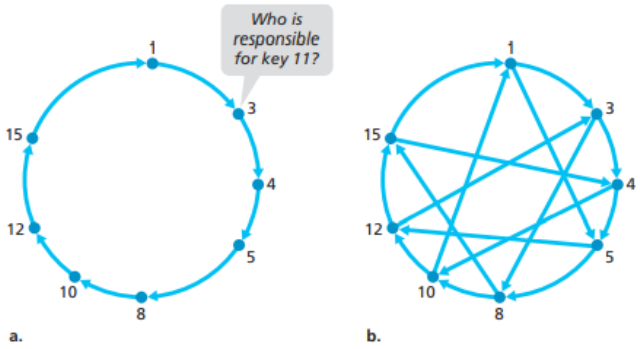


Figure 2.27 ♦ (a) A circular DHT. Peer 3 wants to determine who is responsible for key 11. (b) A circular DHT with shortcuts

- Each peer is only aware of its immediate successor and predecessor; for example, peer 5 knows the IP address and identifier for peers 8 and 4 but does not necessarily know anything about any other peers that may be in the DHT.
- Now suppose that peer 3 wants to determine which peer in the DHT is responsible for key 11.
- Using the circular overlay, the origin peer (peer 3) creates a message saying "Who is responsible for key 11?" and sends this message clockwise around the circle.
- Whenever a peer receives such a message, because it knows the identifier of its successor and predecessor, it can determine whether it is responsible for (that is, closest to) the key in question.
- If a peer is not responsible for the key, it simply sends the message to its successor. So, for example, when peer 4 receives the message asking about key 11, it determines that it is not responsible for the key (because its successor is closer to the key), so it just passes the message along to peer 5.
- This process continues until the message arrives at peer 12, who determines that it is the closest peer to key11.
- At this point, peer 12 can send a message back to the querying peer, peer 3, indicating that it is responsible for key 11.
- Although each peer is only aware of two neighboring peers, to find the node responsible for a key (in the worst case), all N nodes in the DHT will have to forward a message around the circle;
- N/2 messages are sent on average.
- Shortcuts are used to expedite the routing of query messages. Specifically, when a peer receives a message that is querying for a key, it forwards the message to the neighbor (successor neighbor or one of the shortcut neighbors) which is the closet to the key.
- When peer 4 receives the message asking about key 11, it determines that the closet peer to the key (among its neighbors) is its shortcut neighbor 10 and then forwards the message directly to peer 10.
- Clearly, shortcuts can significantly reduce the number of messages used to process a Query

4. Peer Churn

- In P2P systems, **a peer can come or go without warning.**
- Thus, when designing a DHT, we also must be concerned about maintaining the DHT

overlay in the presence of such peer churn.

- To handle peer churn, we will now require each peer to track its first and second successors;
- **for example,** peer 4 now tracks both peer 5 and peer 8. We also require each peer to periodically verify that its two successors are alive
- Let's now consider how the DHT is maintained when a peer abruptly leaves.
- For example, suppose peer 5 in above figure abruptly leaves.
- In this case, the two peers preceding the departed peer (4 and 3) learn that 5 has departed, since it no longer responds to ping messages.
- Peers 4 and 3 thus need to update their successor state information.
- Let's consider how peer 4 updates its state:

1. Peer 4 replaces its first successor (peer 5) with its second successor (peer 8).

2. Peer 4 then asks its new first successor (peer 8) for the identifier and IP address of its immediate successor (peer 10). Peer 4 then makes peer 10 its second successor.

3. Let's say a peer with identifier 13 wants to join the DHT, and at the time of joining, it only knows about peer 1's existence in the DHT.

4. Peer 13 would first send peer 1 a message, saying "what will be 13's predecessor and successor?" This message gets forwarded through the DHT until it reaches peer 12, who realizes that it will be 13's predecessor and that its current successor, peer 15, will become 13's successor.

5. Next, peer 12 sends this predecessor and successor information to peer 13. Peer 13 can now join the DHT by making peer 15 its successor and by notifying peer 12 that it should change its immediate successor to 13.

7. Socket Programming: Creating Network Applications

- Two types of network-applications:
  1) First type is an implementation whose operation is specified in a protocol standard (RFC)
     - Such an application is referred to as "open".
     - The client & server programs must conform to the rules dictated by the RFC.
  2) Second type is a proprietary network-application.
     - and close socket → clientSocket.close()

     ished in a RFC.
     - A single developer creates both the client and server programs.
     - The developer has complete control over the code.
- During development phase, the developer must decide whether the application uses TCP or UDP.

1. Socket Programming with UDP

- Consider client-server application in which following events occurs:

1) The client
   → reads a line of characters (data) from the keyboard and
   → sends the data to the server.
2) The server receives the data and converts the characters to uppercase.
3) The server sends the modified data to the client.
4) The client receives the modified data and displays the line on its screen.

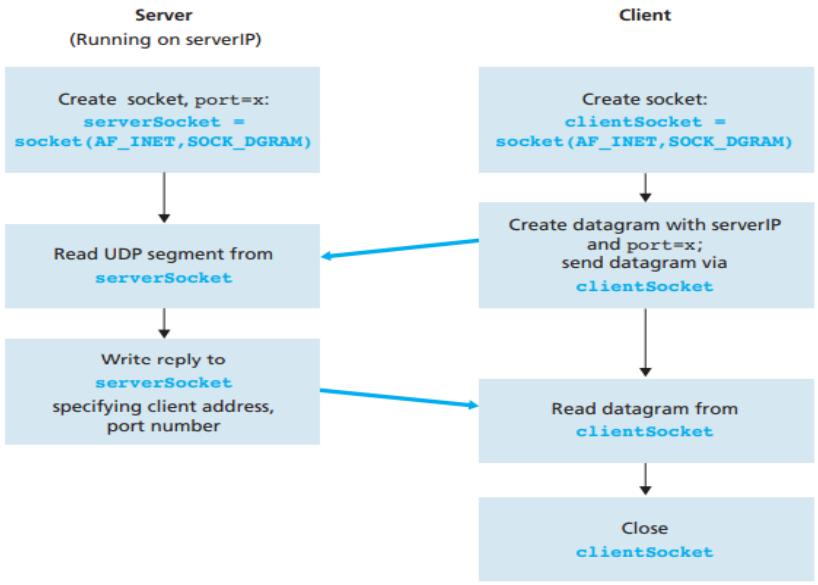


Figure 2.28 ♦ The client-server application using UDP

- Figure 2.28 highlights the main socket-related activity of the client and server that communicate over the UDP transport service.
- The client program is called **UDPClient.py,** and the server program is called **UDPServer.py.**

- **The client-side of the application is as follows:**

Python UDPClient

include Python's socket library

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET,
                      socket.SOCK_DGRAM)
```

create UDP socket for server

get user keyboard input

```
message = raw_input('Input lowercase sentence:')
```

Attach server name, port to message; send into socket

```
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
```

read reply characters from socket into string

print out received string and close socket

```
print modifiedMessage
clientSocket.close()
```

Here, AF_INET indicates address family
SOCK_DGRAM indicates UDP as socket type contains server's IP address & port#

- **The server-side of the application is as follows:**

Python UDPServer

```
from socket import *
serverPort = 12000
```

create UDP socket

```
serverSocket = socket(AF_INET, SOCK_DGRAM)
```

bind socket to local port number 12000

```
serverSocket.bind((' ', serverPort))
print "The server is ready to receive"
```

loop forever

```
while 1:
```

Read from UDP socket into message, getting client's address (client IP and port)

```
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
```

send upper case string back to this client

```
    serverSocket.sendto(modifiedMessage, clientAddress)
```

2. Socket Programming with TCP:

- Unlike UDP, TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection.
- One end of the TCP connection is attached to the client socket and the other end is attached to a server socket.
- When creating the TCP connection, we associate with it the client socket address (IP address and port number) and the server socket address (IP address and port number).
- With the TCP connection established, when one side wants to send data to the other side, it just drops the data into the TCP connection via its socket. This is different from UDP, for which the server must attach a destination address to the packet before dropping it into the socket.
- During the three-way handshake, the client process knocks on the welcoming door of the server process. When the server "hears" the knocking, it creates a new door— more precisely, a new socket that is dedicated to that particular client

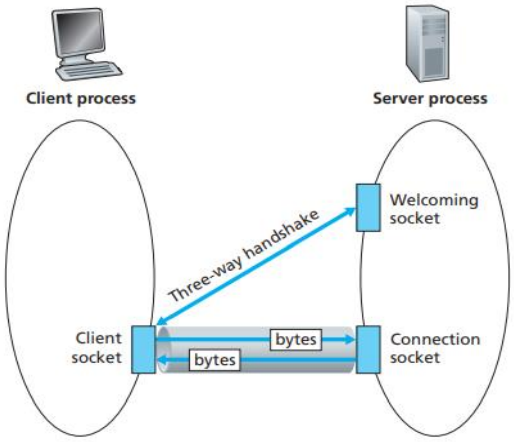


Figure 2.29 ♦ The TCPServer process has two sockets

- In Figure 2.29, the client process can send arbitrary bytes into its socket, and TCP guarantees that the server process will receive (through the connection socket) each byte in the order sent.
- TCP thus provides a reliable service between the client and server processes. Furthermore, just as people can go in and out the same door, the client process not only sends bytes into but also receives bytes from its socket; similarly, the server process not only receives bytes from but also sends bytes into its connection socket.
- We use the same simple client-server application to demonstrate socket programming with TCP: The client sends one line of data to the server, the server capitalizes the line and sends it back to the client.
- Figure 2.30 highlights the main socket-related

activity of the client and server that communicate over the TCP transport service.

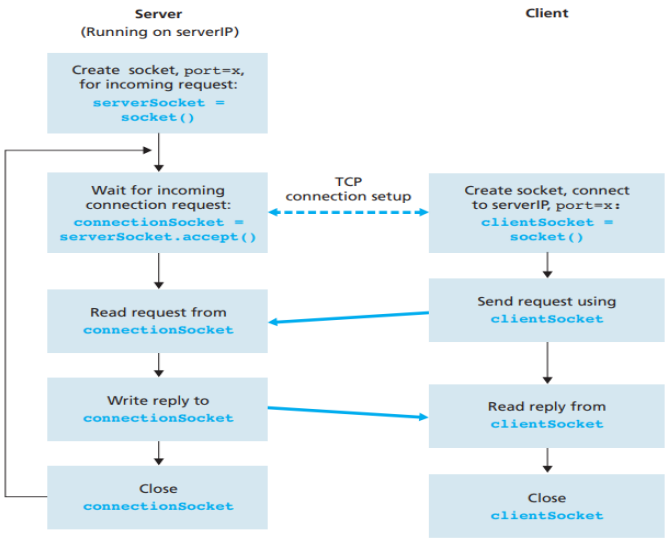


Figure 2.30 ♦ The client-server application using TCP

Example app: TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Example app: TCP server

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

Module-1 Questions

1) Explain client-server & P2P architecture. (8*)
2) With block diagram, explain how application processes communicate through a socket. (8*)
3) Explain 4 transport services available to applications. (4)
4) Briefly explain 2 transport layer protocols. (4)
5) With block diagram, explain the working of Web & HTTP. (8*)
6) Explain HTTP non-persistent & persistent connections. (8*)
7) With general format, explain HTTP request- & response-messages. (8*)
8) With a diagram, explain how cookies are used in user-server interaction. (6*)
9) With a diagram, explain the working of web caching. (6*)
10) With a diagram, explain the working of FTP. (6*)
11) With a diagram, explain the working of e-mail system. (6*)
12) Briefly explain 3 mail access protocols. (6*)
13) Briefly explain the working of DNS. (8*)
14) With general format, explain DNS messages. (6*)
15) Explain P2P File Distribution. (6)
16) With a diagram, explain the working of BitTorrent. (6*)
17) With a diagram, explain the working of Distributed Hash Table. (6)
18) Draw flow diagram for the client-server application using TCP. Also, write code for the client- & server-sides of the application. (8*)
19) Draw flow diagram for the client-server application using UDP. Also, write code for the client- & server-sides of the application. (8)

Module-2 Transport Layer

Transport layer resides in between Application layer and Network layer. It has the critical role of providing communication services directly to the application processes running on different hosts.

3.1 Introduction and Transport-Layer Services:

- A transport-layer protocol provides for **logical communication** between application processes running on different hosts.
- **Logical communication** - from an application's perspective, it is assumed as, the hosts running the processes were directly connected; in reality, the hosts may be in remote location, connected via numerous routers and a wide range of link types.
- Application processes use the logical communication provided by the transport layer to send messages to each other, free from the worry of the details of the physical infrastructure used to carry these messages.
- **Figure 3.1,** transport-layer provides logical rather than physical communication between application processes.
- As shown in Figure 3.1, transport-layer protocols are implemented in the end systems but not in network routers.
- On the **sending side**, the transport layer converts the application-layer messages it receives from a sending application process into transport-layer packets, known as **transport-layer segments.**
- This is done by (possibly) breaking the application messages into smaller chunks and adding a transport-layer header to each chunk to create the transport-layer segment.
- The transport layer then passes the segment to the network layer at the sending end system, where the segment is encapsulated within a network-layer packet (a datagram) and sent to the destination.
- On the **receiving side**, the network layer extracts the transport-layer segment from the datagram and passes the segment up to the transport layer.
- The transport layer then processes the received segment, making the data in the segment available to the receiving application.
- Internet has two protocols—**TCP and UDP**. Each of these protocols provides a different set of transport-layer services to the invoking application.

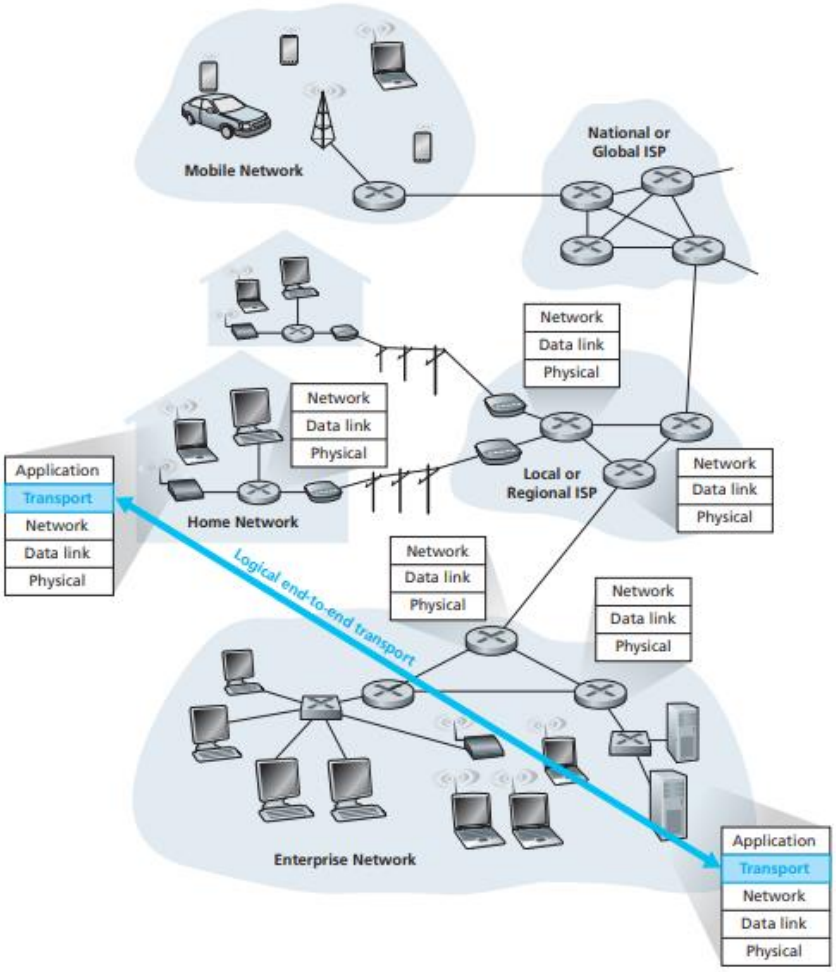


Figure 3.1 ♦ The transport layer provides logical rather than physical communication between application processes

3.1.1 Relationship Between Transport and Network Layers:

- Transport-layer protocol provides logical communication between **processes** running on different hosts, a network-layer protocol provides logical communication between **hosts**.
- Transport-layer protocols live in the end systems.
- Within an end system, a transport protocol moves messages from application processes to the network layer and vice versa.
- Intermediate routers will not recognize, any information that the transport layer may have added to the application messages.
- Transport layer provides **reliable service** where as network layer provides **unreliable service**.

3.1.2 Overview of the Transport Layer in the Internet:

Two protocols at transport layer are:

- **UDP** (User Datagram Protocol), provides an unreliable, connectionless service to the invoking application.
- **TCP** (Transmission Control Protocol), provides a reliable, connection-oriented service to the invoking application.
- Transport layer packet is referred as a segment. Transport-layer packet for TCP is referred as a segment and for UDP as a datagram.

1. Responsibility of IP at Network layer:

- The Internet's network-layer protocol has a name—IP, for Internet Protocol. IP provides logical communication between hosts.
- The IP service model is a **best-effort delivery service**. IP makes "best effort" to deliver segments between communicating hosts.
- IP does not guarantee segment delivery, orderly delivery of segments and the integrity of the data in the segments. Hence, IP is an **unreliable service**.
- Every host has at least one network-layer address, called **IP address.**

2. Responsibility of UDP and TCP at Transport layer:

- UDP and TCP extends IP's delivery service between two end systems to a delivery service between two processes running on the end systems.
- Extending host-to-host delivery to process-to-process delivery is called transport-layer **multiplexing** and **demultiplexing**.
- UDP and TCP provides integrity checking by including error detection fields in their segments' headers.

3. Services provided by UDP & TCP:

UDP is an unreliable service—it does not guarantee that data sent by one process will arrive intact to the destination process. UDP traffic is unregulated.

TCP Service:

- Provides reliable data transfer. Using flow control, sequence numbers, acknowledgments, and timers.
- TCP ensures that data is delivered from sending process to receiving process, correctly and in order.
- TCP converts IP's unreliable service between end systems

into a reliable data transport service between processes.

- TCP provides **congestion control**.
  - TCP congestion control prevents any one TCP connection from swamping the links and routers between communicating hosts with an excessive amount of traffic.
  - TCP strives to give each connection traversing a congested link an equal share of the link bandwidth.
  - Regulates the rate at which the sending sides of TCP connections can send traffic into the network.

3.2 Multiplexing and Demultiplexing

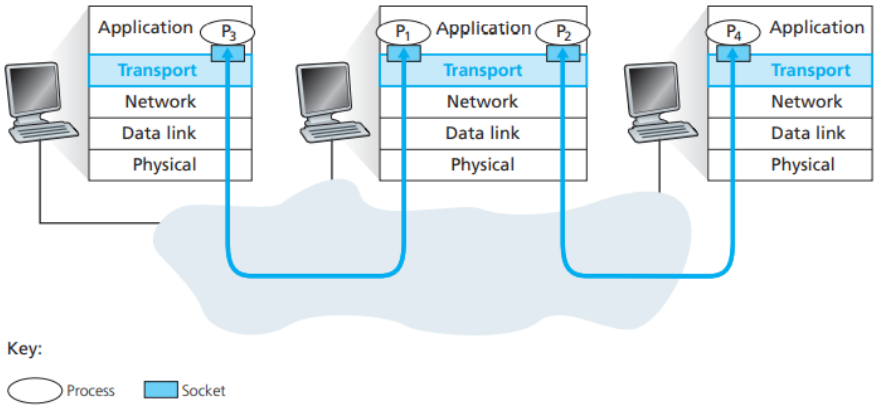


Figure 3.2 ♦ Transport-layer multiplexing and demultiplexing

- When the transport layer in receiving host receives data from the network layer below, it needs to direct the received data to one of the four processes P1, P2, P3, P4.
- A process can have one or more sockets, through which data passes from the network to the process and through which data passes from the process to the network. Each socket has a unique identifier.
- Thus, as shown in Figure above, the transport layer in the receiving host does not deliver data directly to a process, but instead to an intermediary socket.
- Each transport-layer segment has a set of fields in the segment to redirect data to above layer. At the receiving end, the transport layer examines these fields to identify the receiving socket and then directs the segment to that socket.
- Delivering the data in a transport-layer segment to the correct socket is called **demultiplexing**.

- Gathering data chunks at the source host from different sockets, encapsulating each
  data chunk with header information to create segments, and passing the segments to
  the network layer is called **multiplexing**.
  - The transport layer in the middle host in Figure above must demultiplex segments
    arriving from the network layer below to either process P1 or P2 above;
  - Demultiplexing is done by directing the arriving segment's data to the corresponding
    process's socket.
  - The transport layer in the middle host must also gather outgoing data from these sockets, form transport-layer segments, and pass these segments down to the network
    layer

Transport-layer multiplexing requires:

(1) Unique identifiers for sockets

(2) Each segment must have special fields that indicate the socket to which the segment is to be delivered.

These special fields, are the source port number field and the destination port number field.

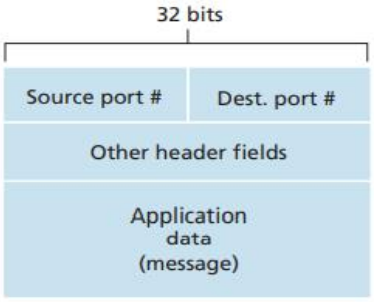


Figure 3.3 ♦ Source and destination port-number fields in a transport-layer segment

- Each port number is a 16-bit number, ranging from **0 to 65535**.
- The port numbers ranging from 0 to 1023 are called **well-known port numbers**
- They are reserved for use by well-known application protocols such as **HTTP (80)** and **FTP (21).**
- Each socket in the host could be assigned a port number, and when a segment arrives at the host, the transport layer examines the destination port number in the segment and directs the segment to the corresponding socket.
- The segment's data then passes through the socket into the attached process.

Connectionless Multiplexing and Demultiplexing

- **clientSocket = socket (socket.AF_INET, socket.SOCK_DGRAM)**
- When a UDP socket is created, the transport layer automatically assigns a port number to the socket.
- The transport layer assigns a port number in the range 1024 to 65535 that is currently not being used by any other UDP port in the host.
- Associate a specific port number (say, 19157) to this UDP socket via the socket bind() method:
- **clientSocket.bind((‘’, 19157))**

UDP multiplexing/demultiplexing:

- Suppose a process in Host A, with UDP port 19157, wants to send a chunk of application data to a process with UDP port 46428 in Host B.
- The transport layer in Host A creates a transport-layer segment that includes the application data, the source port number (19157), the destination port number (46428).
- The transport layer then passes the resulting segment to the network layer.
- The network layer encapsulates the segment in an IP datagram and makes a best-effort attempt to deliver the segment to the receiving host.
- If the segment arrives at the receiving Host B, the transport layer at the receiving host examines the destination port number in the segment (46428) and delivers the segment to its socket identified by port 46428.
- As UDP segments arrive from the network, Host B directs (demultiplexes) each segment to the appropriate socket by examining the segment’s destination port number.
- UDP socket is fully identified by a two-tuple consisting of a destination IP address and a destination port number.
- If two UDP segments have different source IP addresses and/or source port numbers, but have the same **destination IP** address and **destination port** number, then the two segments will be directed to the same destination process via the same destination socket.

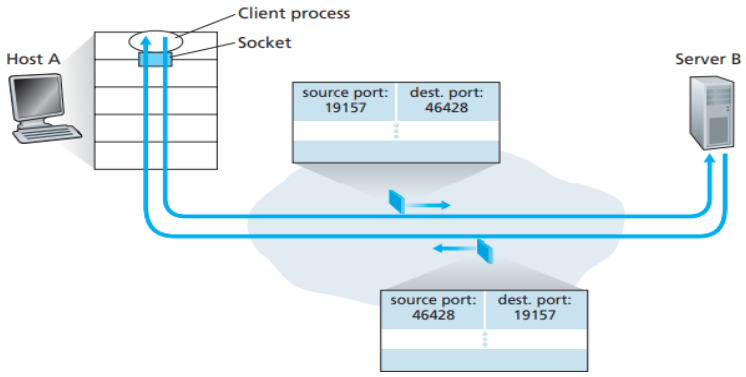


Figure 3.4 ♦ The inversion of source and destination port numbers

- **Host A-to-B:** In the segment, the source port number serves as part of a "return address"—when B wants to send a segment back to A, the destination port in the B-to-A segment will take its value from the source port value of the A-to-B segment.
- **UDPServer.py**, the server uses the recvfrom() method to extract the clientside (source) port number from the segment it receives from the client; it then sends a new segment to the client, with the extracted source port number serving as the destination port number in this new segment.

Connection-Oriented Multiplexing and Demultiplexing:

- Difference between a TCP socket and a UDP socket is that a TCP socket is identified by a **four-tuple:** (source IP address, source port number, destination IP address, destination port number).
- When a TCP segment arrives from the network to a host, the host uses all four values to direct (demultiplex) the segment to the appropriate socket.
- Two arriving TCP segments with different source IP addresses or source port numbers will be directed to two different sockets.
- The TCP server application has a "welcoming socket," that waits for connection establishment requests from TCP clients on port number 12000.
- The TCP client creates a socket and sends a connection establishment request segment with the lines:

clientSocket = socket(AF_INET, SOCK_STREAM)

clientSocket.connect((serverName,12000))

- A connection-establishment request is more a TCP segment with destination port number 12000 and a special connection-establishment bit set in the TCP header.
- The segment also includes a source port number that was chosen by the client.

- When the host operating system of the computer running the server process receives the incoming connection-request segment with destination port 12000, it locates the server process that is waiting to accept a connection on port number 12000.
- The server process then

  creates a new socket:

  connectionSocket, addr = serverSocket.accept( )

- The transport layer at the server notes the following four values in the connection-request segment:

  (1) the source port number in the segment,

  (2) the IP address of the source host,

  (3) the destination port number in the segment, and

  (4) its own IP address.

- The newly created connection socket is identified by these four values; all subsequently arriving segments whose source port, source IP address, destination port, and destination IP address match these four values will be demultiplexed to this socket.
- With the TCP connection, the client and server can now send data to each other.

- The server host may support many simultaneous TCP connection sockets, with each socket attached to a process, and with each socket identified by its own four tuple.

- When a TCP segment arrives at the host, all four fields (source IP address, source port, destination IP address, destination port) are used to direct (demultiplex) the segment to the appropriate socket.

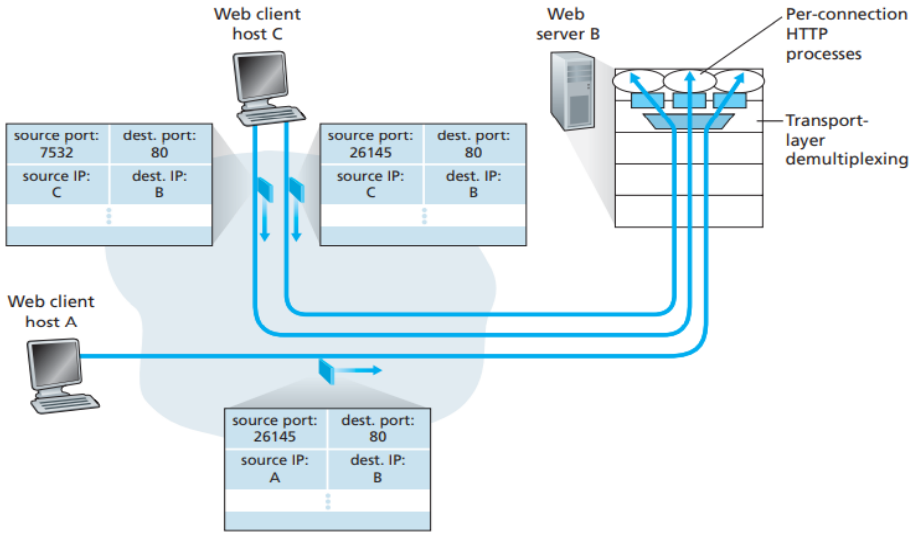


Figure 3.5 ♦ Two clients, using the same destination port number (80) to communicate with the same Web server application

- Eg. Consider above Figure, in which Host C initiates two HTTP sessions to server B, and Host A initiates one HTTP session to B. Hosts A and C and server Beach have their own unique IP address—A, C, and B, respectively.
- Host C assigns two different source port numbers (26145 and 7532) to its two HTTP connections.
- Because Host A is choosing source port numbers independently of C, it might also assign a source port of 26145 to its HTTP connection.
- Server B will still be able to correctly demultiplex the two connections having the same source port number, since the two connections have different source IP addresses.

Web Servers and TCP:

- Consider a host running a Web server, such as an Apache Web server, on port 80.
- When clients (for example, browsers) send segments to the server, all segments will have destination port 80.
- In particular, both the initial connection-establishment segments and the segments carrying HTTP request messages will have destination port 80.
- The server distinguishes the segments from the different clients using source IP addresses and source port numbers.
- As shown in Figure, each of these processes has its own connection socket through which HTTP requests arrive and HTTP responses are sent.
- There is not always a one-to-one correspondence between connection sockets and processes.
- Web servers often use only one process and create a new thread with a new connection socket for each new client connection.

3.3 Connectionless Transport: UDP

- The transport layer has to provide a multiplexing/demultiplexing service in order to pass data between the network layer and the correct application-level process.
- UDP, **does the multiplexing/demultiplexing function, error checking**.
- UDP takes messages from the application process, attaches source and destination port number fields for the multiplexing/demultiplexing service, adds two other small fields, and passes the resulting segment to the network layer.
- The network layer encapsulates the transport-layer segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host.
- If the segment arrives at the receiving host, UDP uses the destination port number to deliver the segment's data to the correct application process.
- UDP has no handshaking between sending and receiving transport-layer entities before sending a segment. Hence, UDP is said to be connectionless.

UDP is best suited for many applications for the following reasons:

1. Finer application-level control over what data is sent, and when.

- Under UDP, as soon as an application process passes data to UDP, UDP will package the data inside a UDP segment and immediately pass the segment to the network layer.
- TCP, on the other hand, has a congestion-control mechanism that throttles the transport-layer TCP sender when one or more links between the source and destination hosts become excessively congested.
- TCP will continue to resend a segment until the receipt of the segment has been acknowledged by the destination.
- Since real-time applications often require a minimum sending rate, do not want to overly delay segment transmission, and can tolerate some data loss, TCP's service model is not particularly well matched to these applications' needs.

2. No connection establishment.

- TCP uses a three-way handshake before it starts to transfer data. UDP just sends data away without any formal preliminaries.
- Thus UDP does not introduce any delay to establish a connection.
- HTTP uses TCP rather than UDP, since reliability is critical for Web pages with text.

3. No connection state

- **TCP** maintains connection state in the end systems. This connection state includes receive and send buffers, congestion-control parameters and sequence and acknowledgment number parameters.
- **UDP**, does not maintain connection state and does not track any of these parameters.

4. Small packet header overhead.

- The TCP segment has 20 bytes of header overhead in every segment, whereas UDP has only 8 bytes of overhead.
- E-mail, remote terminal access, the Web, and file transfer run over TCP—all these applications need the reliable data transfer service of TCP.

- UDP and TCP are used today with multimedia applications, such as Internet phone, real-time video conferencing, and streaming of stored audio and video.
- The lack of congestion control in UDP can result in high loss rates between a UDP sender and receiver.

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

Figure 3.6 ♦ Popular Internet applications and their underlying transport protocols

3.3.1 UDP Segment Structure:

- The application data occupies the data field of the UDP segment.
- **For example,** For a streaming audio application, audio samples fill the data field.
- The UDP header has only four fields, each consisting of two bytes.
- The port numbers allow the destination host to pass the application data to the correct process running on the destination end system (that is, to perform the demultiplexing function).
- The length field specifies the number of bytes in the UDP segment (header plus data).
- An explicit length value is needed since the size of the data field may differ from one UDP segment to the next.
- The checksum is used by the receiving host to check whether errors have been introduced into the segment.
- The length field specifies the length of the UDP segment, including the header, in bytes.

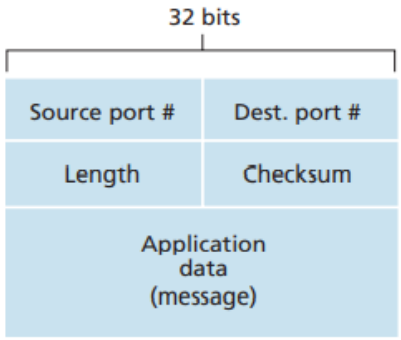


Figure 3.7 ♦ UDP segment structure

3.3.2 UDP Checksum

- The UDP checksum provides for error detection.
- That is, the checksum is used to determine whether bits within the UDP segment have been altered as it moved from source to destination.
- UDP at the sender side performs the 1s complement of the sum of all the 16-bit words in the segment, with any overflow encountered during the sum being wrapped around.
- This result is put in the checksum field of the UDP segment.

Example, suppose that we have the following three 16-bit words:

0110011001100000
0101010101010101
1000111100001100

The sum of first two of these 16-bit words is

0110011001100000
0101010101010101
1011101110110101

Adding the third word to the above sum gives

1011101110110101
1000111100001100
0100101011000010

- Last addition had overflow, which was wrapped around. The 1s complement is obtained by converting all the 0s to 1s and converting all the 1s to 0s.
- Thus the 1s complement of the sum 0100101011000010 is 1011010100111101, **which becomes the checksum**.
- At the receiver, all four 16-bit words are added, including the checksum. If no errors are introduced into the packet, then the sum at the receiver will be 1111111111111111.
- If one of the bits is a 0, then errors have been introduced into the packet.
- Take 1's compliment of generated sum, which is all 0's. Hence, no error has been detected.

- UDP provides error checking because there is no guarantee that all the links between source and destination provide error checking; that is, one of the links may use a link-layer protocol that does not provide error checking.

- Even if segments are correctly transferred across a link, it's possible that bit errors could be introduced when a segment is stored in a router's memory.
- Neither link-by-link reliability nor in-memory error detection is guaranteed, UDP must provide error detection at the transport layer, on an **end-end** basis, if the end-end data transfer service is to provide error detection.
- UDP has no error recovery method.

3.4 Principles of Reliable Data Transfer:

- With the reliable channel, no transferred data bits are corrupted or lost, and all are delivered in the order in which they were sent.
- This is the service model offered by TCP to the Internet applications that invoke it. It is the responsibility of a reliable data transfer protocol to implement this service abstraction.
- The layer below the reliable data transfer protocol may be unreliable. For example, TCP is a reliable data transfer protocol that is implemented on top of an unreliable (IP) end-to-end network layer.
- **Figure 3.8(b) illustrates the interfaces for data transfer protocol.**
- The sending side of the data transfer protocol will be invoked from above by a call to rdt_send().Here rdt stands for reliable data transfer protocol and _send indicates that the sending side of rdt is being called.
- It will pass the data to be delivered to the upper layer at the receiving side.
- On the receiving side, **rdt_rcv()** will be called when a packet arrives from the sending side of the channel.
- When the rdt protocol wants to deliver data to the upper layer, it will do so by calling deliver_data().
- Here **unidirectional data transfer** is considered, that is, data transfer from the sending to the receiving side.

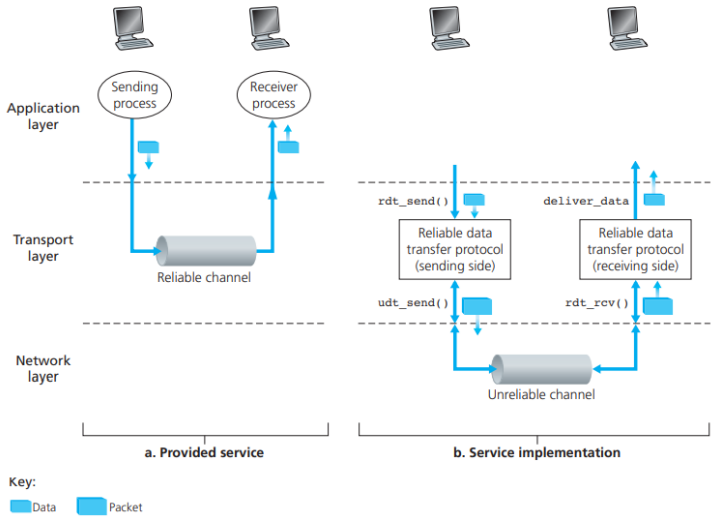
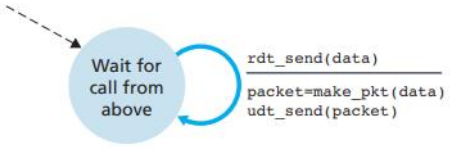


Figure 3.8 ♦ Reliable data transfer: Service model and service implementation

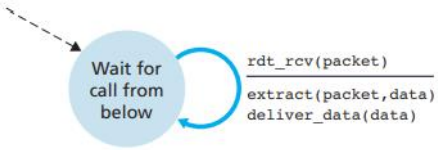
- In addition to exchanging packets containing the data to be transferred, the sending and receiving sides of rdt will also need to exchange control packets back and forth.
- Both the send and receive sides of rdt send packets to the other side by a call to **udt_send( ).**

3.4.1 Building a Reliable Data Transfer Protocol

1. Reliable Data Transfer over a Perfectly Reliable Channel: rdt1.0



a. rdt1.0: sending side



b. rdt1.0: receiving side

Figure 3.9 ♦ rdt1.0 – A protocol for a completely reliable channel

- Consider the simplest case, in which the underlying channel is completely reliable.
- The protocol is called **rdt1.0**.
- The finite-state machine (FSM) definitions for the rdt1.0 sender and receiver are shown in Figure above.
- The FSM in Figure (a) defines the operation of the sender, while the FSM in Figure (b) defines the operation of the receiver.
- There are **separate** FSMs for the sender and for the receiver. The sender and receiver FSMs in Figure each have just one state.
- The arrows in the FSM description indicate the transition of the protocol from one state to another.
- Since each FSM in Figure has just one state, a transition is necessarily from the one state back to itself;
- The event causing the transition is shown above the horizontal line labeling the **transition,** and the **actions** taken when the event occurs are shown below the horizontal line.
- When no action is taken on an event, or no event occurs and an action is taken, the symbol '_' below or above the horizontal is used . It explicitly denotes the lack of an action or event.
- The initial state of the FSM is indicated by the dashed arrow.
- The sending side of rdt simply accepts data from the upper layer via the **rdt_send(data)** event, creates a packet containing the data via the action make_pkt(data)) and sends the packet into the channel.
- The **rdt_send(data)** event would result from a procedure call by the upper-layer application.
- On the receiving side, rdt receives a packet from the underlying channel via the **rdt_rcv(packet)** event, removes the data from the packet (via the action extract (packet, data)) and passes the data up to the upper layer (via the action deliver_data(data)).
- The **rdt_rcv(packet)** event would result from a procedure call (for example, to rdt_rcv()) from the lower layer protocol.
- The packet flows from the sender to receiver; with a perfectly reliable channel there is no need for the receiver side to provide any feedback to the sender
- The receiver is able to receive data as fast as the sender happens to send data. Thus, there is no need for the sender to slow down its sending rate.

2. Reliable Data Transfer over a Channel with Bit Errors: rdt2.0

A more realistic model of the underlying channel is one in which bits in a packet may be corrupted.

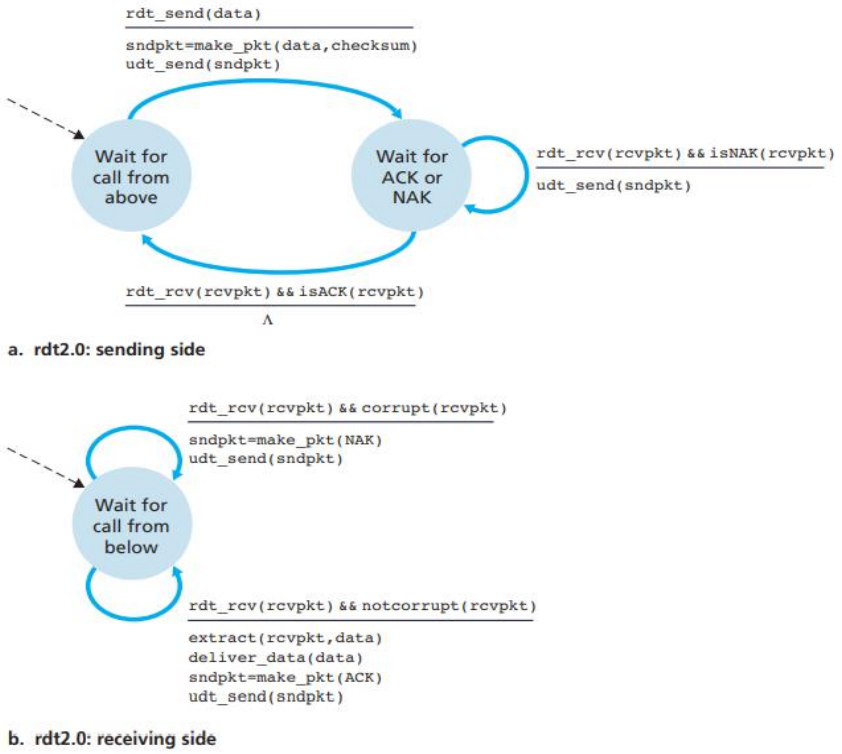


Figure 3.10 ♦ rdt2.0–A protocol for a channel with bit errors

- Such bit errors occur in the physical components of a packet is transmitted, propagates, or is buffered.
- Assume that all transmitted packets are received (although their bits may be corrupted) in the order in which they were sent.
- This message-dictation protocol uses both **positive acknowledgments** and **negative acknowledgments**. These control messages allow the receiver to let the sender know what has been received correctly, and what has been received in error and thus requires repeating.
- Reliable data transfer protocols based on retransmission are known as **ARQ (Automatic Repeat reQuest) protocols.**
- Three additional protocol capabilities are required in ARQ protocols to handle the presence of bit errors:

1. **Error detection.**
   - A mechanism is needed to allow the receiver to detect bit errors if occurred. UDP uses the Internet checksum field for this purpose.
   - Error detection techniques allow the receiver to detect and possibly correct packet bit errors.

- These techniques require that extra bits(checksum) be sent from the sender to the receiver; these bits will be gathered into the packet checksum field of the rdt2.0 data packet.

2. **Receiver feedback**.
   - Since the sender and receiver are typically executing on different end systems, it requires for the receiver to provide explicit feedback to the sender.
   - The positive (ACK) and negative (NAK) acknowledgment replies in the message are examples of such feedback. Rdt2.0 protocol sends ACK and NAK packets back from the receiver to the sender.
   - ACK packets is just one bit long; for example, a 0 value could indicate a NAK and a value of 1 could indicate an ACK.
3. **Retransmission.**
   - A packet that is received in error at the receiver will be retransmitted by the sender.
   - Figure shows the FSM representation of rdt2.0, a data transfer protocol employing error detection, positive acknowledgments, and negative acknowledgments.
   - **The sender side of rdt2.0 has two states:**
     i. The send-side protocol is waiting for data to be passed down from the upper layer. When the **rdt_send(data)** event occurs, the sender will create a **packet (sndpkt)** containing the data to be sent, along with a packet checksum and then send the packet via the **udt_send(sndpkt)** operation.
     ii. In the rightmost state, the sender protocol is waiting for an ACK or a NAK packet from the receiver. If an ACK packet is received (the notation rdt_rcv(rcvpkt) && isACK (rcvpkt) in Figure corresponds to this event), the sender knows that the most recently transmitted packet has been received correctly.
   - Thus, the protocol returns to the state of waiting for data from the upper layer. If a NAK is received, the protocol retransmits the last packet and waits for an ACK or NAK to be returned by the receiver in response to the retransmitted data packet.
   - When the sender is in the wait-for-ACK-or-NAK state, it cannot get more data from the upper layer; that is, the **rdt_send()** event cannot occur; that will happen only after the sender receives an ACK and leaves this state.
   - Thus, the sender will not send a new piece of data until it is sure that the receiver has correctly received the current packet.
   - Because of this behavior, protocols rdt2.0 is known as **stop-and-wait** protocol.
   - The **receiver-side** FSM for rdt2.0 still has a single state. On packet arrival, the receiver replies with either an ACK or a NAK, depending on whether or not the received packet is corrupted.
   - Notation **rdt_rcv(rcvpkt) && corrupt(rcvpkt)** corresponds to the event in which a packet is received and is found to be in error.

- ACK or NAK packet could be corrupted and solution could be adding checksum bits to ACK/NAK packets in order to detect such errors.

- The difficulty here is that if an ACK or NAK is corrupted, the sender has no way of knowing whether or not the receiver has correctly received the last piece of transmitted data.

- Consider three possibilities for handling **corrupted ACKs or NAKs:**
  1. For the **first possibility**, if Sender receives garbled ACK/NAK , he creates a new packet –asking the receiver to resend ACK/NAK. But creation of such new packet leads to issues.
  2. A **second alternative** is to add enough checksum bits to allow the sender not only to detect, but also to recover from, bit errors. This solves the immediate problem for a channel that can corrupt packets but not lose them.
  3. A **third approach** is for the sender simply to resend the current data packet when it receives a garbled ACK or NAK packet. This approach, introduces **duplicate packets** into the sender to-receiver channel. The fundamental difficulty with duplicate packets is that the receiver doesn't know whether the ACK or NAK it last sent was received correctly at the sender.

Solution :

- A simple solution to this new problem is to add a new field to the data packet and have the sender number its data packets by putting a sequence number into this field.
- The receiver then need only check this sequence number to determine whether or not the received packet is a retransmission.
- For this simple case of a stop-and wait protocol, **a 1-bit sequence number** will suffice, since it will allow the receiver to know whether the sender is resending the previously transmitted packet (the sequence number of the received packet has the same sequence number as the most recently received packet) or a new packet.
- Since it is assumed that channel does not lose packets, **ACK and NAK** packets do not themselves need to indicate the sequence number of the packet they are acknowledging.
- The sender knows that a received ACK or NAK packet (whether garbled or not) was generated in response to its most recently transmitted data packet.

RDT 2.1 sender(a) :

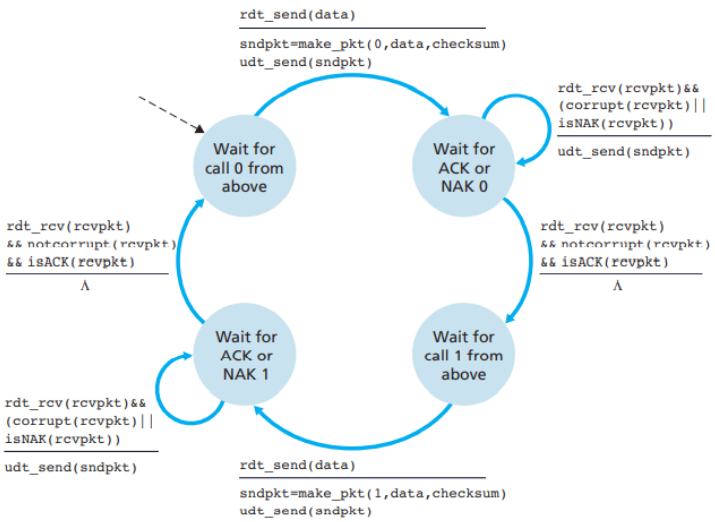


Figure 3.11 ♦ rdt2.1 sender

RDT 2.1 receiver(b) :

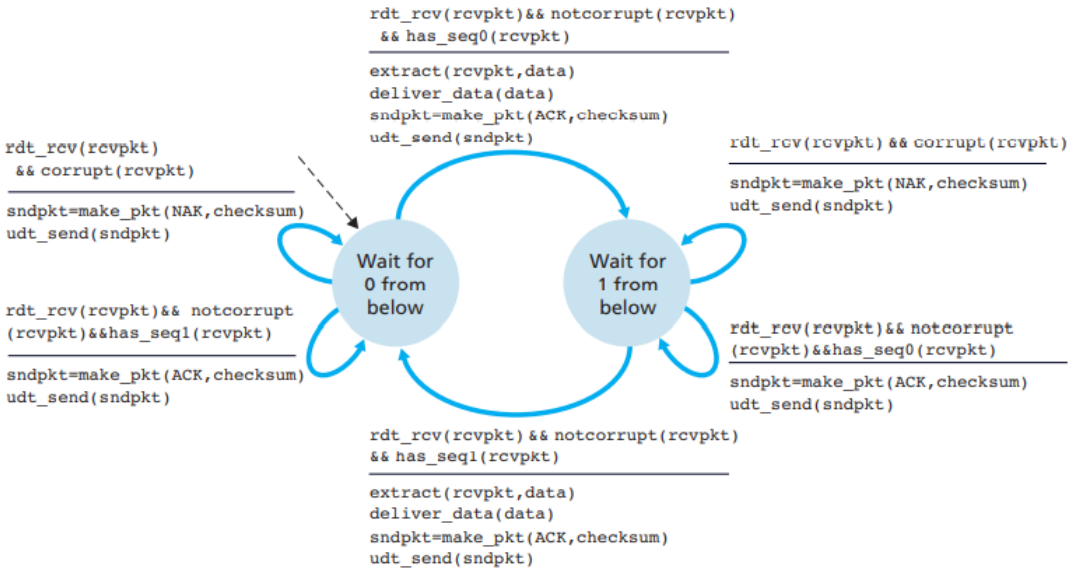


Figure 3.12 ♦ rdt2.1 receiver

- Figures (a) and (b) shows the FSM description for rdt2.1.
- The rdt2.1 sender and receiver FSMs each now have twice as many states as before.

- This is because the protocol state must now reflect whether the packet currently being sent (by the sender) or expected (at the receiver) should have a sequence number of 0 or 1.
- **Protocol rdt2.1** uses both positive and negative acknowledgments from the receiver to the sender.
- When an out-of-order packet is received, the receiver sends a positive acknowledgment for the packet it has received.
- When a corrupted packet is received, the receiver sends a **negative acknowledgment**.
- The same effect as a **NAK** could be accomplished if, instead of sending a NAK, we send an ACK for the last correctly received packet.
- A sender that receives **two ACKs** for the same packet (that is, receives duplicate ACKs) knows that the receiver did not correctly receive the packet following the packet that is being ACKed twice.

RDT 2.2 sender:

NAK-free reliable data transfer protocol for a channel with bit errors is rdt2.2.

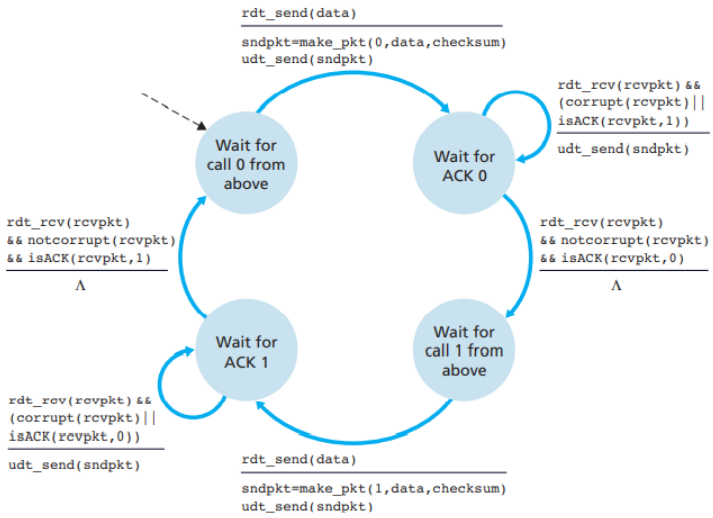


Figure 3.13 ♦ rdt2.2 sender

RDT 2.2 receiver:

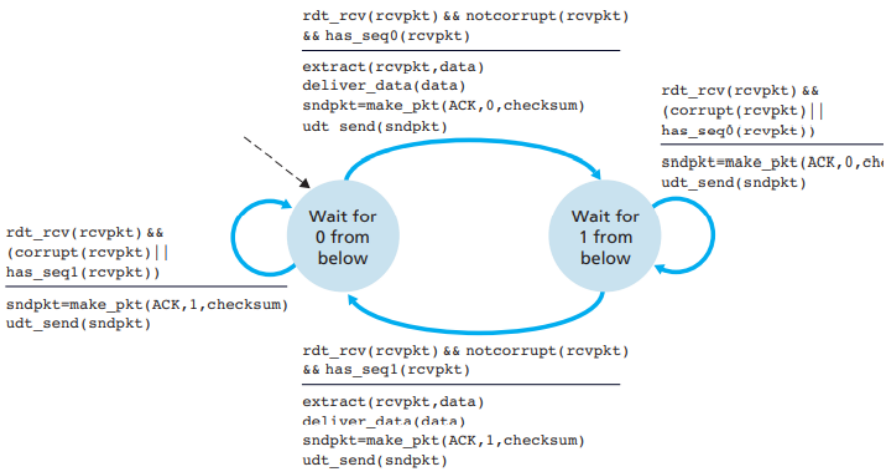


Figure 3.14 ♦ rdt2.2 receiver

- The sequence number is included in the packet i.e being acknowledged by an ACK message (this is done by including the ACK,0 or ACK1 argument in **make_pkt()** in the receiver FSM)
- The sender must now check the sequence number of the packet being acknowledged by a received ACK message (this is done by including the 0 or 1 argument in is **ACK()** in the sender FSM).

3. Reliable Data Transfer over a Lossy Channel with Bit Errors: rdt3.0

- The use of check summing, sequence numbers, ACK packets, and retransmissions—are solutions for corrupted data , out of order data , lost packet problems.
- Detecting and recovering from lost packets is the responsibility of sender.
- Suppose that the sender transmits a data packet and either that packet, or the receiver's ACK of that packet, gets lost. In either case, no reply is forthcoming at the sender from the receiver.
- Solution is setting a timer by the sender.
- The sender must clearly wait at least as long as a round-trip delay between the sender and receiver plus amount of time is needed to process a packet at the receiver.
- If an **ACK** is not received within this timer, the packet is retransmitted. If a packet experiences a large delay, the sender may retransmit the packet even though neither the data packet nor its ACK have been lost.
- This introduces the possibility of **duplicate data packets** in the sender-to-receiver channel.
- **Sequence numbers** can be used to handle the case of duplicate packets.

- The sender does not know whether a data packet was lost, an ACK was lost, or if the packet or ACK was simply overly delayed. Retransmission is the solution for all these problems.
- Implementing a time-based retransmission mechanism requires a **countdown timer** that can interrupt the sender after a given amount of time has expired.
- The sender will thus need to be able to

  (1) start the timer each time a packet (either a first-time packet or a retransmission) is sent,

  (2) respond to a timer interrupt (taking appropriate actions)

  (3) stop the timer

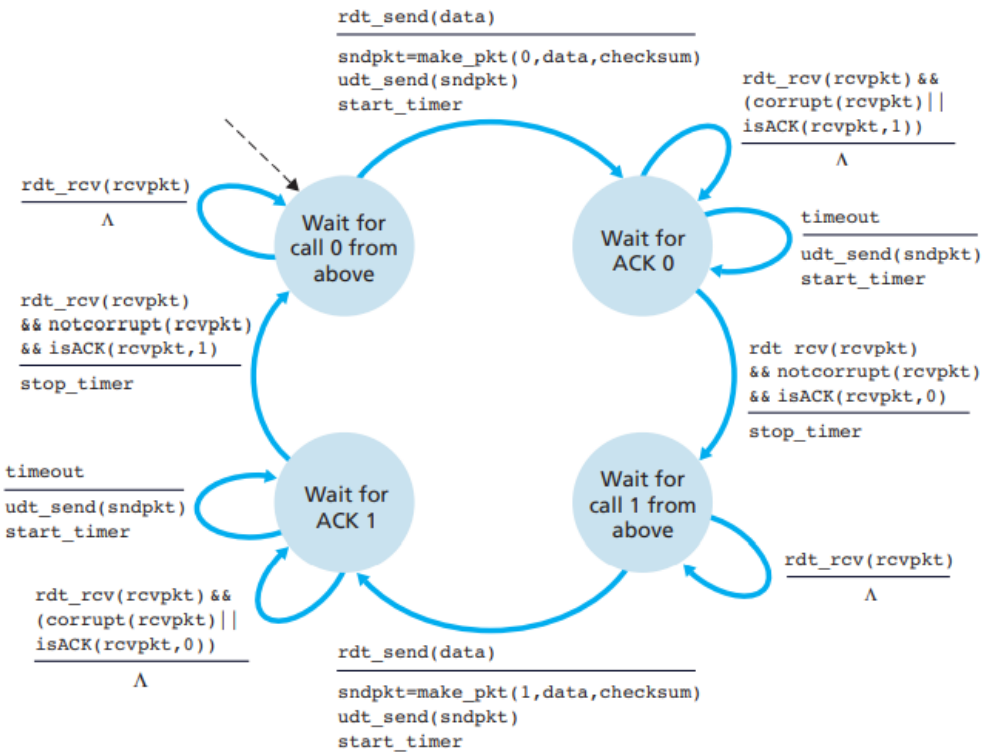


Figure 3.15 ♦ rdt3.0 sender

- Figure shows the sender FSM for rdt3.0, a protocol that reliably transfers data over a channel that can corrupt or lose packets.

- In Figure below, time moves forward from the top of the diagram toward the bottom of the diagram;
- Receive time for a packet is necessarily later than the send time for a packet as a result of transmission and propagation delays.

3.4.2 Pipelined Reliable Data Transfer Protocols:

- Consider an idealized case of two hosts, one located on the West Coast of the United States and the other located on the East Coast, as shown in Figure below.

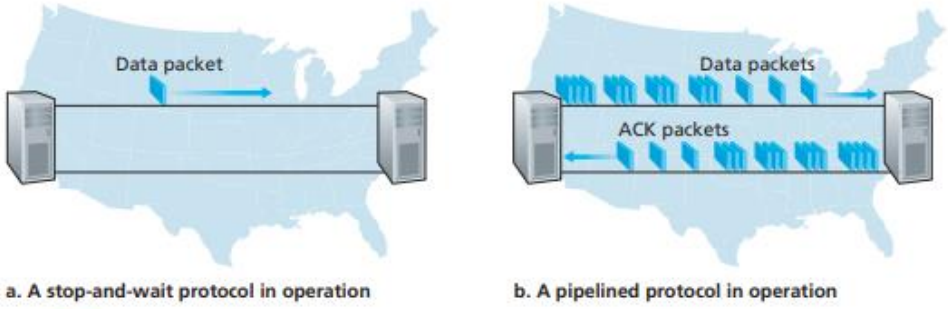


Figure 3.17 ♦ Stop-and-wait versus pipelined protocol

- The round-trip propagation delay between these two end systems, RTT, is approximately 30 milliseconds. **RTT=30ms**
- Suppose that they are connected by a channel with a transmission rate, R, of 1 Gbps (109 bits per second).
- R=1Gbps With a packet size, L, of 1,000 bytes (8,000 bits) per packet, including both header fields and data.
- L=8000 bitsps The time needed to actually transmit the packet into the 1 Gbps link is **dtrans =L/R = 8000 bits/packet / 109bits/sec = 8 microseconds**
- **Figure (a)** shows that with stop-and-wait protocol, if the sender begins sending the packet at , t = 0, then at t = L/R = 8 microseconds;
- The packet then makes its 15-msec journey from sender to receiver, RTT/2=15ms.
- The last bit of the packet is emerging at the receiver at, t = RTT/2 + L/R =15.008 msec.
- The **ACK** emerges back at the sender at t = RTT + L/R = 30.008 msec. (RTT= 15+15=30ms).
- At this point, the sender can now transmit the next message.
- Thus, in 30.008 msec, the sender was sending for only 0.008 msec.
- If we define the utilization of the sender (or the channel) as the fraction of time the sender is actually busy sending bits into the channel, the analysis in below Figure (a) shows that the **stop-and-wait protocol** has sender utilization, **Usender**, of

Usender = L/R / (RTT + L/R) = .008 / 30.008 = 0.00027.

- That is, the sender was busy only 2.7 hundredths of one percent of the time!
- The solution to this performance problem is: Rather than operate in a stop-and-wait manner, the sender is allowed to send multiple packets without waiting for acknowledgments, as illustrated in below **Figure (b).**
    - Figure (b) shows that if the sender is allowed to transmit three packets before having to wait for acknowledgments, the utilization of the sender is essentially tripled.

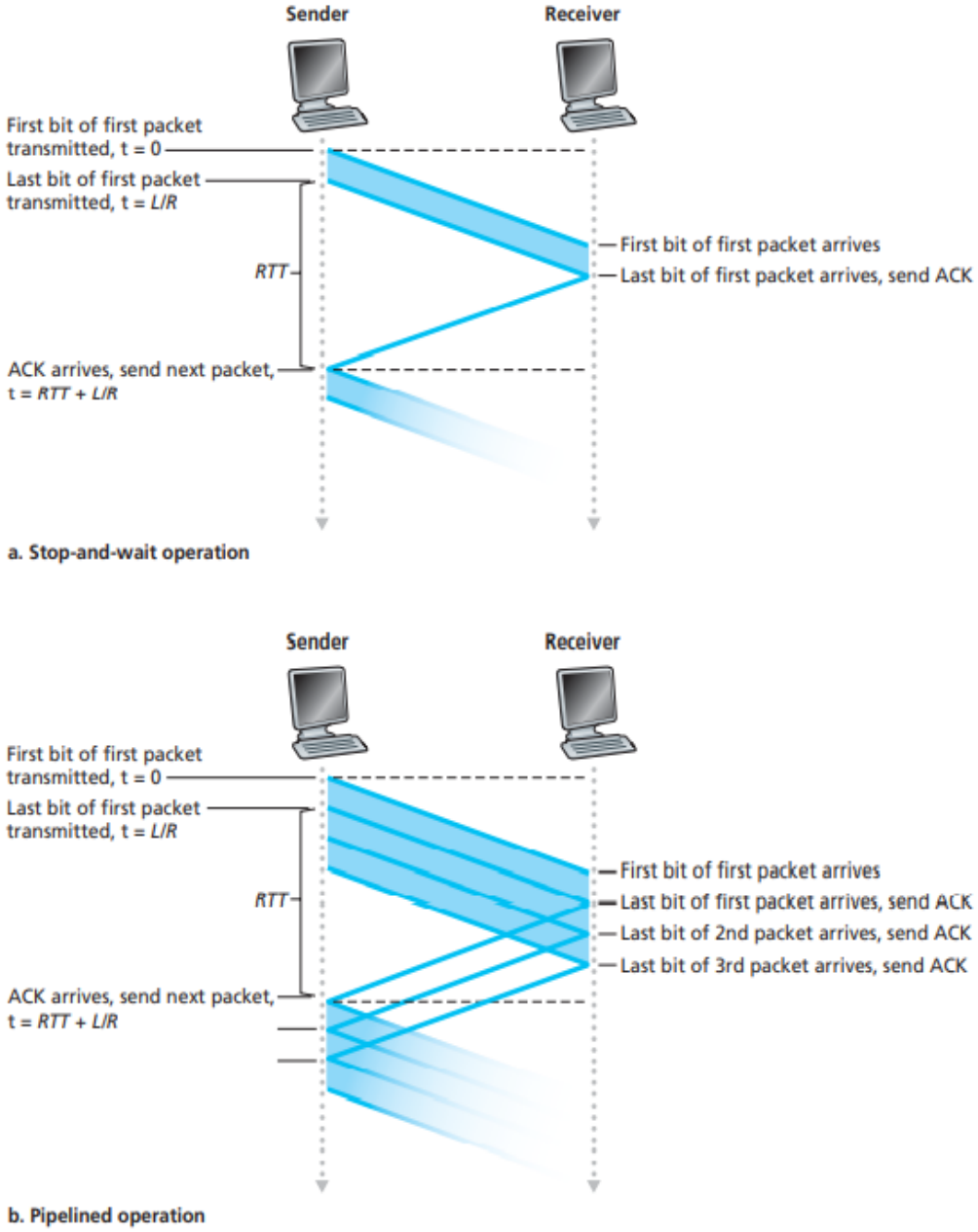


Figure 3.18 ♦ Stop-and-wait and pipelined sending

- Since the many in-transit sender-to-receiver packets can be visualized as filling a pipeline, this technique is known **as pipelining.**

Pipelining has the following consequences for reliable data transfer protocols:

The range of sequence numbers must be increased, since each in-transit packet (not counting retransmissions) must have a unique sequence number and there may be multiple, in-transit, unacknowledged packets.

1. The sender and receiver sides of the protocols may have to buffer more than one packet. The sender will have to buffer packets that have been transmitted but not yet acknowledged. Buffering of correctly received packets may also be needed at the receiver.
2. The range of sequence numbers needed and the buffering requirements will depend on the manner in which a data transfer protocol responds to lost, corrupted and overly delayed packets.

Two basic approaches toward pipelined error recovery can be: Go-Back-N and selective repeat.

3.4.3 Go-Back-N (GBN):

- In **a Go-Back-N (GBN)** protocol, the sender is allowed to transmit multiple packets without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, N, of unacknowledged packets in the pipeline.

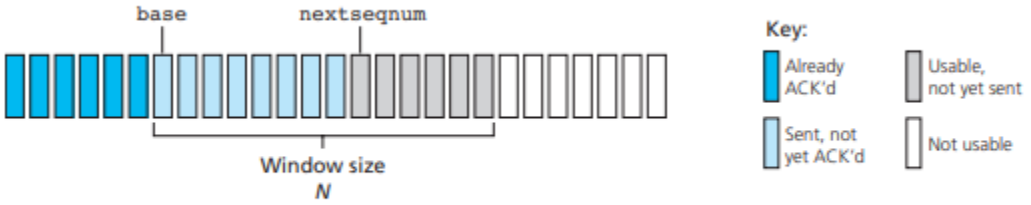


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

- Figure above shows the sender's view of the range of sequence numbers in a GBN protocol.
- Define **base** to be the sequence number of the oldest unacknowledged packet and **nextseqnum** to be the smallest unused sequence number (that is, the sequence number of the next packet to be sent).
- Sequence numbers in the interval [0, base-1] correspond to packets that have already been transmitted and acknowledged.

- The interval [base, nextseqnum-1] corresponds to packets that have been sent but not yet acknowledged.
- Sequence numbers in the interval [nextseqnum, base+N-1] can be used for packets that can be sent immediately, should data arrive from the upper layer.
- Finally, **sequence numbers** greater than or equal to base+N cannot be used until an unacknowledged packet currently in the pipeline has been acknowledged.

- The range of permissible sequence numbers for transmitted but not yet acknowledged packets can be viewed as a window of size N over the range of sequence numbers.
- As the protocol operates, this window slides forward over the sequence number space. For this reason, N is often referred to as the **window size** and the GBN protocol as a **sliding-window protocol.**
- A packet's sequence number is carried in a fixed-length field in the packet header. If k is the number of bits in the packet sequence number field, the range of sequence numbers is thus $[0,2^k - 1]$.
- The sequence number space can be thought of as a ring of size 2^k, where sequence number $2^k - 1$ is immediately followed by sequence number 0.
- **The GBN sender must respond to three types of events:**

1. **Invocation from above**.
    a. When **rdt_send()** is called from above, the sender first checks to see if the window is full, that is, whether there are N outstanding, unacknowledged packets.
    b. If the window is not full, a packet is created and sent, and variables are appropriately updated.
    c. If the window is full, the sender simply returns the data back to the upper layer, an implicit indication that the window is full. The upper layer would have to try again.
2. **Receipt of an ACK**.

    In our GBN protocol, an acknowledgment for a packet with sequence number n will be taken to be a cumulative acknowledgment, indicating that all packets with a sequence number up to and including n have been correctly received at the receiver.

3. **A timeout event**.

    A timer will be used to recover from lost data or acknowledgment packets. If a timeout occurs, the sender resends all packets that have been previously sent but that have not yet been acknowledged.

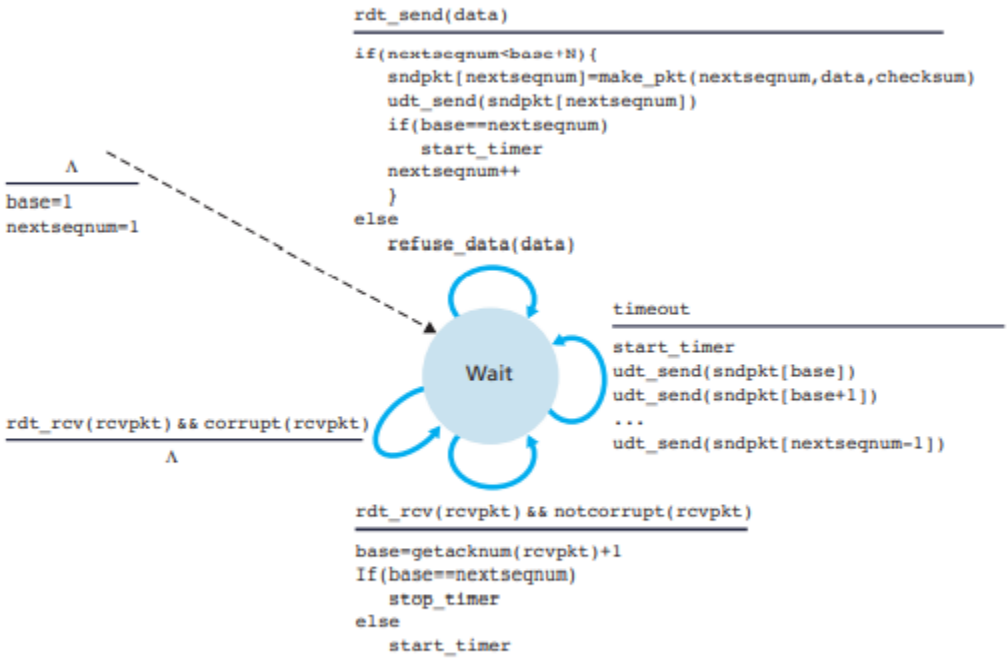


Figure 3.20 ♦ Extended FSM description of GBN sender

1. **Sender** in above Figure uses only a single timer, which can be thought of as a timer for the oldest transmitted but not yet acknowledged packet.
   - If an ACK is received but there are still additional transmitted but not yet acknowledged packets, the timer is restarted.
   - If there are no outstanding, unacknowledged packets, the timer is stopped.
2. The **receiver's action:** If a packet with sequence number n is received correctly and is in order, the receiver sends an ACK for packet n and delivers the data portion of the packet to the upper layer.
   a. In all other cases, the receiver discards the packet and resends an ACK for the most recently received in-order packet.
   b. Since packets are delivered one at a time to the upper layer, if packet k has been received and delivered, then all packets with a sequence number lower than k have also been delivered.
   c. Thus, cumulative acknowledgments is used for GBN.
   d. In **GBN protocol**, the receiver discards **out-of-order** packets because the receiver must deliver data in order to the upper layer.

- Suppose the packet n is expected, but packet **n + 1** arrives. Because data must be delivered in order, the receiver could buffer (save) packet **n + 1** and then deliver this packet to the upper layer after it had later received and delivered packet n.

- If packet n is lost, both it and packet n + 1 will eventually be retransmitted as a result of the GBN retransmission rule at the sender.
- Thus, the receiver can simply discard packet **n + 1.**
- The advantage of this approach is the receiver need not buffer **any** out-of-order packets.
- Thus, while the sender must maintain the upper and lower bounds of its window and the position of **nextseqnum** within this window.
- The only piece of information the receiver need maintain is the sequence number of the next in-order packet.
- This value is held in the variable expected **seqnum**, shown in the receiver FSM in Figure below.

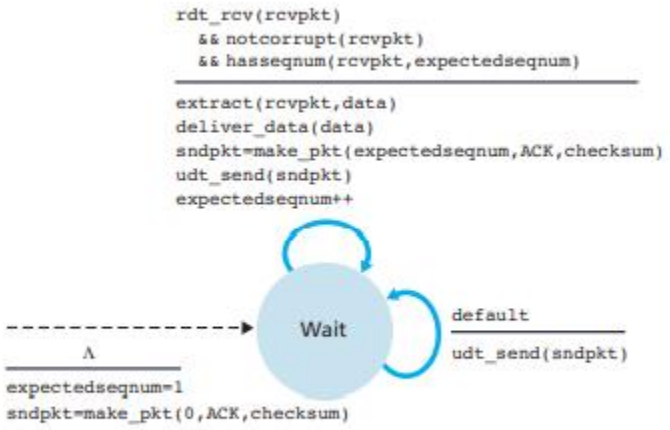


Figure 3.21 ♦ Extended FSM description of GBN receiver

The **disadvantage** of discarding a correctly received packet is that the subsequent retransmission of that packet might be lost or garbled and thus even more retransmissions would be required.

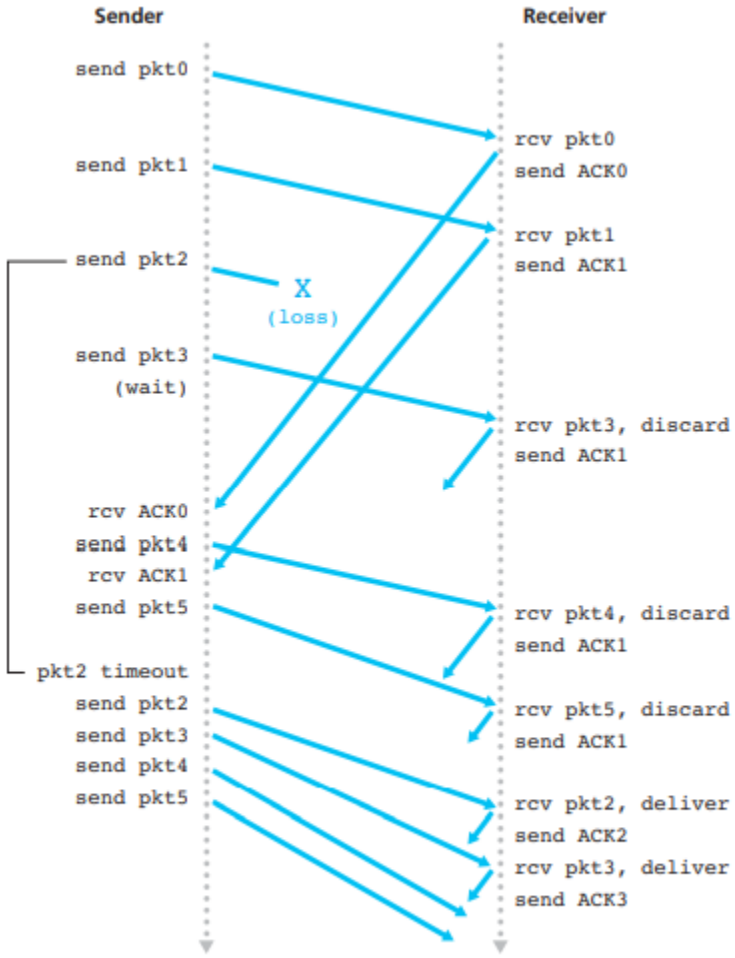


Figure 3.22 ♦ Go-Back-N in operation

- Figure above shows the operation of the GBN protocol for the case of a window size of **four packets**.
- Because of this window size limitation, the sender sends packets 0 through 3 but then must wait for one or more of these packets to be acknowledged before proceeding.
- As each successive **ACK** (for example, ACK0 and ACK1) is received, the window slides forward and the sender can transmit one new packet (pkt4 and pkt5, respectively).
- On the receiver side, packet 2 is lost and thus packets 3, 4, and 5 are found to be out of order and are discarded.
- In the sender, the events would be:

  (1) a call from the upper-layer entity to invoke **rdt_send(),**

  (2) a timer interrupt,

  (3) a call from the lower layer to invoke **rdt_rcv()** when a packet arrives.

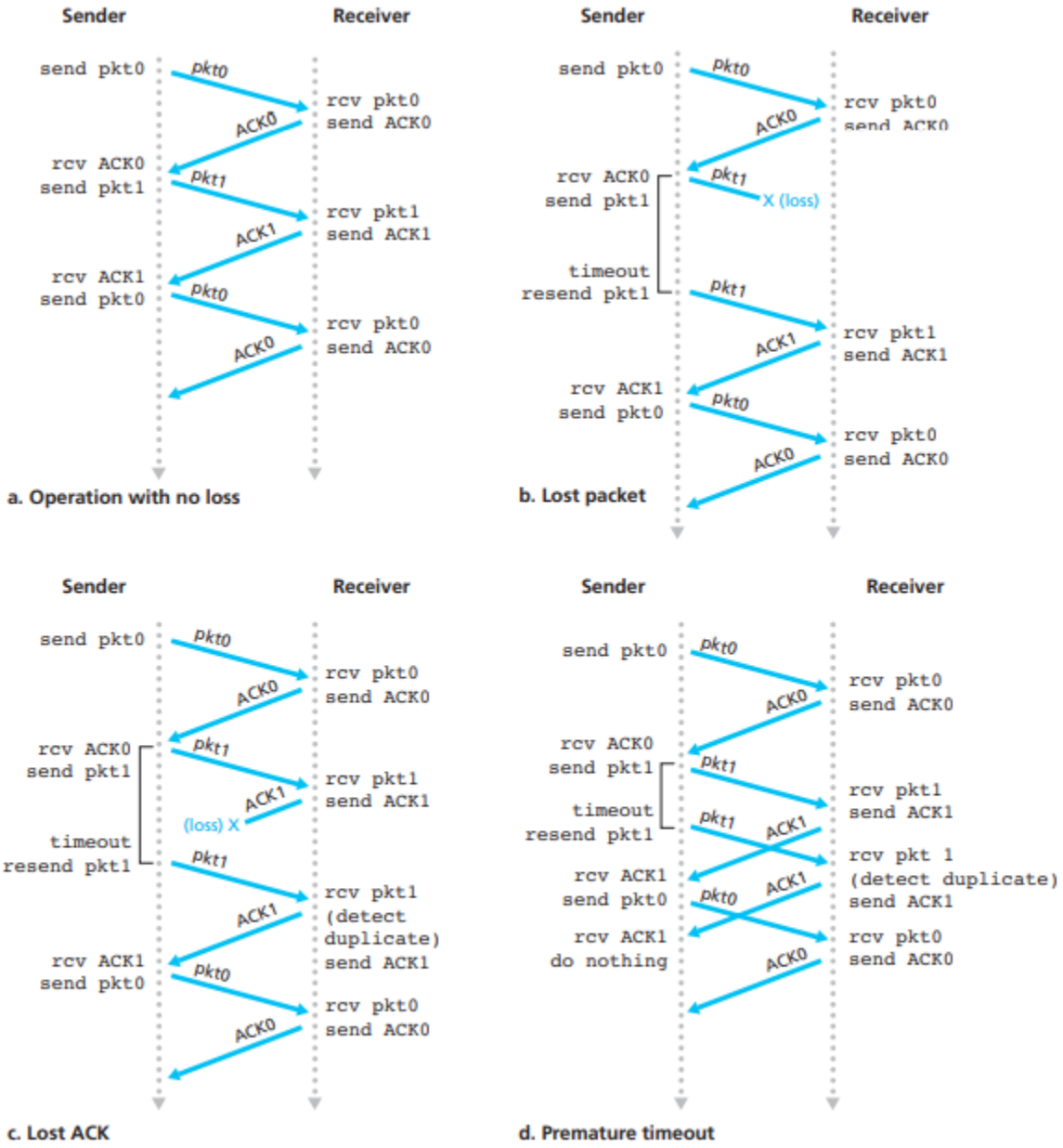


Figure 3.16 ♦ Operation of `rdt3.0`, the alternating-bit protocol

- In Figures (b)–(d), the send-side brackets indicate the times at which a timer is set and later times out.
- Because packet sequence numbers alternate between 0 and 1, protocol rdt3.0 is also known as **the alternating-bit protocol.**

- In **a Go-Back-N (GBN)** protocol, the sender is allowed to transmit multiple packets without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, N, of unacknowledged packets in the pipeline.

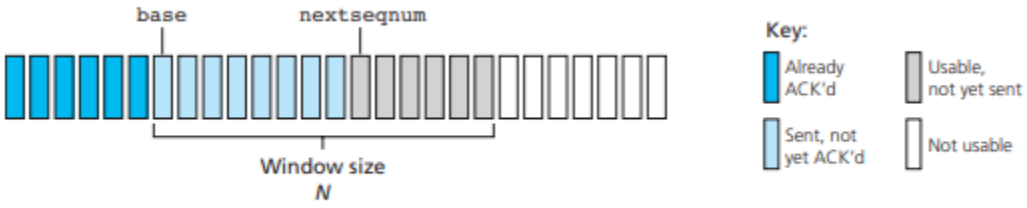


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

- Figure above shows the sender's view of the range of sequence numbers in a GBN protocol.
- Define **base** to be the sequence number of the oldest unacknowledged packet and **nextseqnum** to be the smallest unused sequence number (that is, the sequence number of the next packet to be sent).
- Sequence numbers in the interval [0, base-1] correspond to packets that have already been transmitted and acknowledged.
- The interval [base, nextseqnum-1] corresponds to packets that have been sent but not yet acknowledged.
- Sequence numbers in the interval [nextseqnum, base+N-1] can be used for packets that can be sent immediately, should data arrive from the upper layer.
- Finally, **sequence numbers** greater than or equal to base+N cannot be used until an unacknowledged packet currently in the pipeline has been acknowledged.
- The range of permissible sequence numbers for transmitted but not yet acknowledged packets can be viewed as a window of size N over the range of sequence numbers.
- As the protocol operates, this window slides forward over the sequence number space. For this reason, N is often referred to as the **window size** and the GBN protocol as a **sliding-window protocol.**
- A packet's sequence number is carried in a fixed-length field in the packet header. If k is the number of bits in the packet sequence number field, the range of sequence numbers is thus $[0,2^k - 1]$.
- The sequence number space can be thought of as a ring of size 2^k, where sequence number $2^k - 1$ is immediately followed by sequence number 0.
- **The GBN sender must respond to three types of events:**

4. **Invocation from above**.
    a. When **rdt_send()** is called from above, the sender first checks to see if the window is full, that is, whether there are N outstanding, unacknowledged packets.
    b. If the window is not full, a packet is created and sent, and variables are appropriately updated.
    c. If the window is full, the sender simply returns the data back to the upper layer, an implicit indication that the window is full. The upper layer would have to try again.
5. **Receipt of an ACK**.

    In our GBN protocol, an acknowledgment for a packet with sequence number n will be taken to be a cumulative acknowledgment, indicating that all packets with a sequence number up to and including n have been correctly received at the receiver.

6. **A timeout event**.

    A timer will be used to recover from lost data or acknowledgment packets. If a timeout occurs, the sender resends all packets that have been previously sent but that have not yet been acknowledged.

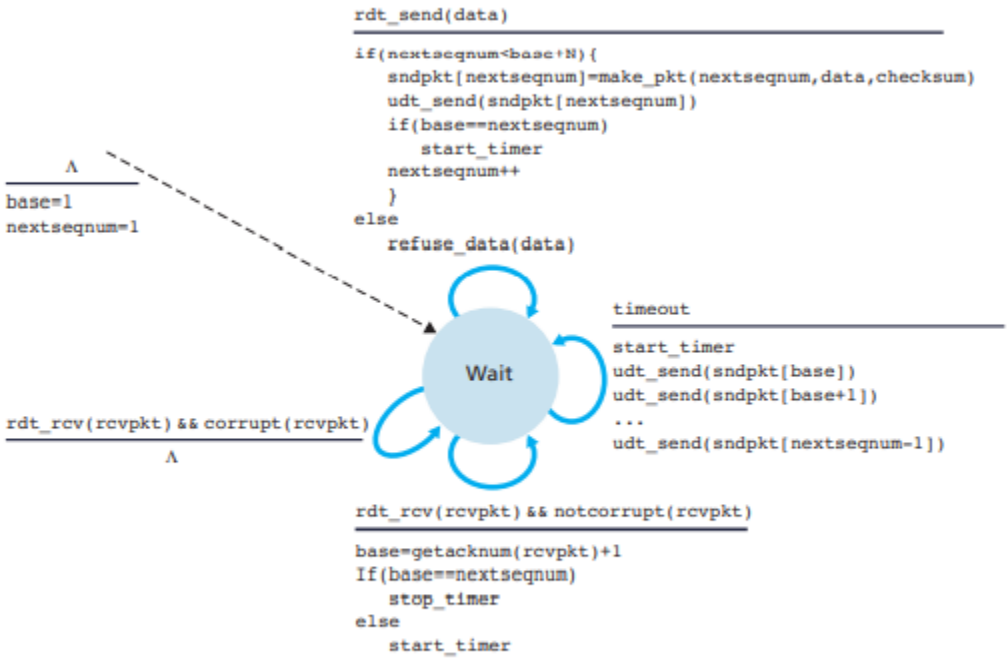


Figure 3.20 ♦ Extended FSM description of GBN sender

3. **Sender** in above Figure uses only a single timer, which can be thought of as a timer for the oldest transmitted but not yet acknowledged packet.
    - If an ACK is received but there are still additional transmitted but not yet acknowledged packets, the timer is restarted.
    - If there are no outstanding, unacknowledged packets, the timer is stopped.
4. The **receiver's action:** If a packet with sequence number n is received correctly and is in order, the receiver sends an ACK for packet n and delivers the data portion of the packet to the upper layer.
    a. In all other cases, the receiver discards the packet and resends an ACK for the most recently received in-order packet.
    b. Since packets are delivered one at a time to the upper layer, if packet k has been received and delivered, then all packets with a sequence number lower than k have also been delivered.
    c. Thus, cumulative acknowledgments is used for GBN.
    d. In **GBN protocol**, the receiver discards **out-of-order** packets because the receiver must deliver data in order to the upper layer.

- Suppose the packet n is expected, but packet **n + 1** arrives. Because data must be delivered in order, the receiver could buffer (save) packet **n + 1** and then deliver this packet to the upper layer after it had later received and delivered packet n.
- If packet n is lost, both it and packet n + 1 will eventually be retransmitted as a result of the GBN retransmission rule at the sender.
- Thus, the receiver can simply discard packet **n + 1.**
- The advantage of this approach is the receiver need not buffer **any** out-of-order packets.
- Thus, while the sender must maintain the upper and lower bounds of its window and the position of **nextseqnum** within this window.
- The only piece of information the receiver need maintain is the sequence number of the next in-order packet.
- This value is held in the variable expected **seqnum**, shown in the receiver FSM in Figure below.

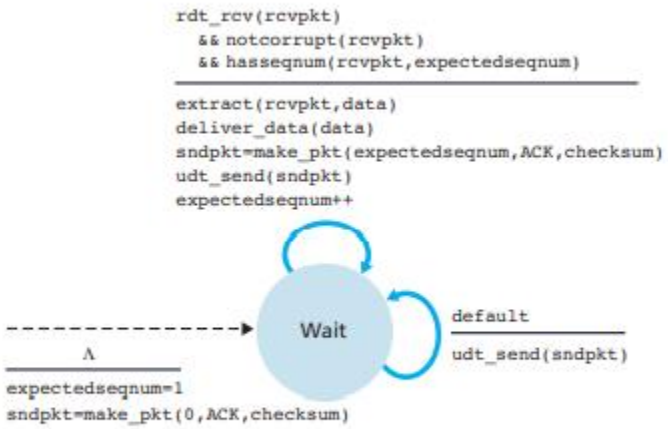


Figure 3.21 ♦ Extended FSM description of GBN receiver

The **disadvantage** of discarding a correctly received packet is that the subsequent retransmission of that packet might be lost or garbled and thus even more retransmissions would be required.

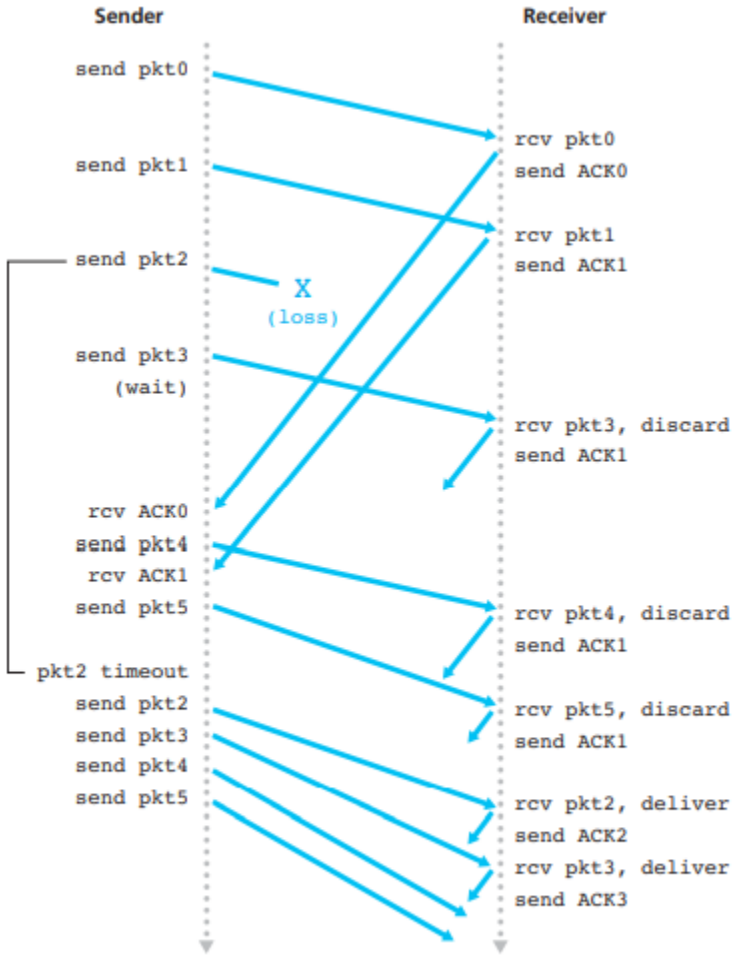


Figure 3.22 ♦ Go-Back-N in operation

- Figure above shows the operation of the GBN protocol for the case of a window size of **four packets**.
- Because of this window size limitation, the sender sends packets 0 through 3 but then must wait for one or more of these packets to be acknowledged before proceeding.
- As each successive **ACK** (for example, ACK0 and ACK1) is received, the window slides forward and the sender can transmit one new packet (pkt4 and pkt5, respectively).
- On the receiver side, packet 2 is lost and thus packets 3, 4, and 5 are found to be out of order and are discarded.
- In the sender, the events would be:

  (1) a call from the upper-layer entity to invoke **rdt_send(),**

  (2) a timer interrupt,

  (3) a call from the lower layer to invoke **rdt_rcv()** when a packet arrives.

3.4.4 Selective Repeat (SR):

- The GBN protocol allows the sender to potentially "fill the pipeline" with packets, thus avoiding the channel utilization problems.
- **GBN has performance problems:** When the window size and bandwidth-delay product are both large, many packets can be in the pipeline.
- A single packet error can thus cause GBN to retransmit a large number of packets, many unnecessarily.
- As the probability of channel errors increases, the pipeline can become filled with these unnecessary retransmissions.
- **Selective-repeat** protocols avoid unnecessary retransmissions by having the sender retransmit only those packets that it suspects were received in error (that is, were lost or corrupted) at the receiver.
- This needed retransmission will require that the receiver individually acknowledge correctly received packets.
- A **window size of N** will again be used to limit the number of outstanding, unacknowledged packets in the pipeline.
- The sender will have already received ACKs for some of the packets in the window. Figure shows the SR sender's view of the sequence number space.
- Figure details the various actions taken by **the SR sender**.
- **The SR receiver** will acknowledge a correctly received packet whether or not it is in order.
- **Out-of-order** packets are buffered until any missing packets (that is, packets with lower sequence numbers) are received, at which point a batch of packets can be delivered in order to the upper layer.

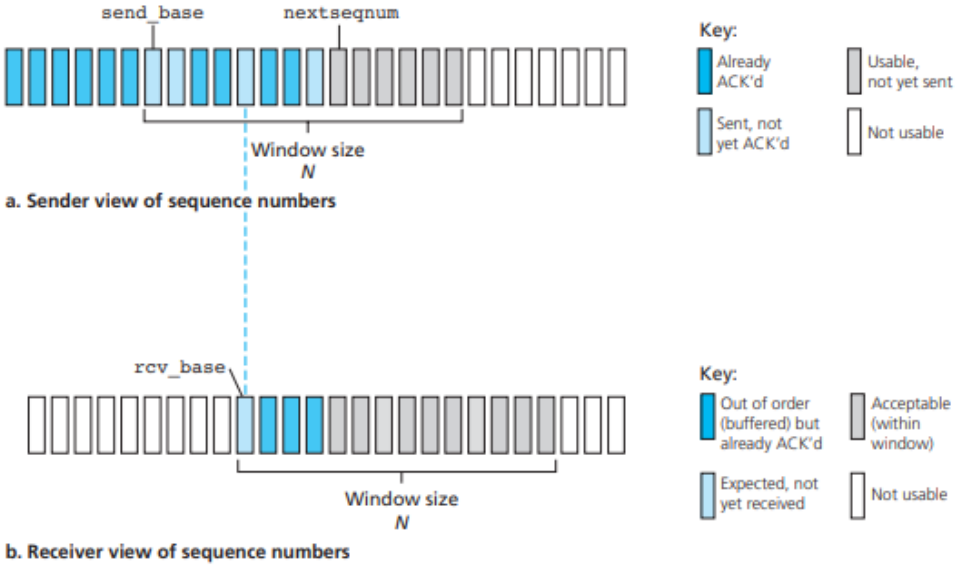


Figure 3.23 ♦ Selective-repeat (SR) sender and receiver views of sequence-number space

Figure above itemizes the various actions taken by the SR receiver.

In Step 2 in Figure above, the receiver reacknowledges (rather than ignores) already received packets with certain sequence numbers below the current window base.

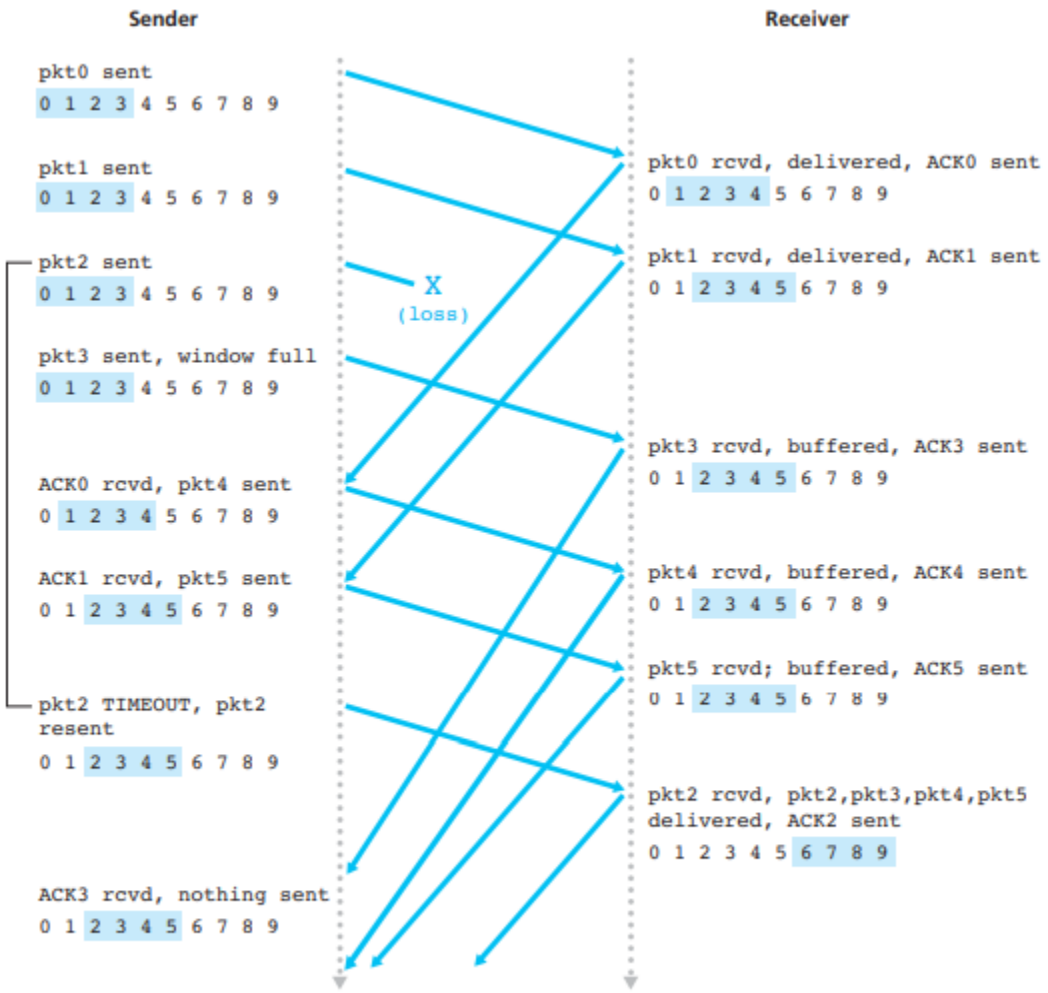


Figure 3.26 ♦ SR operation

- Above Figure shows an example of SR operation in the presence of lost packets.
- Here, the receiver initially buffers packets 3, 4, and 5, and delivers them together with packet 2 to the upper layer when packet 2 is finally received.
- **For example**, if there is no ACK for packet **send_base** propagating from the receiver to the sender, the sender will eventually retransmit packet **send_base**, even though it is clear that the receiver has already received.

SR Sender Events and actions :

1. **Data received from above.**
   a. When data is received from above, the SR sender checks the next available sequence number for the packet.
   b. If the sequence number is within the sender's window, the data is packetized and sent; otherwise, it is either buffered or returned to the upper layer for later transmission, as in GBN.

2. **Timeout.**

a. Timers are used to protect against lost packets.
b. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout.
c. A single hardware timer can be used to mimic the operation of multiple logical timers

3. **ACK received**.
   a. If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window.
   b. If the packet's sequence number is equal to **send_base**, the window base is moved forward to the unacknowledged packet with the smallest sequence number.
   c. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

SR receiver events and actions :

1.**Packet with sequence number in [rcv_base, rcv_base+N-1] is correctly received:**

a. The received packet falls within the receiver's window and a selective ACK packet is returned to the sender.
b. If the packet was not previously received, it is buffered.
c. If this packet has a sequence number equal to the base of the receive window (rcv_base) then this packet, and any previously buffered and consecutively numbered (beginning with rcv_base) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer.
d. As an example, consider above Figure.
e. When a packet with a sequence number of rcv_base=2 is received, it and packets 3, 4, and 5 can be delivered to the upper layer.

2. **Packet with sequence number in [rcv_base-N, rcv_base-1] is correctly received.**

In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.

3. **Otherwise.**

a. Ignore the packet. that packet. If the receiver were not to acknowledge this packet, the sender's window would never move forward.

b. The sender and receiver will not always have an identical view of what has been received correctly and what has not. For SR protocols, this means that the sender and receiver windows will not always coincide.

c. The lack of synchronization between sender and receiver windows has important consequences when we are faced with the reality of a finite range of sequence numbers.

d. For example, with a finite range of four packet sequence numbers, 0, 1, 2, 3, and a window size of three.

f. Suppose packets 0 through 2 are transmitted and correctly received and acknowledged at the receiver.
g. At this point, the receiver's window is over the fourth, fifth, and sixth packets, which have sequence numbers 3, 0, and 1, respectively.

Consider two scenarios:

1. **In the first scenario**, shown in Figure (a) below, the ACKs for the first three packets are lost and the sender retransmits these packets. The receiver thus next receives a packet with sequence number 0—a copy of the first packet sent.
2. **In the second scenario**, shown in Figure (b) below, the ACKs for the first three packets are all delivered correctly. The sender thus moves its window forward and sends the fourth, fifth, and sixth packets, with sequence numbers 3, 0, and 1, respectively. The packet with sequence number 3 is lost, but the packet with sequence number 0 arrives—a packet containing new data.

- Consider **the receiver's view point in Figure below**, since the receiver cannot "see" the actions taken by the sender.
- All the receiver observes is the sequence of messages it receives from the channel and sends into the channel.
- **The two scenarios in Figure are identical.**

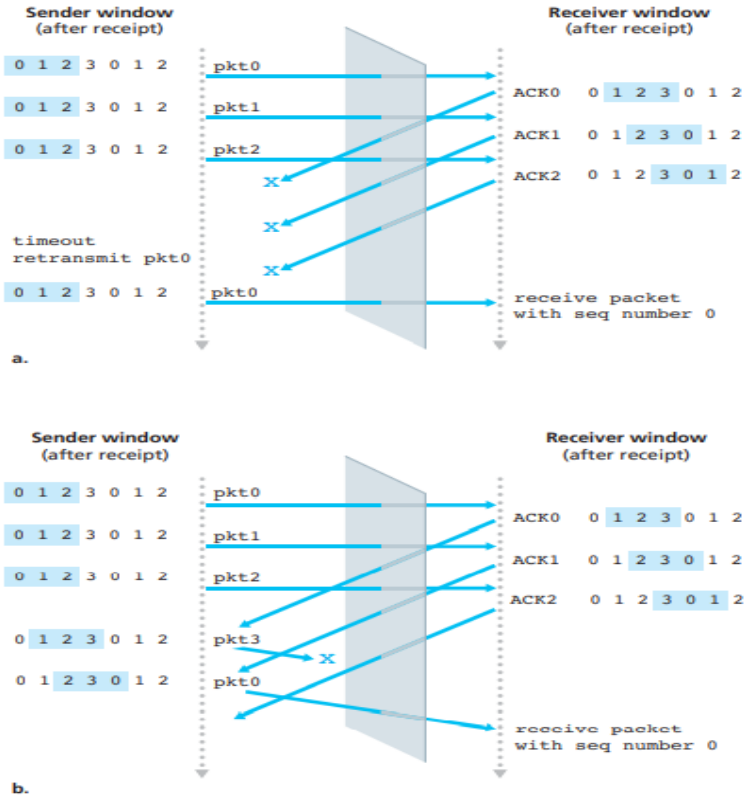


Figure 3.27 ♦ SR receiver dilemma with too-large windows: A new packet or a retransmission?

- There is no way of distinguishing the retransmission of the first packet from an original transmission of the fifth packet.

Solution is the window size must be less than or equal to half the size of the sequence number space for SR protocols.

3.4.5 Summary of Reliable Data Transfer Mechanisms and their Use

Table 2.2: Summary of reliable data transfer mechanisms and their use

Mechanism	Use, Comments
Checksum	Used to detect bit errors in a transmitted packet.
Timer	Used to timeout/retransmit a packet because the packet (or its ACK) was lost. Because timeouts can occur when a packet is delayed but not lost, duplicate copies of a packet may be received by a receiver.
Sequence-number	Used for sequential numbering of packets of data flowing from sender to receiver. Gaps in the sequence-numbers of received packets allow the receiver to detect a lost packet. Packets with duplicate sequence-numbers allow the receiver to detect

	duplicate copies of a packet.
Acknowledgment	Used by the receiver to tell the sender that a packet or set of packets has been received correctly. Acknowledgments will typically carry the sequence-number of the packet or packets being acknowledged. Acknowledgments may be individual or cumulative, depending on the protocol.
Negative acknowledgment	Used by the receiver to tell the sender that a packet has not been received correctly. Negative acknowledgments will typically carry the sequence-number of the packet that was not received correctly.
Window, pipelining	The sender may be restricted to sending only packets with sequence-numbers that fall within a given range. By allowing multiple packets to be transmitted but not yet acknowledged, sender utilization can be increased over a stop-and-wait mode of operation.

3.5 Connection-Oriented Transport: TCP

- TCP is a reliable connection-oriented protocol.
  - Connection-oriented means a connection is established b/w sender & receiver before sending the data.
  - Reliable service means TCP guarantees that the data will arrive to destination-process
  - correctly.
- TCP provides flow-control, error-control and congestion-control.

.5.1 The TCP Connection:

- TCP is said to be **connection-oriented** because before one application process can begin to send data to another, the two processes must first "handshake" with each other—that is, they must send some preliminary segments to each other to establish the parameters of the ensuing data transfer.
- TCP connection establishment, both sides of the connection will initialize many TCP state variables associated with the TCP connection.
- TCP protocol runs only in the end systems and not in the intermediate network elements (routers and link-layer switches), the intermediate network elements do not maintain TCP connection state.
- A TCP connection provides a **full-duplex service:** If there is a TCP connection between Process A on one host and Process B on another host, then application layer data can flow from Process A to Process B at the same time as application layer data flows from Process B to Process A.
- A TCP connection is also always **point-to-point**, that is, between a single sender and a single receiver. So-called "**multicasting**"—the transfer of data from one sender to many receivers in a single send operation—is not possible with TCP.
- **Python client program command:**

clientSocket.connect((serverName,serverPort))

- where **serverName** is the name of the server and **serverPort** identifies the process on the server.
- TCP in the client then proceeds to establish a TCP connection with TCP in the server.
- Client first sends a special TCP segment; the server responds with a second special TCP segment and finally the client responds again with a third special segment**.**
- The first two segments carry no payload, that is, no application-layer data; the third of these segments may carry a payload. Because three segments are sent between the two hosts, this connection- establishment procedure is often referred to as a **three-way handshake**
- Once a TCP connection is established, the two application processes can send data to each other. Consider the sending of data from the client process to the server process.
- The client process passes a stream of data through the socket.
- Once the data passes through the socket, the data is in the hands of TCP running in the client.

- As Figure shows, TCP directs this data to the connection's **send buffer,** which is one of the buffers that is set aside during the initial three-way handshake.

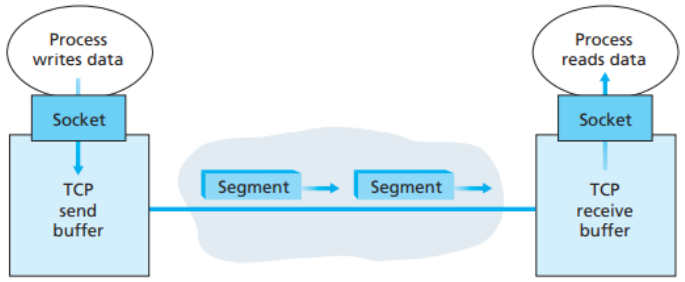


Figure 3.28 ♦ TCP send and receive buffers

- From time to time, TCP will grab chunks of data from the send buffer and pass the data to the network layer.
- The maximum amount of data that can be grabbed and placed in a segment is limited by **the maximum segment size (MSS).**
- The MSS is set by first determining the length of the largest link-layer frame that can be sent by the local sending host (maximum transmission unit, MTU) and then setting the MSS to ensure that a TCP segment (when encapsulated in an IP datagram) plus the TCP/IP header length (typically 40 bytes) will fit into a single link-layer frame.
- TCP pairs each chunk of client data with a TCP header, thereby forming TCP segments. The segments are passed down to the network layer, where they are separately encapsulated within network-layer IP datagrams.
- The IP datagrams are then sent into the network. When TCP receives a segment at the other end, the segment's data is placed in the TCP connection's receive buffer, as shown in Figure above.
- **Send & Receive Buffers**
  - As shown in Figure 3.28, consider sending data from the client-process to the server-process.

    At Sender

    i) The client-process passes a stream-of-data through the socket.
    ii) Then, TCP forwards the data to the send-buffer.
    iii) Each chunk-of-data is appended with a header to form a segment.
    iv) The segments are sent into the network.

    At Receiver

    i) The segment's data is placed in the receive-buffer.
    ii) The application reads the stream-of-data from the receive-buffer.

.5.2 TCP Segment Structure:

- The TCP segment consists of header fields and a data field. The data field contains a chunk of application data. MSS limits the maximum size of a segment's data field.
- When TCP sends a large file, such as an image as part of a Web page, it typically breaks the file into chunks of size MSS.
- Interactive applications , often transmit data chunks that are smaller than the MSS;

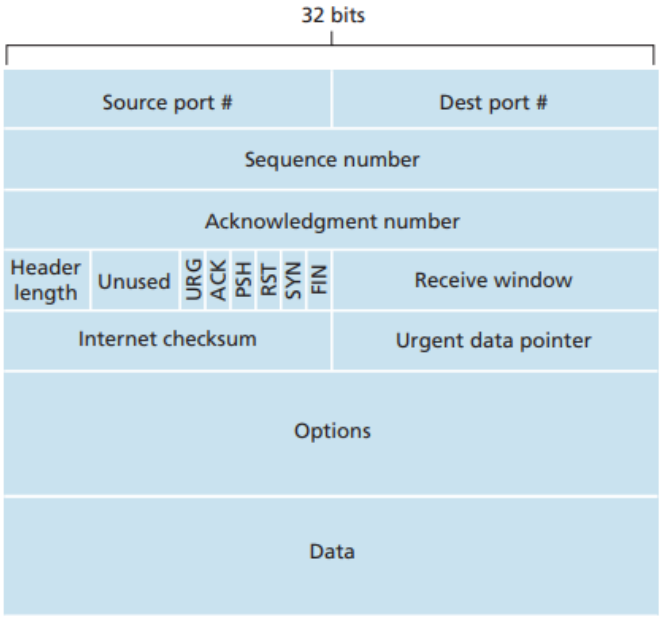


Figure 3.29 ♦ TCP segment structure

- Figure above shows the structure of the TCP segment. The header includes source and destination port numbers, which are used for multiplexing / demultiplexing data from/to upper-layer applications.
- **Header includes a checksum field for error detection.**
  A TCP segment header also contains the following fields:
  - The 32-bit **sequence number field** and the 32-bit **acknowledgment number field** are used by the TCP sender and receiver in implementing a reliable data transfer service.
  - The 16-bit receive **window field** is used for flow control.
  - The **4-bit header length field** specifies the length of the TCP header in 32-bit words.
  - The **TCP header** can be of variable length due to the TCP options field.
  - The **optional and variable-length** options field is used when a sender and receiver negotiate the **maximum segment size** (MSS) or as a window scaling factor for use in high-speed networks.
  - A **time-stamping option** is also defined.
  - **The flag field contains 6 bits.**

- **The ACK bit** is used to indicate that the value carried in the acknowledgment field is valid; that is, the segment contains an acknowledgment for a segment that has been successfully received.
- **The RST, SYN, and FIN bits** are used for connection setup and teardown.
- Setting the **PSH bit** indicates that the receiver should pass the data to the upper layer immediately.
- **URG bit** is used to indicate that there is data in this segment that the sending-side upper-layer entity has marked as "urgent."
- The location of the last byte of this urgent data is indicated by the 16-bit urgent data pointer field.

- TCP must inform the receiving- side upper-layer entity when urgent data exists and pass it a pointer to the end of the urgent data.

Sequence Numbers and Acknowledgment Numbers

- Two of the most important fields in the TCP segment header are the sequence number field and the acknowledgment number field.
- These fields are a critical part of TCP's reliable data transfer service.
- TCP views data as an unstructured, but ordered, stream of bytes.
- TCP's use of sequence numbers reflects this view in that sequence numbers are over the stream of transmitted bytes and not over the series of transmitted segments.
- The sequence number for a segment is therefore the byte-stream number of the first byte in the segment.
- **Example.** Suppose a process in Host A wants to send a stream of data to a process in Host B over a TCP connection.
  - The TCP in Host A will implicitly number each byte in the data stream.
  - Suppose that the data stream consists of a file consisting of 500,000 bytes, that the MSS is 1,000 bytes, and that the first byte of the data stream is numbered 0.

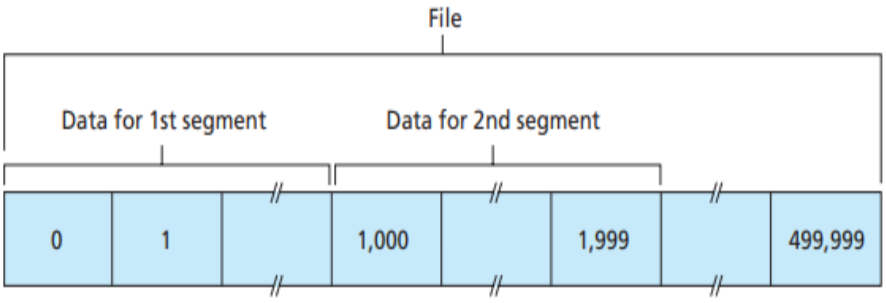


Figure 3.30 ♦ Dividing file data into TCP segments

- As shown in Figure above, TCP constructs 500 segments out of the data stream.
- The first segment gets assigned sequence number 0, the second segment gets assigned sequence number 1,000, the third segment gets assigned sequence number 2,000, and so on.
- Each sequence number is inserted in the sequence number field in the header of the appropriate TCP segment.
- Consider **acknowledgment numbers.**
- TCP is full-duplex, so that Host A may be receiving data from Host B while it sends data to Host B (as part of the same TCP connection).
- The acknowledgment number that Host A puts in its segment is the sequence number of the next byte Host A is expecting from Host B.
- Suppose that Host A has received all bytes numbered 0 through 535 from B and suppose that it is aboutto send a segment to Host B.
- Host A is waiting for byte 536 and all the subsequentbytes in Host B's data stream. So Host A puts 536 in the acknowledgment numberfield of the segment it sends to B.
- Suppose Host A has received one segment from Host B containing bytes 0 through 535 and another segment containing bytes 900 through 1,000.
- For some reason Host A has not yet received bytes 536 through 899. In this example, Host A is still waiting for byte 536 (and beyond) in order to re-create B's data stream.
- Thus, A's next segment to B will contain 536 in the acknowledgment number field.
- Because TCP only acknowledges bytes up to the first missing byte in the stream, TCP is said to provide cumulative acknowledgments.

- Host A received the third segment (bytes 900 through 1,000) before receiving the second segment (bytes 536 through 899). Thus, the third segment arrived out of order.
  The issue is:
- Host A receives out-of-order segments in a TCP connection.
  There are two choices:
  (1) the receiver immediately discards out-of-order segments
  (2) the receiver keeps the out-of-order bytes and waits for the missing bytes to fill in the gaps.
- **In Figure above,** we assumed that the initial sequence number was zero.
- Both sides of a TCP connection randomly choose an initial sequence number.

Telnet: A Case Study for Sequence and Acknowledgment Numbers:

- Telnet is a popular application-layer protocol used for remote-login.
- Telnet runs over TCP.
  - Suppose Host A initiates a Telnet session with Host B.
  - Because Host A initiates the session, it is labeled the client, and Host B is labeled the server.
  - Each character typed by the user (at the client) will be sent to the remote host; the remote host will send back a copy of each character, which will be displayed on the Telnet user's screen.
  - "Echo back" is used to ensure that characters seen by the Telnet user have already been received and processed at the remote site.
  - Each character thus traverses the network twice between the time the user hits the key and the time the character is displayed on the user's monitor.
  - Suppose the user types a single **letter, 'C'.**
  - As shown in Figure below, the starting sequence numbers are 42 and 79 for the client and server, respectively.
  - The sequence number of a segment is the sequence number of the first byte in the data field.
  - Thus, the first segment sent from the client will have sequence number 42; the first segment sent from the server will have sequence number 79.
  - The acknowledgment number is the sequence number of the next byte of data that the host is waiting for.

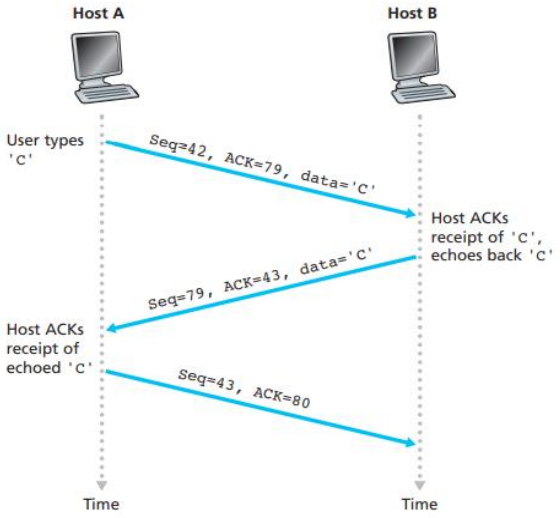


Figure 3.31 ♦ Sequence and acknowledgment numbers for a simple Telnet application over TCP

- After the TCP connection is established but before any data is sent, the client is waiting for byte 79 and the server is waiting for byte 42.

As shown in Figure above, three segments are sent.

1. **The first segment** is sent from the client to the server, containing the 1-byte ASCII representation of the letter 'C' in its data field.
   - This first segment also has 42 in its sequence number field.
   - Because the client has not yet received any data from the server, this first segment will have 79 in its acknowledgment number field.
2. **The second segment** is sent from the server to the client.
   - It serves a dual purpose.
   - First it provides an acknowledgment of the data the server has received. By putting 43 in the acknowledgment field, the server is telling the client that it has successfully received everything up through byte 42 and is now waiting for bytes 43 onward.
   - The second purpose of this segment is to echo back the **letter 'C.'**
   - Thus, the second segment has the ASCII representation of 'C' in its data field. This second segment has the sequence number 79, the initial sequence number of the server-to client data flow of this TCP connection, as this is the very first byte of data that the server is sending.

- This acknowledgment is said to be piggybacked on the server-to-client data segment.

3. **The third segment** is sent from the client to the server.
   - Its purpose is to acknowledge the data it has received from the server.
   - This segment has an empty data field .
   - The segment has 80 in the acknowledgment number field because the client has received the stream of bytes up through byte sequence number 79 and it is now waiting for bytes 80 .

.5.3 Round-Trip Time Estimation and Timeout:

- TCP, uses a timeout/retransmit mechanism to recover from lost segments. The timeout should be larger than the connection's round-trip time (RTT), that is, the time from when a segment is sent until it is acknowledged.

Estimating the Round-Trip Time

- TCP estimates the round-trip time between sender and receiver.

This is accomplished as follows:

- The sample RTT, denoted **SampleRTT**, for a segment is the amount of time between when the segment is sent and when an acknowledgment for the segment is received.
- Instead of measuring a SampleRTT for every transmitted segment, most TCP implementations take only one SampleRTT measurement at a time.
- That is, at any point in time, the SampleRTT is being estimated for only one of the transmitted but currently unacknowledged segments, leading to a new value of SampleRTT approximately once every RTT.
- Also, TCP never computes a SampleRTT for a segment that has been retransmitted; it only measures SampleRTT for segments that have been transmitted once .
- The **SampleRTT** values will fluctuate from segment to segment due to congestion in the routers and to the varying load on the end systems.
- Because of this fluctuation, any given SampleRTT value may be atypical. In order to estimate a typical RTT, it is therefore natural to take some sort of average of the SampleRTT values.
- TCP maintains an average, called EstimatedRTT, of the SampleRTT values.
- Upon obtaining a new SampleRTT, TCP updates EstimatedRTT according to the following formula:

EstimatedRTT = (1 – α) • EstimatedRTT + α • SampleRTT

- The formula above is written in the form of a programming-language statement—the new value of EstimatedRTT is a weighted combination of the previous value of EstimatedRTT and the new value for SampleRTT.
- The recommended value of α is **α = 0.125 (that is, 1/8)** in which case the formula above becomes:

EstimatedRTT = 0.875 • EstimatedRTT + 0.125 • SampleRTT

- EstimatedRTT is a weighted average of the SampleRTT values.
- In statistics, such an average is called an exponential weighted moving average (EWMA).
- The word "exponential" appears in EWMA because the weight of a given SampleRTT decays exponentially fast as the updates proceed.

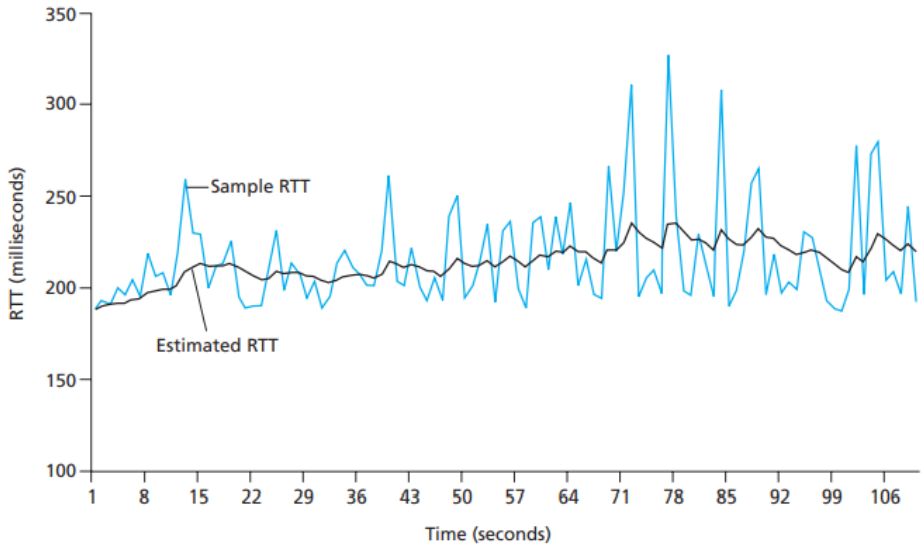


Figure 3.32 ♦ RTT samples and RTT estimates

- Figure shows the SampleRTT values and EstimatedRTT for a value of α = 1/8 for a TCP connection between say gaia.cs.umass.edu(site 1) to fantasia.eurecom.fr(site 2).
- Clearly, the variations in the SampleRTT are smoothed out in the computation of the EstimatedRTT
- In addition to having an estimate of the RTT, it is also valuable to have a measure of the variability of the RTT.
- The RTT variation, DevRTT, as an estimate of how much SampleRTT typically deviates from EstimatedRTT:

DevRTT = (1 – β) • DevRTT + β •| SampleRTT – EstimatedRTT |

- DevRTT is an EWMA of the difference between SampleRTT and EstimatedRTT.
- If the SampleRTT values have little fluctuation, then DevRTT will be small; if there is a lot of fluctuation, DevRTT will be large. The recommended value of **β is 0.25.**

Setting and Managing the Retransmission Timeout Interval

- Given values of EstimatedRTT and DevRTT, the interval should be greater than or equal to EstimatedRTT, or unnecessary retransmissions would be sent. But the timeout interval should not be too much larger than EstimatedRTT; otherwise, when a segmentis lost, TCP would not quickly retransmit the segment, leading to large data transfer delays.
- It is therefore desirable to set the timeout equal to the EstimatedRTT plus some margin.
- The margin should be large when there is a lot of fluctuation in the SampleRTT values; it should be small when there is little fluctuation.
- The value of DevRTT should thus come into play here.
- All of these considerations are taken into account in TCP's method for determining the retransmission timeout interval:

TimeoutInterval = EstimatedRTT + 4 • DevRTT

- When a timeout occurs, the value of TimeoutInterval is doubled to avoid a premature timeout occurring for a subsequent segment that will soon be acknowledged.
- As soon as a segment is received and EstimatedRTT is updated, the TimeoutInterval is again computed using the formula above.

.5.4 Reliable Data Transfer:

- The Internet's network-layer service (IP service) is **unreliable.**
- IP does not guarantee datagram delivery, **does not guarantee in-order delivery** of datagrams and does not guarantee the integrity of the data in the datagrams. With IP service, datagrams can overflow router buffers and never reach their destination, datagrams can arrive out of order, and bits in the datagram can get corrupted (flipped from 0 to 1 and vice versa).
- Transport-layer segments are carried across the network by IP datagrams,

transport-layer segments can suffer from these problems as well.

- TCP creates a reliable data transfer service on top of IP's unreliable best effort service.
- TCP's reliable data transfer service ensures that the data stream that a process reads out of its TCP receive buffer is uncorrupted, without gaps, without duplication, and in sequence;

- that is, the byte stream is exactly the same byte stream that was sent by the end system on the other side of the connection.
- The recommended TCP timer management procedures use only a single retransmission timer, even if there are multiple transmitted but not yet acknowledged segments.
- Suppose that data is being sent in only one direction, from Host A to Host B, and that Host A is sending a large file.
- **Three major events related to data transmission and retransmission in the TCP sender:**
  1. data received from application above;
  2. timer timeout;
  3. ACK receipt.

- Upon the occurrence of the first major event, TCP receives data from the application, encapsulates the data in a segment, and passes the segment to IP.
  - Each segment includes a sequence number that is the byte-stream number of the first data byte in the segment.
  - If the timer is already not running for some other segment, TCP starts the timer when the segment is passed to IP.
  - The expiration interval for this timer is the TimeoutInterval, which is calculated from

  EstimatedRTT and DevRTT

Simplified TCP sender :

```
/* Assume sender is not constrained by TCP flow or congestion control, that data from
    above
is less than MSS in size, and that data transfer is in one direction only. */
NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber
loop (forever) {
switch(event)
event: data received from application above
```

```
create TCP segment with sequence number NextSeqNum
if (timer currently not running)
start timer
pass segment to IP
NextSeqNum=NextSeqNum+length(data)
break;
event: timer timeout
retransmit not-yet-acknowledged segment with
smallest sequence number
start timer
break;
event: ACK received, with ACK field value of y
if (y > SendBase) {
SendBase=y
    if (there are currently any not-yet-acknowledged segments)
start timer
}
break;
} /* end of loop forever */
```

- The **second major event** is the timeout. TCP responds to the timeout event by retransmitting the segment that caused the timeout.
- TCP then restarts the **timer.**
- The **third major event** that must be handled by the TCP sender is the arrival of an acknowledgment segment (ACK) from the receiver.
- On the occurrence of this event, TCP compares the ACK value y with its variable SendBase.
- The TCP state variable SendBase is the sequence number of the oldest unacknowledged byte.
- TCP uses cumulative acknowledgments, so that y acknowledges the receipt of all bytes before byte number y.
- If y > SendBase, then the ACK is acknowledging one or more previously unacknowledged segments.
- Thus the sender updates its SendBase variable; it also restarts the timer if there currently are any not-yet-acknowledged segments

A Few Interesting Scenarios:

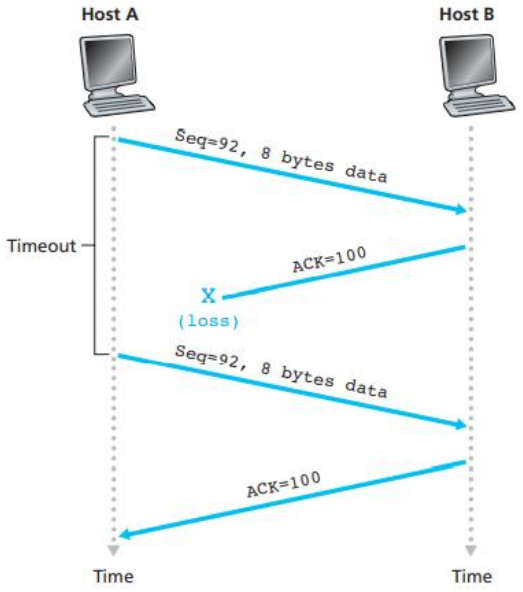


Figure 3.34 ♦ Retransmission due to a lost acknowledgment

Figure above depicts **the first scenario,**

- In which Host A sends one segment to Host B.
- Suppose that this segment has sequence number 92 and contains 8 bytes of data.
- After sending this segment, Host A waits for a segment from B with acknowledgment number 100.
- Although the segment from A is received at B, the acknowledgment from B to A gets lost.
- In this case, the timeout event occurs, and Host A retransmits the same segment. When Host B receives the retransmission, it observes from the sequence number that the segment contains data that has already been received.
- Thus, TCP in Host B will discard the bytes in the retransmitted segment.

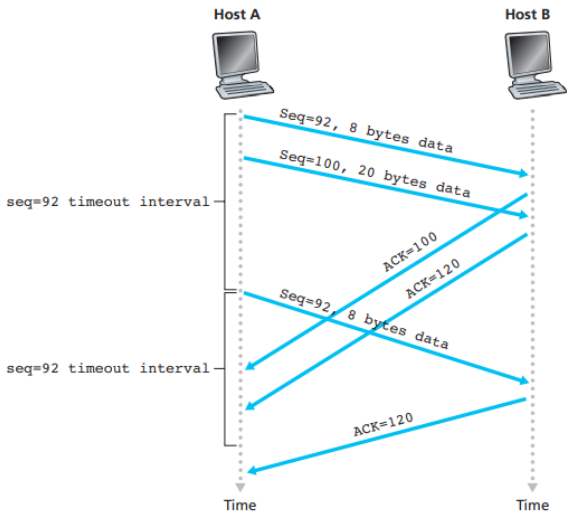


Figure 3.35 ♦ Segment 100 not retransmitted

In a second scenario, shown in Figure,

- Host A sends two segments back to back.
- The first segment has sequence number 92 and 8 bytes of data, and the second segment has sequence number 100 and 20 bytes of data.
- Suppose that both segments arrive intact at B, and B sends two separate acknowledgments for each of these segments.
- The first of these acknowledgments has acknowledgment number 100; the second has acknowledgment number 120.
- Suppose now that neither of the acknowledgments arrives at Host A before the timeout.
- When the timeout event occurs, Host A resends the first segment with sequence number 92 and restarts the timer.
- As long as the ACK for the second segment arrives before the new timeout, the second segment will not be retransmitted.

In a third and final scenario,

- Suppose Host A sends the two segments, exactly as in the second example.
- The acknowledgment of the first segment is lost in the network, but just before the timeout event, Host A receives an acknowledgment with acknowledgment number 120.
- Host A therefore knows that Host B has received everything up through byte 119; so Host A does not resend either of the two segments. This scenario is illustrated in Figure below.

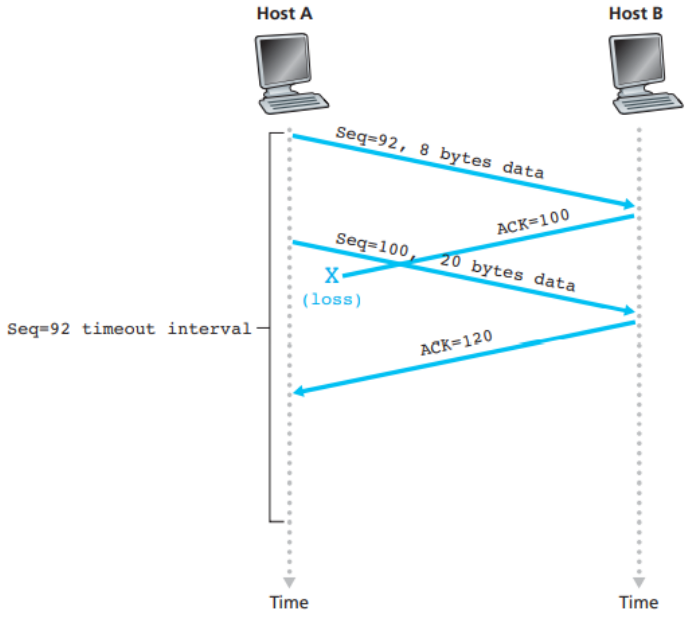


Figure 3.36 ♦ A cumulative acknowledgment avoids retransmission of the first segment

Doubling the Timeout Interval

- The length of the timeout interval after a timer expiration.
- In this modification, whenever the timeout event occurs, TCP retransmits the not-yet acknowledged segment with the smallest sequence number, as described above.
- But each time TCP retransmits, it sets the next timeout interval to twice the previous value, rather than deriving it from the last **EstimatedRTT and DevRTT .**
- **For example**, suppose TimeoutInterval associated with the oldest not yet acknowledged segment is .75 sec when the timer first expires.
- TCP will then retransmit this segment and set the new expiration time to 1.5 sec.
- If the timer expires again 1.5 sec later, TCP will again retransmit this segment, now setting the expiration time to 3.0 sec.
- Thus the intervals grow exponentially after each retransmission.
- However, whenever the timer is started after either of the two other events (that is, data received from application above, and ACK received), the TimeoutInterval is derived from the most recent values of EstimatedRTT and DevRTT.
- This modification provides a limited form of congestion control.
- The timer expiration is most likely caused by congestion in the network, that is, too many packets arriving at one (or more) router queues in the path between the source and destination, causing packets to be dropped and/or long queuing delays.
- In times of congestion, if the sources continue to retransmit packets persistently, the congestion may get worse.
- Instead, TCP acts more politely, with each sender retransmitting after longer and longer intervals.

Fast Retransmit:

- One of the problems with timeout-triggered retransmissions is that the timeout period can be relatively long.
- When a segment is lost, this long timeout period forces the sender to delay resending the lost packet, thereby increasing the end-to end delay.
- The sender can often detect packet loss well before the timeout event occurs by noting so called duplicate ACKs.
- A duplicate ACK is an ACK that reacknowledges a segment for which the sender has already received an earlier acknowledgment.
- When a TCP receiver receives a segment with a sequence number that is larger than the next, expected, in-order sequence number, it detects a gap in the data stream—that is, a missing segment.
- This gap could be the result of lost or reordered segments within the network.
- **Table :**

Event	TCP Receiver Action
Arrival of in-order segment with expected sequence number. All data up to expected sequence number already acknowledged.	Delayed ACK. Wait up to 500 msec for arrival of another in-order segment. If next in-order segment does not arrive in this interval, send an ACK.
Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK transmission.	Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment with higher-than-expected sequence number. Gap detected.	Immediately send duplicate ACK, indicating sequence number of next expected byte (which is the lower end of the gap).
Arrival of segment that partially or completely fills in gap in received data.	Immediately send ACK, provided that segment starts at the lower end of gap.

Table 3.2 ♦ TCP ACK Generation Recommendation [RFC 5681]

- Since TCP does not use negative acknowledgments, the receiver cannot send an explicit negative acknowledgment back to the sender.
- Instead, it simply reacknowledges (that is, generates a duplicate ACK for) the last in-order byte of data it has received.
- Because a sender often sends a large number of segments back to back, if one segment is lost, there will likely be many back-to-back duplicate ACKs.
- If the TCP sender receives **three duplicate** ACKs for the same data, it takes this as an indication that the segment following the segment that has been ACKed three times has been lost.

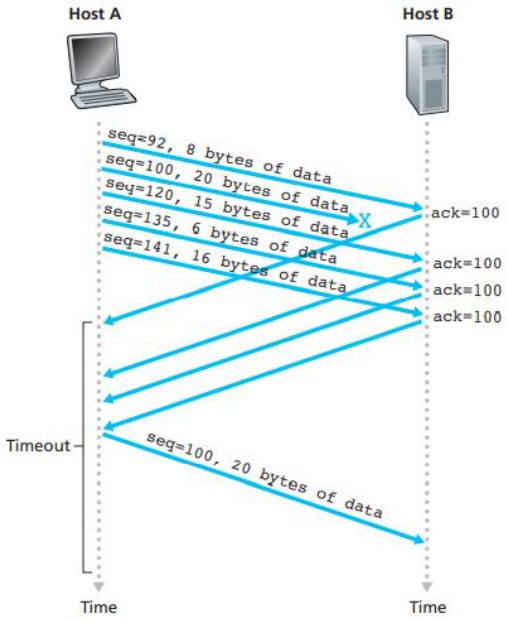


Figure 3.37 ♦ Fast retransmit: retransmitting the missing segment before the segment's timer expires

- In the case that three duplicate ACKs are received, the TCP sender performs a fast retransmit , retransmitting the missing segment before that segment's timer expires.
- This is shown in Figure above, where the second segment is lost, then retransmitted before its timer expires.
- For TCP with fast retransmit, the following code snippet replaces the ACK received event in Figure 3.37:

```
event: ACK received, with ACK field value of y
if (y > SendBase) {
SendBase=y
if (there are currently any not yet
acknowledged segments)
start timer
}
else { /* a duplicate ACK for already ACKed segment */
increment number of duplicate ACKs
received for y
if (number of duplicate ACKS received
for y==3)
/* TCP fast retransmit */
resend segment with sequence number y
}
break;
```

Go-Back-N or Selective Repeat:

- TCP sender need only maintain the smallest sequence number of a transmitted but unacknowledged byte (SendBase) and the sequence number of the next byte to be sent (NextSeqNum).
- TCP looks a lot like a GBN-style protocol. But there are some striking differences between TCP and Go-Back-N.
- Many TCP implementations will buffer correctly received but out-of-order .Suppose that the acknowledgment for packet n < N gets lost, but the remaining N – 1 acknowledgments arrive at the sender before their respective timeouts.
- In this example, GBN would retransmit not only packet n, but also all of the subsequent packets n + 1, n + 2, . . . , N. TCP, on the other hand, would retransmit at most one segment, namely, segment n.
- Moreover, TCP would not even retransmit segment n if the acknowledgment for segment n + 1 arrived before the timeout for segment n. A proposed modification to TCP, the so-called **Selective acknowledgment.**
- TCP receiver acknowledges out-of-order segments selectively rather than just cumulatively acknowledging the last correctly received, inorder segment. When combined with selective retransmission—skipping the retransmission of segments that have already been selectively acknowledged by the receiver.

.5.5 Flow Control:

- Hosts on each side of a TCP connection sets aside a receive buffer for the connection. When the TCP connection receives bytes that are correct and insequence, it places the data in the receive buffer.
- The associated application process will read data from this buffer, but not necessarily at the instant the data arrives.
- The receiving application may be busy with some other task and may not even attempt to read the data until long after it has arrived.
- If the application is relatively slow at reading the data, the sender can very easily overflow the connection's receive buffer by sending too much data too quickly.
- TCP provides a flow-control service to its applications to eliminate the possibility of the sender overflowing the receiver's buffer.
- Flow control is thus a speed-matching service—matching the rate at which the sender is sending against the rate at which the receiving application is reading.
- TCP sender can also be throttled due to congestion within the IP network; this form of sender control is referred to as congestion control.
- TCP provides flow control by having the sender maintain a variable called the **receive window.**

- Informally, the receive window is used to give the sender an idea of how much free buffer space is available at the receiver.
- Because TCP is full-duplex, the sender at each side of the connection maintains a distinct receive window.
- Suppose that Host A is sending a large file to Host B over a TCP connection.
- Host B allocates a receive buffer to this connection; denote its size by **RcvBuffer.** From time to time, the application process in Host B reads from the buffer.

Define the following variables:

- **LastByteRead:** the number of the last byte in the data stream read from the buffer by the application process in B.
- **LastByteRcvd:** the number of the last byte in the data stream that has arrived from the network and has been placed in the receive buffer at B Because TCP is not permitted to overflow the allocated buffer, we must have

  LastByteRcvd – LastByteRead <= RcvBuffer

- The receive window, denoted rwnd is set to the amount of spare room in the buffer:

  rwnd = RcvBuffer – [LastByteRcvd – LastByteRead]

- Because the spare room changes with time, rwnd is dynamic. The variable rwnd is illustrated in Figure below.

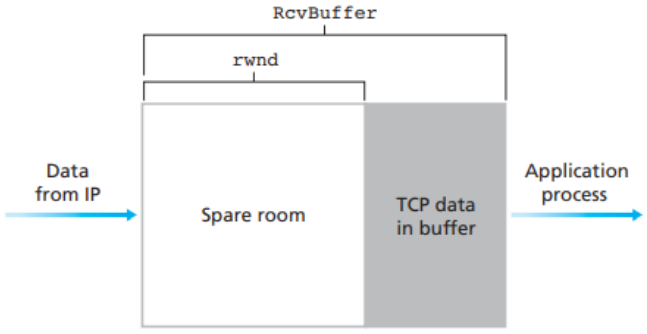


Figure 3.38 ♦ The receive window (rwnd) and the receive buffer (RcvBuffer)

- Host B tells Host A how much spare room it has in the connection buffer by placing its current value of rwnd in the receive window field of every segment it sends to A.
- Initially, Host B sets rwnd = RcvBuffer. Host B must keep track of several connection-specific variables.
- Host A in turn keeps track of **two variables, LastByteSent and Last- ByteAcked.**
- The difference between these two variables, **LastByteSent – LastByteAcked**, is the amount of unacknowledged data that A has sent into the connection.

- By keeping the amount of unacknowledged data less than the value of rwnd, Host A is assured that it is not overflowing the receive buffer at Host B.
- Thus, Host A makes sure throughout the connection's life that

LastByteSent – LastByteAcked < = rwnd

- Problem with this scheme: Suppose Host B's receive buffer becomes full so that **rwnd = 0.**
- After advertising rwnd = 0 to Host A, also suppose that B has nothing to send to A. As the application process at B empties the buffer, TCP does not send new segments with new rwnd values to Host A; indeed, TCP sends a segment to Host A only if it has data to send or if it has an acknowledgment to send.
- Therefore, Host A is never informed that some space has opened up in Host B's receive buffer—Host A is blocked and can transmit no more data!
- **To solve this problem, the TCP** specification requires Host A to continue to send segments with one data byte when B's receive window is zero.
- These segments will be acknowledged by the receiver.
- Eventually the buffer will begin to empty and the acknowledgments will contain a nonzero **rwnd value.**

.5.6 TCP Connection Management:

- Suppose a process running in one host (client) wants to initiate a connection with another process in another host (server).
- The client application process first informs the client TCP that it wants to establish a connection to a process in the server.
- The TCP in the client then proceeds to establish a TCP connection with the TCP in the server in the following manner:

Step 1. The client-side TCP first sends a special TCP segment to the server-side TCP. This special segment contains no application-layer data.

- But one of the flag bits in the segment's header (refer TCP segment Figure), the SYN bit, is set to 1.
- For this reason, this special segment is referred to as a SYN segment.
- In addition, the client randomly chooses an initial sequence number (client_isn) and puts this number in the sequence number field of the initial TCP SYN segment.
- This segment is encapsulated within an IP datagram and sent to the server.

Step 2. Once the IP datagram containing the TCP SYN segment arrives at the server host, the server extracts the TCP SYN segment from the datagram, allocates the TCP buffers and variables to the connection, and sends a connection-granted segment to the client TCP.

- This connection-granted segment also contains no application layer data.
- It contains **three important pieces of information in the segment header :**

1) **The SYN bit is set to 1.**

2) The acknowledgment field of the TCP segment header is set to client_isn+1.

3) Server chooses its own initial sequence number (server_isn) and puts this value in

- The sequence number field of the TCP segment header.
- This connection-granted segment is saying, in effect, "Server received clients SYN packet to start a connection with clients initial sequence number, client_isn.
- Server is agree to establish this connection.
- Servers initial sequence number is server_isn." The connection granted segment is referred to as a SYNACK segment.

Step 3. Upon receiving the SYNACK segment, the client also allocates buffers and

variables to the connection.

- The client host then sends the server yet another segment;
- This last segment acknowledges the server's connection-granted segment (the client does so by putting the value server_isn+1 in the acknowledgment field of the TCP segment header).
- The SYN bit is set to zero, since the connection is established.
- This third stage of the three-way handshake may carry client-to-server data in the segment payload.
- **Once these three steps** have been completed, the client and server hosts can send segments containing data to each other. In each of these future segments, the SYN bit will be set to zero.
- In order to establish the connection, **three packets are sent between the two hosts**, as illustrated in Figure.
- For this reason, this connection establishment procedure is often referred to as a three-way handshake.

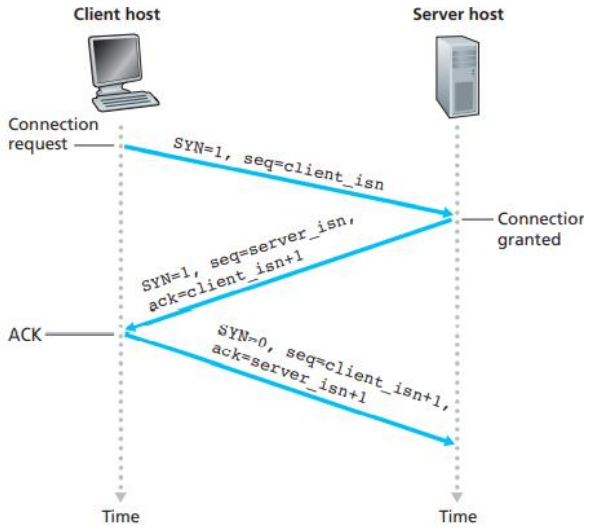


Figure 3.39 ♦ TCP three-way handshake: segment exchange

- Either of the two processes participating in a TCP connection can end the connection. When a connection ends, the "resources" (that is, the buffers and variables) in the hosts are deallocated.
- Example, suppose the client decides to close the connection, as shown in Figure below.
- The client application process issues a close command.
- This causes the client TCP to send a special TCP segment to the server process.

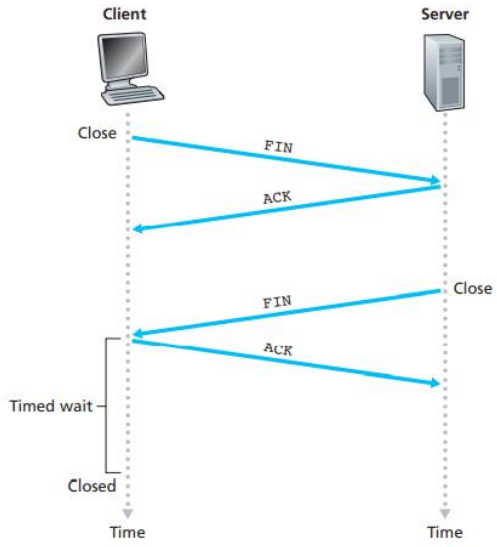


Figure 3.40 ♦ Closing a TCP connection

- This special segment has a flag bit in the segment's header, the FIN bit, set to 1.
- When the server receives this segment, it sends the client an acknowledgment segment in return.
- The server then sends its own shutdown segment, which has the FIN bit set to 1.

- Finally, the client acknowledges the server's shutdown segment.
- At this point, all the resources in the two hosts are now deallocated.
- During the life of a TCP connection, the TCP protocol running in each host makes transitions through various TCP states.

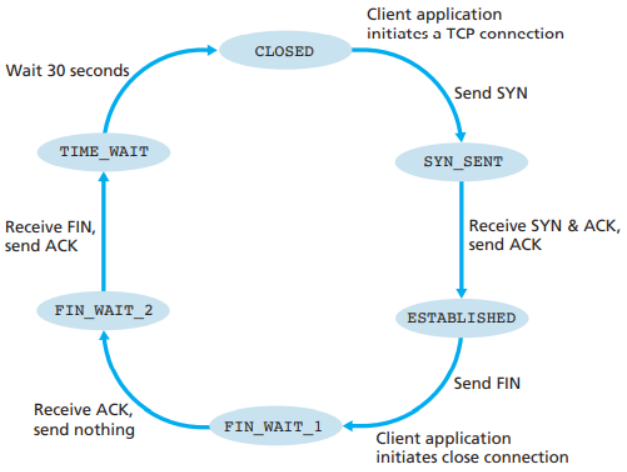


Figure 3.41 ♦ A typical sequence of TCP states visited by a client TCP

- Figure above illustrates a typical sequence of TCP states that are visited by the client TCP.
- The client TCP begins in the CLOSED state. The application on the client side initiates a new TCP connection.
- This causes TCP in the client to send a SYN segment to TCP in the server. After having sent the SYN segment, the client TCP enters the SYN_SENT state. While in the SYN_SENT state, the client TCP waits for a segment from the server TCP that includes an acknowledgment for the client's previous segment and has the SYN bit set to 1.
- Having received such a segment, the client TCP enters the ESTABLISHED state. While in the ESTABLISHED state, the TCP client can send and receive TCP segments containing payload (message) data
- Suppose that the client application decides it wants to close the connection. This causes the client TCP to send a TCP segment with the FIN bit set to 1 and to enter the FIN_WAIT_1 state.
- While in the FIN_WAIT_1 state, the client TCP waits for a TCP segment from the server with an acknowledgment.
- When it receives this segment, the client TCP enters the FIN_WAIT_2 state.
- While in the FIN_WAIT_2 state, the client waits for another segment from the server with the FIN bit set to 1; after receiving this segment, the client TCP acknowledges the server's segment and enters the TIME_WAIT state.
- The TIME_WAIT state lets the TCP client resend the final acknowledgment in case the ACK is lost.

- The time spent in the TIME_WAIT state is implementation-dependent, but typical values are 30 seconds, 1 minute, and 2 minutes.
- After the wait, the connection formally closes and all resources on the client side (including port numbers) are released.

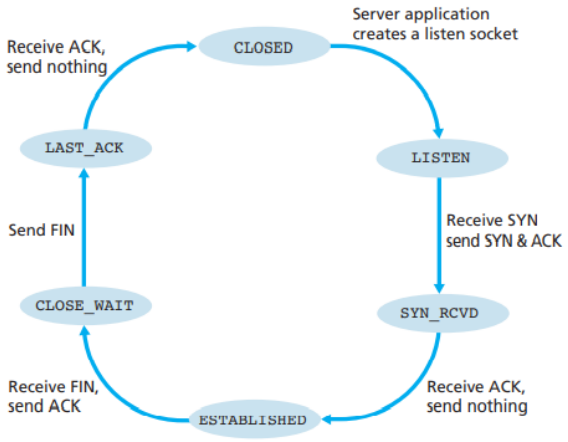


Figure 3.42 ♦ A typical sequence of TCP states visited by a server-side TCP

- Figure above illustrates the series of states typically visited by the server-side TCP, assuming the client begins connection teardown.
- If a host receives a TCP segment whose port numbers or source IP address do not match with any of the ongoing sockets in the host.
- For example, suppose a host receives a TCP SYN packet with destination port 80, but the host is not accepting connections on port 80.
- Then the host will send a special reset segment to the source. This TCP segment has the RST flag bit set to 1.
- Thus, when a host sends a reset segment, Source sends TCP SYN segment with destination port 6789 to target host.
- **There are three possible outcomes:**

1. **The source host receives a TCP SYNACK** segment from the target host. Since this means that an application is running with TCP port 6789 on the target post, returns "open."
2. The source host receives a TCP RST segment from the target host.
    a. This means that the SYN segment reached the target host, but the target host is not running an application with TCP port 6789.
    b. But the attacker at least knows that the segments destined to the host at port 6789 are not blocked by any firewall on the path between source and target hosts.
3. The source receives nothing. SYN segment was blocked by an intervening firewall and never reached the target host.

.6 Principles of Congestion Control

- Cause of network congestion is too many sources attempting to send data at too high a rate.
- To treat the cause of network congestion, mechanisms are needed to throttle senders in the face of network congestion.

The Causes and the Costs of Congestion

Scenario 1: Two Senders, a Router with Infinite Buffers

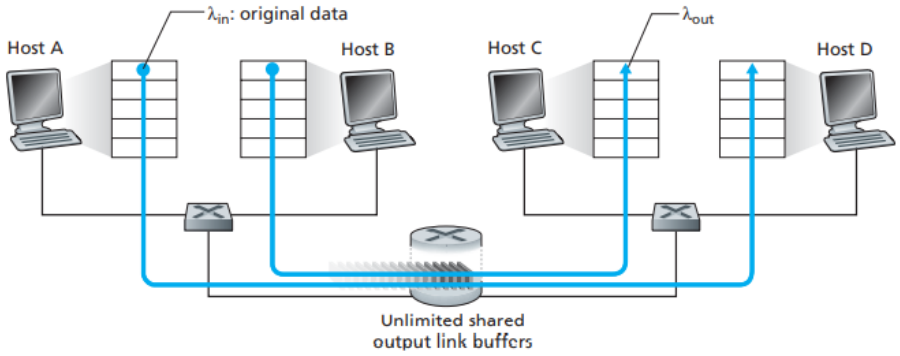


Figure 3.43 ♦ Congestion scenario 1: Two connections sharing a single hop with infinite buffers

- The simplest congestion scenario possible: Two hosts (A and B) each have a connection that shares a single hop between source and destination, as shown in Figure above.
- Assume that the application in Host A is sending data into the connection at an average rate of λ in bytes/sec.
- These data are original in the sense that each unit of data is sent into the socket only once.
- The underlying transport-level protocol is a simple one.
- Data is encapsulated and sent; no error recovery (for example, retransmission), flow control, or congestion control is performed.
- Ignoring the additional overhead due to adding transport- and lower-layer header information, the rate at which Host A offers traffic to the router in this first scenario is thus λ in bytes/sec.
- Host B operates in a similar manner, and we assume for simplicity that it too is sending at a rate of λ in bytes/sec.
- Packets from Hosts A and B pass through a router and over a shared outgoing link of capacity R.
- The router has buffers that allow it to store incoming packets when the packet-arrival rate exceeds the outgoing link's capacity.

In this first scenario, we assume that the router has an infinite amount of buffer space.

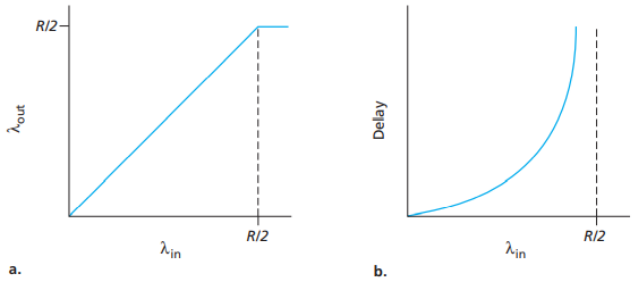


Figure 3.44 ♦ Congestion scenario 1: Throughput and delay as a function of host sending rate

- Figure above plots the performance of Host A's connection under this first scenario.
- The left graph plots the per-connection throughput (number of bytes per second at the receiver) as a function of the connection-sending rate.
- For a sending rate between **0 and R/2,** the throughput at the receiver equals the sender's sending rate—everything sent by the sender is received at the receiver with a finite delay.
- When the sending rate is above R/2, however, the throughput is only R/2.
- This upper limit on throughput is a consequence of the sharing of link capacity between two connections.
- The link simply cannot deliver packets to a receiver at a steady-state rate that exceeds **R/2.**
- Even if Hosts A and B set their sending rates high, they will each never see a throughput higher than R/2.
- Achieving a per-connection throughput of R/2 is good because the link is fully utilized in delivering packets to their destinations.
- The right-hand graph in Figure a above, shows the consequence of operating near link capacity.
- As the sending rate approaches R/2 (from the left), the average delay becomes larger and larger.
- When the sending rate exceeds R/2, the average number of queued packets in the router is unbounded and the average delay between source and destination becomes infinite (assuming that the connections operate at these sending rates for an infinite period of time and there is an infinite amount of buffering available).
- Thus, while operating at an aggregate throughput of near R may be ideal from a throughput standpoint, it is far from ideal from a delay standpoint.
- Thus, Large queuing delays are experienced as the packet arrival rate nears the link capacity.

Scenario 2: Two Senders and a Router with Finite Buffers

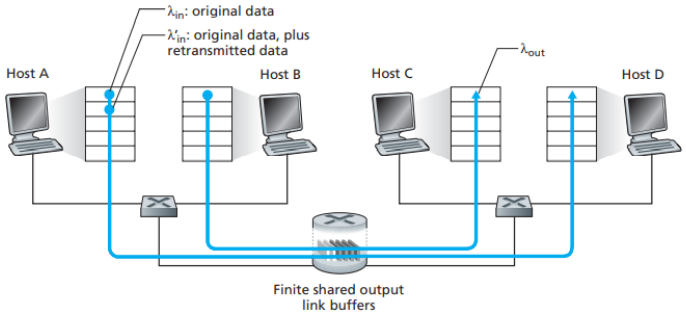


Figure 3.45 ♦ Scenario 2: Two hosts (with retransmissions) and a router with finite buffers

- **Scenario 1 is modified in the following two ways:(Figure).**

1. The amount of router buffering is assumed to be finite.
    a. A consequence of this assumption is that packets will be dropped when arriving to an already full buffer.
2. Assume that each connection is reliable.
    a. If a packet containing a transport-level segment is dropped at the router, the sender will eventually retransmit it.
    b. The rate at which the application sends original data into the socket by in bytes/sec.
    c. The rate at which the transport layer sends segments (containing original data and retransmitted data) into the network will be denoted **λin bytes/sec**.
    d. λin is referred to as the offered load to the network.

- The performance realized under scenario 2 will now depend strongly on how retransmission is performed.

1.Consider the unrealistic case that Host A is able to somehow determine whether or not a buffer is free in the router and thus sends a packet only when a buffer is free.

- In this case, no loss would occur, λ in would be equal to λ in, and the throughput of the connection would be equal to λ in.

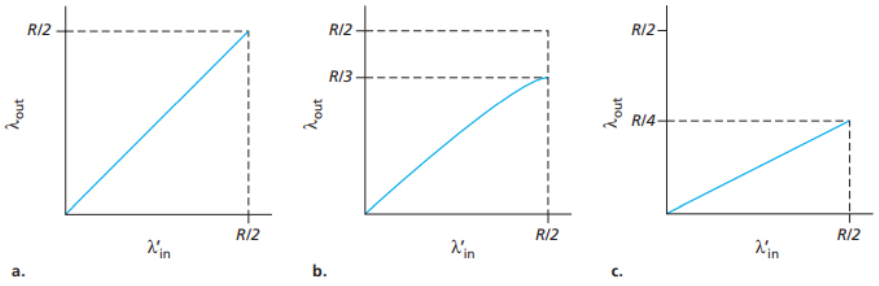


Figure 3.46 ♦ Scenario 2 performance with finite buffers

This case is shown in Figure (a).

- From a throughput standpoint, performance is ideal—everything that is sent is received.
- The average host sending rate cannot exceed R/2 under this scenario, since packet loss is assumed never to occur.

2.Consider a realistic case in which the sender retransmits only when a packet is known for certain to be lost.

- In this case, the performance is shown in Figure (b).
- Consider the case that the offered load, λ in (the rate of original data transmission plus retransmissions), equals R/2. According to Figure (b), at this value of the offered load, the rate at which data are delivered to the receiver application is R/3.
- Thus, out of the **0.5R** units of data transmitted, **0.333R** bytes/sec (on average) are original data and 0.166R bytes/sec (on average) are retransmitted data.
- The sender must perform retransmissions in order to compensate for dropped (lost) packets due to buffer overflow.
- Finally, consider the case that the sender may time out prematurely and retransmit a packet that has been delayed in the queue but not yet lost.
- In this case, both the original data packet and the retransmission may reach the receiver.
- The receiver needs but one copy of this packet and will discard the retransmission.
- In this case, the work done by the router in forwarding the retransmitted copy of the original packet was wasted, as the receiver will have already received the original copy of this packet.
- The router would have better used the link transmission capacity to send a different packet instead.
- The unneeded retransmissions by the sender in the face of large delays may cause a router to use its link bandwidth to forward unneeded copies of a packet.
- **Figure (c)** shows the throughput versus offered load when each packet is assumed to be forwarded (on average) twice by the router. Since each packet is forwarded twice, the throughput will have an asymptotic value of R/4 as the offered load approaches R/2.

Scenario 3: Four Senders, Routers with Finite Buffers, and Multihop Paths

- In final congestion scenario, four hosts transmit packets, each over overlapping two-hop paths, as shown in Figure below.

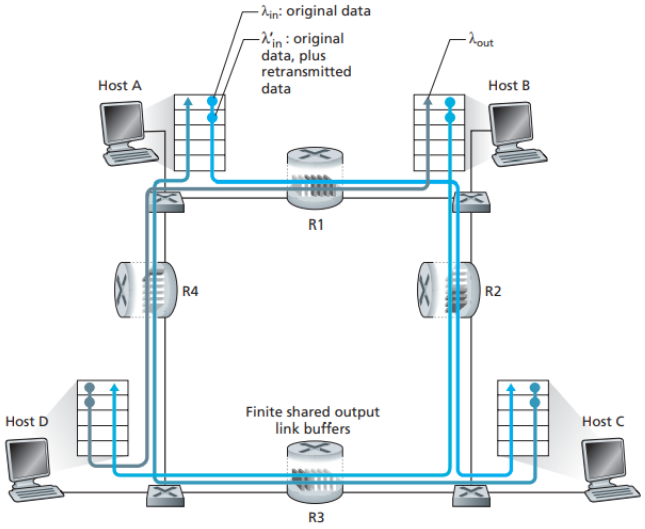


Figure 3.47 ♦ Four senders, routers with finite buffers, and multihop paths

- Assume that each host uses a timeout/retransmission mechanism to implement a reliable data transfer service, that all hosts have the same value of λin, and that all router links have capacity R bytes/sec.
- Consider the connection from Host A to Host C, passing through routers R1 and R2. The A–C connection shares router R1 with the D–B connection and shares router R2 with the B–D connection.
- For extremely small values of λin, buffer overflows are rare (as in congestion scenarios 1 and 2), and the throughput approximately equals the offered load.
- For slightly larger values of λin, the corresponding throughput is also larger, since more original data is being transmitted into the network and delivered to the destination, and overflows are still rare.
- Thus, for small values of λin, an increase in λin results in an increase in λout. Having considered the case of extremely low traffic, consider the case that λin (and hence λin) is extremely large.
- Consider **router R2.**
- The A–C traffic arriving to router R2 (which arrives at R2 after being forwarded from R1) can have an arrival rate at R2 that is at most R, the capacity of the link from R1 to R2, regardless of the value of λin. If λin is extremely large for all connections (including the B–D connection), then the arrival rate of B–D traffic at R2 can be much larger than that of the A–C traffic.
- Because the A–C and B–D traffic must compete at router R2 for the limited amount of buffer space, the amount of A–C traffic that successfully gets through R2 (that is, is not lost due to buffer overflow) becomes smaller and smaller as the offered load from B–D gets larger and larger.
- In the limit, as the offered load approaches infinity, an empty buffer at R2 is immediately filled by a B–D packet, and the throughput of the A–C connection at R2 goes to zero.

- This, in turn, implies that the A–C end-to-end throughput goes to zero in the limit of heavy traffic.

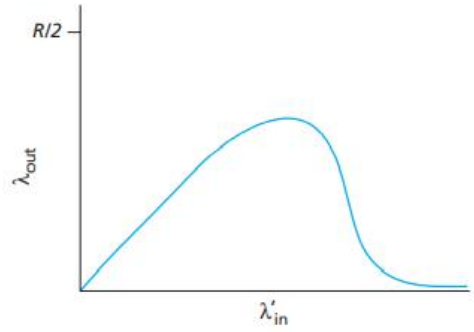


Figure 3.48 ♦ Scenario 3 performance with finite buffers and multihop paths

- The offered load versus throughput trade off shown in Figure 3.48.
- The reason for the eventual decrease in throughput with increasing offered load is evident when one considers the amount of wasted work done by the network.
- In the high-traffic scenario, whenever a packet is dropped at a second-hop router, the work done by the first-hop router in forwarding a packet to the second-hop router ends up being "wasted."
- The transmission capacity used at the first router to forward the packet to the second router could have been used to transmit a different packet.
- **Example,** when selecting a packet for transmission, it might be better for a router to give priority to packets that have already traversed some number of upstream routers.
- When a packet is dropped along a path, the transmission capacity that was used at each of the upstream links to forward that packet to the point at which it is dropped ends up having been wasted.

3.6.2 Approaches to Congestion Control:

The two broad approaches to congestion control that are taken in practice and discuss specific network architectures and congestion-control protocols embodying these approaches.

1. **End-to-end congestion control :** In an end-to-end approach to congestion control, the network layer provides **no explicit support** to the transport layer for congestion control purposes.
    a. Even the presence of congestion in the network must be inferred by the end systems based only on observed network behavior (for example, packetloss and delay).

b. Segment loss is taken as an indication of network-congestion and the window-size is decreased accordingly.

2. **Network-assisted congestion control**. With network-assisted congestion control, network layer components (that is, routers) provide **explicit feedback** to the sender regarding the congestion state in the network.
   a. This feedback may be as simple as **a single bit indicating congestion at a link**.
   b. Congestion information is fed back from the network to the sender in one of **two ways:**
      i) **Direct feedback** may be sent from a network-router to the sender (Figure 3.49).
         - This form of notification typically takes the form of a choke packet.
      ii) **A router marks** a field in a packet flowing from sender to receiver to indicate congestion.
         - Upon receipt of a marked packet, the receiver then notifies the sender of the congestion indication.
         - This form of notification takes at least a full round-trip time.

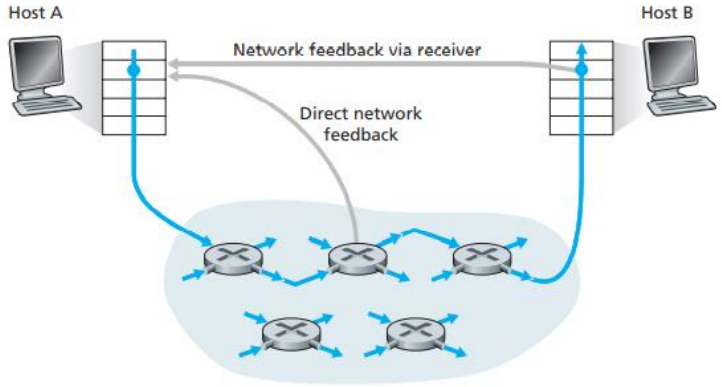


Figure 3.49 ♦ Two feedback pathways for network-indicated congestion information

- This approach is used in ATM available bit-rate (ABR) congestion control.
- More sophisticated network feedback is also possible.
- For example, one form of ATM ABR congestion control allows a router to inform the sender explicitly of the transmission rate it (the router) can support on an outgoing link.
- For network-assisted congestion control, congestion information is typically fed back from the network to the sender in one of two ways, as shown in **Figure below.**

3.6.3 Network Assisted Congestion Control Example:

ATM ABR Congestion Control

- ATM (Asynchronous Transfer Mode) protocol uses network-assisted approach for congestion-control.
- ABR (Available Bit Rate) has been designed as an elastic data-transfer-service.
  i) When the network is underloaded, ABR has to take advantage of the spare available bandwidth.
  ii) When the network is congested, ABR should reduce its transmission-rate.

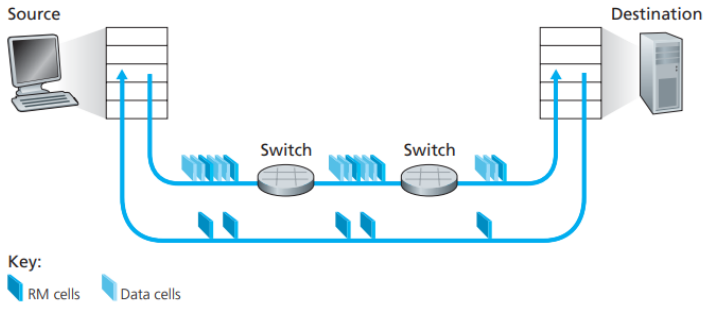


Figure 3.50 ♦ Congestion-control framework for ATM ABR service

- Figure 2.41 shows the framework for ATM ABR congestion-control.
- Data-cells are transmitted from a source to a destination through a series of intermediate switches.
- RM-cells are placed between the data-cells. (RM □ Resource Management).
- The RM-cells are used to send congestion-related information to the hosts & switches.
- When an RM-cell arrives at a destination, the cell will be sent back to the sender
- Thus, RM-cells can be used to provide both
  → direct network feedback and
  → network feedback via the receiver.

Three Methods to indicate Congestion

- ATM ABR congestion-control is a rate-based approach.
- ABR provides 3 mechanisms for indicating congestion-related information:

1) EFCI Bit

- Each data-cell contains an EFCI bit. (EFCI □ Explicit forward congestion indication)
- A congested-switch sets the EFCI bit to 1 to signal congestion to the destination.
- The destination must check the EFCI bit in all received data-cells.
- If the most recently received data-cell has the EFCI bit set to 1, then the destination
  → sets the CI bit to 1 in the RM-cell (CI □ congestion indication)
  → sends the RM-cell back to the sender.
- Thus, a sender can be notified about congestion at a network switch.

2) CI and NI Bits

- The rate of RM-cell interspersion is a tunable parameter.
- The default value is one RM-cell every 32 data-cells. (NI □ No Increase)
- The RM-cells have a CI bit and a NI bit that can be set by a congested-switch.
- A switch
  → sets the NI bit to 1 in a RM-cell under mild congestion and
  → sets the CI bit to 1 under severe congestion conditions.

3) ER Setting

- Each RM-cell also contains an ER field. (ER explicit rate)
- A congested-switch may lower the value contained in the ER field in a passing RM-cell.
- In this manner, ER field will be set to minimum supportable rate of all switches on the path.

.7 TCP Congestion Control:

- TCP has congestion-control mechanism.
- TCP uses end-to-end congestion-control rather than network-assisted congestion-control
- Here is how it works:
  - Each sender limits the rate at which it sends traffic into its connection as a function of perceived congestion.
    - i) If sender perceives that there is little congestion, then sender increases its data-rate.
    - ii) If sender perceives that there is congestion, then sender reduces its data-rate.
- This approach raises three questions:
  1) How does a sender limit the rate at which it sends traffic into its connection?
  2) How does a sender perceive that there is congestion on the path?
  3) What algorithm should the sender use to change its data-rate?
- The sender keeps track of an additional variable called the congestion-window (cwnd).
- The congestion-window imposes a constraint on the data-rate of a sender.
- The amount of unacknowledged-data at a sender will not exceed minimum of (cwnd & rwnd), that is:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{cwnd}, \text{rwnd}\}$$

- The sender's data-rate is roughly cwnd/RTT bytes/sec.
- Explanation of Loss event:
  - A "loss event" at a sender is defined as the occurrence of either
    → timeout or
    → receipt of 3 duplicate ACKs from the receiver.
  - Due to excessive congestion, the router-buffer along the path overflows. This causes a datagram to be dropped.
  - The dropped datagram, in turn, results in a loss event at the sender.
  - The sender considers the loss event as an indication of congestion on the path.
- How congestion is detected?
  - Consider the network is congestion-free.
  - Acknowledgments for previously unacknowledged segments will be received at the sender.
  - TCP
    → will take the arrival of these acknowledgments as an indication that all is well and
    → will use acknowledgments to increase the window-size (& hence data-rate).
  - TCP is said to be self-clocking because
    → acknowledgments are used to trigger the increase in window-size

- **Congestion-control algorithm has 3 major components:**
  1) Slow start
  2) Congestion avoidance and
  3) Fast recovery.

.7.1 Slow Start

- When a TCP connection begins, the value of cwnd is initialized to 1 MSS.
- TCP doubles the number of packets sent every RTT on successful transmission.
- Here is how it works:
  - As shown in **Figure 3.51,** the TCP
    - → sends the first-segment into the network and
    - → waits for an acknowledgment.
  - When an acknowledgment arrives, the sender
    - → increases the congestion-window by one MSS and
    - → sends out 2 segments.
  - When two acknowledgments arrive, the sender
    - → increases the congestion-window by one MSS and
    - → sends out 4 segments.
  - This process results in a doubling of the sending-rate every RTT.
- Thus, the TCP data-rate starts slow but grows exponentially during the slow start phase.

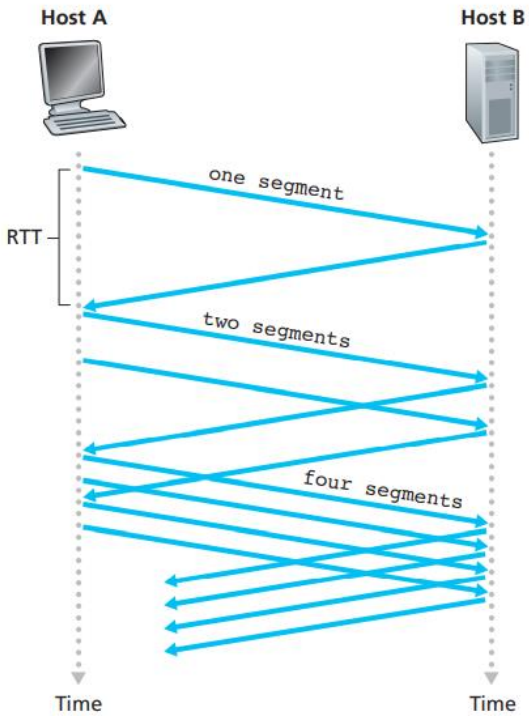


Figure 3.51 ♦ TCP slow start

- When should the exponential growth end?
  - Slow start provides several answers to this question.
    1) If there is a loss event, the sender
       → sets the value of cwnd to 1 and
       → begins the slow start process again. (ssthresh □ "slow start threshold")
       → sets the value of ssthresh to cwnd/2.
    2) When the value of cwnd equals ssthresh, TCP enters the congestion avoidance state.
    3) When three duplicate ACKs are detected, TCP
       → performs a fast retransmit and
       → enters the fast recovery state.
- TCP's behavior in slow start is summarized in FSM description in **Figure 3.52.**

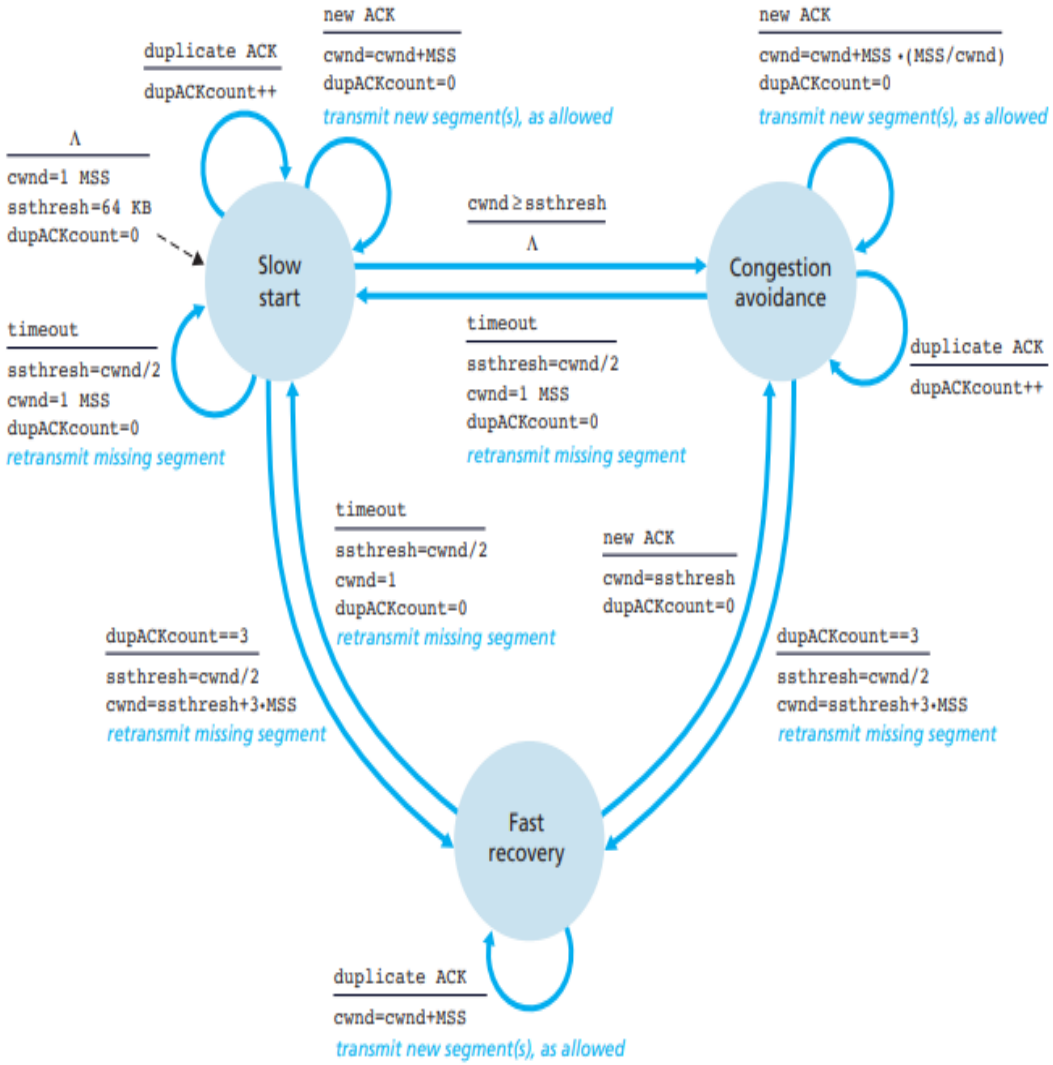


Figure 3.52 ♦ FSM description of TCP congestion control

.7.2 Congestion Avoidance

- On entry to congestion-avoidance state, the value of cwnd is approximately half its previous value.
- Thus, the value of cwnd is increased by a single MSS every RTT.
- The sender must increases cwnd by MSS bytes (MSS/cwnd) whenever a new acknowledgment arrives
- When should linear increase (of 1 MSS per RTT) end?
  1) When a timeout occurs.
     - When the loss event occurred,
       → value of cwnd is set to 1 MSS and
       → value of ssthresh is set to half the value of cwnd.
  2) When triple duplicate ACK occurs.
     - When the triple duplicate ACKs were received,
       → value of cwnd is halved.
       → value of ssthresh is set to half the value of cwnd.

.7.3 Fast Recovery

- The value of cwnd is increased by 1 MSS for every duplicate ACK received.
- When an ACK arrives for the missing segment, the congestion-avoidance state is entered.
- If a timeout event occurs, fast recovery transitions to the slow-start state.
- When the loss event occurred
  → value of cwnd is set to 1 MSS, and
  → value of ssthresh is set to half the value of cwnd.
- There are 2 versions of TCP:

1) TCP Tahoe

- An early version of TCP was known as TCP Tahoe.
- TCP Tahoe
  → cut the congestion-window to 1 MSS and
  → entered the slow-start phase after either
  i) timeout-indicated or
  ii) triple-duplicate-ACK-indicated loss event.

2) TCP Reno

- The newer version of TCP is known as TCP Reno.
- TCP Reno incorporated fast recovery.
- Figure 3.53 illustrates the evolution of TCP's congestion-window for both Reno and Tahoe.

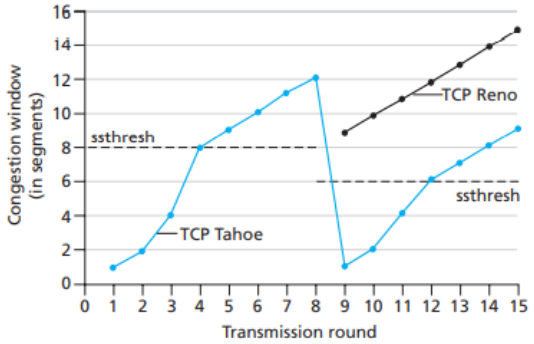


Figure 3.53 ♦ Evolution of TCP's congestion window (Tahoe and Reno)

3.7.4 TCP Congestion Control Retrospective

- TCP's congestion-control consists of (AIMD ->additive increase, multiplicative decrease)
  → Increasing linearly (additive) value of cwnd by 1 MSS per RTT and
  → Halving (multiplicative decrease) value of cwnd on a triple duplicate-ACK event.
- For this reason, TCP congestion-control is often referred to as an AIMD.
- AIMD congestion-control gives rise to the "saw tooth" behavior shown in Figure 2.45.
- TCP
  → increases linearly the congestion-window-size until a triple duplicate-ACK event occurs and
  → decreases then the congestion-window-size by a factor of 2

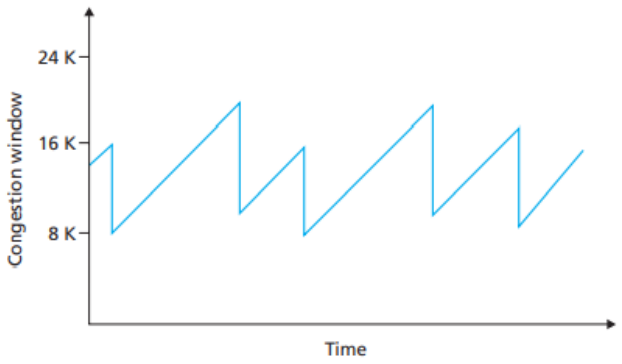


Figure 3.54 ♦ Additive-increase, multiplicative-decrease congestion control

3.7.5 TCP Fairness

- Congestion-control mechanism is fair if each connection gets equal share of the link-bandwidth.
- As shown in Figure 3.55, consider 2 TCP connections sharing a single link with transmission-rate R.
- Assume the two connections have the same MSS and RTT.

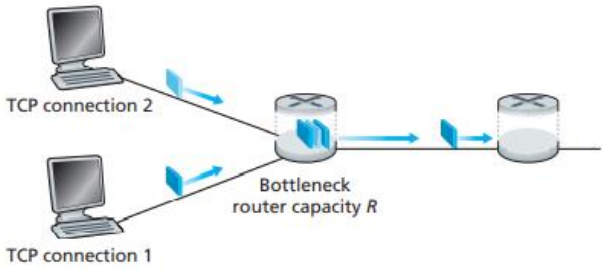


Figure 3.55 ♦ Two TCP connections sharing a single bottleneck link

- Figure 3.56 plots the throughput realized by the two TCP connections.
  - If TCP shares the link-bandwidth equally b/w the 2 connections, then the throughput falls along the 45-degree arrow starting from the origin.

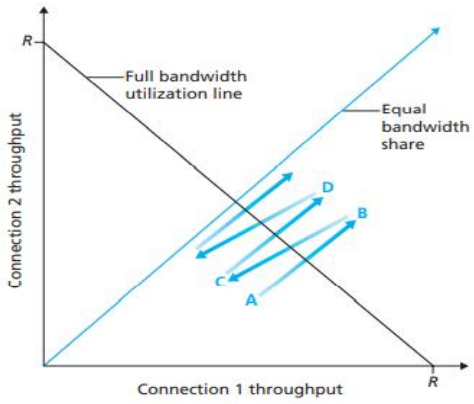


Figure 3.56 ♦ Throughput realized by TCP connections 1 and 2

Fairness and UDP

- Many multimedia-applications (such as Internet phone) often do not run over TCP.
- Instead, these applications prefer to run over UDP. This is because
  - → applications can pump their audio into the network at a constant rate and
  - → occasionally lose packets.

Fairness and Parallel TCP Connections

- Web browsers use multiple parallel-connections to transfer the multiple objects within a Web page.
- Thus, the application gets a larger fraction of the bandwidth in a congested link.
- '.' Web-traffic is so pervasive in the Internet; multiple parallel-connections are common nowadays.

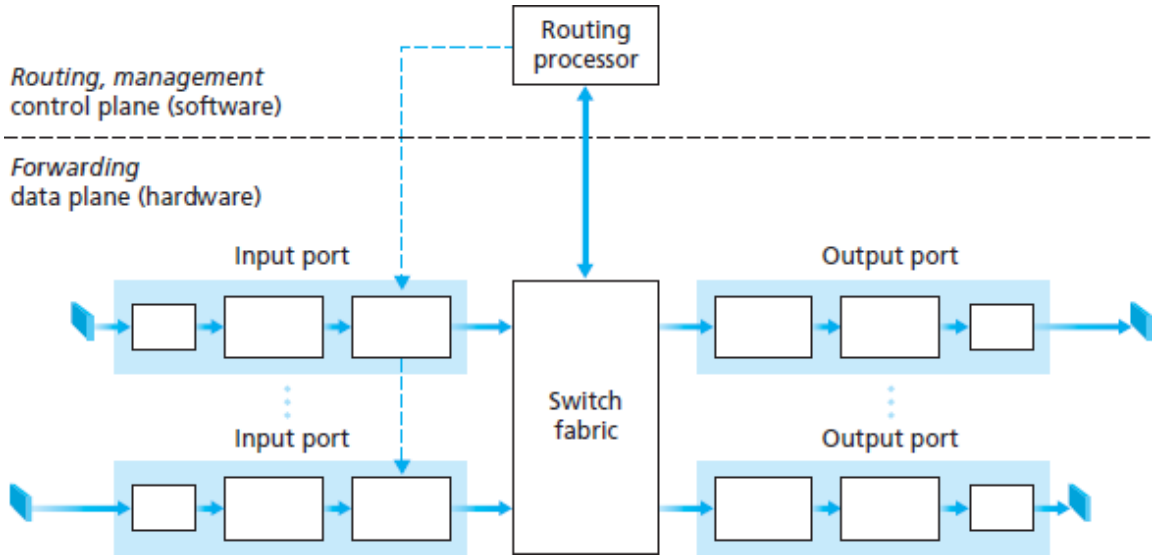
MODULE-WISE QUESTIONS

1) With a diagram, explain multiplexing and demultiplexing. (6*)
2) Explain the significance of source and destination-port-no in a segment. (4*)
3) With a diagram, explain connectionless multiplexing and demultiplexing. (4)
4) With a diagram, explain connection oriented multiplexing and demultiplexing. (4)
5) Briefly explain UDP & its services. (6*)
6) With general format, explain various field of UDP segment. Explain how checksum is calculated (8*)
7) With a diagram, explain the working of rdt1.0. (6)
8) With a diagram, explain the working of rdt2.0. (6*)
9) With a diagram, explain the working of rdt2.1. (6)
10) With a diagram, explain the working of rdt3.0. (6*)
11) With a diagram, explain the working of Go-Back-N. (6*)
12) With a diagram, explain the working of selective repeat. (6*)
13) Explain the following terms: (8)
    i) Sequence-number
    ii) Acknowledgment
    iii) Negative acknowledgment
14) Window, pipelining briefly explain TCP & its services. (6*)
15) With general format, explain various field of TCP segment. (6*)
16) With a diagram, explain the significance of sequence and acknowledgment numbers. (4*)
17) With a diagram, explain the reliable data transfer with few interesting scenarios. (8)
18) With a diagram, explain fast retransmit in TCP. (6*)
19) With a diagram, explain flow Control in TCP. (6)
20) With a diagram, explain connection management in TCP. (8*)
21) With a diagram, explain the causes of congestion with few scenarios. (8)
22) Briefly explain approaches to congestion control. (6*)
23) With a diagram, explain ATM ABR congestion control. (8)
24) With a diagram, explain slow start in TCP. (6*)
25) With a diagram, explain fast recovery in TCP. (6*)

Module – 3

The Network layer: What's Inside a Router?: Input Processing, Switching, Output Processing, Where Does Queuing Occur? Routing control plane, IPv6,A Brief foray into IP Security, Routing Algorithms: The Link-State (LS) Routing Algorithm, The Distance-Vector (DV) Routing Algorithm, Hierarchical Routing, Routing in the Internet, Intra-AS Routing in the Internet: RIP, Intra-AS Routing in the Internet: OSPF, Inter/AS Routing: BGP, Broadcast and Multicast Routing: Broadcast Routing Algorithms and Multicast.

What's inside a router ?



A high-level view of a generic router architecture is shown in Figure above. 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 above.

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.

The lookup function is also performed at the input port; this will occur in the rightmost box of the input port. Here the forwarding table is consulted to determine the router output port to which an arriving packet will be forwarded via the switching fabric.

• *Switching fabric.* The switching fabric connects the router's input ports to its output ports. This switching fabric is completely contained within the 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.

• *Routing processor.* The routing processor executes the routing protocols, maintains routing tables and attached link state information and computes the forwarding table for the router.

It also performs the network management functions 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, as shown in Figure.

These forwarding functions are collectively referred to as the **router forwarding plane**.

Example: Consider 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, the datagram-processing pipeline must operate *N* times faster.

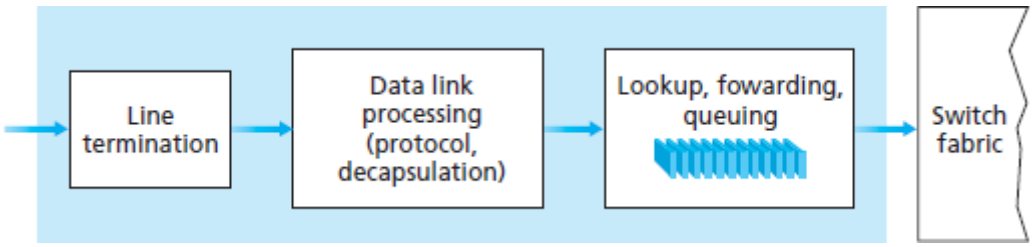
Input Processing :

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 below.

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.



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.

Important action in input port processing are :

(1) physical- and link-layer processing must occur ;

(2) the packet's version number, checksum and time-to-live field

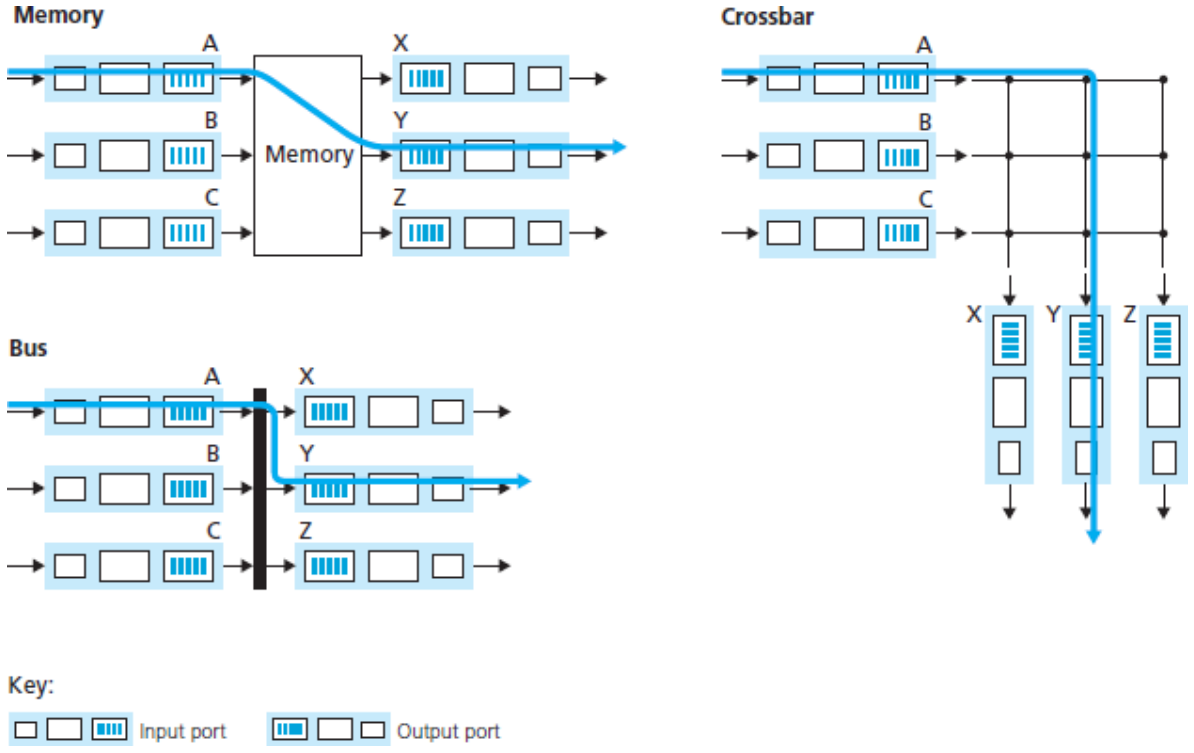
(3) counters used for network management must be updated.

4.3.2 Switching

The switching fabric is at the very heart of a router, as it is through fabric that the packets are actually switched (that is, forwarded) from an input port to an output port.

Switching can be accomplished in following ways, as shown in Figure below:

• ***Switching via memory***.



Switching between input and output ports is done under direct control of the CPU (routing processor). An input port with an arriving packet first signals the routing processor via an interrupt.

The packet is then copied from the input port into processor memory. The routing processor then extracts the destination address from the header, looking up the appropriate output port in the forwarding table and copies the packet to the output port's buffers.

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.

• ***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 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.

Since every packet must cross the single bus, the switching speed of the router is limited to the bus speed;

• ***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 2*N* buses that connect *N* input ports to *N* output ports, as shown in Figure above.

Each vertical bus intersects each horizontal bus at a cross point, which can be opened or closed at any time by the switch fabric controller.

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.

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.

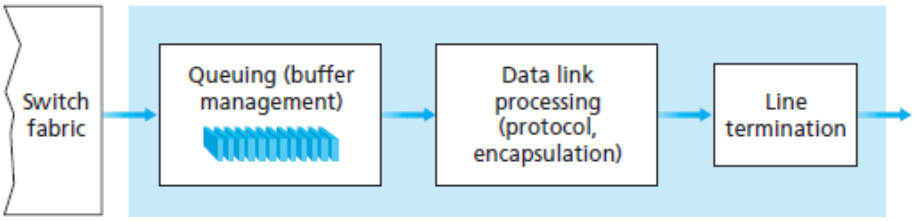
Thus, crossbar networks are capable of forwarding multiple packets in parallel.

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.

4.3.3 Output Processing

Output port processing, shown in Figure below, 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?

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.

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.

At the 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.

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.

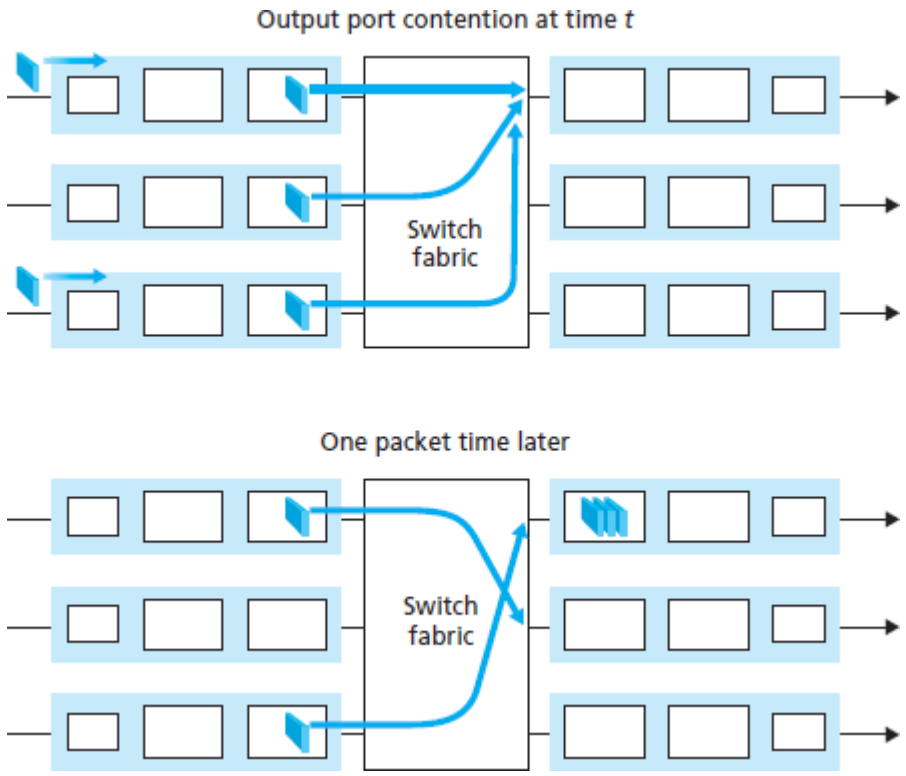
Define the switching fabric transfer rate *Rswitch* as the rate at which packets can be moved from input port to output port. If *Rswitch* is *N* times faster than *Rline*, then only negligible queuing will occur at the input ports.

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.

Suppose *Rswitch* is *N* times faster than *Rline*. Packets arriving at each of the *N* input ports are destined to the same output port.

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 transmit just one of the *N* packets that had just previously been queued.

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 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. 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)*.

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.

When there are a large number of TCP flows (*N*) passing through a link, the amount of buffering needed is $B = RTT \_ C/\sqrt{N}$

With a large number of flows typically passing through large backbone router links, the value of *N* can be large, with the decrease in needed buffer size becoming quite significant.

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 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.

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.

One of the widely implemented Active Queue Management 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*.

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.

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.

(3) packets are moved from a given input queue to their desired output queue in an FCFS manner.

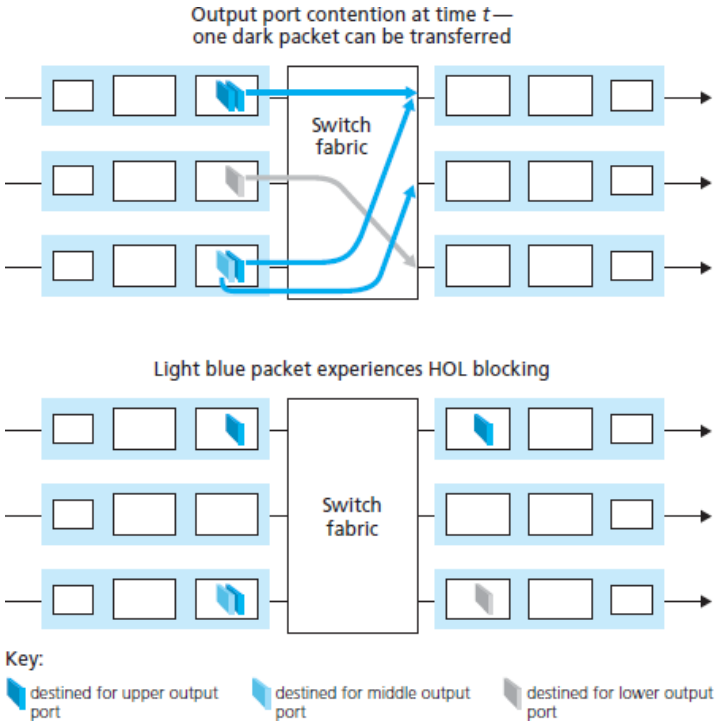
Multiple packets can be transferred in parallel, as long as their output ports are different. However, if two packets at the front of two input queues are destined for the same output queue, then one of the packets will be blocked and must wait at the input queue—the switching fabric can transfer only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

Figure below 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.

Due to HOL blocking, the input queue will grow to unbounded length under certain assumptions as soon as the packet arrival rate on the input links reaches only 58 percent of their capacity.



4.3.5 The Routing Control Plane

Router control plane architectures in which part of the control plane is implemented in the routers (e.g., local measurement/reporting of link state, forwarding table installation and maintenance) along with the data plane and part of the control plane can be implemented externally to the router (e.g., in a centralized server, which could perform route calculation).

A well-defined API dictates how these two parts interact and communicate with each other. By separating the software control plane from the hardware data plane (with a minimal router-resident control plane) can simplify routing by replacing distributed routing calculation with centralized routing calculation, and enable network innovation by allowing different customized control planes to operate over fast hardware data planes.

4.4 The Internet Protocol (IP): Forwarding and Addressing in the Internet

Internet addressing and forwarding are important components of the Internet Protocol (IP). There are two versions of IP in use today i.e IPv4 & IPv6. We'll first examine the widely deployed IP protocol version 4, which is usually referred to simply as IPv4.

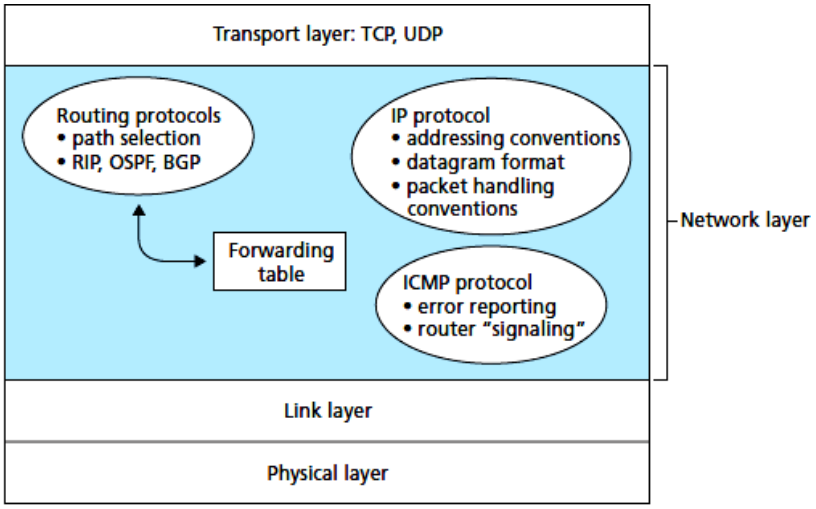


Figure 4.12 ♦ A look inside the Internet's network layer

The components that make up the Internet's network layer
As shown in Figure 4.12, the Internet's network layer has three major components.

- The first component is the IP protocol, is the principal communications **protocol** in the Internet **protocol** suite for relaying datagrams across network boundaries..
- The second major component is the routing component, which determines the path a datagram follows from source to destination.Routing protocols compute the forwarding tables that are used to forward packets through the network.
- The final component of the network layer is a facility to report errors in datagrams and respond to requests for certain network-layer information. Example for the Internet's network-layer error- and information-reporting protocol, the Internet Control Message Protocol (ICMP)

4.4.1 Datagram Format

A network-layer packet is referred to as a *datagram*. This provides an overview of the syntax and semantics of the IPv4 datagram.

The key fields in the IPv4 datagram are the following:

- ***Version number***. These 4 bits specify **the IP protocol version of the datagram**.By looking at the version number, the router can determine how to interpret the remainder of the IP datagram. Different versions of IP use different datagram formats. The datagram format for the current version of IP, IPv4, is shown in Figure 4.13. The datagram format for the new version of IP (IPv6) is discussed at the end of this section.

- ***Header length***. Because an IPv4 datagram can contain a variable number of options (which are included in the IPv4 datagram header), these 4 bits are needed **to determine where in the IP datagram the data actually begins.** Most IP datagrams do not contain options, so the typical IP datagram has a **20-byte header.**
- ***Type of service.*** The type of service (TOS) bits were included in the IPv4 header to allow **different types of IP datagrams to be distinguished from each other** (for example, datagrams particularly requiring low delay, high throughput, or reliability). For example, it might be useful to distinguish real-time datagrams (such as those used by an IP telephony application) from non-real-time traffic (for example, FTP).
- ***Datagram length***. This is the **total length of the IP datagram** (header plus data), measured in bytes. Since this field is 16 bits long, the theoretical maximum size of the IP datagram is 65,535 bytes. However, datagrams are rarely larger than 1,500 bytes.

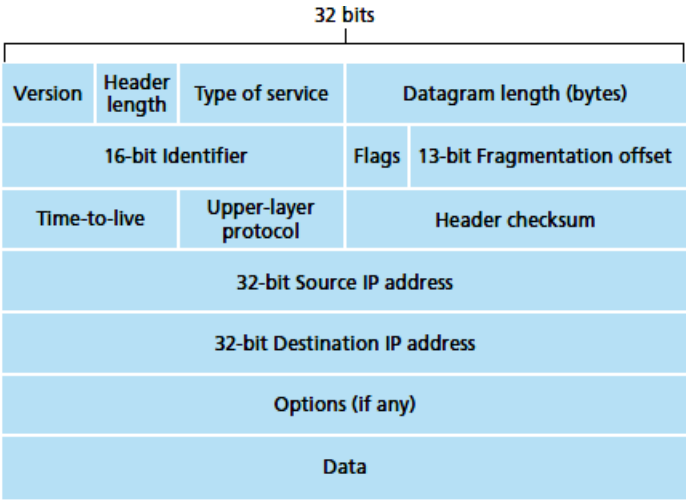


Figure 4.13 ♦ IPv4 datagram format

- ***Identifier, flags, fragmentation offset***. These three fields deal with **IP fragmentation**. Interestingly, the new version of IP, IPv6, does not allow for fragmentation at routers.
- ***Time-to-live***. The time-to-live (TTL) field is included **to ensure that datagrams do not circulate forever** (due to, for example, a long-lived routing loop) in the network. This field is decremented by one each time the datagram is processed by a router. If the TTL field reaches 0, the datagram must be dropped.
- ***Protocol***. This field is used only when an IP datagram reaches its final destination.The value of this field indicates the specific transport-layer protocol to which the data portion of this IP datagram should be passed. For example, a value of 6 indicates that the data portion is passed to TCP, while a value of 17 indicates that the data is passed to UDP. Note that the protocol number in the IP datagram has a role that is analogous to the role of the port number field in the transport layer segment. The protocol number is the glue that binds the network and transport layers together, whereas the port number is the glue that binds the transport and application layers together.
- ***Header checksum***. The header checksum aids a router in detecting bit errors in a received IP datagram. The header checksum is computed by treating each 2 bytes in the header as a number and summing these numbers using 1s complement arithmetic.

A router computes the header checksum for each received IP datagram and detects an error condition if

the checksum carried in the datagram header does not equal the computed checksum. Routers typically discard datagrams for which an error has been detected.

why does TCP/IP perform error checking at both the transport and network layers?

- First, note that only the IP header is checksummed at the IP layer, while the TCP/UDP checksum is computed over the entire TCP/UDP segment.

- Second, TCP/UDP and IP do not necessarily both have to belong to the same protocol stack. TCP can, in principle, run over a different protocol (for example, ATM) and IP can carry data that will not be passed to TCP/UDP.

• *Source and destination IP addresses.* When a source creates a datagram, it inserts its IP address into the source IP address field and inserts the address of the ultimate destination into the destination IP address field.

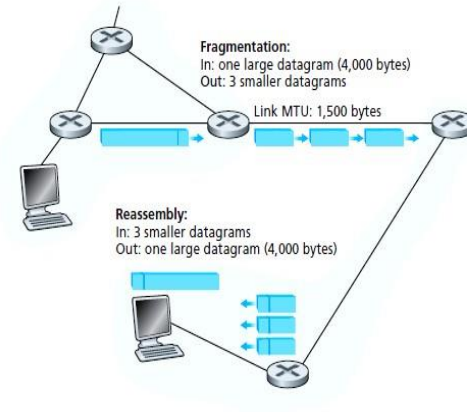
• *Options.* The options fields allow an IP header to be extended. Header options were meant to be used rarely—hence the decision to save overhead by not including the information in options fields in every datagram header.

• *Data (payload).* the data field of the IP datagram contains the transport-layer segment (TCP or UDP) to be delivered to the destination. However, the data field can carry other types of data, such as ICMP messages.

Note that an IP datagram has a total of 20 bytes of header (assuming no options). If the datagram carries a TCP segment, then each (nonfragmented) datagram carries a total of 40 bytes of header (20 bytes of IP header plus 20 bytes of TCP header) along with the application-layer message.

IP Datagram Fragmentation

- Not all link-layer protocols can carry network-layer packets of the same size. Some protocols can carry big datagrams, whereas other protocols can carry only little packets.
  - For example, Ethernet frames can carry up to 1,500 bytes of data, whereas frames for some wide-area links can carry no more than 576 bytes.
- The maximum amount of data that a link-layer frame can carry is called the **maximum transmission unit (MTU).**
- Because each IP datagram is encapsulated within the link-layer frame for transport from one router to the next router, the MTU of the link-layer protocol places a hard limit on the length of an IP datagram.
- That each of the links along the route between sender and destination can use different link-layer protocols, and each of these protocols can have different MTUs.
- If we have to send data through a link with MTU that is smaller than the length of the IP datagram. How are you going to squeeze this oversized IP datagram into the payload field of the link-layer frame?
- The solution is to fragment the data in the IP datagram into two or more smaller IP datagrams, encapsulate each of these smaller IP datagrams in a separate link-layer frame; and send these frames over the outgoing link. Each of these smaller datagrams is referred to as a **fragment**.



- Fragments need to be reassembled before they reach the transport layer at the destination. Indeed, both TCP and UDP are expecting to receive complete, unfragmented segments from the network layer.
- The job of datagram reassembly lies the end systems rather than in network routers.

When a destination host receives a series of datagrams from the same source, it needs to determine whether any of these datagrams are fragments of some original, larger datagram. If

some datagrams are fragments, it must further determine when it has received the last fragment and how the fragments it has received should be
pieced back together to form the original datagram.
To allow the destination host to perform these reassembly tasks, the designers of IP (version 4) put *identification,*
flag, and *fragmentation offset* fields in the IP datagram header.

- When a datagram is created, the sending host stamps the datagram with an identification number as well
  as source and destination addresses. Typically, the sending host increments the identification
  number for each datagram it sends.
- When a router needs to fragment a datagram, each resulting datagram (that is, fragment) is stamped with the source address, destination address, and identification number of the original datagram.
- When the destination receives a series of datagrams from the same sending host, it can examine the identification numbers of the datagrams to determine which of the datagrams are actually fragments of the same larger datagram.
- Because IP is an unreliable service, one or more of the fragments may never arrive at the destination.For this reason, in order for the destination host to be absolutely sure it has received the last fragment of the original datagram, the last fragment has a flag bit set to 0,whereas all the other fragments have this flag bit set to 1.
- Also, in order for the destination host to determine whether a fragment is missing (and also to be able to
  reassemble the fragments in their proper order), the offset field is used to specify where the fragment fits within the original IP datagram.

Fragment	Bytes	ID	Offset	Flag
1st fragment	1,480 bytes in the data field of the IP datagram	identification = 777	offset = 0 (meaning the data should be inserted beginning at byte 0)	flag = 1 (meaning there is more)
2nd fragment	1,480 bytes of data	identification = 777	offset = 185 (meaning the data should be inserted beginning at byte 1,480. Note that 185 · 8 = 1,480)	flag = 1 (meaning there is more)
3rd fragment	1,020 bytes (= 3,980–1,480–1,480) of data	identification = 777	offset = 370 (meaning the data should be inserted beginning at byte 2,960. Note that 370 · 8 = 2,960)	flag = 0 (meaning this is the last fragment)

Table 4.2 ♦ IP fragments

Figure 4.14 illustrates an example. A datagram of 4,000 bytes (20 bytes of IP header plus 3,980 bytes of IP payload) arrives at a router and must be forwarded to a link with an MTU of 1,500 bytes. This implies that the 3,980 data bytes in the original datagram must be allocated to three separate fragments (each of which is also an IP datagram). Suppose that the original datagram is stamped with an identification number of 777. The characteristics of the three fragments are shown in Table 4.2. The values in Table 4.2 reflect the requirement that the amount of original payload data in all but the last fragment be a multiple of 8 bytes, and that the offset value be specified in units of 8-byte chunks.

But fragmentation also has its Disadvantages.

- First, it complicates routers and end systems, which need to be designed to accommodate datagram fragmentation and reassembly.
- Second, fragmentation can be used to create lethal DoS attacks, whereby the attacker sends a series of bizarre and unexpected fragments. A classic example is the Jolt2 attack, where the attacker sends a stream of small fragments to the target host, none of which has an offset of zero. The target can collapse as it attempts to rebuild datagrams out of the degenerate packets. Another class of exploits sends overlapping IP fragments, that is, fragments whose offset values are set so that the fragments do not align properly. Vulnerable operating systems, not knowing what to do with overlapping fragments,
  can crash [Skoudis 2006].

4.4.2 IPv4 Addressing

A host typically has only a single link into the network; when IP in the host wants to send a datagram, it does
so over this link. The boundary between the host and the physical link is called an **interface**. Now consider a router and its interfaces. Because a router's job is to receive a datagram on one link and forward the datagram on some other link, a router necessarily has two or more links to which it is connected. **The boundary between the router and any one of its links is also called an interface.** A router thus has multiple interfaces, one for each of its links. Because every host and router is capable of sending and
receiving IP datagrams, IP requires each host and router interface to have its own IP address. Thus, an IP address is technically associated with an interface, rather than with the host or router containing that interface.

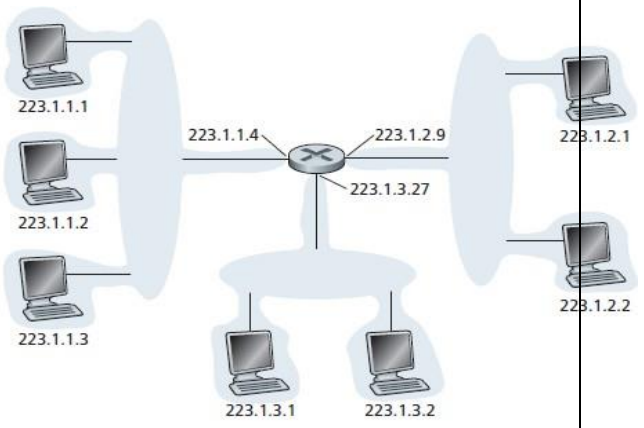
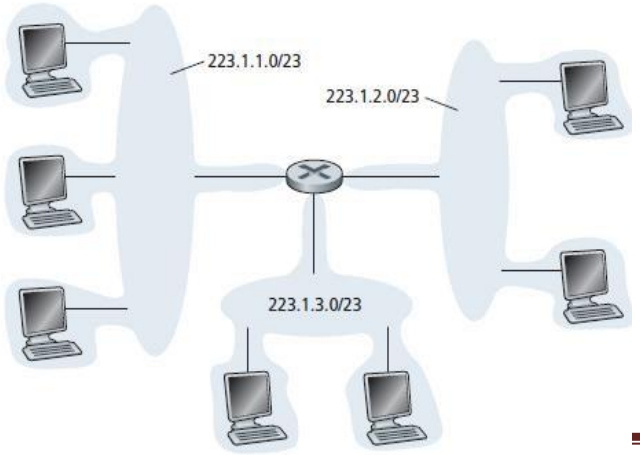


Figure 4.15 ♦ Interface addresses and subnets

Each IP address is 32 bits long (equivalently, 4 bytes), and there are thus a total of 2^{32} possible IP addresses. By approximating 210 by 103, it is easy to see that there are about 4 billion possible IP addresses. These addresses are typically written in so-called **dotted-decimal notation**, in which each byte of the address is written in
its decimal form and is separated by a period (dot) from other bytes in the address.

Each interface on every host and router in the global Internet must have an IP address that is globally unique.



gure 4.16 ♦ Subnet addresses

These addresses cannot be chosen randomly,however. A portion of an interface's IP address will be determined by the subnet to which it is connected.

Figure 4.15 provides an example of IP addressing and interfaces. In this figure, one router (with three interfaces) is used to interconnect seven hosts.
Notice,the three hosts in the upper-left portion of Figure 4.15, and the router interface to which they are connected, all

have an IP address of the form 223.1.1.xxx. That is, they all have the same leftmost 24 bits in their IP address.

The four interfaces are also interconnected to each other by a network *that contains no routers*. This network could be interconnected by an Ethernet LAN, in which case the interfaces would be interconnected by an Ethernet, or by a wireless access point

In IP terms, this network interconnecting three host interfaces and one router interface forms a **subnet** [RFC 950]. (A subnet is also called an *IP network* or simply a *network* in the Internet literature.)

IP addressing assigns an address to this subnet: 223.1.1.0/24, where the /24 notation, sometimes known as a **subnet mask**, indicates that the leftmost 24 bits of the 32-bit quantity define the subnet address. The subnet 223.1.1.0/24 thus consists of the three host interfaces (223.1.1.1, 223.1.1.2, and 223.1.1.3) and one router interface (223.1.1.4). Any additional hosts attached to the 223.1.1.0/24 subnet would be *required* to have an address of the form 223.1.1.xxx.

There are two additional subnets shown in Figure 4.15: the 223.1.2.0/24 network and the 223.1.3.0/24 subnet. Figure 4.16 illustrates the three IP subnets present in Figure 4.15.

The IP definition of a subnet is not restricted to Ethernet segments that connect multiple hosts to a router interface.

Figure 4.17, which shows three routers that are interconnected with each other by point-to-point links. Each router has three interfaces, one for each point-to-point link and one for the broadcast link that directly connects the router to a pair of hosts. What subnets are present here? Three subnets, 223.1.1.0/24, 223.1.2.0/24, and 223.1.3.0/24, are similar to the subnets we encountered in Figure 4.15.

But note that there are three additional subnets in this example as well: one subnet, 223.1.9.0/24, for the interfaces that connect routers R1 and R2; another subnet, 223.1.8.0/24, for the interfaces that connect routers R2 and R3; and a third subnet, 223.1.7.0/24, for the interfaces that connect routers R3 and R1.

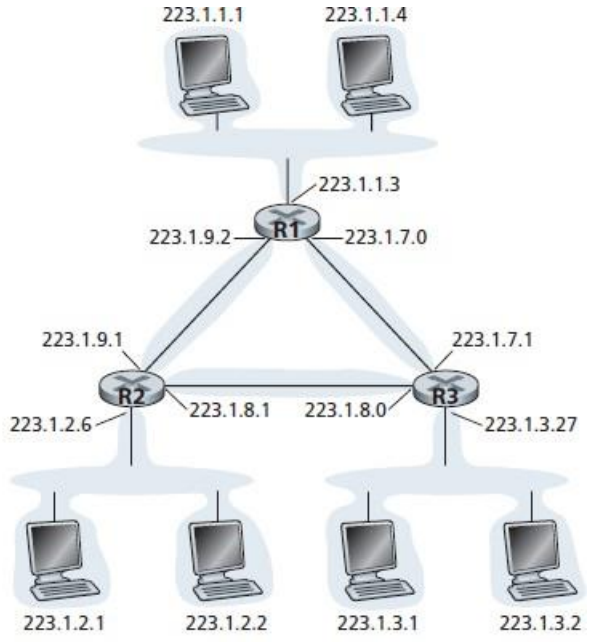


Figure 4.17 ♦ Three routers interconnecting six subnets

For a general interconnected system of routers and hosts,we can use the following recipe to define the subnets in the system:

To determine the subnets, detach each interface from its host or router, creating islands of isolated networks, with interfaces terminating the end points of the isolated networks. Each of these isolated networks is called a ***subnet****.*

If we apply this procedure to the interconnected system in Figure 4.17, we get six islands or subnets.It's clear that an organization (such as a company or academic institution) with multiple Ethernet segments and point-to-point links will have multiple subnets, with all of the devices on a given subnet having the same subnet address.

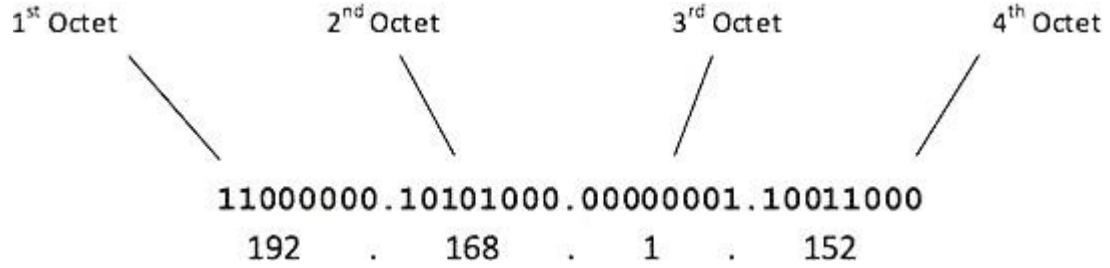
In principle, the different subnets could have quite different subnet addresses. In practice, however, their subnet addresses often have much in common.
To understand why, let's next turn our attention to how addressing is handled in The Internet's address assignment strategy is known as **Classless Interdomain Routing** (**CIDR**—pronounced *cider*) [RFC 4632]. CIDR generalizes the notion of subnet addressing.

- As with subnet addressing, the 32-bit IP address is divided into two parts and again has the dotted-decimal form *a.b.c.d/x*, where *x* indicates the number of bits in the first part of the address.
- The *x* most significant bits of an address of the form *a.b.c.d/x* constitute the network portion of the IP address, and are often referred to as the **prefix** (or *network prefix*) of the address.
- An organization is typically assigned a block of contiguous addresses, that is, a range of addresses with a common prefix. In this case, the IP addresses of devices within the organization will share the common prefix.
- only these *x* leading prefix bits are considered by routers outside the organization's network. That is, when a router outside the organization forwards a datagram whose destination address is inside the organization, only the leading *x* bits of the address need be considered.
- This considerably reduces the size of the forwarding table in these routers, since a *single* entry of the form *a.b.c.d/x* will be sufficient to forward packets to *any* destination within the organization.
- The remaining 32-*x* bits of an address can be thought of as distinguishing among the devices *within* the organization, all of which have the same network prefix.These are the bits that will be considered when forwarding packets at routers*within* the organization. These lower-order bits may (or may not) have an additional subnetting structure, such as that discussed above.
- For example, suppose the first 21 bits of the CIDRized address a.b.c.d/21 specify the organization's network prefix and are common to the IP addresses of all devices in that organization. The remaining 11 bits then identify the specific hosts in the organization.
- The organization's internal structure might be such that these 11 rightmost bits are used for subnetting
  within the organization, as discussed above. For example, a.b.c.d/24 might refer to a specific subnet within the organization.

Before CIDR was adopted, the network portions of an IP address were constrained to be 8, 16, or 24 bits in length, an addressing scheme known as **classful addressing**, since subnets with 8-, 16-, and 24-bit subnet addresses were known as class A, B, and C networks, respectively. The requirement that the subnet portion of
an IP address be exactly 1, 2, or 3 bytes long turned out to be problematic for supporting the rapidly growing number of organizations with small and medium-sized subnets. A class C (/24) subnet could accommodate only up to 28 – 2 = 254 hosts (two of the 28 = 256 addresses are reserved for special use)—too small for many
organizations. However, a class B (/16) subnet, which supports up to 65,634 hosts, was too large. Under classful addressing, an organization with, say, 2,000 hosts was typically allocated a class B (/16) subnet address. This led to a rapid depletion of the class B address space and poor utilization of the assigned address space. For example, the organization that used a class B address for its 2,000 hosts was allocated enough of the address space for up to

65,534 interfaces—leaving more than 63,000 addresses that could not be used by other organizations.

Internet Protocol hierarchy contains several classes of IP Addresses to be used efficiently in various situations as per the requirement of hosts per network. Broadly, the IPv4 Addressing system is divided into five classes of IP Addresses. All the five classes are identified by the first octet of IP Address.
Internet Corporation for Assigned Names and Numbers is responsible for assigning IP addresses.
The first octet referred here is the left most of all. The octets numbered as follows depicting dotted decimal notation of IP Address:



The number of networks and the number of hosts per class can be derived by this formula:

Number of networks = 2^network_bits

Number of Hosts/Network = 2^host_bits – 2

When calculating hosts' IP addresses, 2 IP addresses are decreased because they cannot be assigned to hosts, i.e. the first IP of a network is network number and the last IP is reserved for Broadcast IP.

- **Class A Address**
  The first bit of the first octet is always set to 0 (zero). Thus the first octet ranges from 1 – 127, i.e.

  00000001 – 01111111
  1 – 127

  Class A addresses only include IP starting from 1.x.x.x to 126.x.x.x only. The IP range 127.x.x.x is reserved for loopback IP addresses.
  The default subnet mask for Class A IP address is 255.0.0.0 which implies that Class A addressing can have 126 networks (2^7-2) and 16777214 hosts (2^{24}-2).
  Class A IP address format is thus: **0NNNNNNN**.HHHHHHHH.HHHHHHHH.HHHHHHHH
- **Class B Address**
  An IP address which belongs to class B has the first two bits in the first octet set to 10, i.e.

  10000000 – 10111111
  128 – 191

  Class B IP Addresses range from 128.0.x.x to 191.255.x.x. The default subnet mask for Class B is 255.255.x.x.
  Class B has 16384 (2^{14}) Network addresses and 65534 (2^{16}-2) Host addresses.
  Class B IP address format is: **10NNNNNN.NNNNNNNN**.HHHHHHHH.HHHHHHHH

- **Class C Address**
  The first octet of Class C IP address has its first 3 bits set to 110, that is:

  11000000 – 11011111
  192 – 223

  Class C IP addresses range from 192.0.0.x to 223.255.255.x. The default subnet mask for Class C is 255.255.255.x.
  Class C gives 2097152 (2^{21}) Network addresses and 254 (2^8-2) Host addresses.
  Class C IP address format is: **110NNNNN.NNNNNNNN.NNNNNNNN**.HHHHHHHH
- **Class D Address**
  Very first four bits of the first octet in Class D IP addresses are set to 1110, giving a range of:

  11100000 – 11101111
  224 – 239

  Class D has IP address rage from 224.0.0.0 to 239.255.255.255. Class D is reserved for Multicasting. In multicasting data is not destined for a particular host, that is why there is no need to extract host address from the IP address, and Class D does not have any subnet mask.
- **Class E Address**
  This IP Class is reserved for experimental purposes only for R&D or Study. IP addresses in this class ranges from 240.0.0.0 to 255.255.255.254. Like Class D, this class too is not equipped with any subnet mask.

Obtaining a Block of Addresses

- In order to obtain a block of IP addresses for use within an organization's subnet, a network administrator might first contact its ISP, which would provide addresses from a larger block of addresses that had already been allocated to the ISP.
  For example, the ISP may itself have been allocated the address block 200.23.16.0/20.

- The ISP, in turn, could divide its address block into eight equal-sized contiguous address blocks and give one of these address blocks out to each of up to eight organizations that are supported by this ISP, as shown below.
- ISP's block 200.23.16.0/20 11001000 00010111 00010000 00000000
  Organization 0 200.23.16.0/23 11001000 00010111 00010000 00000000
  Organization 1 200.23.18.0/23 11001000 00010111 00010010 00000000
  Organization 2 200.23.20.0/23 11001000 00010111 00010100 00000000
  . . . . . .
  Organization 7 200.23.30.0/23 11001000 00010111 00011110 00000000
- While obtaining a set of addresses from an ISP is one way to get a block of addresses, it is not the only way. Clearly, there must also be a way for the ISP itself to get a block of addresses. Is there a global authority that has ultimate responsibility for managing the IP address space and allocating address blocks to ISPs and other organizations? IP addresses are managed under the authority of the Internet Corporation for Assigned Names and Numbers (ICANN) [ICANN 2012],based on guidelines set forth in [RFC 2050].

- The role of the nonprofit ICANN organization [NTIA 1998] is not only to allocate IP addresses, but also to manage the DNS root servers. It also has the very contentious job of assigning domain names and resolving domain name disputes. The ICANN allocates addresses to regional Internet registries (for example, ARIN, RIPE, APNIC, and LACNIC, which together form the Address Supporting Organization of ICANN [ASO-ICANN 2012]), and handle the allocation/management of addresses within their regions.

Obtaining a Host Address: the Dynamic Host Configuration Protocol

- Once an organization has obtained a block of addresses, it can assign individual IP addresses to the host and router interfaces in its organization. A system administrator will typically manually configure the IP addresses into the router (often remotely, with a network management tool).
- Host addresses can also be configured manually, but more often this task is now done using the **Dynamic Host Configuration Protocol (DHCP)** [RFC 2131]. DHCP allows a host to obtain (be allocated) an IP address automatically.
- A network administrator can configure DHCP so that a given host receives the same IP address each time it connects to the network, or a host may be assigned a **temporary IP address** that will be different each time the host connects to the network.
- In addition to host IP address assignment, DHCP also allows a host to learn additional information, such as its subnet mask, the address of its first-hop router (often called the default gateway), and the address of its local DNS server.
- Because of DHCP's ability to automate the network-related aspects of connecting a host into a network, it is often referred to as a **plug-and-play protocol**.
- DHCP is also enjoying widespread use in residential Internet access networks and in wireless LANs, where hosts join and leave the network frequently.
- Consider, for example, the student who carries a laptop from a dormitory room to a library to a classroom. It is likely that in each location, the student will be connecting into a new subnet and hence will need a new IP address at each location. DHCP is ideally suited to this situation, as there are many users coming and going, and addresses are needed for only a limited amount of time.

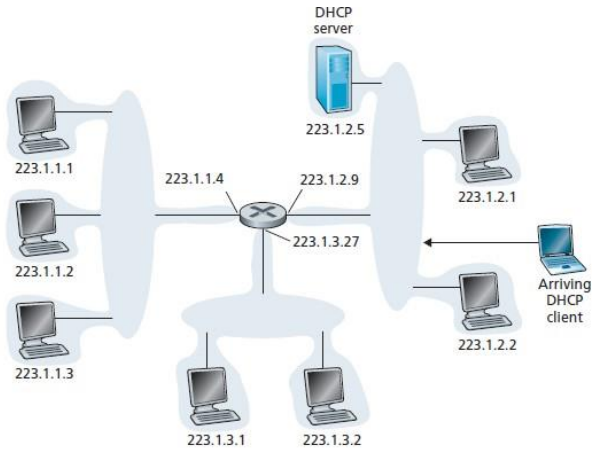
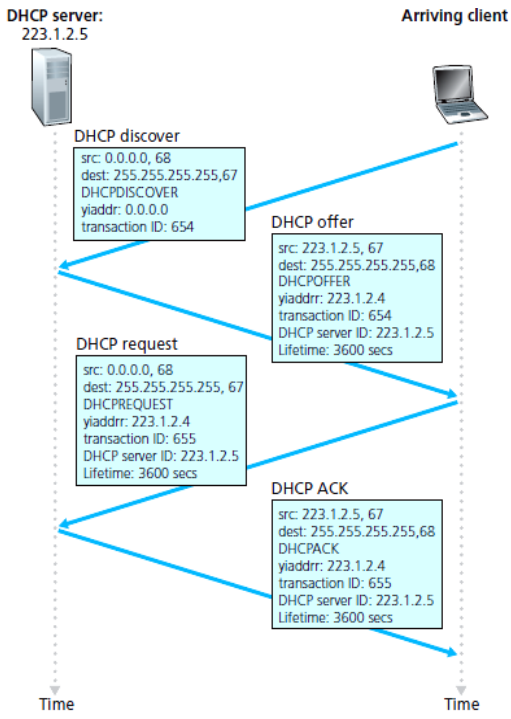


Figure 4.20 ♦ DHCP client-server scenario

- DHCP is similarly useful in residential ISP access networks. Consider, for example, a residential ISP that has 2,000 customers, but no more than 400 customers are ever online at the same time. In this case, rather than needing a block of 2,048 addresses, a DHCP server that assigns addresses dynamically needs only a block of 512 addresses (for example, a block of the form a.b.c.d/23).
- As the hosts join and leave, the DHCP server needs to update its list of available IP addresses. Each time a host joins, the DHCP server allocates an arbitrary address from its current pool of available addresses; each time a host leaves, its address is returned to the pool.

- DHCP is a client-server protocol. A client is typically a newly arriving host wanting to obtain network configuration information, including an IP address for itself. In the simplest case, each subnet (in the addressing sense of Figure 4.17) will have a DHCP server. If no server is present on the subnet, a DHCP relay agent (typically a router) that knows the address of a DHCP server for that network is needed.



ure 4.21 ♦ DHCP client-server interaction

- Figure 4.20 shows a DHCP server attached to subnet 223.1.2/24, with the router serving as the relay agent for arriving clients attached to subnets 223.1.1/24 and 223.1.3/24. In our discussion below, we'll assume that a DHCP server is available on the subnet.
- For a newly arriving host, the DHCP protocol is a four-step process, as shown in Figure 4.21 for the network setting shown in Figure 4.20. In this figure, yiaddr (as in "your Internet address") indicates the address being allocated to the newly arriving client. The four steps are:
  - ***DHCP server discovery.*** The first task of a newly arriving host is to find a DHCP server with which to interact. This is done using a **DHCP discover message**, which a client sends within a UDP packet to port 67. The UDP packet is encapsulated in an IP datagram.
  - The DHCP client creates an IP datagram containing its DHCP discover message along with the broadcast destination IP address of 255.255.255.255 and a "this host" source IP address of 0.0.0.0. The DHCP client passes the IP datagram to the link layer, which then broadcasts this frame to all nodes attached to the subnet
  - ***DHCP server offer(s).*** A DHCP server receiving a DHCP discover message responds to the client with a **DHCP offer message** that is broadcast to all nodes on the subnet, again using the IP broadcast address of 255.255.255.255. Since several DHCP servers can be present on the subnet, the client may find itself in the enviable position of being able to choose from among several offers. Each server offer message contains the transaction ID of the received discover message, the proposed IP address for the client, the network mask, and an IP **address lease time**—the amount of time for which the IP address will be valid. It is common for the server to set the lease time to several hours or days.
  - *DHCP request.* The newly arriving client will choose from among one or more server offers and respond to its selected offer with a **DHCP request message**, echoing back the configuration parameters.
  - *DHCP ACK.* The server responds to the DHCP request message with a **DHCP ACK message**, confirming the requested parameters.

Once the client receives the DHCP ACK, the interaction is complete and the client can use the DHCP-allocated IP address for the lease duration. Since a client may want to use its address beyond the lease's expiration, DHCP also provides a mechanism that allows a client to renew its lease on an IP address.

The value of DHCP's plug-and-play capability is clear, considering the fact that the alternative is to manually configure a host's IP address. Consider the student who moves from classroom to library to dorm room with a laptop, joins a new subnet, and thus obtains a new IP address at each location. It is unimaginable that a system
administrator would have to reconfigure laptops at each location, and few students (except those taking a computer networking class!) would have the expertise to configure their laptops manually. From a mobility aspect, however, DHCP does have shortcomings. Since a new IP address is obtained from DHCP each time a node connects to a new subnet, a TCP connection to a remote application cannot be maintained as a mobile node moves between subnets.

Network Address Translation (NAT)

Given our discussion about Internet addresses and the IPv4 datagram format, we're now well aware that every IP-capable device needs an IP address. With the proliferation of small office, home office (SOHO) subnets, this would seem to imply that whenever a SOHO wants to install a LAN to connect multiple machines, a range of
addresses would need to be allocated by the ISP to cover all of the SOHO's machines. If the subnet grew bigger (for example, the kids at home have not only their own computers, but have smartphones and networked Game Boys as well), a larger block of addresses would have to be allocated. But what if the ISP had already allocated the contiguous portions of the SOHO network's current address range?
And what typical homeowner wants (or should need) to know how to manage IP addresses in the first place? Fortunately, there is a simpler approach to address allocation that has found increasingly widespread use in such scenarios: **network address translation** (**NAT**) [RFC 2663; RFC 3022; Zhang 2007].

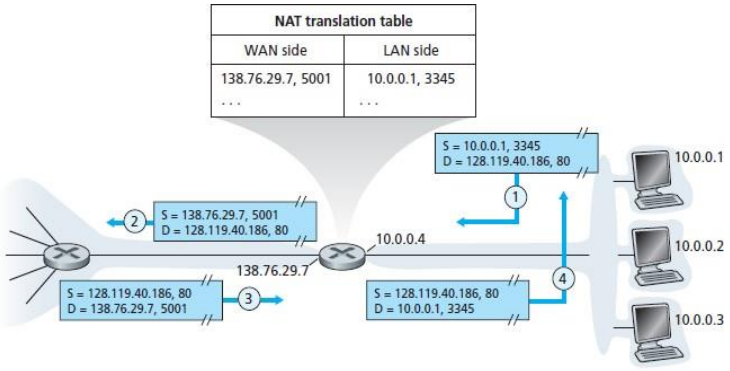


Figure 4.22 ♦ Network address translation

Figure 4.22 shows the operation of a NAT-enabled router.

- The NAT-enabled router, residing in the home, has an interface that is part of the home network on the

right of Figure 4.22.

- Addressing within the home network is exactly as we have seen

above—all four interfaces in the home network have the same subnet address of 10.0.0/24. The address space 10.0.0.0/8 is one of three portions of the IP address space that is reserved in [RFC 1918] for a private network or a **realm** with private addresses, such as the home network in Figure 4.22.

- A *realm with private addresses* refers to a network whose addresses only have meaning to devices within that network. To see why this is important, consider the fact that there are hundreds of thousands of home networks, many using the same

address space, 10.0.0.0/24.Devices within a given home network can send packets to each other using 10.0.0.0/24 addressing.

- However, packets forwarded *beyond* the home network into the larger global Internet clearly cannot use these addresses (as either a source or a destination address) because there are hundreds of thousands of networks using this block of addresses.
- That is, the 10.0.0.0/24 addresses can only have meaning within the given home network. But if private addresses only have meaning within a given network, how is addressing handled when packets are sent to or received from the global Internet, where addresses are necessarily unique?
- The NAT-enabled router does not *look* like a router to the outside world. Instead the NAT router behaves to the outside world as a *single* device with a *single* IP address. In Figure 4.22, all traffic leaving the home router for the larger Internet has a source IP address of 138.76.29.7, and all traffic entering the home router must have a destination address of 138.76.29.7.
- In essence, the NAT-enabled router is hiding the details of the home network from the outside world. If all datagrams arriving at the NAT router from the WAN have the same destination IP address (specifically, that of the WAN-side interface of the NAT router), then how does the router know the internal host to which it should forward a given datagram?
- The trick is to use a **NAT translation table** at the NAT router, and to include port numbers as well as IP addresses in the table entries.

Consider the example in Figure 4.22.

- Suppose a user sitting in a home network behind host 10.0.0.1 requests a Web page on some Web server (port 80) with IP address 128.119.40.186.
- The host 10.0.0.1 assigns the (arbitrary) source port number 3345 and sends the datagram into the LAN. The NAT router receives the datagram, generates a new source port number 5001 for the datagram, replaces the source IP address with its WAN-side IP address 138.76.29.7, and replaces the original source port number 3345 with the new source port number 5001.
- When generating a new source port number, the NAT router can select any source port number that is not currently in the NAT translation table. (Note that because a port number field is 16 bits long, the NAT protocol can support over 60,000 simultaneous connections with a single WAN-side IP address for the router!) NAT in the router also adds an entry to its NAT translation table.
- The Web server, blissfully unaware that the arriving datagram containing the HTTP request has been manipulated by the NAT router, responds with a datagram whose destination address is the IP address of the NAT router, and whose destination port number is 5001.
- When this datagram arrives at the NAT router, the router indexes the NAT translation table using the destination IP address and destination port number to obtain the appropriate IP address (10.0.0.1) and destination port number (3345) for the browser in the home network.
- The router then rewrites the datagram's destination address and destination port number, and forwards the datagram into the home network.

DISADVANTAGES

- First, port numbers are meant to be used for addressing processes, not for addressing hosts.
- Second, routers are supposed to process packets only up to layer 3.
- Third, the NAT protocol violates the so-called end-to-end argument; that is, hosts should be talking directly with each other, without interfering nodes modifying IP addresses and port numbers.
- And fourth, we should use IPv6 to solve the shortage of IP addresses, rather than recklessly patching up the problem with a stopgap solution like NAT.
- Yet another major problem with NAT is that it interferes with P2P applications, including P2P file-sharing applications and P2P Voice-over-IP applications. Recall from Chapter 2 that in a P2P application, any participating Peer A should be able to initiate a TCP connection to any other participating Peer B. The essence of the problem is that if Peer B is behind a NAT, it cannot act as a server and accept TCP connections. In this case, Peer A can first contact Peer B through an intermediate Peer C, which is not behind a NAT and to which B has established an ongoing TCP connection. Peer A can then ask Peer B, via Peer C, to initiate a TCP connection directly back to Peer A. Once the direct P2P TCP connection Is established between Peers A and B, the two peers can exchange messages or files. This hack, called **connection reversal**, is actually used by many P2P applications for **NAT traversal**.

UPnP

NAT traversal is provided by Universal Plug and Play (UPnP), a protocol that allows a host to discover and configure a nearby NAT.

UPnP requires that both the host and the NAT be UPnP compatible. With UPnP, an application running in a host can request a NAT mapping between its (*private IP address, private port number*) and the (*public IP address, public port number*) for some requested public port number.

If the NAT accepts the request and creates the mapping, then nodes from the outside can initiate TCP connections to (*public IP address, public port number*).

UPnP lets the application know the value of (*public IP address, public port number*), so that the application can advertise it to the outside world.

Example: Suppose a host, behind a UPnP-enabled NAT, has private address 10.0.0.1 and is running BitTorrent on port 3345. Suppose that the public IP address of the NAT is 138.76.29.7.

BitTorrent application naturally wants to be able to accept connections from other hosts, so that it can trade chunks with them.

BitTorrent application in a host asks the NAT to create a “hole” that maps (10.0.0.1, 3345) to (138.76.29.7, 5001). (The public port number 5001 is chosen by the application.)

The BitTorrent application in a host could also advertise to its tracker that it is available at (138.76.29.7, 5001). Hence, an external host running BitTorrent can contact the tracker and learn that BitTorrent application is running at (138.76.29.7, 5001).

The external host can send a TCP SYN packet to (138.76.29.7, 5001). When the NAT receives the SYN packet, it will change the destination IP address and port number in the packet to (10.0.0.1, 3345) and forward the packet through the NAT.

UPnP allows external hosts to initiate communication sessions to NATed hosts, using either TCP or UDP.

Internet Control Message Protocol (ICMP)

ICMP, 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, an error message such as “Destination network unreachable.” This message had its origins in ICMP.

If an IP router is unable to find a path to the host specified in Telnet, FTP, or HTTP application then router creates a type-3 ICMP message and sends it to host indicating the error.

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.

ICMP message types are shown in Figure below.

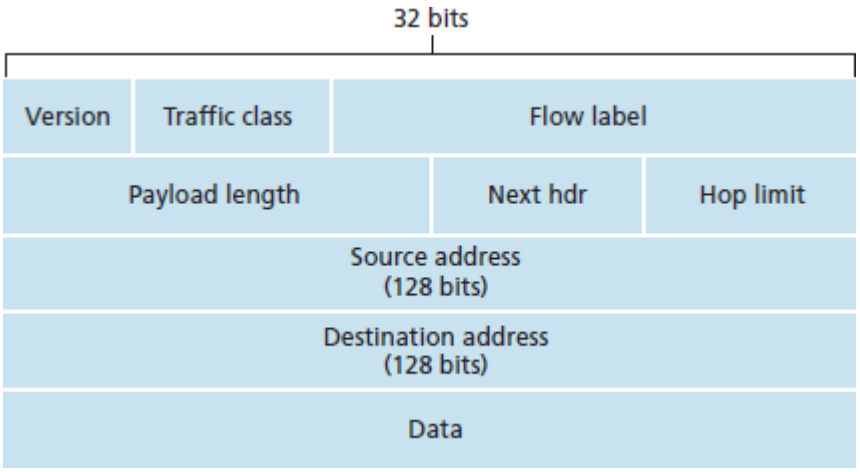
- The well-known ping program sends an ICMP type 8 code 0 message to the specified host.
- The destination host, seeing the echo request, sends back a type 0 code 0 ICMP echo reply.
- Client program needs to be able to instruct the operating system to generate an ICMP message of type 8 code 0.
- ICMP message is the source quench message whose purpose is 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.

- Traceroute program allows to trace a route from a host to any other host in the world. 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 *n*th datagram arrives at the *n*th router, the *n*th 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 *n*th router from the ICMP message.
- 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 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.

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

IPv6

IPv6 Datagram Format



The format of the IPv6 datagram is shown in Figure above. The most important changes introduced in IPv6 are evident in the datagram format:

• ***Expanded addressing capabilities***. IPv6 increases the size of the IP address from 32 to 128 bits.

IPv6 has introduced a new type of address, called an anycast address, which allows a datagram to be delivered to any one of a group of hosts.

• ***A streamlined 40-byte header***. The 40-byte fixed-length header allows for faster processing of the IP datagram.

• ***Flow labeling and priority.***

IPV6 allows "labelling of packets belonging to particular flows for which the sender requests special handling, such as a non default quality of service or real-time service."

For example, audio and video transmission might likely be treated as a flow.

Traditional applications, such as file transfer and e-mail, might not be treated as flows.

The IPv6 header also has an 8-bit traffic class field. This field, like the TOS field in IPv4, can be used to give priority to certain datagrams within a flow, or it can be used to give priority to datagrams from certain applications over datagrams from other applications.

The following fields are defined in IPv6:

• ***Version***. This 4-bit field identifies the IP version number. IPv6 carries a value of 6 in this field.

•***Traffic class***. This 8-bit field is similar to the TOS field we saw in IPv4.

• ***Flow label.*** *T*his 20-bit field is used to identify a flow of datagrams.

• ***Payload length***. This 16-bit value is treated as an unsigned integer giving the number of bytes in the IPv6 datagram following the fixed-length, 40-byte datagram header.

• ***Next header***. This field identifies the protocol to which the contents (data field) of this datagram will be delivered. The field uses the same values as the protocol field in the IPv4 header.

• ***Hop limit.*** The contents of this field are decremented by one by each router that forwards the datagram. If the hop limit count reaches zero, the datagram is discarded.

• ***Source and destination addresses***. The various formats of the IPv6 128-bit address

• ***Data***. This is the payload portion of the IPv6 datagram. When the datagram reaches its destination, the payload will be removed from the IP datagram and passed on to the protocol specified in the next header field.

Following are the differences between IPV4 & IPV6 :

• ***Fragmentation/Reassembly***. IPv6 does not allow for fragmentation and reassembly at intermediate routers; these operations can be performed only by the source and destination. If an IPv6 datagram received by a router is too large to be forwarded over the outgoing link, the router simply drops the datagram and sends a "Packet Too Big" ICMP error message (see below) back to the sender.

The sender can then resend the data, using a smaller IP datagram size. Fragmentation and reassembly is a time-consuming operation; removing this functionality from the routers and placing it squarely in the end systems considerably speeds up IP forwarding within the network.

• ***Header checksum***. *T*he transport-layer (for example, TCP and UDP) and link-layer (for example, Ethernet) protocols in the Internet layers perform checksumming, the designers of IP probably felt that this functionality was sufficiently redundant in the network layer that it could be removed.

Since the IPv4 header contains a TTL field (similar to the hop limit field in IPv6), the IPv4 header checksum needed to be recomputed at every router. As with fragmentation and reassembly, was a costly operation in IPv4.

• ***Options***. An options field is no longer a part of the standard IP header. Options field is one of the possible next headers pointed to from within the IPv6 header. That is, just as TCP or UDP protocol headers can be the next header within an IP packet. The removal of the options field results in a fixed-length, 40-byte IP header.

Transitioning from IPV4 to IPV6 :

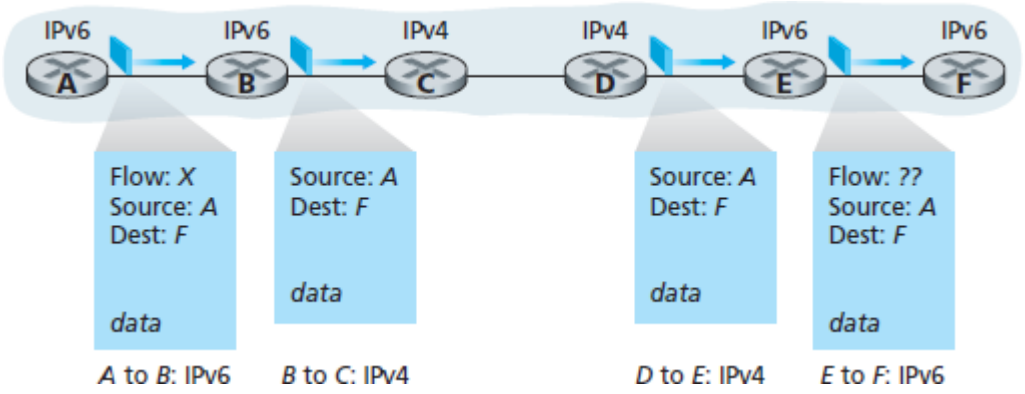
IPv6-capable nodes has a dual-stack approach, where IPv6 nodes have a complete IPv4 implementation. Such a node, referred to as an IPv6/IPv4 node , has the ability to send and receive both IPv4 and IPv6 datagrams.

When interoperating with an IPv4 node, an IPv6/IPv4 node can use IPv4 datagrams; when interoperating with an IPv6 node, it can speak IPv6.

IPv6/IPv4 nodes must have both IPv6 and IPv4 addresses. They will be able to determine whether the node is IPv6-capable or IPv4-only.

In the dual-stack approach, if either the sender or the receiver is only IPv4- capable, an IPv4 datagram must be used.

It is possible that two IPv6-capable nodes can end up, sending IPv4 datagrams to each other. This is illustrated in Figure below.



Suppose Node A is IPv6-capable and wants to send an IP datagram to Node F, which is also IPv6-capable. Nodes A and B can exchange an IPv6 datagram.

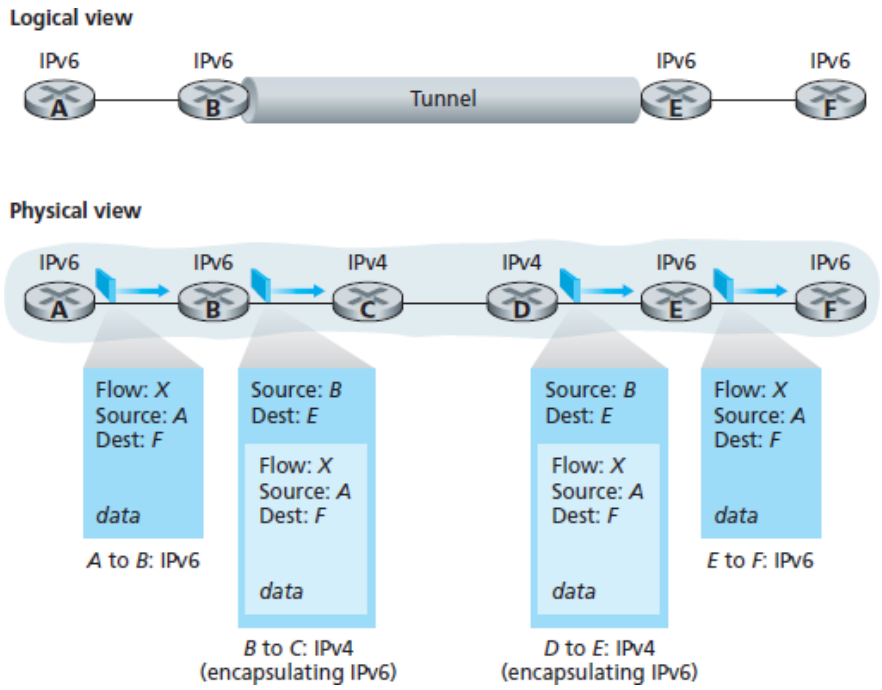
Node B must create an IPv4 datagram to send to C. Certainly, the data field of the IPv6 datagram can be copied into the data field of the IPv4 datagram and appropriate address mapping can be done.

In performing the conversion from IPv6 to IPv4, there will be IPv6-specific fields in the IPv6 datagram that have no counterpart in IPv4.

The information in these fields will be lost. Thus, even though E and F can exchange IPv6 datagrams, the arriving IPv4 datagrams at E from D do not contain all of the fields that were in the original IPv6 datagram sent from A.

An alternative to the dual-stack approach, is known as **tunneling**.

Tunneling can solve the problem noted above, allowing, for example, E to receive the IPv6 datagram originated by A. The basic idea behind tunneling is the following.



Suppose two IPv6 nodes (for example, B and E in Figure above) want to interoperate using IPv6 datagrams but are connected to each other by intervening IPv4 routers.

The intervening set of IPv4 routers between two IPv6 routers is referred as a **tunnel**, as illustrated in Figure above.

With tunneling, the IPv6 node on the sending side of the tunnel (for example, B) takes the *entire* IPv6 datagram and puts it in the data (payload) field of an IPv4 datagram.

This IPv4 datagram is then addressed to the IPv6 node on the receiving side of the tunnel (for example, E) and sent to the first node in the tunnel (for example, C).

The intervening IPv4 routers in the tunnel route this IPv4 datagram among themselves, unaware that the IPv4 datagram itself contains a complete IPv6 datagram.

The IPv6 node on the receiving side of the tunnel eventually receives the IPv4 datagram, determines that the IPv4 datagram contains an IPv6 datagram, extracts the IPv6 datagram, and then routes the IPv6 datagram.

IP Security

IPsec, is a popular secure network-layer protocols .

IPsec has been designed to be backward compatible with IPv4 and IPv6. If two hosts want to securely communicate, IPsec needs to be available only in those two hosts.

On the sending side, the transport layer passes a segment to IPsec. IPsec then encrypts the segment, appends additional security fields to the segment, and encapsulates the resulting payload in an ordinary IP datagram.

The sending host then sends the datagram into the Internet, which transports it to the destination host. There, IPsec decrypts the segment and passes the unencrypted segment to the transport layer.

The services provided by an IPsec session include:

• *Cryptographic agreement*. Mechanisms that allow the two communicating hosts to agree on cryptographic algorithms and keys.

• *Encryption of IP datagram payloads*. When the sending host receives a segment from the transport layer, IPsec encrypts the payload. The payload can only be decrypted by IPsec in the receiving host.

• *Data integrity*. IPsec allows the receiving host to verify that the datagram's header fields and encrypted payload were not modified while the datagram was in route from source to destination.

• *Origin authentication*. When a host receives an IPsec datagram from a trusted source, the host is assured that the source IP address in the datagram is the actual source of the datagram.

4.5 Routing Algorithms

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 purpose of a routing algorithm is : given a set of routers, with links connecting the routers, a routing algorithm finds a "good" path i.e least cost path from source router to destination router.

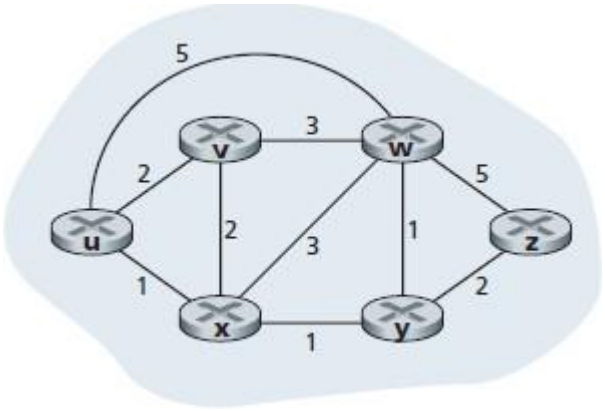
A **graph** $G = (N,E)$ is a set N of nodes and a collection E of edges, where each edge is a pair of nodes from N.

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.

An edge's cost reflects the physical length of the corresponding link .

Consider figure below , 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 $c(x,y) = \infty$.

Only undirected graphs (i.e., graphs whose edges do not have a direction) are considered, 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.



A path in a graph $G = (N,E)$ is a sequence of nodes $(x1, x2,..., xp)$ such that each of the pairs $(x1,x2)$, $(x2,x3),...,(xp\text{-}1,xp)$ are edges in E.

The cost of a path $(x1,x2,..., xp)$ is the sum of all the edge costs along the path, that is, $c(x1,x2) + c(x2,x3) + ... + c(xp\text{-}1,xp)$.

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**. 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.

Classification of routing algorithms :

Routing algorithms can be classified 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.

The algorithm takes the connectivity between all nodes and all link costs as inputs.

Algorithms with global state information are referred to as **link-state** (**LS**) **algorithms**, since the algorithm must be aware of the cost of each link in the network.

- 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.

  Each node begins with only the knowledge of the costs of its own directly attached links.

  Through an iterative process of calculation and exchange of information with its neighboring nodes, a node gradually calculates the least-cost path to a destination or set of destinations.

  The decentralized routing algorithm 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.

Routing algorithms is classified 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.
- **Dynamic routing algorithms** change the routing paths as the network traffic loads or topology change.

Routing algorithms is classified 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 isassociated with a link that is currently congested, a routing algorithm will tend to choose routes around such a congested link.
- Internet routing algorithms (such as RIP, OSPF, and BGP) are **load-insensitive**, as a link's cost does not explicitly reflect its current level of congestion.

The Link-State (LS) Routing Algorithm

Link costs are available as input to the LS algorithm. 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.

The link-state routing algorithm is known as ***Dijkstra's algorithm.***

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.

The following notations are defined:

• $D(v)$: cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.

• $p(v)$: previous node (neighbor of v) along the current least-cost path from the source to v.

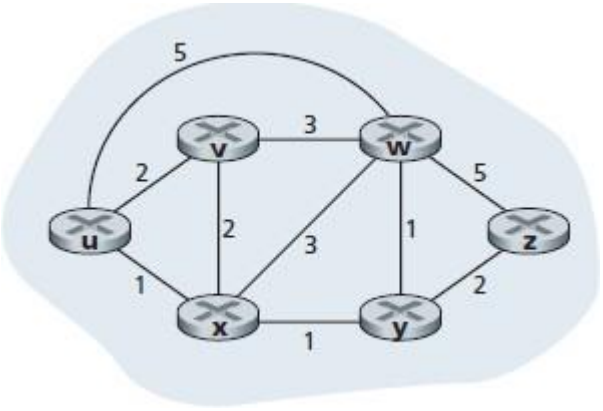
• $N\_$: subset of nodes; v is in $N\_$ if the least-cost path from the source to v is definitively known.

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) = ∞
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
```



Consider the network in Figure above and compute the least-cost paths from *u* to all possible destinations.

A tabular summary of the algorithm's computation is shown in Table below, where each line in the table gives the values of the algorithm's variables at the end of the iteration.

Consider the following few first steps:

• 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.

The cost to *w* is set to 5 since this is the cost of the direct 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, consider the nodes that are 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. 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*_ so that *N*_ now contains *u*, *x*, and *y*. The cost to the remaining nodes not yet in *N*_, 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.

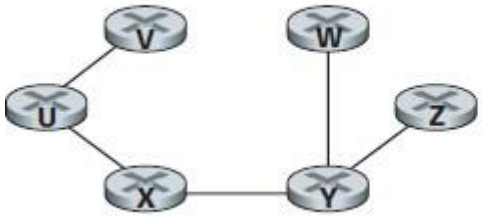
• And so on. . . .

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

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					

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 below shows the resulting least-cost paths and forwarding table in *u* for the network shown in above figure .



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

Computational Complexity : In the first iteration, we need to search through all *n* nodes to determine the node, *w*, not in *N*_ 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 searched through over all the iterations is $n(n + 1)/2$, and thus the preceding implementation of the LS algorithm has worst-case complexity of order *n* squared: $O(n2)$.

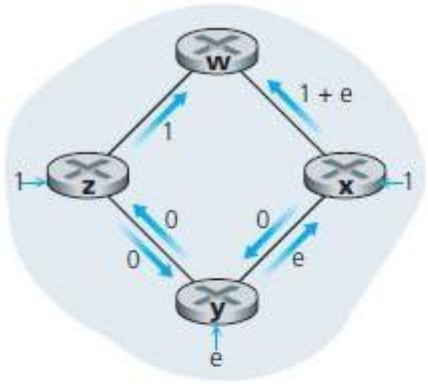
Figure below shows a simple network topology where link costs are equal to the load carried on the link. For example, reflecting the delay that would be experienced. In this example, link costs are not symmetric; that is, $c(u,v)$ equals $c(v,u)$ only if the load carried on both directions on the link (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 (a) with the link costs corresponding to the amount of traffic carried.

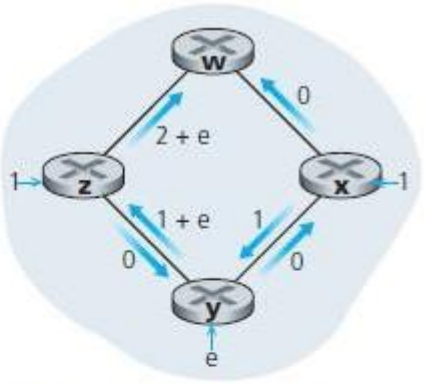
When the LS algorithm is next run, node *y* determines (based on the link costs shown in Figure (a)) that the clockwise path to *w* has a cost of 1, while the counter clockwise path to *w* (which it had been using) has a cost of $1 + e$.

Hence *y*'s 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 (b).

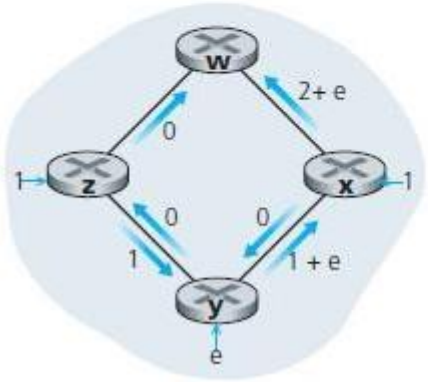
When the LS algorithm is run next, nodes *x*, *y*, and *z* all detect a zero-cost path to *w* in the counter clockwise direction, and all route their traffic to the counter clockwise routes. The next time the LS algorithm is run, *x*, *y*, and *z* all then route their traffic to the clockwise routes.



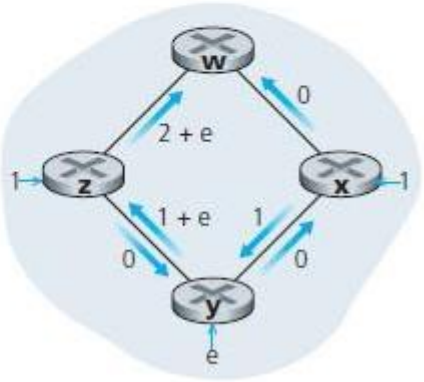
a. Initial routing



b. *x*, *y* detect better path to *w*, clockwise



c. *x*, *y*, *z* detect better path to *w*, counterclockwise



d. *x*, *y*, *z*, detect better path to *w*, clockwise

Solutions to above problem :

- Mandate that link costs not depend on the amount of traffic carried—an unacceptable solution since one goal of routing is to avoid highly congested links.
- Ensure that not all routers run the LS algorithm at the same time.

The Distance-Vector (DV) Routing Algorithm

Distancevector (**DV**) algorithm is iterative, asynchronous, and distributed.

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 i.e* process continues until no more information is exchanged between neighbors.

The algorithm is *asynchronous* i.e it does not require all of the nodes to operate in lockstep with each other.

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 Bellman-Ford equation, namely,

$d_x(y) = \min_v\{c(x,v) + d_v(y)\}$

where the *minv* in the equation is taken over all of *x*'s neighbors.

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*.

Evaluate for source node *u* and destination node *z* in Figure . The source node *u has three neighbors: nodes v, x, and w.*

dv(z) = 5, dx(z) = 3, and dw(z) = 3.

Substitute these values into Equation above, along with the costs c(u,v) = 2, c(u,x) = 1, and c(u,w) = 5, gives *:*

du(z) = min{2 + 5, 5 + 3, 1 + 3} = 4 ;

The basic idea is as follows:

Each node x begins with Dx(y), an estimate of cost of the least-cost path from itself to node y, for all nodes in N.

Let **D**x = [Dx(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 = [Dx(y): y in N], containing x's estimate of its cost to all destinations, y, in N

• The distance vectors of each of its neighbors, that is, **D**v = [Dv(y): y in N] for each neighbor v of x.

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:

$D_x(y) \_ \min_v\{c(x,v) + D_v(y)\}$ 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 update their own distance vectors.

Distance-Vector (DV) Algorithm

At each node, *x*:

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

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* 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*.

The next-hop router $v^*(y)$ is the neighbour *v* that achieves the minimum in Line 14 of the DV algorithm.

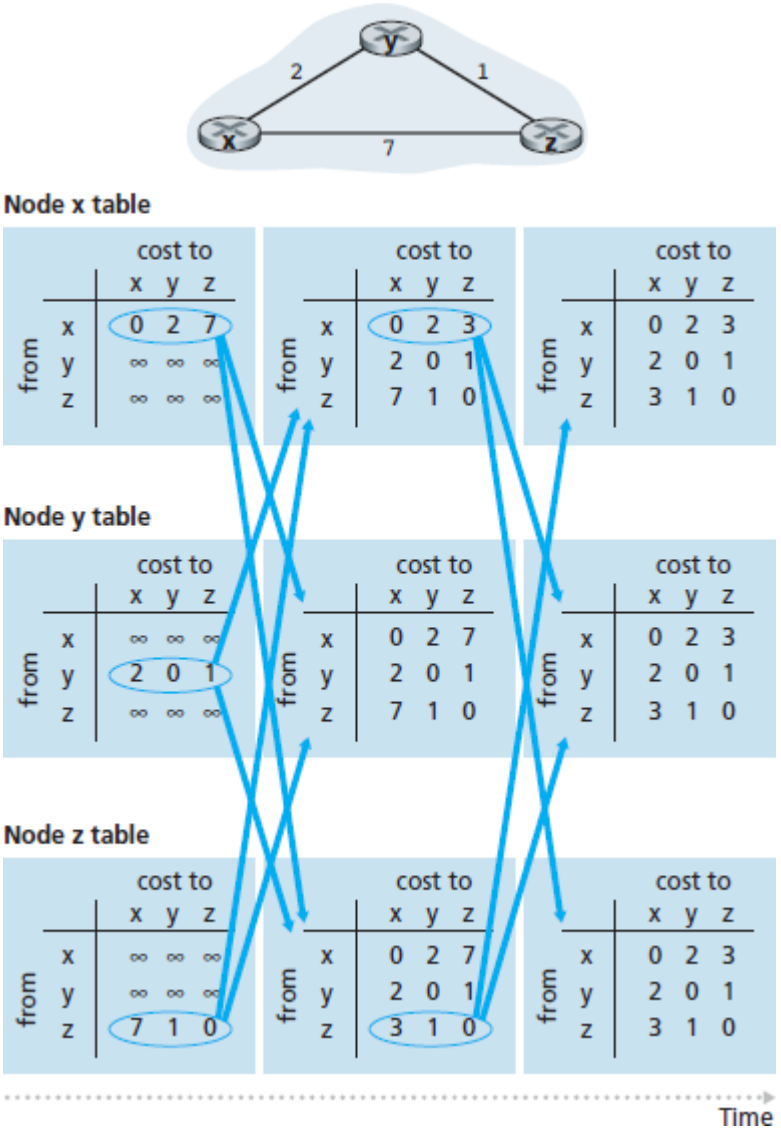
Thus, in Lines 13–14, for each destination *y*, node *x* also determines $v^*(y)$ and updates its forwarding table for destination *y*.

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.

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).

Figure below 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.



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 $\mathbf{D}x = [Dx(x), Dx(y), Dx(z)] = [0, 2, 7]$.

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 above by the arrows from the first column of tables to the second column of tables. For example, node *x* sends its distance vector $\mathbf{D}x = [0, 2, 7]$ to both nodes *y* and *z*. After receiving the updates, each node recomputes its own distance vector.

For example, node *x* computes

$Dx(x) = 0$

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

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

The second column therefore displays, for each node, the node's new distance vector along with distance vectors just received from its neighbors.

For example, that node *x*'s estimate for the least cost to node *z*, $Dx(z)$, has changed from 7 to 3.

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 above by the arrows from the second column of tables to the third column of tables.

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.

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 (a) below illustrates a scenario where the link cost from *y* to *x* changes from 4 to 1.

Focus is 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.

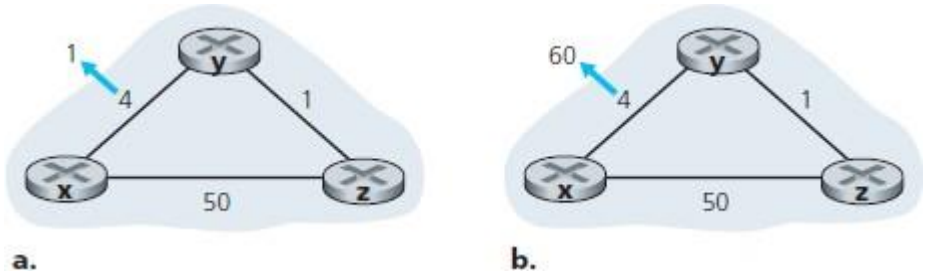
• At time *t*2, *y* receives *z*'s update and updates its distance table. *y*'s least costs do not change and hence *y* does not send any message to *z*. The algorithm comes to a quiescent state.

Thus, only two iterations are required for the DV algorithm to reach a quiescent state.

Consider *increase in* link cost. Suppose that the link cost between *x* and *y* increases from 4 to 60, as shown in Figure (b) below.

1. Before the link cost changes, $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, and $D_z(x) = 5$. 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

$D_y(x) = \min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\} = \min\{60 + 0, 1 + 5\} = 6.$



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, a **routing loop is -- i**n 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

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. 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 $Dy(x) = 8$ and sends *z* its distance vector. *z* then determines $Dz(x) = 9$ and sends *y* its distance vector, and so on.

Distance-Vector Algorithm: Adding Poisoned Reverse

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 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*.

Poisoned reverse solves the particular looping problem encountered before in Figure (b) above. As a result of the poisoned reverse, *y*'s distance table indicates $Dz(x) = \infty$.

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, $D_y(x) = 60$.

After receiving the 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 $D_y(x) = 51$. 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 $D_y(x) = \infty$ (even though *y* knows that $D_y(x) = 51$ in truth).

A Comparison of LS and DV Routing Algorithms

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.

Differences between LS and DV algorithm :

N is the set of nodes (routers) and *E* is the set of edges (links).

• ***Message complexity***. LS requires each node to know the cost of each link in the network. This requires O(|N| |E|) messages to be sent.

If 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.

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***. *Complexity* of LS is O(|N|2). 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***. 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 separated under **LS**, providing a degree of **robustness**. Under DV, a node can advertise incorrect least-cost paths to any or all destinations.

Hierarchical Routing

One router is indistinguishable from another i.e all routers executes the same routing algorithm to compute routing paths through the entire network.

In practice, this model and its view of a homogenous set of routers all executing the same routing algorithm is simple for two important reasons:

• ***Scale***. As the number of routers becomes large, the overhead involved in computing, storing and communicating routing information is prohibitive.

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!

• ***Administrative autonomy.*** 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 the above problems can be solved by organizing routers into **autonomous systems** (**ASs**), with each AS consisting of a group of routers that are typically under the same administrative control .

Routers within the same AS all run the same routing algorithm (for example, an LS or DV algorithm) and have information about each other.

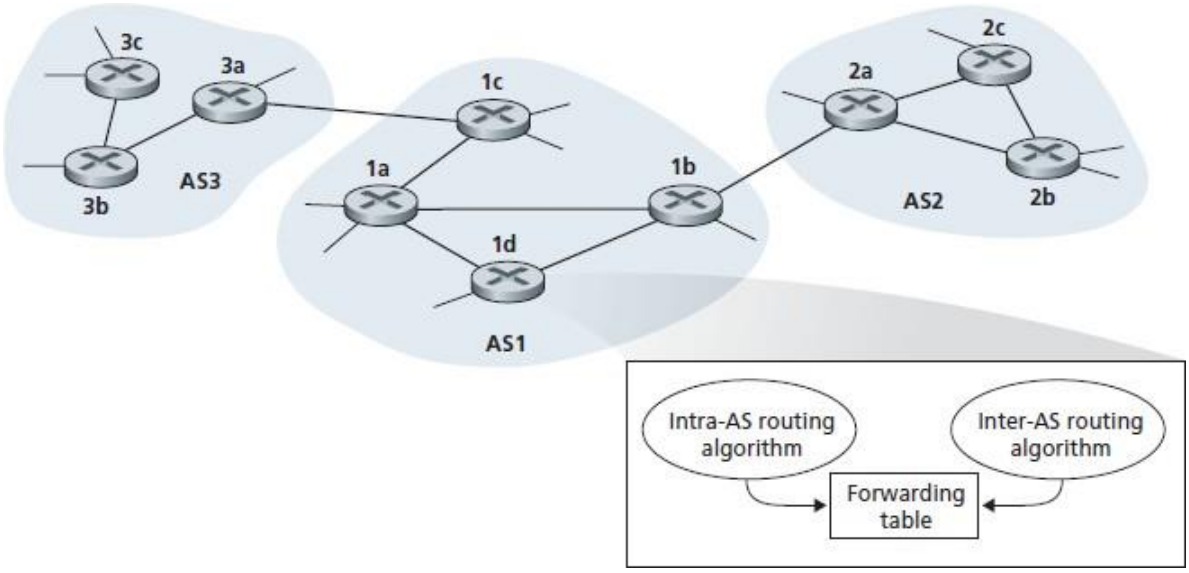
The routing algorithm running within an autonomous system is called an **intraautonomous**

system routing protocol. It is necessary, to connect ASs 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 below provides a simple example with three ASs: 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. Intra-AS routing protocols running in AS1, AS2, and AS3 need not be the same. The routers 1b, 1c, 2a, and 3a are all gateway routers.



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 above 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.

AS1 needs :

(1) to learn which destinations are reachable via AS2 and which destinations are reachable via AS3.

(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 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.

Consider a subnet *x* and suppose that AS1 learns from the inter-AS routing protocol that subnet *x* is reachable from AS3 but is *not* reachable from AS2.

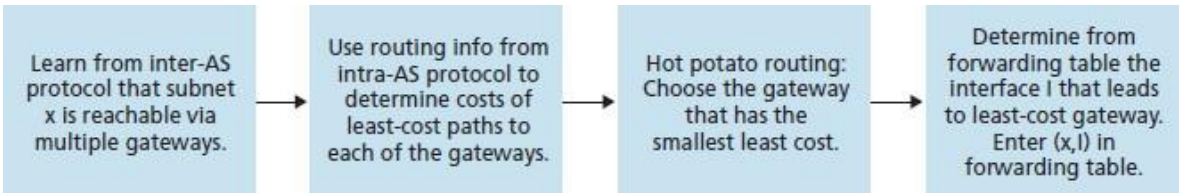
AS1 then propagates this information to all of its routers. When router 1d learns that subnet *x* is reachable from AS3, and hence from gateway 1c, it then determines, from the information provided by the intra-AS routing protocol, the router interface that is on the least-cost path from router 1d to gateway router 1c. Say this is interface *I*. The router 1d can then put the entry (*x*, *I*) into its forwarding table.

Hot Potato Routing :

In hot-potato routing, the AS gets rid of the packet (the hot potato) as quickly 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.

Eg : 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 below 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.

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. 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 intra-AS routing protocol it chooses; Each pair of connected ASs needs to run the same inter-AS routing protocol to exchange reachability information.

Routing in the Internet

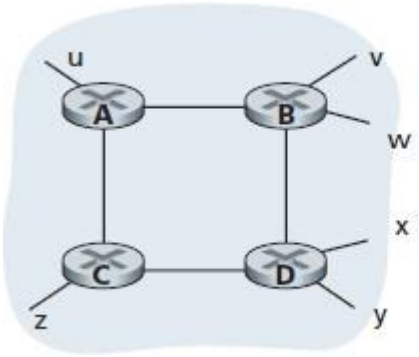
Intra-AS Routing in the Internet: RIP

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**.

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)**.

RIP is a distance-vector protocol that operates in a manner very close to the idealized DV protocol. In RIP (and also in OSPF), costs are 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 below 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.



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

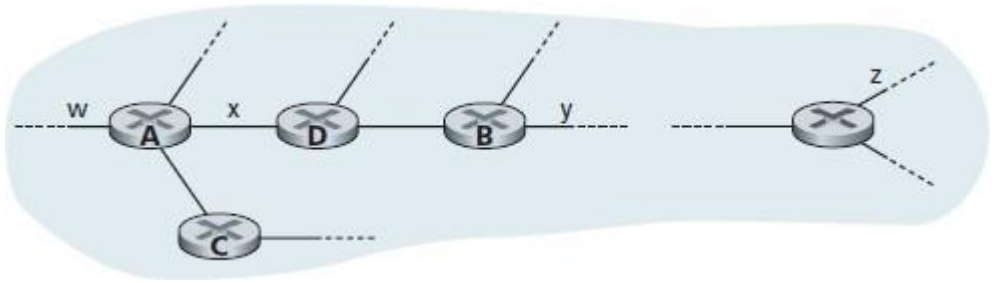
The maximum cost of a path is limited to 15, thus limiting the use of RIP to autonomous systems that are fewer than 15 hops in diameter.

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**.

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, and z) are labeled.* Dotted lines indicate that the AS continues on;



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 below shows the routing table for router *D.*

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,
- The third column indicates the number of hops 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.* A routing table will have one row for each subnet in the AS.

Advertisement from D :

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

Suppose that 30 seconds later, router *D* receives from router *A* the advertisement shown in Figure below. This advertisement is the routing table information from router *A*!

This information indicates, in particular, that subnet *z* is only four hops away from router *A.* Router *D,* upon receiving this advertisement, merges the advertisement (Figure below) with the old routing table (Figure above).

In particular, router *D* learns that there is now a path through router *A* to subnet *z* that is shorter than the path through router *B.* Thus, router *D* updates its routing table to account for the shorter shortest path, as shown in Figure below.

Advertisement from A :

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

RIP routers exchange advertisements approximately every 30 seconds. If a router does not hear from its neighbor at least once every 180 seconds, that neighbor is considered to be no longer reachable;

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

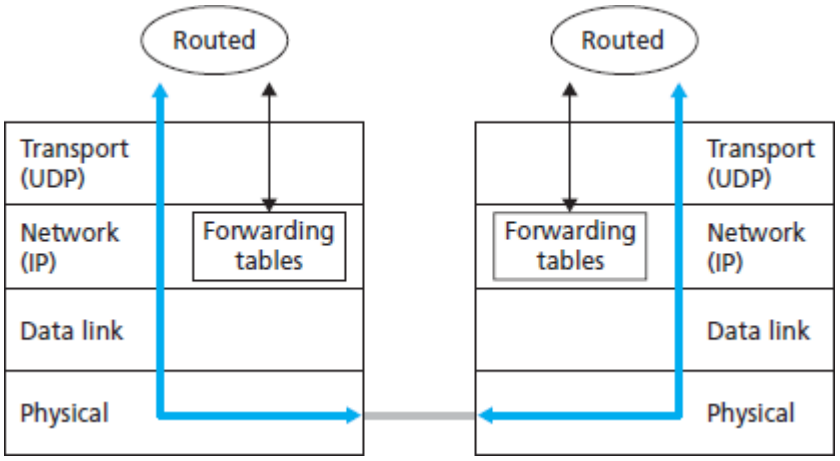
Routing table in router *D* after receiving advertisement from router *A*

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.

RIP uses a transport-layer protocol (UDP) on top of a network layer protocol (IP) to implement network-layer functionality (a routing algorithm).

Figure below shows RIP implementation in a UNIX system, for example, a UNIX workstation serving as a router. A process called *routed* executes RIP, that is, maintains routing information and exchanges messages with *routed* processes running in neighboring routers.



Intra-AS Routing in the Internet: OSPF

OSPF routing is widely used for intra-AS routing in the Internet.

The Open in OSPF indicates that the routing protocol specification is publicly available.

The most recent version of OSPF,version 2.

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. 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.

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. It also broadcasts a link's state periodically (at least once every 30 minutes), even if the link's state has not changed.

OSPF advertisements are contained in OSPF messages that are carried directly by IP

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 in the OSPF protocol within an AS, thus preventing malicious intruders 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.

Simple authentication: The same password is configured on each router. When a router sends an OSPF packet, it includes the password in plaintext.

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.

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.

• ***Integrated support for unicast and multicast routing***. Multicast OSPF (MOSPF) provides extensions to OSPF to provide for multicast routing

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***. *T*he most significant advance in OSPF is the ability to structure an autonomous system hierarchically.

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 non border 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.

Inter-AS Routing: BGP

The **Border Gateway Protocol** version 4, is the standard inter-AS routing protocol. It is referred to as BGP4 or simply as **BGP**. As an inter-AS routing protocol 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.

BGP allows each subnet to advertise its existence to the rest of the Internet.

BGP Basics

In BGP, pairs of routers exchange routing information over semipermanent TCP connections using port 179. The semi-permanent TCP connections for the network in graph(refer fig 1 in hierarchical routing) are shown in Figure below.

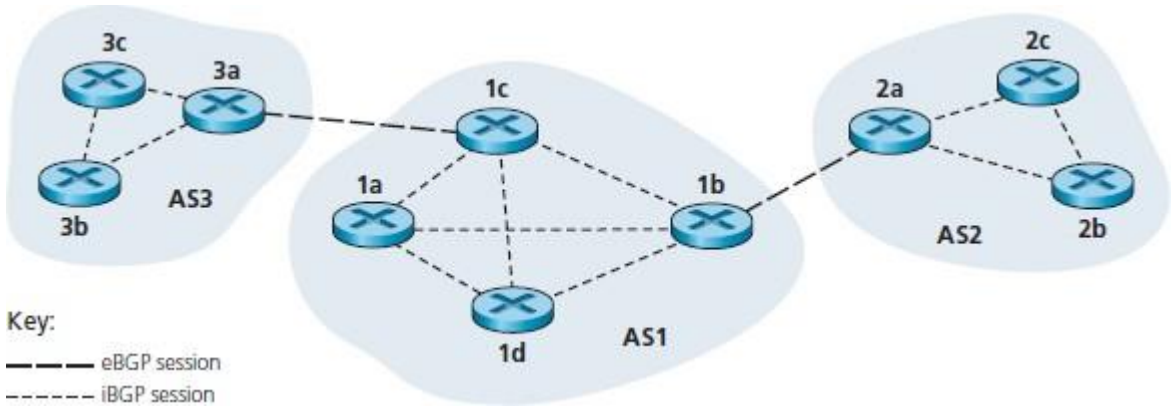
There is one such BGP TCP connection for each link that directly connects two routers in two different ASs;

Thus, in Figure below, 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.

Figure below 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 connection is called a **BGP session**.

Furthermore, a BGP session that spans two Ass 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 below, the eBGP sessions are shown with the long dashes; the iBGP sessions are shown with the short dashes.



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.

Suppose that only the first three of those four subnets are in AS2 and the fourth subnet, 138.16.67/24, is in AS3.

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. 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.

Path Attributes and BGP Routes

In BGP, an autonomous system is identified by its globally unique **autonomous system number** (**ASN**) .

When a router advertises a prefix across a BGP session, it includes with the prefix a number of **BGP attributes**.

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 ASPATH attribute.

For example, consider Figure above 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.

• **NEXT- HOP** : Providing the critical link between the inter-AS and intra-AS routing protocols, the NEXT-HOP attribute is of important use. *The NEXT-HOP is the router interface that begins the AS-PATH.*

Refer above Figure. Consider the gateway router 3a in AS3 when advertises a route to gateway router 1c in AS1 using eBGP. The route includes the advertised prefix, say *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.

Consider 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.

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.

Intra-AS routing algorithm has determined the least-cost path to all subnets attached to the routers in AS1, including to the subnet for the link between 1c and 3a.

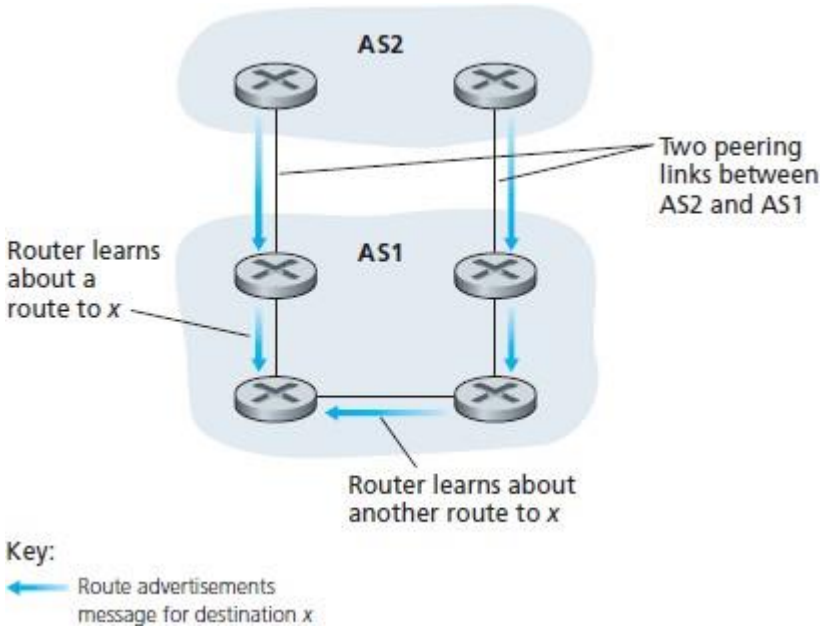
From this least-cost path from 1d to the 1c-3a subnet, 1d determines its router interface *l* that begins this path and then adds the entry (*x*, *l*) to its forwarding table.

Thus, NEXT-HOP attribute is used by routers to configure their forwarding tables.

• Figure below 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 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 to determine the appropriate interface.



BGP Route Selection

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 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. 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. This process is called hot-potato routing.

• If more than one route still remains, the router uses BGP identifiers to select the route;

Routing Policy

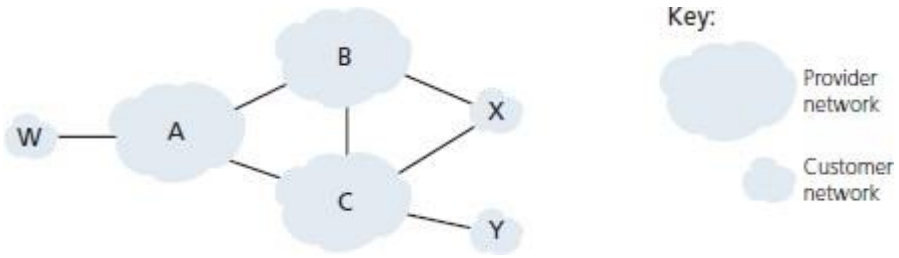


Figure above shows six interconnected autonomous systems: A, B, C, W, X, and Y. It is important to note that A, B, C, W, X, and Y are ASs, not routers.

Assume that autonomous systems W, X, and Yare stub networks and that A, B, and C are backbone provider networks. 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 **multihomed stub network,** since it is connected to the rest of the network via two different providers .

However, like Wand Y, X itself must be the source/destination of all traffic leaving/entering X.

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.

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.

Consider 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 if B advertise the path BAW to C 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.

4.7 Broadcast and Multicast Routing

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.

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).

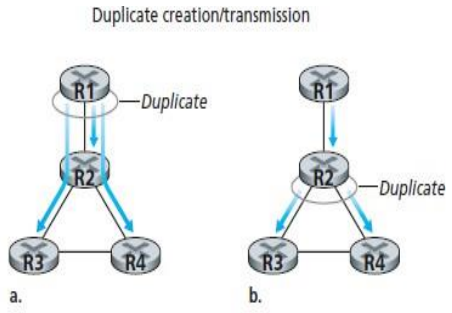


Figure 4.43 ♦ Source-duplication versus in-network duplication

- 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.
- 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. 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 to rely on the unicast routing infrastructure to achieve broadcast.

Uncontrolled Flooding

The most noticeable 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 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 .
    - 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 counter clockwise.
    - 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.

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.

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 neighbours.
  - 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 packet is duplicated and forwarded to all the node's neighbours
  - The Gnutella protocol, uses sequence-number-controlled flooding to broadcast queries in its overlay network.
- A second approach to controlled flooding is known as **reverse path forwarding (RPF)** [Dalal 1978], also sometimes referred to as **reverse path broadcast (RPB).**
  - 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.
  - 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).

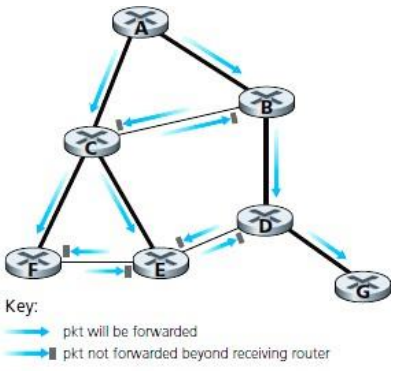


Figure 4.44 ♦ Reverse path forwarding

    - 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.

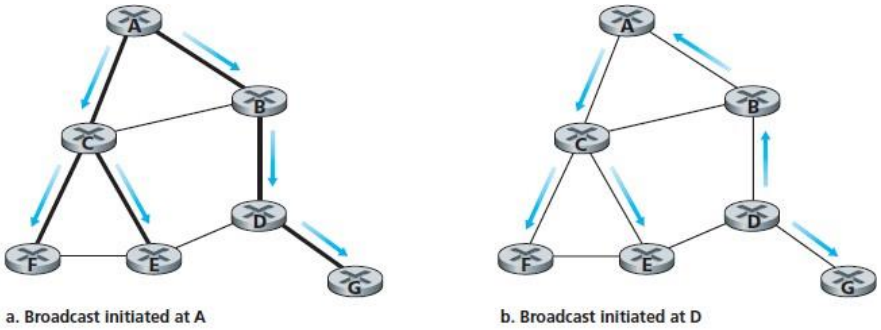
Spanning-Tree Broadcast

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.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` = (N,E`) such that E` is a subset of E, G` is connected, G` contains no cycles, and G` 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 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].
    - 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.



a. Broadcast initiated at A b. Broadcast initiated at D

Figure 4.45 ♦ Broadcast along a 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 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 16bit 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 TTLfield 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.

4.7.2 Multicast

Multicast service is where 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
  - 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
- 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.

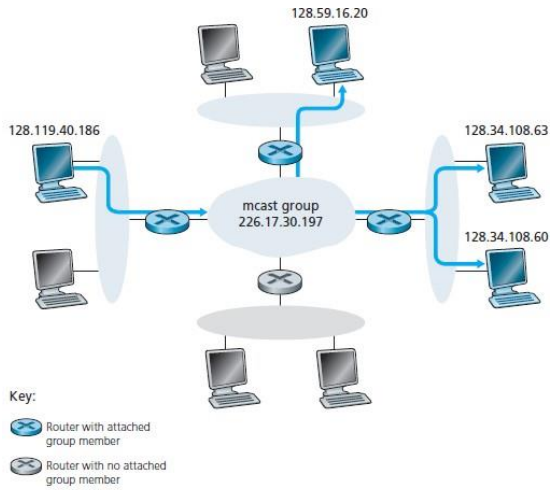


Figure 4.47 ♦ The multicast group: A datagram addressed to the group is delivered to all members of the multicast group

- 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.

Internet Group Management Protocol

The IGMP protocol version 3 [RFC 3376] operates between a host and its directly attached router as shown in Figure 4.48.

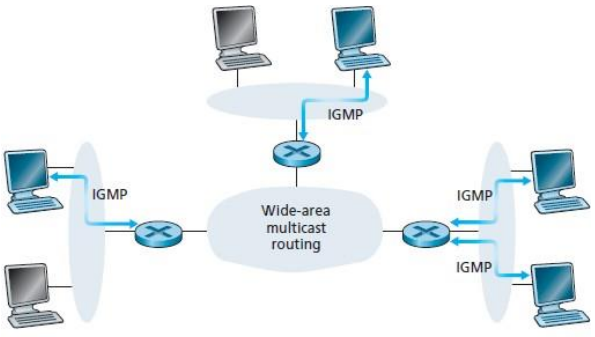


Figure 4.48 ♦ The two components of network-layer multicast in the Internet: IGMP and multicast routing protocols

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 the Internet, so that multicast datagrams are routed to their final destinations.

- This latter functionality is accomplished by network-layer multicast routing algorithms.
- 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. 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 softstate 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 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
- It has been argued that soft-state protocols result in simpler control than hardstate 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.

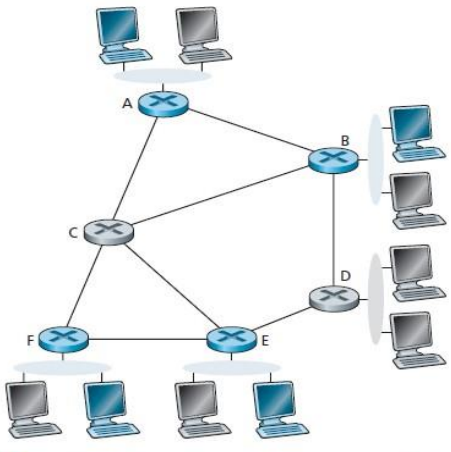


Figure 4.49 ♦ Multicast hosts, their attached routers, and other routers

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. Two approaches have been adopted for determining the multicast routing tree. 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 (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.
  - Acritical question for center-based tree multicast routing is the process used to select the center.

- **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.
  - 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.

- 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).

- 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.
- 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 intradomain 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.

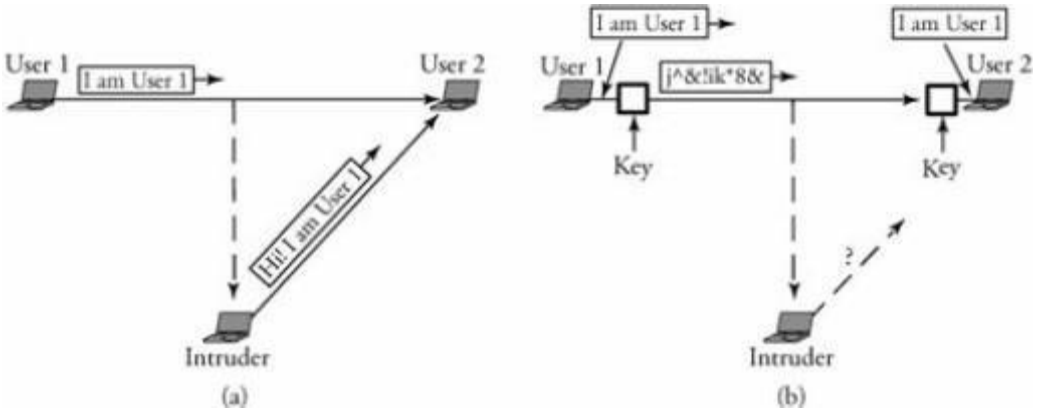
Network Security

Overview of Network Security

- Network security is required by the users to communicate on the network.
- If medium is insecure then an intruder may intercept, read and modify the transmitted-data from sender to receiver.

Elements of Network Security

1) **Confidentiality:** Information should be available only to those who have rightful access to it
2) **Authenticity and integrity:** The sender of a message and the message itself should be verified at the receiving-point



(a) Message content and sender identity falsified by intruder; (b) a method of applied security

- In figure a, user 1 sends a message ("i am user 1") to user 2. Since the network lacks any security system, an intruder can receive the message and change its content to a different message ("hi i am user 1") and send it to user 2. User 2 may not know that this falsified message is really from user 1(authentication).
- In figure 10.1b, a security block is added to each side of the communication, and a secret key that only users 1 and 2 would know about is included. Therefore, the message is changed to a form that cannot be altered by the intruder.

Threats To Network Security

Internet infrastructure attacks are broadly classified into 4 categories

1) DNS hacking

2) Routing table poisoning
3) Packet mistreatment
4) Denial of Service (DOS)

DNS HACKING ATTACKS

- DNS server is a distributed hierarchical and global directory that translates domain names into numerical IP address.
- DNS is a critical infrastructure, and all hosts contact DNS to access servers and start connections.
- Name-resolution services in the modern Internet environment are essential for email transmission, navigation to web sites, or data transfer. Thus, an attack on DNS can potentially affect a large portion of the Internet.
- A DNS hacking attack can appear in any of the following forms
  1) **Masquerading Attack:** The attacker poses as a trusted entity and obtains all the secret information. The attacker can stop any message from being transmitted further or can change the content or redirect the packet to bogus servers. This action is also known as a middle-man attack.
  2) **Domain Highjacking Attack:** Whenever a user enters a domain address, he is forced to enter into the attacker's Web site.
  3) **Information Leakage Attack:** The attacker sends a query to all hosts identifies which IP addresses are not used and uses those IP address to make other types of attacks
  4) **Information-Level Attack(Cache Poisoning):** This forces a server to correspond with other than the correct answer. The hacker tricks a remote name-servers into caching the answer for a third-party domain by providing malicious information and redirects traffic to a preselected site.

ROUTING TABLE POISONING

- This is the undesired modification of routing tables. This results in a lower throughput of the network.
- Two types of attacks are: i) link attack and ii)router attack.

Link Attack

- Link attack occurs when a hacker gets access to a link and thereby intercepts, interrupts or modifies routing messages. This act similarly on both the link-state and the distance-vector protocols.
- If an attacker succeeds in placing an attack in a link-state routing protocol, a router may send incorrect updates about its neighbors or remain silent even if the link state of its neighbor has changed

Router Attack

- Router Attack may affect the link-state protocol or even the distance-vector protocol.
- In link-state protocol, if routers are attacked, they become malicious. As a result, routers may add a non existing link to a routing table delete an existing link or change the cost of a link.
- In the distance-vector protocol, an attacker may cause routers to send wrong updates about any node in the network, thereby misleading a router and resulting in network problems.

PACKET MISTREATMENT ATTACKS

- Packet mistreatment attacks can occur during any data transmission.
- A hacker may capture certain data packets and mistreat them.
- The attack may result in congestion lowering throughput & DOS attacks
- Link-attack causes interruption, modification or replication of data packets. Whereas, a router-attack can misroute all packets and may result in congestion or DOS

Following are some examples:

1) **Interruption:** If an attacker intercepts packets, they may not be allowed to be propagated to their destinations.
2) **Modification:** Attackers may succeed in accessing the content of a packet. They can then change the address of the packet or change the data of the packet. This kind of attack can be detected by digital signature mechanism.
3) **Replication**: An attacker may trap a packet and duplicate it. This kind of attack can be detected by using the sequence number for each packet.
4) **Malicious Misrouting of Packets**: A hacker may attack a router and change its routing table, resulting in misrouting of data packets.

5) **Ping of death:** An attacker may send a ping message, which is large and therefore must be fragmented for transport. The receiver then starts to reassemble the fragments as the ping fragments arrive. The total packet length becomes too large and might cause a system crash.

DOS ATTACKS (DENIAL OF SERVICE)

- DOS is a type of security breach that prohibits a user from accessing normally provided services.
- DOS can cost the target person a large amount of time and money.
- DOSaffects the destination rather than a data-packet or router.
- They take important servers out of action for few hours, thereby denying service to all users.

Two types of attacks are:

1) *Single-source:* An attacker sends a large number of packets to a target system to overwhelm & disable it
2) *Distributed:* A large number of hosts are used to flood unwanted traffic to a single target. The target cannot then be accessible to other users in the network.

Overview of Security Methods

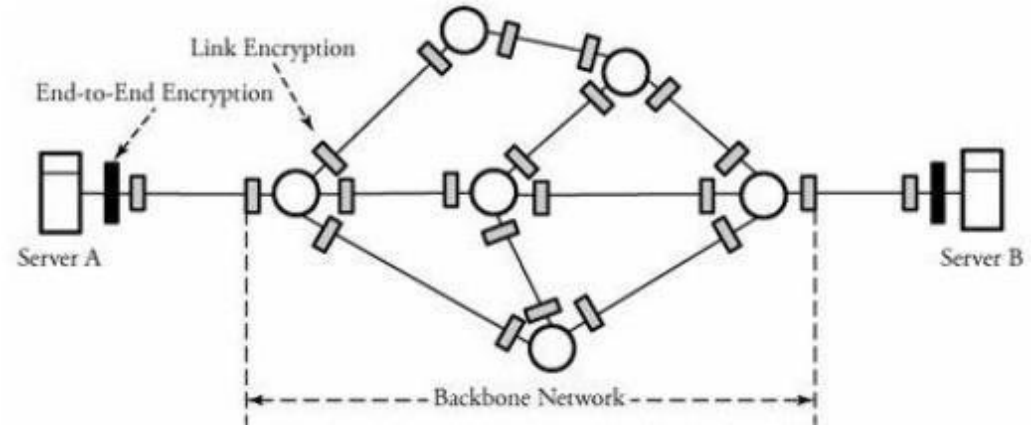
Common solutions that can protect computer communication networks from attacks are classified are cryptographic techniques or authentication techniques(verification).

Cryptographic Techniques

- Cryptography is the process of transforming a piece of information or message shared by two parties into some sort of code.
- The message is scrambled before transmission so that it is undetectable by outside watchers.
- The scrambled-message needs to be decoded at the receiving-end before any further processing.
- The main tool used to encrypt a message M is a secret-key K.
- The fundamental operation used to encrypt a message is the exclusive-OR($\oplus$).
- Assume that we have one-bit M and a secret-bit K. A simple encryption is carried out using $M \oplus K$.

- To decrypt this message, the second party can detect M by performing the following operation: (M ⊕ K) ⊕ K = M
- In *end-to-end encryption*, secret coding is carried out at both end systems. In *link encryption*, all the traffic passing over that link is secured.
- Two types of encryption techniques are secret-key & public-key encryption
  1) In *secret-key model*, both sender & receiver conventionally use same key for an encryption process.
  2) In *public-key model*, a sender and a receiver each use a different key.
- The public-key system is more powerful than the secret key system & provides better security and message privacy.

  Drawbacks of public-key system: slow speed and more complex computationally



Authentication Techniques

Encryption methods offer the assurance of message confidentiality. A networking-system must be able to verify the authenticity of the message and the sender of the message. These forms of security techniques are known as authentication techniques.

Authentication techniques are categorized as

i) authentication with message digest

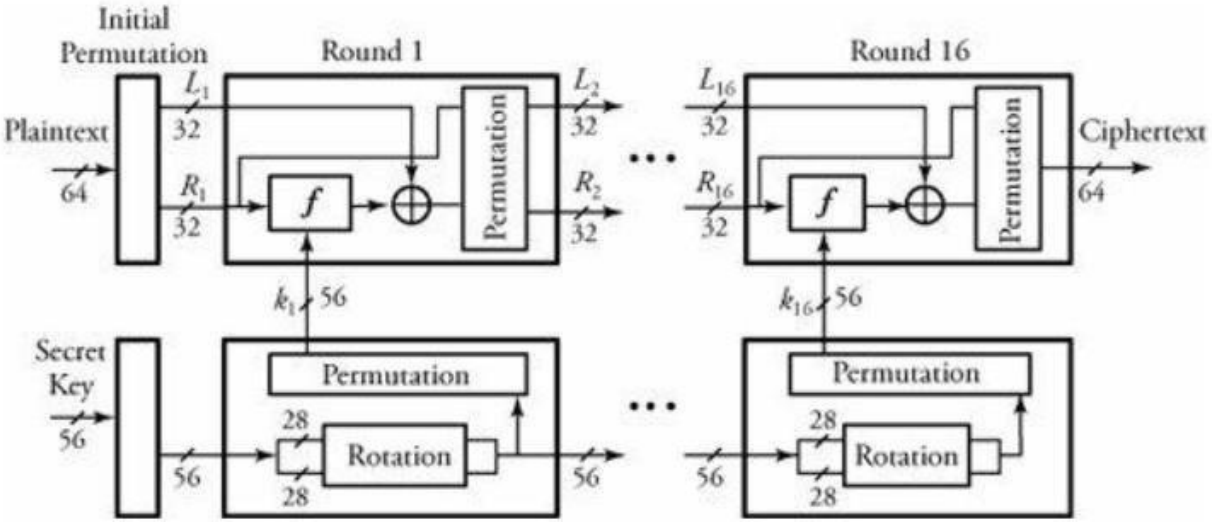
ii) authentication with digital signature.

Secret Key Encryption Protocols

- This is also called as symmetric encryption or single-key encryption.
- Sender and receiver conventionally use the same key for an encryption process.
- This consist of an encryption-algorithm, a key and a decryption-algorithm
- The encrypted-message is called **cipher text.**
- Two popular protocols are:
  1) DES (Data Encryption Standard)
  2) AES (Advanced Encryption Standard)
- A shared secret-key between a transmitter and a receiver is assigned at the transmitter and receiver points.
- At the receiving end, the encrypted information can be transformed back to the original data by using decryption algorithm and secret key.

DES (Data Encryption Standard)

- Plaintext messages are converted into 64-bit blocks & each block is encrypted using a key.
- The key length is 56 bits.
- DES consists of 16 identical rounds of an operation.



Begin DES Algorithm

1) Initialize. Before round 1 begins, all 64 bits of the message and all 56 bits of the secret key are separately permuted (shuffled).

2) Each incoming 64-bit message is broken into two 32-bit halves denoted by Li and Ri respectively.
3) The 56 bits of the key are also broken into two 28-halves, and each half is rotated one or two bit positions, depending on the round.
4) All 56 bits of the key are permuted, producing version ki of the key on round i.
5) Li and Ri are determined by

   $L_i = R_{i-1}$

   and

   $R_i = L_{i-1} \oplus F(R_{i-1}, k_i)$

6) All 64 bits of a message are permuted.

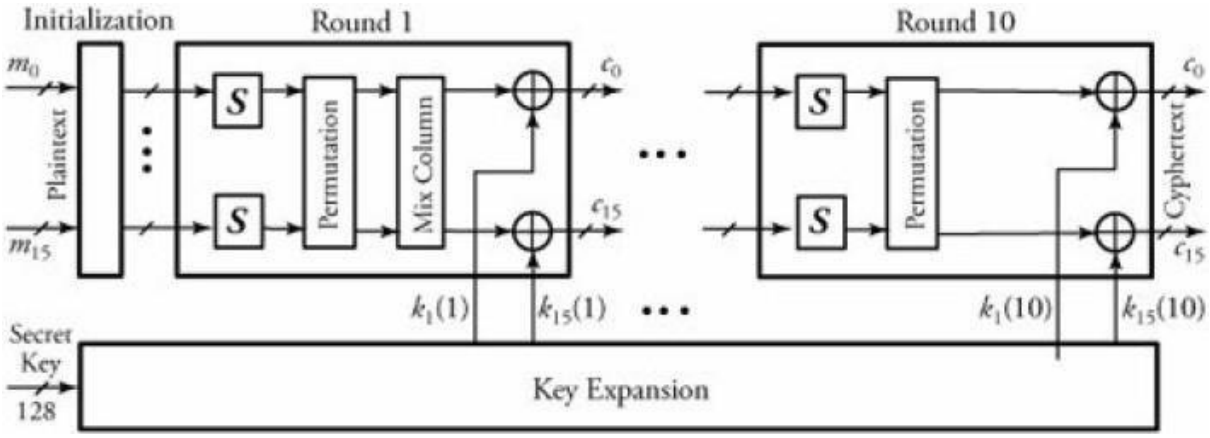
Operation of function F()

- Out of 56 bits of key k_i, function F( ) chooses 48 bits.
- The 32-bit R_{i-1} is expanded from 32 bits to 48 bits so that it can be combined with 48-bit k_i. The expansion of R_{i-1} is carried out by first breaking R_{i-1} into eight 4-bit chunks and then expanding each chunk by copying the leftmost bit and the rightmost bit from left and right adjacent chunks , respectively.
- F( ) also partitions the 48 bits of ki into eight 6-bit chunks.
- The corresponding eight chunks of R_{i-1} and eight chunks of kiare combined as follows

  $R_{i-1} = Ri\text{-}1 \oplus k_i$

AES (Advanced Encryption Standard)

- AES has better security strength than DES.
- In AES message is divided into 128-bit block, and it uses 128 or 192 or 256 bit key.
- Based on the key size number of rounds can be 10,12 or 14.
- The plaintext is formed as 16 bytes m0 through m15 and is fed into round 1 after an initialization stage.
- In this round, substitute-units(S) perform a byte-by-byte substitution of blocks.
- The ciphers move through a permutation-stage to shift rows to mix-columns.
- At the end of this round, all 16 blocks of ciphers are Exclusive-ORed with the 16 bytes of round 1 key k0(1) through k15(1).



Public Key Encryption Protocols

- This is also called as asymmetric or two key encryption.
- A sender/receiver pair use different keys.
- This is based on mathematical functions rather than on substitution or permutation.
- Two popular protocols are:

  i) RSA protocol

  ii)Diffie-Hillman key-exchange protocol.
- Either of the two related keys can be used for encryption; the other one for decryption.
- Each system publishes its encryption key by placing it in a public-register & sorts out key as public one. The companion key is kept private.
- If A wishes to send a message to B, A encrypts the message by using B's public key.
- At receiving end, B decrypts the message by using its private key.
- No other recipients can decrypt the message, since only B knows its private key.
- The public-key system is more powerful than the secret key system & provides better
- Drawbacks of public-key system:
    - slow speed
    - more complex computationally

RSA ALGORITHM

• Assume that a plaintext m must be encrypted to a cipher text c.

• This has three phases: key generation, encryption and decryption.

Key Generation Algorithm

1) Choose two prime numbers a and b and compute n=a.b

2) *Find x.* Select encryption-key x such that x and (a-1)(b-1) are relatively prime.

3) *Find y.* Calculate decryption-key y.

$xy \bmod (a-1)(b-1) = 1$

4) At this point, a and b can be discarded.

5) The public key = {x, n}

6) The private key = {y, n}

Encryption

1) Both sender and receiver must know the value of n.

2) The sender knows the value of x and only the receiver knows the value of y.

3) Ciphertext c is constructed by

$c=m^x \bmod n$

Decryption

Given the ciphertext c, the plaintext m is extracted by

$m=c^y \bmod n.$

DIFFIE-HILLMAN KEY-EXCHANGE PROTOCOL

• Two end users can agree on a shared secret-code without any information shared in advance.

• This protocol is normally used for VPN (virtual private network).

• Assume that user-1 wishes to communicate with user-2.

Key Generation Algorithm

1) User-1 selects a prime number 'a', random integer number 'x1', and a generator 'g'. Then creates 'y1' such that

$y^1 = g^{x1} \bmod a$

2) User-2 performs the same function and creates y2 such that

$y^2 = g^{x2} \bmod a$

3) User-1 then sends y1 to user-2. Now, user-1 forms its key k1 using the information its partner sent as

$k_1 = y_2{}^{x1} \bmod a$

4) User-2 forms its key ka using the information its partner send it as

$k_2 = y_1^{x2} \bmod a$

5) The two keys k1 and k2 are equal. The two users can now encrypt their messages, each using its own key

AUTHENTICATION

- Message-authentication verifies the authenticity of both the message-sender and themessage-content.
- Message-sender is authenticated through implementation of a digital signature.
- Message-content is authenticated through implementation of a hash function and encryption of the resulting message-digest.
- Hash-function is used to produce a "fingerprint" of a message.
- The hash-value is added at the end of message before transmission.
- The receiver re-computes the hash-value from the received message and compares it to the received hash value.
- If the two hash-values are the same, the message was not altered during transmission.
- Once a hash-function is applied on a message m, the result is known as a message-digest h(m).
- The hash-function has the following properties

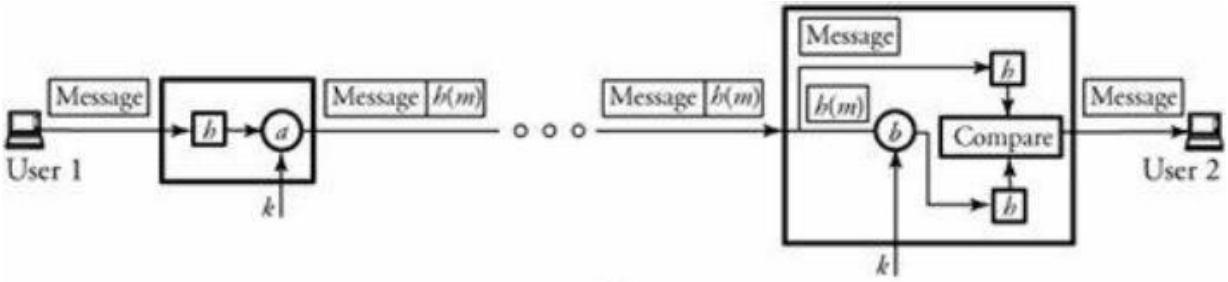
  1) Unlike the encryption-algorithm, the authentication algorithm is not required to be reversible.

  2) Given a message-digest h(m),it is computationally infeasible to find m.

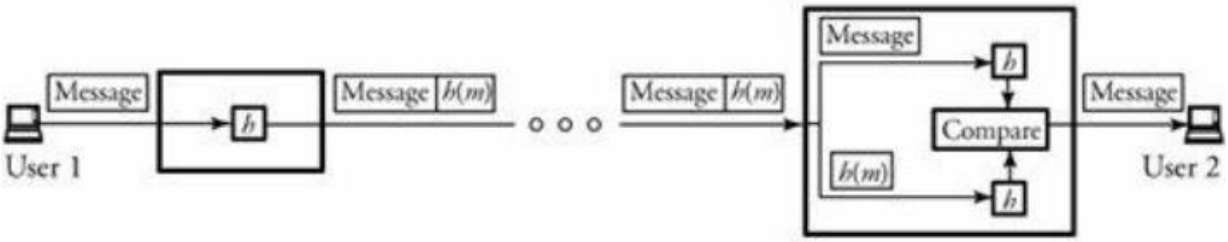
  3) This is computationally infeasible to find two different messages m1 and m2 such that h(m1)=h(m2).

• Message-authentication can be implemented by two methods.

1) In first method, a hash-function is applied on a message and then a process of encryption is implemented. At the receiver site, the received message-digest is decrypted and the comparison is made between the decrypted h(m) and the message-digest made locally from the received message. compare it with the one made locally at its site for any judgments on the integrity of the message.



2) In second method, no encryption is involved. The two parties share a secret key. Hence, at the receiving site, the comparison is made between the received h(m) and the message-digest made locally from the received message.



Secure Hash Algorithm (SHA)

- The Secure Hash Algorithm (SHA) was proposed as part of the digital signature standard. SHA-1, the first version of this standard, takes messages with a maximum length of 2^{24} and produces a 160-bit digest.
- With this algorithm, SHA-1 uses five registers, R_1 through R_5, to maintain a "state" of 20 bytes.
- The first step is to pad a message m with length l_m. The message length is forced to $l_m = 448 \bmod 512$. In other words, the length of the padded message becomes 64 bits less than the multiple of 512 bits.
- After padding, the second step is to expand each block of 512-bit (16 32 bits) words $\{m_0, m_1, ..., m_{15}\}$ to words of 80 32 bits using:

  $$w_i = m_i \text{ for } 0 \le i \le 15$$

  And

  $$w_i = w_{i-3} \oplus w_{i-8} \oplus w_{i-14} \oplus w_{i-16} \hookleftarrow 1 \text{ for } 16 \le i \le 79,$$

  where $\hookleftarrow$ means left rotation by j bits.
- Then, the 80 steps (i = 0, 1, 2, ..., 79) of the four rounds are described as follows

  $$\delta = (R_1 \hookleftarrow 5) + F_i(R_2, R_3, R_4) + R_5 + w_i + C_i$$

$$R_5 = R_4$$
$$R_4 = R_3$$
$$R_3 = R_2 \hookleftarrow 30$$
$$R_2 = R_1$$
$$R_1 = \delta,$$

Where C_i is a constant value specified by the standard for round i .

$$F_i(a,b,c) = \begin{cases} (a \cap b) \cup (\bar{a} \cap c) & 0 \le i \le 19 \\ a \oplus b \oplus c & 20 \le i \le 39 \\ (a \cap b) \cup (a \cap c) \cup (b \cap c) & 40 \le i \le 59 \\ a \oplus b \oplus c & 60 \le i \le 79 \end{cases}.$$

The message digest is produced by concatenation of the values in R_1 through R_5.

Authentication and Digital Signature

- A digital signature on a message is required for the authentication and identification of the right sender.
- RSA algorithm can be used to implement digital signature.
- The message is encrypted with the sender's private key. Thus, the entire encrypted message serves as a digital signature.
- At the receiving end, the receiver can decrypt the message using the public key. This authenticates that the packet comes from the right user.

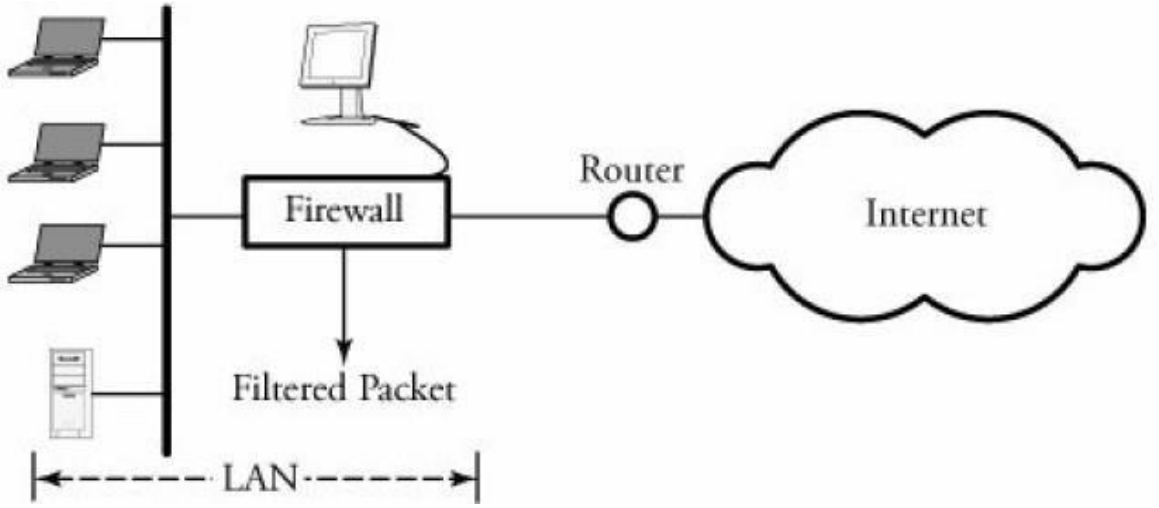
Firewalls

- Firewall is placed between hosts of a certain network and the outside world.
- Firewall is used to protect the network from unwanted web sites and potential hackers.
- The main objective is to monitor and filter packets coming from unknown sources.
- Firewall can also be used to control data traffic.
- Firewall can be a software program or a hardware device.

  1) Software firewalls can be installed in home computers by using an Internet connection with gateways.
- 2) Hardware firewalls are more secure than software firewalls are not expensive.

A firewall controls the flow of traffic by one of the following three methods:

1) Packet filtering: A firewall filters those packets that pass through. If packets can get through the filter, they reach their destinations: otherwise, they are discarded
2) A firewall filters packets based on the source IP address. This filtering is helpful when a host has to be protected from any unwanted external packets.
3) Denial of Service (DOS). This method controls the number of packets entering a network.



Module – 5

MULTIMEDIA NETWORKING

Multimedia Networking Applications

➔ Properties of Video

- Most salient characteristic of video is its high bit rate.
  - Video distributed over the Internet typically ranges from 100 kbps for low-quality video conferencing to over 3 Mbps for streaming high-definition movies.
  - Video streaming consumes most bandwidth, having a bit rate of more than ten times greater than that of the normal HTTP and music-streaming applications.
- Video can be compressed.
  - A video is a sequence of images, typically being displayed at a constant rate, for example, at 24 or 30 images per second.
  - An uncompressed, digitally encoded image consists of an array of pixels, with each pixel encoded into a number of bits to represent luminance and color.
  - There are two types of redundancy in video, both of which can be exploited by **video compression**.
  - **Spatial redundancy** is the redundancy within a given image. Intuitively, an image that consists of mostly white space has a high degree of redundancy and can be efficiently compressed without significantly sacrificing image quality.
  - **Temporal redundancy** reflects repetition from image to subsequent image. If, for example, an image and the subsequent image are exactly the same, there is no reason to reencode the subsequent image; it is instead more efficient simply to indicate during encoding that the subsequent image is exactly the same.
  - We can also use compression to create multiple versions of the same video, each at a different quality level. For example, we can use compression to create, say, three versions of the same video, at rates of 300 kbps, 1 Mbps, and 3 Mbps.

➔ Properties of Audio

- Digital audio has significantly lower bandwidth requirements than video.

- Analog audio can be converted to a digital signal using pulse code modulation with the following steps:
  - The analog audio signal is **sampled** at some fixed rate.
  - Each of the samples is then rounded to one of a finite number of values. This operation is referred to as **quantization**. The number of such finite values called quantization values.
  - Each of the quantization values are **encoded** by representing with a fixed number of bits.
- PCM-encoded speech and music, however, are rarely used in the Internet. Instead, as with video, compression techniques are used to reduce the bit rates of the stream.
- A popular compression technique for near CD-quality stereo music is MPEG 1 layer 3, more commonly known as MP3.
- MP3 encoders can compress to many different rates; 128 kbps is the most common encoding rate and produces very little sound degradation.
- As with video, multiple versions of a prerecorded audio stream can be created, each at a different bit rate.

➔ Types of Multimedia Network Applications

Multimedia applications are classified into three broad categories:

(i) Streaming stored audio/video

(ii) Conversational voice/video-over-IP

(iii) Streaming live audio/video

1) Streaming Stored Audio and Video

- In this class of applications, the underlying medium is prerecorded video, such as a movie, a television show, a prerecorded sporting event, or a prerecorded user generated video (such as those commonly seen on YouTube).
- These prerecorded videos are placed on servers, and users send requests to the servers to view the videos on demand.
- Many Internet companies today provide streaming video, including YouTube (Google), Netflix, and Hulu.
- By some estimates, streaming stored video makes up over 50 percent of the downstream traffic in the Internet access networks today.

Streaming stored video has three key distinguishing features.

- **Streaming:** In a streaming stored video application, the client typically begins video playout within a few seconds after it begins receiving the video from the server. This means that the client will be playing out from one location in the video while at the same time receiving later parts of the video from the server. This technique, known as streaming, avoids having to download the entire video file before playout begins.
- **Interactivity:** Because the media is prerecorded, the user may pause, reposition forward, reposition backward, fast-forward, and so on through the video content. The time from when the user makes such a request until the action manifests itself at the client should be less than a few seconds for acceptable responsiveness.
- **Continuous playout:** Once playout of the video begins, it should proceed according to the original timing of the recording. Therefore, data must be received from the server in time for its playout at the client; otherwise, users experience video frame freezing or frame skipping.

2) Conversational Voice- and Video-over-IP

- Real-time conversational voice over the Internet is often referred to as Internet telephony. It is also commonly called Voice-over-IP (VoIP).
- Conversational video is similar, except that it includes the video of the participants as well as their voices.
- Most of today's voice and video conversational systems allow users to create conferences with three or more participants.
- Conversational voice and video are widely used in the Internet today, with the Internet companies Skype, QQ, and Google Talk boasting hundreds of millions of daily users.
- Timing considerations and tolerance of data loss are important for conversational voice and video applications.
- Timing considerations are important because audio and video conversational applications are highly delay-sensitive. For a conversation with two or more interacting speakers, the delay from when a user speaks or moves until the action is manifested at the other end should be less than a few hundred milliseconds.

- On the other hand, conversational multimedia applications are loss-tolerant— occasional loss only causes occasional glitches in audio/video playback, and these losses can often be partially or fully concealed.

3) Streaming Live Audio and Video

- This third class of applications is similar to traditional broadcast radio and television, except that transmission takes place over the Internet.
- These applications allow a user to receive a live radio or television transmission—such as a live sporting event or an ongoing news event—transmitted from any corner of the world.
- Today, thousands of radio and television stations around the world are broadcasting content over the Internet.
- Live, broadcast-like applications often have many users who receive the same audio/video program at the same time.
- Although the distribution of live audio/video to many receivers can be efficiently accomplished using the IP multicasting techniques, multicast distribution is more often accomplished today via application-layer multicast (using P2P networks or CDNs) or through multiple separate unicast streams.
- As with streaming stored multimedia, the network must provide each live multimedia flow with an average throughput that is larger than the video consumption rate. Because the event is live, delay can also be an issue, although the timing constraints are much less stringent than those for conversational voice.

Streaming Stored Video

- For streaming video applications, prerecorded videos are placed on servers, and users send requests to these servers to view the videos on demand.
- The user may watch the video from beginning to end without interruption, may stop watching the video well before it ends, or interact with the video by pausing or repositioning to a future or past scene.
- Streaming video systems can be classified into three categories:
  1. UDP streaming

2. HTTP streaming
3. Adaptive HTTP streaming.

- A common characteristic of all three forms of video streaming is the extensive use of client-side application buffering to mitigate the effects of varying end-to-end delays and varying amounts of available bandwidth between server and client.
- When the video starts to arrive at the client, the client need not immediately begin playout, but can instead build up a reserve of video in an application buffer. Once the client has built up a reserve of several seconds of buffered-but-not-yet-played video, the client can then begin video playout.
- There are two important advantages provided by such client buffering. First, client side buffering can absorb variations in server-to-client delay. Second, if the server-to-client bandwidth briefly drops below the video consumption rate, a use can continue to enjoy continuous playback, again as long as the client application buffer does not become completely drained.

➔ UDP Streaming

- With UDP streaming, the server transmits video at a rate that matches the client's video consumption rate by clocking out the video chunks over UDP at a steady rate.
- For example, if the video consumption rate is 2 Mbps and each UDP packet carries 8,000 bits of video, then the server would transmit one UDP packet into its socket every (8000 bits)/(2 Mbps) = 4 msec.
- UDP does not employ a congestion-control mechanism, the server can push packets into the network at the consumption rate of the video without the rate-control restrictions of TCP.
- Before passing the video chunks to UDP, the server will encapsulate the video chunks within transport packets specially designed for transporting audio and video, using the Real-Time Transport Protocol (RTP).
- The client and server also maintain, in parallel, a separate control connection over which the client sends commands regarding session state changes (such as pause, resume, reposition,

and so on). The Real-Time Streaming Protocol is a popular open protocol for such a control connection.

Limitation:

- Due to the unpredictable and varying amount of available bandwidth between server and client, constant-rate UDP streaming can fail to provide continuous playout.
- It requires a media control server, such as an RTSP server, to process client-to-server interactivity requests and to track client state for each ongoing client session.
- Many firewalls are configured to block UDP traffic, preventing the users behind these firewalls from receiving UDP video.

➔ HTTP Streaming

- In HTTP streaming, the video is simply stored in an HTTP server as an ordinary file with a specific URL.
- When a user wants to see the video, the client establishes a TCP connection with the server and issues an HTTP GET request for that URL.
- The server then sends the video file, within an HTTP response message, as quickly as possible, that is, as quickly as TCP congestion control and flow control will allow.
- On the client side, the bytes are collected in a client application buffer. Once the number of bytes in this buffer exceeds a predetermined threshold, the client application begins playback—specifically, it periodically grabs video frames from the client application buffer, decompresses the frames, and displays them on the user's screen.

Advantages:

- The use of HTTP over TCP also allows the video to traverse firewalls and NATs more easily.
- Streaming over HTTP also obviates the need for a media control server, such as an RTSP server, reducing the cost of a large-scale deployment over the Internet.

Limitation and solution:

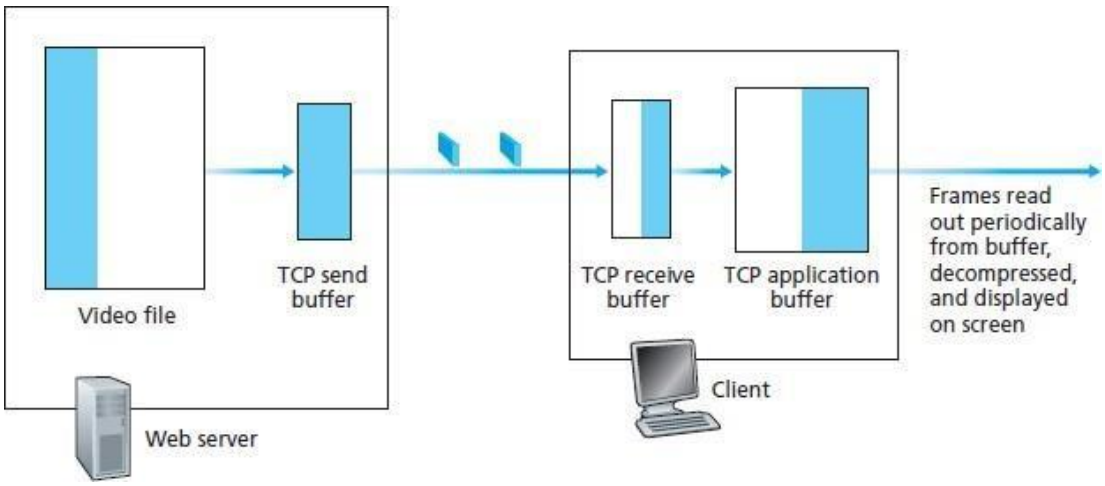
When transferring a file over TCP, the server-to client transmission rate can vary significantly due to TCP's congestion control mechanism. Packets can also be significantly delayed due to

TCP's retransmission mechanism. Because of these characteristics of TCP, it was believed that video streaming would never work well over TCP. Over time, however, designers of streaming video systems learned that TCP's congestion control and reliable-data transfer mechanisms do not necessarily preclude continuous playout when client buffering and prefetching are used.

Prefetching Video:

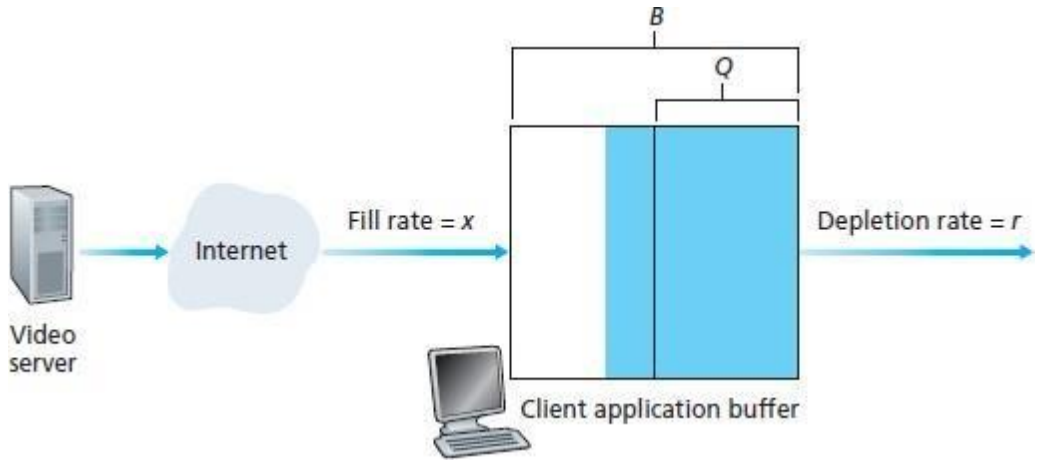
The client can attempt to download the video at a rate higher than the consumption rate, thereby prefetching video frames that are to be consumed in the future. This prefetched video is naturally stored in the client application buffer.

Client Application Buffer and TCP Buffers:



- Here TCP send buffer is shown to be full, the server is momentarily prevented from sending more bytes from the video file into the socket.
- On the client side, the client application reads bytes from the TCP receive buffer and places the bytes into the client application buffer.
- At the same time, the client application periodically grabs video frames from the client application buffer, decompresses the frames, and displays them on the user's screen.
- Consider now what happens when the user pauses the video during the streaming process. During the pause period, bits are not removed from the client application buffer, even though bits continue to enter the buffer from the server. If the client application buffer is finite, it may eventually become full, which will cause "back pressure" all the way back to the server.

Analysis of Video Streaming:



- Let B denote the size (in bits) of the client's application buffer, and let Q denote the number of bits that must be buffered before the client application begins playout.
- Let r denote the video consumption rate—the rate at which the client draws bits out of the client application buffer during playback.
- Let's assume that the server sends bits at a constant rate x whenever the client buffer is not full.
- If x < r (that is, if the server send rate is less than the video consumption rate), then the client buffer will never become full.
- When the available rate in the network is more than the video rate, after the initial buffering delay, the user will enjoy continuous playout until the video ends.

Early Termination and Repositioning the Video:

➔ HTTP streaming systems often make use of the **HTTP byte-range** header in the HTTP GET request message, which specifies the specific range of bytes the client currently wants to retrieve from the desired video.

➔ This is particularly useful when the user wants to reposition (that is, jump) to a future point in time in the video.

➔ When the user repositions to a new position, the client sends a new HTTP request, indicating with the byte-range header from which byte in the file should the server send data.

➔ When the server receives the new HTTP request, it can forget about any earlier request and instead send bytes beginning with the byte indicated in the byterange request.

➔ Adaptive Streaming and DASH

Shortcoming of HTTP Streaming:

All clients receive the same encoding of the video, despite the large variations in the amount of bandwidth available to a client, both across different clients and also over time for the same client.

Solution: **DASH**

- In DASH - Dynamic Adaptive Streaming over HTTP, the video is encoded into several different versions, with each version having a different bit rate and, correspondingly, a different quality level. The client dynamically requests chunks of video segments of a few seconds in length from the different versions.
- With DASH, each video version is stored in the HTTP server, each with a different URL.
- The HTTP server also has a manifest file, which provides a URL for each version along with its bit rate.
- The client first requests the manifest file and learns about the various versions.
- The client then selects one chunk at a time by specifying a URL and a byte range in an HTTP GET request message for each chunk.
- While downloading chunks, the client also measures the received bandwidth and runs a rate determination algorithm to select the chunk to request next.
- Naturally, if the client has a lot of video buffered and if the measured receive bandwidth is high, it will choose a chunk from a high-rate version. And naturally if the client has little video buffered and the measured received bandwidth is low, it will choose a chunk from a low-rate version.
- By dynamically monitoring the available bandwidth and client buffer level, and adjusting the transmission rate with version switching, DASH can often achieve continuous playout at the best possible quality level without frame freezing or skipping.

Content Distribution Networks

- Streaming stored video to locations all over the world while providing continuous playout and high interactivity is clearly a challenging task.

- For an Internet video company, the most straightforward approach to providing streaming video service is to build a single massive data center which stores all of its videos in the data center, and stream the videos directly from the data center to clients worldwide.
- But this approach faces some problems:
  - Single massive date center is single point of failure
  - It leads long path to distant clients
  - It may create network congestion.
  - Popular video will likely be sent many times over the same communication links. Not only does this waste network bandwidth, but the Internet video company itself will be paying its provider ISP (connected to the data center) for sending the same bytes into the Internet over and over again.
- In order to meet the challenge of distributing massive amounts of video data to users distributed around the world, almost all major video-streaming companies make use of Content Distribution Networks (CDNs).
- A CDN manages servers in multiple geographically distributed locations, stores copies of the videos (and other types of Web content, including documents, images, and audio) in its servers, and attempts to direct each user request to a CDN location that will provide the best user experience.
- The CDN may be a private CDN,that is, owned by the content provider itself; for example, Google's CDN distributes YouTube videos and other types of content.
- The CDN may alternatively be a third-party CDN that distributes content on behalf of multiple content providers; for example, Akamai's CDN is a third party CDN that distributes Netflix and Hulu content, among others.
- CDNs typically adopt one of two different server placement philosophies:
  - **Enter Deep:** One philosophy, pioneered by Akamai, is to enter deep into the access networks of Internet Service Providers, by deploying server clusters in access ISPs all over the world. The goal is to get close to end users, thereby improving user-perceived delay and throughput by decreasing the number of links and routers between the end user and the CDN cluster from which it receives content.
  - **Bring Home:** A second design philosophy, taken by Limelight and many other CDN companies, is to bring the ISPs home by building large clusters at a smaller number (for

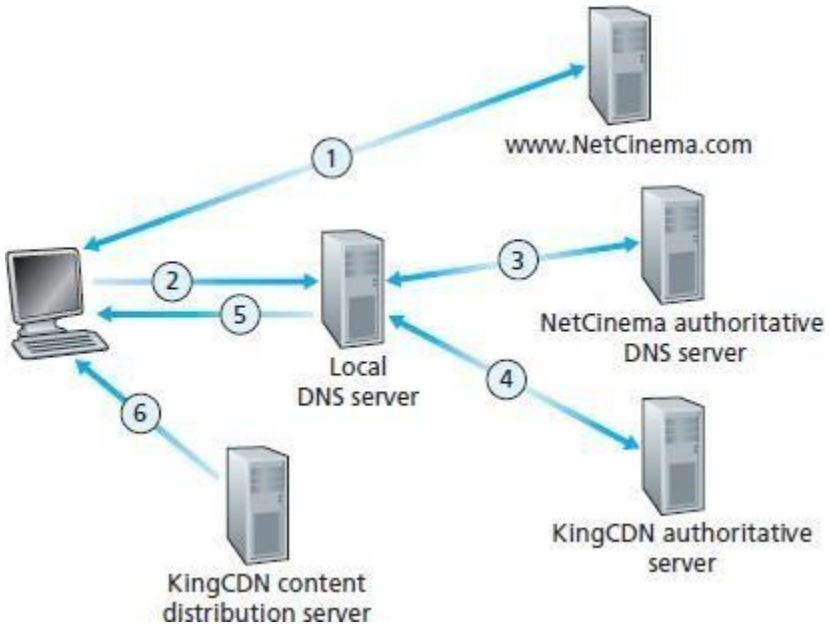
example, tens) of key locations and connecting these clusters using a private high-speed network. Instead of getting inside the access ISPs, these CDNs typically place each cluster at a location that is simultaneously near the PoPs of many tier-1 ISPs

➔ CDN Operation

When a browser in a user's host is instructed to retrieve a specific video (identified by a URL), the CDN must intercept the request so that it can

(1) Determine a suitable CDN server cluster for that client at that time.

(2) Redirect the client's request to a server in that cluster.



1. The user visits the Web page at NetCinema.
2. When the user clicks on the link http://video.netcinema.com/6Y7B23V, the user's host sends a DNS query for video.netcinema.com.
3. The user's Local DNS Server (LDNS) relays the DNS query to an authoritative DNS server for NetCinema, which observes the string "video" in the hostname video.netcinema.com. To "hand over" the DNS query to KingCDN, instead of returning an IP address, the NetCinema authoritative DNS server returns to the LDNS a hostname in the KingCDN's domain, for example, a1105.kingcdn.com.
4. From this point on, the DNS query enters into KingCDN's private DNS infrastructure. The user's LDNS then sends a second query, now for a1105.kingcdn.com, and KingCDN's DNS system eventually returns the IP addresses of a KingCDN content server to the LDNS. It is

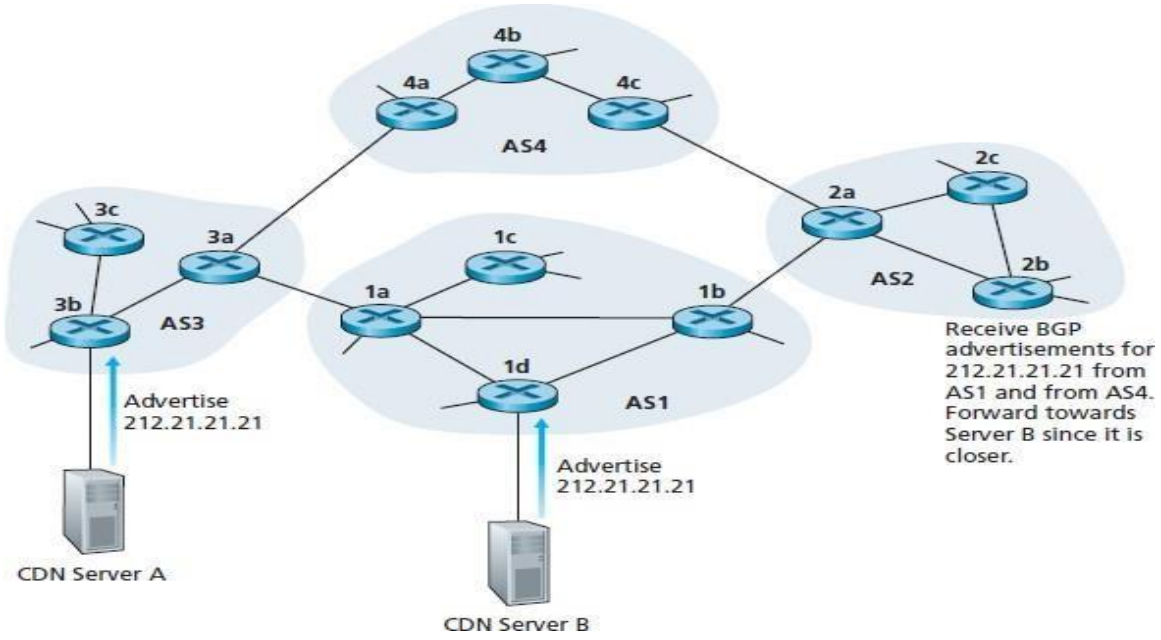
thus here, within the KingCDN's DNS system, that the CDN server from which the client will receive its content is specified.

5. The LDNS forwards the IP address of the content-serving CDN node to the user's host.
6. Once the client receives the IP address for a KingCDN content server, it establishes a direct TCP connection with the server at that IP address and issues an HTTP GET request for the video. If DASH is used, the server will first send to the client a manifest file with a list of URLs, one for each version of the video, and the client will dynamically select chunks from the different versions.

➔ Cluster Selection Strategies

- Cluster Selection Strategies is a mechanism for dynamically directing clients to a server cluster or a data center within the CDN.
- The CDN learns the IP address of the client's LDNS server via the client's DNS lookup. After learning this IP address, the CDN needs to select an appropriate cluster based on this IP address.
- One simple strategy is to assign the client to the cluster that is geographically closest. Using commercial geo-location databases each LDNS IP address is mapped to a geographic location. When a DNS request is received from a particular LDNS, the CDN chooses the geographically closest cluster.
- For some clients, the solution may perform poorly, since the geographically closest cluster may not be the closest cluster along the network path.
- In order to determine the best cluster for a client based on the current traffic conditions, CDNs can instead perform periodic real-time measurements of delay and loss performance between their clusters and clients.
- An alternative to sending extraneous traffic for measuring path properties is to use the characteristics of recent and ongoing traffic between the clients and CDN servers.
- Such solutions, however, require redirecting clients to (possibly) suboptimal clusters from time to time in order to measure the properties of paths to these clusters.
- A very different approach to matching clients with CDN servers is to use IP anycast. The idea behind IP anycast is to have the routers in the Internet route the client's packets to the "closest" cluster, as determined by BGP.

- During the IP-anycast configuration stage, the CDN company assigns the same IP address to each of its clusters, and uses standard BGP to advertise this IP address from each of the different cluster locations.
- When a BGP router receives multiple route advertisements for this same IP address, it treats these advertisements as providing different paths to the same physical location.
- Following standard operating procedures, the BGP router will then pick the "best" route to the IP address according to its local route selection mechanism.
- After this initial configuration phase, the CDN can do its main job of distributing content. When any client wants to see any video, the CDN's DNS returns the anycast address, no matter where the client is located.
- When the client sends a packet to that IP address, the packet is routed to the "closest" cluster as determined by the preconfigured forwarding tables, which were configured with BGP.



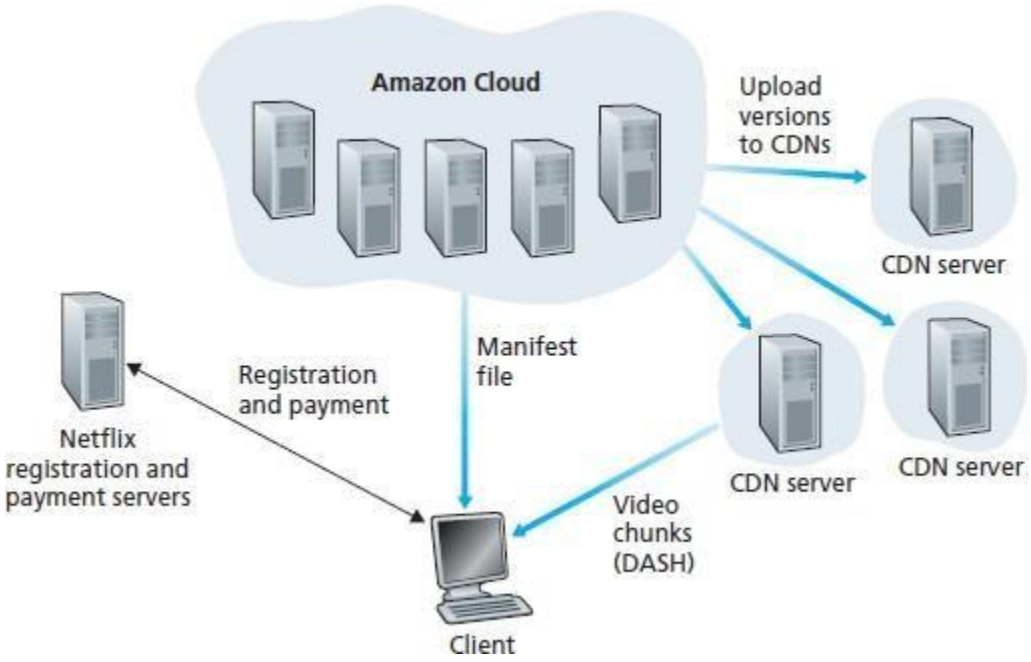
➔ Case Studies: Netflix, YouTube, and Kankan

Netflix

- Netflix is the leading service provider for online movies and TV shows in the United States.
- In order to rapidly deploy its large-scale service, Netflix has made extensive use of third-party cloud services and CDNs. Indeed, Netflix is an interesting example of a company

deploying a large-scale online service by renting servers, bandwidth, storage, and database services from third parties while using hardly any infrastructure of its own.

- Basic Architecture:



- Netflix has four major components: the registration and payment servers, the Amazon cloud, multiple CDN providers, and clients.
- In its own hardware infrastructure, Netflix maintains registration and payment servers, which handle registration of new accounts and capture credit-card payment information.
- Netflix runs its online service by employing machines (or virtual machines) in the Amazon cloud. Some of the functions taking place in the Amazon cloud include:
  - **Content ingestion:** Before Netflix can distribute a movie to its customers, it must first ingest and process the movie. Netflix receives studio master versions of movies and uploads them to hosts in the Amazon cloud.
  - **Content processing:** The machines in the Amazon cloud create many different formats for each movie, suitable for a diverse array of client video players running on desktop computers, smartphones, and game consoles connected to televisions. A different version is created for each of these formats and at multiple bit rates, allowing for adaptive streaming over HTTP using DASH.

- **Uploading versions to the CDNs:** Once all of the versions of a movie have been created, the hosts in the Amazon cloud upload the versions to the CDNs.

- The Web pages for browsing the Netflix video library are served from servers in the Amazon cloud.
- When the user selects a movie to "Play Now," the user's client obtains a manifest file, also from servers in the Amazon cloud. The manifest file includes a variety of information, including a ranked list of CDNs and the URLs for the different versions of the movie, which are used for DASH playback.
- The ranking of the CDNs is determined by Netflix, and may change from one streaming session to the next.
- Typically the client will select the CDN that is ranked highest in the manifest file.
- After the client selects a CDN, the CDN leverages DNS to redirect the client to a specific CDN server.
- The client and that CDN server then interact using DASH.

Youtube:

- With approximately half a billion videos in its library and half a billion video views per day, YouTube is indisputably the world's largest video-sharing site.
- YouTube began its service in April 2005 and was acquired by Google in November 2006.
- Google does not employ third-party CDNs but instead uses its own private CDN to distribute
- YouTube videos.
- Google has installed server clusters in many hundreds of different locations. From a subset of about 50 of these locations, Google distributes YouTube video.
- Google uses DNS to redirect a customer request to a specific cluster.
- Most of the time,
- Google's cluster selection strategy directs the client to the cluster for which the RTT between client and cluster is the lowest; however, in order to balance the load across clusters, sometimes the client is directed (via DNS) to a more distant cluster.
- If a cluster does not have the requested video, instead of fetching it from somewhere else and relaying it to the client, the cluster may return an HTTP redirect message, thereby redirecting the client to another cluster.

- YouTube employs HTTP streaming. YouTube often makes a small number of different versions available for a video, each with a different bit rate and corresponding quality level.
- YouTube processes each video it receives, converting it to a YouTube video format and creating multiple versions at different bit rates. This processing takes place entirely within Google data centers.

Kankan

- Kankan allows the service provider to significantly reduce its infrastructure and bandwidth costs.
- This approach uses P2P delivery instead of client-server (via CDNs) delivery. P2P video delivery is used with great success by several companies in China, including Kankan (owned and operated by Xunlei), PPTV (formerly PPLive), and PPs (formerly PPstream).
- Kankan, currently the leading P2P-based video-on-demand provider in China, has over 20 million unique users viewing its videos every month.
- At a high level, P2P video streaming is very similar to BitTorrent file downloading.
- When a peer wants to see a video, it contacts a tracker (which may be centralized or peer-based using a DHT) to discover other peers in the system that have a copy of that video.
- This peer then requests chunks of the video file in parallel from these other peers that have the file.
- Different from downloading with BitTorrent, however, requests are preferentially made for chunks that are to be played back in the near future in order to ensure continuous playback.
- The Kankan design employs a tracker and its own DHT for tracking content.

Network Support for Multimedia

There exist three broad approaches towards providing network-level support for multimedia applications.

Approach	Granularity	Guarantee	Mechanisms	Complexity	Deployment to date
Making the best of best-effort service.	all traffic treated equally	none, or soft	application-layer support, CDNs, overlays, network-level resource provisioning	minimal	everywhere
Differentiated service	different classes of traffic treated differently	none, or soft	packet marking, policing, scheduling	medium	some
Per-connection Quality-of-Service (QoS) Guarantees	each source-destination flows treated differently	soft or hard, once flow is admitted	packet marking, policing, scheduling; call admission and signaling	light	little

- **Making the best of best-effort service:** The application-level mechanisms and infrastructure can be successfully used in a well-dimensioned network where packet loss and excessive end-to-end delay rarely occur. When demand increases are forecasted, the ISPs deploy additional bandwidth and switching capacity to continue to ensure satisfactory delay and packet-loss performance.
- **Differentiated service:** With differentiated service, one type of traffic might be given strict priority over another class of traffic when both types of traffic are queued at a router.
- **Per-connection Quality-of-Service (QoS) Guarantees:** With per-connection QoS guarantees, each instance of an application explicitly reserves end-to-end bandwidth and thus has a guaranteed end-to-end performance. A hard guarantee means the application will receive its requested quality of service (QoS) with certainty. A soft guarantee means the application will receive its requested quality of service with high probability.

➔ Dimensioning Best-Effort Networks

- A first approach to improving the quality of multimedia applications is through providing enough link capacity throughout the network so that network congestion, and its consequent

packet delay and loss, never (or only very rarely) occurs. With enough link capacity, packets could zip through today's Internet without queuing delay or loss.

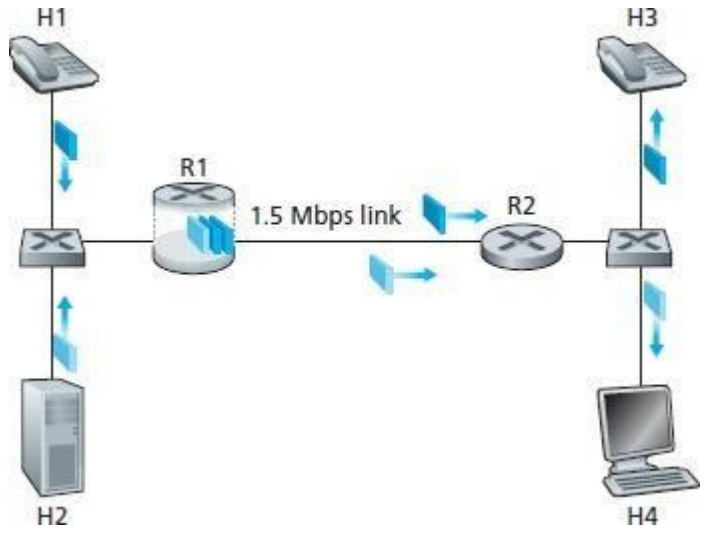
- Challenges:
  - The question of how much capacity to provide at network links in a given topology to achieve a given level of performance is often known as **bandwidth provisioning**.
  - The even more complicated problem of how to design a network topology (where to place routers, how to interconnect routers with links, and what capacity to assign to links) to achieve a given level of end-to-end performance is a network design problem often referred to as **network dimensioning**.

➔ Providing Multiple Classes of Service

The simplest enhancement to the one-size-fits-all best-effort service in today's Internet is to divide traffic into classes, and provide different levels of service to these different classes of traffic.

The type-of-service (ToS) field in the IPv4 header can be used for this purpose.

Motivating Scenarios



Here H1 and H3 are using audio application, H2 and H4 are using HTTP web application.

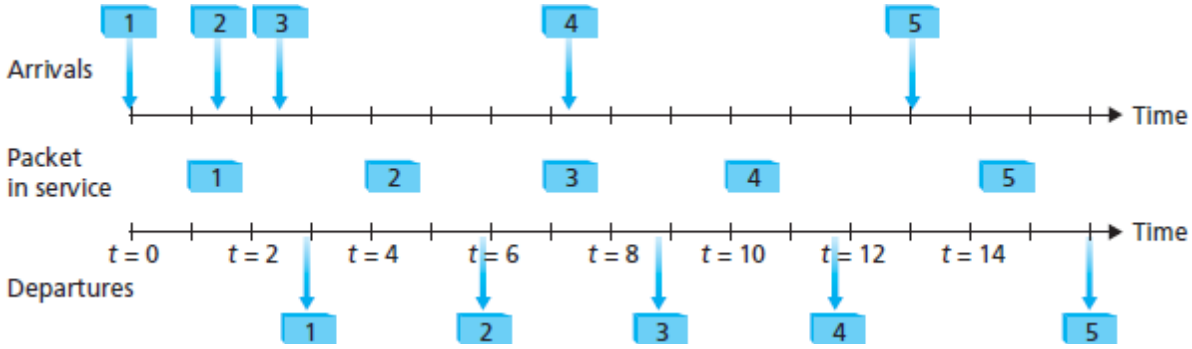
- In the best-effort Internet, the audio and HTTP packets are mixed in the output queue at R1 and (typically) transmitted in a first-in-first-out (FIFO) order.

- In this scenario, a burst of packets from the Web server could potentially fill up the queue, causing IP audio packets to be excessively delayed or lost due to buffer overflow at R1.
- Solution for this is differentiating traffic class and assigning suitable priority to it.
- **Packet marking** allows a router to distinguish among packets belonging to different classes of traffic.
- Now suppose that the router is configured to give priority to packets marked as belonging to the 1 Mbps audio application. Since the outgoing link speed is 1.5 Mbps, even though the HTTP packets receive lower priority, they can still, on average, receive 0.5 Mbps of transmission service. But if the audio application starts sending packets at a rate of 1.5 Mbps or higher, the HTTP packets will starve, that is, they will not receive any service on the R1-to-R2 link.
- Therefore it is desirable to provide a degree of traffic isolation among classes so that one class is not adversely affected by another class of traffic that misbehaves.
- If a traffic class or flow must meet certain criteria, then a policing mechanism can be put into place to ensure that these criteria are indeed observed. If the policed application misbehaves, the policing mechanism will take some action so that the traffic actually entering the network conforms to the criteria.
- A complementary approach for providing isolation among traffic classes is for the link-level packet-scheduling mechanism to explicitly allocate a fixed amount of link bandwidth to each class.
- While providing isolation among classes or flows, it is desirable to use resources (for example, link bandwidth and buffers) as efficiently as possible.

➔ Scheduling Mechanisms

First-In-First-Out (FIFO)

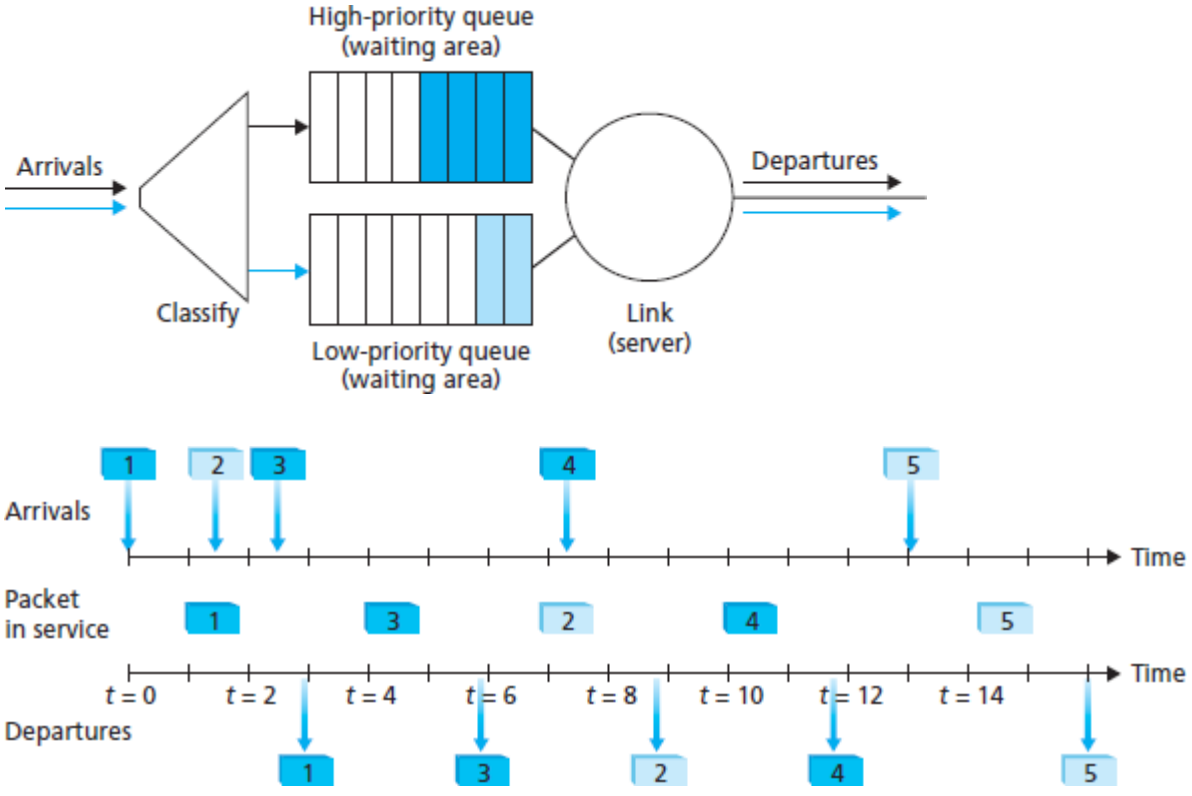
The FIFO (also known as first-come-first-served, or FCFS) scheduling discipline selects packets for link transmission in the same order in which they arrived at the output link queue.



Priority Queuing

Under priority queuing, packets arriving at the output link are classified into priority classes at the output queue.

Each priority class typically has its own queue. When choosing a packet to transmit, the priority queuing discipline will transmit a packet from the highest priority class that has a nonempty queue. The choice among packets in the same priority class is typically done in a FIFO manner.



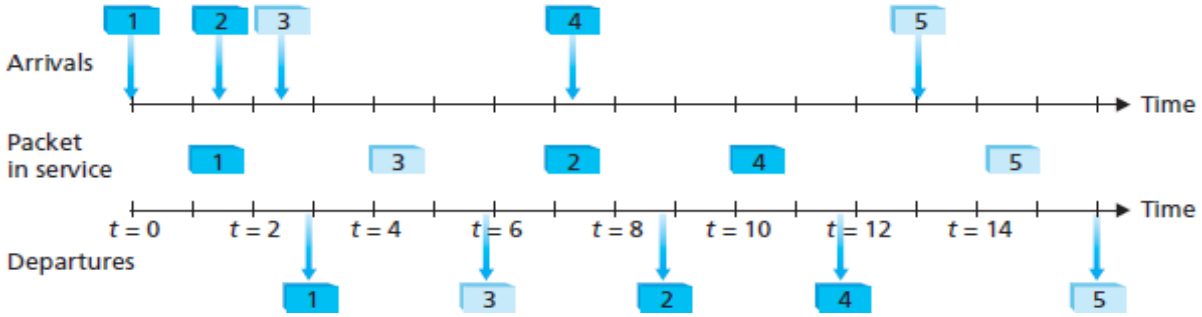
Round Robin

Under the round robin queuing discipline, packets are sorted into classes as with priority queuing.

However, rather than there being a strict priority of service among classes, a round robin scheduler alternates service among the classes.

In the simplest form of round robin scheduling, a class 1 packet is transmitted, followed by a class 2 packet, followed by a class 1 packet, followed by a class 2 packet, and so on.

A work-conserving round robin discipline that looks for a packet of a given class but finds none will immediately check the next class in the round robin sequence.

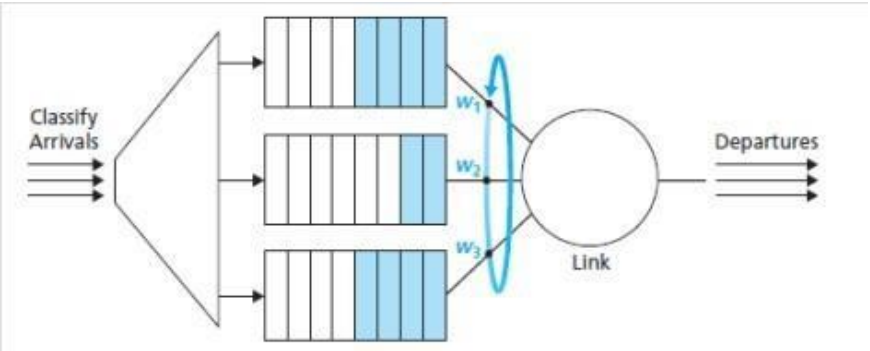


Weighted Fair Queuing (WFQ)

A generalized abstraction of round robin queuing that has found considerable use in QoS architectures is weighted fair queuing (WFQ) discipline.

Here arriving packets are classified and queued in the appropriate per-class waiting area. As in round robin scheduling, a WFQ scheduler will serve classes in a circular manner—first serving class 1, then serving class 2, then serving class 3, and then (assuming there are three classes) repeating the service pattern.

WFQ is also a work-conserving queuing discipline and thus will immediately move on to the next class in the service sequence when it finds an empty class queue.

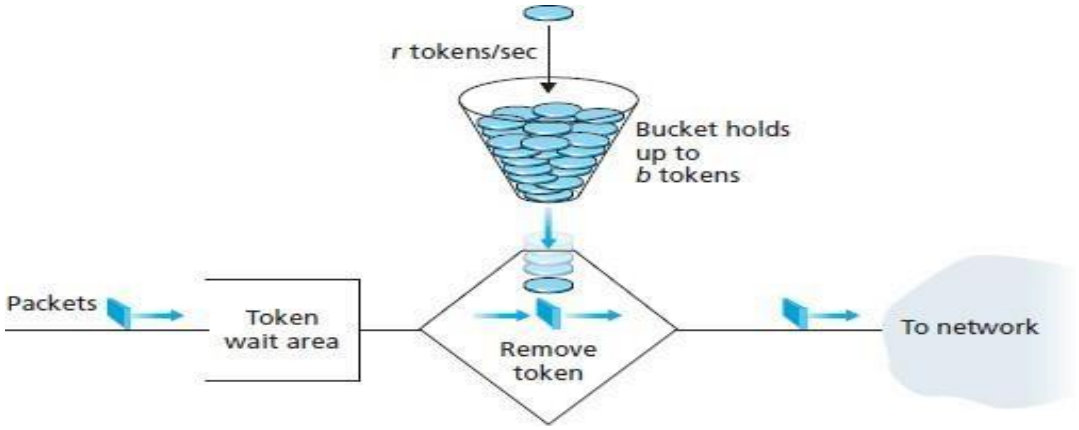


➔ Policing: The Leaky Bucket

Three important policing criteria:

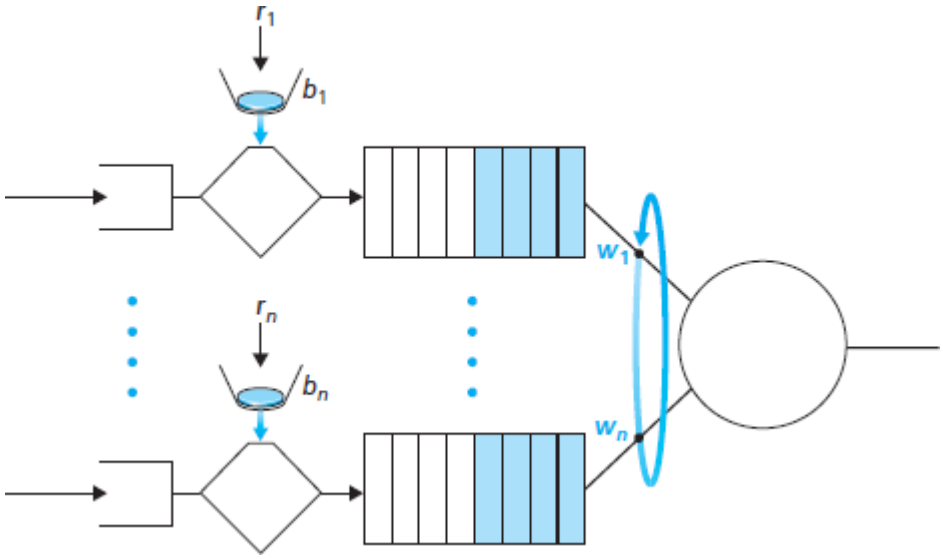
- **Average rate:** The network may wish to limit the long-term average rate (packets per time interval) at which a flow's packets can be sent into the network. A crucial issue here is the interval of time over which the average rate will be policed.
- **Peak rate:** While the average-rate constraint limits the amount of traffic that can be sent into the network over a relatively long period of time, a peak-rate constraint limits the maximum number of packets that can be sent over a shorter period of time.
- **Burst size:** The network may also wish to limit the maximum number of packets (the "burst" of packets) that can be sent into the network over an extremely short interval of time.

The leaky bucket mechanism is an abstraction that can be used to characterize these policing limits.



- A leaky bucket consists of a bucket that can hold up to b tokens.
- Tokens are added to this bucket as follows. New tokens, which may potentially be added to the bucket, are always being generated at a rate of r tokens per second.
- If the bucket is filled with less than b tokens when a token is generated, thenewly generated token is added to the bucket; otherwise the newly generated token is ignored, and the token bucket remains full with b tokens.
- Suppose that before a packet is transmitted into the network, it must first remove a token from the token bucket. If the token bucket is empty, the packet must wait for a token.

- Because there can be at most b tokens in the bucket, the maximum burst size for a leaky-bucket policed flow is b packets. Furthermore, because the token generation rate is r, the maximum number of packets that can enter the network of any interval of time of length t is rt + b.
- Leaky Bucket + Weighted Fair Queuing = Provable Maximum Delay in a Queue



➔ Diffserv

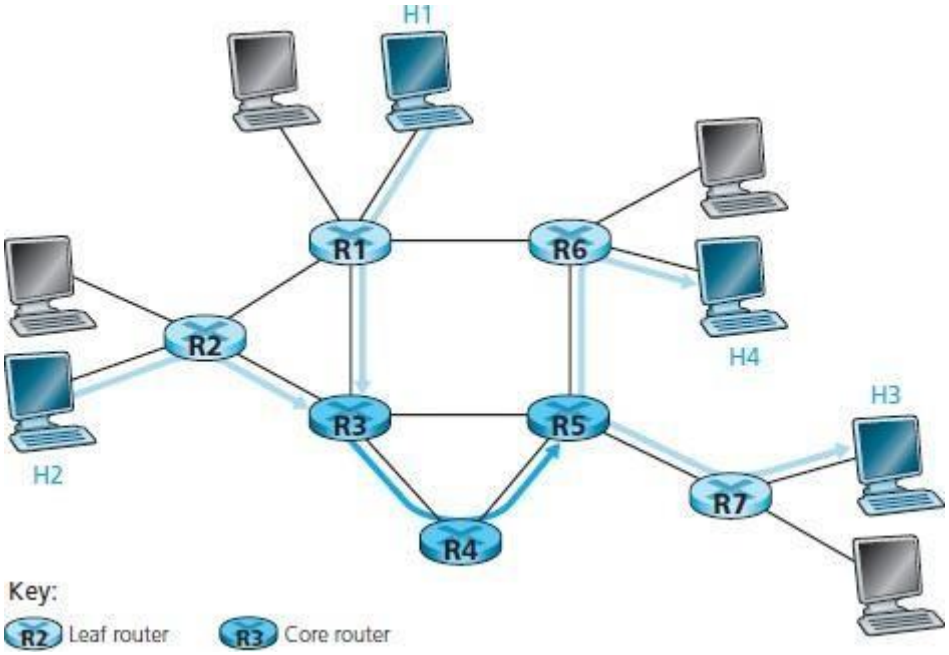
Diffserv provides service differentiation—that is, the ability to handle different classes of traffic in different ways within the Internet in a scalable manner.

The need for scalability arises from the fact that millions of simultaneous source-destination traffic flows may be present at a backbone router.

The Diffserv architecture consists of two sets of functional elements:

1) Edge functions: packet classification and traffic conditioning. At the incoming edge of the network arriving packets are marked. The mark that a packet receives identifies the class of traffic to which it belongs. Different classes of traffic will then receive different service within the core network.

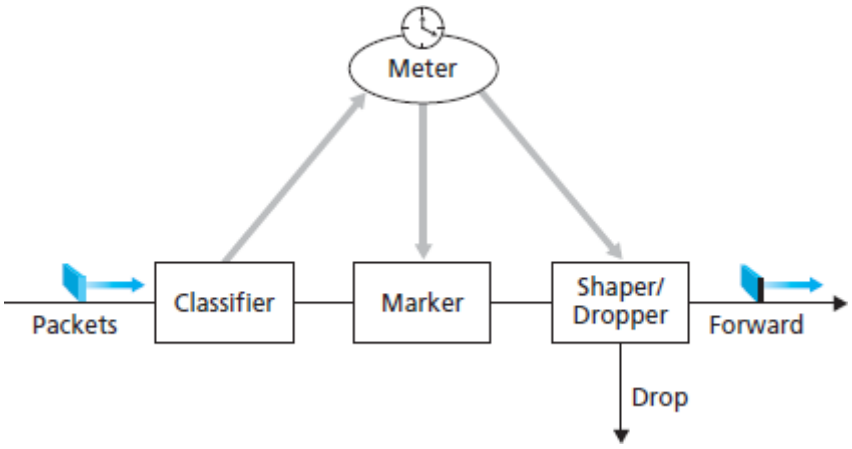
2) Core function: forwarding. When a DS-marked packet arrives at a Diffserv capable router, the packet is forwarded onto its next hop according to the so-called per-hop behavior (PHB) associated with that packet's class. The per-hop behavior influences how a router's buffers and link bandwidth are shared among the competing classes of traffic.



- Packets arriving to the edge router are first classified. The classifier selects packets based on the values of one or more packet header fields (for example, source address, destination address, source port, destination port, and protocol ID) and steers the packet to the appropriate marking function.
- In some cases, an end user may have agreed to limit its packet-sending rate to conform to a declared traffic profile. The traffic profile might contain a limit on the peak rate, as well as the burstiness of the packet flow.
- As long as the user sends packets into the network in a way that conforms to the negotiated traffic profile, the packets receive their priority marking and are forwarded along their route to the destination.
- On the other hand, if the traffic profile is violated, out-of-profile packets might be marked differently, might be shaped (for example, delayed so that a maximum rate constraint would be observed), or might be dropped at the network edge.
- The role of the metering function, is to compare the incoming packet flow with the negotiated traffic profile and to determine whether a packet is within the negotiated traffic profile.
- The second key component of the Diffserv architecture involves the per-hop behavior (PHB) performed by Diffserv-capable routers. PHB is rather cryptically, but carefully, defined as “a

description of the externally observable forwarding behavior of a Diffserv node applied to a particular Diffserv behavior aggregate".

- A PHB can result in different classes of traffic receiving different performance.

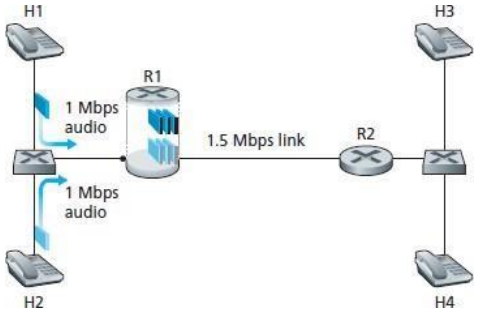


- The **expedited forwarding** PHB specifies that the departure rate of a class of traffic from a router must equal or exceed a configured rate.
- The **assured forwarding** PHB divides traffic into four classes, where each AF class is guaranteed to be provided with some minimum amount of bandwidth and buffering.

Per-Connection Quality-of-Service (QoS) Guarantees: Resource Reservation and Call Admission

Consider two 1 Mbps audio applications transmitting their packets over the 1.5 Mbps link. The combined data rate of the two flows (2 Mbps) exceeds the link capacity.

There is simply not enough bandwidth to accommodate the needs of both applications at the same time. If the two applications equally share the bandwidth, each application would lose 25 percent of its transmitted packets.



If sufficient resources will not always be available, and QoS is to be guaranteed, a call admission process is needed in which flows declare their QoS requirements and are then either admitted to the network (at the required QoS) or blocked from the network (if the required QoS cannot be provided by the network).

The process of having a flow declare its QoS requirement, and then having the network either accept the flow (at the required QoS) or block the flow is referred to as the call admission process.

Resource reservation: The only way to guarantee that a call will have the resources (link bandwidth, buffers) needed to meet its desired QoS is to explicitly allocate those resources to the call—a process known in networking parlance as resource reservation. Once resources are reserved, the call has on-demand access to these resources throughout its duration, regardless of the demands of all other calls. If a call reserves and receives a guarantee of x Mbps of link bandwidth, and never transmits at a rate greater than x, the call will see loss- and delay-free performance.

Call admission: If resources are to be reserved, then the network must have a mechanism for calls to request and reserve resources. Since resources are not infinite, a call making a call admission request will be denied admission, that is, be blocked, if the requested resources are not available. Such a call admission is performed by the telephone network—we request resources when we dial a number. If the circuits (TDMA slots) needed to complete the call are available, the circuits are allocated and the call is completed. If the circuits are not available, then the call is blocked, and we receive a busy signal. A blocked call can try again to gain admission to the network, but it is not allowed to send traffic into the network until it has successfully completed the call admission process. Of course, a router that allocates link bandwidth should not allocate more than is available at that link. Typically, a call may reserve only a fraction of the link's bandwidth, and so a router may allocate link bandwidth to more than one call. However, the sum of the allocated bandwidth to all calls should be less than the link capacity if hard quality of service guarantees are to be provided.

Call setup signaling: The call admission process described above requires that a call be able to reserve sufficient resources at each and every network router on its source-to-destination path to ensure that its end-to-end QoS requirement is met. Each router must determine the local resources required by the session, consider the amounts of its resources that are already

committed to other ongoing sessions, and determine whether it has sufficient resources to satisfy the per-hop QoS requirement of the session at this router without violating local QoS guarantees made to an already-admitted session. A signaling protocol is needed to coordinate these various activities—the per-hop allocation of local resources, as well as the overall end-to-end decision of whether or not the call has been able to reserve sufficient resources at each and every router on the end-to-end path. The RSVP protocol was proposed for this purpose within an Internet architecture for providing qualityof- service guarantees.