BMS
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
Avalahalli, Doddaballapur Main Road, Bengaluru – 560064

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Lecture Notes

Course

COMPUTER NETWORKS AND SECURITY

18CS52

2020-2021

Table of Contents

Syllabus

Module 1	Contact Hours
Application Layer: Principles of Network Applications: Network Application Architectures, Processes Communicating, Transport Services Available to Applications, Transport Services Provided by the Internet, Application-Layer Protocols. The Web and HTTP: Overview of HTTP, Non-persistent and Persistent Connections, HTTP Message Format, User-Server Interaction: Cookies, Web Caching, The Conditional GET, File Transfer: FTP Commands & Replies, Electronic Mail in the Internet: SMTP, Comparison with HTTP, Mail Message Format, Mail Access Protocols, DNS; The Internet's Directory Service: Services Provided by DNS, Overview of How DNS Works, DNS Records and Messages, Peer-to-Peer Applications: P2P File Distribution, Distributed Hash Tables, Socket Programming: creating Network Applications: Socket Programming with UDP, Socket Programming with TCP.	10
Module 2	
Transport Layer : Introduction and Transport-Layer Services: Relationship Between Transport and Network Layers, Overview of the Transport Layer in the Internet,Multiplexing and Demultiplexing: Connectionless Transport: UDP,UDP Segment Structure,UDP Checksum, Principles of Reliable Data Transfer: Building a Reliable Data Transfer Protocol, Pipelined Reliable Data Transfer Protocols, Go-Back-N, Selective repeat,Connection-Oriented Transport TCP: The TCP Connection, TCP Segment Structure, Round-Trip Time Estimation and Timeout, Reliable Data Transfer, Flow Control, TCP Connection Management, Principles of Congestion Control: The Causes and the Costs of Congestion, Approaches to Congestion Control, Network-assisted congestion-control example, ATM ABR Congestion control, TCP Congestion Control: Fairness.	10
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, BroadcastRouting Algorithms and Multicast.	10
Module 4	
Network Security: Overview of Network Security:Elements of Network Security , Classification of Network Attacks ,Security Methods ,Symmetric-Key Cryptography :Data Encryption Standard (DES),Advanced Encryption Standard (AES) , Public-Key Cryptography :RSA Algorithm ,Diffie-Hellman Key-Exchange Protocol , Authentication :Hash Function , Secure Hash Algorithm (SHA) , Digital Signatures , Firewalls and Packet Filtering ,Packet Filtering , Proxy Server .	10
Module 5	
Multimedia Networking: Properties of video, properties of Audio, Types of multimedia Network Applications, Streaming stored video: UDP Streaming, HTTP Streaming, Adaptive streaming and DASH, content distribution Networks Voice-over-IP :Limitations of the Best-Effort IP Service ,Removing Jitter at the Receiver for Audio ,Recovering from Packet Loss Protocols for Real-Time Conversational Applications ,RTP , SIP	10
Textbooks: 1. James F Kurose and Keith W Ross, Computer Networking, A Top-Down Approach, Sixth edition,Pearson,2017 . 2. Nader F Mir, Computer and Communication Networks, 2nd Edition, Pearson, 2014.	

Module – 1

APPLICATION LAYER

1.1 Principles of Network Applications

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

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 and the Web server program running in the Web server host.

1.1.1 Network Application Architectures.

There are two different network application architecture, they are

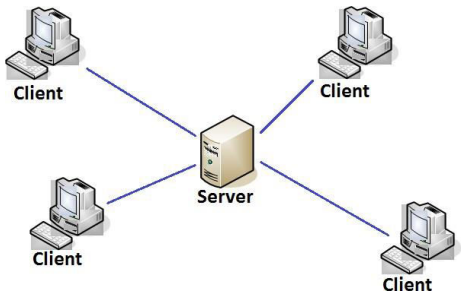
1) Client Server Architecture

2) P2P Architecture

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.
- In client-server architecture, clients do not directly communicate with each other.
- 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.



Client Server Architecture

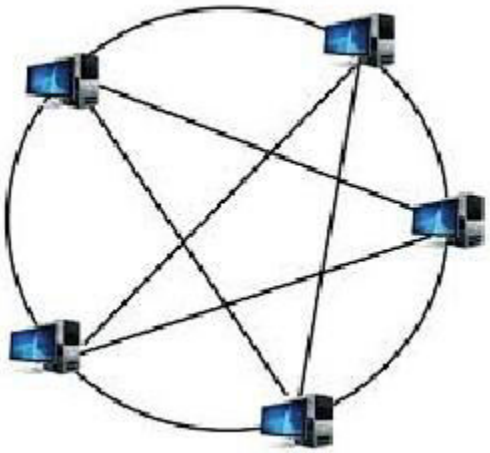
- In a client-server application, a single-server host is incapable of keeping up with all the requests from clients. For this reason, a data center, housing a large number of hosts, is often used to create a powerful virtual server.
- The most popular Internet services—such as search engines (e.g., 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.

Peer-to-peer (P2P) 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.
- Many of today's 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



P2P 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.

1.1.2 Processes Communicating

- A Process is a program or application under execution.
- When processes are running on the same or 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.

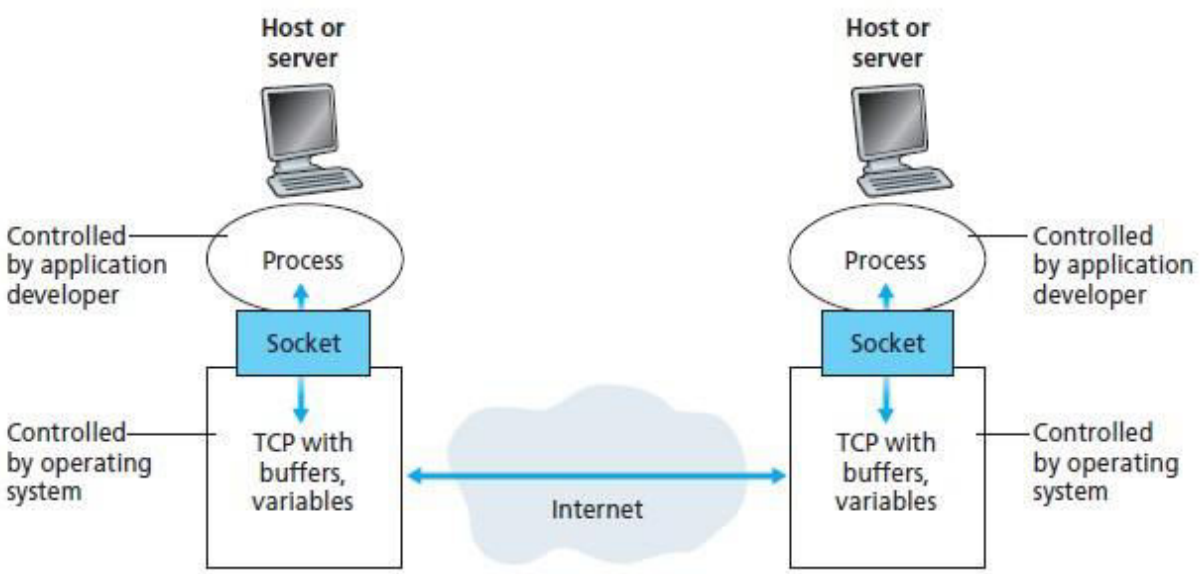
Client and Server Processes

- A network application consists of pairs of processes that send messages to each other over a network.

  For example, in the Web application a client browser process exchanges messages with a Web server process.
- In the context of a communication session between a pair of processes, the process that initiates the communication is labeled as the client. The process that waits to be contacted to begin the session is the server.

The Interface between the Process and the Computer Network

- A process sends messages into, and receives messages from, the network through a software interface called a socket.
- 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 at the sending side pushes messages through the socket. At the other side of the socket, the transport-layer protocol has the responsibility of getting the messages to the socket of the receiving process.



Application processes, sockets, and underlying transport protocol

Addressing Processes

- 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.
- 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.

1.1.3 Transport Services Available to Applications

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.
- If a protocol provides such a guaranteed data delivery service, it is said to provide reliable data transfer. One important service that a transport-layer protocol can potentially provide to an application is process-to-process reliable data transfer.
- When a transport protocol provides this service, the sending process can just pass its data into the socket and know with complete confidence that the data will arrive without errors at the receiving process.
- When a transport-layer protocol doesn't provide reliable data transfer, some of the data sent by the sending process may never arrive at the receiving process. 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

- Transport-layer protocol could provide guaranteed available throughput at some specified rate.
- With such a service, the application could request a guaranteed throughput of r bits/sec, and the transport protocol would then ensure that the available throughput is always at least r bits/sec. Such a guaranteed throughput service would appeal to many applications.
  For example, if an Internet telephony application encodes voice at 32 kbps, it needs to send data into the network and have data delivered to the receiving application at this rate.
- If the transport protocol cannot provide this throughput, the application would need to encode at a lower rate or may have to give up.
- Applications that have throughput requirements are said to be bandwidth-sensitive applications. Many current multimedia applications are bandwidth sensitive
- Elastic applications can make use of as much, or as little, throughput as happens to be available. Electronic mail, file transfer, and Web transfers are all elastic applications.

3) Timing

- A transport-layer protocol can also provide timing guarantees.
- Interactive real-time applications, such as Internet telephony, virtual environments, teleconferencing, and multiplayer games require tight timing constraints on data delivery in order to be effective.

4) Security

- Transport protocol can provide an application with one or more security services.
  For example, in the sending host, a transport protocol can encrypt all data transmitted by the sending process, and in the receiving host, the transport-layer protocol can decrypt the data before delivering the data to the receiving process.
- A transport protocol can provide security services like confidentiality, data integrity and end-point authentication.

1.1.4 Transport Services Provided by the Internet

The Internet makes two transport protocols available to applications, UDP and TCP.

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–1Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

Requirements of selected network applications

TCP Services

The TCP service model includes a connection-oriented service and a reliable data transfer service.

1) Connection-oriented service:

- In TCP the client and server exchange transport layer control information with each other before the application-level messages begin to flow.
- This handshaking procedure alerts the client and server, allowing them to prepare for an onslaught of packets.
- After the handshaking phase, a TCP connection is said to exist between the sockets of the two processes.
- The connection is a full-duplex connection in that the two processes can send messages to each other over the connection at the same time.
- When the application finishes sending messages, it must tear down the connection.

2) Reliable data transfer service:

- The communicating processes can rely on TCP to deliver all data sent without error and in the proper order.

- When one side of the application passes a stream of bytes into a socket, it can count on TCP to deliver the same stream of bytes to the receiving socket, with no missing or duplicate bytes.

TCP also includes a congestion-control mechanism.

UDP Services

- UDP is connectionless, so there is no handshaking before the two processes start to communicate.
- UDP provides an unreliable data transfer service—that is, when a process sends a message into a UDP socket, UDP provides no guarantee that the message will ever reach the receiving process.
- UDP does not include a congestion-control mechanism, so the sending side of UDP can pump data into the layer below (the network layer) at any rate it pleases.

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

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

1.1.5 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.

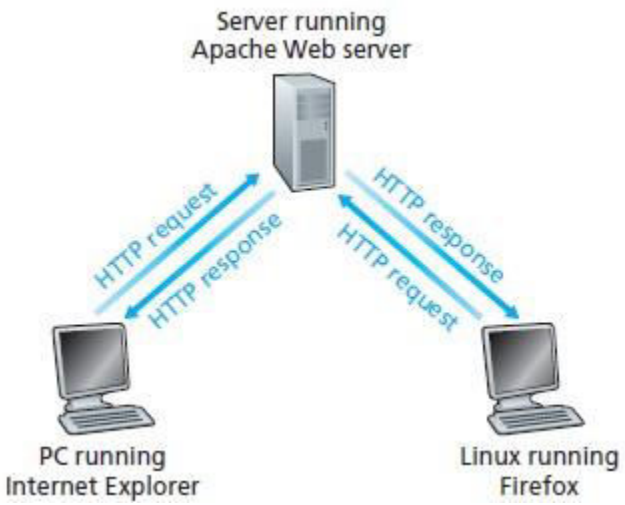
1.2 The Web and HTTP

1.2.1 Overview of HTTP

- The Hyper Text Transfer Protocol (HTTP), the Web's application-layer protocol, is at the heart of the Web.
- HTTP is implemented in two programs: a client program and a 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.
- A Web page consists of objects. An object is simply a file like HTML file, a JPEG image, a Java applet, or a video clip—that is addressable by a single URL.
- Most Web pages consist of a base HTML file and several referenced objects. For example, if a Web page contains HTML text and five JPEG images, then the Web page has six objects: the base HTML file plus the five images.
- The base HTML file references the other objects in the page with the objects' URLs. Each URL has two components: the hostname of the server that houses the object and the object's path name.

  For example, the **URL http://www.google.in/home/picture.gif** has **www.google.in** for a hostname and **/home/picture.gif** for a path name.
- HTTP defines how Web clients request Web pages from Web servers and how servers transfer Web pages to clients.
- When a user requests a Web page (for example, clicks on a hyperlink), the browser sends HTTP request messages for the objects in the page to the server. The server receives the requests and responds with HTTP response messages that contain the objects.
- HTTP uses TCP as its underlying transport protocol. The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces.

- It is important to note that the server sends requested files to clients without storing any state information about the client. If a particular client asks for the same object twice in a period of a few seconds, the server does not respond by saying that it just served the object to the client; instead, the server resends the object, as it has completely forgotten what it did earlier. Because an HTTP server maintains no information about the clients, HTTP is said to be a stateless protocol.



1.2.2 Non-Persistent and Persistent Connections

If Separate TCP connection is used for each request and response, then the connection is said to be non persistent. If same TCP connection is used for series of related request and response, then the connection is said to be persistent.

HTTP with Non-Persistent Connections

Let's suppose the page consists of a base HTML file and 10 JPEG images, and that all 11 of these objects reside on the same server.

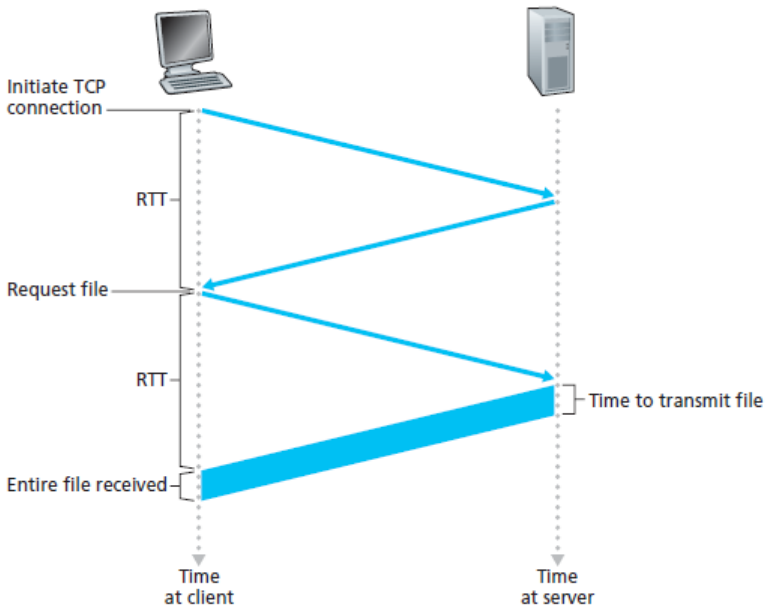
Further suppose the URL for the base HTML file is

http://www.someSchool.edu/someDepartment/home.index

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.
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.
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.



- Round-trip time (RTT) is the time it takes for a small packet to travel from client to server and then back to the client.
- The RTT includes packet-propagation delays, packet queuing delays in intermediate routers and switches, and packet-processing delays.
- When a user clicks on a hyperlink, the browser 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.

- 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.

HTTP with Persistent Connections

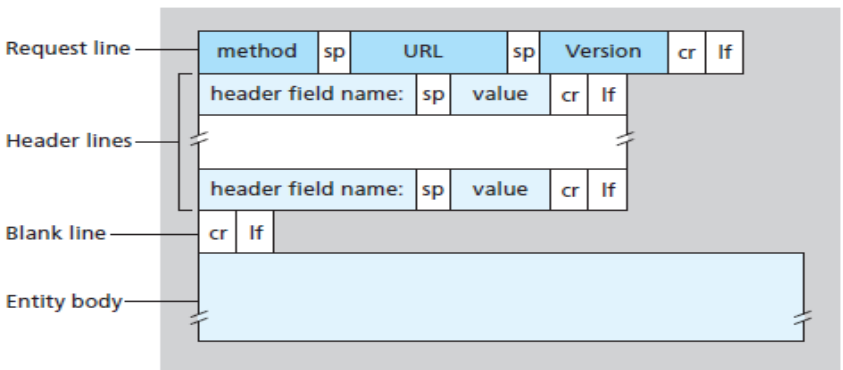
Non-persistent connections have some shortcomings.

1. A brand-new connection must be established and maintained for each requested object. For each of these connections, TCP buffers must be allocated and TCP variables must be kept in both the client and server. This can place a significant burden on the Web server, which may be serving requests from hundreds of different clients simultaneously.
2. Each object suffers a delivery delay of two RTTs— one RTT to establish the TCP connection and one RTT to request and receive an object.

With persistent connections, the server leaves the TCP connection open after sending a response. Subsequent requests and responses between the same client and server can be sent over the same connection. In particular, an entire Web page can be sent over a single persistent TCP connection. Moreover, multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection.

1.2.3 HTTP Message Format

HTTP Request Message:



Where sp – space, cr – carriage return and lf – line feed.

Method:

There are five HTTP methods:

- **GET:** The GET method is used when the browser requests an object, with the requested object identified in the URL field.
- **POST:** With a POST message, the user is still requesting a Web page from the server, but the specific contents of the Web page depend on what the user entered into the form fields. If the value of the method field is POST, then the entity body contains what the user entered into the form fields.
- **PUT:** The PUT method is also used by applications that need to upload objects to Web servers.
- **HEAD:** Used to retrieve header information. It is used for debugging purpose.
- **DELETE:** The DELETE method allows a user, or an application, to delete an object on a Web server.

URL: Specifies URL of the requested object

Version: This field represents HTTP version, usually HTTP/1.1

Header line:

Ex:

Host: www.someschool.edu

Connection: close

User-agent: Mozilla/5.0

Accept-language: fr

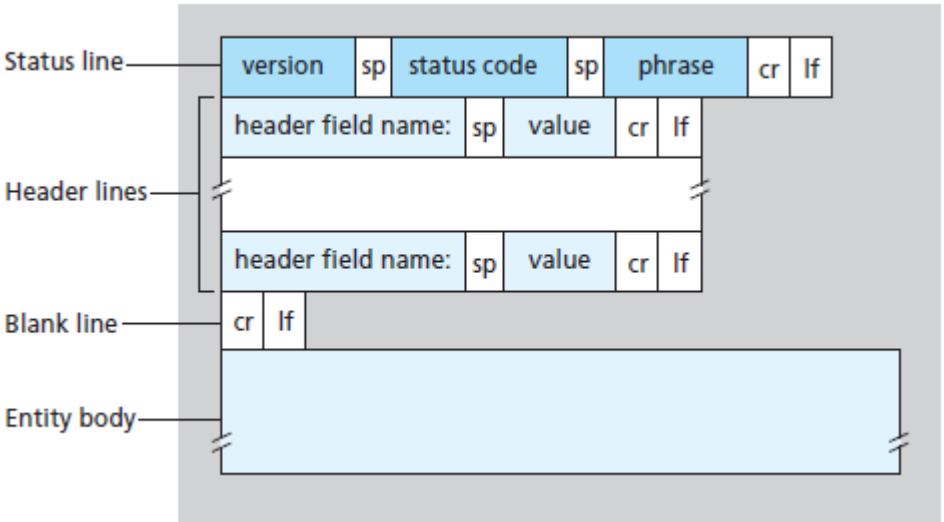
The header line **Host:www.someschool.edu** specifies the host on which the object resides.

By including the **Connection:close** header line, the browser is telling the server that it doesn't want to bother with persistent connections; it wants the server to close the connection after sending the requested object.

The **User-agent:** header line specifies the user agent, that is, the browser type that is making the request to the server. Here the user agent is Mozilla/5.0, a Firefox browser.

The **Accept-language:** header indicates that the user prefers to receive a French version of the object, if such an object exists on the server; otherwise, the server should send its default version.

HTTP Response Message



Ex:

```
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 **status line** has three fields: the protocol version field, a status code, and a corresponding status message.

Version is HTTP/1.1

The status code and associated phrase indicate the result of the request. Some common status codes and associated phrases include:

• 200 OK: Request succeeded and the information is returned in the response.

• 301 Moved Permanently: Requested object has been permanently moved; the new URL is specified in Location: header of the response message. The client software will automatically retrieve the new URL.

• 400 Bad Request: This is a generic error code indicating that the request could not be understood by the server.

• 404 Not Found: The requested document does not exist on this server.

• 505 HTTP Version Not Supported: The requested HTTP protocol version is not supported by the server.

Header fields:

- The server uses the **Connection: close** header line to tell the client that it is going to close the TCP connection after sending the message.
- The **Date:** header line indicates the time and date when the HTTP response was created and sent by the server.
- The **Server:** header line indicates that the message was generated by an Apache Web server; it is analogous to the User-agent: header line in the HTTP request message.
- The **Last-Modified:** header line indicates the time and date when the object was created or last modified.
- The **Content-Length:** header line indicates the number of bytes in the object being sent.
- The **Content-Type:** header line indicates that the object in the entity body is HTML text.

1.2.4 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.

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.

Ex:

Suppose a user, who always accesses the Web using Internet Explorer from her home PC, contacts Amazon.com for the first time. Let us suppose that in the past he has already visited the eBay site. When the request comes into the Amazon Web server, the server creates a unique identification number and creates an entry in its back-end database that is indexed by the identification number. The Amazon Web server then responds to Susan's browser, including in the HTTP response a Set-cookie: header, which contains the identification number.

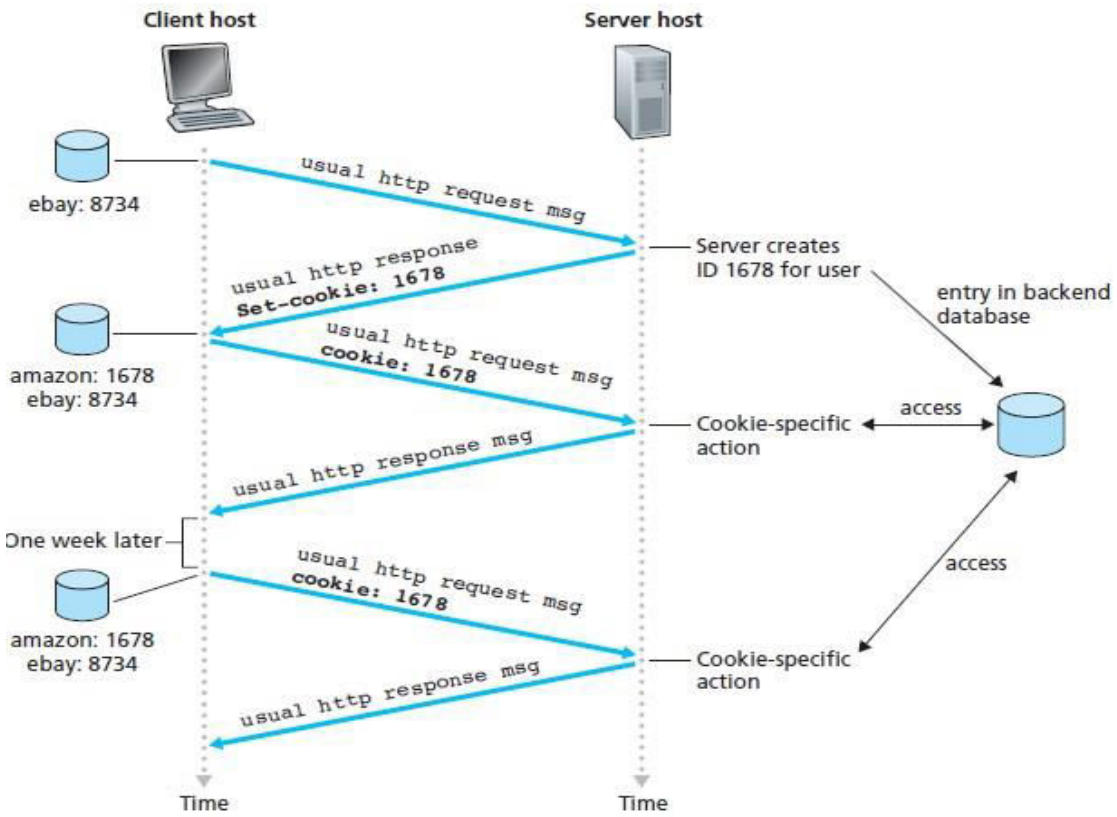
For example, the header line might be:

Set-cookie: 1678

When users browser receives the HTTP response message, it sees the Set-cookie: header. The browser then appends a line to the special cookie file that it manages. This line includes the hostname of the server and the identification number in the Set-cookie: header.

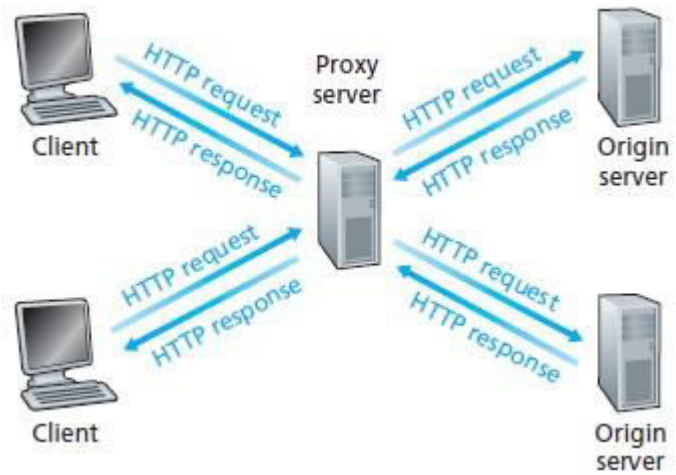
As user continues to browse the Amazon site, each time he requests a Web page, his browser consults his cookie file, extracts his identification number for this site, and puts a cookie header line that includes the identification number in the HTTP request. Specifically, each of his HTTP requests to the Amazon server includes the header line:

Cookie: 1678



1.2.5 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 its own disk storage and keeps copies of recently requested objects in this storage.
- A user's browser can be configured so that all of the user's HTTP requests are first directed to the Web cache.



Ex: Suppose a browser is requesting the object http://www.someschool.edu/campus.gif. Here is what happens:

1. The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.
2. The Web cache checks to see if it has a copy of the object stored locally. If it does, the Web cache returns the object within an HTTP response message to the client browser.
3. If the Web cache does not have the object, the Web cache opens a TCP connection to the origin server, that is, to www.someschool.edu. The Web cache then sends an HTTP request for the object into the cache-to-server TCP connection.
4. After receiving this request, the origin server sends the object within an HTTP response to the Web cache.
5. When the Web cache receives the object, it stores a copy in its local storage and sends a copy, within an HTTP response message, to the client browser (over the existing TCP connection between the client browser and the Web cache).

- When web cache receives requests from and sends responses to a browser, it is a server. When it sends requests to and receives responses from an origin server, it is a client.
- 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.
- Web caching has seen deployment in the Internet for two reasons. First, a Web cache can substantially reduce the response time for a client request. Second, Web caches can substantially reduce traffic on an institution's access link to the Internet.

1.2.6 The Conditional GET

- Although caching can reduce user-perceived response times, it introduces a new problem—the copy of an object residing in the cache may be stale. In other words, the object housed in the Web server may have been modified since the copy was cached at the client.
- HTTP has a mechanism that allows a cache to verify that its objects are up to date. This mechanism is called the conditional GET.
- An HTTP request message is a so-called conditional GET message if (1) the request message uses the GET method and (2) the 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:

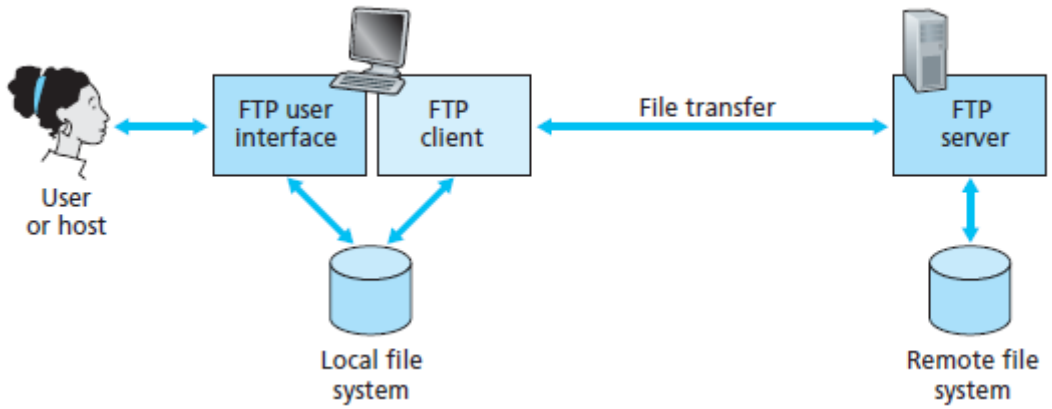
```
HTTP/1.1 304 Not Modified
Date: Sat, 15 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
(empty entity body)
```

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.

1.3 File Transfer: FTP

- 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).



- 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.

- When a user starts an FTP session with a remote host, the client side of FTP (user) first initiates a control TCP connection with the server side (remote host) on server port number 21.
- The client side of FTP sends the user identification and password over this control connection. The client side of FTP also sends, over the control connection, commands to change the remote directory.
- When the server side receives a command for a file transfer over the control connection (either to, or from, the remote host), the server side initiates a TCP data connection to the client side.

- FTP sends exactly one file over the data connection and then closes the data connection. If, during the same session, the user wants to transfer another file, FTP opens another data connection.
- Thus, with FTP, the control connection remains open throughout the duration of the user session, but a new data connection is created for each file transferred within a session (that is, the data connections are non-persistent).
- Throughout a session, the FTP server must maintain state about the user. In particular, the server must associate the control connection with a specific user account, and the server must keep track of the user's current directory as the user wanders about the remote directory tree.

1.3.1 FTP Commands and Replies

Some of the more common commands are given below:

• USER username: Used to send the user identification to the server.

• PASS password: Used to send the user password to the server.

• LIST: Used to ask the server to send back a list of all the files in the current remote directory. The list of files is sent over a (new and non-persistent) data connection rather than the control TCP connection.

• RETR filename: Used to retrieve (that is, get) a file from the current directory of the remote host. This command causes the remote host to initiate a data connection and to send the requested file over the data connection.

• STOR filename: Used to store (that is, put) a file into the current directory of the remote host.

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.

• 331 Username OK, password required

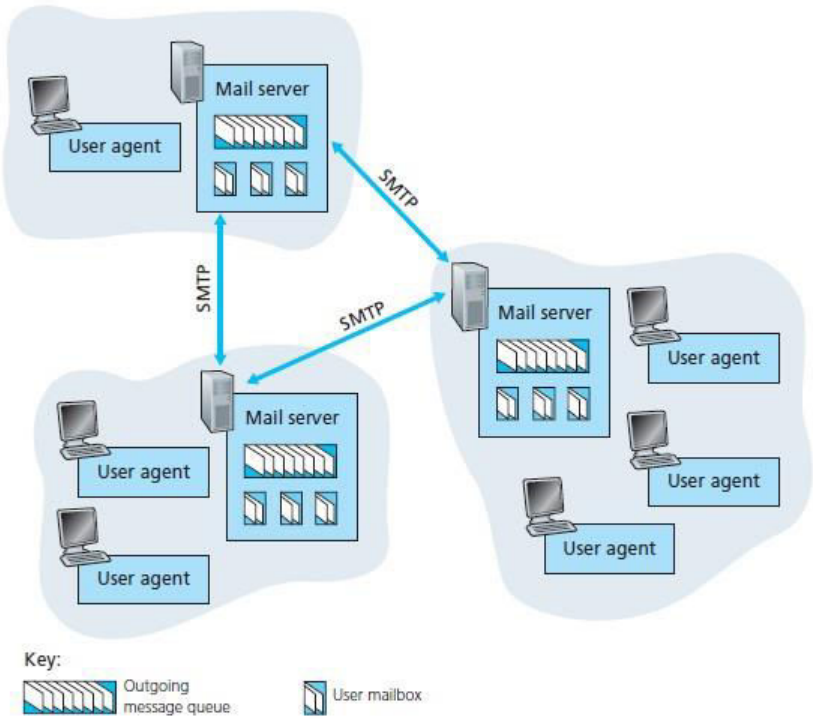
• 125 Data connection already open; transfer starting

• 425 Can't open data connection

• 452 Error writing file

1.4 Electronic Mail in the Internet

E-mail has three major components: user agents, mail servers, and the Simple Mail Transfer Protocol (SMTP).



- **User agents** allow users to read, reply to, forward, save, and compose messages.
- **Mail servers** form the core of the e-mail infrastructure. Each recipient has a mailbox located in one of the mail servers. A typical message starts its journey in the sender's user agent, travels to the sender's mail server, and travels to the recipient's mail server, where it is deposited in the recipient's mailbox.
- **SMTP** is the principal application-layer protocol for Internet electronic mail. It uses the reliable data transfer service of TCP to transfer mail from the sender's mail server to the recipient's mail server. As with most application-layer protocols, SMTP has two sides: a client side, which executes on the sender's mail server, and a server side, which executes on the recipient's mail server.

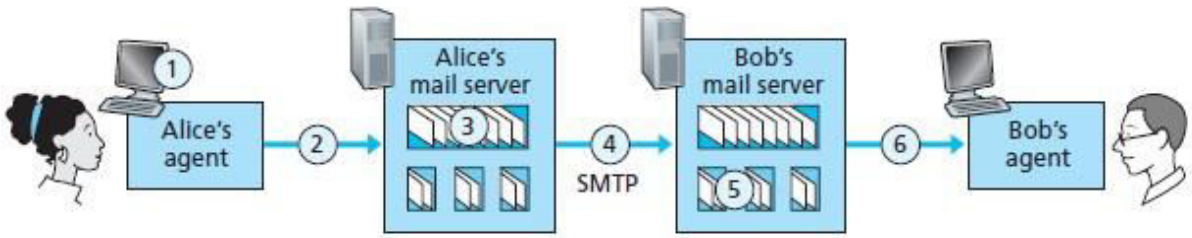
1.4.1 SMTP

SMTP transfers messages from senders' mail servers to the recipients' mail servers. It restricts the body (not just the headers) of all mail messages to simple 7-bit ASCII.

Suppose Alice wants to send Bob a simple ASCII message.

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.



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 Hello 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

1.4.2 Comparison 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.

1.4.3 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.
```

1.4.4 Mail Access Protocols

SMTP protocol delivers the mail to the mail server. To fetch the mail from mail server receiver used mail access protocols.

There are currently a number of popular mail access protocols, including Post Office Protocol—Version 3 (POP3), Internet Mail Access Protocol (IMAP), and HTTP.

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 dele commands.

Ex:

```
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.

IMAP

- 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.

Web-Based E-Mail

More and more users today are sending and accessing their e-mail through their Web browsers. In this case user communicates with its remote mailbox via HTTP.

1.5 DNS—The Internet's Directory Service

- All the hosts connected to network is identified by IP address. But it is difficult for human beings to remember these IP address to access a particular host. Hence hosts are identified by hostnames. Ex: google.com
- But the routers require IP address to forward the packet.
- In order to map hostname with the IP address DNS is used.

1.5.1 Services Provided by DNS

- The DNS is (1) a distributed database implemented in a hierarchy of DNS servers, and (2) an application-layer protocol that allows hosts to query the distributed database.

- DNS is commonly employed by other application-layer protocols—including HTTP, SMTP, and FTP—to translate user-supplied hostnames to IP addresses.

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:

- **Host aliasing:** A host with a complicated hostname can have one or more alias names. For example, a hostname such as relay1.west-coast.enterprise.com could have, say, two aliases such as enterprise.com and www.enterprise.com. In this case, the hostname relay1.westcoast. enterprise.com is said to be a **canonical hostname.** Alias hostnames, when present, are typically more mnemonic than canonical hostnames. DNS can be invoked by an application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.
- **Mail server aliasing:** For obvious reasons, it is highly desirable that e-mail addresses be mnemonic. For example, if Bob has an account with Hotmail, Bob's e-mail address might be as simple as bob@hotmail.com. However, the 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). 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.

- **Load distribution:** DNS is also used to perform load distribution among replicated servers, such as replicated Web servers. Busy sites, such as cnn.com, are replicated over multiple servers, with each server running on a different end system and each having a different IP address. For replicated Web servers, a set of IP addresses is thus associated with one canonical hostname. The DNS database contains this set of IP addresses. When clients make a DNS query for a name mapped to a set of addresses, the server responds with the entire set of IP addresses, but rotates the ordering of the addresses within each reply. Because a client typically sends its HTTP request message to the IP address that is listed first in the set, DNS rotation distributes the traffic among the replicated servers.

1.5.2 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:

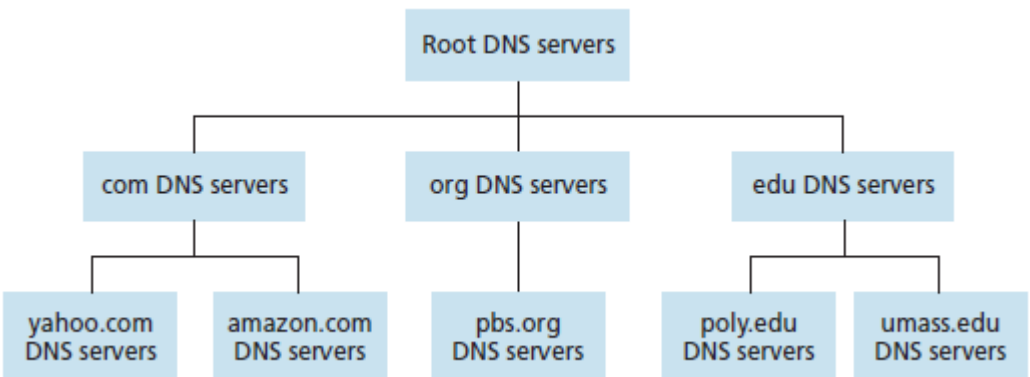
- A single point of failure. If the DNS server crashes, so does the entire Internet!
- Traffic volume. A single DNS server would have to handle all DNS queries.
- Distant centralized database. A single DNS server cannot be "close to" all the querying clients. If we put the single DNS server in New York City, then all queries from Australia

must travel to the other side of the globe, perhaps over slow and congested links. This can lead to significant delays.

- Maintenance. The single DNS server would have to keep records for all Internet hosts. Not only would this centralized database be huge, but it would have to be updated frequently to account for every new host.

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.



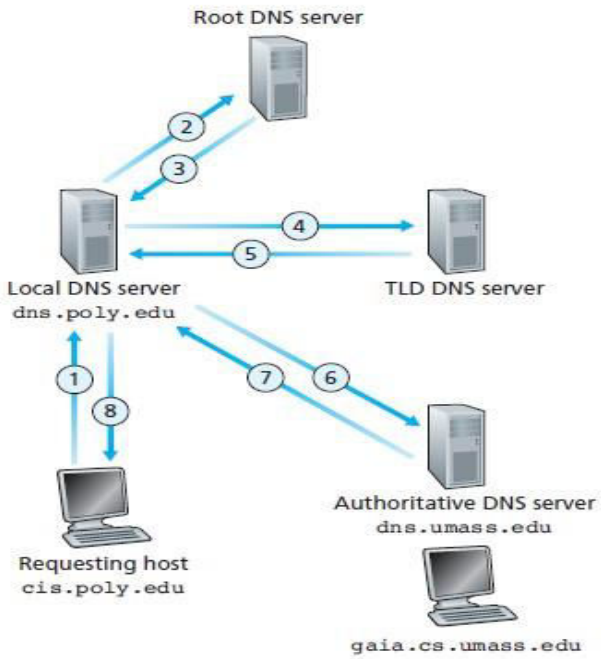
- **Root DNS servers.** In the Internet there are 13 root DNS servers (labeled A through M), most of which are located in North America.

  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. All together, there are 247 root servers.
- **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.
- **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.

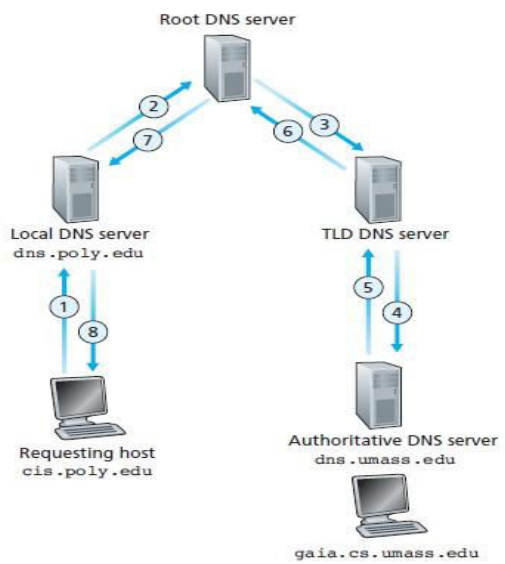
Two type of Interaction:

1) Recursive Queries:



Here DNS query is sent to local DNS server then to root server, then to TLD server and finally to authoritative DNS server. DNS response arrives in the reverse order.

2) Iterative Queries:



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.

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).

1.5.3 DNS Records and Messages

The DNS servers that together implement the DNS distributed database store **resource records** (RRs).

A resource record is a four-tuple that contains the following fields:

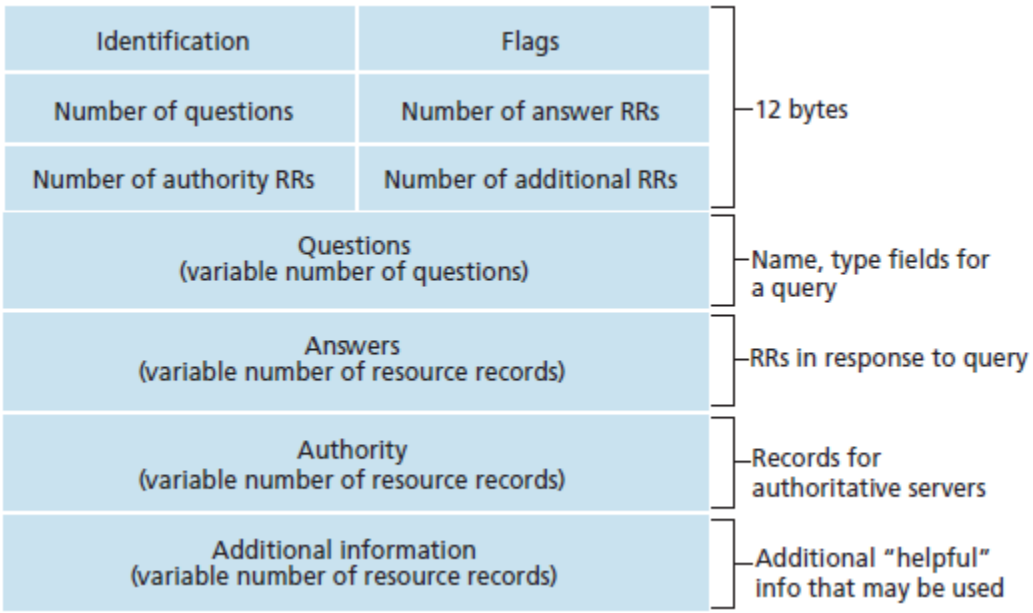
(Name, Value, Type, TTL)

TTL is the time to live of the resource record; it determines when a resource should be removed from a cache.

The meaning of Name and Value depend on Type:

- If Type=A, then Name is a hostname and Value is the IP address for the hostname.
- If Type=NS, then Name is a domain (such as foo.com) and Value is the hostname of an authoritative DNS server that knows how to obtain the IP addresses for hosts in the domain.
- 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.
- If Type=MX, then Value is the canonical name of a mail server that has an alias hostname Name.

DNS Messages



- The first 12 bytes is the header section, which has a number of fields.
- The first field is a 16-bit number that identifies the query. This identifier is copied into the reply message to a query, allowing the client to match received replies with sent queries.
- There are a number of flags in the flag field.

  A 1-bit query/reply flag indicates whether the message is a query (0) or a reply (1). A1-bit authoritative flag is set in a reply message when a DNS server is an authoritative server for a queried name.

  A 1-bit recursion-desired flag is set when a client (host or DNS server) desires that the DNS server perform recursion when it doesn't have the record.

  A 1-bit recursion available field is set in a reply if the DNS server supports recursion.
- In the header, there are also four number-of fields. These fields indicate the number of occurrences of the four types of data sections that follow the header.
- The **question** section contains information about the query that is being made. This section includes (1) a name field that contains the name that is being queried, and (2) a type field that indicates the type of question being asked about the name
- In a reply from a DNS server, the **answer** section contains the resource records for the name that was originally queried.

- The **authority** section contains records of other authoritative servers.
- The **additional** section contains other helpful records.

Inserting Records into the DNS Database

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)

1.6 Peer-to-Peer Applications

In P2P architecture, there is minimal (or no) reliance on always-on infrastructure servers. Instead, pairs of intermittently connected hosts, called peers, communicate directly with each other.

1.6.1 P2P File Distribution

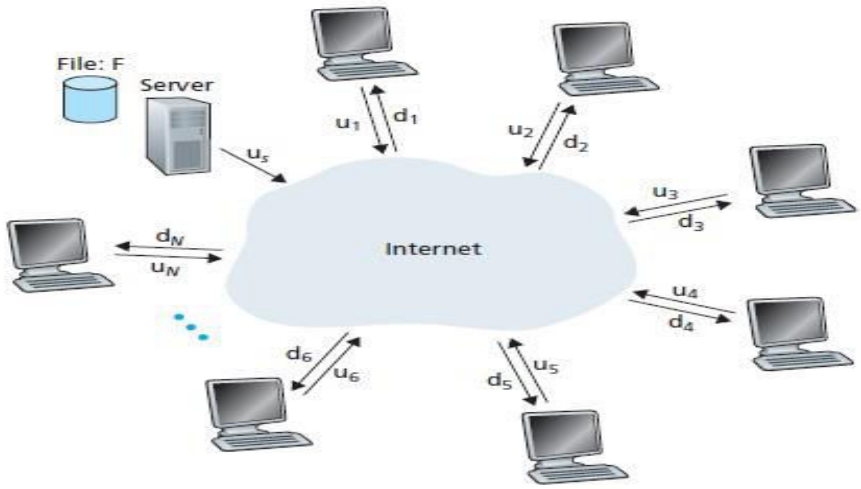
- In P2P file distribution, each peer can redistribute any portion of the file it has received to any other peers, thereby assisting the server in the distribution process.
- The most popular P2P file distribution protocol is BitTorrent.

Scalability of P2P Architectures

As shown in below Figure the server and the peers are connected to the Internet with access links. Denote the upload rate of the server's access link by u_s, the upload rate of the ith peer's access link by u_i, and the download rate of the ith peer's access link by d_i. Also denote the size of the file to be distributed (in bits) by F and the number of peers that want to obtain a copy of the

file by N.

The **distribution time** is the time it takes to get a copy of the file to all N peers.



In the client-server architecture, none of the peers aids in distributing the file. We make the following observations:

- The server must transmit one copy of the file to each of the N peers. Thus the server must transmit NF bits. Since the server's upload rate is us, the time to distribute the file must be at least NF/us.
- Let dmin denote the download rate of the peer with the lowest download rate, that is, dmin = min{d1,dp,...,dN}. The peer with the lowest download rate cannot obtain all F bits of the file in less than F/dmin seconds. Thus the minimum distribution time is at least F/dmin.

  Putting these two observations together, we obtain

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

In the P2P architecture we make the following observations:

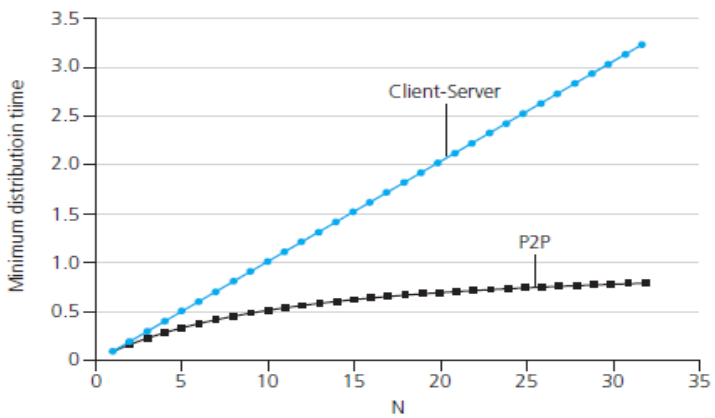
- At the beginning of the distribution, only the server has the file. To get this file into the community of peers, the server must send each bit of the file at least once into its access link. Thus, the minimum distribution time is at least F/us.
- As with the client-server architecture, the peer with the lowest download rate cannot obtain all F bits of the file in less than F/dmin seconds. Thus the minimum distribution time is at least F/dmin.

- Finally, observe that the total upload capacity of the system as a whole is equal to the upload rate of the server plus the upload rates of each of the individual peers, that is, utotal = us + u1 + … + uN. The system must deliver (upload) F bits to each of the N peers, thus delivering a total of NF bits. This cannot be done at a rate faster than utotal. Thus, the minimum distribution time is also at least NF/(us + u1 + … + uN).

Putting these three observations together, we obtain the minimum distribution time for P2P, denoted by D_{P2P}.

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

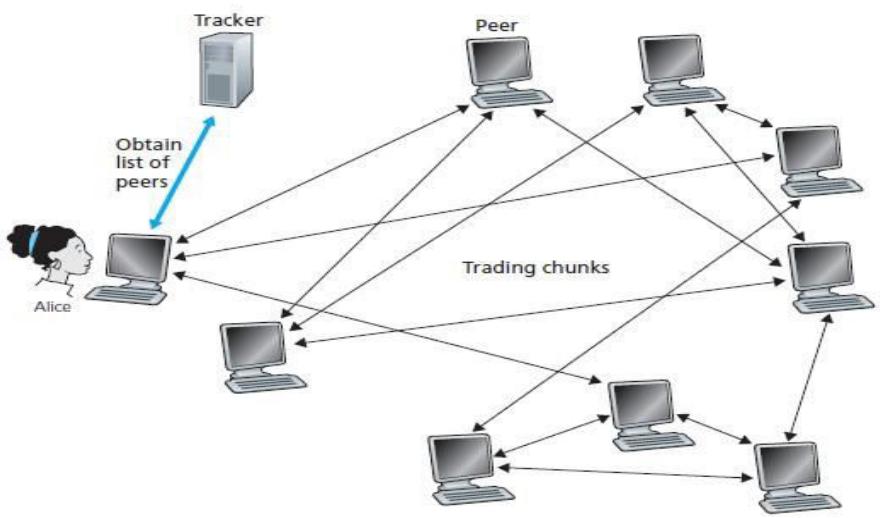
Below Figure compares the minimum distribution time for the client-server and P2P architectures assuming that all peers have the same upload rate u.



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, with a typical chunk size of 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, it 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 a tracker.
- When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent. In this manner, the tracker keeps track of the peers that are participating in the torrent.
- 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 new peer.
- Possessing this list of peers, new peer attempts to establish concurrent TCP connections with all the peers on this list. All the peers with which new peer succeeds in establishing a TCP connection will be called as "neighboring peers."
- As time evolves, some of these peers may leave and other peers (outside the initial 50) may attempt to establish TCP connections.
- Periodically, peer will ask each of its neighboring peers (over the TCP connections) for the list of the chunks they have. If peer has L different neighbors, it will obtain L lists of chunks. With this knowledge, peer will issue requests (again over the TCP connections) for chunks currently it does not have.
- In deciding which chunks to request, peer uses a technique called **rarest first**. The idea is to determine, from among the chunks peer does not have, the chunks that are the rarest among its neighbors and then request those rarest chunks first. In this manner, the rarest chunks get more quickly redistributed, aiming to equalize the numbers of copies of each chunk in the torrent.



- 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.

1.6.2 Distributed Hash Tables (DHTs)

- 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.

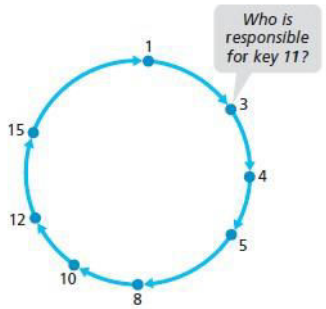
- Such an approach is completely unscalable as it would require each peer to know about all other peers and have each query sent to all peers.
- An elegant approach to designing a DHT is to first assign an identifier to each peer, where each identifier is an integer in the range $[0, 2^n-1]$ for some 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.

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 2^n).

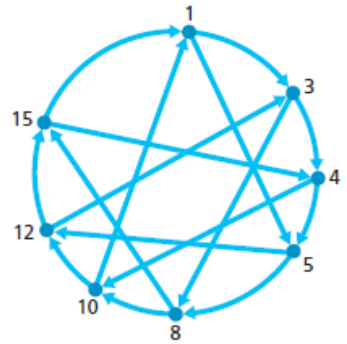


Each peer is only aware of its immediate successor and predecessor; for example, peer 5 knows the IP address and identifier for peers 8 and 4 but does not necessarily know anything about any other peers that may be in the DHT.

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 key 11. 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.

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.

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. 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. 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.

1.7 Socket Programming: Creating Network Applications

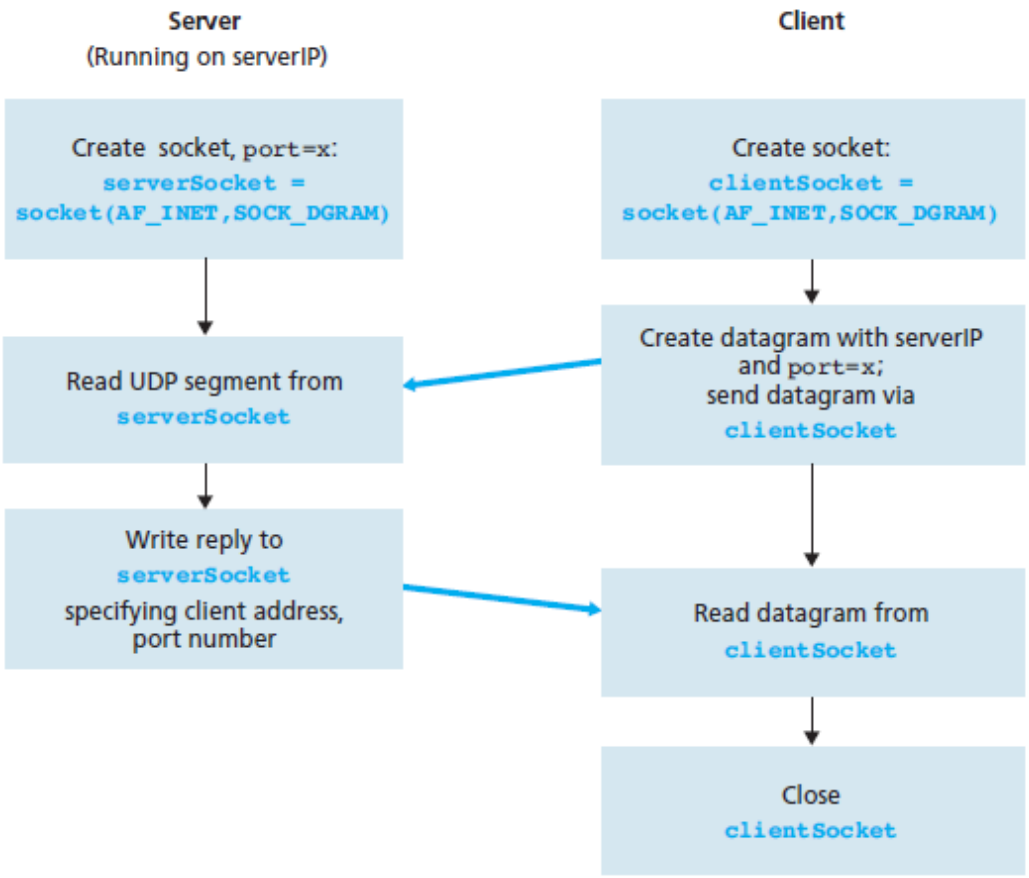
- A typical network application consists of a pair of programs—a client program and a server program—residing in two different end systems.
- When these two programs are executed, a client process and a server process are created, and these processes communicate with each other by reading from, and writing to, sockets.
- When creating a network application, the developer's main task is therefore to write the code for both the client and server programs.

1.7.1 Socket Programming with UDP

Before the sending process can push a packet of data out the socket door, when using UDP, it must first attach a destination address to the packet. After the packet passes through the sender's socket, the Internet will use this destination address to route the packet through the Internet to the socket in the receiving process. When the packet arrives at the receiving socket, the receiving process will retrieve the packet through the socket, and then inspect the packet's contents and take appropriate action.

Example application:

1. The client reads a line of characters (data) from its 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.


Server
(Running on serverIP)
Client
Create socket, port=x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)
Create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)
Create datagram with serverIP
and port=x;
send datagram via
clientSocket
Read UDP segment from
serverSocket
Write reply to
serverSocket
specifying client address,
port number
Read datagram from
clientSocket
Close
clientSocket

UDPClient.py

Here is the code for the client side of the application:

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

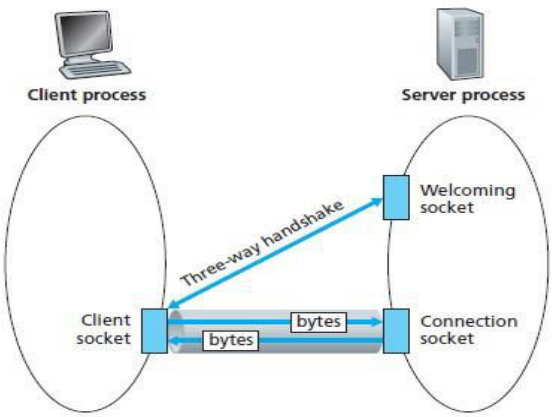
UDPServer.py

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
        message, clientAddress = serverSocket.recvfrom(2048)
        modifiedMessage = message.upper()
        serverSocket.sendto(modifiedMessage, clientAddress)
```

1.7.2 Socket Programming with TCP

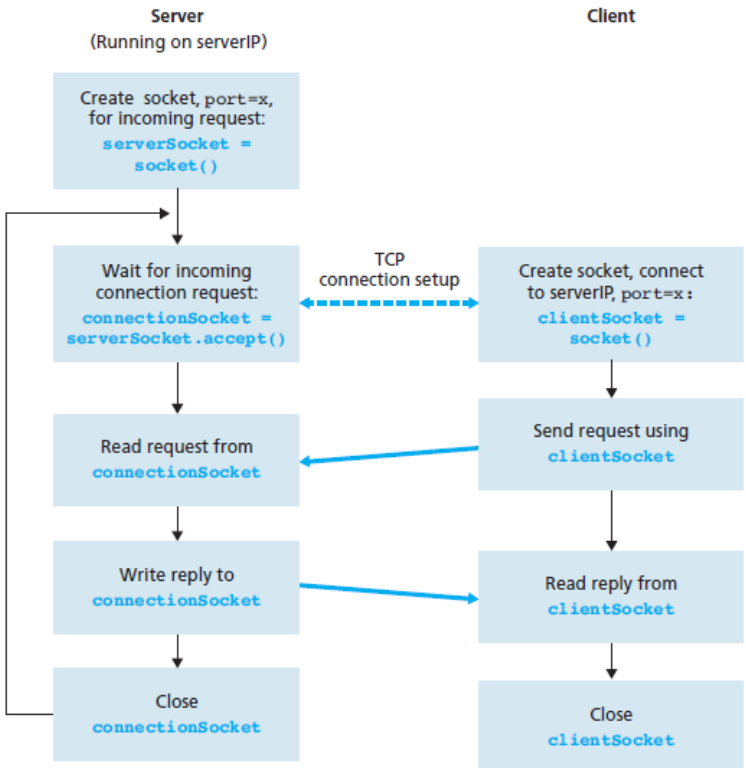
- Unlike UDP, TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection.

- 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.



TCPClient.py

```
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()
```



TCPServer.py

```
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()
```

Module – 2

TRANSPORT LAYER

2.1 Introduction and Transport-Layer Services

- A transport-layer protocol provides for logical communication between application processes running on different hosts.
- 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.
- 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.

2.1.1 Relationship between Transport and Network Layers

- Transport Layer provides Process to process delivery service whereas network layer provides end to end delivery of data.
- Transport-layer protocol provides logical communication between *processes* running on different hosts, a network-layer protocol provides logical communication between hosts.

- Within an end system, a transport protocol moves messages from application processes to the network edge (that is, the network layer) and vice versa, but it doesn't have any say about how the messages are moved within the network core.
- The services that a transport protocol can provide are often constrained by the service model of the underlying network-layer protocol. If the network-layer protocol cannot provide delay or bandwidth guarantees for transport layer segments sent between hosts, then the transport-layer protocol cannot provide delay or bandwidth guarantees for application messages sent between processes.

2.1.2 Overview of the Transport Layer in the Internet

The Internet supports two transport layer protocols:

1) UDP (User Datagram Protocol), which provides an unreliable, connectionless service to the invoking application.

2) TCP (Transmission Control Protocol), which provides a reliable, connection-oriented service to the invoking application.

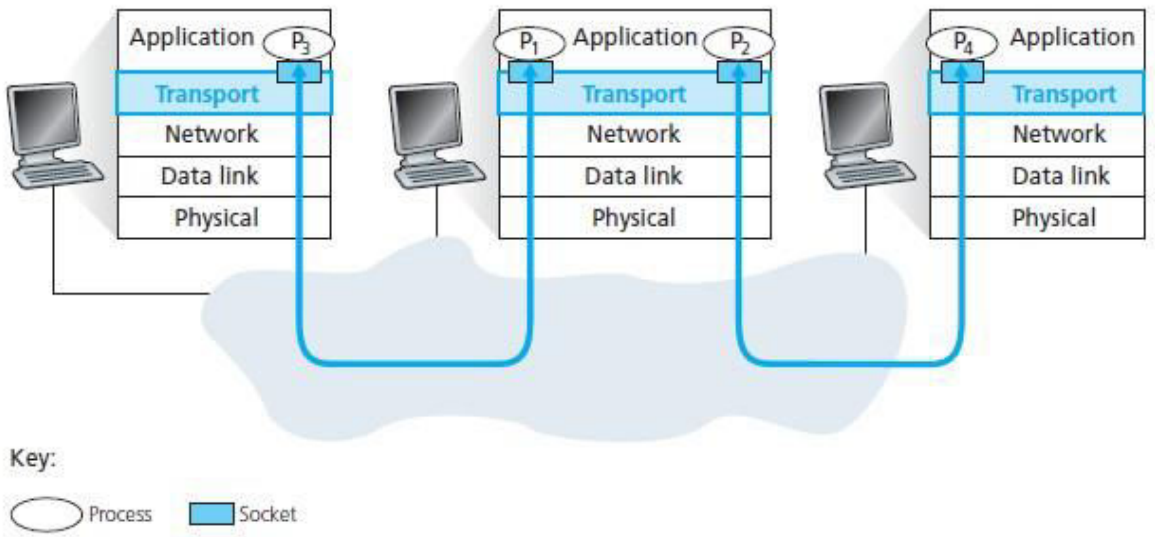
- The Internet's network-layer protocol has Internet Protocol. IP provides logical communication between hosts.
- The IP service model is a best-effort delivery service. This means that IP makes its "best effort" to deliver segments between communicating hosts, but it makes no guarantees. In particular, it does not guarantee segment delivery, it does not guarantee orderly delivery of segments, and it does not guarantee the integrity of the data in the segments.
- The most fundamental responsibility of UDP and TCP is to extend 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 also provide integrity checking by including error detection fields in their segments' headers.
- UDP is an unreliable service it does not guarantee that data sent by one process will arrive intact to the destination process.
- TCP, on the other hand, offers several additional services to applications. First and foremost, it 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.

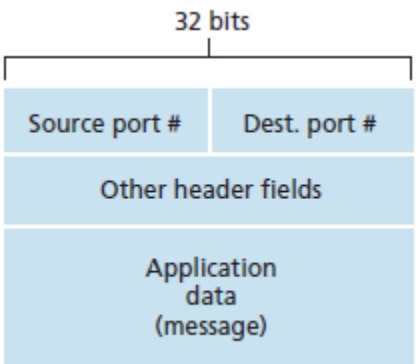
- TCP thus converts IP's unreliable service between end systems into a reliable data transport service between processes.
- TCP also provides congestion control. TCP congestion control prevents any one TCP connection from swamping the links and routers between communicating hosts with an excessive amount of traffic.
- UDP traffic, on the other hand, is unregulated. An application using UDP transport can send at any rate it pleases, for as long as it pleases.

2.2 Multiplexing and Demultiplexing

- At the destination host, the transport layer receives segments from the network layer just below.
- The transport layer has the responsibility of delivering the data in these segments to the appropriate application process running in the host.
- A process can have one or more sockets, doors through which data passes from the network to the process and through which data passes from the process to the network.
- The transport layer in the receiving host does not actually deliver data directly to a process, but instead to an intermediary socket.
- Because at any given time there can be more than one socket in the receiving host, each socket has a unique identifier.
- Each transport-layer segment has a set of fields in the segment to help receiver to deliver data to appropriate process socket.
- At the receiving end, the transport layer examines these fields to identify the receiving socket and then directs the segment to that socket. This job of delivering the data in a transport-layer segment to the correct socket is called **demultiplexing.**
- The job of 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.**



- Transport-layer multiplexing requires (1) that sockets have unique identifiers, and (2) that each segment 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.
- 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 and are restricted, which means that they are reserved for use by well-known application protocols such as HTTP (which uses port number 80) and FTP (which uses port number 21).



- UDP performs connectionless multiplexing and demultiplexing. TCP performs connection oriented multiplexing and demultiplexing.

2.3 Connectionless Transport: UDP

- UDP is a connectionless protocol which performs only multiplexing/demultiplexing function and some light 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.

Many applications are better suited for UDP 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 also 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 to establish the connection before it starts to transfer data. UDP just sends the data without any formal preliminaries. Thus UDP does not introduce any delay to establish a connection.

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, on the other hand, does not maintain connection state and does not track any of these parameters. For this reason, a server devoted to a particular application can typically support many more active clients when the application runs over UDP rather than TCP.

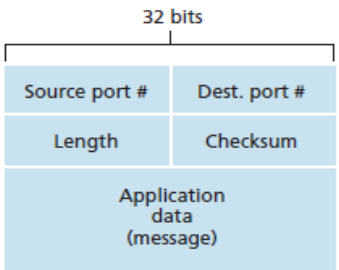
4) Small packet header overhead:

The TCP segment has 20 bytes of header overhead in every segment, whereas UDP has only 8 bytes of overhead.

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

Popular Internet applications and their underlying transport protocols

2.3.1 UDP Segment Structure



- 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
- The length field specifies the number of bytes in the UDP segment (header plus data).
- The checksum is used by the receiving host to check whether errors have been introduced into the segment.
- The application data occupies the data field of the UDP segment.

2.3.2 UDP Checksum

The checksum is used to determine whether bits within the UDP segment have been altered as it moved from source to destination.

Step1: Add all the data elements using binary addition (Modulo-2 addition). If you get extra bit wrap it.

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

Step 2: Take 1s complement of the result.

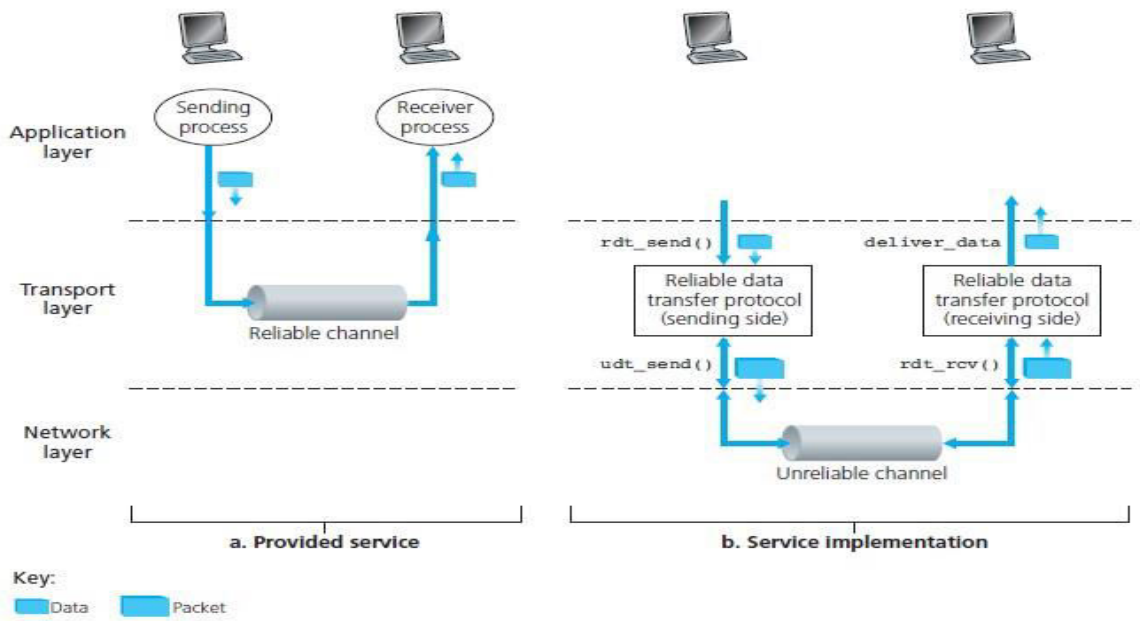
The 1s complement is obtained by converting all the 0s to 1s and converting all the 1s to 0s. Thus the 1s complement of the sum 0100101011000010 is 1011010100111101, which becomes the checksum.

Step 3: Data along with checksum is transmitted to receiver.

Step 4: at the receiver side add all the data and checksum using binary addition. Wrap the extra bit and take 1s complement of the result. This will be the checksum. If checksum is all 0's receiver has received error free data otherwise it has received corrupted data.

2.4 Principles of Reliable Data Transfer

The service abstraction provided to the upper-layer entities is that of a reliable channel through which data can be transferred. With a reliable channel, no transferred data bits are corrupted (flipped from 0 to 1, or vice versa) or lost, and all are delivered in the order in which they were sent.It is the responsibility of a reliable data transfer protocol to implement this service abstraction. This task is made difficult by the fact that the layer below the reliable data transfer protocol may be unreliable.

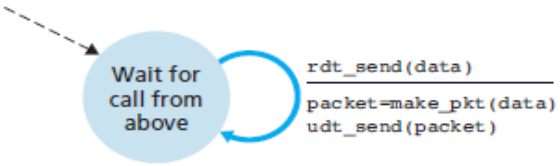


- The sending side of the data transfer protocol will be invoked from above by a call to rdt_send(). It will pass the data to be delivered to the upper layer at the receiving side.
- On the receiving side, rdt_rcv() will be called when a packet arrives from the receiving side of the channel.
- When the rdt protocol wants to deliver data to the upper layer, it will do so by calling deliver_data().
- Both the send and receive sides of rdt send packets to the other side by a call to udt_send()

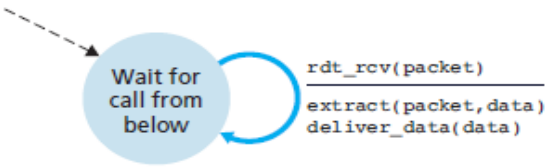
2.4.1 Building a Reliable Data Transfer Protocol

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

- 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.
- 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)).
- Here all packet flow is 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 since nothing can go wrong.
- Also we have assumed that the receiver is able to receive data as fast as the sender happens to send data. Thus, there is no need for the receiver to ask the sender to slow down.



a. rdt1.0: sending side



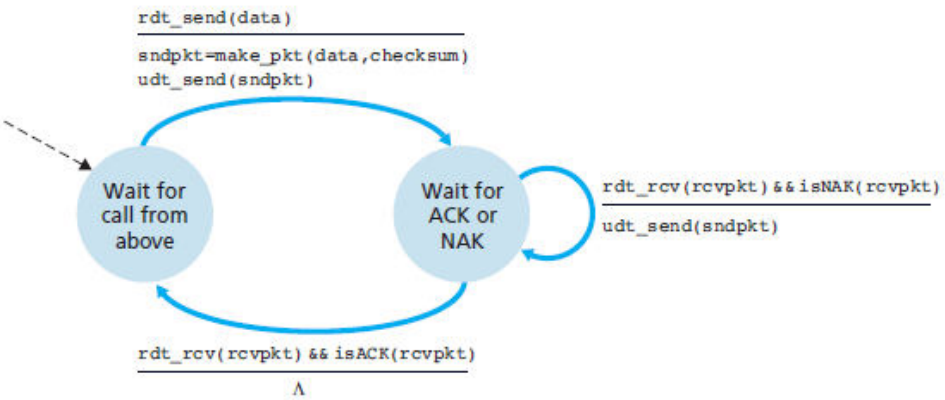
b. rdt1.0: receiving side

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

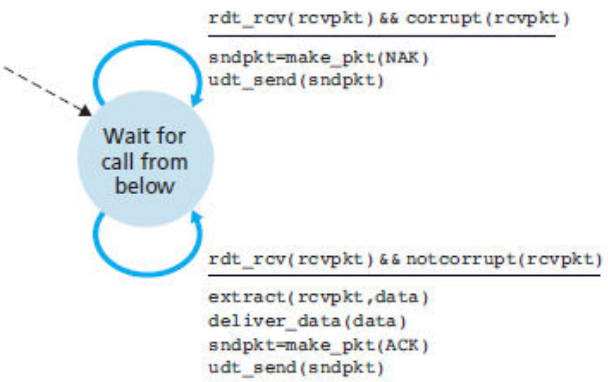
A more realistic model of the underlying channel is one in which bits in a packet may be corrupted.

Three additional protocol capabilities are required in (Automatic Repeat Request) ARQ protocols to handle the presence of bit errors:

- Error detection. First, a mechanism is needed to allow the receiver to detect when bit errors have occurred.
- Receiver feedback. Since the sender and receiver are typically executing on different end systems, possibly separated by thousands of miles, the only way for the sender to learn of the receiver's view of the world is for the receiver to provide explicit feedback to the sender. The positive (ACK) and negative (NAK) acknowledgment replies in the message-dictation scenario are examples of such feedback.
- Retransmission. A packet that is received in error at the receiver will be retransmitted by the sender.



a. rdt2.0: sending side



b. rdt2.0: receiving side

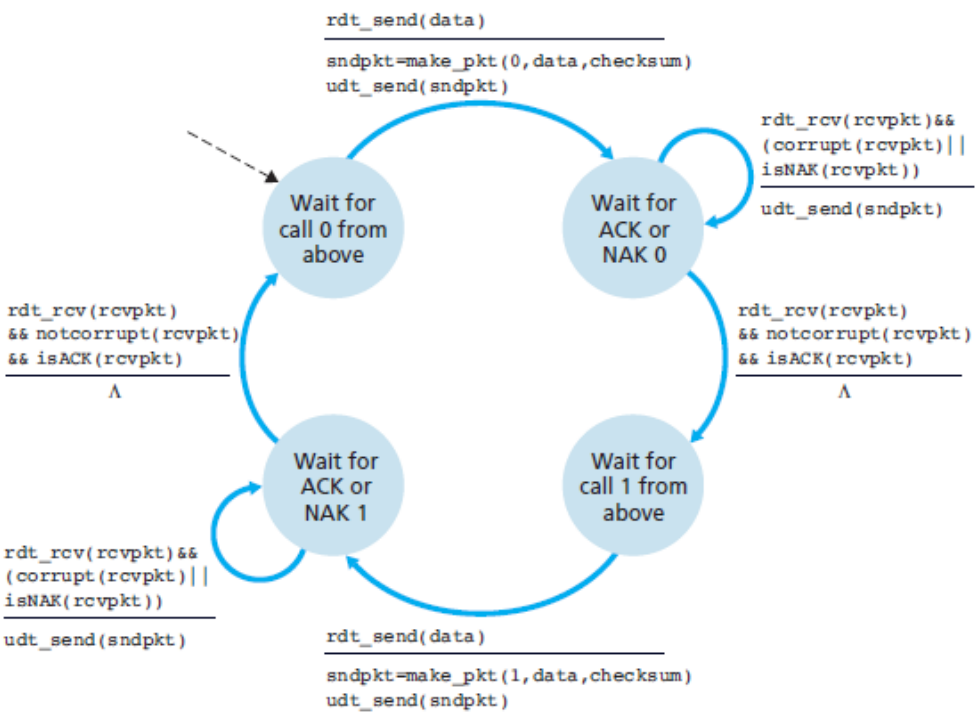
- The sender side has two states. In the leftmost state, the send-side protocol is waiting for data to be passed down from the upper layer.
- When the rdt_send(data) event occurs, the sender will create a packet (sndpkt) containing the data to be sent, along with a packet checksum and then send the packet via the udt_send(sndpkt) operation.
- 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)), the sender knows that the most recently transmitted packet has been received correctly and 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 can not 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 are known as stop-and-wait protocols.

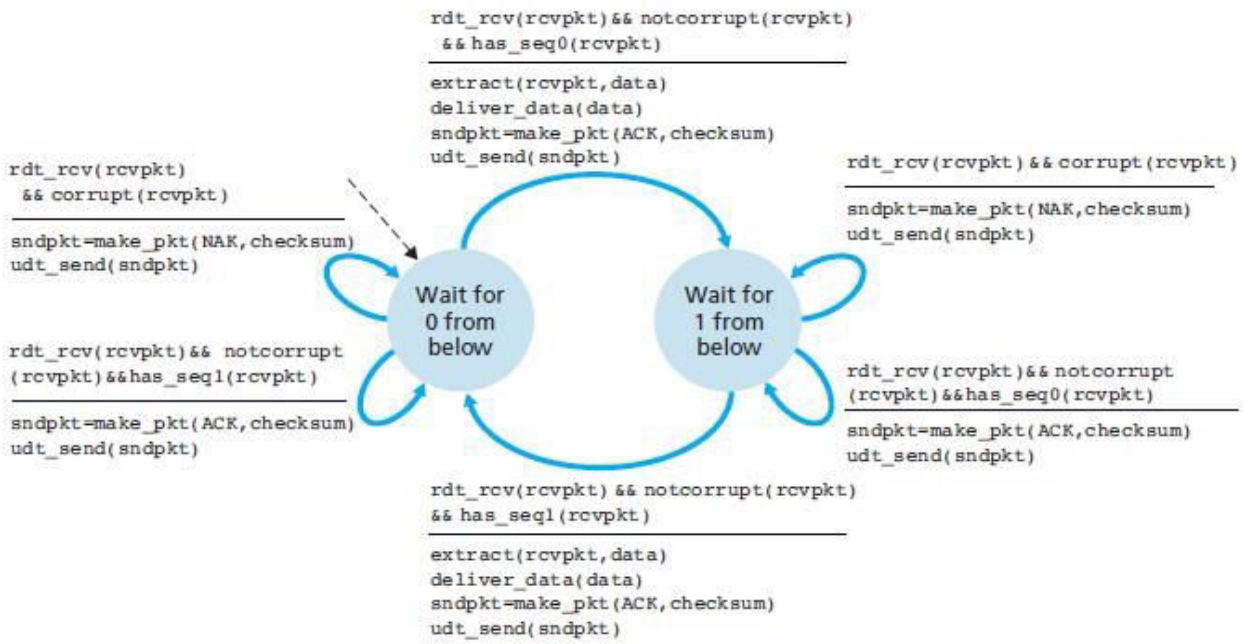
If ACK or NAK is corrupted (Duplicate Packet):

- In this case the sender resends the current data packet when it receives a garbled ACK or NAK packet. This approach, however, introduces duplicate packets into the sender-to-receiver channel.
- A simple solution to this 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 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 or a new packet.

Sender:



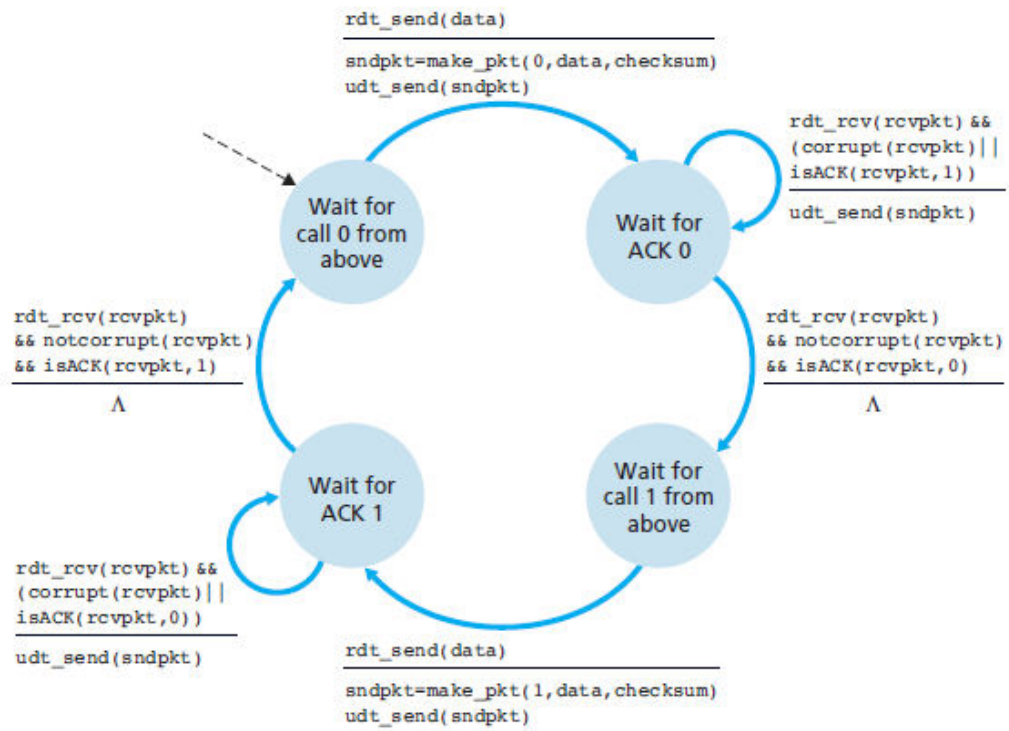
Receiver:



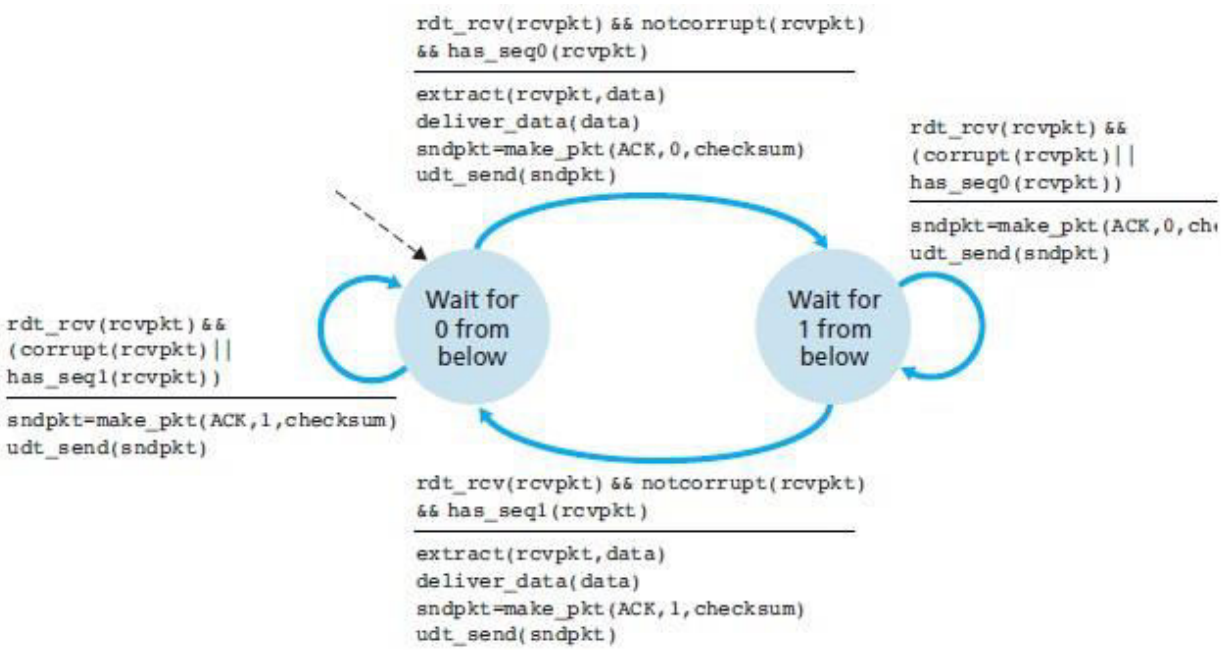
Duplicate ACK

- 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. We can accomplish the same effect as a NAK 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.

Sender:



Receiver:



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

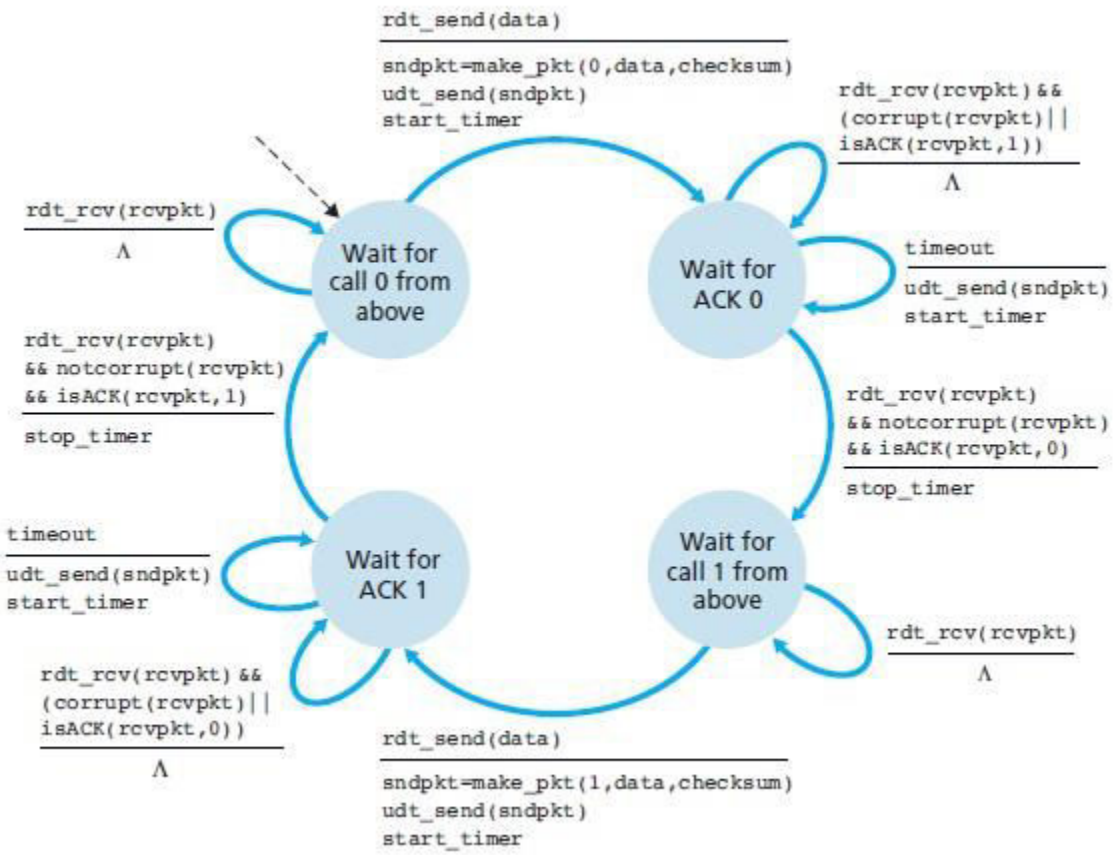
Suppose now that in addition to corrupting bits, the underlying channel can lose packets as well.

Two additional concerns must now be addressed by the protocol: how to detect packet loss and what to do when packet loss occurs.

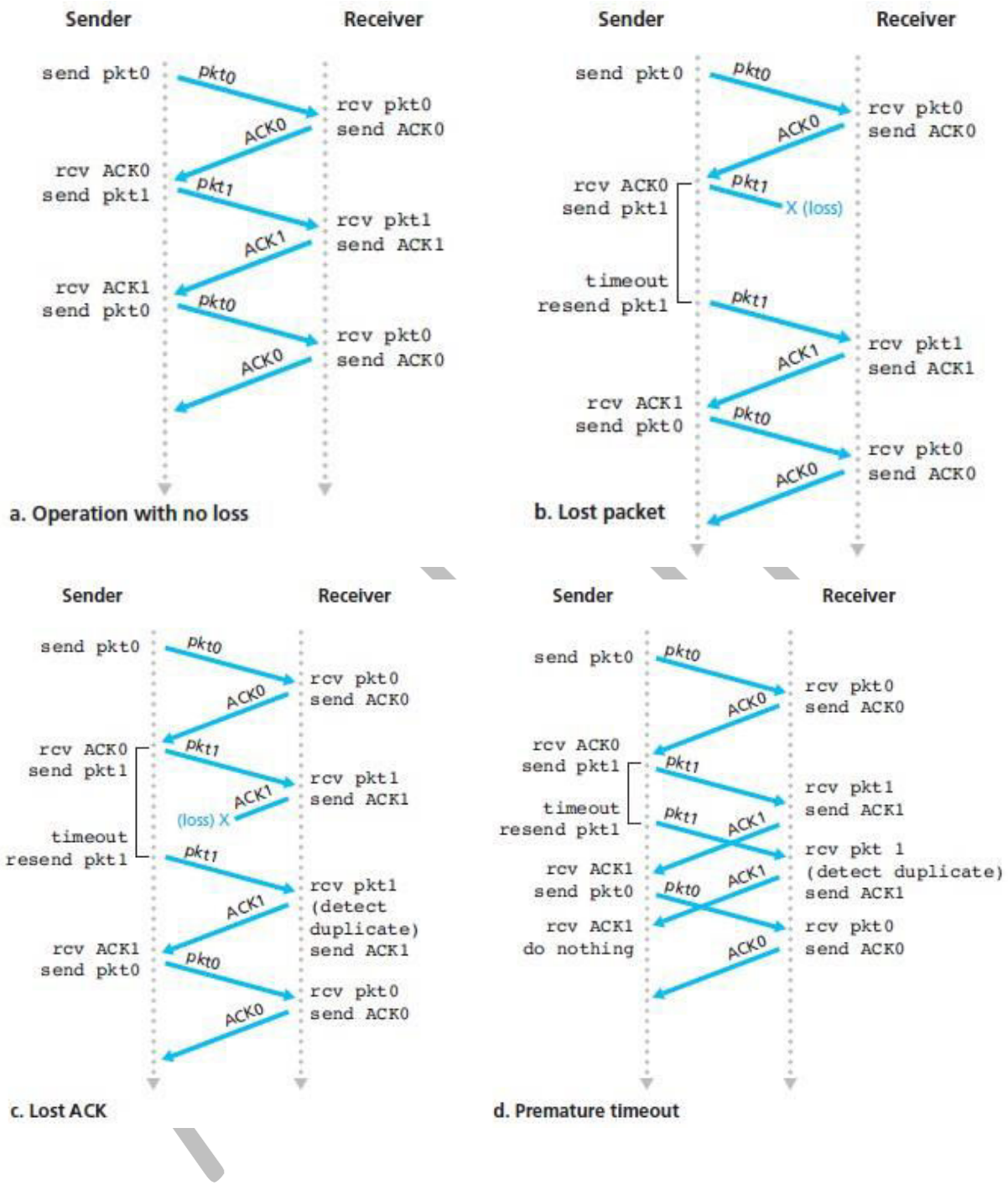
Suppose that the sender transmits a data packet and either that packet, or the receiver's ACK of that packet, gets lost. In either case, no reply is forthcoming at the sender from the receiver. If the sender is willing to wait long enough so that it is certain that a packet has been lost, it can simply retransmit the data packet.

But how long must the sender wait to be certain that something has been lost? The sender must clearly wait at least as long as a round-trip delay between the sender and receiver plus whatever amount of time is needed to process a packet at the receiver.

The approach thus adopted in practice is for the sender to judiciously choose a time value such that packet loss is likely, although not guaranteed, to have happened. If an ACK is not received within this time, the packet is retransmitted.



Following figures depicts various scenarios:



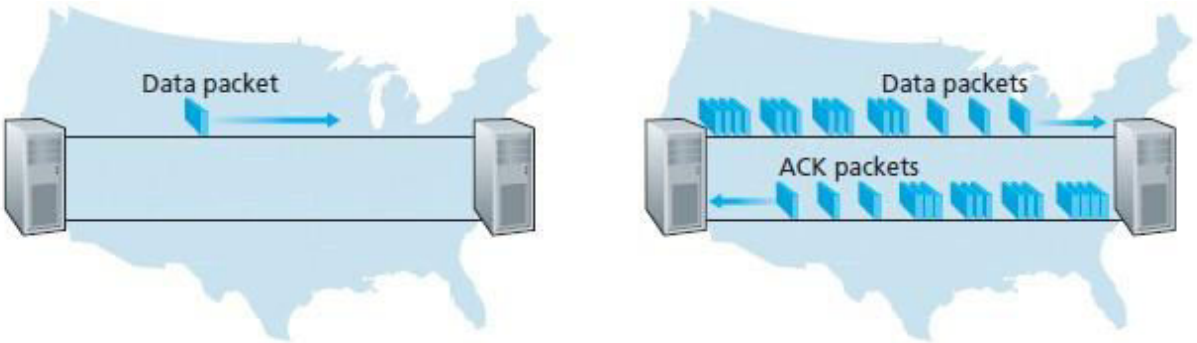
2.4.2 Pipelined Reliable Data Transfer Protocols

Let us consider two hosts located at two different locations.

The speed-of-light round-trip propagation delay between these two end systems, RTT, is approximately 30 milliseconds. Suppose that they are connected by a channel with a

transmission rate, R, of 1 Gbps (10^9 bits per second). With a packet size, L, of 1,000 bytes (8,000 bits) per packet, including both header fields and data, the time needed to actually transmit the packet into the 1 Gbps link is

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits/packet}}{10^9 \text{ bits/sec}} = 8 \text{ microseconds}$$



a. A stop-and-wait protocol in operation

b. A pipelined protocol in operation

Above Figure (a) shows that with our stop-and-wait protocol, if the sender begins sending the packet at t = 0, then at t = L/R = 8 microseconds, the last bit enters the channel at the sender side. The packet then makes its 15-msec cross-country journey, with the last bit of the packet emerging at the receiver at t = RTT/2 + L/R = **15.008 msec.**

Assuming for simplicity that ACK packets are extremely small (so that we can ignore their transmission time) and that the receiver can send an ACK as soon as the last bit of a data packet is received, the ACK emerges back at the sender at t = RTT + L/R = **30.008 msec.**

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 shows that the stop-and-wait protocol has a rather dismal sender utilization, Usender, of

$$U_{\text{sender}} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

That is, the sender was busy only 2.7 hundredths of one percent of the time!

The solution to this particular performance problem is simple: Rather than operate in a stop-and-wait manner, the sender is allowed to send multiple packets without waiting for

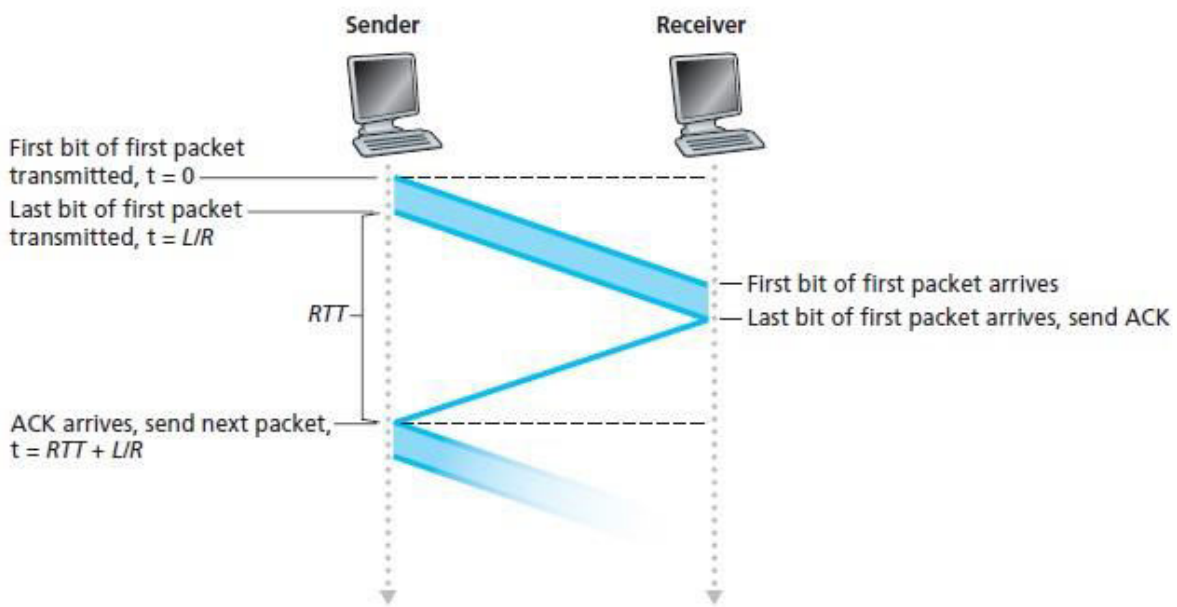
acknowledgments. if the sender is allowed to transmit three packets before having to wait for acknowledgments.

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:

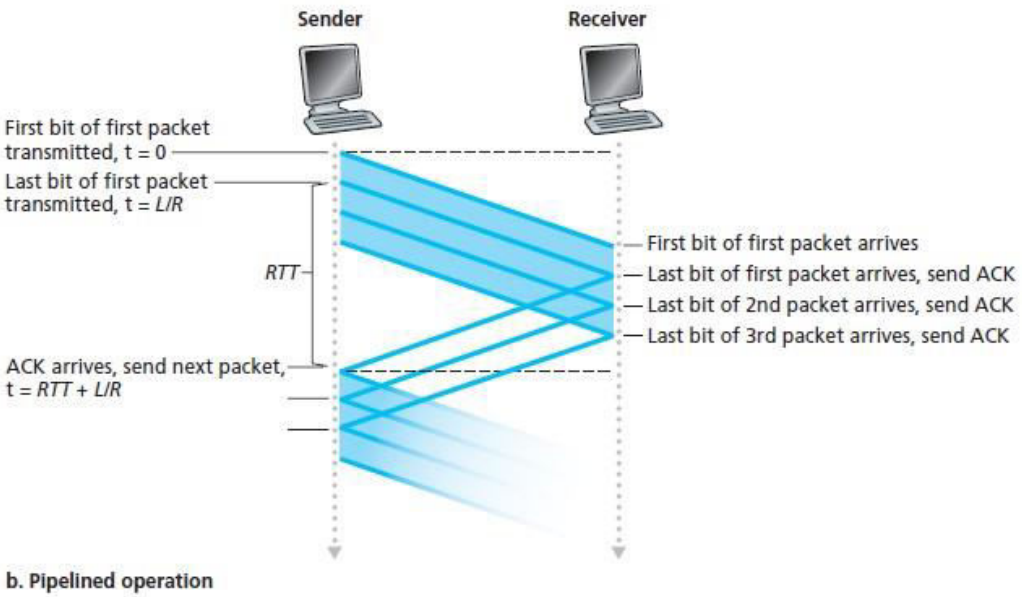
• The range of sequence numbers must be increased, since each in-transit packet (not counting retransmissions) must have a unique sequence number and there may be multiple, in-transit, unacknowledged packets.

• The sender and receiver sides of the protocols may have to buffer more than one packet. Minimally, 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, as discussed below.

• 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 identified: Go-Back-N and selective repeat.



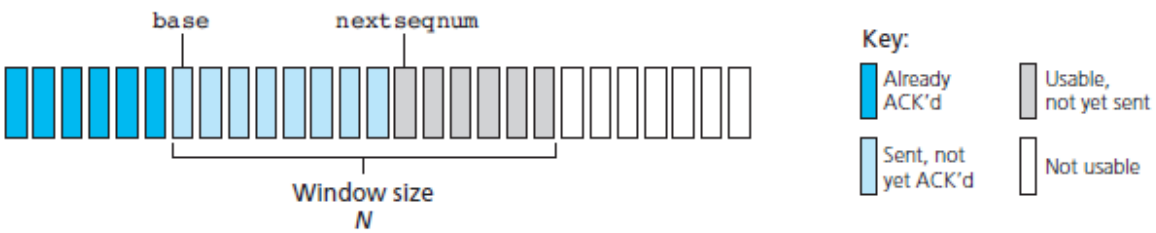
a. Stop-and-wait operation



b. Pipelined operation

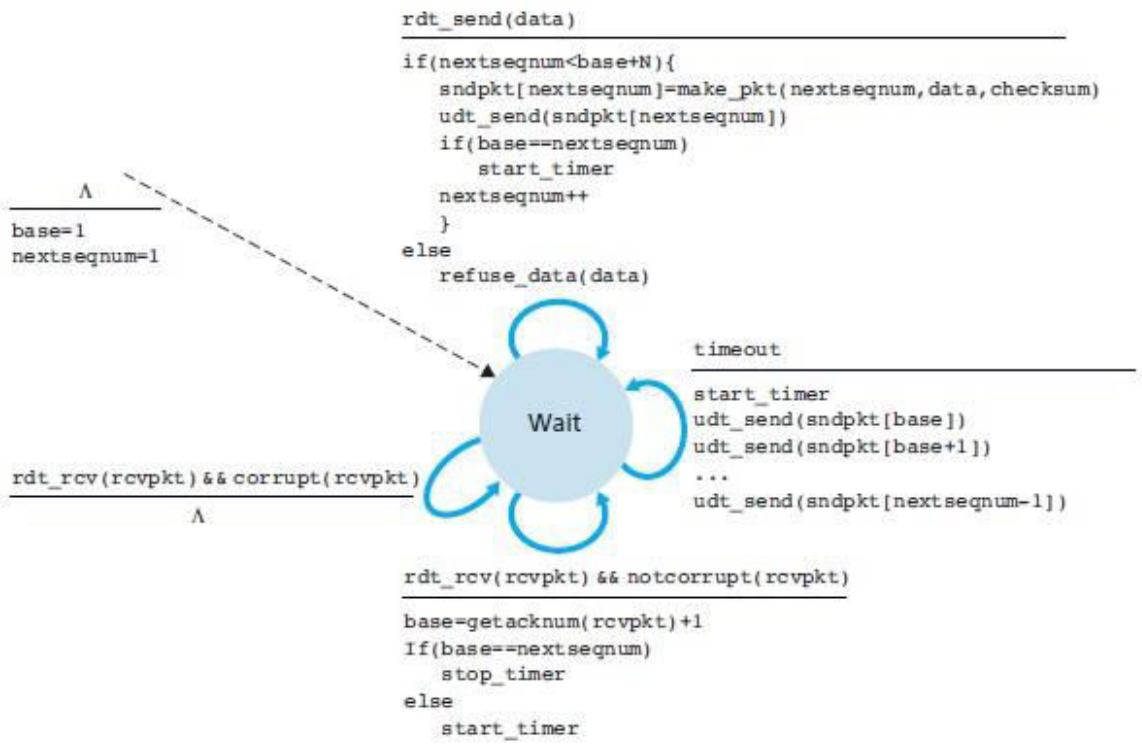
2.4.3 Go-Back-N (GBN)

- In a Go-Back-N (GBN) protocol, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, N, of unacknowledged packets in the pipeline.
- If base is the sequence number of the oldest unacknowledged packet and nextseqnum is 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.

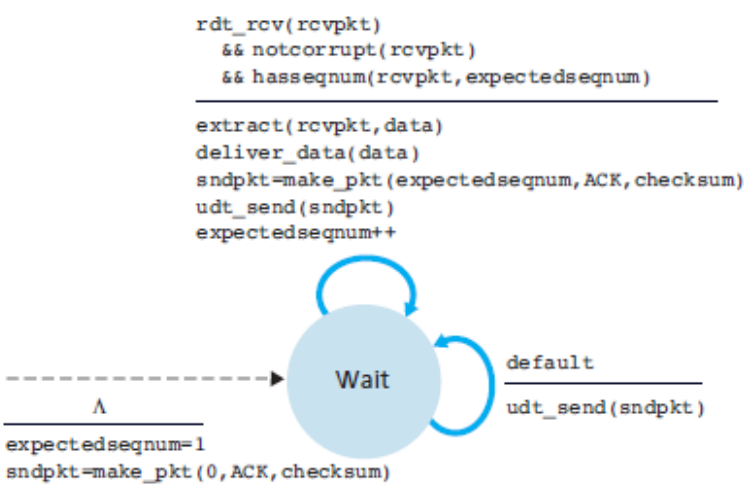


- Sequence numbers greater than or equal to base+N cannot be used until an unacknowledged packet currently in the pipeline has been acknowledged.

- 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. N is often referred to as the **window size** and the GBN protocol itself as a **sliding-window protocol.**
- If k is the number of bits in the packet sequence number field, the range of sequence numbers is thus [0,2k – 1]. With a finite range of sequence numbers, all arithmetic involving sequence numbers must then be done using modulo 2k arithmetic.



GBN Sender



GBN Receiver

The GBN sender must respond to three types of events:

- **Invocation from above.** 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. If the window is not full, a packet is created and sent, and variables are appropriately updated. 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 presumably then have to try again later. In a real implementation, the sender would more likely have either buffered (but not immediately sent) this data, or would have a synchronization mechanism that would allow the upper layer to call rdt_send() only when the window is not full.
- **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. We'll come back to this issue shortly when we examine the receiver side of GBN.
- **A timeout event.** The protocol's name, "Go-Back-N," is derived from the sender's behavior in the presence of lost or overly delayed packets. As in the stop-and-wait protocol, a timer will again 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. 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.

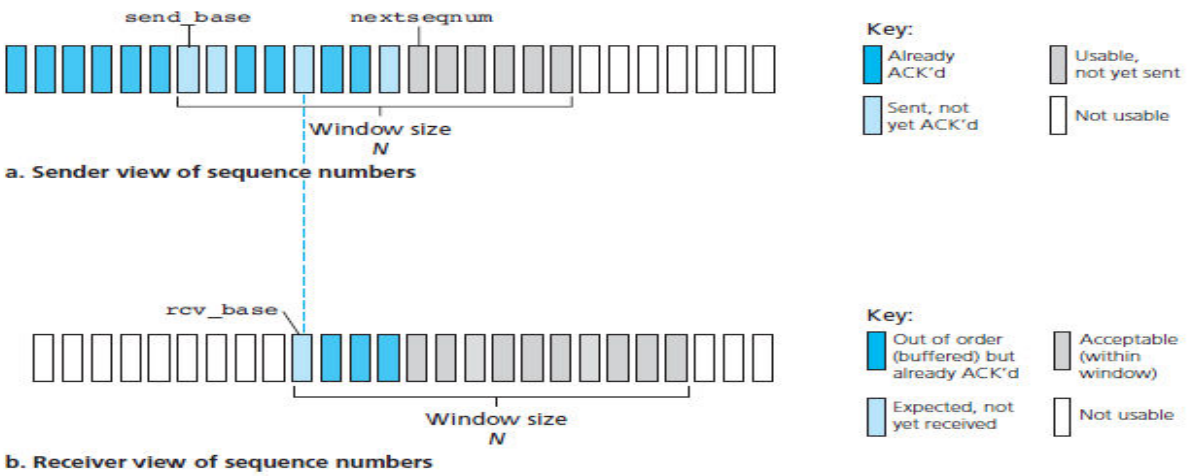
Receiver:

- 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. In all other cases, the receiver discards the packet and resends an ACK for the most recently received in-order packet.
- In our GBN protocol, the receiver discards out-of-order packets. GBN discard a correctly received but out-of-order packet.
- Suppose now that 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. However, 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.

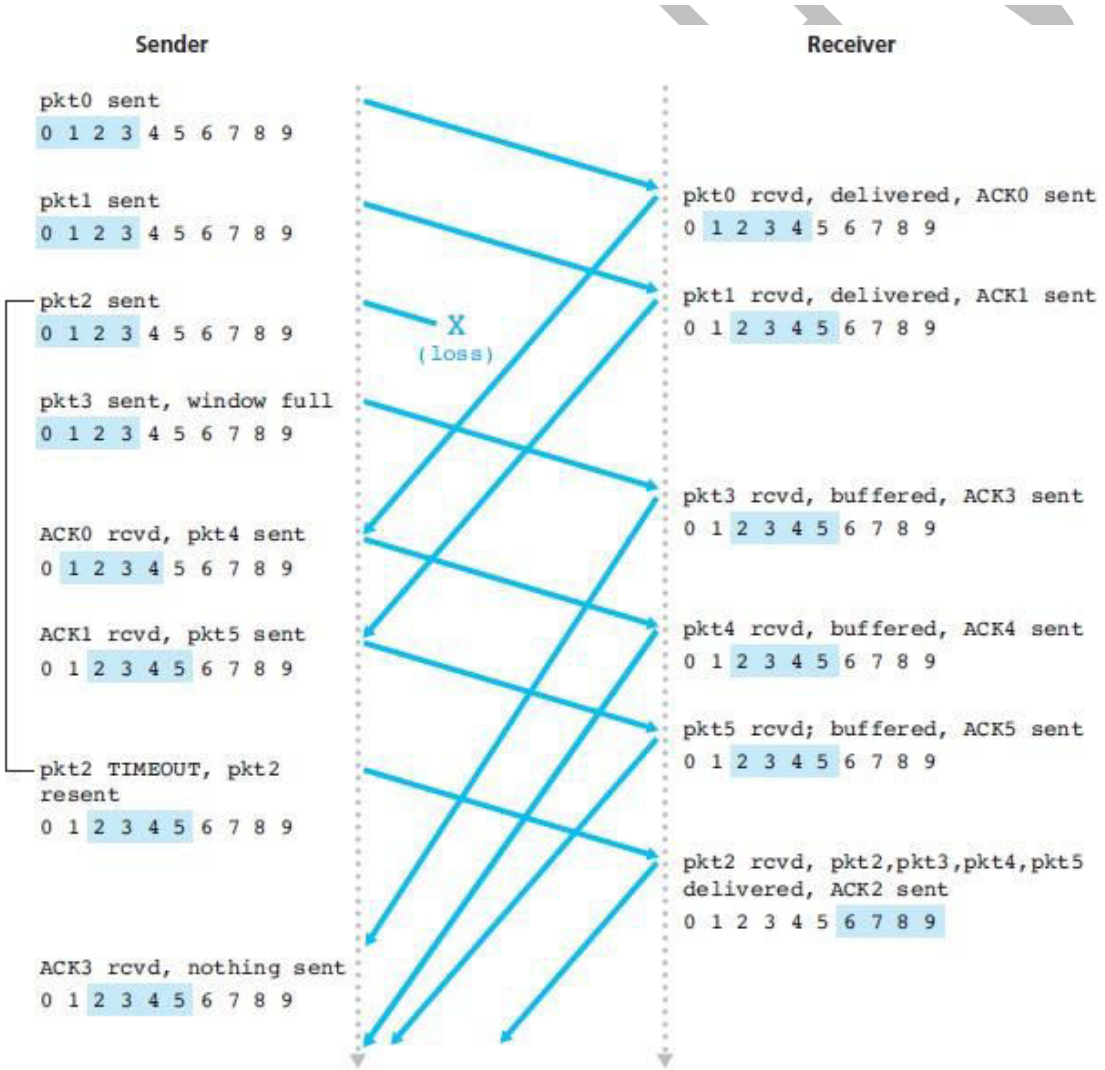
2.4.4 Selective Repeat (SR)

- Limitation of GBN: GBN itself suffers from performance problems. In particular, 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.
- As the name suggests, 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 individual, as 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 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.



1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. 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.* Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].
3. *ACK received.* If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. 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. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

SR sender events and actions



1. *Packet with sequence number in* `[rcv_base, rcv_base+N-1]` *is correctly received.* In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (`rcv_base` in Figure 3.22), 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. As an example, consider Figure 3.26. 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.* Ignore the packet.

SR receiver events and actions

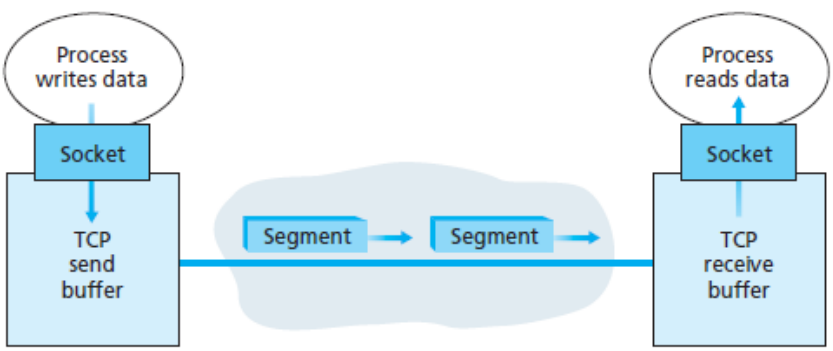
Summary of reliable data transfer mechanisms and their use

Mechanism	Use, Comments
Checksum	Used to detect bit errors in a transmitted packet.
Timer	Used to timeout/retransmit a packet, possibly because the packet (or its ACK) was lost within the channel. Because timeouts can occur when a packet is delayed but not lost (premature timeout), or when a packet has been received by the receiver but the receiver-to-sender ACK has been 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. We'll see shortly that the window size may be set on the basis of the receiver's ability to receive and buffer messages, or the level of congestion in the network, or both.

2.5 Connection-Oriented Transport: TCP

2.5.1 The TCP Connection

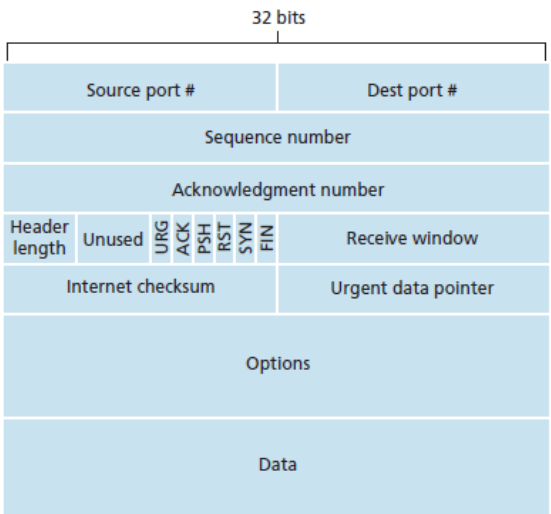
- TCP is said to be connection-oriented because connection has to be established between two application processes before they start transmitting data.
- As part of TCP connection establishment, both sides of the connection will initialize many TCP state variables associated with the TCP connection.
- 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. Multicasting is not allowed.
- Once a TCP connection is established, the two application processes can send data to each other.
- Let's 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 door, the data is in the hands of TCP running in the client.



- 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.
- 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 typically 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 plus the TCP/IP header length 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.
- The application reads the stream of data from this buffer.
- Each side of the connection has its own send buffer and its own receive buffer.

2.5.2 TCP Segment Structure



- The TCP segment consists of header fields and a data field.
- The data field contains a chunk of application data.
- The minimum length of TCP header is 20 bytes.
- The header includes **source and destination port numbers**, which are used for multiplexing/demultiplexing data from/to upper-layer applications.
- The 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. It is used to indicate the number of bytes that a receiver is willing to accept.
  - 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.
  - 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.
  - Finally, the **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

The sequence number for a segment is the byte-stream number of the first byte in the segment.

Example: Suppose that 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. 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.

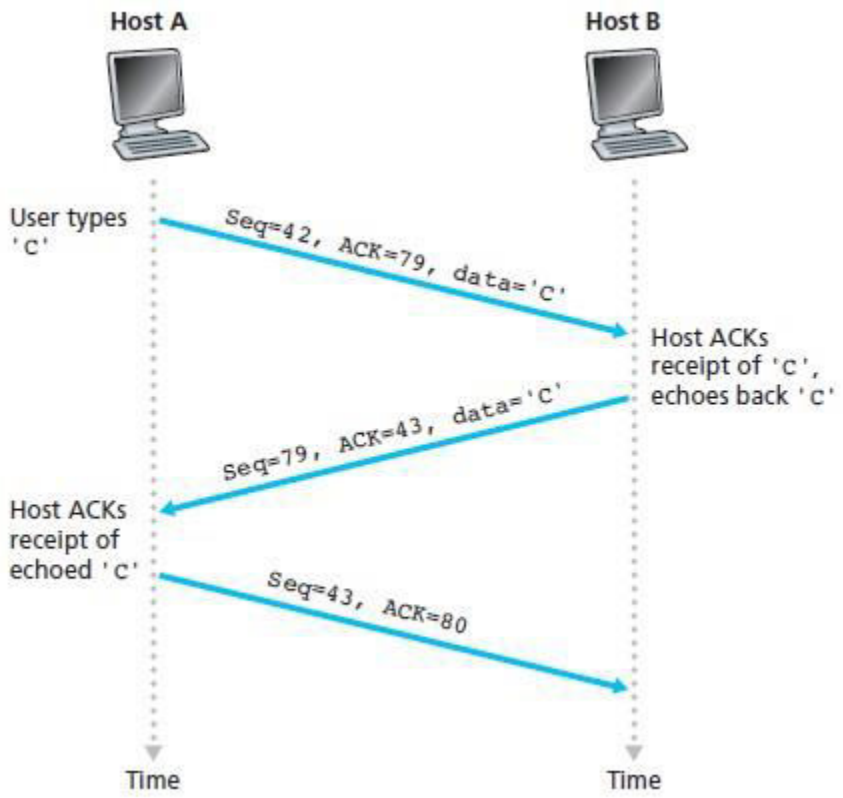
The **acknowledgment number** that Host A puts in its segment is the sequence number of the next byte Host A is expecting from Host B.

Example: Suppose that Host A has received all bytes numbered 0 through 535 from B and suppose that it is about to send a segment to Host B. Host A is waiting for byte 536 and all the subsequent bytes in Host B's data stream. So Host A puts 536 in the acknowledgment number field of the segment it sends to B.

Telnet: A Case Study for Sequence and Acknowledgment Numbers

Telnet is a popular application-layer protocol used for remote login. It runs over TCP and is designed to work between any pair of hosts.

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. This "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.



2.5.3 Estimating the Round-Trip Time

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.

The SampleRTT values will fluctuate from segment to segment due to congestion in the routers and to the varying load on the end systems.

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.

$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$$

Here $\alpha = 0.125$

In statistics, This kind of average is called an exponential weighted moving average (EWMA).

In addition to having an estimate of the RTT, it is also valuable to have a measure of the variability of the RTT : DevRTT.

DevRTT, is an estimate of how much SampleRTT typically deviates from EstimatedRTT:

$$DevRTT = (1 - \beta) \cdot DevRTT + \beta \cdot |SampleRTT - EstimatedRTT|$$

Here $\beta = 0.25$

Now Timeout can be calculated as:

$$TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT$$

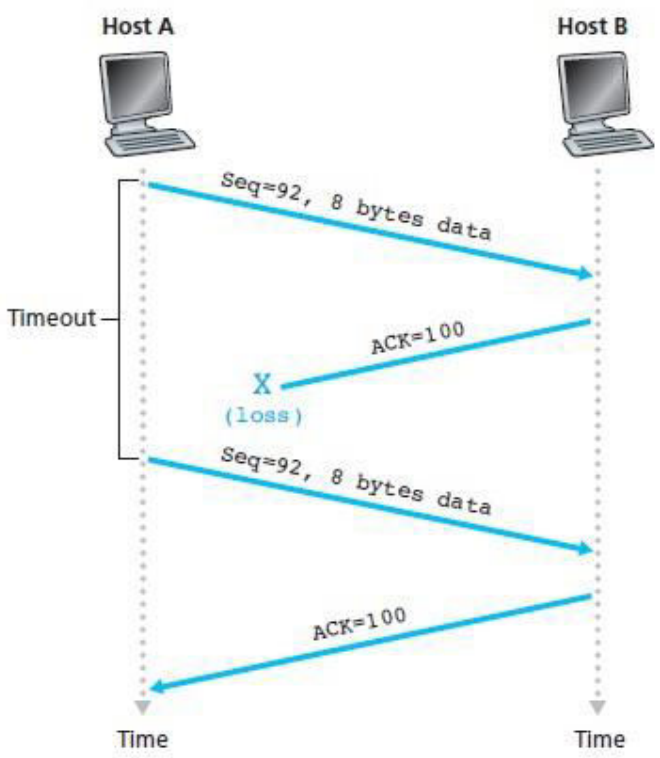
2.5.4 Reliable Data Transfer

- 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.

```
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 */
```

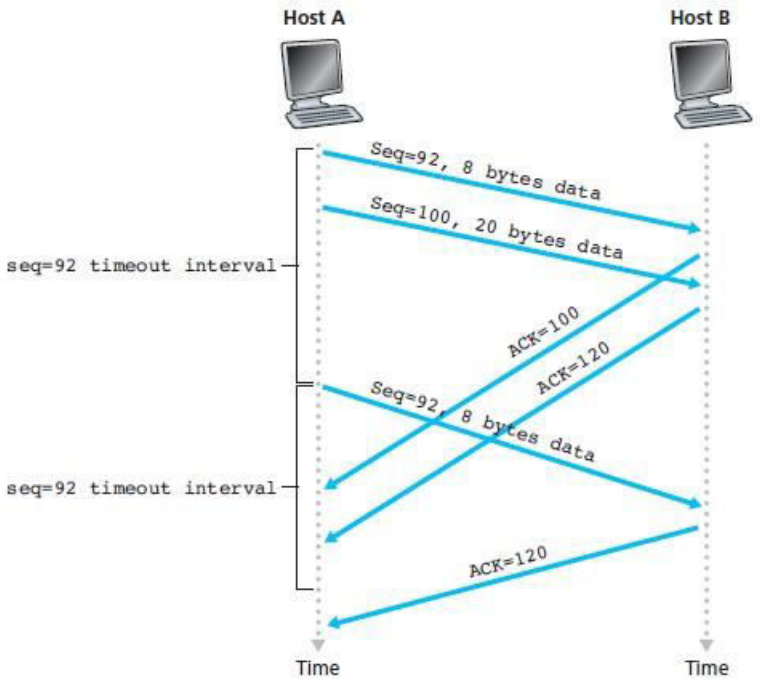
A Few Interesting Scenarios:

1. 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. Of course, 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.

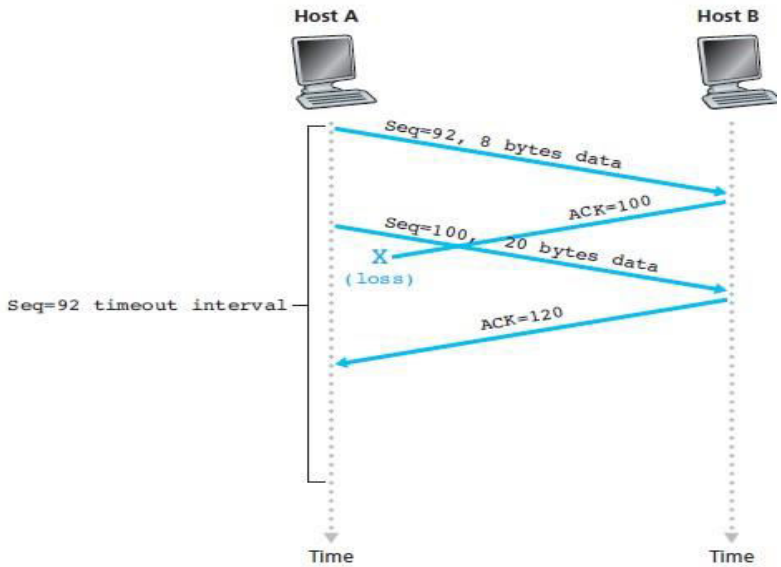


2. Host A sends two segments back to back. The first segment has sequence number 92 and 8 bytes of data, and the second segment has sequence number 100 and 20 bytes of data. Suppose that both segments arrive intact at B, and B sends two separate acknowledgments for each of these segments. The first of 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.



3. 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.



Doubling the Timeout Interval

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 Timeout Interval associated with the oldest not yet acknowledged segment is 0.75 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.

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. Fortunately, 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.

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.

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.

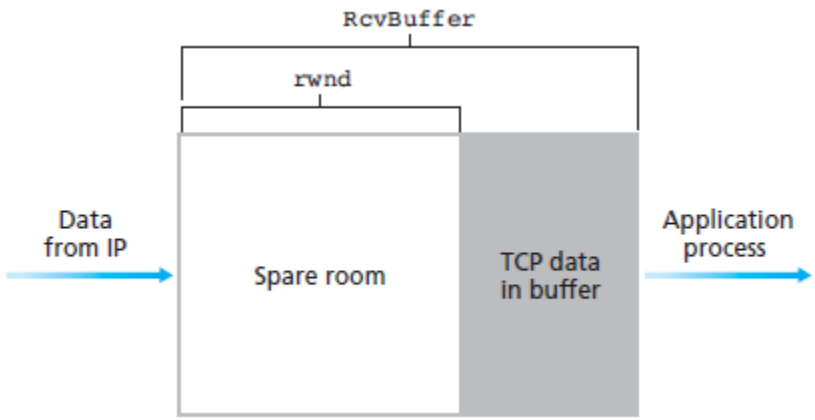
Because a sender often sends a large number of segments back to back, if one segment is lost, there will likely be many back-to-back duplicate ACKs. If the TCP sender receives three duplicate ACKs for the same data, it takes this as an indication that the segment following the segment that has been ACKed three times has been lost. 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.

```
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;
```

2.5.5 Flow Control

- TCP provides a flow-control service to its applications to eliminate the possibility of the sender overflowing the receiver's buffer.
- Flow control is a speed-matching service—matching the rate at which the sender is sending against the rate at which the receiving application is reading.
- 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.
- 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]

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 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

2.5.6 TCP Connection Management

TCP has 3 phases

1) Connection Establishment phase

2) Data transmission phase

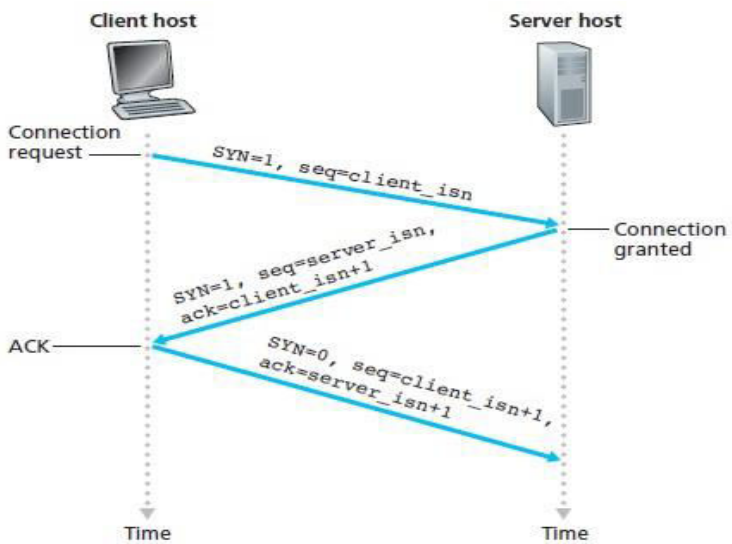
3) Connection Termination phase

Connection establishment phase:

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, 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.

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. However, it does contain three important pieces of information in the segment header. First, the SYN bit is set to 1. Second, the acknowledgment field of the TCP segment header is set to client_isn+1. Finally, the server chooses its own initial sequence number (server_isn) and puts this value in the sequence number field of the TCP segment header. This is referred as 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 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.

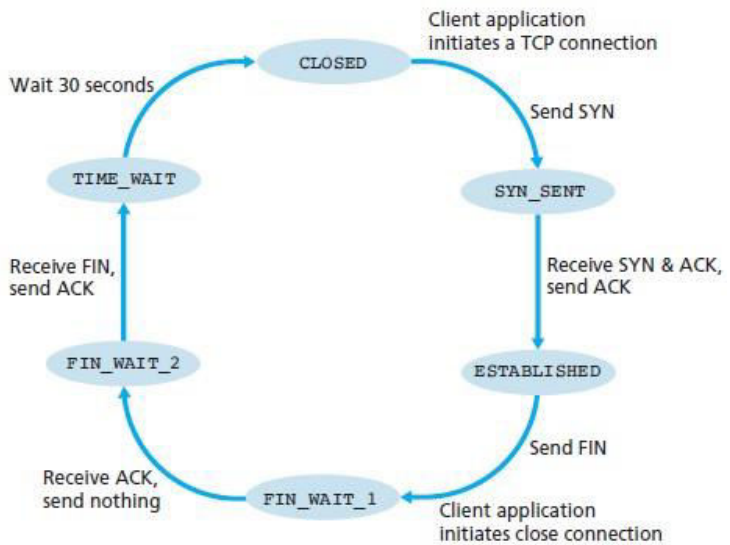


Connection Termination phase:

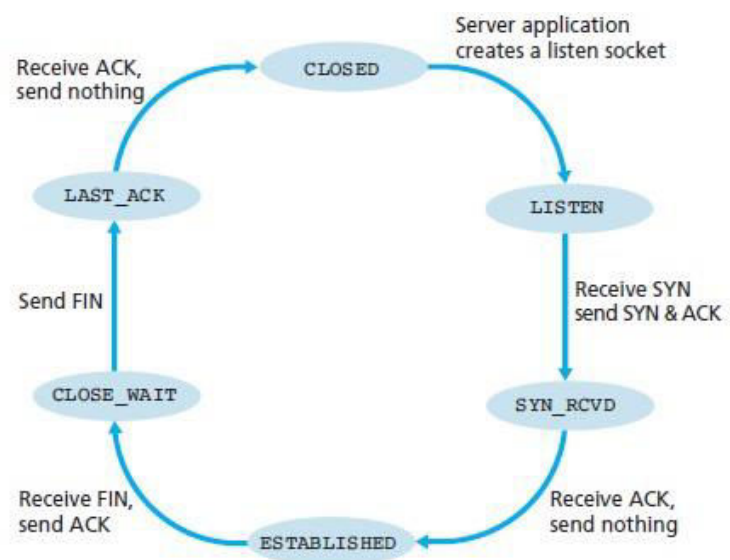
Either of the two processes participating in a TCP connection can end the connection. When a connection ends, the "resources" (that is, the buffers and variables) in the hosts are deallocated. For connection termination TCP sends segment with FIN flag set to 1. When the server receives this segment, 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.

State transition diagram:

Client



Server:

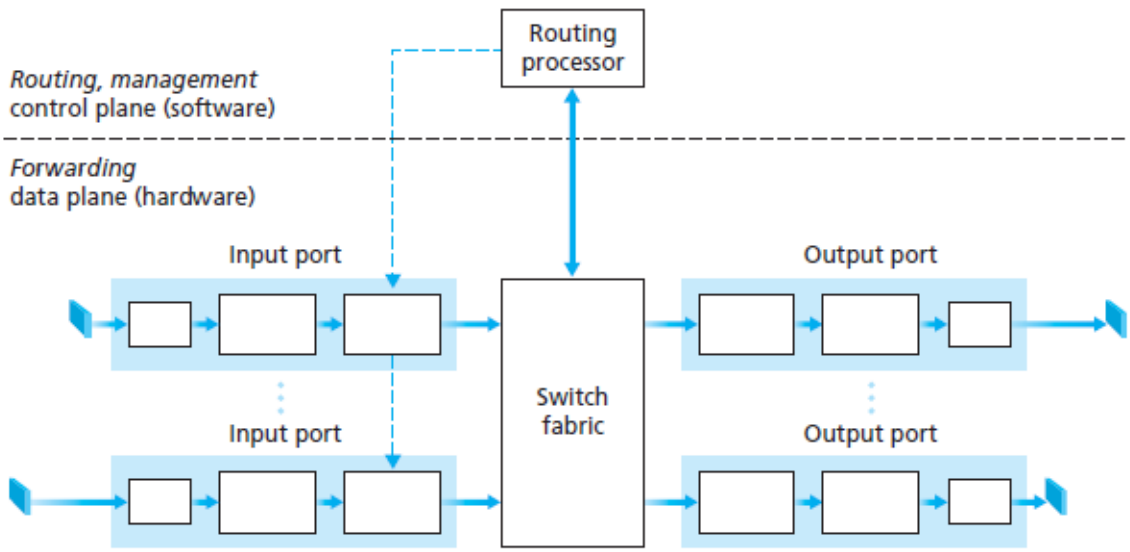


Module – 3

NETWORK LAYER

Structure of a Router

high-level view of a generic router architecture is shown below. Four router components can be identified:

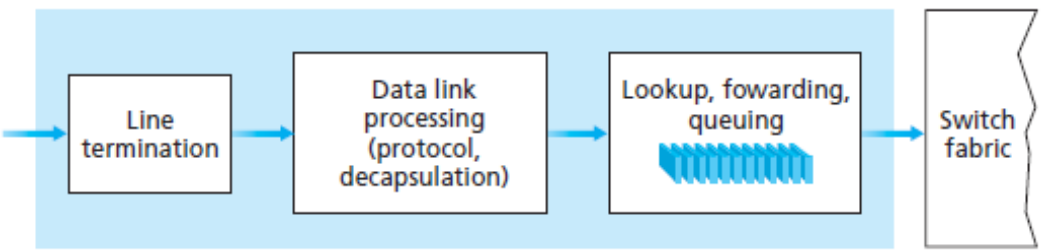


- **Input ports:** An input port performs several key functions.
  It performs the physical layer function of terminating an incoming physical link at a router.
  An input port also performs link-layer functions needed to interoperate with the link layer at the other side of the incoming link.
  The lookup function is also performed at the input port.
  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.
- **Output port:**. An output port stores packets received from the switching fabricand 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.

- A router's input ports, output ports, and switching fabric together implement the forwarding function and are almost always implemented in hardware. These forwarding functions are sometimes collectively referred to as the **router forwarding plane**.
- **Router control plane** functions are usually implemented in software and execute on the routing processor

Input Processing



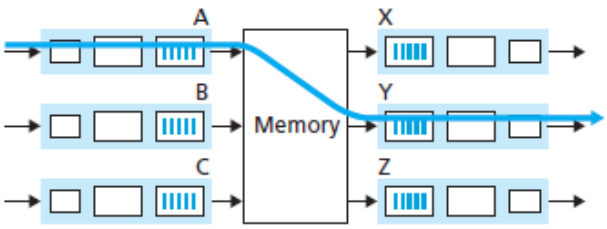
- The input port's line termination function and link-layer processing implement the physical and link layers for that individual input link.
- The lookup performed in the input port is central to the router's operation—it is here that the router uses the forwarding table to look up the output port to which an arriving packet will be forwarded via the switching fabric.
- The forwarding table is computed and updated by the routing processor, with a shadow copy typically stored at each input port. The forwarding table is copied from the routing processor to the line cards over a separate bus.
- 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.

Switching

The switching fabric switches the packet from an input port to an output port. Switching can be accomplished in a number of ways:

1. Switching via memory:

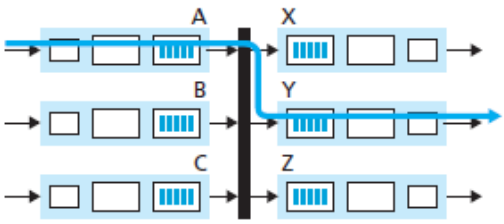
- The simplest, earliest routers were traditional computers, with switching between input and output ports being done under direct control of the CPU (routing processor).
- Input and output ports functioned as traditional I/O devices in a traditional operating system.
- An input port with an arriving packet first signaled the routing processor via an interrupt.
- The packet was then copied from the input port into processor memory.
- The routing processor then extracted the destination address from the header, looked up the appropriate output port in the forwarding table, and copied the packet to the output port's buffers.
- Here two packets cannot be forwarded at the same time, even if they have different destination ports, since only one memory read/write over the shared system bus can be done at a time.
- Many modern routers switch via memory. A major difference from early routers is that the lookup of the destination address and the storing of the packet into the appropriate memory location are performed by processing on the input line cards.



2. Switching via a bus:

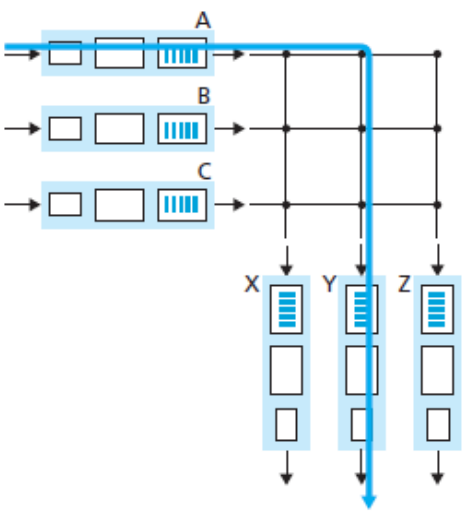
- In this approach, an input port transfers a packet directly to the output port over a shared bus, without intervention by the routing processor.
- This is typically done by having the input port pre-pend a switch-internal label (header) to the packet indicating the local output port to which this packet is being transferred and transmitting the packet onto the bus.
- The packet is received by all output ports, but only the port that matches the label will keep the packet.
- The label is then removed at the output port, as this label is only used within the switch to cross the bus.

- If multiple packets arrive to the router at the same time, each at a different input port, all but one must wait since only one packet can cross the bus at a time.



3. 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 multiprocessor computer architecture.
- A crossbar switch is an interconnection network consisting of 2N buses that connect N input ports to N output ports.
- Each vertical bus intersects each horizontal bus at a crosspoint, which can be opened or closed at any time by the switch fabric.

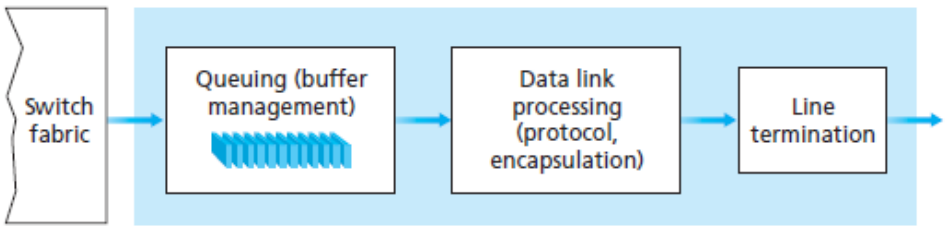


When a packet arrives from port A and needs to be forwarded to port Y, the switch controller closes the crosspoint at the intersection of busses A and Y, and port A then sends the packet onto its bus, which is picked up (only) by bus Y. Note that a packet from port B can be forwarded to port X at the same time, since the A-to-Y and B-to-X packets use different input and output busses. Thus, unlike the previous two switching approaches, crossbar networks are capable of forwarding multiple packets in parallel. However, if two packets

from two different input ports are destined to the same output port, then one will have to wait at the input, since only one packet can be sent over any given bus at a time.

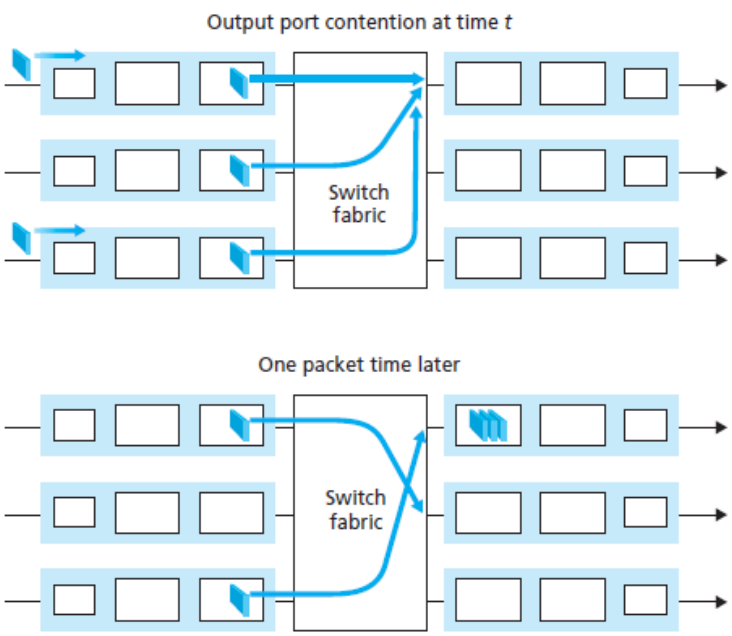
Output Processing

Output port processing takes packets that have been stored in the output port's memory and transmits them over the output link. This includes selecting and de-queuing packets for transmission, and performing the needed link layer and physical-layer transmission functions.



Queuing

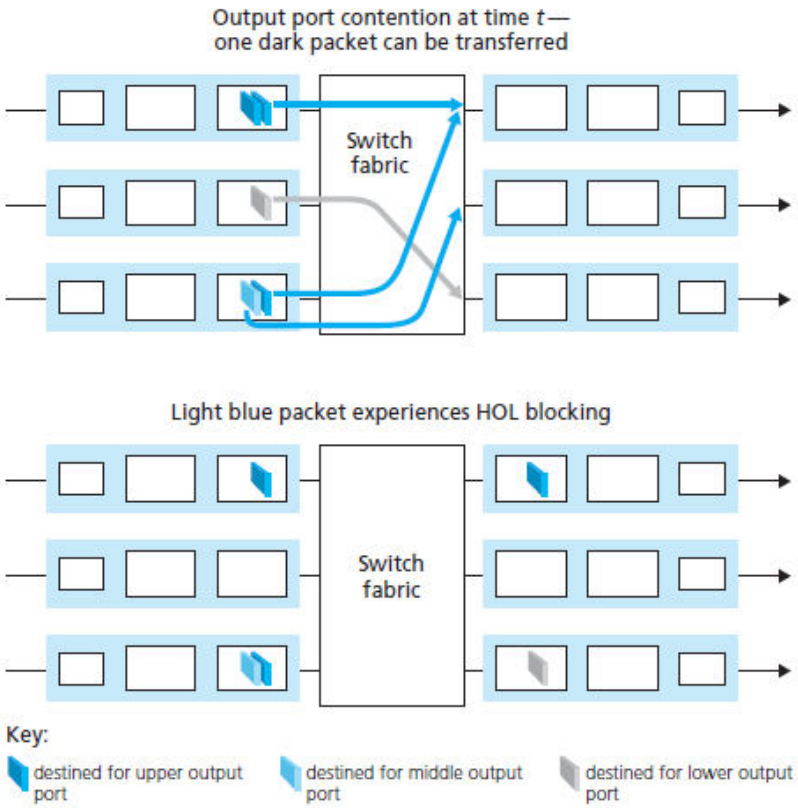
- Packet queues may form at both the input ports and the output ports.
- The location and extent of queuing will depend on the traffic load, the relative speed of the switching fabric, and the line speed.
- When many packets arrive from same source at a faster rate than switching rate queuing at input port occurs.
- When many packets are destined towards same output port queuing at output port occurs.



- A consequence of output port queuing is that a **packet scheduler** at the output port must choose one packet among those queued for transmission.
- Many packet scheduling algorithms like first-come-first-served (FCFS) scheduling, priority queuing, fair queuing or a more sophisticated scheduling discipline such as weighted fair queuing (WFQ) is available.
- Packet scheduling plays a crucial role in providing quality-of-service guarantees.
- Similarly, if there is not enough memory to buffer an incoming packet, a decision must be made to either drop the arriving packet (a policy known as **drop-tail**) or remove one or more already-queued packets to make room for the newly arrived packet.
- In some cases, it may be advantageous to drop (or mark the header of) a packet before the buffer is full in order to provide a congestion signal to the sender.
- A number of packet-dropping and -marking policies (which collectively have become known as **active queue management (AQM)** algorithms) have been proposed and analyzed.
- One of the most widely studied and implemented AQM algorithms is the **Random Early Detection (RED)** algorithm. Under RED, a weighted average is maintained for the length of the output queue.
- If the average queue length is less than a minimum threshold, min_{th}, 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, max_{th}, when a packet arrives, the packet is marked or dropped.
- Finally, if the packet arrives to find an average queue length in the interval [min_{th}, max_{th}], the packet is marked or dropped with a probability that is typically some function of the average queue length, min_{th}, and max_{th}.

Consider the following scenario:

Suppose that in the below figure 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.

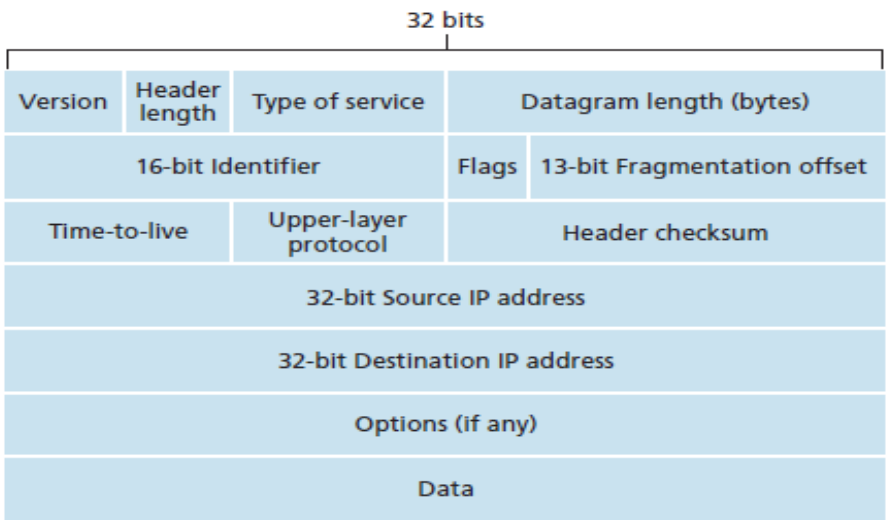


The Internet Protocol (IP)

Internet addressing and forwarding are important components of the Internet Protocol (IP).

There are two versions of IP in use today: IPv4, IPv6.

Datagram Format



- **Version number:** These 4 bits specify the IP protocol version of the datagram.
- **Header length:** Because an IPv4 datagram can contain a variable number of options these 4 bits specify the total header bytes.
- **Type of service:** The type of service (TOS) bits were included in the IPv4 header to allow different types of IP datagrams (for example, datagrams particularly requiring low delay, high throughput, or reliability) to be distinguished from each other.
- **Datagram length:** This is the total length of the IP datagram (header plus data), measured in bytes.
- **Identification:** this field represents identification number assigned to related fragments.
- **Flag:** there are three flags, first bit is unused, second bit is do not fragment bit, If this bit is set intermediate nodes should not perform fragmentation. Last bit is more fragment bit. More fragment bit represents more fragments to follow after this.
- **Fragmentation offset:** Starting byte of a fragment (In multiples of 8 bytes).
- **Time-to-live:** The time-to-live (TTL) field is included to ensure that datagrams do not circulate forever in the network. It represents hop limit.
- **Protocol:** Represents the upper layer protocol, 6 for TCP, 17 for UDP.
- **Header checksum:** field is used for error detection.
- **Source and destination IP addresses:** represents 32 bit source and destination IP address.
- **Options:** The options fields allow an IP header to be extended. It allows to include additional functionalities.
- **Data (payload):** represents the data that has to be transmitted.

IP Datagram Fragmentation

- 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.
- IP datagram is divided into smaller packets according to MTU. These smaller packets are called fragments and process is called fragmentation.
- Fragments need to be reassembled before they reach the transport layer at the destination.

- Receiver needs to reassemble all the fragments belong to same original IP datagram. In order to identify all the related fragments **Identification** field is used.
- There are three flags, first bit is unused, second bit is do not fragment bit, If this bit is set intermediate nodes should not perform fragmentation. Last bit is more fragment bit. More fragment bit represents more fragments to follow after this.
- In order for the destination host to determine whether a fragment is missing the offset field is used to specify where the fragment fits within the original IP datagram.

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. Suppose that the original datagram is stamped with an identification number of 777. Following table shows the fragmentation.

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)

At the destination, the payload of the datagram is passed to the transport layer only after the IP layer has fully reconstructed the original IP datagram. If one or more of the fragments does not arrive at the destination, the incomplete datagram is discarded and not passed to the transport layer.

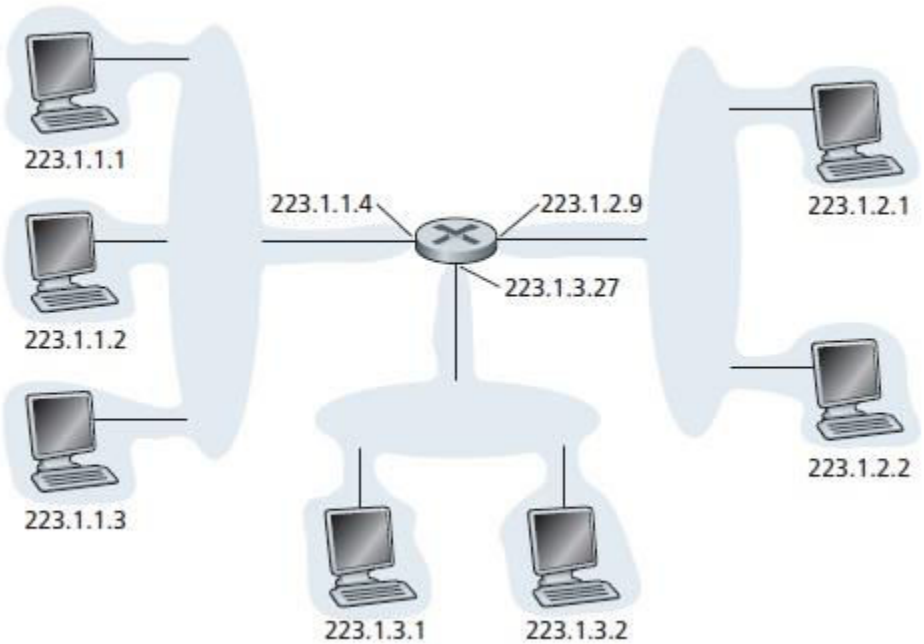
IPv4 Addressing

- Each IP address is 32 bits long (equivalently, 4 bytes), and there are thus a total of 232 possible IP addresses. Approximately there are about 4 billion possible IP addresses.

- IP 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.
- Ex: 193.32.216.9
- The address 193.32.216.9 in binary notation is 11000001 00100000 11011000 00001001
- Each interface on every host and router in the global Internet must have an IP address that is globally unique.
- IP address has 2 parts: Network ID and Host ID. Network ID is used to identify the network and Host ID is used to identify host in the network.
- A network can be divided into smaller sub networks called subnets.
- Initially IP address was divided into 5 classes. We call this as classful addressing. It leads to shortage of IP Address. Now Classless Inter Domain Routing (CIDR) addressing is used. In CIDR IP address is represented as a.b.c.d/x where x represents number of bits used for Network ID.
- Example.

  If 256 hosts are there last 8 bit is allocated to host ID remaining 24 bit is allocated to network ID. Here /x value is /24.

Below figure shows sub netting.



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.
- Host addresses can also be configured manually, but more often this task is now done using the Dynamic Host Configuration Protocol (DHCP).
- DHCP allows a host to obtain (be allocated) an IP address automatically.
- DHCP works over UDP with port number 67.

DHCP involves four steps:

1) DHCP server discovery

Host broadcast DHCP discovery message with source address 0.0.0.0 and destination address 255.255.255.255

2) DHCP server offer(s)

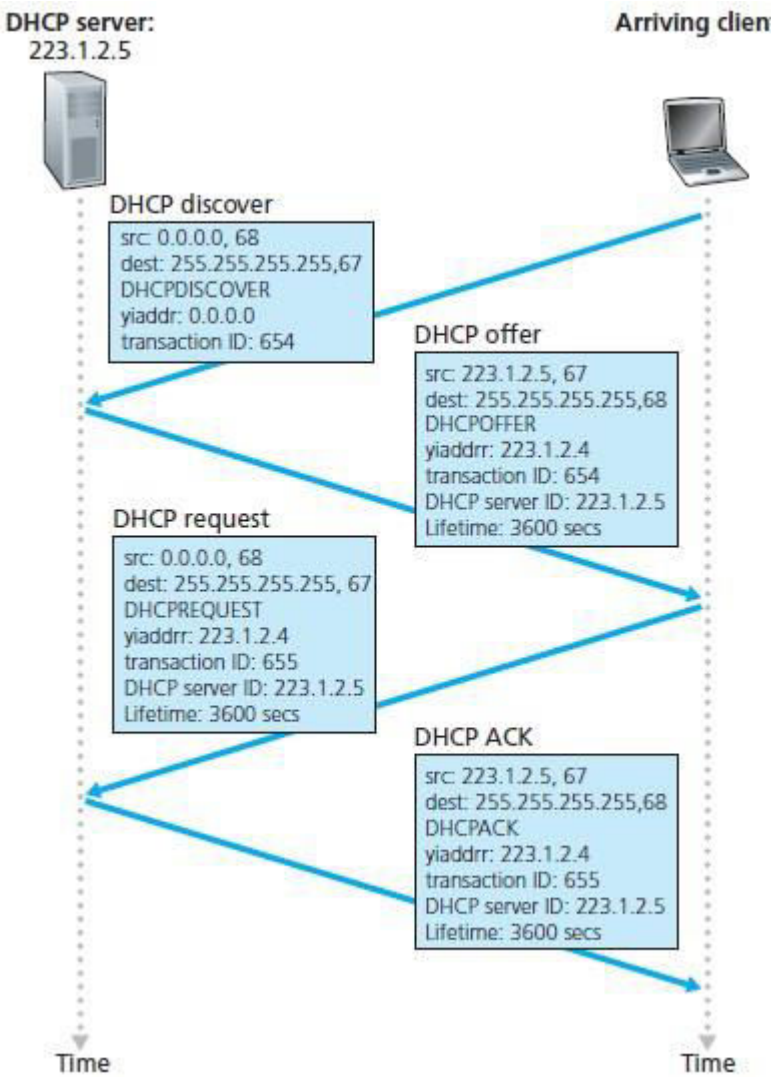
A DHCP server receiving a DHCP discover message responds to the client with a DHCP offer message that is broadcast to all nodes on the subnet, again using the IP broadcast address of 255.255.255.255.

3) 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.

4) DHCP ACK

The server responds to the DHCP request message with a DHCP ACK message, confirming the requested parameters.



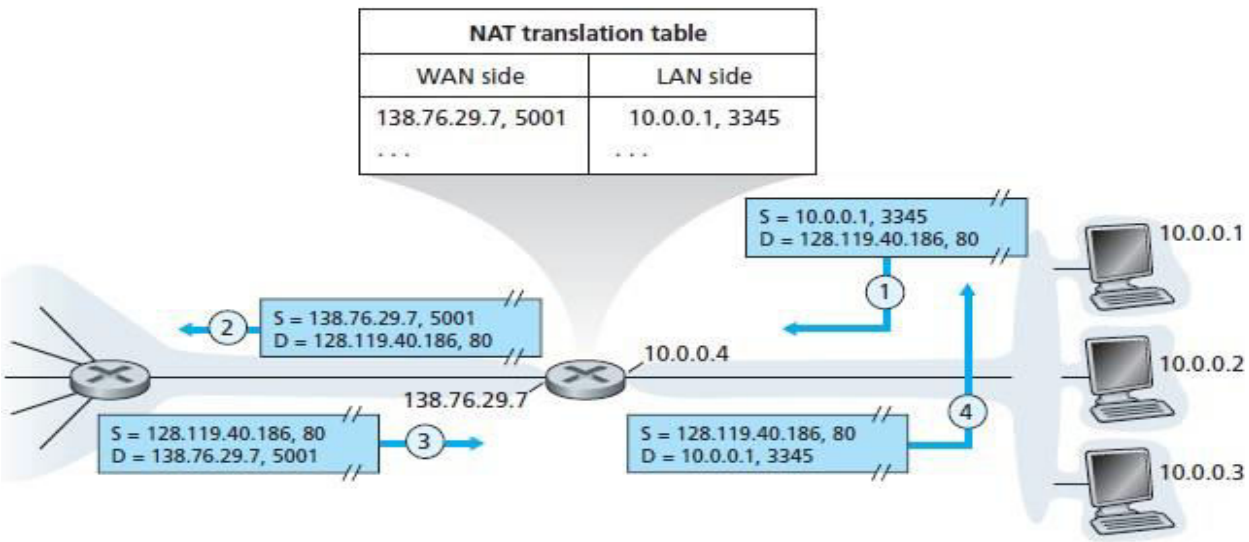
Network Address Translation (NAT)

- Private IP addresses shown below are used inside company/campus/organization or home network.

  10.0.0.0 to 10.255.255.255

  171.16.0.0 to 171.31.255.255

  192.168.0.0 to 192.168.255.255
- But company/campus/organization or home network connect to internet through global IP address.
- To convert from private IP address to global IP address and vice-versa NAT is used.
- NAT can be illustrated with the following diagram.



- Here the host with IP 10.0.0.1 sends the IP datagram with source address, port number 10.0.0.1, 3345 and destination address, port number 128.119.40.186, 80.
- NAT router maintains a NAT table as shown above
- NAT router make an entry in its table and replaces the source IP address with global IP address 138.76.29.7 and port number 5001 and send it to destination.
- When the response comes from destination, the global IP address will be replaced by private IP address according to entry available in NAT table. Then the message is delivered to appropriate host.

UPnP

- NAT traversal is increasingly provided by Universal Plug and Play (UPnP), which is 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).
- Furthermore, 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.

Internet Control Message Protocol (ICMP)

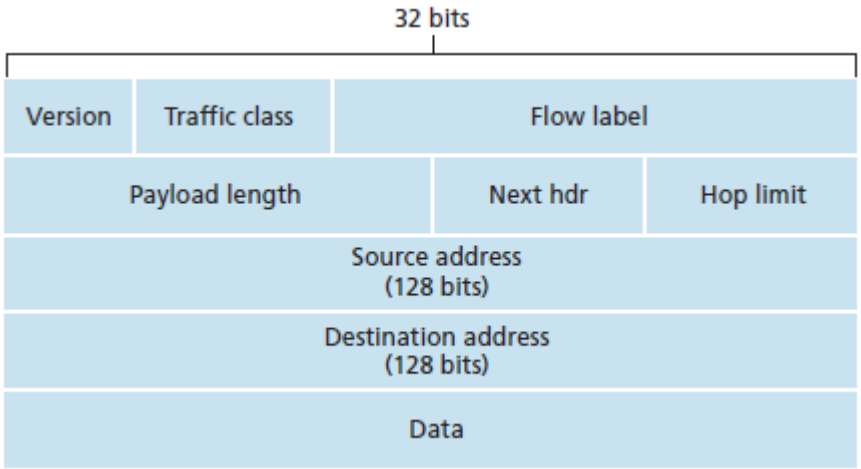
- ICMP is used by hosts and routers to communicate network- layer information to each other. The most typical use of ICMP is for error reporting.
- ICMP is part of IP but architecturally it lies just above IP hence ICMP messages are carried inside IP datagrams. That is, ICMP messages are carried as IP payload, just as TCP or UDP segments are carried as IP payload.
- ICMP messages have a type and a code field, and contain the header and the first 8 bytes of the IP datagram that caused the ICMP message to be generated in the first place.
- Popular ICMP messages are listed below

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

- 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.
- Congested router send an ICMP source quench message to a host toforce that host to reduce its transmission rate.

IPv6

IPv6 Datagram Format



The most important changes introduced in IPv6 are:

- **Expanded addressing capabilities:** IPv6 increases the size of the IP address from 32 to 128 bits.
- **A streamlined 40-byte header:** 40 bytes of mandatory header is used in IPv6 whereas IPv4 uses 20 bytes of mandatory header.
- **Flow labeling and priority:** Flow label refers to labeling 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. 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.

The following fields are defined in IPv6:

- **Version:** This 4-bit field identifies the IP version number.
- Traffic class: This 8-bit field specify priority.
- **Flow label:** this 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 next following 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:** 128 bit IPv6 address.
- **Data:** This is the payload portion of the IPv6 datagram.

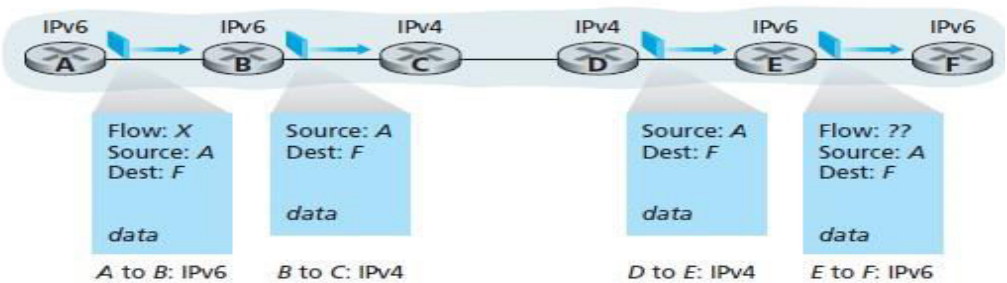
Following fields appearing in the IPv4 datagram are no longer present in the IPv6 datagram:

- **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.
- **Header checksum:** Because the transport-layer and link-layer protocols in the Internet layers perform check-summing, the designers of IP probably felt that this functionality was sufficiently redundant in the network layer that it could be removed.
- **Options:** An options field is no longer a part of the standard IP header. Instead of option field extension headers are used.

Transitioning from IPv4 to IPv6

1) Dual-stack:

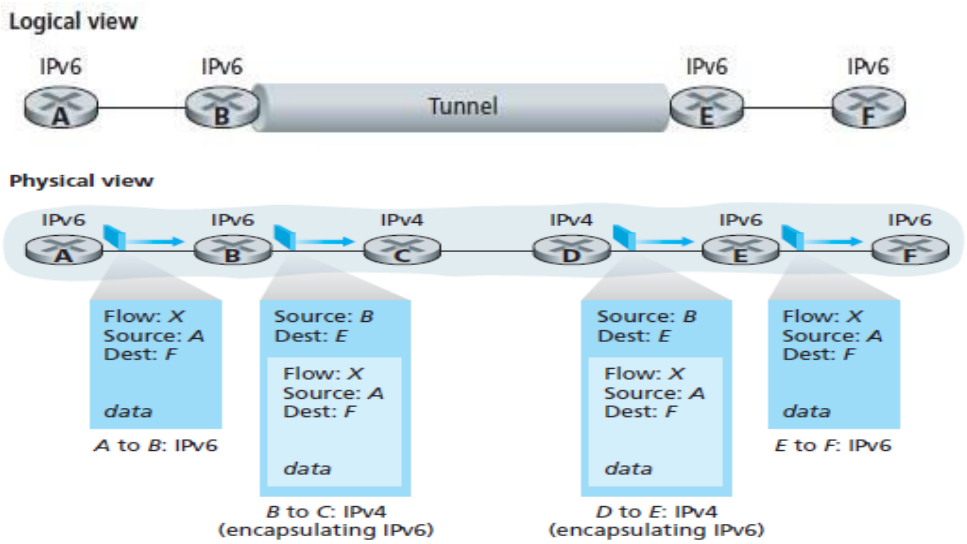
- Here IPv6 nodes also have a complete IPv4 implementation. Such a node, has the ability to send and receive both IPv4 and IPv6 datagrams. When interoperating with an IPv4 node, an IPv6/IPv4 node can use IPv4 datagrams; when interoperating with an IPv6 node, it can speak IPv6. IPv6/IPv4 nodes must have both IPv6 and IPv4 addresses. They must furthermore be able to determine whether another 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. As a result, it is possible that two IPv6- capable nodes can end up, in essence, sending IPv4 datagrams to each other.



Example: 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. However, 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. However, 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.

2) Tunneling

Tunneling can solve the problem noted above. The basic idea behind tunneling is the following. Suppose two IPv6 nodes want to interoperate using IPv6 datagrams but are connected to each other by intervening IPv4 routers. We refer to the intervening set of IPv4 routers between two IPv6 routers as a tunnel. With tunneling, the IPv6 node on the sending side of the tunnel takes the entire IPv6 datagram and puts it in the data (payload) field of an IPv4 datagram.



IP Security

IPsec is the security protocol used for IP security. 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 bedecrypted 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 en 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.

Network Security

Advances in the field of computer networks have made information security even more important. Computer systems have to be equipped with mechanisms for securing data. Computer networks need provisions that secure data from possible intrusions. Security is especially crucial in wireless networks, as a wireless medium, by its nature, is vulnerable to intrusions and attacks. This chapter focuses on *network security*. The following major topics in network security are covered.

- *Overview of network security: elements and threats*
- *Overview of security methods*
- *Secret-key encryption protocols*
- *Public-key encryption protocols*
- *Authentication: message digest and digital signatures*
- *Security of IP and wireless networks*
- *Firewalls*

We begin by identifying and classifying types of network threats, hackers, and attacks, including DNS hacking attacks and router attacks. Network security can be divided into two broad categories: *cryptographic techniques* and *authentication techniques* (verification). Both secret-key and public-key encryption protocols are presented. We then discuss message authentication and digital signature methods, through which a receiver can be assured that an incoming message is from whom it says it is.

OVERVIEW OF NETWORK SECURITY

Network security is a top-priority issue in data networks. As communication networks are growing rapidly, security issues have pushed to the forefront of concern for end users, administrators, and equipment suppliers. Despite enormous joint efforts by various groups to develop effective security solutions for networks, hackers continue to pose new, serious threats by taking advantage of weaknesses present in the Internet infrastructure.

Elements of Network Security

Network security is concerned mainly with the following two elements:

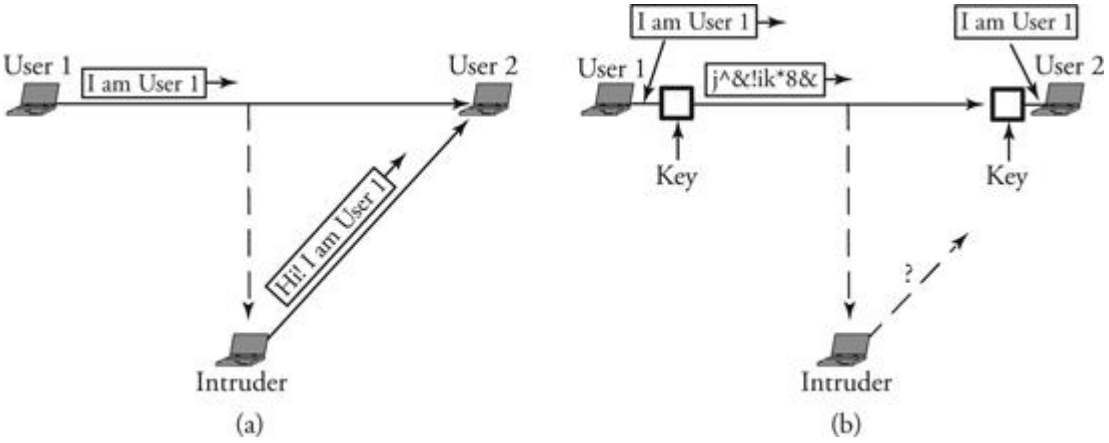
1. *Confidentiality*. Information should be available only to those who have rightful access to it.

2. *Authenticity and integrity*. The sender of a message and the message itself should be verified at the receiving point.

In Figure 10.1, user 1 sends a message (“I am user 1”) to user 2. In part (a) of the figure, the network lacks any security system, so an intruder can receive the message, 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) and that the content of the message is what user 1 (confidentiality). In part (b) of the figure, 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, who would be disabled in this communication transaction.

Figure 10.1 (a) Message content and sender identity falsified by intruder; (b) a method of applied security



In general, no protocol or network architecture can ensure full security. Internet routing is based on a distributed system of many routers, switches, and protocols. These protocols have a number of points of vulnerabilities that can be exploited to cause such problems as misdelivery or nondelivery of user traffic, misuse of network resources, network congestion and packet delays, and the violation of local routing policies.

Threats to Network Security

Internet infrastructure *attacks* are broadly classified into four categories, as follows:

1. DNS hacking
2. Routing table poisoning
3. Packet mistreatment
4. Denial of service

Among these threats, the first three attacks are related to network infrastructure; *denial-of-service attacks* are related to end systems.

DNS Hacking Attacks

As mentioned in Chap As mentioned in Chapter 9, the *Domain Name System* (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. In the normal mode of operation, hosts send UDP queries to the DNS server. Servers reply with a proper answer, or direct the queries to smarter servers. A DNS server also stores information other than host addresses.

Name-resolution services in the modern Internet environment are essential for e-mail 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* may result in the lack of data authenticity and integrity and can appear in any of the following forms:

1. An *information-level attack* forces a server to correspond with other than the correct answer. With cache poisoning, a hacker tricks a remote name server into caching the answer for a third-party domain by providing malicious information for the domain's authorized servers. Hackers can then redirect traffic to a preselected site.

2. In a *masquerading attack*, the adversary poses as a trusted entity and obtains all the secret information. In this guise, 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*.

3. The attacker normally sends queries to each host and receives in reply the DNS host name. In an *information leakage attack*, the attacker sends queries to all hosts and identifies which IP addresses are not used. Later on, the intruder can use those IP addresses to make other types of attacks.

4. Once a domain name is selected, it has to be registered. Various tools are available to register domain names over the Internet. If the tools are not smart enough, an invader might obtain secure information and use it to highjack the domain later. In the *domain highjacking attack*, whenever a user enters a domain address, she/he is forced to enter into the attacker's Web site. This can be very irritating and can cause a great loss of Internet usage ability.

Routing Table Poisoning Attacks

A *routing table poisoning attack* is the undesired modification of routing tables. An attacker can do this by maliciously modifying the routing information update packets sent by routers. This is a challenging and important problem, as a routing table is the basis of routing in the Internet. Any false entry in a routing table could lead to significant consequences, such as congestion, an overwhelmed host, looping, illegal access to data, and network partition. Two types of routing table poisoning attacks are the *link attack* and the *router attack*.

A *link attack* occurs when a hacker gets access to a link and thereby intercepts, interrupts, or modifies routing messages on packets. Link attacks act similarly on both the link-state and the distance-vector protocols discussed in Chapter 7. If an attacker succeeds in placing an attack in a link-state routing protocol, a router may send incorrect updates about its neighbours or remain silent even if the link state of its neighbor has changed. The attack through a link can be so severe that the attacker can program a router to either drop packets from a victim or readdress packets to a victim, resulting in a lower throughput of the network. Sometimes, a router can stop an intended packet from being forwarded further. However, since more than one path to any destination exists, the packet ultimately reaches its destination.

Router attacks may affect the link-state protocol or even the distance-vector protocol. If link-state protocol routers are attacked, they become malicious. They may add a nonexisting link to a routing table, delete an existing link, or even change the cost of a link. This attack may cause a router to simply ignore the updates sent by its neighbors, leading to a serious impact on the operability of the network traffic flow.

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.

Most unprotected routers have no way to validate updates. Therefore, both link-state and distance-vector router attacks are very effective. In the distance-vector protocol, for example, a malicious router can send wrong information in the form of a distance vector to all its neighbors. A neighbor may not be able to detect this kind of attack and thus proceeds to update its routing table, based on wrong distance vectors. The error can in turn be propagated to a great portion of the network before being detected.

Packet-Mistreatment Attacks

A *packet-mistreatment attack* can occur during any data transmission. A hacker may capture certain data packets and mistreat them. This type of attack is very difficult to detect. The attack may result in congestion, lowering throughput, and denial-of-service attacks. Similar to routing table poisoning attacks, packet-mistreatment attacks can also be subclassified into *link attacks* and *router attacks*. The link attack causes interruption, modification, or replication of data packets. A router attack can misroute all packets and may result in congestion or denial of service. Following are some examples of a packet-mistreatment attack:

• *Interruption*. If an attacker intercepts packets, they may not be allowed to be propagated to their destinations, resulting in a lower throughput of the network. This kind of attack cannot be detected easily, as even in normal operations, routers can drop some packets, for various reasons.

• *Modification*. Attackers may succeed in accessing the content of a packet while in transit and change its content. They can then change the address of the packet or even change its data. To solve this kind of problem, a digital signature mechanism, discussed later in this chapter, can be used.

• *Replication*. An attacker might trap a packet and replay it. This kind of attack can be detected by using the sequence number for each packet.

• *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.

• *Malicious misrouting of packets*. A hacker may attack a router and change its routing table, resulting in misrouting of data packets, causing a denial of service.

Denial-of-Service Attacks

A *denial-of-service attack* is a type of security breach that prohibits a user from accessing normally provided services. The denial of service does not result in information theft or any kind of information loss but can nonetheless be very dangerous, as it can cost the target person a large amount of time and money. Denial-of-service attacks affect the destination rather than a data packet or router.

Usually, a denial-of-service attack affects a specific network service, such as e-mail or DNS. For example, such an attack may overwhelm the DNS server in various ways and make it inoperable. One way of initiating this attack is by causing buffer overflow. Inserting an executable code inside memory can potentially cause a buffer overflow. Or, an adversary may use various tools to send large numbers of queries to a DNS server, which then is not able to provide services in a timely manner.

Denial-of-service attacks are easy to generate but difficult to detect. They take important servers out of action for few hours, thereby denying service to all users. There are yet a few other situations that can cause this kind of attack, such as UDP flood, a TCP flood and ICMP flood. In all these attacks, the hacker's main aim is to overwhelm victims and disrupt services provided to them.

Denial-of-service attacks are two types

1. *Single-source*. An attacker sends a large number of packets to a target system to overwhelm and disable it. These packets are designed such that their real sources cannot be identified.

2. *Distributed*. In this type of attack, 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, as it is processing the flood of traffic.

The flood may be either a UDP flood or a TCP SYN flood. UDP flooding is used against two target systems and can stop the services offered by either system. Hackers link the UDP character-generating services of a system to another one by sending UDP packets with spoofed return addresses. This may create an infinite looping between the two systems, leading to system uselessness.

Normally, a SYN packet is sent by a host to a user who intends to establish a connection. The user then sends back an acknowledgment. In the TCP SYN flood, a hacker sends a large number of SYN packets to a target user. Since the return addresses are spoofed, the target user queues up a SYN/ACK packet and never processes it. Therefore, the target system keeps on waiting. The result may be a hard disk crash or reboot.

SECURITY METHODS

Common solutions that can protect computer communication networks from attacks are classified as *cryptographic techniques* or *authentication techniques* (verification).

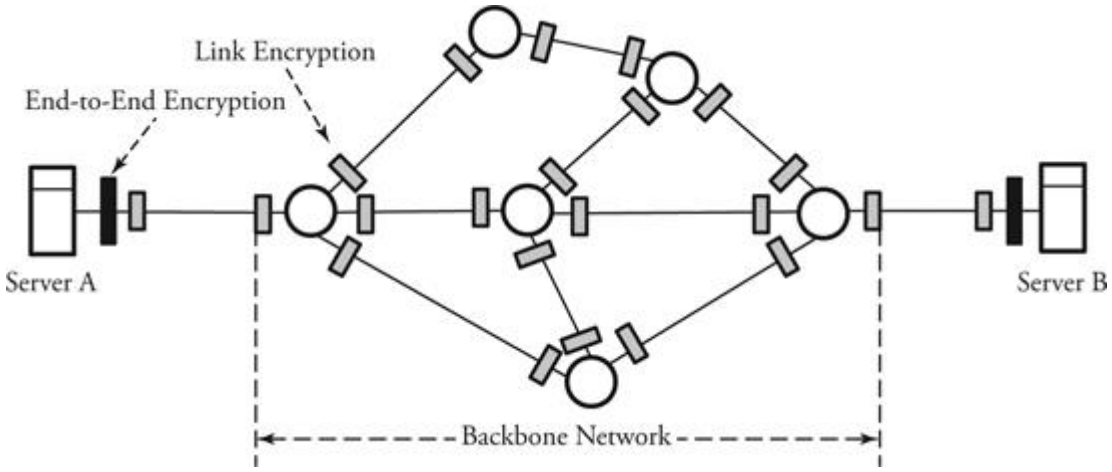
Cryptographic Techniques

Cryptography has a long and fascinating history. Centuries ago, cryptography was used as a tool to protect national secrets and strategies. Today, network engineers focus on *cryptography* methods for computer communication networks. Cryptography is the process of tranforming 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. This kind of message needs to be decoded at the receiving end before any further processing.

The main tool that network security experts are using to encrypt a message M is a secret key K; the fundamental operation often used to encrypt a message is the Exclusive-OR ($\oplus$). Suppose 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—if he/she has the key, K—can easily detect M by performing the following:

In computer communication networks, data can travel between two users while it is encrypted. In Figure 10.2, two servers are exchanging data while two types of encryption devices are installed in their communication network. The first encryption device is *end-to-end encryption*, whereby secret coding is carried out at both end systems. In this figure, server A encodes its data, which can be decoded only at the other end server. The second phase of encryption is *link encryption*, which secures all the traffic passing over that link.

Figure 10.2 Overview of encryption points in a communication network



The two types of encryption techniques are *secret-key encryption* and *public-key encryption*. In a secret-key model, both sender and receiver conventionally use the same key for an encryption process. In a 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 and provides better security and message privacy. But the biggest drawback of public-key encryption is speed. The public-key system is significantly more complex computationally and may not be practical in many cases. Hence, the public-key system is used only to establish a session to exchange a session key. Then, this session key is used in a secret-key system for encrypting messages for the duration of the session.

Authentication Techniques

Encryption methods offer the assurance of message confidentiality. However, a networking system must be able to verify the authenticity of the message and the sender of the message. These forms of security techniques in computer networks are known as *authentication techniques* and are categorized as *authentication with message digest* and *authentication with digital signature.* Message authentication protects a user in a network against data falsification and ensures data integrity. These methods do not necessarily use keys.

SECRET-KEY ENCRYPTION PROTOCOLS

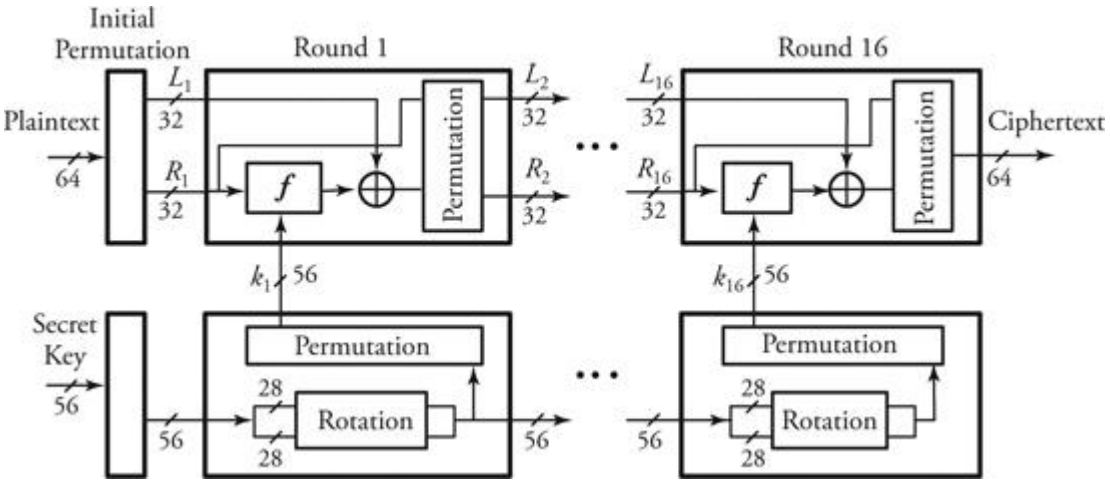
Secret-key encryption protocols, sometimes known as *symmetric encryption*, or *single-key encryption* protocols, are conventional encryption models. They typically consist of an encryption algorithm, a key, and a decryption algorithm. At the end point, the encrypted message is called *ciphertext*. Several standard mechanisms can be used to implement a secret-key encryption algorithm. Here, we focus on two protocols: *Data Encryption Standard* (DES) and *Advanced Encryption Standard* (AES).

In these algorithms, a shared secret key between a transmitter and a receiver is assigned at the transmitter and receiver points. The encryption algorithm produces a different key at any time for a specific transmission. Changing the key changes the output of the algorithm. At the receiving end, the encrypted information can be transformed back to the original data by using a decryption algorithm and the same key that was used for encryption. The security of conventional encryption depends on the secrecy of the key, not on the secrecy of the encryption algorithm. Consequently, the algorithm need not be kept secret; only the key has to be secret.

Data Encryption Standard (DES)

With the *Data Encryption Standard* (DES), plaintext messages are converted into 64-bit blocks, each encrypted using a key. The key length is 64 bits but contains only 56 usable bits; thus, the last bit of each 8 byte in the key is a parity bit for the corresponding byte. DES consists of 16 identical rounds of an operation, as shown in Figure 10.3. The details of the algorithm on each 64-bit block of message at each round *i* of operation are as follows.

Figure 10.3 The Data Encryption Standard (DES)



Begin DES Algorithm

1. Initialize. Before round 1 begins, all 64 bits of an incoming 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 L_i and R_i, respectively.

3. The 56 bits of the key are also broken into two 28-bit 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 k_i of the key on round *i*.

5. In this step, is a logic Exclusive-OR, and the description of function *F*() appears next.
Then, L_i and R_i are determined by

(10.2)

$$L_i = R_{i-1}$$

and

(10.3)

$$R_i = L_{i-1} \oplus F(R_{i-1}, k_i).$$

6. All 64 bits of a message are permuted.

The operation of function *F*() at any round *i* of DES is as follows.

1. Out of 52 bits of k_i, function *F*() chooses 48 bits.

2. 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.

3. Function $F()$ also partitions the 48 bits of k_i into eight 6-bit chunks.

4. The corresponding eight chunks of R_{i-1} and eight chunks of k_i are combined as follows:

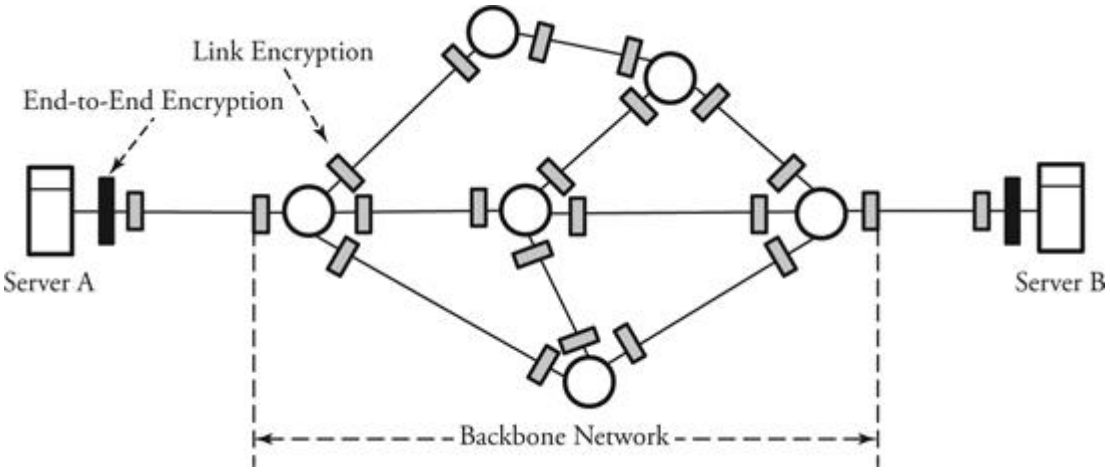
(10.4) $$R_{i-1} = R_{i-1} \oplus k_i.$$

At the receiver, the same steps and the same key are used to reverse the encryption. It is now apparent that the 56-bit key length may not be sufficient to provide full security. This argument is still controversial. Triple DES provides a solution for this controversy: three keys are used, for a total of 168 bits. It should also be mentioned that DES can be implemented more efficiently in hardware than in software.

Advanced Encryption Standard (AES)

The *Advanced Encryption Standard* (AES) protocol has a better security strength than DES. AES supports 128-bit symmetric block messages and uses 128-, 192-, or 256-bit keys. The number of rounds in AES is variable from 10 to 14 rounds, depending on the key and block sizes. Figure 10.4 illustrates the encryption overview of this protocol, using a 128-bit key. There are ten rounds of encryptions for the key size of 128 bits. All rounds are identical except for the last round, which has no mix-column stage.

Overview of Advanced Encryption Standard (AES) protocol



A single block of 128-bit plaintext (16 bytes) as an input arrives from the left. The plaintext is formed as 16 bytes m_0 through m_{15} and is fed into round 1 after an initialization stage. In this round, substitute units—indicated by S in the figure—perform a byte-by-byte substitution of blocks. The ciphers, in the form of rows and columns, 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 $k_0(1)$ through $k_{15}(1)$. The 128-bit key is expanded for ten rounds. The AES *decryption algorithm* is fairly simple and is basically the reverse of the encryption algorithm at each stage of a round. All stages of each round are reversible.

PUBLIC-KEY ENCRYPTION PROTOCOLS

The introduction of *public-key encryption* brought a revolution to the field of cryptography. Public-key cryptography provided a very clever method for key exchange. In the public-key encryption model, a sender/receiver pair uses different keys. This model is sometimes known as *asymmetric*, or *two-key*, *encryption*.

Public-key algorithm is based on mathematical functions rather than on substitution or permutation, although the security of any encryption scheme indeed depends on the length of the key and the computational work involved in breaking an encrypted message. Several public-key encryption protocols can be implemented. Among them, the following two protocols are the focus of our study:

- *Rivert, Shamir, and Aldeman* (RSA) protocol
- *Diffie-Hillman key-exchange* protocol.

In the public-key encryption methods, either of the two related keys can be used for encryption; the other one, for decryption. It is computationally infeasible to determine the decryption key given only the algorithm and the encryption key. Each system using this encryption method generates a pair of keys to be used for encryption and decryption of a message that it will receive. Each system publishes its encryption key by placing it in a public register or file and sorts out the key as a 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. On receiving the message, *B* decrypts it with the private *B* key. No other recipients can decrypt the message, since only *B* knows its private key. This way, public-key encryption secures an incoming communication as long as a system controls its private key. However, public-key encryption has extra computational overhead and is more complex than the conventional one.

RSA Algorithm

Rivert, Shamir, and Aldeman developed the RSA public-key encryption and signature scheme. This was the first practical public-key encryption algorithm. RSA is based on the intractability of factoring large integers. Assume that a plaintext *m* must be encrypted to a ciphertext *c*. The RSA algorithm has three phases for this: *key generation*, *encryption*, and *decryption*.

Key Generation

In the RSA scheme, the key length is typically 512 bits, which requires an enormous computational power. A plaintext is encrypted in blocks, with each block having a binary value less than some number *n*. Encryption and decryption are done as follows, beginning with the generation of a public key and a private key.

Begin Key Generation Algorithm

1. Choose two roughly 256-bit prime numbers, *a* and *b*, and derive $n = ab$. (A number is prime if it has factors of 1 and itself.)

2. Find *x*. Select encryption key *x* such that *x* and $(a - 1)(b - 1)$ are relatively prime. (Two numbers are relatively prime if they have no common factor greater than 1.)

3. Find *y*. Calculate decryption key *y*:

(10.5) $$xy \bmod (a-1)(b-1) = 1.$$

4. At this point, a and b can be discarded.

5. The public key = $\{x, n\}$.

6. The private key = $\{y, n\}$.

In this algorithm, x and n are known to both sender and receiver, but only the receiver must know y. Also, a and b must be large and about the same size and both greater than 1,024 bits. The larger these two values, the more secure the encryption.

Encryption

Both sender and receiver must know the value of n. The sender knows the value of x, and only the receiver knows the value of y. Thus, this is a public-key encryption, with the public key $\{x, n\}$ and the private key $\{y, n\}$. Given $m < n$, ciphertext c is constructed by

(10.6) $$c = m^x \bmod n.$$

Note here that if a and b are chosen to be on the order of 1,024 bits, $n \approx 2,048$. Thus, we are not able to encrypt a message longer than 256 characters.

Decryption

Given the ciphertext, c, the plaintext, m, is extracted by

(10.7) $$m = c^y \bmod n.$$

In reality, the calculations require a math library, as numbers are typically huge. One can see easily how Equations (10.6) and (10.7) work.

Example. For an RSA encryption of a 4-bit message of 1,000, or $m = 9$, we choose $a = 3$ and $b = 11$. Find the public and the private keys for this security action, and show the ciphertext.

Solution. Clearly, $n = ab = 33$. We select $x = 3$, which is relatively prime to $(a - 1)(b - 1) = 20$. Then, from $xy \bmod (a - 1)(b - 1) = 3y \bmod 20 = 1$, we can get $y = 7$. Consequently, the public key and the private key should be {3, 33} and {7, 33}, respectively. If we encrypt the message, we get $c = m^x \bmod n = 9^3 \bmod 33 = 3$. The decryption process is the reverse of this action, as $m = c^y \bmod n = 3^7 \bmod 33 = 9$.

Diffie-Hillman Key-Exchange Protocol

In the *Diffie-Hillman key-exchange* protocol, two end users can agree on a shared secret code without any information shared in advance. Thus, intruders would not be able to access the transmitted communication between the two users or discover the shared secret code. This protocol is normally used for *virtual private networks* (VPNs), explained in Chapter 16. The essence of this protocol for two users, 1 and 2, is as follows. Suppose that user 1 selects a prime a, a random integer number x_1, and a generator g and creates y_1 $\{1, 2, ..., a - 1\}$ such that

(10.8) $$y_1 = g^{x_1} \bmod a.$$

In practice, the two end users agree on a and g ahead of time. User 2 performs the same function and creates y_2:

(10.9) $$y_2 = g^{x_2} \bmod a.$$

User 1 then sends y_1 to user 2. Now, user 1 forms its key, k_1, using the information its partner sent as

(10.10) $$k_1 = y_2^{x_1} \bmod a,$$

and user 2 forms its key, k_2, using the information its partner sent it as

(10.11) $$k_2 = y_1^{x_2} \bmod a.$$

It can easily be proved that the two Keys k_1 and k_2 are equal. Therefore, the two users can now encrypt their messages, each using its own key created by the other one's information.

AUTHENTICATION

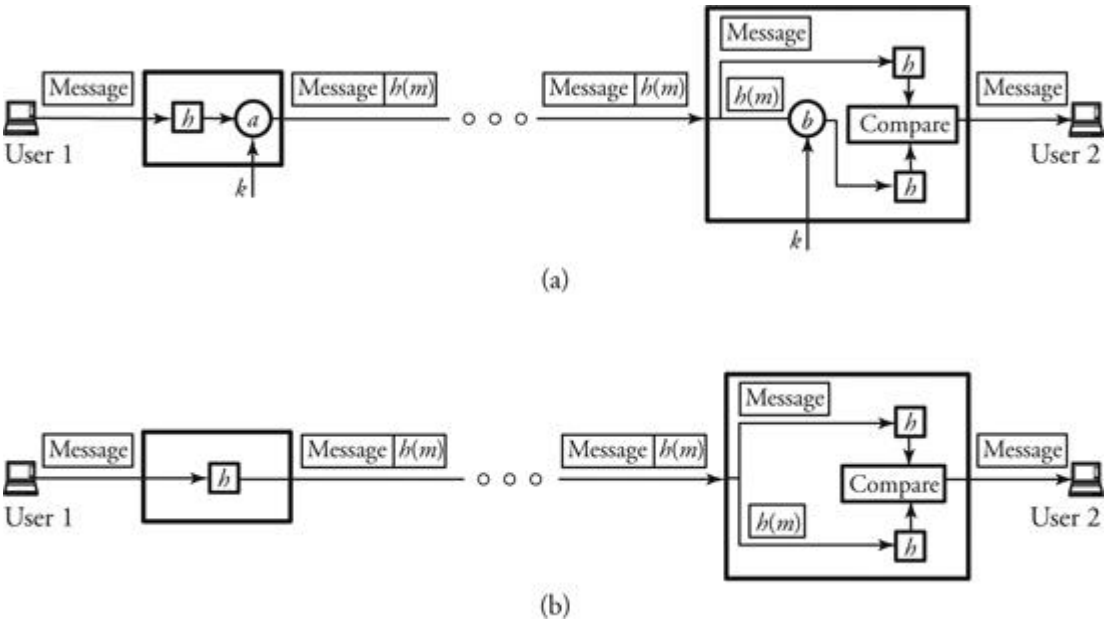
Authentication techniques are used to verify identity. Message authentication verifies the authenticity of both the message content and the message sender. Message content is authenticated through implementation of a hash function and encryption of the resulting message digest. The sender's authenticity can be implemented by use of a digital signature.

A common technique for authenticating a message is to implement a *hash function*, which is used to produce a "fingerprint" of a message. The hash value is added at the end of message before transmission. The receiver recomputes 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, or $h(m)$. The hash function has the following properties.

- Unlike the encryption algorithm, the authentication algorithm is not required to be reversible.
- Given a message digest $h(m)$, it is computationally infeasible to find m.
- It is computationally infeasible to find two different messages m_1 and m_2 such that $h(m_1) = h(m_2)$.

Message authentication can be implemented by two methods. In the first method, as shown in Figure 10.5 (a), a hash function is applied on a message, and then a process of encryption is implemented. Thus, a message digest can also be encrypted in this method. At this stage, the encryption can be a public key or a secret key. The authenticity of a message in this method is assured only if the sender and the receiver share the encryption key. At the receiver site, the receiving user 2 has to decrypt the received message digest and compare it with the one made locally at its site for any judgments on the integrity of the message.

Figure 10.5 Message authentication: (a) combined with encryption; (b) use of the hash function



In the second method, as shown in Figure 10.5 (b), no encryption is involved in the process of message authentication. This method assumes that 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. This technique is more popular in the security infrastructure of the Internet Protocol. Among the message authentication protocols are the MD5 *hash algorithm* and the *Secure Hash Algorithm* (SHA). SHA is the focus of our discussion.

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. The number of padding bits can be as low as 1 bit and as high as 512 bits. The padding includes a 1 bit and as many 0 bits as required. Therefore, the least-significant 64 bits of the message length are appended to convert the padded message to a word with a 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:

(10.12) $w_i = m_i \text{ for } 0 \le i \le 15$

And

(10.13) $w_i = w_{i-3} \oplus w_{i-8} \oplus w_{i-14} \oplus w_{i-16} \hookleftarrow 1 \text{ for } 16 \le i \le 79,$

Where $\hookleftarrow j$ means left rotation by j bits. This way, bits are shifted several times if the incoming block is mixed with the state. Next, bits from each block of w_i are mixed into the state in four steps, each maintaining 20 rounds. For any values of a, b, and c, and bit number i, we define a function $F_i(a, b, c)$ as follows:

(10.14)

$$F_i(a,b,c) = \begin{cases} (a \cap b) \cup (\bar{a} \cap c) & 0 \le i \le 19 \\ a \oplus b \oplus c & 20 \le i \le 39 \\ (a \cap b) \cup (a \cap c) \cup (b \cap c) & 40 \le i \le 59 \\ a \oplus b \oplus c & 60 \le i \le 79 \end{cases}.$$

Then, the 80 steps (i = 0, 1, 2, ..., 79) of the four rounds are described as follows:

(10.15)

$$\delta = (R_1 \hookleftarrow 5) + F_i(R_2, R_3, R_4) + R_5 + w_i + C_i$$

(10.16) $$R_5 = R_4$$

(10.17) $$R_4 = R_3$$

(10.18) $$R_3 = R_2 \hookleftarrow 30$$

(10.19) $$R_2 = R_1$$

(10.20) $$R_1 = \delta,$$

where C_i is a constant value specified by the standard for round i. The message digest is produced by concatenation of the values in R_1 through R_5.

Authentication and Digital Signature

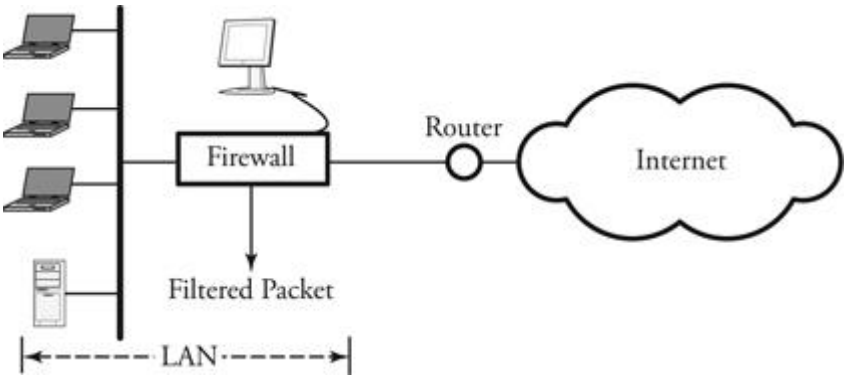
A digital signature is one of the most important required security measures. Much like a person's signature on a document, a digital signature on a message is required for the authentication and identification of the right sender. The digital signature is supposed to be unique to an individual and serves as a means of identifying the sender. An electronic signature is not as easy as it was with the paper-based system. The digital signature requires a great deal of study, research, and skill. Even if these requirements are met, there is no guarantee that all the security requirements have been met.

The technical method of providing a sender's authentication is performed through cryptography. Many cryptographic mechanisms have been developed. Among them, the RSA algorithm implements both encryption and digital signature. When RSA is applied, the message is encrypted with the sender's private key. Thus, the entire encrypted message serves as a digital signature. This means that at the receiving end, the receiver can decrypt it, using the public key. This authenticates that the packet comes from the right user .

FIREWALLS

As the name suggests, a *firewall* protects data from the outside world. A firewall can be a software program or a hardware device. A firewall is a popular security mechanism for networks. A firewall is a simple router implemented with a special program. This unit is placed between hosts of a certain network and the outside world, as shown in Figure 10.8, and the rest of the network. The security issues faced by a smaller network like the one used at home are similar to larger networks. A firewall is used to protect the network from unwanted Web sites and potential hackers.

Figure 10.8 A simple configuration of a secured network using a firewall



A firewall is placed on the link between a network router and the Internet or between a user and a router. The objective of such a configuration is to monitor and filter packets coming from unknown sources. Consequently, hackers do not have access to penetrate through a system if a firewall protects the system. For a large company with many small networks, the firewall is placed on every connection attached to the Internet. Companies can set rules about how their networks or particular systems need to work in order to maintain security. Companies can also set rules on how a system can connect to Web sites. These precautionary rules are followed in order to attain the advantage of having a firewall. Hence, the firewall can control how a network works with an Internet connection. A firewall can also be used to control data traffic.

Software firewall programs can be installed in home computers by using an Internet connection with these so-called gateways, the computer with such a software can access Web servers only through this software firewall. But hardware firewalls are more secure than software firewalls. Moreover, hardware firewalls are not expensive. Some firewalls also offer virus protection. The biggest security advantage of installing a firewall in a business network is to protect from any outsider logging on to the network under protection. Firewalls are preferred for use in almost all network security infrastructures, as they allow the implementation of a security policy in one centralized place rather than end to end. Sometimes, a firewall is put where a large amount of data flow is normally expected.

A firewall controls the flow of traffic by one of the following three methods. The first method is *packet filtering*. Apart from forwarding packets between networks, a firewall filters those packets that pass through. If packets can get through the filter, they reach their destinations; otherwise, they are discarded. A firewall can be programmed to throw away certain packets addressed to a particular IP host or TCP port number. This condition is especially useful if a particular host does not want to respond to any access from an external source.

The second method is that 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. The third method, denial of service, was explained earlier. This method controls the number of packets entering a network.

Internet routing is based on a distributed system of many routers, switches, and protocols. These protocols have a number of points of vulnerabilities that can be exploited to cause such problems as misdelivery or nondelivery of user traffic, misuse of network resources, network congestion and packet delays, and the violation of local routing policies.

Module – 5

MULTIMEDIA NETWORKING

Multimedia Networking Applications

A multimedia network application can be defined as any application that employs audio or Video.

Properties of Video

1) High Bit Rate

- Video distributed over the Internet use
    - 100 kbps for low-quality video conferencing.
    - 3 Mbps for streaming high-definition (HD) movies.
- The higher the bit-rate,
    - Better the image quality and
    - Better the overall user viewing experience.
- A video can be compressed, thereby trading off video-quality with bit-rate.
- A video is a sequence of images, displayed at a constant rate.
- An uncompressed digital image consists of an array of pixels.
- Each pixel is encoded into a number of bits to represent luminance and color.
- There are two types of redundancy in video:

An image that consists of mostly white space has a high degree of redundancy. These images can be efficiently compressed without sacrificing image quality.

2) Temporal Redundancy

Temporal redundancy reflects repetition from image to subsequent image. For example: If image & subsequent image are same, re-encoding of subsequent image can be avoided.

Properties of Audio

- PCM (Pulse Code Modulation) is a technique used to change an analog signal to digital data (digitization).
- PCM consists of 1) Encoder at the sender and 2) Decoder at the receiver.

PCM Encoder

- Digital audio has lower bandwidth requirements than video.
- Consider how analog audio is converted to a digital-signal:
- The analog audio-signal is sampled at some fixed rate. This operation is referred to as sampling.
- For example: 8000 samples per second.
- The value of each sample is an arbitrary real number.

Each sample is then rounded to one of a finite number of values. This process is called quantization. The number of such finite values is called as quantization-values. The number of quantization-values is typically a power of 2. For ex: 256(28) quantization-values. Each of the quantization-values is

represented by a fixed number of bits. Bit representations of all values are then concatenated to form digital representation of the signal. This process is called encoding.

PCM Decoder

For playback through audio speakers, the digital-signal can be converted back to an analog signal. This process is called decoding.

The sound quality may be noticeably degraded.
The decoded signal can better approximate the original analog-signal by increasing
i) Sampling rate and
ii) Number of quantization-values,
Thus, there is a trade-off between
Quality of the decoded signal and
Bit-rate & storage requirements of the digital-signal.

Types of Multimedia Network Applications

Three broad categories of multimedia applications:
1) Streaming stored audio/video
2) Conversational voice/video-over-IP and
3) Streaming live audio/video.

Streaming Stored Audio & Video

- The underlying medium is prerecorded video. For example: a movie.
- These prerecorded videos are placed on servers.
- The users send requests to the servers to view the videos on-demand. Nowadays, many Internet companies provide streaming video. For example: YouTube.
- Three key distinguishing features of streaming stored video:
- The client begins video playout within few seconds after it begins receiving the video from the server.

• At the same time,
i) The client will be playing out from one location in the video.
ii) The client will be receiving later parts of the video from the server.
This technique avoids having to download the entire video-file before playout begins.
The media is pre-recorded, so the user may pause, reposition or fast-forward through video content.
The response time should be less than a few seconds. Once playout of the video begins, it should proceed according to the original timing of the recording. The data must be received from the server in time for its playout at the client. Otherwise, users experience video-frame skipping (or freezing).

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 includes the video of the participants as well as their voices. Most of today's voice applications allow users to create conferences with three or more Participants. Most Internet companies provide voice application.
For example: Skype & Google Talk.
Two parameters are particularly important for voice applications:
1) Timing considerations and
2) Tolerance of data loss

Timing considerations are important because voice applications are highly delay-sensitive. Loss-tolerant means Occasional loss only causes occasional glitches in audio playback & these losses can be partially/fully hidden.

Streaming Live Audio & Video

These applications are similar to broadcast radio, except that transmission takes place over Internet. These applications allow a user to receive a live radio transmitted from any corner of the world.
For example: live cricket commentary. Today, thousands of radio stations around the world are broadcasting content over the Internet.

Streaming Stored Video

- Prerecorded videos are placed on servers.
- Users send requests to these servers to view the videos on-demand.
- The media is pre-recorded, so the user may pause, reposition or fast-forward through video content.

Three categories of applications:

1) UDP streaming
2) HTTP streaming and
3) Adaptive HTTP streaming.

A main characteristic of video-streaming is the extensive use of client-side buffering. Two advantages of client-side buffering:

1) Client-side buffering can mitigate effects of varying end-to-end delays
2) This can mitigate effects of varying amounts of available bandwidth b/w server & client.

UDP Streaming

- The server transmits video at a rate that matches the client‟s video consumption rate.
- The server transmits the video-chunks over UDP at a steady rate.
- UDP does not employ a congestion-control mechanism.
- Therefore, the server can push packets into the network at the video consumption rate.
- Typically, UDP streaming uses a small client-side buffer. (RTP Real-Time Transport Protocol).
- Using RTP, the server encapsulates the video-chunks within transport packets.
- The client & server also maintain a control-connection over which the client sends commands (such as pause, resume and reposition).
- The RTSP (Real-Time Streaming Protocol) is a popular open protocol for a controlconnection.

• Disadvantages:

1) UDP streaming can fail to provide continuous playout of varying amount of available bandwidth
2) Costly & Complex
A media control server (RTSP) is required
→ to process client-to-server interactivity requests and
→ to track client-state for each ongoing client-session.

This increases the overall cost and complexity of deploying a large-scale application. Many firewalls are configured to block UDP traffic. This prevents the users behind the firewalls from receiving the video.

HTTP Streaming

The video is stored in an HTTP server as an ordinary file with a specific URL.
it works as folows:

1) 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.
2) Then, the server responds with the video file, within an HTTP response message.
3) On client side, the bytes are collected in a client application buffer.
4) Once no. of bytes in this buffer exceeds a specific threshold, the client begins playback.

• **Advantages:**
1) Not Costly & Complex
   Streaming over HTTP avoids the need for a media control server (RTSP). This reduces the cost of deploying a large-scale application.

2) No Firewall Problem
The use of HTTP over TCP also allows the video to traverse firewalls and NATs more easily.

3) Prefetching Video
The client downloads the video at a rate higher than the consumption rate. This prefetched video is stored in the client application buffer. Most of Video-streaming applications use HTTP streaming. For example: YouTube.

DASH

The video is encoded into several different versions. Each version has a different bit-rate and a different quality level.

Two main tasks:
1) The client dynamically requests video-chunks from the different versions: low & high.
   i) When the available bandwidth is high, the client selects chunks from a highrate version.
   For ex: Fiber connections can receive a high-quality version.
   ii) When the available bandwidth is low, the client naturally selects from a low-rate version.
   For ex: 3G connections can receive a low-quality version.
2) The client adapts to the available bandwidth if end-to-end bandwidth changes during session.
   This feature is particularly important for mobile-users. The mobile-users see their bandwidth fluctuate as they move with respect to base-stations.

HTTP server stores following files:
1) Each video version with a different URL.
2) Manifest file provides a URL for each version along with its bit-rate.

• Here is how it works:
1) First, the client requests the manifest file and learns about the various versions.
2) Then, the client selects one chunk at a time by specifying URL and byte range in an HTTP GET request message.

3) While downloading chunks, the client
→ measures the received bandwidth and
→ runs a rate determination-algorithm.
i) If measured-bandwidth is high, client will choose chunk from high-rate version.
ii) If measured-bandwidth is low, client will choose chunk from low-rate version
4) Therefore, DASH allows the client to freely switch among different quality-levels.

Content Distribution Networks

Motivation for CDN

The streaming video service can be provided is as follows:

1) Build a single massive data-center.
2) Store all videos in the data-center and
3) Stream the videos directly from the data-center to clients worldwide.

Three major problems with the above approach:

1) More Delay
If links provides a throughput lesser than consumption-rate, the end-to-end throughput will also be below the consumption-rate. This results in freezing delays for the user.
2) Network Bandwidth is wasted A popular video may be sent many times over the same links.
3) Single Point of Failure: If the data-center goes down, it cannot distribute any video streams.

CDN Types

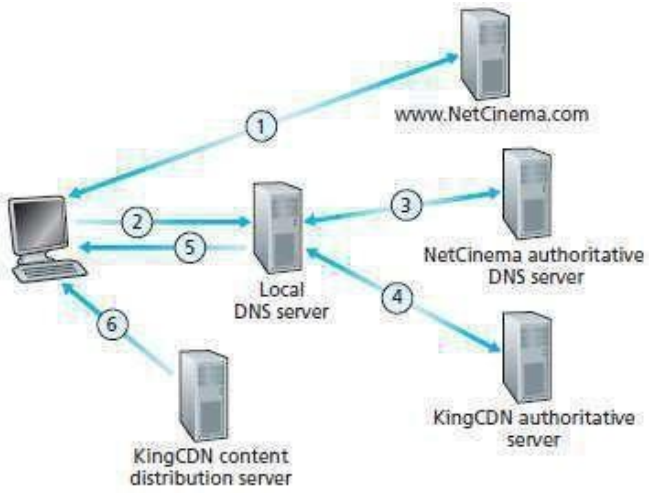
A CDN
→ Manages servers in multiple geographically distributed locations
→ Stores copies of the videos in its servers, and
→ Attempts to direct each user-request to a CDN that provides the best user experience.

The CDN may be a private CDN or a third-party CDN.
A private CDN is owned by the content provider itself. For example: Google‟s CDN distributes YouTube videos
A third-party CDN distributes content on behalf of multiple content providers CDNs.

CDN Operation

When a browser wants to retrieve a specific video, the CDN intercepts the request.
• Then, the CDN
1) Determines a suitable server-cluster for the client and
2) Redirects the client's request to the desired server.

Most CDNs take advantage of DNS to intercept and redirect requests.
• CDN operation is illustrated in Figure 5.2.



Voice-over-IP

Real-time voice over the Internet is often referred to as Internet telephony. It is also commonly called Voice-over-IP (VoIP).

Limitations of the Best-Effort IP Service

The Internet‟s network-layer protocol IP provides best-effort service.
The IP makes best effort to move each datagram from source to destination.
But IP does not guarantee deliver of the packet to the destination.

Three main challenges to the design of real-time applications:
1) Packet-loss
2) Packet delay and
3) Packet jitter.

Packet Loss

By default, most existing VoIP applications run over UDP. The UDP segment is encapsulated in an IP datagram. The datagram passes through router buffers in the path from sender to receiver

Problem:

There is possibility that one or more buffers are full. In this case, the arriving IP datagram may be discarded.

Possible solution:

Loss can be eliminated by sending the packets over TCP rather than over UDP. However, retransmissions are unacceptable for real-time applications „.“ they increase delay. Packet-loss results in a reduction of sender‟s transmission rate, Leading to buffer starvation.

End-to-End Delay

End-to-end delay is the sum of following delays:
1) Transmission, processing, and queuing delays in routers.

2) Propagation delays in links and
3) Processing delays in end-systems.

For VoIP application,

→ Delays smaller than 150 msecs are not perceived by a human listener.
→ Delays between 150 and 400 msecs can be acceptable but are not ideal and
→ Delays exceeding 400 msecs can seriously hinder the interactivity in voice conversations.

Typically, the receiving-side will discard any packets that are delayed more than certain threshold. . For example: more than 400 msecs.

Packet Jitter

Jitter refers to varying queuing delays that a packet experiences in the network‟s routers.
If the receiver

→ ignores the presence of jitter and
→ plays out audio-chunks, then the resulting audio-quality can easily become unintelligible. • Jitter can often be removed by using sequence numbers, timestamps, and a playout delay

Removing Jitter at the Receiver for Audio

For VoIP application, receiver must provide periodic playout of voice-chunks in presence of random jitter. This is typically done by combining the following 2 mechanisms:

1) Prepending each Chunk with a Timestamp. The sender attaches each chunk with the time at which the chunk was generated.
2) Delaying Playout of Chunks at the Receiver

The playout delay of the received chunks must be long. So, the most of the packets are received before their scheduled playout times. This playout delay can either be

→ fixed throughout the duration of the session or
→ vary adaptively during the session- lifetime.

Recovering from Packet Loss

Loss recovery schemes attempt to preserve acceptable audio-quality in the presence of Packet-loss.
Here, packet-loss is defined in a 2 broad sense:

i) A packet is lost if the packet never arrives at the receiver or
ii) A packet is lost if the packet arrives after its scheduled playout time.

VoIP applications often use loss anticipation schemes.
Here, we consider 2 types of loss anticipation schemes:

1) Forward error correction (FEC) and
2) Interleaving.

FEC

• The basic idea of FEC: Redundant information is added to the original packet stream.
• The redundant information can be used to reconstruct approximations of some of the lost packets.

Two FEC mechanisms:

1) Block Coding

A redundant encoded chunk is sent after every n chunk. The redundant chunk is obtained by exclusive OR-ing the n original chunks. If anyone packet in a group is lost, the receiver can fully reconstruct the lost-packet.

Disadvantages:

If 2 or more packets in a group are lost, receiver cannot reconstruct the lost packets.

2) Increases the playout delay. This is because

→ receiver must wait to receive entire group of packets before it can begin playout.

→ A Lower Resolution Redundant Information

→ A lower-resolution audio-stream is sent as the redundant information. For example: The sender creates nominal audio-stream and corresponding low-resolution, low-bit-rate audio-stream. The low-bit-rate stream is referred to as the redundant-stream. The sender constructs the nth packet by

→ taking the nth chunk from the nominal stream and

→ appending the nth chunk to the (n–1)st chunk from the redundant-stream.

Advantage:

Whenever there is packet-loss, receiver can hide the loss by playing out low-bit rate chunk.

Protocols for Real-Time Conversational Applications

- Real-time applications are very popular. For ex: VoIP and video conferencing.
- Two standards bodies are working for real-time applications: 1) IETF and 2) ITU
- Both standards (IETF & ITU) are enjoying widespread implementation in industry products.

RTP

RTP can be used for transporting common formats such as

→ MP3 for sound and

→ MPEG for video

It can also be used for transporting proprietary sound and video formats. Today, RTP enjoys widespread implementation in many products and research prototypes. It is also complementary to other important real-time interactive protocols, such as SIP.

RTP Basics

RTP runs on top of UDP.

The RTP packet is composed of i) RTP header & ii) audio chunk

The header includes

i) Type of audio encoding

ii) Sequence number and

iii) Timestamp.

• The application appends each chunk of the audio-data with an RTP header.

• Here is how it works:

1) At sender-side:

i) A media chunk is encapsulated within an RTP packet.

ii) Then, the packet is encapsulated within a UDP segment.

iii) Finally, the UDP segment is handed over to IP.

2) At receiving-side:

i) The RTP packet is extracted from the UDP segment.

ii) Then, the media chunk is extracted from the RTP packet.

iii) Finally, the media chunk is passed to the media-player for decoding and rendering

If an application uses RTP then the application easily interoperates with other multimedia application. For example: If 2 different companies use RTP in their VoIP product, then users will be able to communicate.

SIP

SIP (Session Initiation Protocol) is an open and lightweight protocol.

Main functions of SIP:

1) It provides mechanisms for establishing calls b/w a caller and a callee over an IP network.
2) It allows the caller to notify the callee that it wants to start a call.
3) It allows the participants to agree on media encodings.
4) It also allows participants to end calls.
5) It provides mechanisms for the caller to determine the current IP address of the callee.
6) It provides mechanisms for call management, such as
   → adding new media streams during the call.
   → changing the encoding during the call
   → inviting new participants during the call,
   → call transfer and → call holding.

Setting up a Call to a Known IP Address

SIP call-establishment process is illustrated in Figure 5.7.

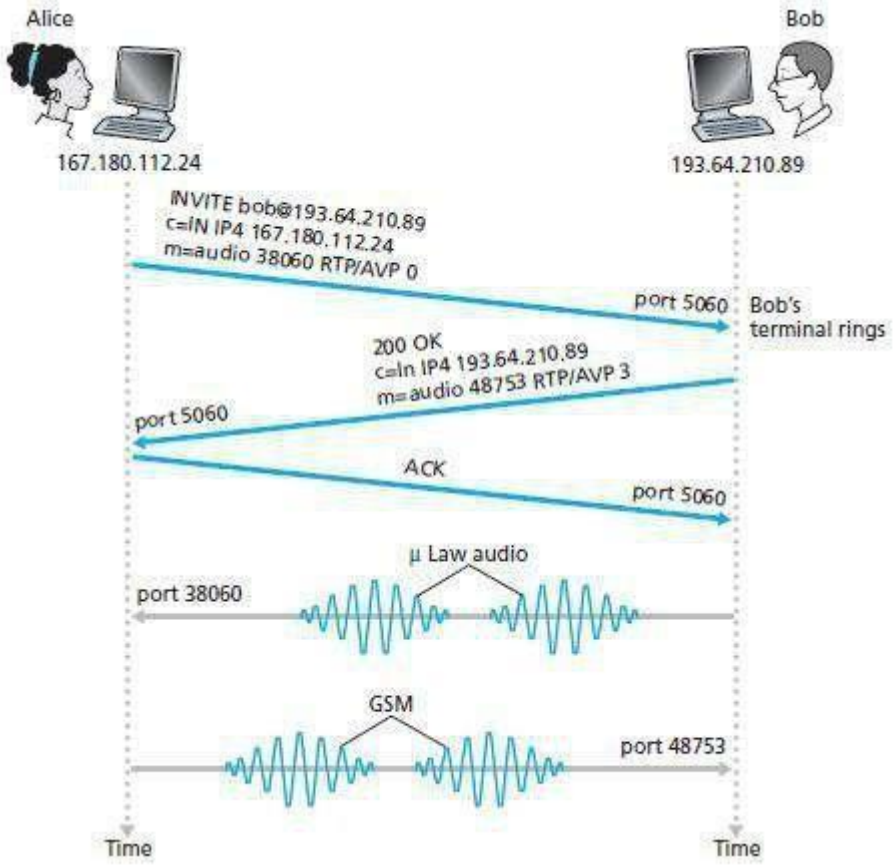


Fig 5.7 SIP call establishment when Alice knows Bob‟s IP address

Consider an example: Alice wants to call Bob. Alice‟s & Bob‟s PCs are both equipped with SIP-based software for making and receiving phone calls. The following events occur:

1) An SIP session begins when Alice sends Bob an INVITE message. This INVITE message is sent over UDP to the well-known port 5060 for SIP.
   The INVITE message includes
   i) An identifier for Bob (bob@193.64.210.89)
   ii) An indication of Alice‟s current IP address
   iii) An indication that Alice desires to receive audio, which is encoded in format AVP 0.
2) Then, Bob sends an SIP response message (which resembles an HTTP response message). The response message is sent over UDP to the wellknown port 5060 for SIP. The response message includes
   i) 200 OK
   ii) An indication of Bob‟s current IP address
   iii) An indication that Bob desires to receive audio, which is encoded in formaAVP 3.
3) Then, Alice sends Bob an SIP acknowledgment message.
4) Finally, Bob and Alice can talk.

Three key characteristics of SIP:

1) SIP is an out-of-band protocol. The SIP message & the media-data use different sockets for sending and receiving.
2) The SIP messages are ASCII-readable and resemble HTTP messages.
3) SIP requires all messages to be acknowledged, so it can run over UDP or TCP.