Module-1

Syllabus: Module-1

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.

(*Important topics are marked in 🟥)

Question-1: With neat diagram, explain the 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.

- clients do not directly communicate with each other.
- The server has a fixed, well-known address, called an IP address.
- Some of the better-known applications with a client-server architecture include the Web, FTP, Telnet, and e-mail.

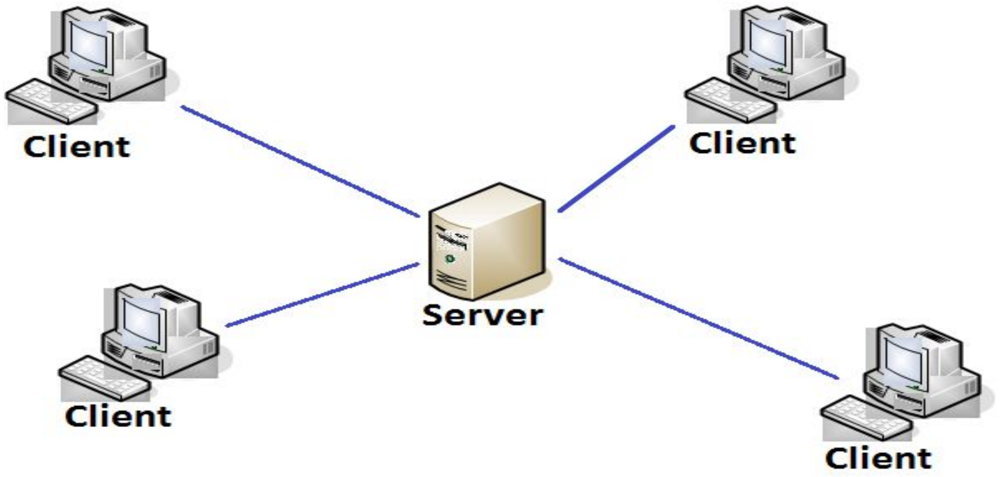


Fig. Client server Architecture

-

- 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.

Peer to Peer Applications:

- In a P2P architecture, there is minimal dependence on dedicated servers in data centers.
- In a P2P architecture, there is minimal dependence on dedicated servers in data centers.
- 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.

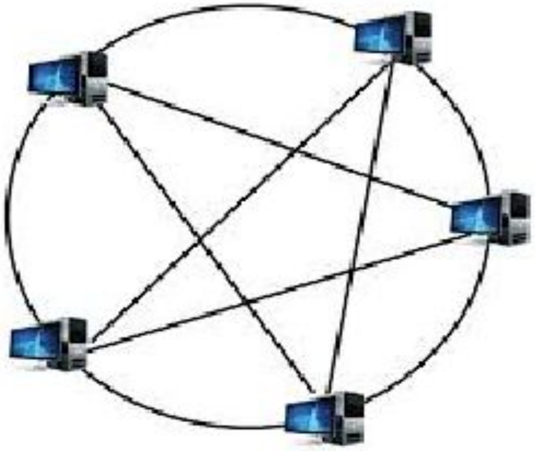


Fig. Peer to Peer Architecture

- 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).

- Peer -to peer applications are added with following 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

Question-2:List and explain different transport services offered to the application layer

Transport Services Available to Applications:

1. 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.

Question-3:Explain HTTP request and response messages. Give example for each

HTTP Request Message: HTTP Request message gives details about the service that should be given by the specific server.

Format of HTTP Request Message:

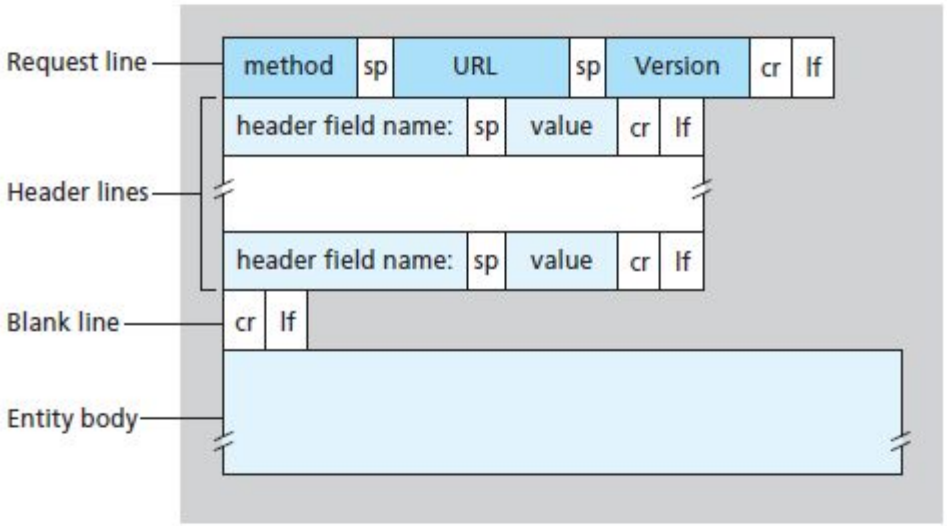


Fig. HTTP Request Message format

- The message consists of five lines, each followed by a carriage return and a line feed. The last line is followed by an additional carriage return and line feed.
- The first line of an HTTP request message is called the request line; the subsequent lines are called the header lines.
- The request line has three fields: the method field, the URL field, and the HTTP version field.
- The method field can take on several different values, including GET, POST, HEAD, PUT, and DELETE.
- The great majority of HTTP request messages use the GET method. The GET method is used when the browser requests an object, with the requested object identified in the URL field.
- An HTTP client often uses the POST method when the user fills out a form
- The HEAD method is similar to the GET method. When a server receives a request with the HEAD method, it responds with an HTTP message but

- The PUT method is also used by applications that need to upload objects to
- Web servers.
- The DELETE method allows a user, or an application, to delete an object on a Web server.
- Below we provide a typical HTTP request message:

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

- The message is written in ordinary ASCII text, so that your ordinary computer-literate human being can read it

- 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 Acceptlanguage: 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.
- The Accept-language: header is just one of many content negotiation headers available in HTTP.

Format of HTTP Request Message:

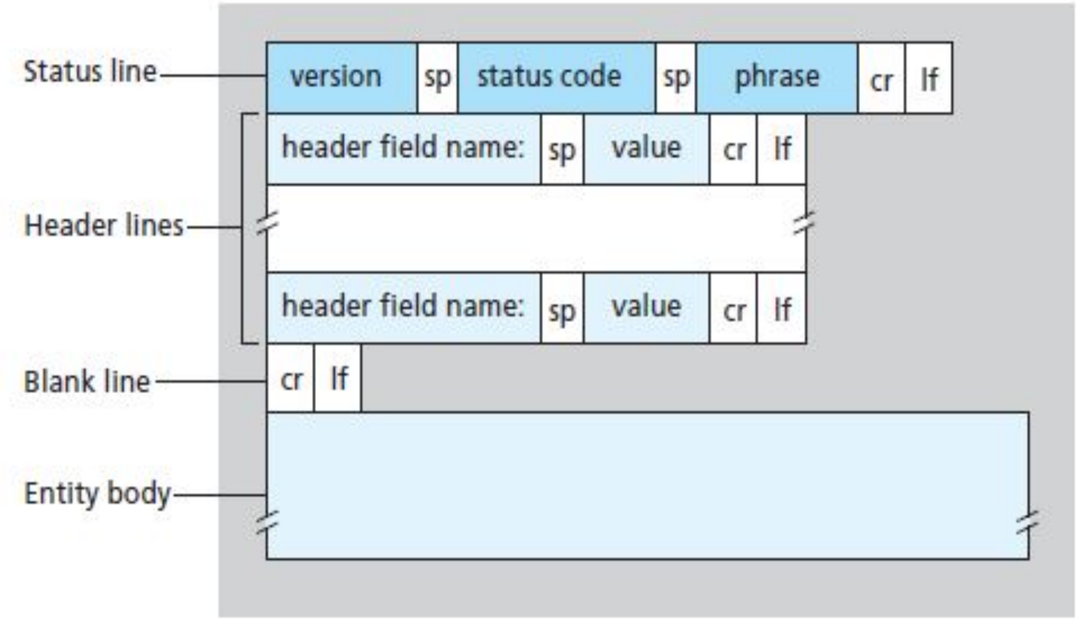


Fig. HTTP Response message

- Response message has three sections: an initial status line, six header lines, and then the entity body.
- The status line has three fields: the protocol version field, a status code, and a corresponding status message.
- 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 specific server
- 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.

Example: HTTP Response Message

```
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/1.1,200 OK: Request succeeded and the information is returned in the response.
- Header fields specify the details about the server Data and other details about the service.

Question 4: Write short note on i).Web Cache ii). Cookie

WEB Cache: also called a proxy server. It 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 fi

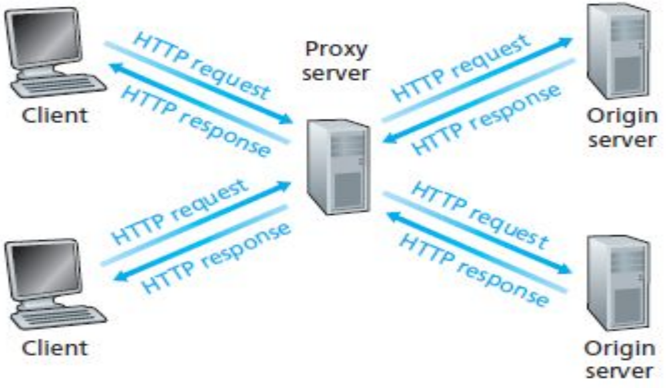


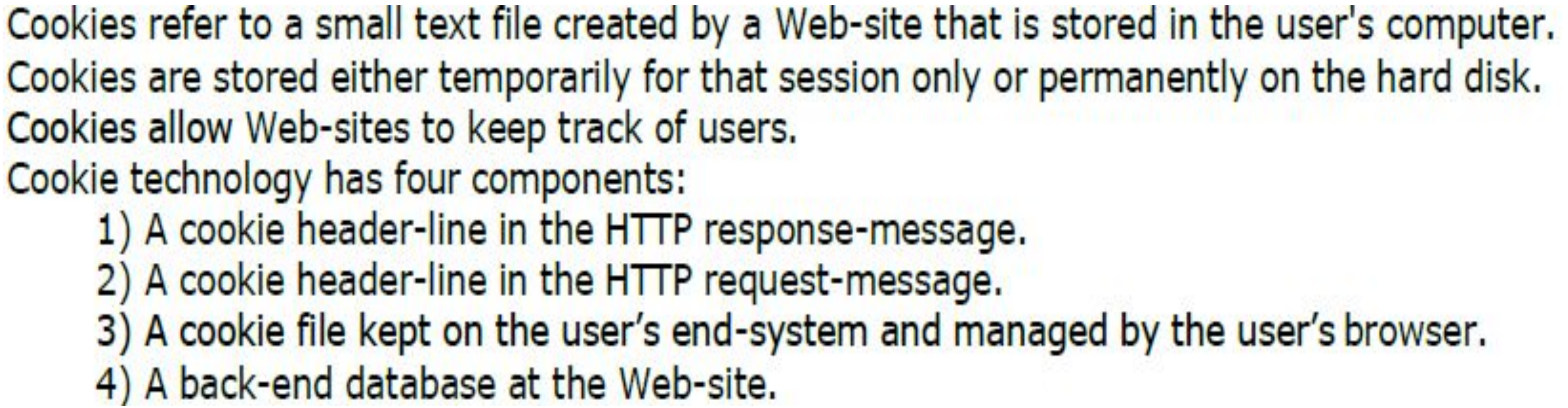
Fig. Web Cache

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

- The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.
- 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.
- 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.
- After receiving this request, the origin server sends the object within an HTTP response to the Web cache.

- 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

Cookie:



Cookies refer to a small text file created by a Web-site that is stored in the user's computer.

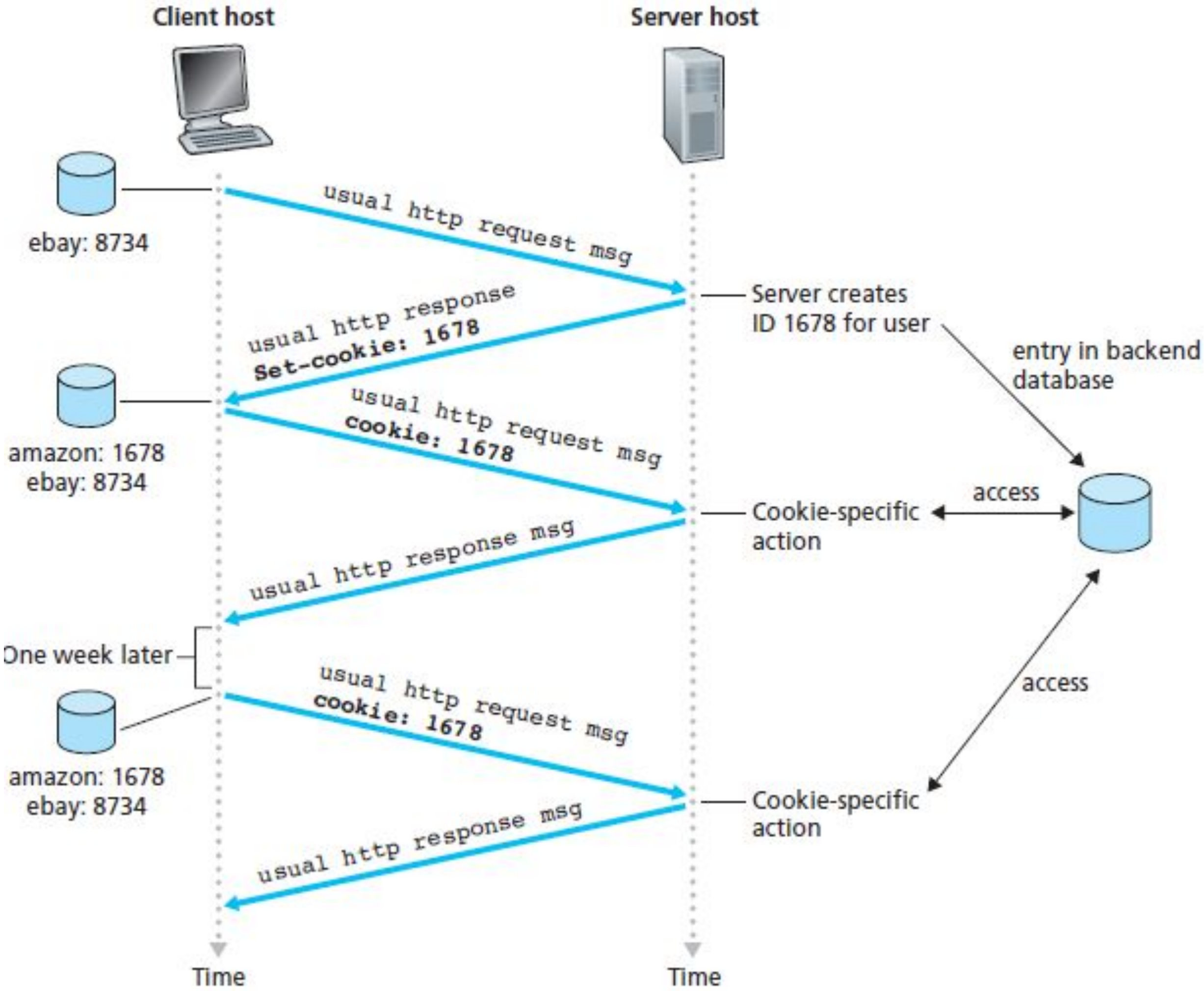
Cookies are stored either temporarily for that session only or permanently on the hard disk.

Cookies allow Web-sites to keep track of users.

Cookie technology has four components:

1) A cookie header-line in the HTTP response-message.
2) A cookie header-line in the HTTP request-message.
3) A cookie file kept on the user's end-system and managed by the user's browser.
4) A back-end database at the Web-site.

Example: 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.

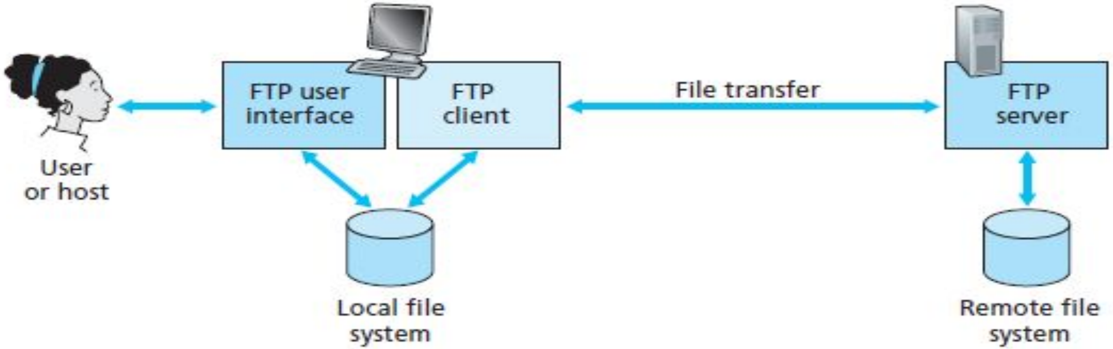

Client host
Server host
ebay: 8734
usual http request msg
Server creates
ID 1678 for user
usual http response
Set-cookie: 1678
entry in backend
database
amazon: 1678
ebay: 8734
usual http request msg
cookie: 1678
access
Cookie-specific
action
usual http response msg
One week later
usual http request msg
cookie: 1678
access
amazon: 1678
ebay: 8734
Cookie-specific
action
usual http response msg
Time
Time

- 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

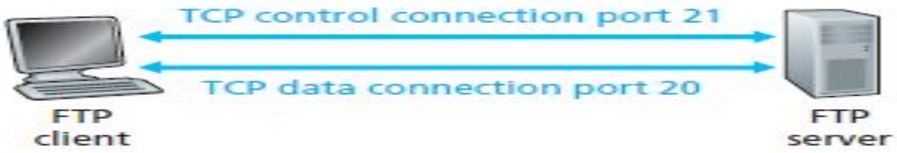
. Question-5: Write a short note on SMTP, FTP

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 verse)



- 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 co



- 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

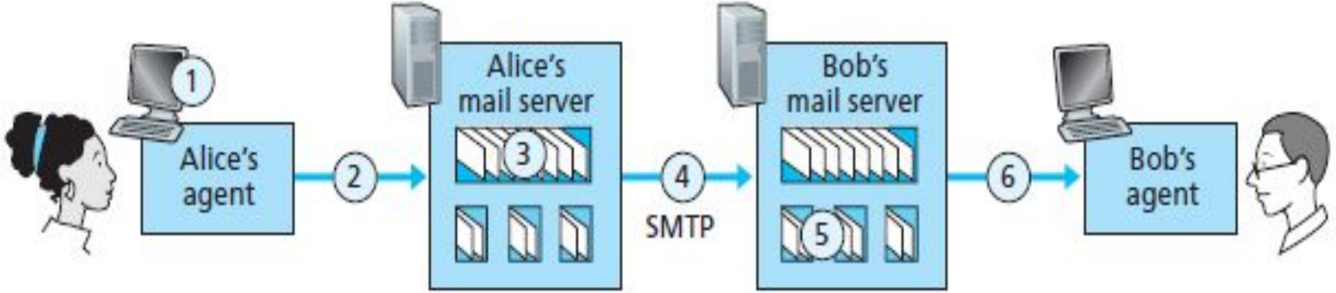
SMTP:

- 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

- 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.

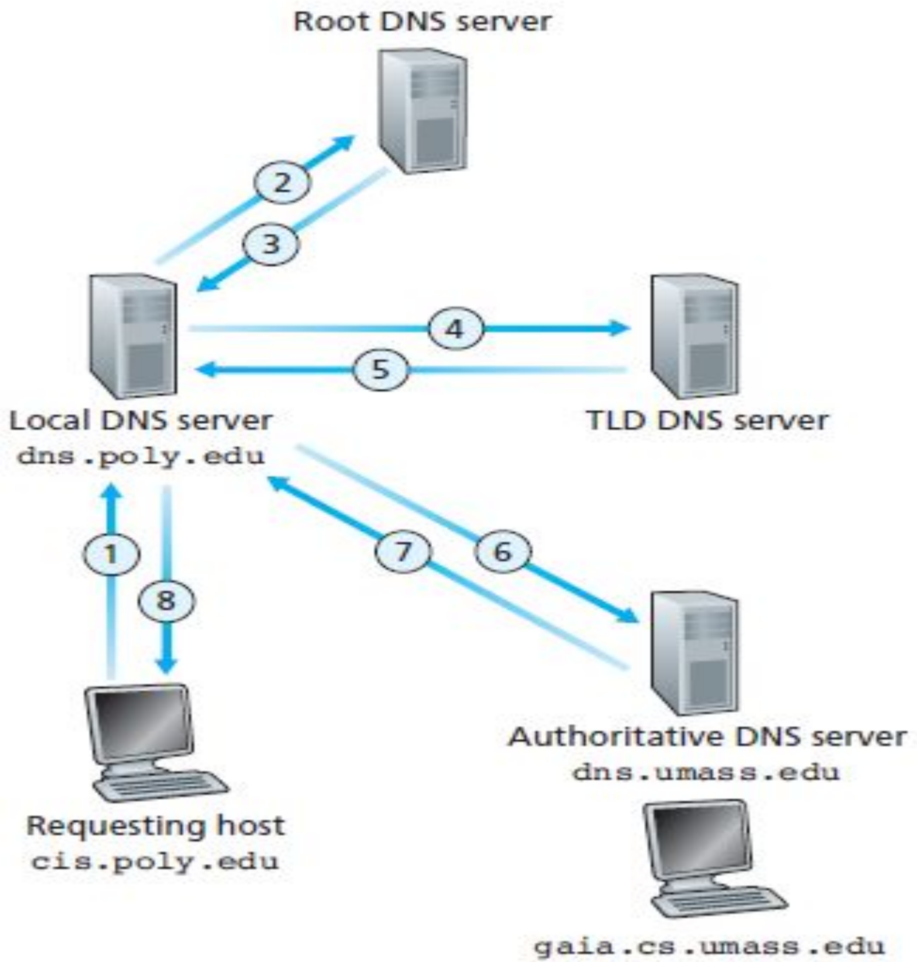
Question 5: Explain iterative and recursive query processing services offered by DNS

- The DNS is a distributed database implemented in a hierarchy of DNS servers, and an application-layer protocol that allows hosts to query the distributed database.
- Uses either iterative or recursive approach for query processing.

Iterative approach:

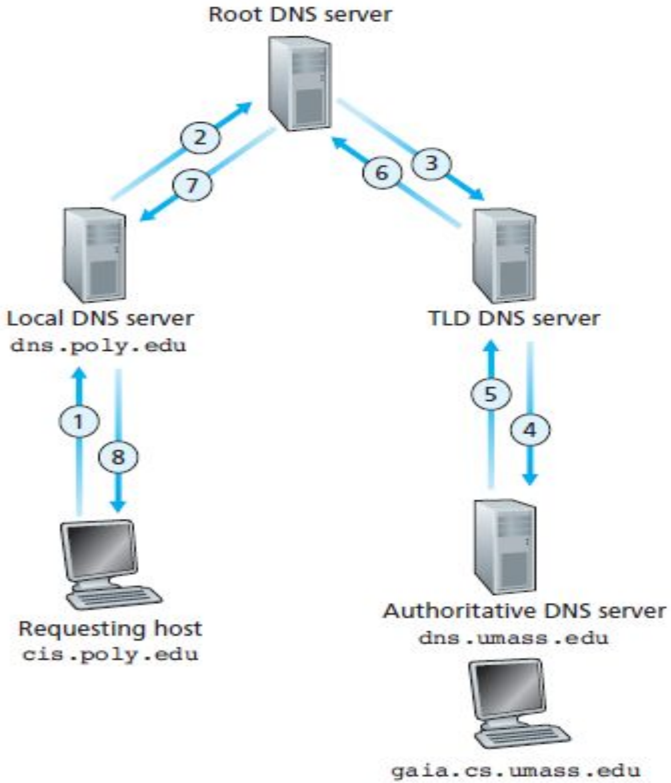
The query message contains the hostname to be translated,namely, gaia.cs.umass.edu.

- The local DNS server forwards the query message to a root DNS server
- The root DNS server takes note of the edu suffix and returns to the local DNS server a list of IP addresses for TLD servers responsible for edu.
- The local DNS server then resends the query message to one of these TLD servers.
- The TLD server takes note of the umass.edu suffix and responds with the IP


Root DNS server
2
3
Local DNS server
dns.poly.edu
4
5
TLD DNS server
1
8
7
6
Requesting host
cis.poly.edu
Authoritative DNS server
dns.umass.edu
gaia.cs.umass.edu

- Finally, the local DNS server resends the query message directly to dns.umass.edu, which responds with the IP address of gaia.cs.umass.edu.

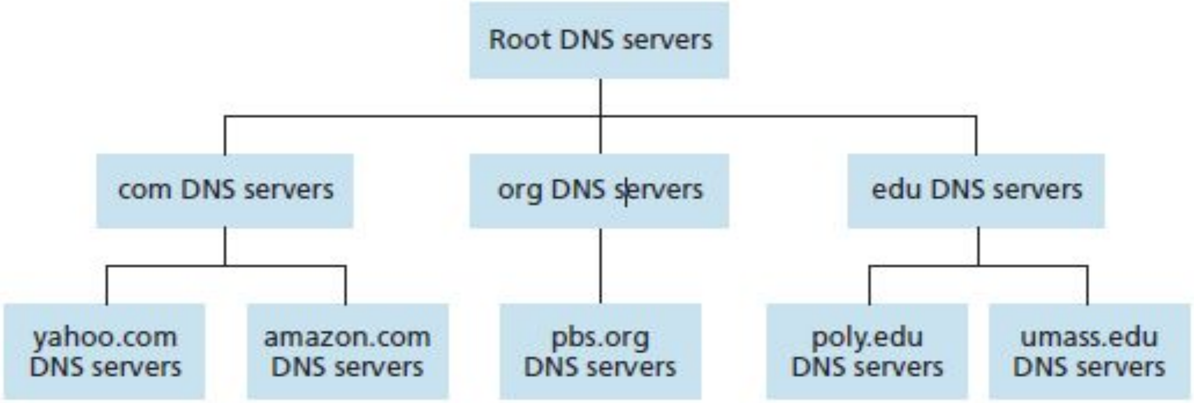
Recursive Query service:



- Here DNS query will be sent to Local DNS server, then to root server.
- Root server sends the query to TLD DNS server
- TLD DNS server sends the query to authoritative DNS server
- Authoritative DNS server sends the IP address of host to local DNS server through a chain communication with TLD server, root server and the client’s Local DNS server which sends it to the host.

Question: Write short note on DNS system Hierarchy

- 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

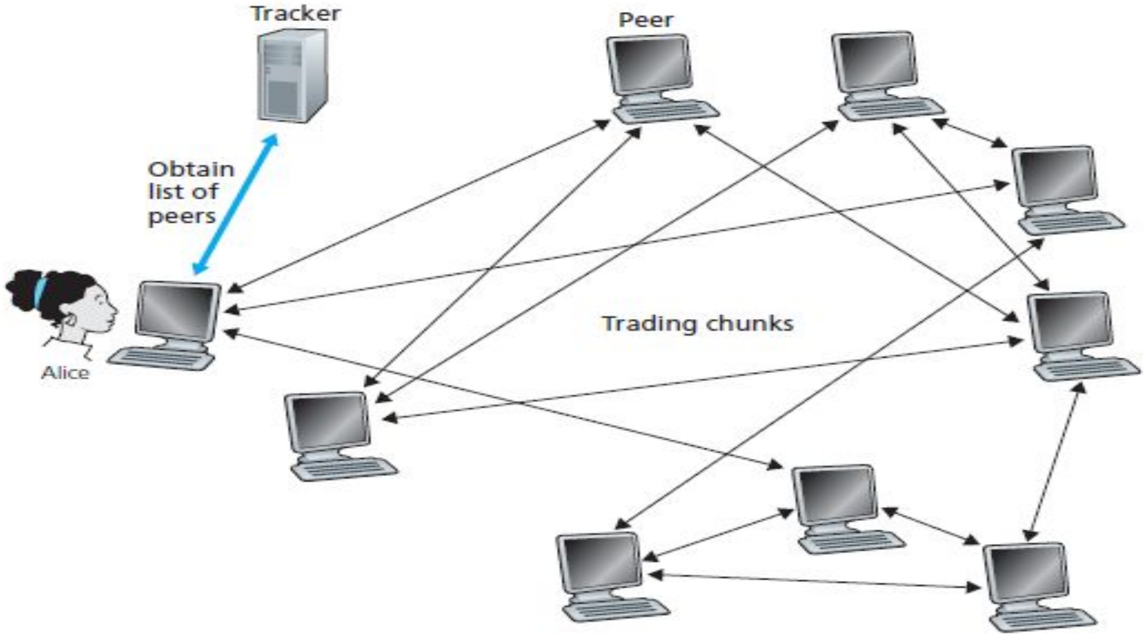
- There is another important type of DNS server called the local DNS server. A local DNS server does not strictly belong to the hierarchy of servers but is nevertheless central to the DNS architecture. Each ISP—such as a university, an academic department, an employee’s company, or a residential ISP—has a local DNS server.

Question:Explain the working of Bit Torrent application

- 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 requests (again over the TCP connections) for chunks

- 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 chur
-



- 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

Question:Write a short note on Clever trading algorithm

- The peer nodes in Bittorrent, uses a technique called rarest first for geeting the rarest chunks from neighbors. The idea is to determine, from among the chunks the node does not have, the chunks that are the rarest among her neighbors (that is, the chunks that have the fewest repeated copies among her neighbors) and then request those rarest chunks first. In this manner, the rarest chunks get more quickly redistributed, aiming to (roughly) equalize the numbers of copies of each chunk in the torrent.
- To determine which requests the node responds to, BitTorrent uses a clever trading algorithm. The basic idea is that node gives priority to the neighbors that are currently supplying her data at the highest rate.
- Specifically, for each of her neighbors,the node continually measures the rate at which it receives bits and determines the four peers that are feeding her bits at the highest rate.

- It then reciprocates by sending chunks to these same four peers. Every 10 seconds, it 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, it also picks one additional neighbor at random and sends it chunks.
- Node will randomly choose a new trading partner and initiate trading with that partner.

Question:Write a short note on DHT

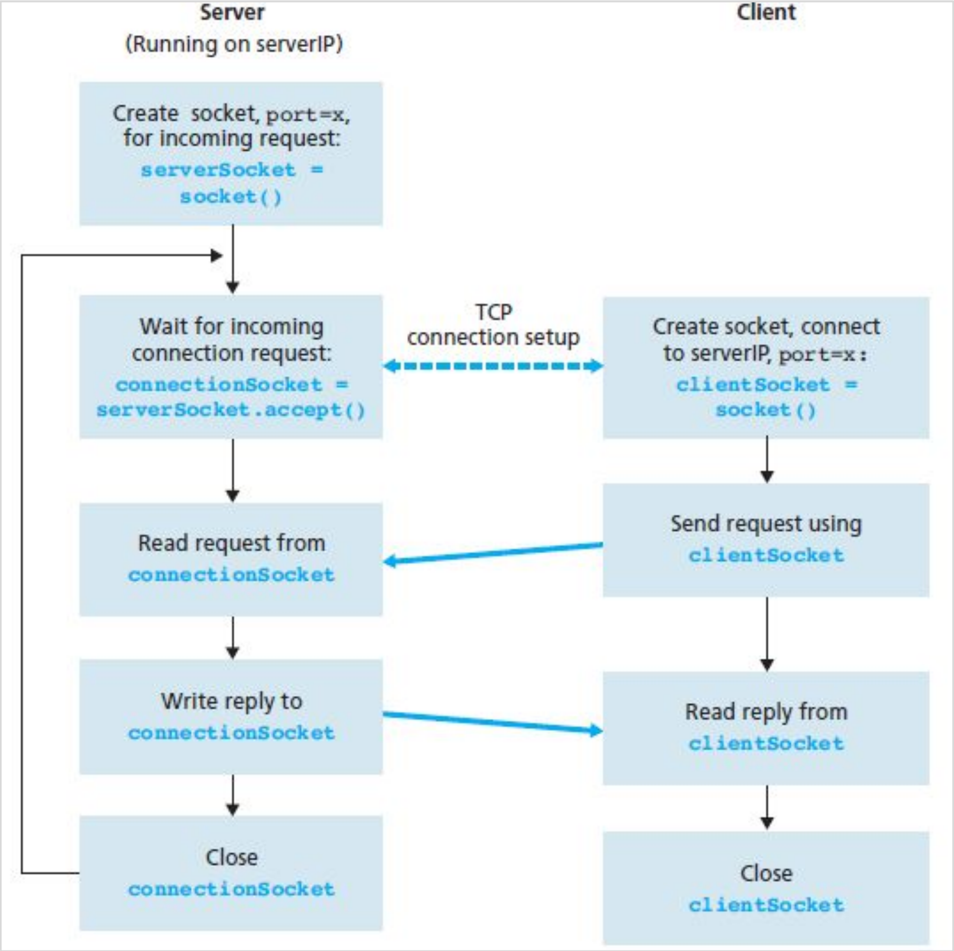
Distributed Hash Tables (DHTs):

- 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, 2n-1] for some fixed n.
- This also require each key to be an integer in the same range.
- To create integers out of such keys, we will use a hash function that

Question:Write and explain the algorithm for TCP enabled socket communication

- 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


Server
(Running on serverIP)
Client
Create socket, port=x, for incoming request:
serverSocket = socket()
Wait for incoming connection request:
connectionSocket = serverSocket.accept()
TCP connection setup
Create socket, connect to serverIP, port=x:
clientSocket = socket()
Read request from connectionSocket
Send request using clientSocket
Write reply to connectionSocket
Read reply from clientSocket
Close connectionSocket
Close clientSocket

TCP Client Program

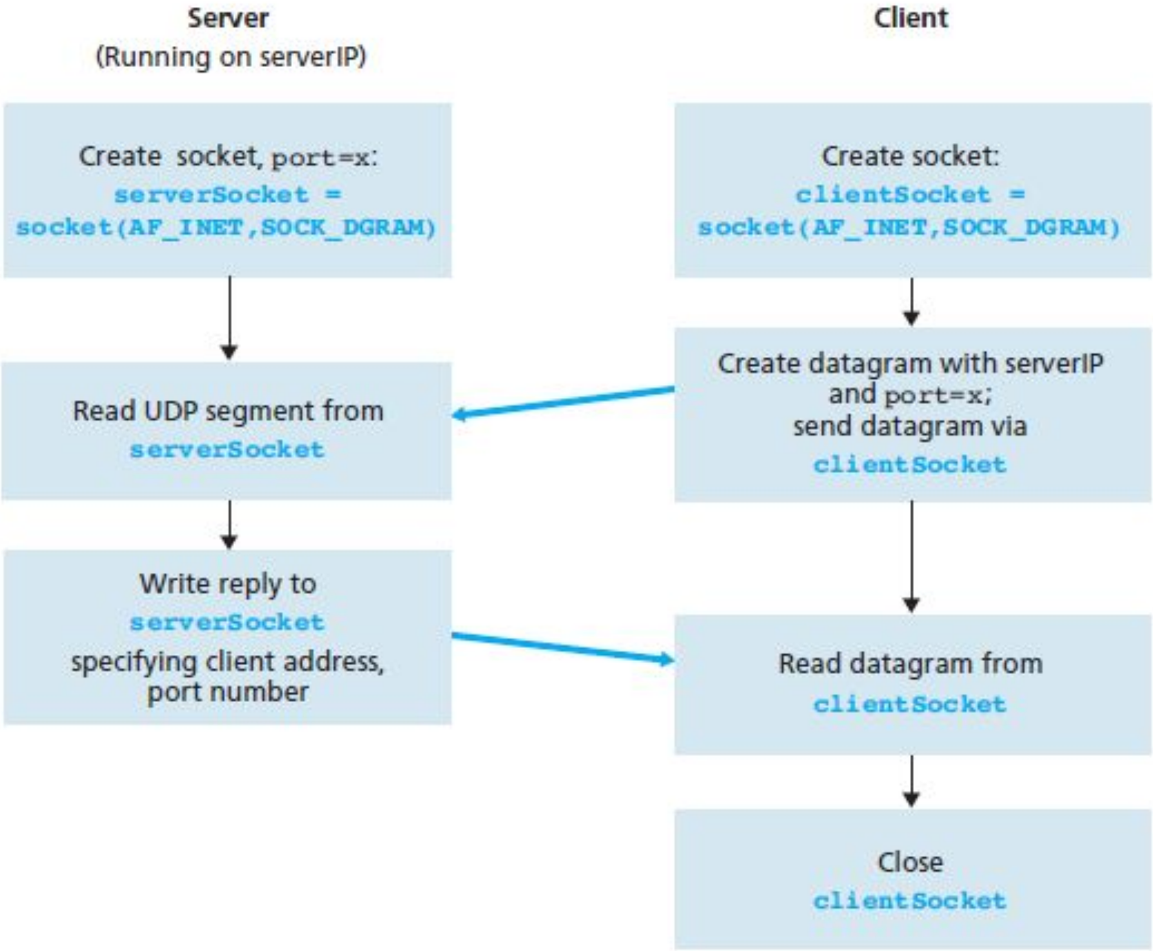
```
from socket import *
serverName = ’servername’
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input(‘Input lowercase sentence:’)
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print ‘From Server:’,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

TCP Server Program:

```
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)
```

Question: UDP Sockets

- 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.



Server

(Running on serverIP)

Create socket, `port=x`:

`serverSocket = socket(AF_INET,SOCK_DGRAM)`

Read UDP segment from `serverSocket`

Write reply to `serverSocket` specifying client address, port number

Client

Create socket:

`clientSocket = socket(AF_INET,SOCK_DGRAM)`

Create datagram with serverIP and `port=x`; send datagram via `clientSocket`

Read datagram from `clientSocket`

Close `clientSocket`

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()
```

UDP Server:

```
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)
```

Module-2

Syllabus: Module-2

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.

Question-1: Explain different services offered by the transport layer in TCP/IP protocol stack

Refer the answer from the module1.

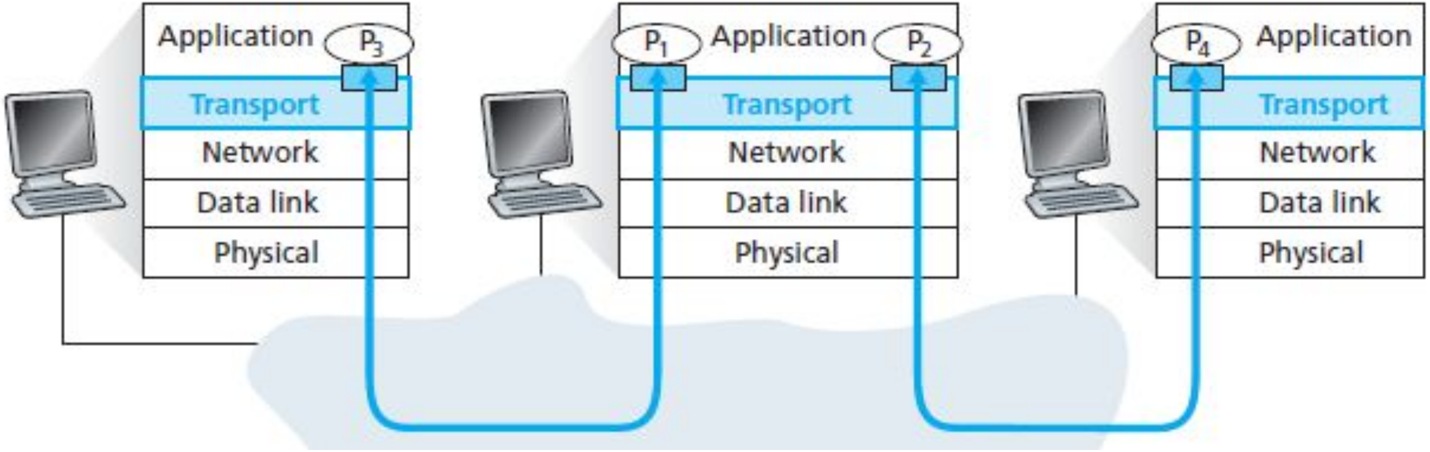
Include following points:

1. TCP and UDP based transport services.
2. Segmentation and reassembly
3. Flow control, congestion control and error control
4. Inter process communication using Sockets

2.Explain Multiplexing and Demultiplexing operation at transport Layer

Multiplexing:

- Transport layer applies multiplexing at the sender end.
- 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.



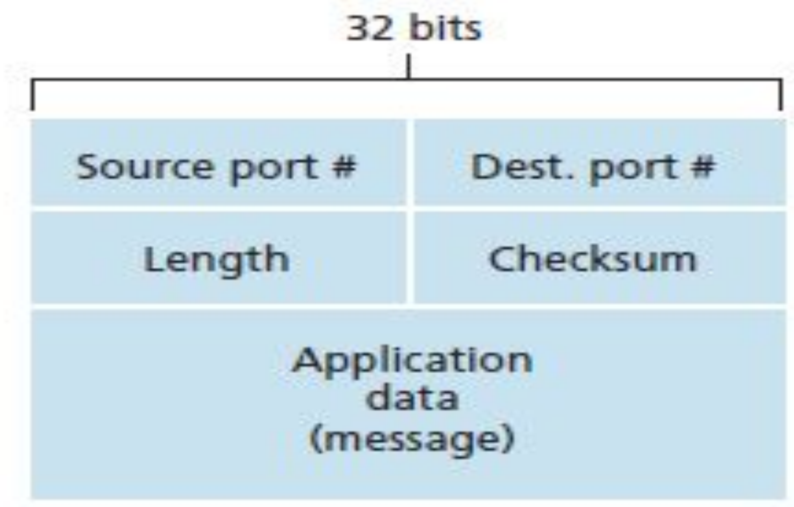
- Transport-layer multiplexing requires that sockets have unique identifiers, and that each segment have special fields that indicate the socket to which the segment is to be delivered.
- These special fields are the source port number field and the destination port number field.
- Multiplexing also enables the transport layer to receive the data from different application protocols and multiplex them and send it over network which uses only single protocol(IP) at network layer.

Demultiplexing:

- At the receiving end, the transport layer examines the received segment headers for verifying the source port number and the destination port number and to direct the data to the appropriate application.
- The process of gathering the segments (single protocol)and connecting the data to different applications at the higher layer(multiple protocols) is called as Demultiplexing.

3. Explain UDP segment structure

UDP segment structure:



32 bits

Source port #	Dest. port #
Length	Checksum
Application data (message)	

- 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.
- Supports UDP transmission at transport layer for quick transmission services.

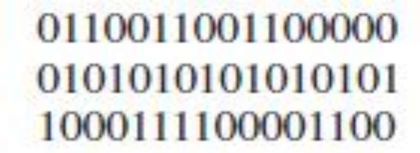
4. Write a short note on UDP checksum

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

- UDP at the sender side performs the 1s complement of the sum of all the 16-bit words in the segment, with any overflow encountered during the sum being wrapped around.
- This result is put in the checksum field of the UDP segment.
- Then the segment will be transmitted by the sender.
- At the receiver, all four 16-bit words are added, including the checksum.
- If no errors are introduced into the packet, then clearly the 1’s complement of the sum at the receiver will be having all the bits as zero.

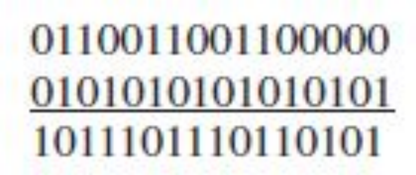
Example:Suppose that we have the following three 16-bit words

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



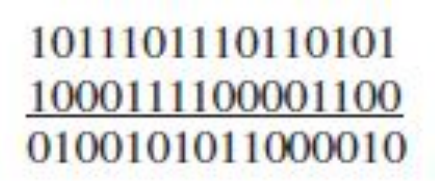
```
0110011001100000
0101010101010101
1000111100001100
```

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



```
0110011001100000
0101010101010101
1011101110110101
```

When this sum is added with the third word



```
1011101110110101
1000111100001100
0100101011000010
```

Step 2: Take 1s complement of the result.

- The 1's 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.

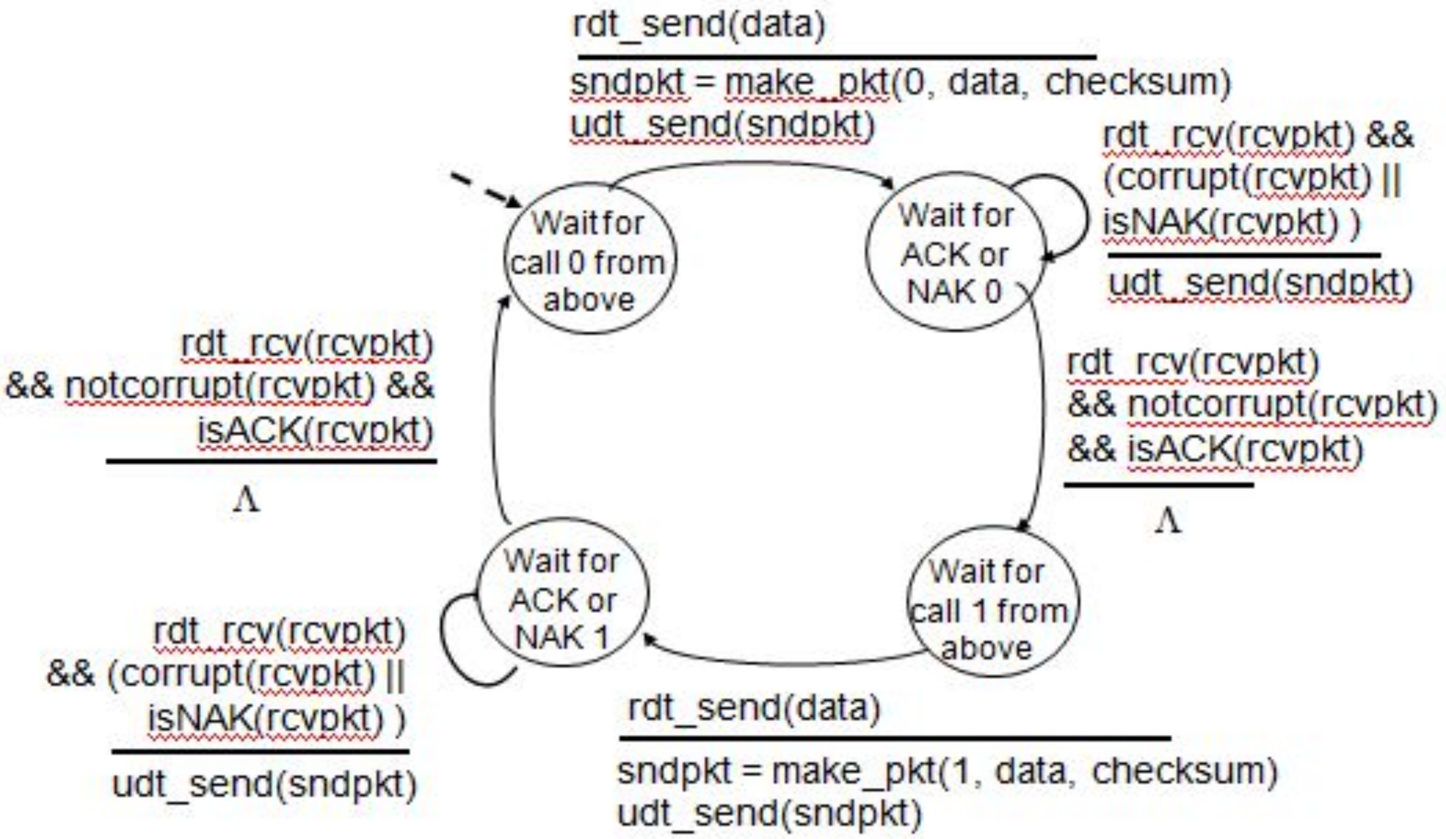
Step 3: Data along with checksum is transmitted to receiver.

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.

5. With FSM, explain the protocol development phase in rdt 2.1

- Rdt 2.1 is designed to overcome the problem of identifying the arrival of duplicate packet at the receiver end.
- Rdt 2.1 has added the mechanism to handle the arrival of duplicate packets at the receiver.
- Handling of duplicate packets is carried out by enabling the following actions
  - sender retransmits current pkt if ACK/NAK corrupted
  - sender adds *sequence number* to each pkt
  - receiver discards (doesn't deliver up) duplicate pkt

rdt2.1 sender: handling garbled ACK/NAKs



rdt_send(data)

sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) ||
isNAK(rcvpkt))

udt_send(sndpkt)

rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt)

Λ

rdt_send(data)

sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt)
&& (corrupt(rcvpkt) ||
isNAK(rcvpkt))

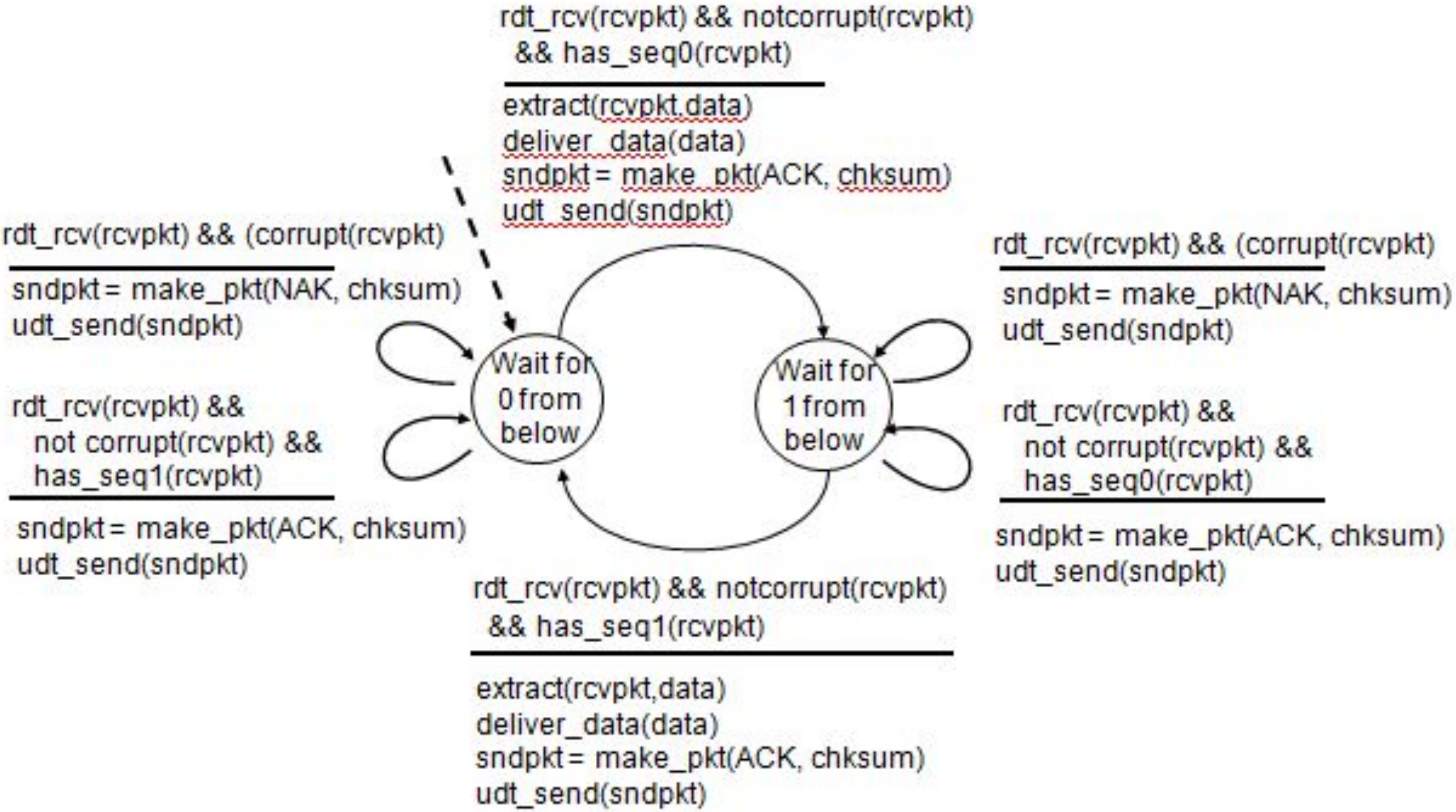
udt_send(sndpkt)

rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt) &&
isACK(rcvpkt)

Λ

- At the sender end, each packet before transmission, added with a sequence number ‘0’ or ‘1’ (the protocol design is for Stop and Wait).
- Checksum is added to help the receiver in detecting the transmission errors.
- Sender checks whether the received ACK/NAK is corrupted. If its got corrupted, then sends the saved copy of the current packet. It' s the receiver’s job, whether to receive the arrived packet or to discard it, after verifying the sequence number of the packet it has received.

rdt2.1: receiver, handling garbled ACK/NAKs



rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq0(rcvpkt)

extract(rcvpkt.data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) && (corrupt(rcvpkt)

sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
not corrupt(rcvpkt) &&
has_seq1(rcvpkt)

sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) && (corrupt(rcvpkt)

sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
not corrupt(rcvpkt) &&
has_seq0(rcvpkt)

sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)

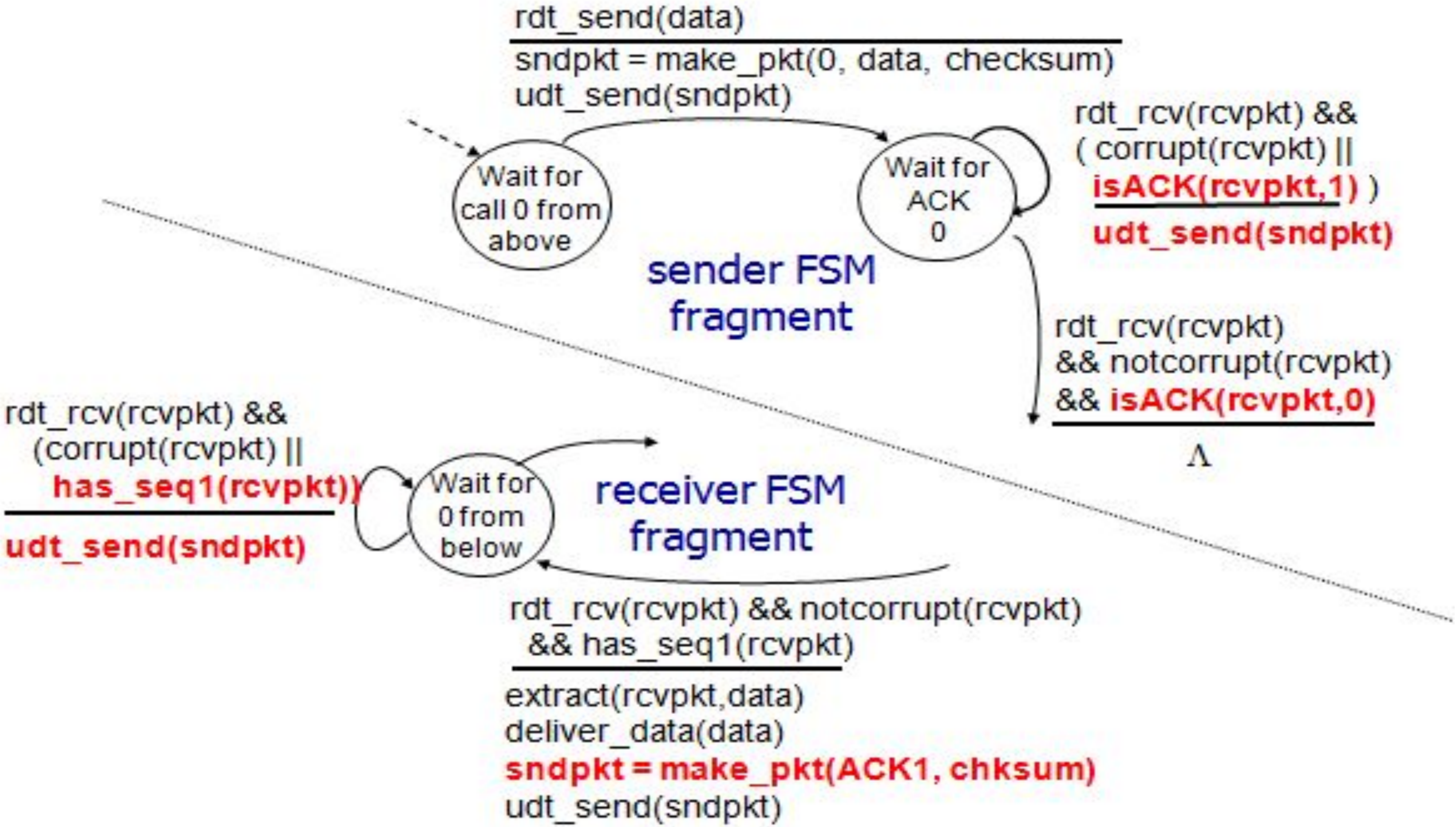
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

- The receiver in rdt 2.1 must check if received packet is duplicate or new one.
    - state indicates whether 0 or 1 is expected pkt seq number
- note: receiver can *not* know if its last ACK/NAK received OK at sender as sender will decide which packet to send next based on the correctness of the received reply(ACK/NAK)

6. With FSM, explain the protocol development phase in rdt 2.2

- **rdt2.2: is a NAK-free protocol.**
- Designed to implement same functionality as rdt2.1(Handling the duplicate packets), using ACKs only.
- instead of NAK, receiver sends ACK for last packet received OK
  - receiver must *explicitly* include seq. number of packet being acknowledged.
-

rdt2.2: sender, receiver fragments



rdt_send(data)

---

sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) ||
isACK(rcvpkt,1))

---

udt_send(sndpkt)

sender FSM fragment

rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& **isACK(rcvpkt,0)**

Λ

rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) ||
has_seq1(rcvpkt))

---

udt_send(sndpkt)

receiver FSM fragment

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)

---

extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK1, chksum)
udt_send(sndpkt)

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

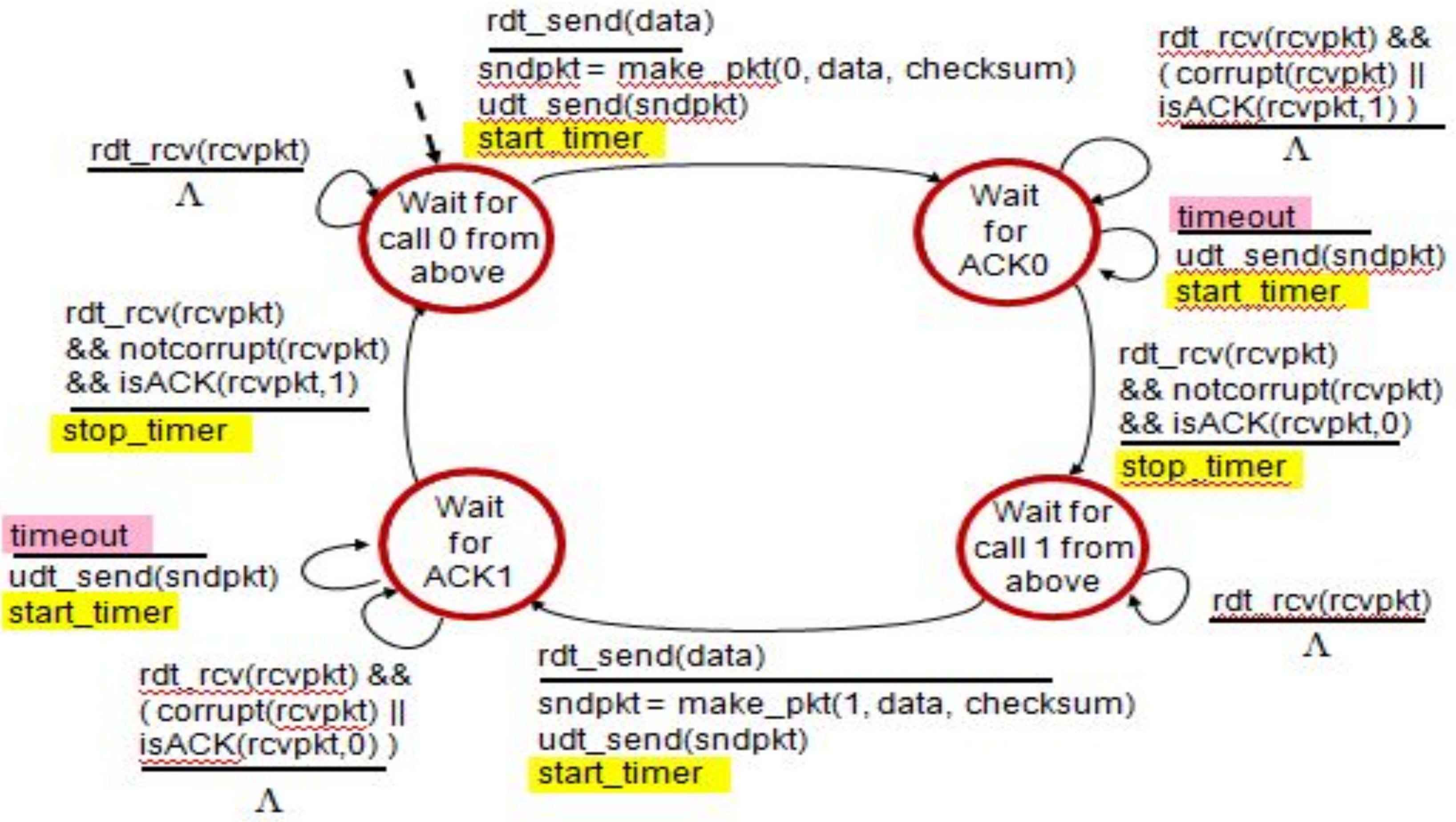
7. With FSM, explain the protocol development phase in rdt 3.0

Rdt 3.0 is designed to achieve reliable data transfer over a lossy channel with bit errors

Approach: sender waits “reasonable” amount of time for ACK, retransmits if no ACK received in this time. If pkt (or ACK) just delayed (not lost):

- retransmission will be duplicate, but seq numbers already handles this.
- receiver must specify seq number of packet being Acknowledged.
- uses countdown timer to interrupt after “reasonable” amount of time

Rdt 3.0 Sender



rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
start_timer

rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
Λ

timeout
udt_send(sndpkt)
start_timer

rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt,0)
stop_timer

rdt_rcv(rcvpkt)
Λ

rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)
start_timer

rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0))
Λ

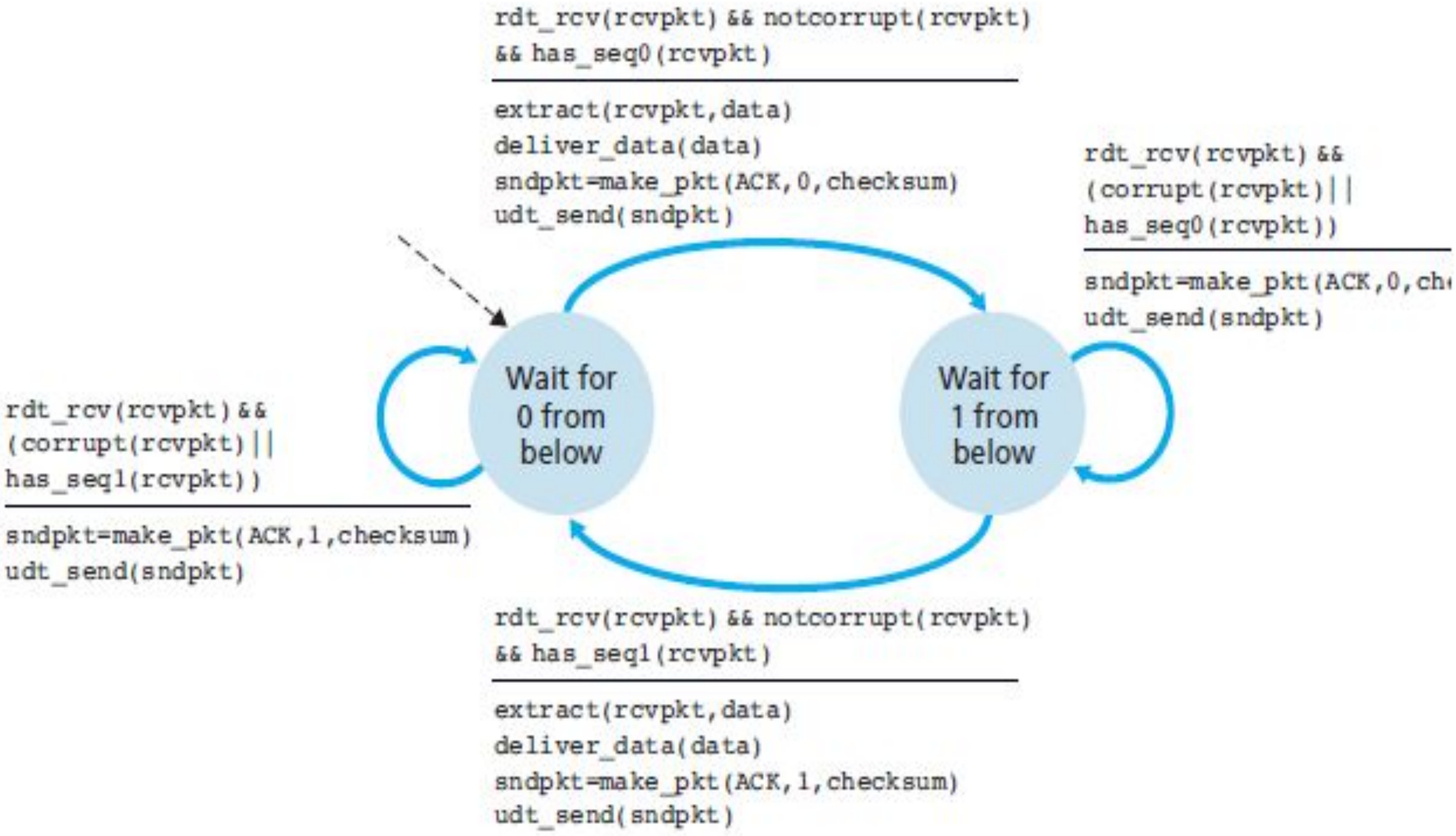
timeout
udt_send(sndpkt)
start_timer

rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt,1)
stop_timer

rdt_rcv(rcvpkt)
Λ

- 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.
- After waiting for the set amount of time(RTT-Timeout), If an ACK is not received at the sender, then the same packet is retransmitted. Note that if a packet experiences a particularly large delay, the sender may retransmit the packet even though neither the data packet nor its ACK have been lost.

Rdt 3.0 Receiver



```
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq0(rcvpkt)
_____________________________________
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,0,checksum)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt)||
has_seq0(rcvpkt))
_____________________________________
sndpkt=make_pkt(ACK,0,ch
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt)&&
(corrupt(rcvpkt)||
has_seq1(rcvpkt))
_____________________________________
sndpkt=make_pkt(ACK,1,checksum)
udt_send(sndpkt)
```

```
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)
_____________________________________
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,1,checksum)
udt_send(sndpkt)
```

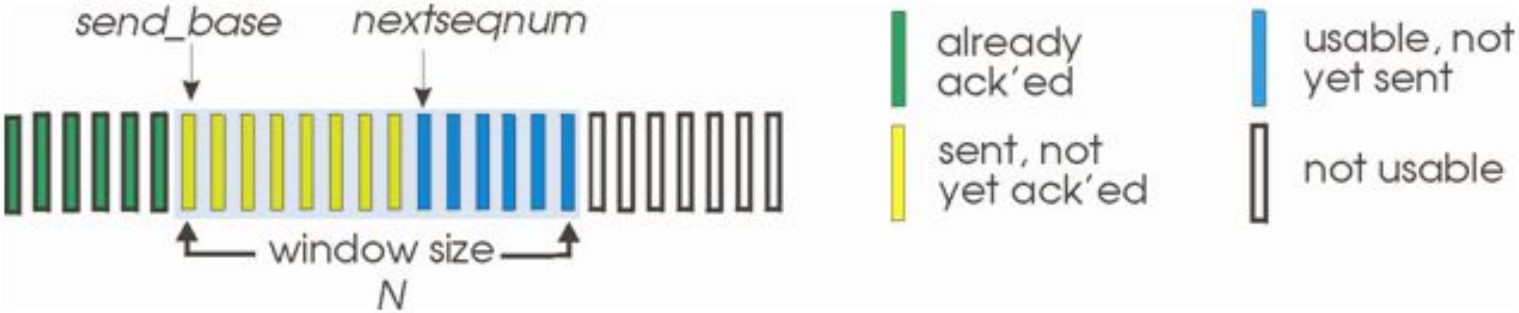
- When the receiver generates an ACK, it will copy the sequence number of the data packet being ACK'ed into this acknowledgement field.
- By examining the contents of the acknowledgment field, the sender can determine the sequence number of the packet being positively acknowledged.

8. Explain the working of Go Back-N protocol

- GO-Back-N is a pipelined reliable data transmission protocol.
- Works using Sliding-Window approach. Uses cumulative acknowledgements.
- In Go back N, sender window size is N and receiver window size is always 1.
- 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.
- Sender decides how many packets to send, based on the allowed window size.

Go-Back-N: sender

- sender: "window" of up to N, consecutive transmitted but unACKed pkts
  - k-bit seq # in pkt header

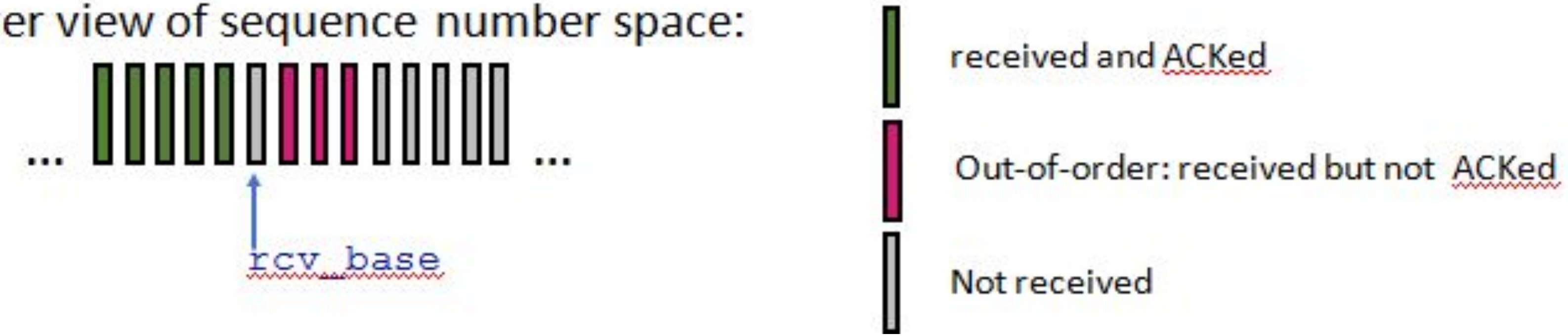


- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window

Go-Back-N: receiver

- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don't buffer) or buffer: an implementation decision
  - re-ACK pkt with highest in-order seq #

Receiver view of sequence number space:



… …

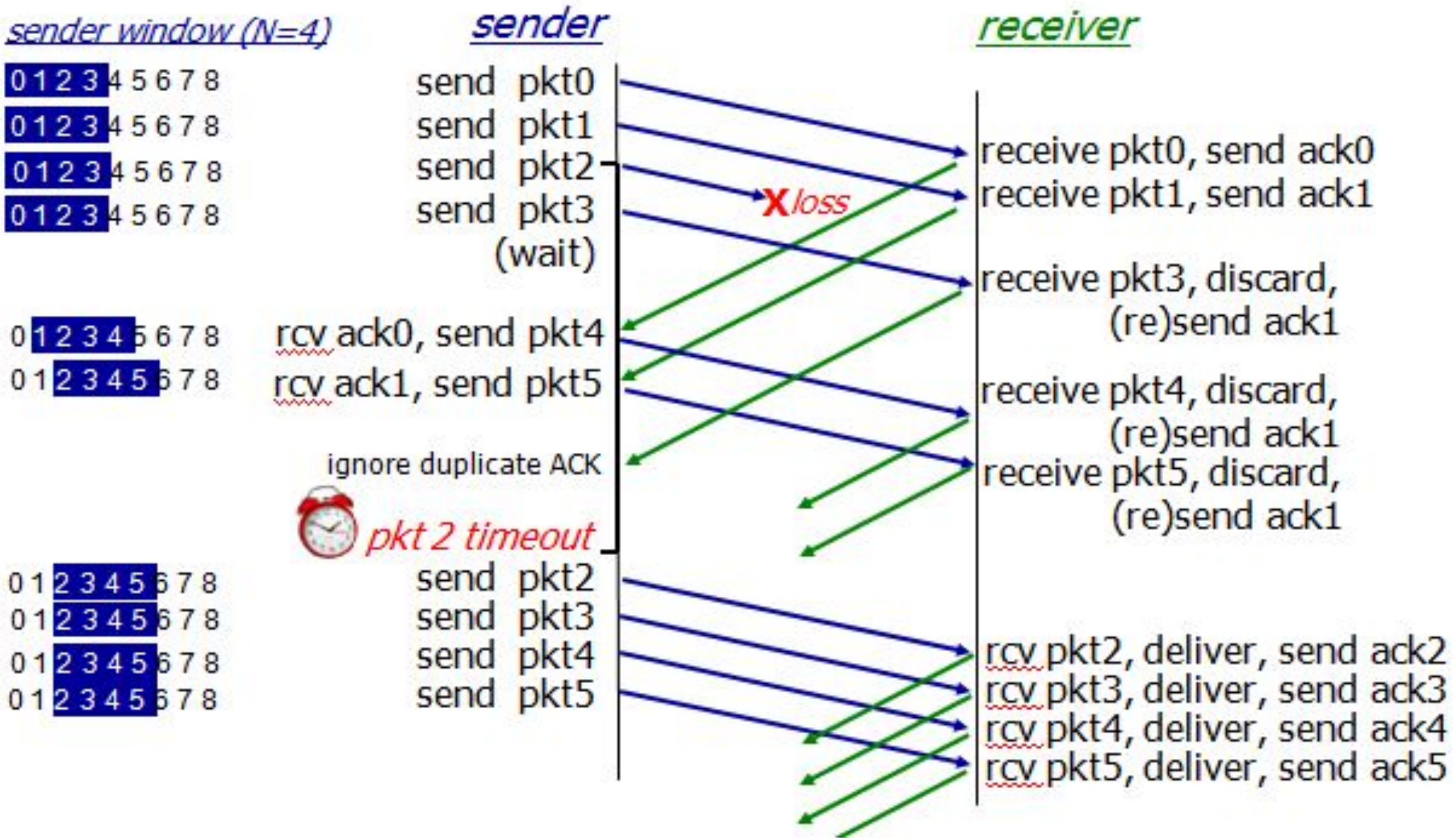
`rcv_base`

🟩 received and ACKed

🟥 Out-of-order: received but not ACKed

▨ Not received

Go-Back-N in action



sender window (N=4)

sender

receiver

0 1 2 3 4 5 6 7 8 send pkt0

0 1 2 3 4 5 6 7 8 send pkt1

0 1 2 3 4 5 6 7 8 send pkt2

0 1 2 3 4 5 6 7 8 send pkt3

(wait)

X *loss*

receive pkt0, send ack0

receive pkt1, send ack1

receive pkt3, discard,
(re)send ack1

0 1 2 3 4 5 6 7 8 rcv ack0, send pkt4

0 1 2 3 4 5 6 7 8 rcv ack1, send pkt5

receive pkt4, discard,
(re)send ack1

ignore duplicate ACK

receive pkt5, discard,
(re)send ack1

pkt 2 timeout

0 1 2 3 4 5 6 7 8 send pkt2

0 1 2 3 4 5 6 7 8 send pkt3

0 1 2 3 4 5 6 7 8 send pkt4

0 1 2 3 4 5 6 7 8 send pkt5

rcv pkt2, deliver, send ack2

rcv pkt3, deliver, send ack3

rcv pkt4, deliver, send ack4

rcv pkt5, deliver, send ack5

Add explanation based on the fig. Given in the previous slide

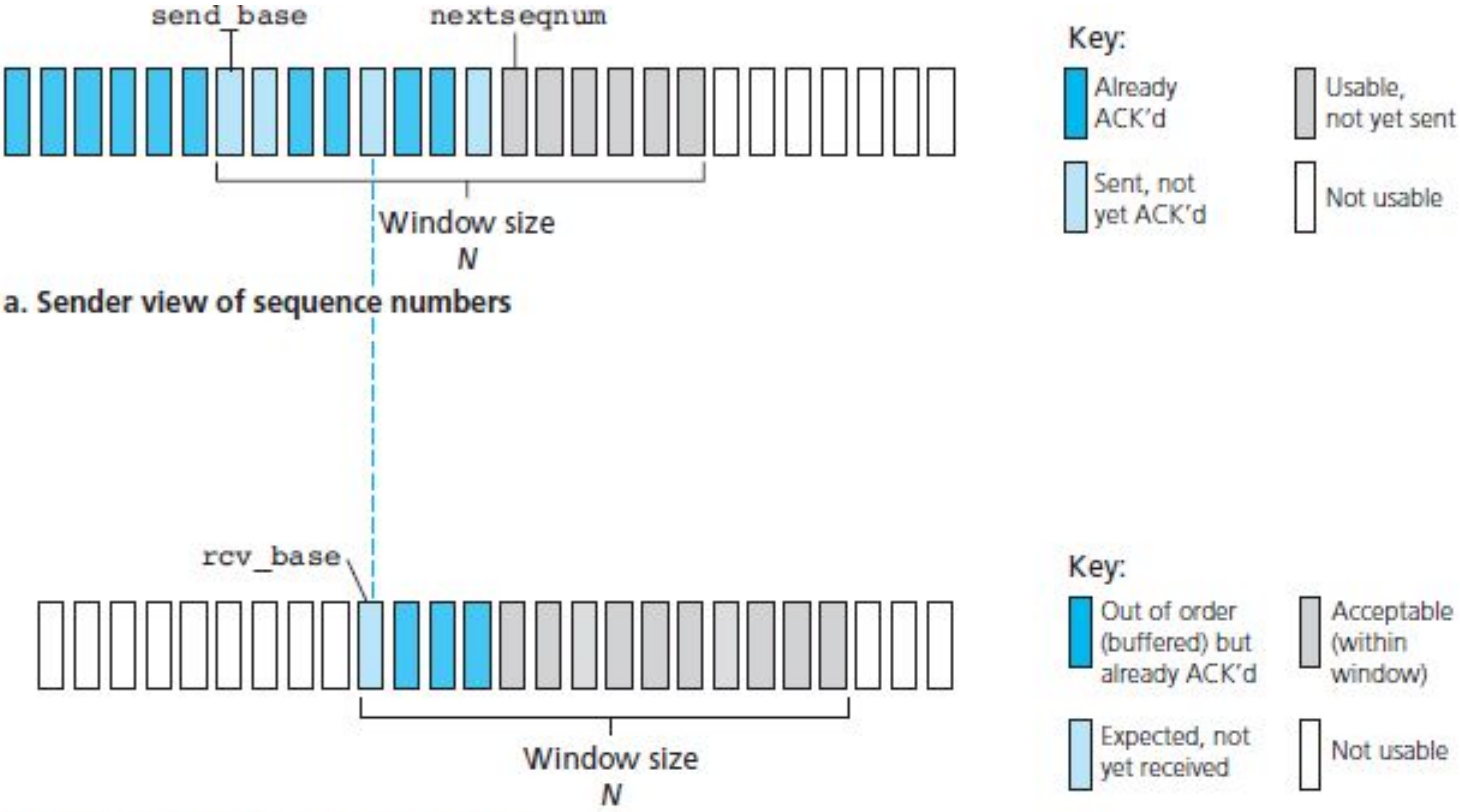
Or

Even you can write the answer by referring the FSM given in the text book.

9. Explain the working of Selective-Repeat Protocol

Selective-Repeat is a pipelined reliable data transmission protocol.

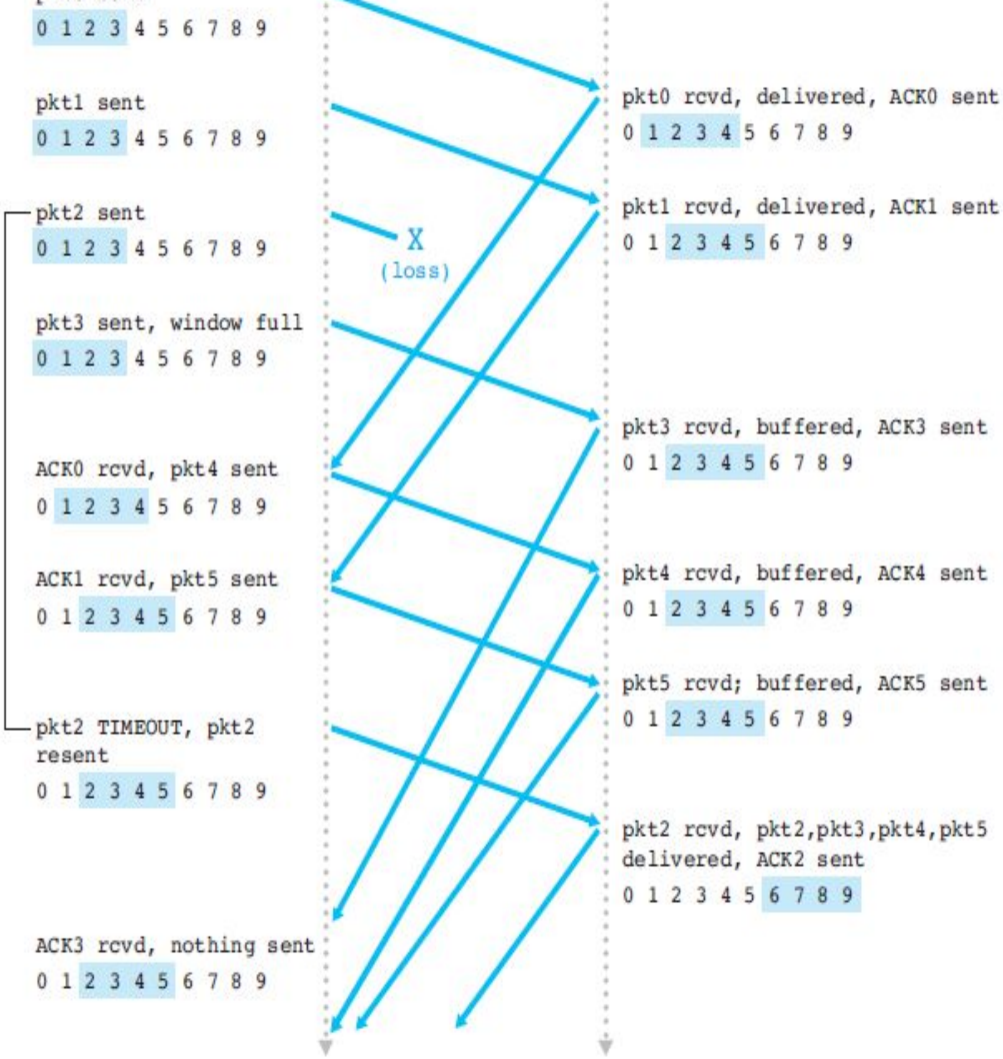
- Works using Sliding-Window approach. Uses cumulative acknowledgements.
- Developed to overcome the drawbacks of Go-Back-N, to avoid unnecessary retransmissions.
- A window of size-N is defined to limit the number of outstanding and unacknowledged packets in the pipeline.
- The SR receiver will acknowledge a correctly received packet whether or not it is in order. Out-of-order packets are buffered until any missing packets are received, at which point a batch of packets can be delivered to the upper layer in the sequence order.



a. Sender view of sequence numbers

b. Receiver view of sequence numbers

Working Example:



SR-Sender: Events and Actions

1. Data received from application layer: When receives data from the application, adds next available sequence number to it. Adds checksum for error detection at the receiver end. Checks whether the sequence number is within the limit of the window. If so, data is packetized and sent. If not, it is buffered for later transmission.
2. ACK Received: If the sender receives the ACK, marks the packet as acknowledged and then slides the window toward the new unacknowledged frame having smallest number. The untransmitted packets from the slided window would be transmitted by the sender.
3. Timeout: If the sender has not received ACK within the anticipated time and the timer expires, the buffered copy of the current frame will be retransmitted.

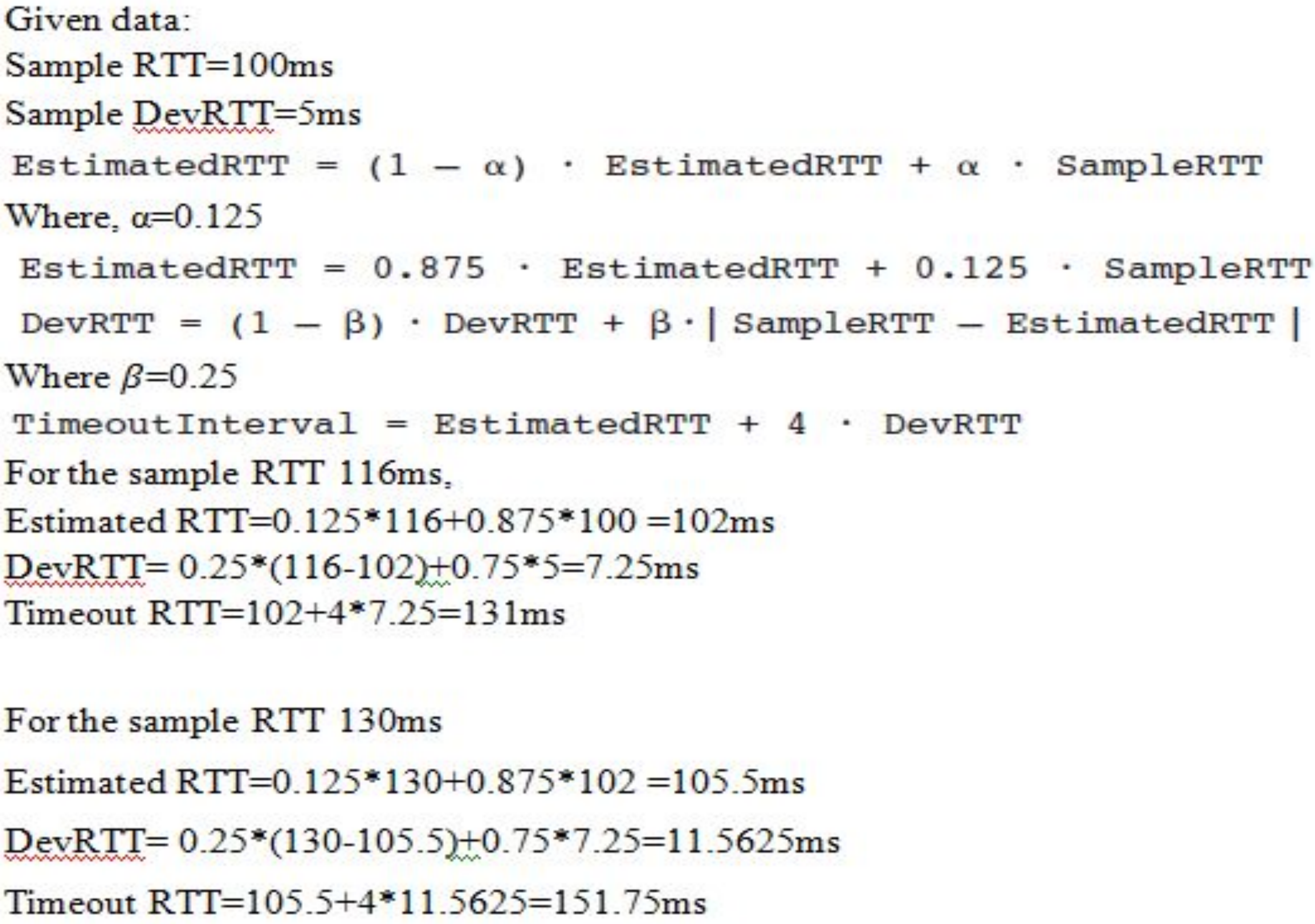
SR-Receiver:Events and Actions

1. Receiving the packet having the sequence number within the logical window limit, buffers it and sends the acknowledgement to the sender. Received in-sequence packets are pushed to the upper layer. Then slides the window to the next set of expected sequence numbers.
2. Discarding of duplicate packet: If the arrived packet has been already acknowledged and buffered at the end of receiver, then it would be discarded silently without pushing it to the higher layer. This mechanism reduces the generation of duplicate packets at the receiving application.

10. Problem solving on RTT estimation(RTT estimation, Timeout calculations)

Problem: Suppose that two measured sample RTT values are 116 ms and 130ms. Compute: i) Estimated RTT after each of these sample RTT value is obtained. Consider weight factor, α=0.125 and estimated RTT is 100 ms just before first of the samples obtained. ii) Compute DevRTT and iii).Timeout values

Consider β=0.25 and DevRTT was 5 ms before first of these samples are obtained.



Given data:

Sample RTT=100ms

Sample DevRTT=5ms

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

Where, α=0.125

$$\text{EstimatedRTT} = 0.875 \cdot \text{EstimatedRTT} + 0.125 \cdot \text{SampleRTT}$$

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

Where β=0.25

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT}$$

For the sample RTT 116ms,

Estimated RTT=0.125*116+0.875*100 =102ms

DevRTT= 0.25*(116-102)+0.75*5=7.25ms

Timeout RTT=102+4*7.25=131ms

For the sample RTT 130ms

Estimated RTT=0.125*130+0.875*102 =105.5ms

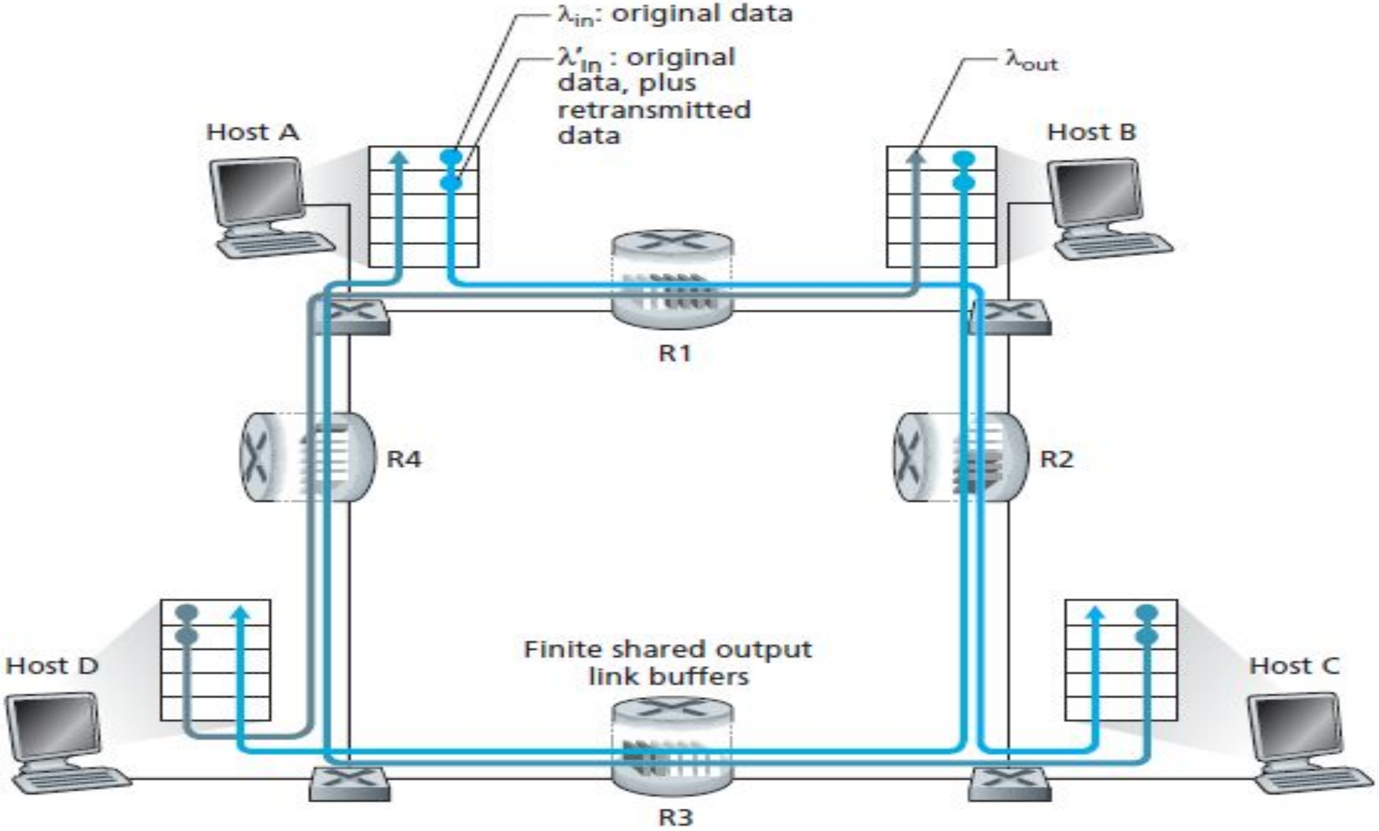
DevRTT= 0.25*(130-105.5)+0.75*7.25=11.5625ms

Timeout RTT=105.5+4*11.5625=151.75ms

11. Explain causes and cost of congestion with 4 routers with limited buffer size. (Explain Delay and throughput graphs also)

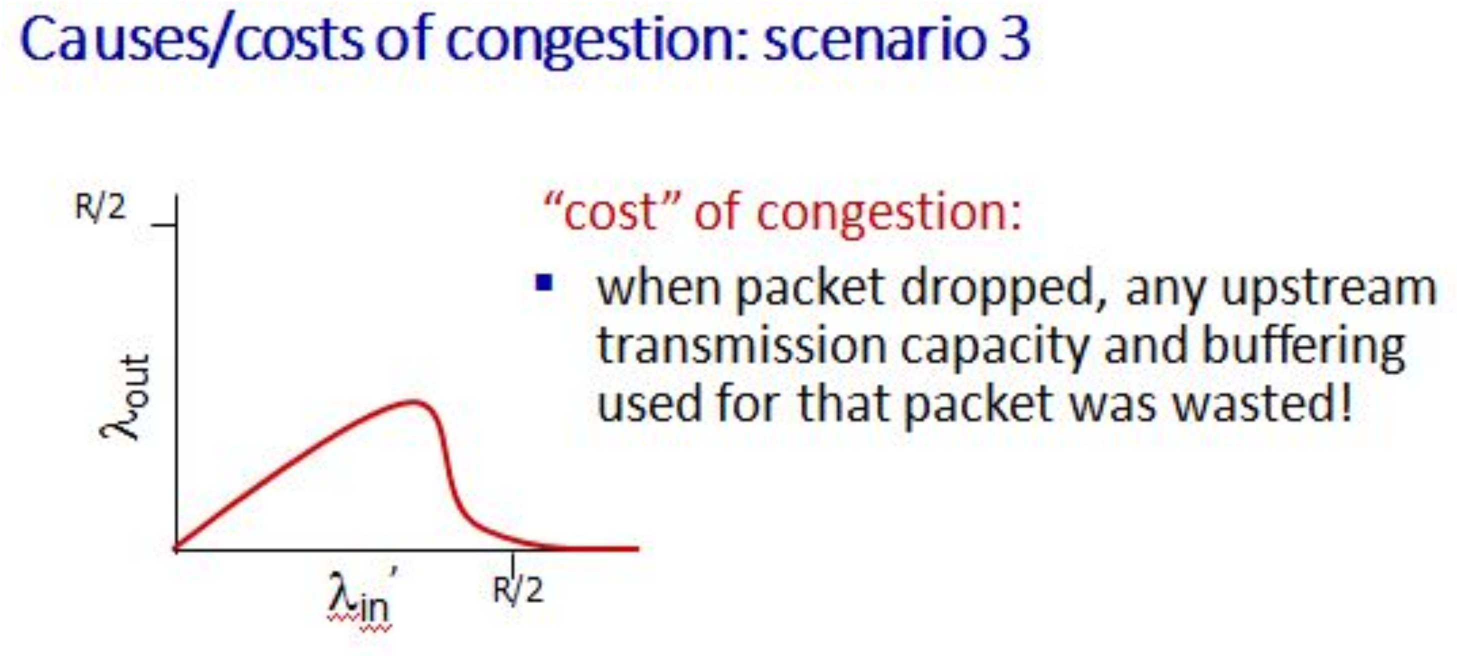
Scenario 3: Four Senders, Routers with Finite Buffers, and Multihop Paths

- In this congestion scenario, four hosts transmit packets, each over overlapping two-hop paths, as shown in Figure below.
- The major causes of congestion are:
  - four senders (Number of nodes)
  - multi-hop paths
  - timeout/retransmit(Transmission delays, packet drops or duplications)
- Costs(consequences) of Congestion are:
  - Decrease in throughput
  - Delay increases as capacity approached
  - loss/retransmission
  - un-needed duplicates of packets
  - upstream transmission capacity / buffering wasted for packets lost downstream


λ_{in}: original data
λ'_{in} : original data, plus retransmitted data
λ_{out}
Host A
Host B
R1
R4
R2
Host D
Host C
Finite shared output link buffers
R3

Let's consider the connection from Host A to Host C, passing through routers R1 and R2. The A–C connection shares router R1 with the D–B connection and shares router R2 with the B–D connection. For extremely small values of λ_{in}, buffer overflows are rare (as in congestion scenarios 1 and 2), and the throughput approximately equals the offered load. For slightly larger values of λ_{in}, the corresponding throughput is also larger, since more original data is being transmitted into the network and delivered to the destination, and overflows are still rare. Thus, for small values of λ_{in}, an increase in λ_{in} results in an increase in λ_{out}.

Having considered the case of extremely low traffic, let's next examine the case that λ_{in} (and hence λ'_{in}) is extremely large. Consider router R2. The A–C traffic arriving to router R2 (which arrives at R2 after being forwarded from R1) can have an arrival rate at R2 that is at most *R*, the capacity of the link from R1 to R2, regardless of the value of λ_{in}. If λ'_{in} is extremely large for all connections (including the B–D connection), then the arrival rate of B–D traffic at R2 can be much larger than that of the A–C traffic. Because the A–C and B–D traffic must compete at router R2 for the limited amount of buffer space, the amount of A–C traffic that successfully gets through R2 (that is, is not lost due to buffer overflow) becomes smaller and smaller as the offered load from B–D gets larger and larger. In the limit, as the offered load approaches infinity, an empty buffer at R2 is immediately filled by a B–D packet, and the throughput of the A–C connection at R2 goes to zero. This, in turn, implies that the A–C end-to-end throughput goes to zero in the limit of heavy traffic.



Causes/costs of congestion: scenario 3

"cost" of congestion:

- when packet dropped, any upstream transmission capacity and buffering used for that packet was wasted!

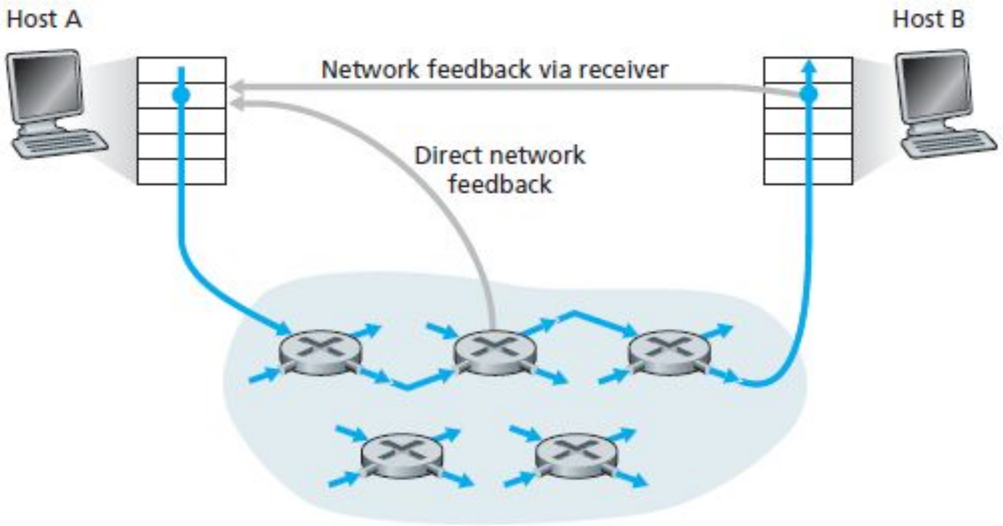
12. Explain network assisted congestion control mechanisms

Network-assisted congestion control:

- With network-assisted congestion control, network-layer components (that is, routers) provide explicit feedback to the sender regarding the congestion state in the network.
- For network-assisted congestion control, congestion information is typically fed back from the network to the sender in one of two ways,
- Direct feedback may be sent from a network router to the sender.
- This form of notification typically takes the form of a choke packet (essentially saying, “I’m congested!”).

- The second form of notification occurs when a router marks/updates a field in a packet flowing from sender to receiver to indicate congestion.
- Upon receipt of a marked packet, the receiver then notifies the sender of the congestion indication. Note that this latter form of notification takes at least a full round-trip time.

Example:congestion-control algorithm in ATM ABR based transmission.

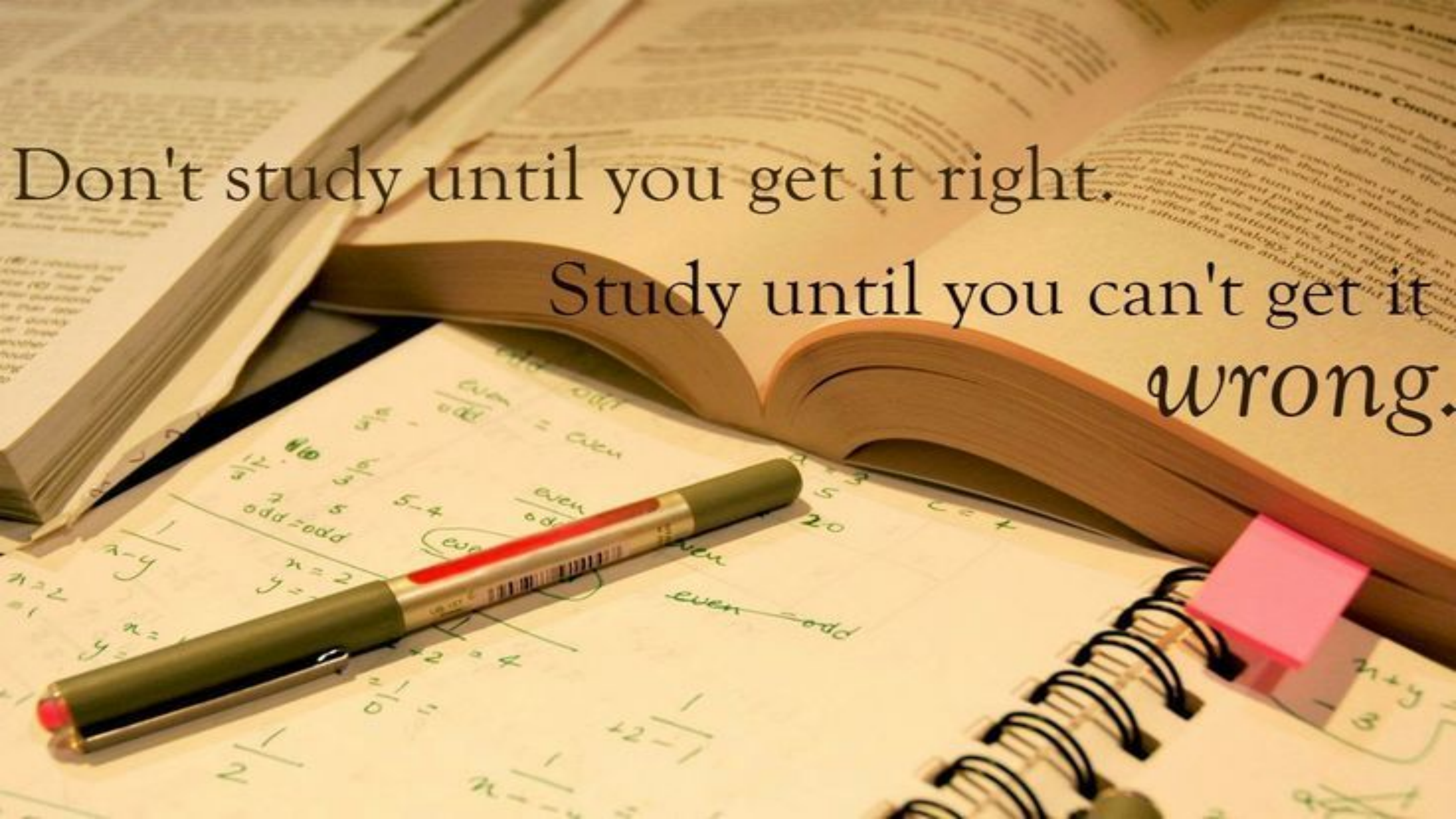


- Data cells are transmitted from a source to a destination through a series of intermediate routers.
- Resource management(RM) cells are embedded within the data cells.
- These RM cells can be used to convey congestion-related information
- among the hosts and routers.
- When an RM cell arrives at a destination, it will be turned around and sent back to the sender(Network feedback via receiver) (possibly after the destination has modified the contents of the RM cell).
- It is also possible for a switch to generate an RM cell itself and send this RM cell directly to a source(Direct feedback)
- RM cells can thus be
- used to provide both direct network feedback and network feedback via the
- Receiver.
- ABR provides three mechanisms for signaling congestion-related information
- from the switches to the receiver:
-

EFCI bit: Each data cell contains an explicit forward congestion indication (EFCI) bit. A congested network switch can set the EFCI bit in a data cell to 1 to signal congestion to the destination host. The destination must check the EFCI bit in all received data cells. When an RM cell arrives at the destination, if the most recently received data cell had the EFCI bit set to 1, then the destination sets the congestion indication bit (the CI bit) of the RM cell to 1 and sends the RM cell back to the sender.

CI and NI bits: RM cells have a congestion indication (CI) bit and a no increase (NI) bit that can be set by a congested network switch. Specifically, a switch can set the NI bit in a passing RM cell to 1 under mild congestion and can set the CI bit to 1 under severe congestion conditions. When a destination host receives an RM cell, it will send the RM cell back to the sender

- **ER setting.** Each RM cell also contains a 2-byte explicit rate (ER) field. A congested switch may lower the value contained in the ER field in a passing RM cell. In this manner, the ER field will be set to the minimum supportable rate of all switches on the source-to-destination path.


Don't study until you get it right.
Study until you can't get it
wrong.

Module-3: Network Layer

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

With a neat diagram, explain the Router structure

- Router is an internetworking device responsible for routing the packets from source to destination. It works at network layer.
- Following figure depicts the major structural components of a router.

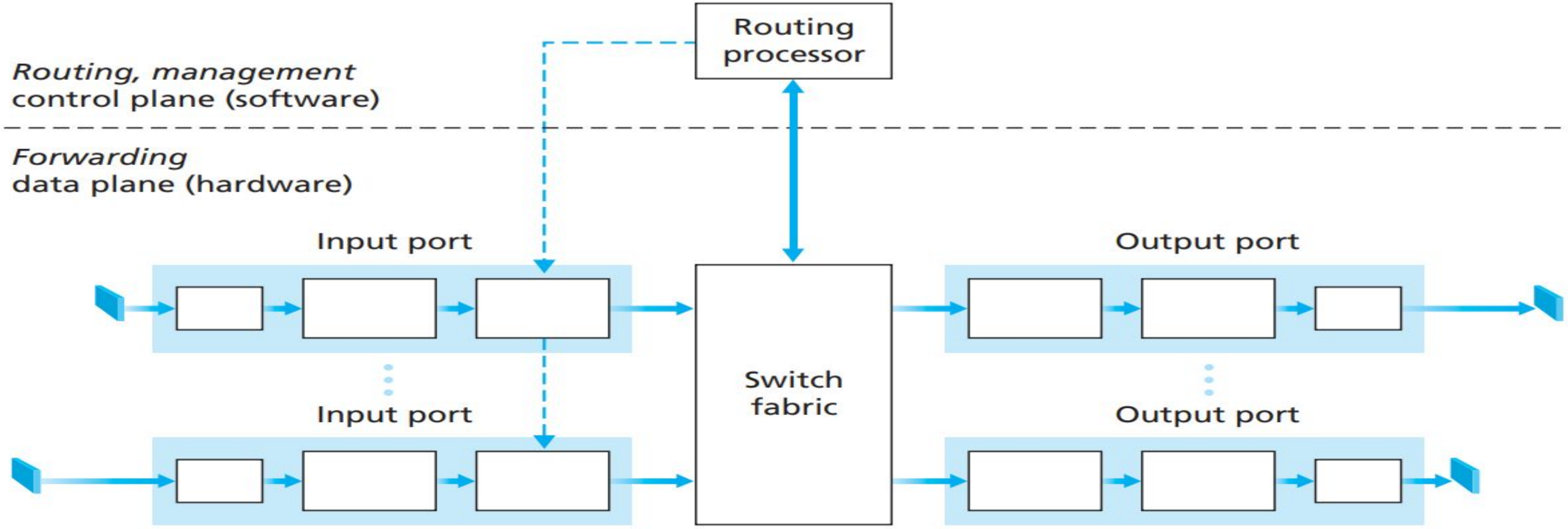


Figure 1. Structural components of a router

Input ports:

- An input port 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.
- Switching can be accomplished in a number of ways:

1.Switching via memory. 2.Switching via a bus 3.Switching via an interconnection network

Switching via memory:

- switching between input and output ports being done under direct control of the CPU (routing processor)
- An input port signals the packet to the processor unit.
- 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 buffer. Input and output ports are connected through a common shared memory.

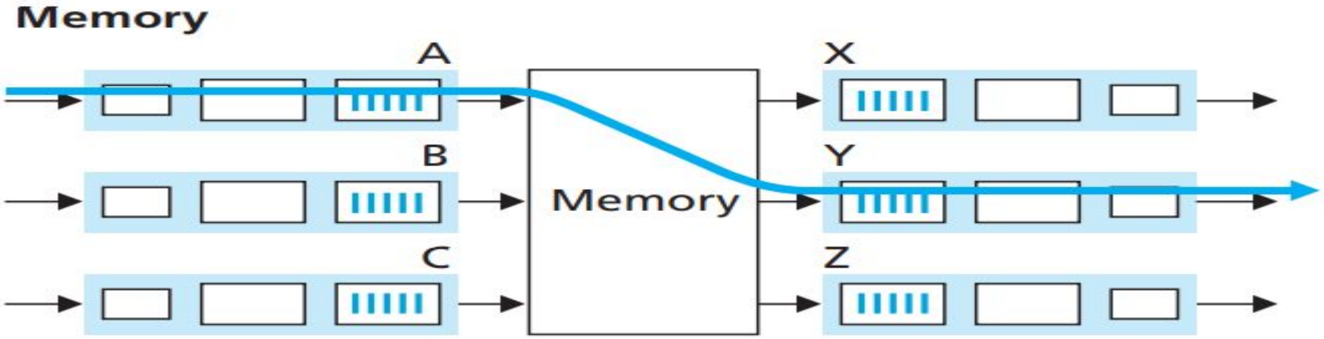


Figure 2. Switching via memory

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 prepend 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.

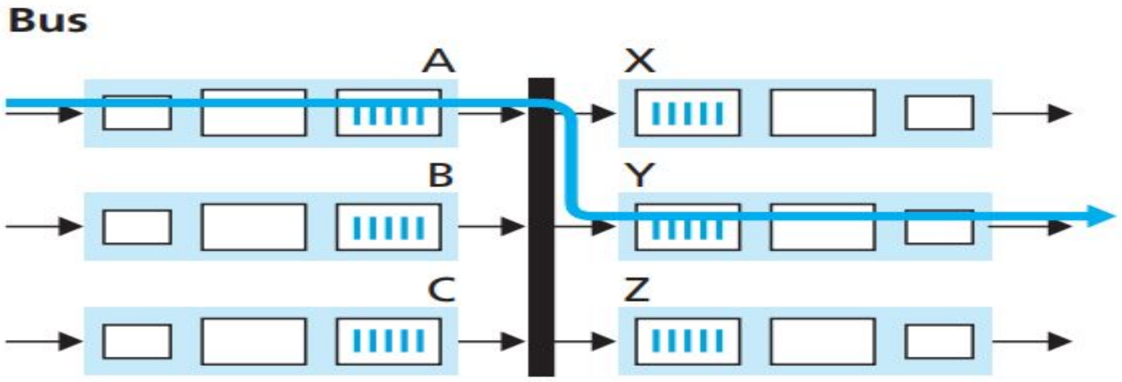


Figure 3. Switching via Bus

Switching via an interconnection network:Uses a crossbar Switch

- 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.

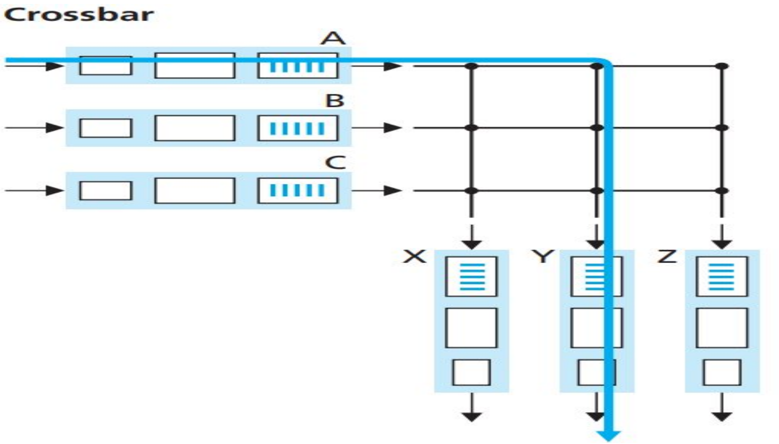


Figure 4. Switching via crossbar switch

- crossbar networks are capable of forwarding multiple packets in parallel.

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.

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.
- Router control plane functions are usually implemented in software and execute on the routing processor

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.
- 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.

2. Explain IPv6 datagram format

IPv6 Datagram Format:

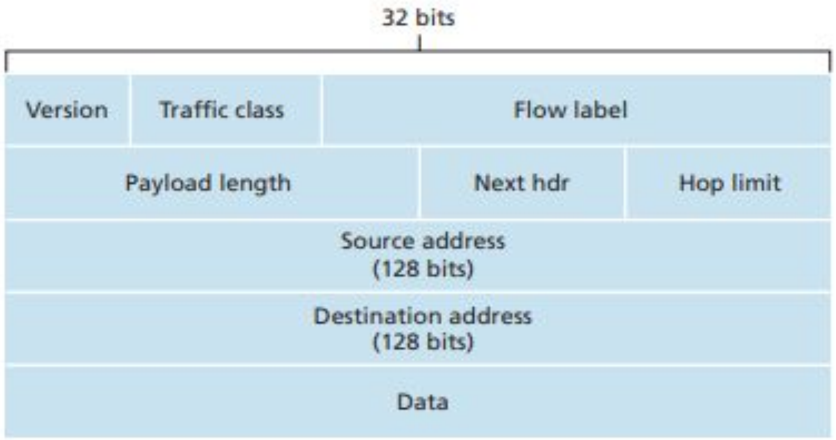


Figure 5. IPv6 datagram

- **IPv6 has expanded addressing capabilities:** 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.

3. Differentiate between IPv4 and IPv6 datagrams

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.

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.

Discuss different approaches for transition from iPv4 to IPv6

Declaring a flag day: A given time and date when all Internet machines would be turned off and upgraded from IPv4 to IPv6. But, a flag day involving hundreds of millions of machines and millions of network administrators and users, is even more unthinkable today.

Dual-stack: 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.

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.

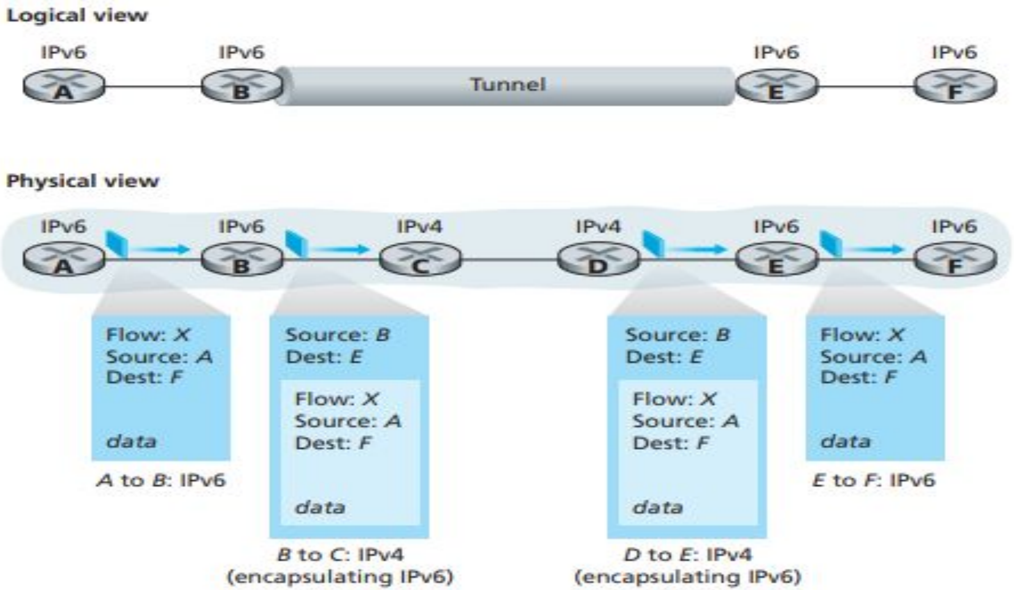


Figure 6. Tunneling

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

The intervening IPv4 routers in the tunnel route this IPv4 datagram among themselves, just as they would any other datagram, blissfully unaware that the IPv4 datagram itself contains a complete IPv6 datagram.

The IPv6 node on the receiving side of the tunnel eventually receives the IPv4 datagram (it is the destination of the IPv4 datagram!), determines that the IPv4 datagram contains an IPv6 datagram, extracts the IPv6 datagram, and then routes the IPv6 datagram exactly as it would if it had received the IPv6 datagram from a directly connected IPv6 neighbor.

Write and explain Dijkstra's routing algorithm OR Explain the working of link state routing algorithm with an example

Dijkstra's algorithm is a Link-State Routing algorithm named after its inventor.

Dijkstra's algorithm computes the least-cost path from one node (the source, which we will refer to as u) to all other nodes in the network.

Dijkstra's **algorithm is iterative** and has the property that after the k^{th} iteration of the algorithm, the least-cost paths are known to k destination nodes, and among the least-cost paths to all destination nodes, these k paths will have the k smallest costs.

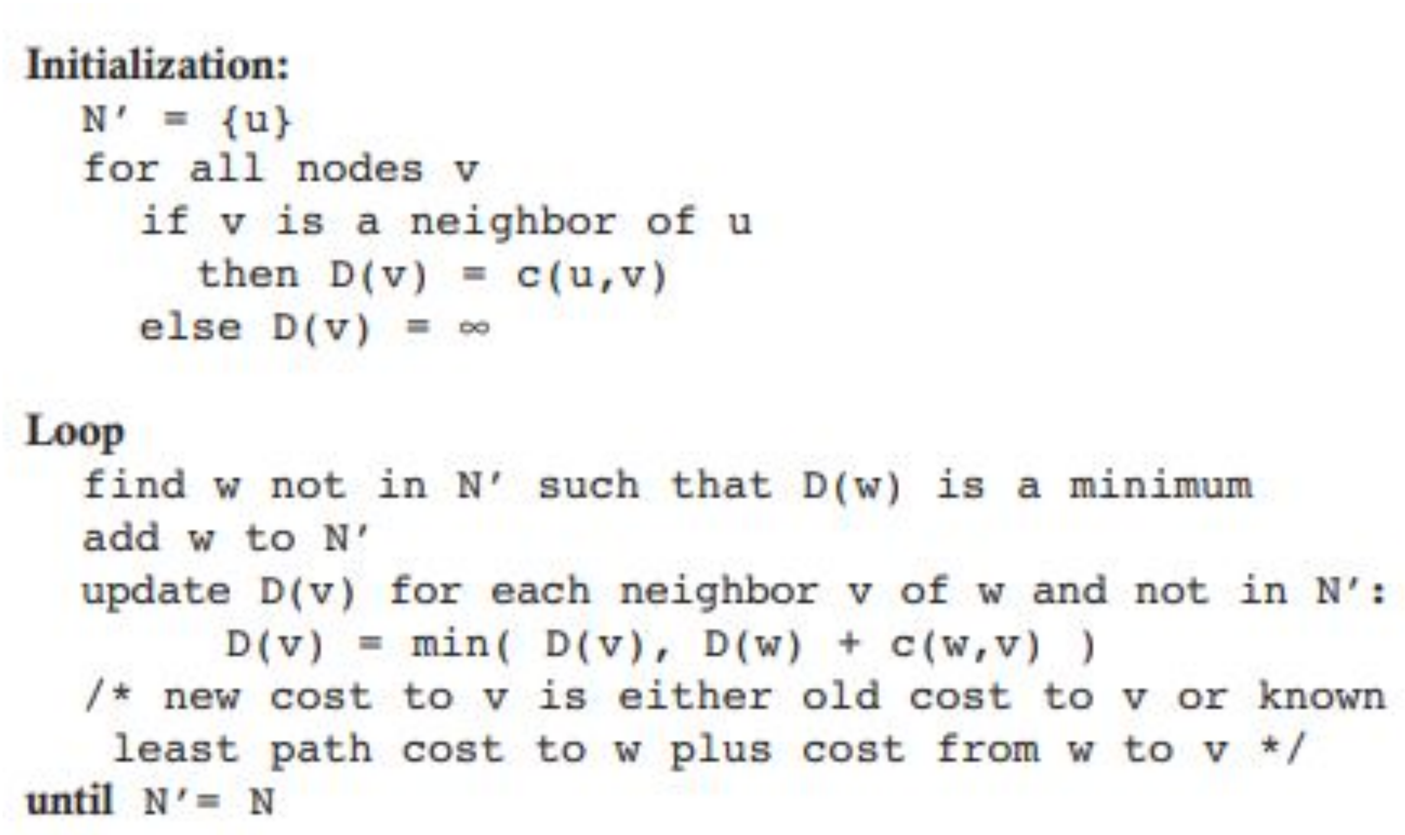
Let us define the following notation:

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

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

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

Link-State (LS) Algorithm for Source Node u



```
Initialization:
   N' = {u}
   for all nodes v
      if v is a neighbor of u
         then D(v) = c(u,v)
      else D(v) = ∞

Loop
   find w not in N' such that D(w) is a minimum
   add w to N'
   update D(v) for each neighbor v of w and not in N':
         D(v) = min( D(v), D(w) + c(w,v) )
   /* new cost to v is either old cost to v or known
    least path cost to w plus cost from w to v */
until N'= N
```

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

Example: Consider the following network topology

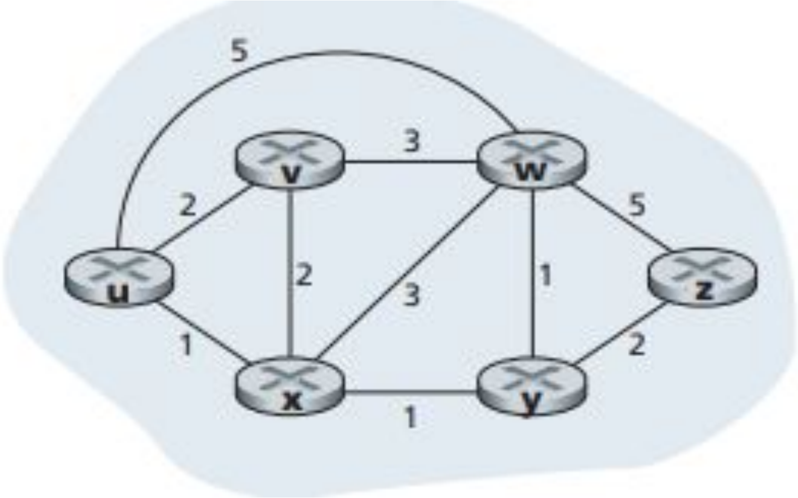
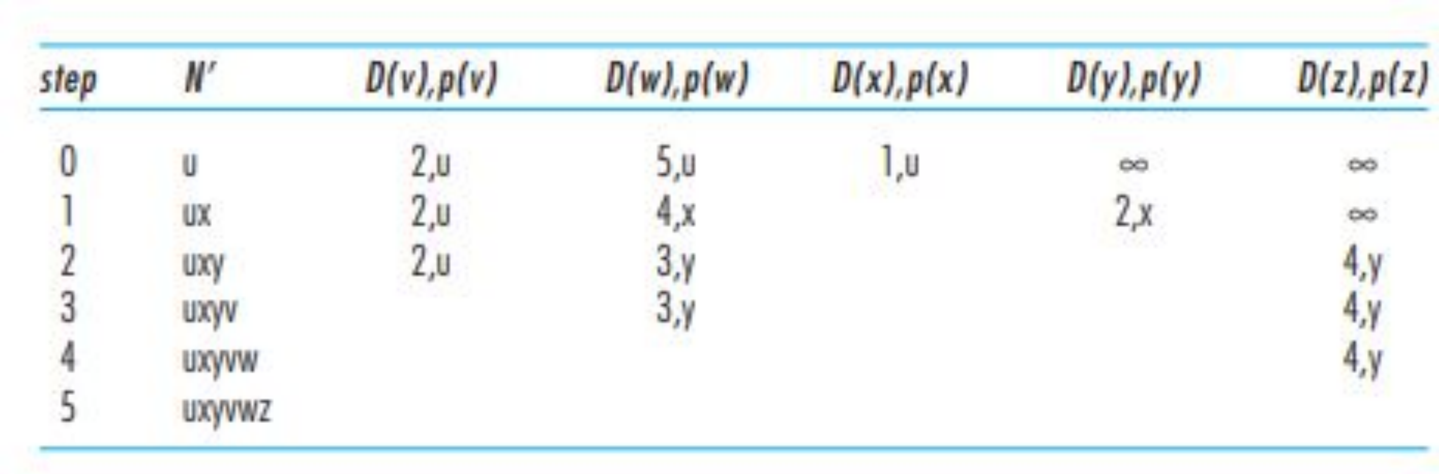


Figure 7. Network topology

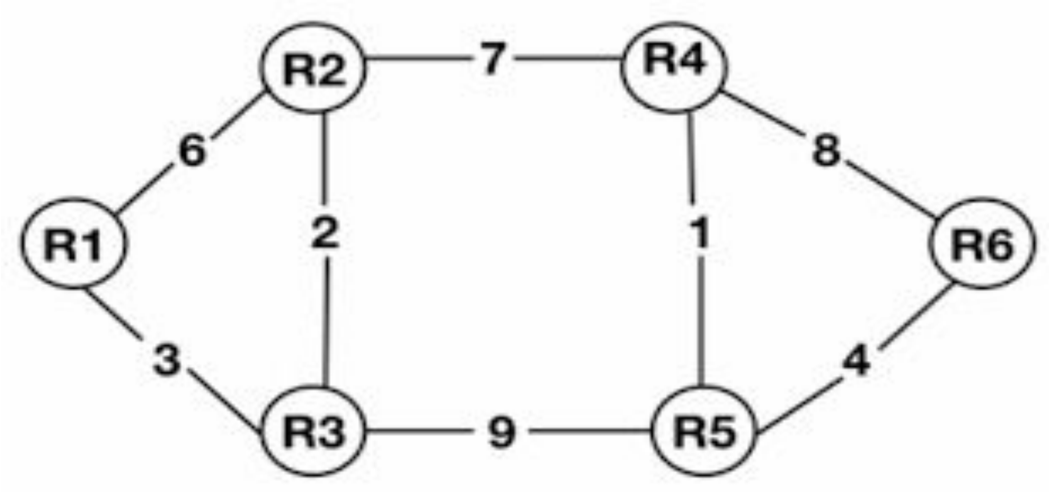
let’s consider the network in Figure 7, and compute the least-cost paths from u to all possible destinations. A tabular summary of the algorithm’s computation is shown in Table given below, where each line in the table gives the values of the algorithm’s variables at the end of the iteration.



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

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

Find the shortest path from node A to all the other nodes for the given network using Dijkstra's algorithm

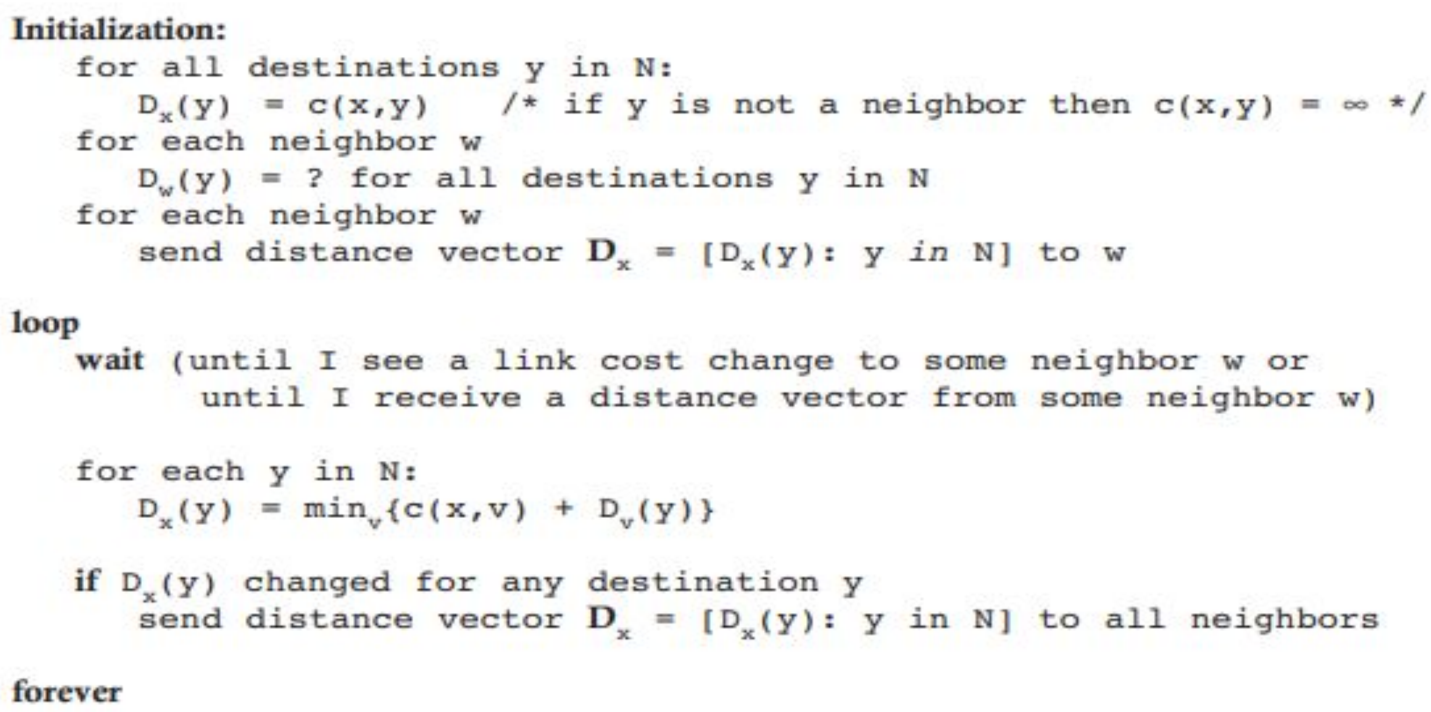


Write and explain Bellman-Ford routing algorithm 8M OR Explain the working of Distance vector routing algorithm

- Distance vector (DV) algorithm is iterative, asynchronous, and distributed.
- Each node x begins with Dx (y), an estimate of the cost of the least-cost path from itself to node y, for all nodes in N. Let Dx = [Dx (y): y in N] be node x's distance vector, which is the vector of cost estimates from x to all other nodes, y, in N. With the DV algorithm, each node x maintains the following routing information.
- For each neighbor v, the cost c(x,v) from x to directly attached neighbor v,
    - Node x's distance vector, that is, Dx = [Dx (y): y in N], containing x's estimate of its cost to all destinations, y, in N
    - The distance vectors of each of its neighbors, that is, Dv = [Dv (y): y in N] for each neighbor v of x from time to time, each node sends a copy of its distance vector to each of its neighbors.
    - When a node x receives a new distance vector from any of its neighbors v, it saves v's distance vector, and then uses the Bellman-Ford equation to update its own distance vector as follows: Dx (y) minv {c(x,v) + Dv (y)} for each node y in N

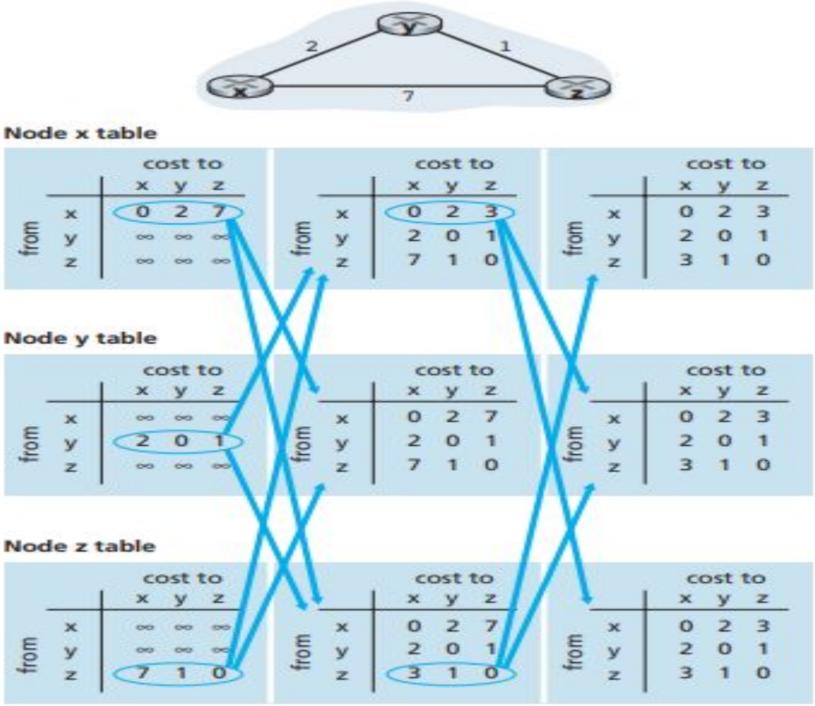
- If node x’s distance vector has changed as a result of this update step, node x will then send its updated distance vector to each of its neighbors, which can in turn update their own distance vectors.

Distance-Vector (DV) Algorithm: At each node, x



```
Initialization:
    for all destinations y in N:
        $D_x(y) = c(x,y)$    /* if y is not a neighbor then $c(x,y) = \infty$ */
    for each neighbor w
        $D_w(y) = ?$ for all destinations y in N
    for each neighbor w
        send distance vector $\mathbf{D}_x = [D_x(y)$: y in N] to w

loop
    wait (until I see a link cost change to some neighbor w or
          until I receive a distance vector from some neighbor w)

    for each y in N:
        $D_x(y) = \min_v\{c(x,v) + D_v(y)\}$

    if $D_x(y)$ changed for any destination y
        send distance vector $\mathbf{D}_x = [D_x(y)$: y in N] to all neighbors

forever
```

- In the DV (Bellman Ford) algorithm , a node x updates its distance-vector estimate when it either sees a cost change in one of its directly attached links or receives a distance vector update from some neighbor.
- Figure given below illustrates the operation of the DV (Bellman Ford) algorithm for the simple three node network shown at the top of the figure.



Node x table

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

Node y table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

		cost to		
		x	y	z
from	x	0	2	7
	y	2	0	1
	z	7	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

Node z table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
	z	7	1	0

		cost to		
		x	y	z
from	x	0	2	7
	y	2	0	1
	z	3	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

- The operation of the algorithm is illustrated in a synchronous manner, where all nodes simultaneously receive distance vectors from their neighbors, compute their new distance vectors, and inform their neighbors if their distance vectors have changed.
- The leftmost column of the figure displays three initial routing tables for each of the three nodes.
- After initialization, each node sends its distance vector to each of its two neighbors. This is illustrated in above given figure by the arrows from the first column of tables to the second column of tables.
- After receiving the updates, each node recomputes its own distance vector.
- The second column therefore displays, for each node, the node's new distance vector along with distance vectors just received from its neighbors.
- After the nodes recompute their distance vectors, they again send their updated distance vectors to their neighbors (if there has been a change).

The process of receiving updated distance vectors from neighbors, recomputing routing table entries, and informing neighbors of changed costs of the least-cost path to a destination continues until no update messages are sent. At this point, no further routing table calculations will occur and the algorithm will enter a quiescent state.

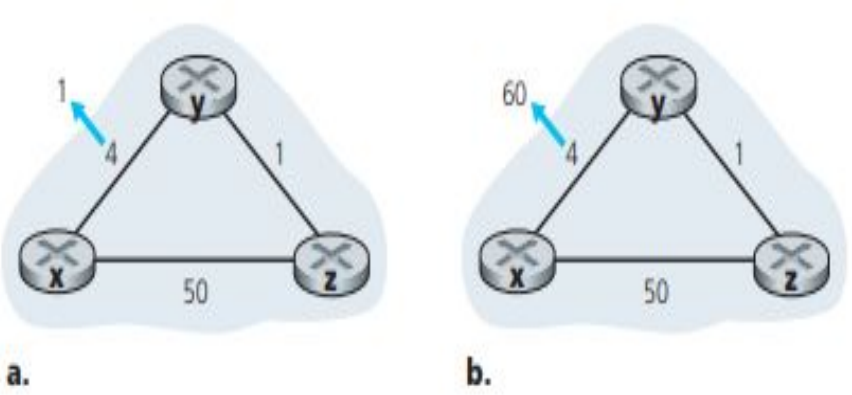
What is ‘Counting to infinity’ problem in distance vector routing? Discuss about the solution.

- When a node running the DV(Bellman Ford) algorithm, detects a change in the link cost from itself to a neighbor, it updates its distance vector and, if there’s a change in the cost of the least-cost path, informs its neighbors of its new distance vector.
- Decreased cost between any set of nodes propagates quickly through the network. The algorithm converges quickly. But when the link cost increases than the cost of link before, algorithm may continue through iterations to find the new shortest path as per the logic of distance vector routing. But after certain number of iterations, it may form routing loops and algorithm never gets converged. This problem is called as counting to infinity problem.

Example:

Example for counting to infinity Problem:

- Suppose that the link cost between nodes x and y in the below given figure, increases from 4 to 60, as shown in Figure(b). Before the link cost changes, D y (x) = 4, D y (z) = 1, Dz (y) = 1, and Dz (x) = 5.
- At time t 0, y detects the link-cost change (the cost has changed from 4 to 60). y computes its new minimum-cost path to x to have a cost of D y (x) = min{c(y,x) + Dx (x), c(y,z) + Dz (x)} = min{60 + 0, 1 + 5} = 6



- Of course, with our global view of the network, we can see that this new cost via z is wrong. But the only information node y has is that its direct cost to x is 60 and that z has last told y that z could get to x with a cost of 5.
- So in order to get to x, y would now route through z, fully expecting that z will be able to get to x with a cost of 5.
- As of t_1 we have a routing loop in order to get to x, y routes through z, and z routes through y.
- A routing loop is like a black hole a packet destined for x arriving at y or z as of t_1 will bounce back and forth between these two nodes forever (or until the forwarding tables are changed).
- Since node y has computed a new minimum cost to x, it informs z of its new distance vector at time t_1.
- Sometime after t_1 , z receives y's new distance vector, which indicates that y's minimum cost to x is 6.

- z knows it can get to y with a cost of 1 and hence computes a new least cost to x of Dz (x) = min{50 + 0,1 + 6} = 7.
- Since z's least cost to x has increased, it then informs y of its new distance vector at t_2.
- In a similar manner, after receiving z's new distance vector, y determines D y (x) = 8 and sends z its distance vector. z then determines Dz (x) = 9 and sends y its distance vector, and so on.
- This problem of ending with routing loops is called Counting to infinity problem.(As this looping may continue forever if there is a link break, which is the only link connecting the node to the network.)

Solution to counting to infinity problem:

The specific looping scenario just described can be avoided using a technique known as Split horizon poisoned reverse. When the algorithm goes through the iterations beyond the number of nodes in the network, the nodes update their default state as initial state by updating the distance to all the nodes as infinity. The shortest path computation begins from the initial state.

Differentiate between Link State and Distance vector routing algorithms

The DV and LS algorithms take complementary approaches towards computing routing.

Message complexity:

- LS requires each node to know the cost of each link in the network. This requires O(|N||E|) messages to be sent. Also, whenever a link cost changes, the new link cost must be sent to all nodes
- The DV algorithm requires message exchanges between directly connected neighbors at each iteration. When link costs change, the DV algorithm will propagate the results of the changed link cost only if the new link cost results in a changed least-cost path for one of the nodes attached to that link.

Speed of convergence:

- Implementation of LS is an $O(|N|^2)$ algorithm requiring $O(|N|\ |E|)$ messages.
- The DV algorithm can converge slowly and can have routing loops while the algorithm is converging. DV also suffers from the count-to-infinity problem.

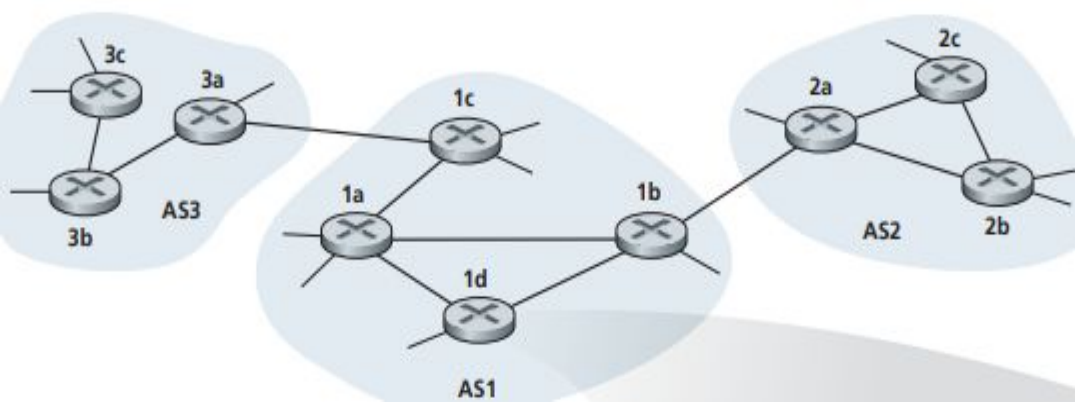
Robustness:

- If a router fails, misbehaves, or is sabotaged, under LS, a router could broadcast an incorrect cost for one of its attached links (but no others). An LS node is computing only its own forwarding tables; other nodes are performing similar calculations for themselves. This means route calculations are somewhat separated under LS, providing a degree of robustness.
- Under DV, a node can advertise incorrect least-cost paths to any or all destinations.

Write a short note on Hierarchical routing

- Today's public Internet consists of hundreds of millions of hosts. Storing routing information at each of these hosts would clearly require enormous amounts of memory.This would result in two major problems. **Scaling of networks** and Maintaining **autonomy of the network** operations.
- Both of these problems can be solved by organizing routers into autonomous systems (ASs), with each AS consisting of a group of routers that are typically under the same administrative control. Routers within the same AS all run the same routing algorithm (for example, an LS or DV algorithm) and have information about each other exactly as was the case in our idealized model in the preceding section.
- The routing algorithm running within an autonomous system is called an intra autonomous system routing protocol. It will be necessary, of course, to connect ASs to each other, and these routers which connects the different ASs are called gateway routers.

- Forwarding table is configured by both intra- and inter-AS routing algorithm
- Intra-AS sets entries for internal dests
- Inter-AS & Intra-As sets entries for external dests



Inter-AS tasks:

Suppose router in AS1 receives datagram for which dest is outside of AS1 Router should forward packet towards on of the gateway routers, but which one? to learn which destinations are reachable through AS2 and which through AS3 to propagate this reachability info to all routers in AS1

- The problems of scale and administrative authority are solved by defining autonomous systems. Within an AS, all routers run the same intra-AS routing protocol. Among themselves, the ASs run the same inter-AS routing protocol. The problem of scale is solved because an intra-AS router need only know about routers within its AS.
- The problem of administrative authority is solved since an organization can run whatever intra-AS routing protocol it chooses; however, each pair of connected ASs needs to run the same inter-AS routing protocol to exchange reachability information. In the following section

Explain the working of OSPF

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

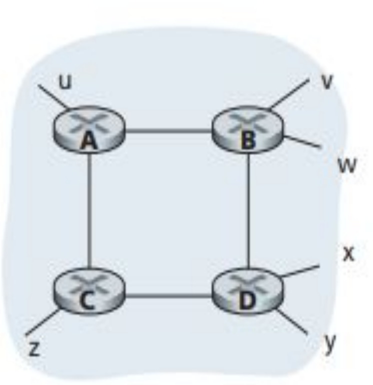
- OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra least-cost path algorithm.
- With OSPF, a router constructs a complete topological map (that is, a graph) of the entire autonomous system.
- The router then locally runs Dijkstra’s shortest-path algorithm to determine a shortest-path tree to all subnets, with itself as the root node.
- Individual link costs are configured by the network administrator (see Principles and Practice: Setting OSPF Weights). The administrator might choose to set all link costs to 1, thus achieving minimum-hop routing, or might choose to set the link weights to be inversely proportional to link capacity in order to discourage traffic from using low-bandwidth links.
- OSPF does not mandate a policy for how link weights are set (that is the job of the network administrator), but instead provides the mechanisms (protocol) for determining least-cost path routing for the given set of link weights

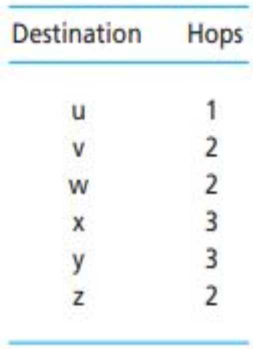
- With OSPF, a router broadcasts routing information to all other routers in the autonomous system, not just to its neighboring routers.
- The OSPF protocol must itself implement functionality such as reliable message transfer and link-state broadcast. The OSPF protocol also checks that links are operational (via a HELLO message that is sent to an attached neighbor) and allows an OSPF router to obtain a neighboring router's database of network-wide link state.
- Exchanges between OSPF routers (for example, link-state updates) can be authenticated. With authentication, only trusted routers can participate in the OSPF protocol within an AS, thus preventing malicious intruders

- Two types of authentication can be configured simple and MD5. With simple authentication, the same password is configured on each router. When a router sends an OSPF packet, it includes the password in plaintext.
- Multiple same-cost paths. When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used (that is, a single path need not be chosen for carrying all traffic when multiple equal-cost paths exist).
- Integrated support for unicast and multicast routing. Multicast OSPF (MOSPF) [RFC 1584] provides simple extensions to OSPF to provide for multicast routing
- Support for hierarchy within a single routing domain. Perhaps the most significant advance in OSPF is the ability to structure an autonomous system hierarchically.

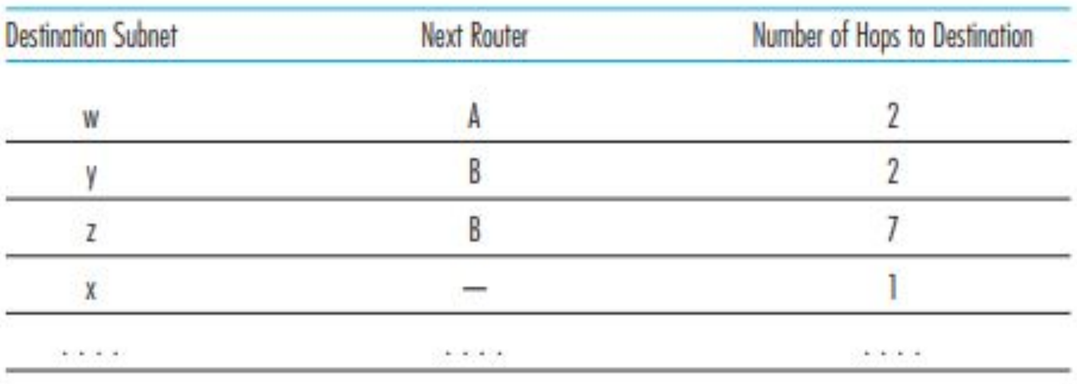
Explain RIP protocol

- RIP (Routing Information Protocol): RIP was one of the earliest intra-AS Internet routing protocols and is still in widespread use today.
- RIP is a distance-vector protocol that operates in a manner very close to the idealized DV protocol(Bellman Ford algorithm)
- In RIP (and also in OSPF), costs are actually from source router to a destination subnet. RIP uses the term hop, which is the number of subnets traversed along the shortest path from source router to destination subnet, including the destination subnet.





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



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

- Note that the routing table has three columns. The first column is for the destination subnet, the second column indicates the identity of the next router along the shortest path to the destination subnet, and the third column indicates the number of hops (that is, the number of subnets that have to be traversed, including the destination subnet) to get to the destination subnet along the shortest path.
- Distance vectors: exchanged among neighbors every 30 sec via Response Message (also called advertisement)
- Each advertisement: list of up to 25 destination nets within AS.

RIP Link failure and recovery:

If no advertisement heard after 180 sec --> neighbor/link declared dead

- routes via neighbor invalidated
- new advertisements sent to neighbors
- neighbors in turn send out new advertisements (if tables changed)
- link failure info quickly propagates to entire net
- poison reverse used to prevent ping-pong loops (infinite distance = 16 hops)

Explain Broadcast routing algorithms

There are three different approaches in broadcast algorithms.

1. **N-wayunicast approach**

- Given N destination nodes, the source node simply makes N copies of the packet, addresses each copy to a different destination, and then transmits the N copies to the N destinations using unicast routing

Drawbacks:

- The first drawback is its inefficiency. If the source node is connected to the rest of the network via a single link, then N separate copies of the (same) packet will traverse this single link
- An implicit assumption of N-way-unicast is that broadcast recipients, and their addresses, are known to the sender. This would add more overhead and, importantly, additional complexity to a protocol

- A final drawback of N-way-unicast relates to the purposes for which broadcast is to be used

2. Uncontrolled Flooding:

- The most obvious technique for achieving broadcast is a flooding approach in which the source node sends a copy of the packet to all of its neighbors. When a node receives a broadcast packet, it duplicates the packet and forwards it to all of its neighbors (except the neighbor from which it received the packet).

Drawback:

- If the graph has cycles, then one or more copies of each broadcast packet will cycle indefinitely.
- When a node is connected to more than two other nodes, it will create and forward multiple copies of the broadcast packet, each of which will create multiple copies of itself (at other nodes with more than two neighbors), and so on creating a broadcast storm.

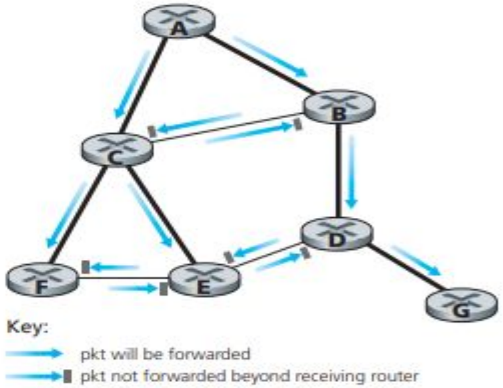
3. Controlled Flooding:

a. **Sequence-number-controlled flooding:**

- A source node puts its address (or other unique identifier) as well as a broadcast sequence number into a broadcast packet, then sends the packet to all of its neighbors.
- Each node maintains a list of the source address and sequence number of each broadcast packet it has already received, duplicated, and forwarded.
- When a node receives a broadcast packet, it first checks whether the packet is in this list. If so, the packet is dropped; if not, the packet is duplicated and forwarded to all the node's neighbors (except the node from which the packet has just been received).

b. Reverse Path Forwarding (RPF):

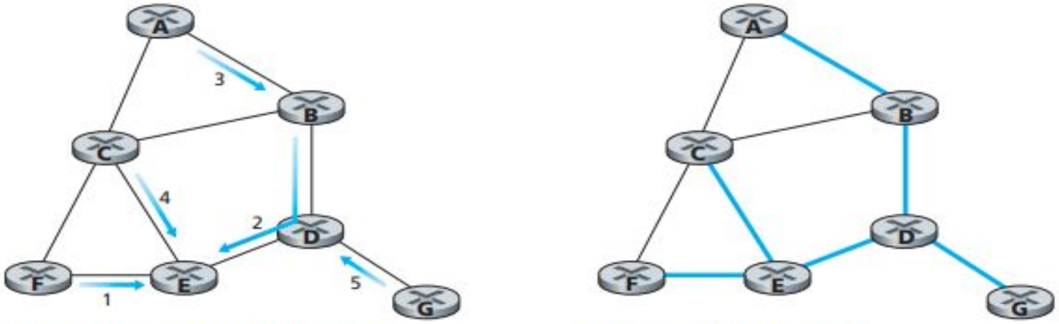
- When a router receives a broadcast packet with a given source address, it transmits the packet on all of its outgoing links (except the one on which it was received) only if the packet arrived on the link that is on its own shortest unicast path back to the source. Otherwise, the router simply discards the incoming packet without forwarding it on any of its outgoing links.
- RPF need only know the next neighbor on its unicast shortest path to the sender; it uses this neighbor's identity only to determine whether or not to flood a received broadcast packet.



- Suppose that the links drawn with thick lines represent the least-cost paths from the receivers to the source (A).
- Node A initially broadcasts a source-A packet to nodes C and B. Node B will forward the source-A packet it has received from A (since A is on its least-cost path to A) to both C and D.
- B will ignore (drop, without forwarding) any source-A packets it receives from any other nodes (for example, from routers C or D).
- Let us now consider node C, which will receive a source-A packet directly from A as well as from B.
- Since B is not on C's own shortest path back to A, C will ignore any source-A packets it receives from B. On the other hand, when C receives a source-A packet directly from A, it will forward the packet to nodes B, E, and F.

C. Spanning-Tree Broadcast:

- While sequence-number-controlled flooding and RPF avoid broadcast storms, they do not completely avoid the transmission of redundant broadcast packets.Ideally, every node should receive only one copy of the broadcast packet.
- This approach first constructs a spanning tree. When a source node wants to send a broadcast packet, it sends the packet out on all of the incident links that belong to the spanning tree. A node receiving a broadcast packet then forwards the packet to all its neighbors in the spanning tree (except the neighbor from which it received the packet).



a. Stepwise construction of spanning tree

b. Constructed spanning tree

- Not only does spanning tree eliminate redundant broadcast packets, but once in place, the spanning tree can be used by any node to begin a broadcast, as shown in the above given figures (a) and (b). Note that a node need not be aware of the entire tree; it simply needs to know which of its neighbors in G are spanning-tree neighbors.

Explain the working of Multicast routing algorithms

- In multicast service, a multicast packet is delivered to only a subset of network nodes.
- A multicast packet is addressed using address indirection. That is, a single identifier is used for the group of receivers, and a copy of the packet that is addressed to the group using this single identifier is delivered to all of the multicast receivers associated with that group.
- In the Internet, the single identifier that represents a group of receivers is a class D multicast IP address. The group of receivers associated with a class D address is referred to as a multicast group.

Two approaches have been adopted for determining the multicast routing tree,

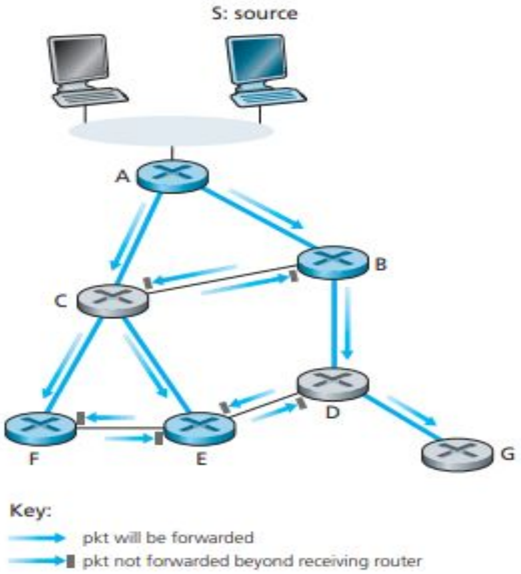
1. Multicast routing using a group-shared tree
2. Multicast routing using a source-based tree

1. Multicast routing using a group-shared tree:

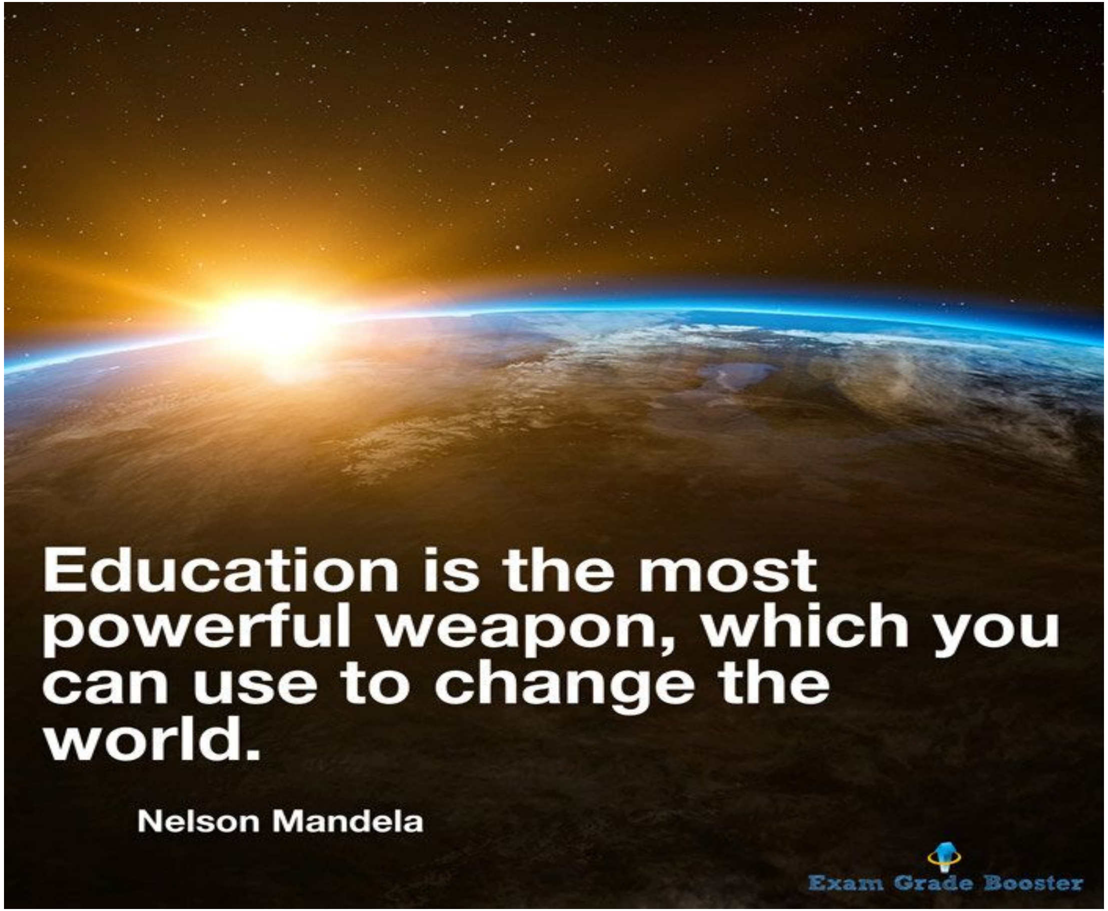
- Multicast routing over a group-shared tree is based on building a tree that includes all edge routers with attached hosts belonging to the multicast group.
- In practice, a center-based approach is used to construct the multicast routing tree, with edge routers with attached hosts belonging to the multicast group sending(via unicast) join messages addressed to the center node.
- A join message is forwarded using unicast routing toward the center until it either arrives at a router that already belongs to the multicast tree or arrives at the center.
- All routers along the path that the join message follows will then forward received multicast packets to the edge router that initiated the multicast join.

2. Multicast routing using a source-based tree:

- Approach constructs a multicast routing tree for each source in the multicast group.
- In practice, an RPF algorithm (with source node x) is used to construct a multicast forwarding tree for multicast datagrams originating at source x.



- In order to avoid unnecessary forward of the group messages to the routers which does not have any group member connected to it, this method makes use of pruning concept.
- A multicast router that receives multicast packets and has no attached hosts joined to that group will send a prune message to its upstream router. If a router receives prune messages from each of its downstream routers, then it can forward a prune message upstream.



Education is the most powerful weapon, which you can use to change the world.

Nelson Mandela

Exam Grade Booster