Study Material By Manikrao Dhore

Sub: Computer Networks

Section II: Transport Layer

CO5: Analyze data flow between peer to peer in an IP network using Application, Transport and Network Layer Protocols

Unit-V Transport Layer
[CO5 → PO1, PO2, PO4, PO5, PSO02 – CO Strength -3,3,2,3,3]

Transport Layer: Services: Berkeley Sockets, Addressing, Connection Establishment, Connection Release, Flow control and Buffering, Multiplexing. HTH Layer Protocols: TCP, TCP, TCP Timer management, UDP. Quality of Service: TCP Congestion Control. Traffic Shaping: AIMD. Real Time Support Protocols: Real Time Transport protocol(RTP), Stream Control Transmission Protocol (SCTP), Quality of Service (QoS), Differentiated services, TCP and UDP for Wireless [4 Hrs]

Need of Transport Layer: Because Network Layer

Is the part of communication subnet and run by WAN carrier?

Unreliable

Connectionless

Packet loss may happen

Data mangling may happen

Router may crash

Transport Layer:

Ultimate goal is to provide efficient reliable and cost effective service.

To enhance the QoS provided by network layer.

The Transport Layer Provides:

Both connection-oriented and connection less services

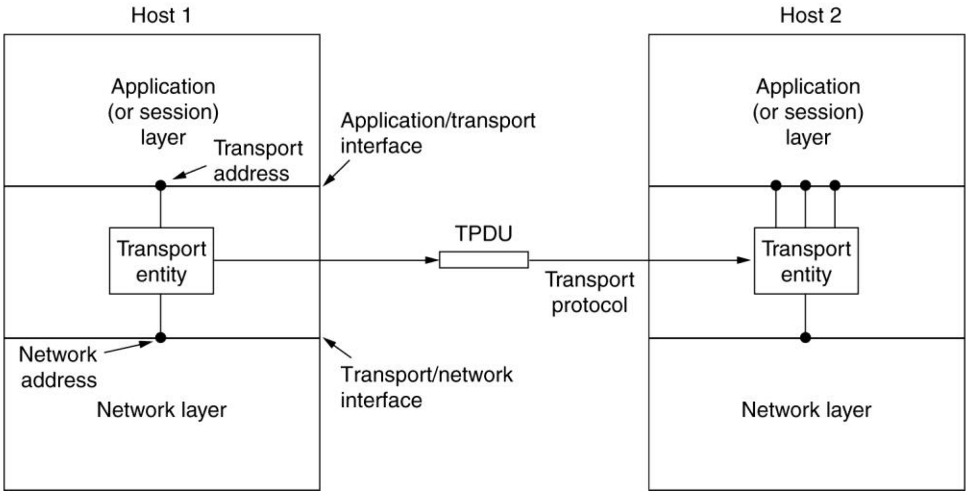
Addressing

Flow control

Congestion control

As well as QoS using various timers

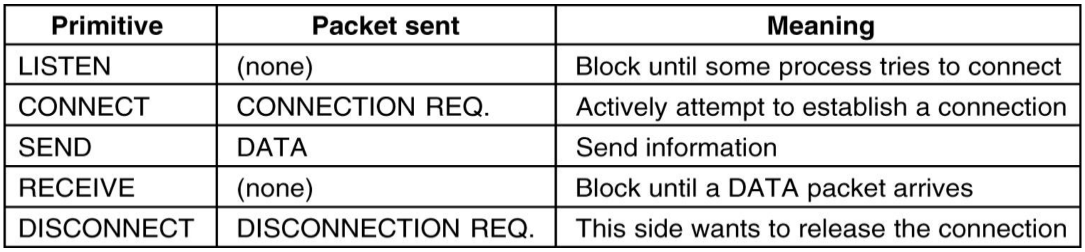
Services provided to Upper Layer and lower layer



Transport Service primitives: To allow user to access the transport services.

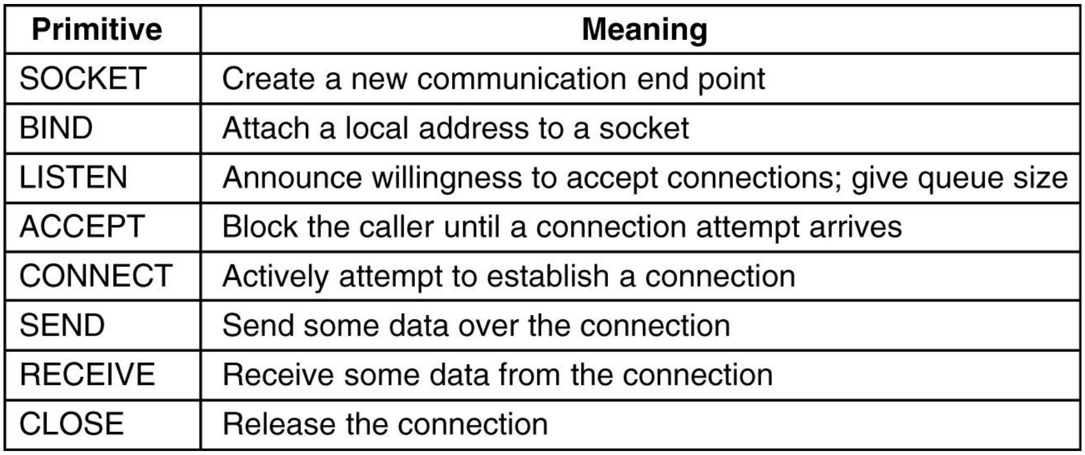
Transport layer service provides some operations to application program, it is called transport service interface

Basic Primitives:



Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

Berkeley Sockets



Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Programming Steps (General)

Step I: Create/Open a socket. Use socket() command

Syntax : socket(int af, int type, int protocol)

where af is address family for TCP/IP af =PF_INET and others af = AF_INET

type = SOCK_STREAM or SOCK_DGRAM or SOCK_RAW

protocol = 0 for TCP/IP

Step II: Name the socket: Attach attribute to the socket like protocol, port, address

Use struct sockaddr_in

Struct sockaddr_in serv

serv.sin_family=AF_INET;

serv.sin_addr.s_addr=inet_addr("10.0.0.111");

serv.sin_port=htons(13);

Step-III: Associate with another socket

bind(int socket, struct sockaddr, int namelen));

Client Side :

Use connect() command to establish connection with remote machine

Connect(int server_socket,(struct sockaddr *)&serversocket, int len(serversocket))

Server :

Use listen() command to check for incoming connection from remote machine

And then use accept() commands for accepting connection

```
listen(int Server_Socket,int  backlogConn);
ServerSocket = accept(int Server_Socket, int Remote_Socket, int *addresslen);
```

Step IV: Use send() or sendto() commands for sending data.

```
send(server_socket,sendbuff,strlen(sendbuff),0);
int sendto(sockfd,buff,buffsize,0,(struct sockaddr *) &clientaddr, int clientaddr_len)
```

Use receive() or receiveto() for receiving data

```
int recv(server_socket,recv_buff,strlen(recv_buff),0);
int recvfrom(listenfd,buff,buffsize,0,(struct sockaddr *) &clientaddr,
int *sockaddress_len)
```

Step V : use close() command to disconnect the connection

Close(socket)

Bare-bones Superstructure for TCP Client Server Socket Programming

Client	Server
Socket Creation: socket()	Socket Creation: socket()
Initialize sockaddr_in structure with **remote** socket name	Initialize sockaddr_in structure with **local** socket name
bind()	bind()
	Listen()
connect() ----------------------→	
	accept()
send() -------------------------→	recv()
<Association created, either side can send or receive>	
recv() ←-----------------------	send()
closesocket()	closesocket() (Connected Socket)
	closesocket() (Listening socket)

Bare-bones Superstructure for UCP Client Server Socket Programming

Set The Remote Socket Name Once

Client	Server

Socket Creation: socket()	Socket Creation: socket()
Initialize sockaddr_in structure with **remote** socket name	Initialize sockaddr_in structure with **local** socket name
bind()	bind()
connect() ----------------------→	Listen()
sendto() ------------------------→	recvfrom()
<Association created, either side can send or receive>	
recvfrom() ←-----------------------	sendto()
closesocket()	closesocket() (Connected Socket)

Bare-bones Superstructure for UCP Client Server Socket Programming

Set The Remote Socket Name For Each Datagram

Client	Server
Socket Creation: socket()	Socket Creation: socket()
Initialize sockaddr_in structure with **remote** socket name	Initialize sockaddr_in structure with **local** socket name
	bind()
sendto() ------------------------→	recvfrom()
recvfrom() ←-----------------------	sendto()
closesocket()	closesocket()

Byte Ordering

Host Byte Order :Little Endian (Intel)   0007 MSB to LSB

Network Byte Order : Big Endian (Motorola) 7000 MSB to LSB

inet_addr("10.0.1.111")- converts IP into 32 bit address

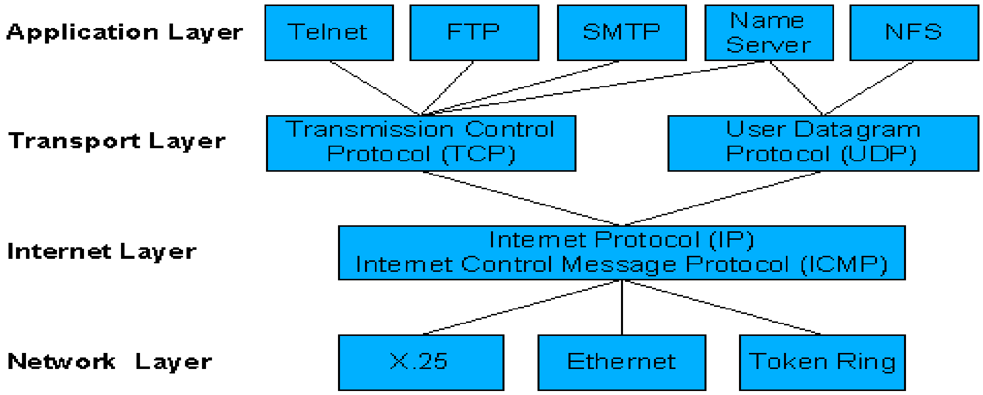
inet_ntoa(32-bit IP address) – convert IP address into std dotted notation

htons(10001): convert host byte order to network byte order hton(s) – s is unsigned short integer

htonl(l) – l stands for unsigned long integer

Htonl and ntohl and inet_addr(IP address)

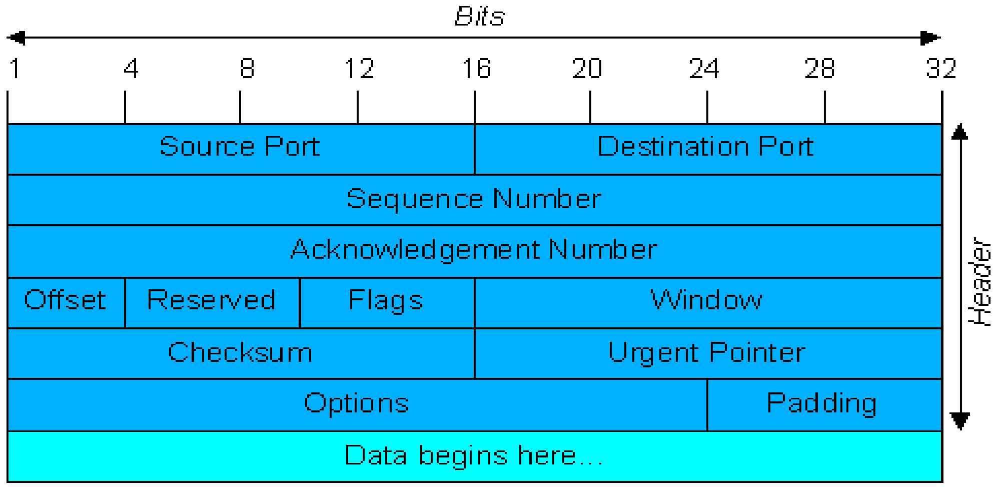
TCP/IP Network Components



TCP

- Port Numbers
- 32-bit Sequence Number
- 32-bit Ack Number
- Flags : URG, ACK, PSH, RST, SYN, FIN
- Urgent Pointers
- Connection Oriented
- Reliable
- Full Duplex
- ID Assignment to multiple virtual circuits
- Works with IP

- Maintain the sessions



TCP Frame Format

Source Port			Destination Port
Sequence Number			
Ack			
Header Length	Reserved	Flags	Window Size
TCP checksum			Urgent Ponter
Data			

Flags : UGR, ACK,PSH, RST, SYN, FIN

Functions of TCP

1. Reliable, Full Duplex and connection oriented
2. Works with IP to move the packets
3. Assign connection id to each virtual circuit
4. Provides message fragmentation and reassembly using sequence number
5. Error checking is enhanced through the use of TCP acks
6. Error detection and correction

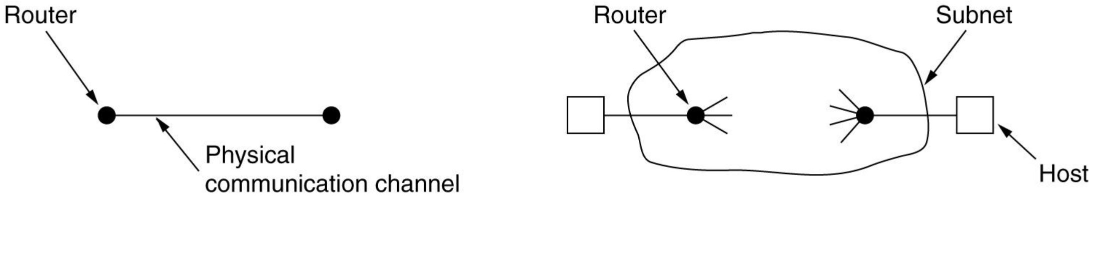
7. Does not support broadcasting and multicasting
8. Byte stream boundaries are not preserved end to end
9. Min segment size is 556 bytes
10. Provides flow control using sliding windows protocol

Functions of DLL and TL?

Sequencing, Flow Control and Error Control.

Why DLL and TL has the same functions?

Parameter	DLL	TL
Routing	Communicate directly through physical channel Uniquely identifies next router. Explicit addressing is not required	Communicate through subnet Explicit addressing of destination router is required. Connection establishment is required
	Less storage capacity is required	Potential capacity is required
	Can handle less connection	Handle large number of connections with dynamic buffer mgt.



(a) (b)

Elements of Transport Protocols

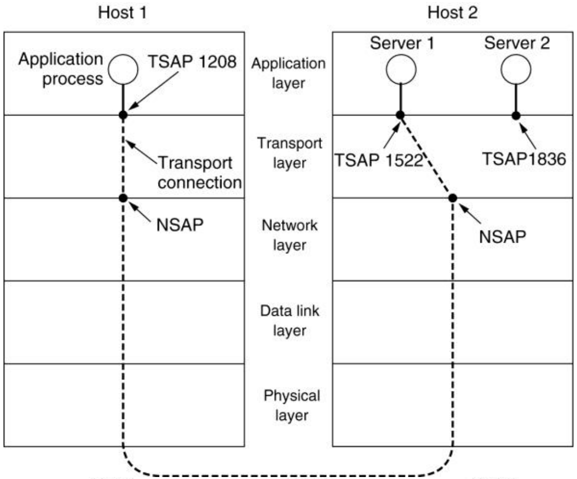
- Addressing
- Connection Establishment
- Connection Release
- Flow Control and Buffering
- Multiplexing

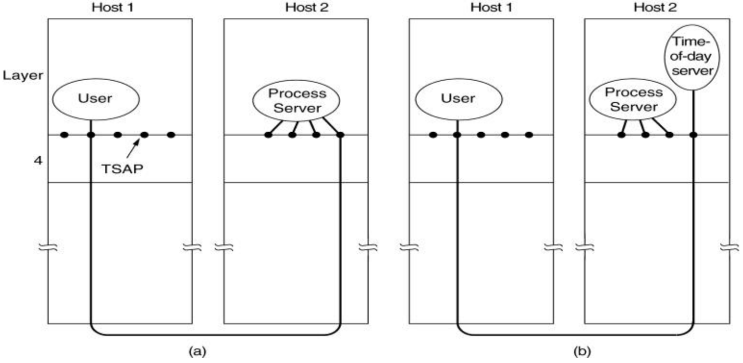
- Crash Recovery

Addressing : How client application knows on which port service is running on the server?

Standard ports : /etc/services

Inetd/process server/name server/directory services





Connection Establishment

Sounds easy but surprisingly tricky

Bank money transfer: delayed packet issue, time out problem

(Problem is solved in Sliding window problem)

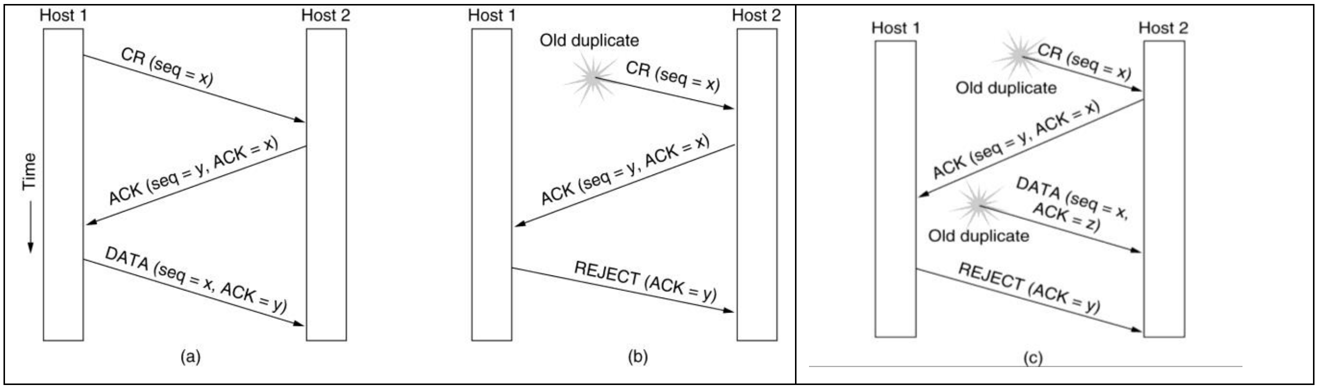
Kill off the aged packets that are hobbling around

Sol:

Restricted subnet design: prevent packet from looping by putting boundary delay

Hop counter: hop count will be decremented and reaches to zero

Time stamping: Requires time synchronization



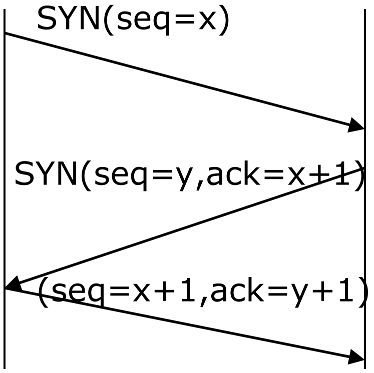
Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST.

(a) Normal operation,

(b) Old CONNECTION REQUEST appearing out of nowhere.

(c) Duplicate CONNECTION REQUEST and duplicate ACK.

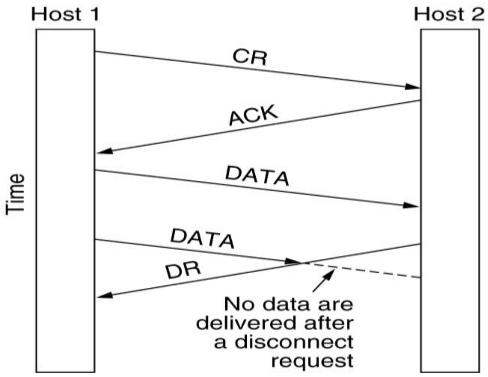
Three way handshake by Tomlinson-1975



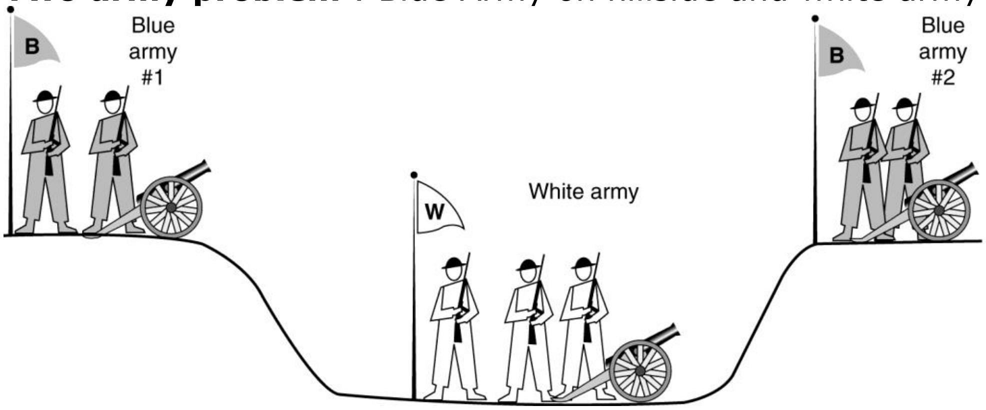
Connection Release

Asymmetric release - telephone

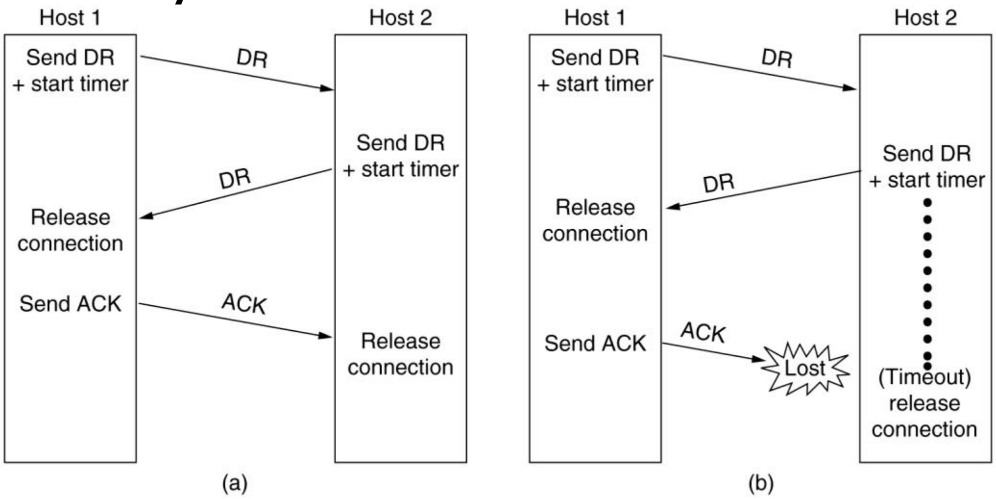
Symmetric release – tcp, pair of two simplex connection FIN



Two army problem : Blue Army on hillside and white army in valley



Three Way connection release



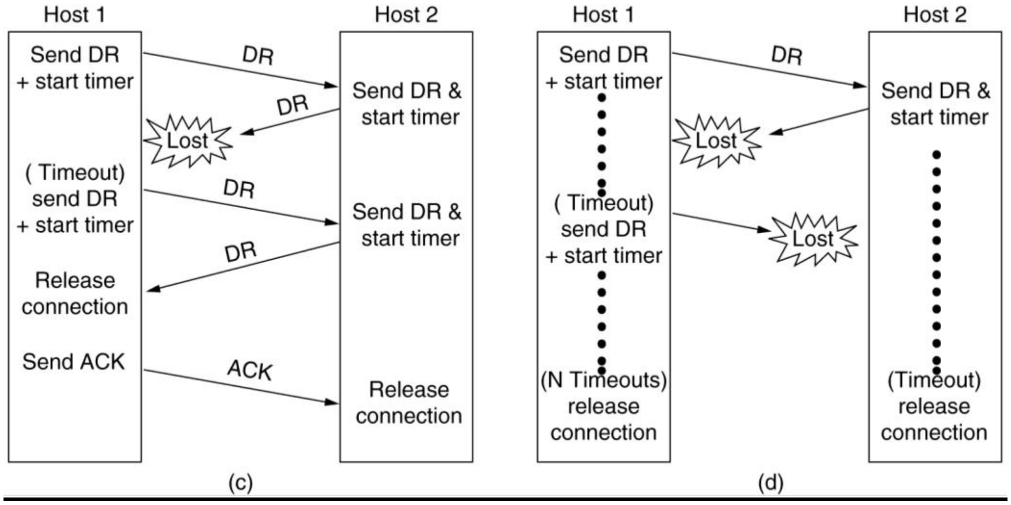


Fig. Four protocol scenarios for releasing a connection. (a) Normal case of a three-way handshake. (b) final ACK lost. (c) Response lost. (d) Response lost and subsequent DRs lost.

Multiplexing

Upward multiplexing and downward multiplexing

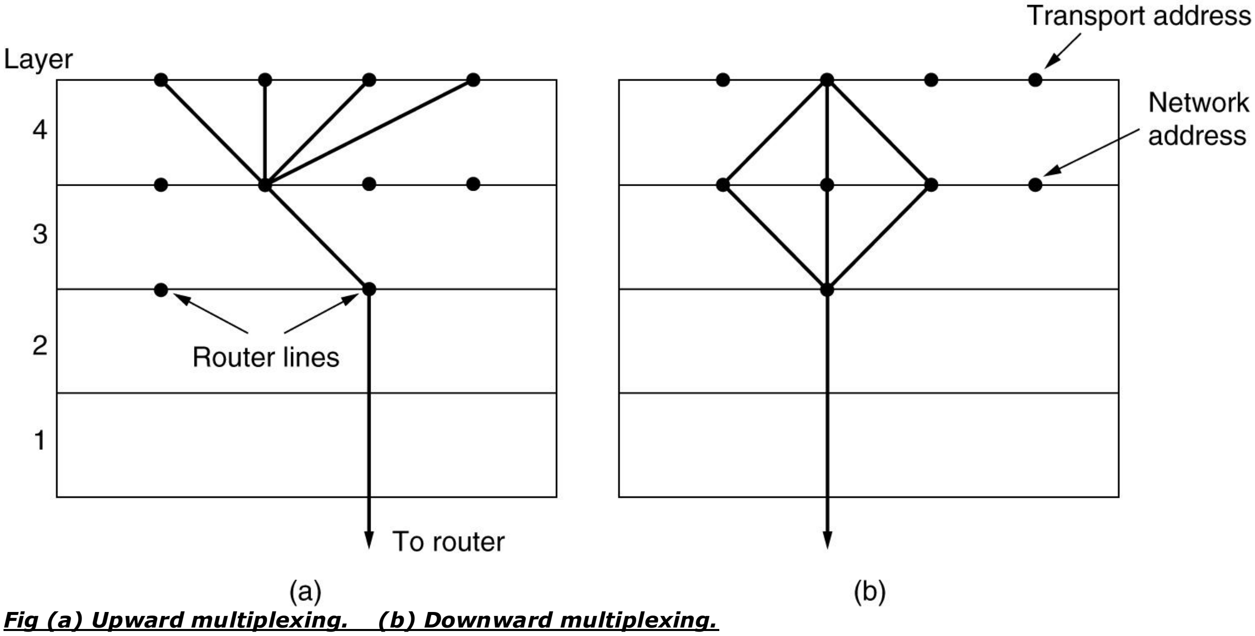


Fig (a) Upward multiplexing. (b) Downward multiplexing.

Crash Recovery

Loss of VC is handed by handling new connection

If server is crashed announce by broadcast and ask client to tell the status. Or Server must have logger

Flow control and Buffering

Use a sliding window protocol to have a fast transmitter to accommodate overrunning a slow receiver

Host server may have multiple connections available but what about the number of lines available with a router?

Buffering

In DLL sender and receiver knows the buffer available and accordingly buffers the packets at both the side

Static/Fixed buffer size: But frame may have variable size
Maximum TPDU Size: Wastage of memory
Variable size TPDU: better utilization but complicated buffer mgt required

Application Oriented Buffering

For interactive application : Low b/w : sender can buffer
High b/w : receiver can buffer

Buffer space must be related with the carrying capacity of the subnet. Sender must dynamically adjusts the window size to match the n/w carrying capacity

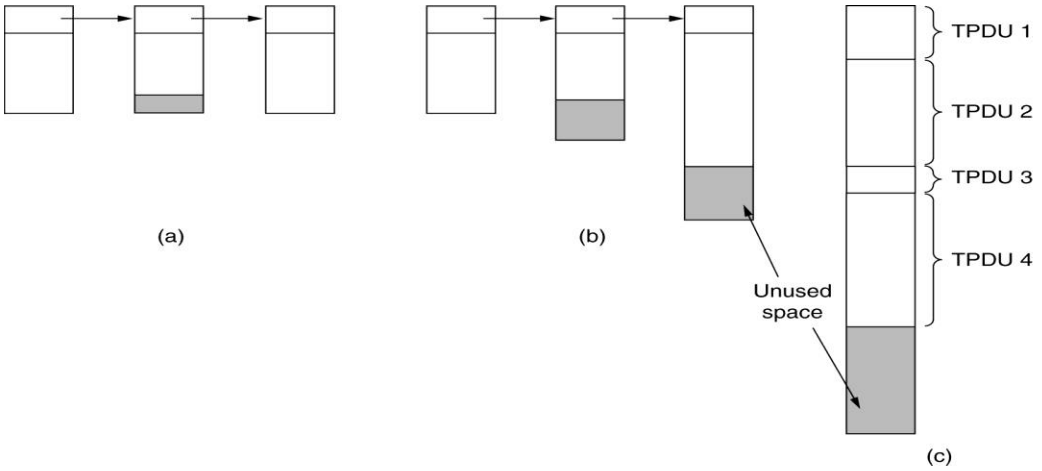
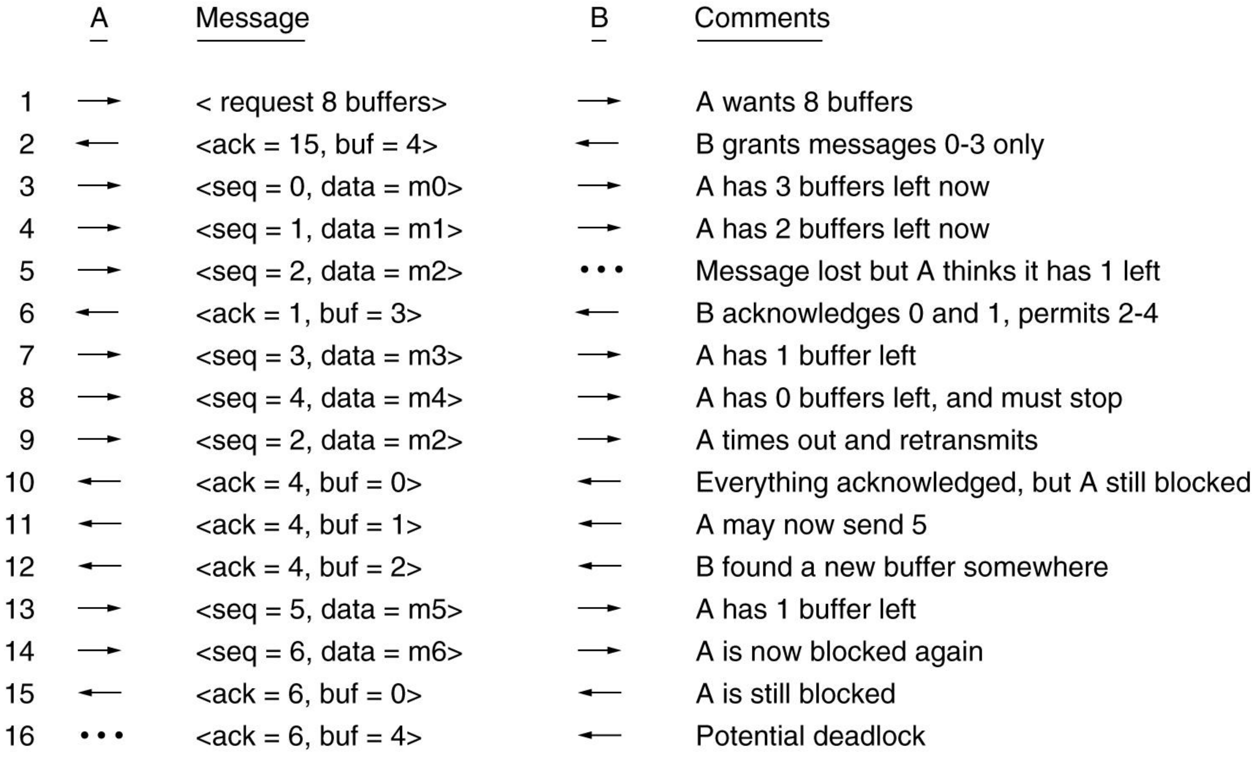


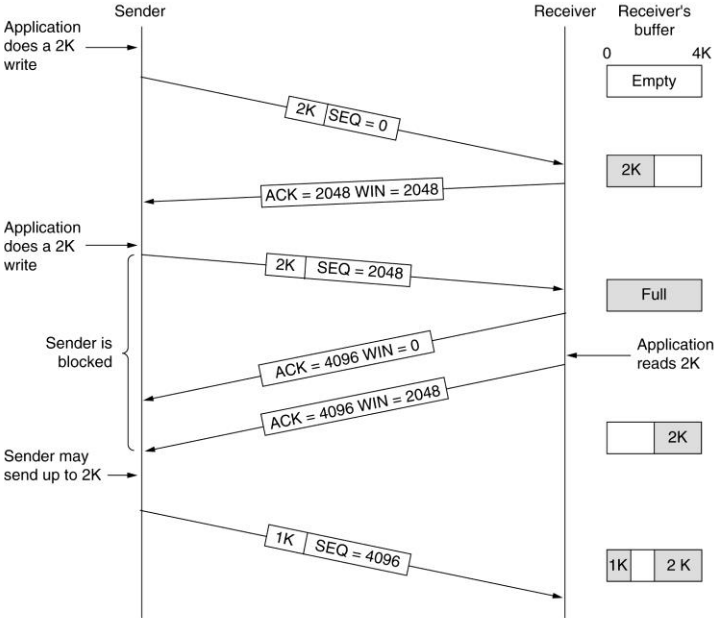
Fig (a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per connection.



	A	Message	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	• • •	<ack = 6, buf = 4>	←	Potential deadlock

Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (…) indicates a lost TPDU.

TCP Transmission Policy



Silly Window Syndrome : Whenever one of the application program creates data very slowly then it reduces the efficiency of data transfer operation. Sometimes TCP sends only 1 byte of data, this problem is called Silly Window Syndrome.

Nagle's Theorem : Never send a data without ACK is received. Buffer the data and when ACK is received then only send the data.

Clarks Solution for fast sender: Sending zero window size till buffer becomes available (either half or full)

Delayed Ack problem : Never send ACK to sender till enough buffer is available

Karn Algorithm : Double the time is ack is not received

TCP Timer Mgt

Retransmission Timer: For the lost or discarded segment this timer handles the retransmission time, the waiting time for ACK.

Normally it is 2 x RTT

Persistence Timer: Zero Window Mgt

Keep alive Timer: Used to prevent long idle /silent connection. Send the probe packet after 2 hours and terminates the connection.

Time Waited Timer: It is normal TCP connection termination after FIN is received. After the ACK sent for ACK , connection is closed after 2 packet lifetime

TCP Congestion Control

Retransmission is not the solution

Actual Window Size = min(receiver WS , congestion WS)

Congestion Avoidance :

Slow Start: Start with 1 segment, get ack for first segment then send 2 segment, get ack, send 3 seg etc

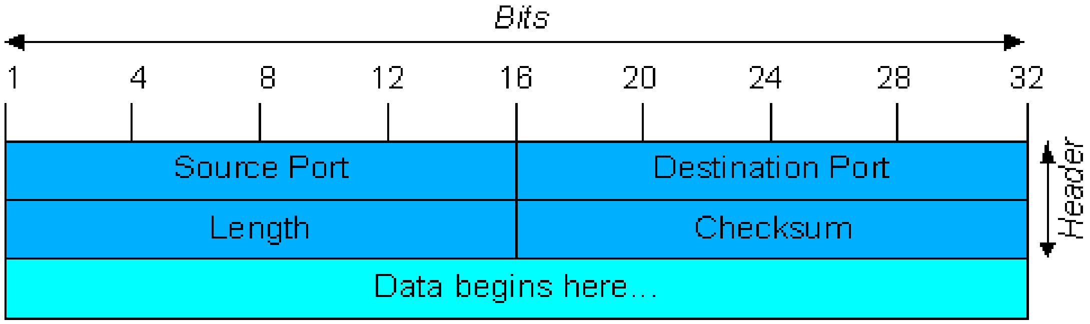
Additive Increase : Increase exponentially upto some threshold and then increase by one

Multiplicative Decrease: If congestion occurs reduce the flow exponentially.

UDP

- Provides a *lossy* connection (data may vanish).

- Does not guarantee packets are delivered in order.
- Useful for real time applications.
- UDP applications can implement their own packet loss checking but it is best to use TCP for this.



Connectionless , unreliable

No ack, No flow control , no error control

Less overhead hence more efficient

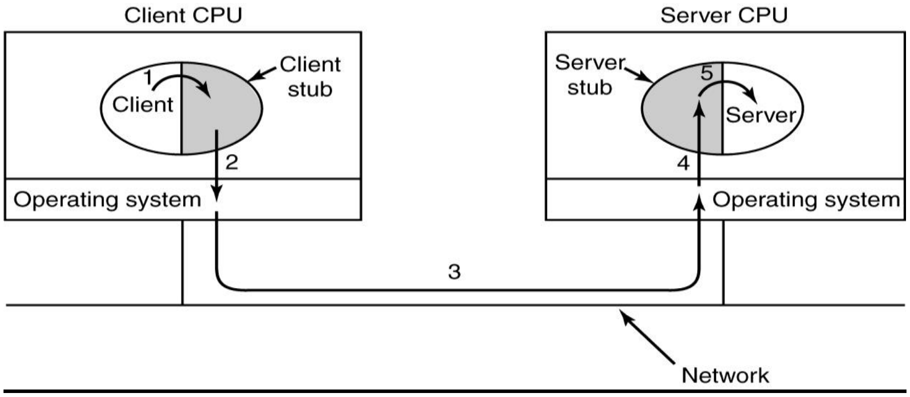
Used for simple request reply applications

Echo, daytime, quote of the day, dns, bootp, tftp, rpc, NTP, snmp, audio and video conferencing

Supports multicasting and broadcasting

Only 8 byte header

Remote Procedure Call


Client CPU
Server CPU
1
Client
Client stub
2
Operating system
Server stub
5
Server
4
Operating system
3
Network