Chapter-03: Transport Layer

3.2 Multiplexing and Demultiplexing

Extends network layers host to host delivery service to process to process delivery for appl'ns running in the host.

Transport layer receives the network segments

'Sockets' are the doors through which data passes from the network to the process and through which the data passes from the process to the network

∴ The transport layer in the receiving host does not actually deliver data directly to a process, but instead to intermediatory socket.

Each socket has a unique identifier

Each transport layer segment has a set of fields in it. And the " " examines these fields to identify the recieving socket & then direct the segment to that socket.

Job of delivering the data in a transport-layer segment to correct socket - Demultiplexing

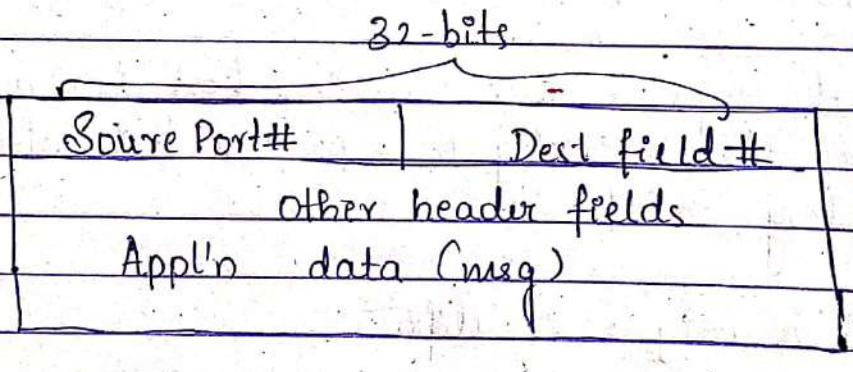
Job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header info. to create segments & passing segments to network layer is called multiplexing

Transport layer multiplexing requires

(1) Sockets have unique identifiers
(2) Each segment have special fields that indicate the socket to which the segment is to be delivered i.e. source port no field and destination port no field.

Port Number = 16 bit (0 - 65535)
Port No's from 0 to 1023 - well known port no's

HTTP : 80
FTP : 21



Network layer encapsulates the segment in an IP datagram

UDP socket is fully identified by two-tuple
(destination IP , destination port address)

TCP socket is fully identified by four-tuple
(source IP, source port , dest IP, dest port)

UDP over TCP:

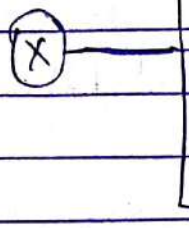
1) fines appl'n - level control over what data is sent and when.
2) No connection establishment
3) No connection state
4) Small packet header overload

UDP applns:

1) multimedia applns
2) DNS
3) RIP - routing table updates
4) SNMP - simple network management protocol

TCP appl'n:

1) E-mail
2) remote login
3) web
4) FTP



Subnet1 $60 = 2^6 = 64$

Subnet2 $90 = 2^7 = 128$

Subnet3 $12 = 2^4 = 16$

$208 = 2^8 = 256$

$32 - 8 = 24$ ↑ Given

Mask for s1 = 26 = 32 - 6

Mask for s2 = 32 - 7 = 25

Mask for s3 = 32 - 4 = 28

Prefix = 230.1.17/24.

Bigg ⟶ Small.

Subnet2 230.1.17.0/25

230.1.17.127/25

Subnet1 230.1.17.128/26

230.1.17.191/26

Subnet3 230.1.17.192/28

230.1.17.207/28

#230.1.17.131

TCP

→ Connection oriented
→ Connection establishment via threeway handshake
  ↳ this helps to set parameters that ensure reliable data transfer
  ↳ it resides in host & not in intermediate network elements
  ↳ segments exchanged helps in setting TCP variables
→ full duplex service
→ Connection is point-to-point (b/w single sender & reciever)
→ Multicasting is not possible
→ flow controlled, in-order byte stream, pipelined
→ Three-way handshaking
  (i) Process that initiates the connection - client process.
  (ii) " " is waiting for the " to be initiated - server process.
  (iii) Client process running in one host wants to initiate the connection with server process running in another host

  (iv) Client appln process informs its transport layer that it wants to establish a connection with server process

Connection establishment

(i) client sends TCP segment
(ii) Server " " "
(iii) client " " "

→ MSS is determined by length of largest link-layer frame that can be sent by the local sending (MTU)
→ MTU for ethernet & PPP link layer protocol have MSS of 1500 bytes

MSS- max segment size or max amt of appl'n layer data in segment

MTU - max transmission unit includes TCP header, when encapsulated as IP datagram

TCP three way handshake → Sender → data through socket → TCP connection sender buffer →
Sender: Data + Header → TCP segment

↓ Network layer

IP datagram

↓

Network

↓

Reciever

↓

extract segments

↓

place in TCP connections recieve buffer

↓

App

→ TCP views data as unstructured, but ordered stream of bytes

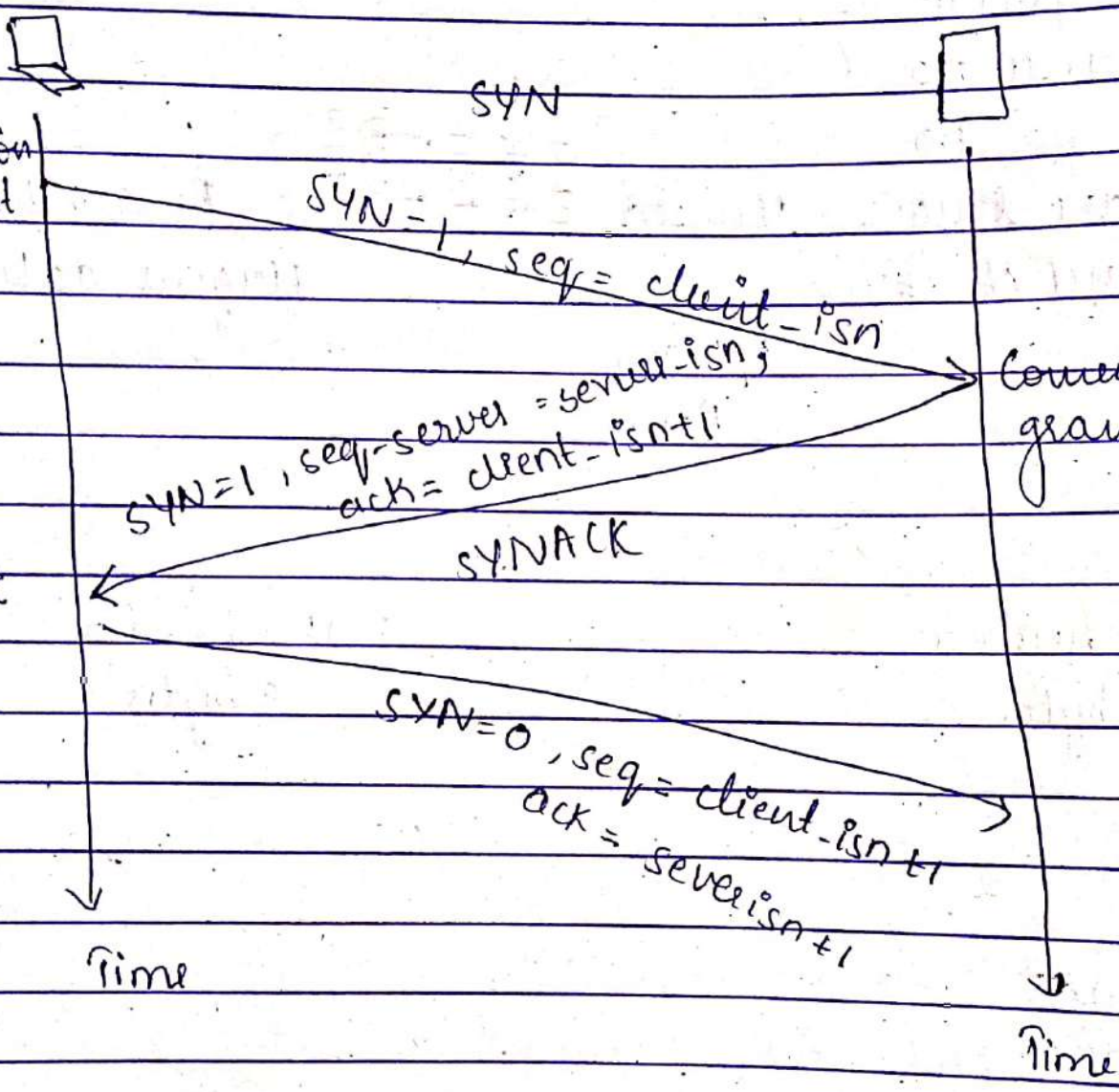
→ TCP provides cumulative acknowledgements

Flow Control

Last byte Rcvd. - Last byte Read $\leq$ Rcvbuffer

$\leq$ rwnd

rwnd = Rcvbuffer - [Last Byte Rcvd - Last byte Read]



[P1] UDP

* Gaming application is loss tolerant
* Data transfer is not so critical
* Delay due to connection establishment & retransmission is not favorable during gaming
* As a student in BVBCET is developing it UDP will be the best choice as it is more simpler

[P2] UDP header:

Source port # | Dest port #
Length | checksum
Data (payload)

TCP header:

Source port # | Dest port #
Sequence no
ACK no.
Header len | unused | SYN | FIN | URG | PSH | ACK | RST | Receive window
Internet Checksum | Urgent data ptr
Optional
Data

TCP not in UDP

Seq no and ACK no - for flow control
Receive window - flow control.
Urgent dat ptr - UDP can't handle urgent data
PSH, SYN, FIN, ACK - TCP connection establishment not in UDP
Optional / Padding - No options in UDP

[P3] 5 bit - sequence no represent 32

$\frac{100}{32} + 1$

0 - 31 (32 packets)
0 - 31 (32 packets) // 64 + 32
0 - 31 (32 ") // 96
97 98 99 100
0 1 2 3 4

[P4] 0045 DF 0000 58FE20
sour dest length checksum

Source = 69 (TFTP) Trivial file transfer
Dest = 56832 // not well known range
length (data + header) = 88
Checksum = 65056
Data = 88 - 8 = 80

dest known → client to server
client → server to client

IP addressing:-

Client requests ⟶ server ⟶ replies ⟶ Reaches router, main firewall, core switch of the LAN then it should be forwarded to proper client in LAN

If it's broadcasted then there will be huge waste of bandwidth coz only one client needs it & all other clients/hosts in LAN rejects it. So it has it to unicast ∴ yielding efficiency of network

IP allocation / network design speaks about selecting the particular host / single machine & delivering to it

Entire network is subdivided into subnets

The packet is first delivered to network & then host

class A - 2^{24} hosts
class B - 2^{16} "
class C - 2^8 "

Needs for classless addressing

* larger blocks will result in address wasting (CA)
* Fewer blocks " " " in-sufficient address
* less address left & more customers

What is SOL?

⇒ Sub-Netting: Dividing network to smaller chunks/ subnetworks with each subnets having it's own subnet address

If no sub-netting is done then only broadcast is the option and this always increases network traffic

Unicast always decreases network traffic to deliver packets efficiently.

2) Mask - to deliver the packet to exact subnet (mathmatical model.

3) Super-Netting.

Mask?

A subnet mask is a 32-bit address used to block/mask portion of the IP address to distinguish the network ID from host ID

Masking: A process that extracts the address of the physical network from IP address

→ Can be done whether we have subnet or not

→ Not have subnet: masking extract the network address from an IP address (returns core switch address) → broadcast

→ Have subnet: masking extract the subnet address from IP address (tells the exact host address)

Three-Levels Of Hierarchy:

Net Id | Host ID

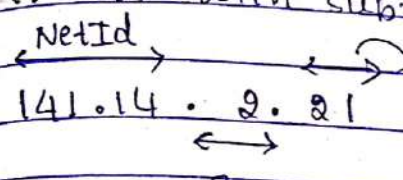
141 . 14 . 2 . 21

← Network → . ← Host →

Access | Access

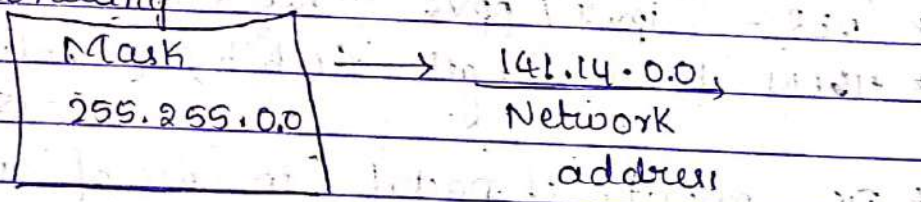
(a) without subnetting

∴ (b) With subnetting

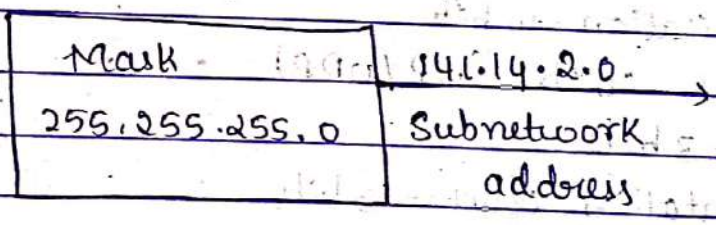


With subnetting the first portion of host id becomes subnet id

(a) Without subnetting



(b) With subnetting



Given an IP address, we can find the subnet address by applying the mask to the address.

Problem: IP address: 200.45.34.56
Subnet mask: 255.255.240.0

Subnetwork address: 200.45.32.0

1100 1000.0010 1101.0010 0010.0011 1000
1111 1111.1111 1111.1111 0000.0000 0000
1100 1000.0010 1101.0010 0000.0000 0000
200 . 45 . 32.0

In class C if subnet mask is 255.255.255.224, then calculate no. of subnet ID ?

→ 8

IP

→ 20 - 65536 bytes

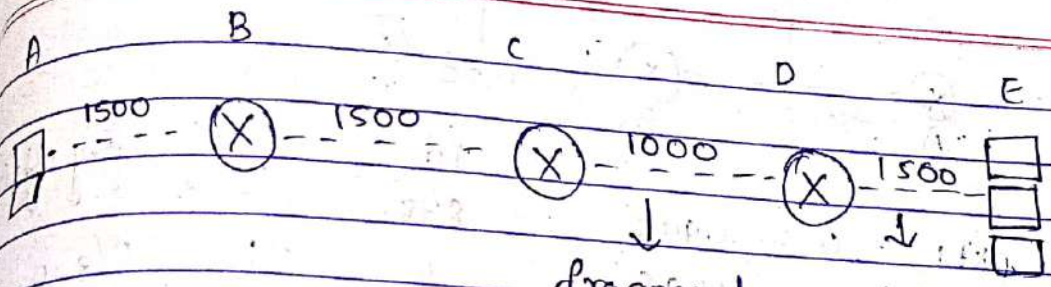
* Header 20-60 bytes
* Data - Actual data
* VER - ipv4 / ppv6 (only 4/6 valid) 4-bits
* HLEN - header length (4 bits) (value x 4) 5-15
* DS - priority of packet (in case of router congestion) (Default service) (8 bits) (max delay, bandwidth, throughput
* Total length - 16 bits (0-65536)

* Identification - 16 bits
* Protool - 8 bits: (TCP / UDP)
* Flags - 3 bits
* Fragmentation offset - 13 bits
* TTL - 8 bits
* Header checksum - 16 bits
* Source IP & dest IP - 32 bits
* Optional / Padding

IP fragmentation

* Dividing the info in chunks

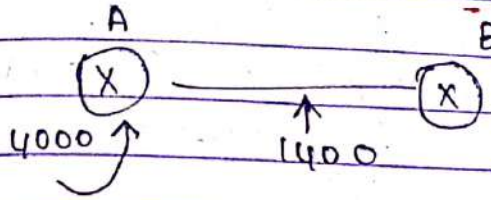
why ? Because on 13 bits is reserved for frag-offset
so only $2^{13} = 8192$ combins are possible



fragment : 1000, 500

defragment (while I have to join, is there any error)

Ex:



MTU → maximum transfer unit of link is 1400
↳ it tells how many bytes of data can be sent

4000

0 - 1399 = 1400 → Pkt1 0/8 = 0
1400 - 2799 = 1400 → Pkt2 1400/8 = 175
2800 - 3999 = 1200 → Pkt3 2800/8 = 350

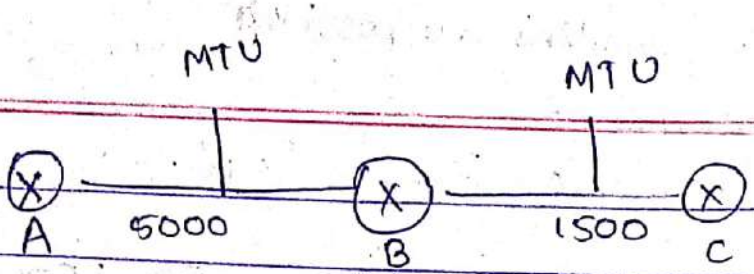
16	3	13
Id	flag	frag. offset

particular pkt -or- sequence no its fixed

flag → 1 if more pkts are there

why /8?
Because

777	1	000	pkt1
777	1	175	pkt2
777	0	350	pkt3



		Id	flag	frg offset
pkt1	0 – 4999	898	1	0/8 = 0
pkt2	0 – 1499	〃	1	1500/8 = 187
pkt3	1500 – 2999	〃	1	3000/8 = 375
pkt4	3000 – 4499	〃	1	4500/8 = 562
pkt5	4500 – 4999	〃	0	

64 ——— 128

64

1

$2^6 = 6$

Mask = 26

→ 192

$2^{32-27} = 32$

* Additive Increase Multiplicative decrease

→ Increase one by one until there is packet loss then reduce the congestion window by half.

$$\text{Throughput} = \frac{\text{No of packets} * \text{No of bytes in each packet} * 8 \text{ bits/byte}}{\text{RTT} * \text{no of RTT's.}}$$

* Slow Start

→ Exponential increase in terms of powers of 2
→ Packet drop occurs then again we start with $2^0 = 1$ congestion window size.
→ Again exponential increase until 1/2 of previous congestion threshold.
→ Then additive increase
→ Drop
→ Back to one exp increase till 1/2 the previous congestion then additive increase

Congestion threshold = $\frac{1}{2}$ of congestion window

RTT Seq#

$$\text{Throughput} = \frac{\text{packets\#} * \text{Size of each packet} * 8}{\text{RTT\#} * \text{Time of each RTT}}$$

Classful addressing

Class	Byte1	Type	NetId	HostId
A	0-127	End hosts	8	24
B	128-191	"	16	16
C	192-223	"	24	8
D	224-239	Multicast address	-	-
E	240-255	Research / future space	-	-

* Network ID = IP address & mask.
  → If mask is not given then check class of IP address and use default mask.

* Broadcast = 255.255.255.255 (limited)
  = Network ID . 255

* No of hosts = $2^h - 2$
  h = no. of zeroes in mask

* No. of subnets = 2^s
  s = | no. of ones in net mask - no. of ones in default mask |

Class	No of one
A	8
B	16
C	24

* Block size = 2^{32-n} , n = mask.
* first address = IP & mask
* last address = IP or inverted mask

Model paper

4a) Multiplexing & demultiplexing.

4b)

CIDR - classless Interdomain routing

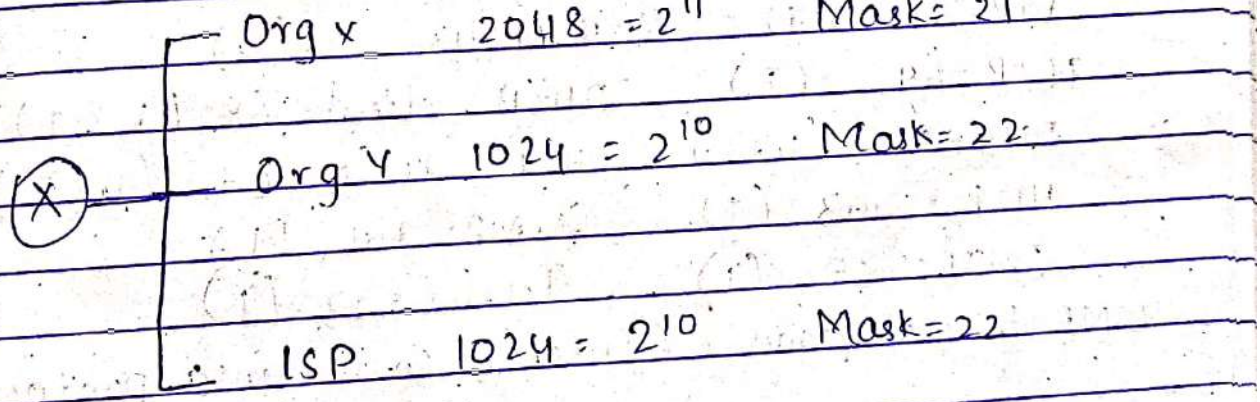
IP 245.248.128.0/20

Block size = $2^{32-20} = 2^{12} = 4096$.

50% to X = 2048

25% to Y = 1024

Remaining = 1024



Org X	Org Y	ISP
245.248.128.0	245.248.136.0	245.248.140.0
245.248.135.255	245.248.139.255	245.248.143.0

UDP format

Source port	Dest port	Length	Checksum
16	16	16	16

Actual data = length - header length
= length - 8

Well-known port is server always

Well-known ports 0-1023
Ephemeral ports 1024-65535 (Temporal ports)
↳ short lived

four well-known ports:

HTTP: 80 (T)	POP: 110 (U)
FTP: 21 (T)	IMAP: 143 (U)
TFTP: 69 (T)	DHCP client: 68 (U&T)
DNS: 53 (U&T)	DHCP server: 67 (U&T)
HTTPS: 443 (T)	SNMP: 161 (U)
SMTP: 25 (T)	Telnet: 23 (T)

ICMP General header (Internet Control msg protocol)

Type	Code	Checksum	Content
8	8	16	32

TCP header

Source# (16)	dest# (16)
Sequence no (32)	
ACK no (32)	
Header length (4) Unused (4) URG ACK PSH RST SYN FIN	Receive window (16)
Internet checksum (16)	Urgent data ptr (16)
Options	
Data	

Bitwise for prob's:

Source - 16
Dest - 16
Header length - 4 (value * 4)
Unused - 6
URG ACK PSH RST SYN FIN - 6
Receive window - 16
Internet checksum - 16
Urgent data pointer - 16 (last byte of urgent data)
Options
Actual data

IP datagram probs:
VER → 4 bits (4/6 only)
HLEN → 4 bits (value * 4) (20-60)
Service → 8 bits // Default Service
Total length → 16 bits
Identification → 16 bits
Flags & fragmentation → 16 bits → Flags (3), fra offset (13)
TTL → 8 bits
Protocol → 8 bits
Checksum → 16 bits
Source addr → 32 bits
Dest addr → 32 bits

5c) Block 130.56.0.0/16

No. of subnets = 1024

$2^{10} = 1024$

Block size = $2^{32-16} = 2^{16}$

Default is 16 bits for class B without subnetting
For subnetting we need extra 10 bits so
mask is /26

255.255.255.11000000

(a) → 255.255.255.192 // Subnet mask.

(b) No. of address in each subnet is nothing but no. of hosts

No. of hosts $= 2^h - 2$

$= 2^6 - 2 = 64 - 2 = 62$

(c) First & last in 1st subnet = 130.56.0.1

130.56.0.62

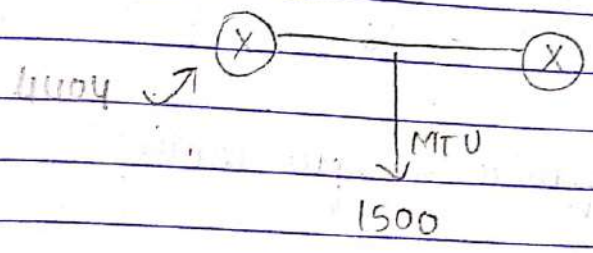
(d) First & last in last subnet = 130.56.255.193

130.56.255.254

6a) No. of fragments = $\left\lceil \frac{\text{Datagram} - \text{header}}{\text{MTU} - \text{header}} \right\rceil$ ceil

Bytes Id # (same for all) fragno. flag fragoffset (first byte / 8)

Bytes	ID	flag	frag offset
0-1479=[1480]	1	1	0
1480-2959	1	1	185
2960-4440	1	0	370



Theory

1) multiplexing & demultiplexing
2) ICMP ip4 , Congestion Control principles
3) Classfull addressing demerits
4) IP addressing - Differentiate classless & classfull
5) rdt FSM - rdt 1.0 rdt 2.0 rdt 3.0
6) Header formats - TCP, UDP, IP, ICMP
7) Pipelining - extra
8) 3 way handshake (timing diagram)
9) Compare virtual circuit network & datagram network
10) III dataplane & control plane

1) * Simple & easy to implement
* Connection establishment time is reduced
* Data integrity or security not needed
* It is loss tolerable appl'n

3) 5-bit sequence no = $2^5 = 32$

0-31 (32)

0-31 (64)

0-31 (96)

97	98	99	100
0	1	2	3

4 is the sequence no after 100th

4) 0045 DF00 0058 FE20

Source Dest length checksum

(a) 69 = Source

(b) 56832 = Dest

(c) 88 = length

(d) Actual data = 80

(e) Server to client

(f) Trivial file transfer protocol

5) UDP protects boundaries of a msg not TCP
It doesn't fragment & reassemble like TCP

6) Not the case

7) | URG | ACK | PSH | RST | SYN | FIN |

→ UAPRSF (Macro)

(i) all zeroes. The segment is part of data transmion without piggybacked acknowledgement

(ii) FIN set - FIN segment to request the termination of connection

(iii) ACK & FIN segment (iv) Request for resetting

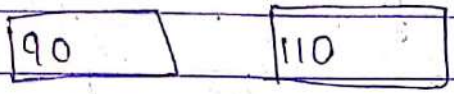
(v) SYN segment (vi) SYN + ACK segment.

9)

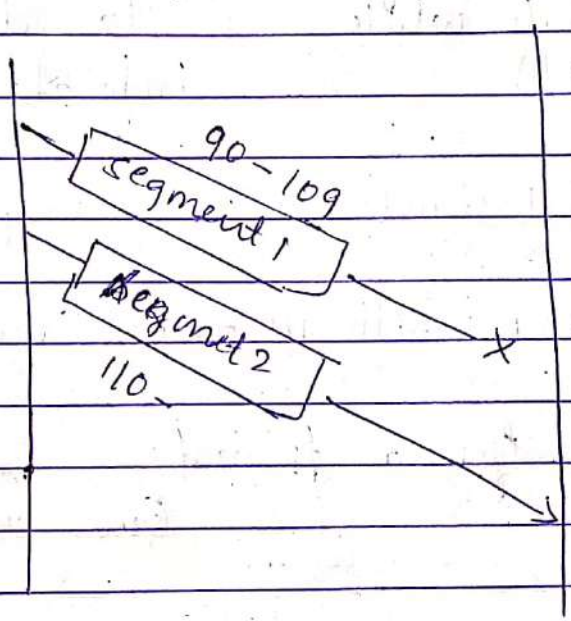
	UDP		TCP
App:	DNS	&	HTTP
	↓		↓
	to get the dest IP		to request data

10) rdt

11)



110 - 90 = 20 bytes



ACK = x, 90

12) Telnet port = 23 Client = Any random no above 1023 client A = 1467 client B = 1513

A → S	1467	23
B → S	1513	23
S → A	23	1467
S → B		1513

In IPv4 - no error correcting mechanism protocol
ICMP → corrects / reports error to sending device
↳ network diagnostics
↳ traceroute (routing path b/w s & d)
↳ ping command (connectivity)
↓ ↘ test connection speed
how long is RTT

Messages

ICMP msgs

↓ ↓

Error reporting
(report problems that any host or router encounters while processing pkt)

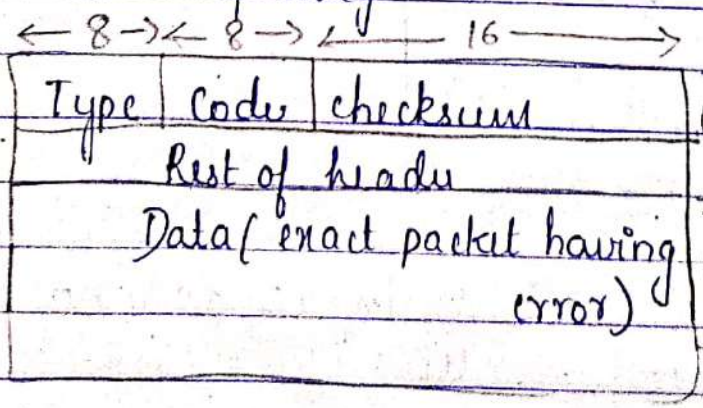
Query msgs
(help host / router to get specific info abt other host)

ICMP is not standalone it is companion of IP

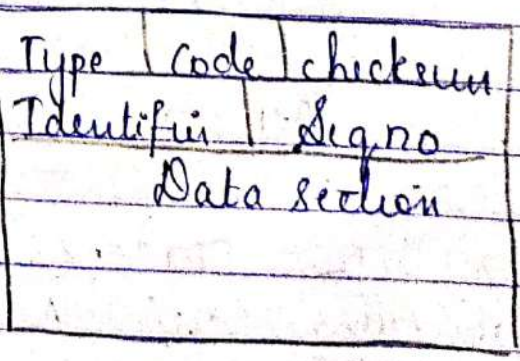
ICMP with IP protocol value is 01

General format

Error reporting



Query msgs



Error reporting msgs
03 - Destination unreachable
04 - Source Quench (reduce flow control)
05 - Redirection
11 - Time Exceeded
12 - Parameter problem.
Type code

Query msgs

08 - Echo request & before data is sent
00 - Echo reply
13 - Timestamp request
14 - Time

Checksum = Type + Code + Rest of header + Data selected

4) 4 - version
5 - hlen 5*4 = 20
00 - service
0054 - Total length
0003 - Id
0000 - flags & frag offset
20 - TTL
06 - protocol
0000 - checksum
7C4E0302 - source
B4 0E0F02 - dest
(a) No options
(b) No fragmentation
(c) Size of data = TL - H = 84 - 20 = 64
(d) No checksum
(e) TTL = 20 => 32d
(f) = 3
(g) No format service/ Routine

IP datagram

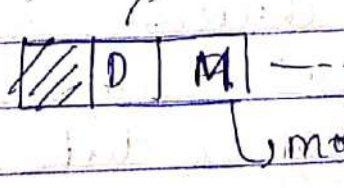
IP precedence values

Precedence value	Meaning
000	Routine or Best effort
001	Priority
010	Immediate
011	Flash
100	Flash override
101	critical
110	Internetwork network
111	Network Control

Types of Service values

TOS value	Meaning
0000 (0)	Normal delivery
0001 (1)	Minimize cost
0010 (2)	Maximize reliability
0100 (4)	" throughput
1000 (8)	Minimize delay

Fragmentation



ICANN Ranges

Well known - 0 - 1023 (assigned & controlled by ICANN)

Registered - 1024 - 49,151 (not " or " " " ")

Dynamic → 49,152 - 65,535 (nor " or registered)

Table 3-9 - IP Precedence Values

Precedence Value	Meaning
000 (0)	Routine or Best Effort
001 (1)	Priority
010 (2)	Immediate
011 (3)	Flash
100 (4)	Flash Override
101 (5)	Critical
110 (6)	Internetwork Control
111 (7)	Network Control

were originally designed to influence the delivery of data based on delay, throughput, cost. (See Table 3-10.) They are usually not used and are therefore set to zero.

Table 3-10 - Type of Service Values

ToS Value	Meaning
0000 (0)	Normal Delivery
0001 (1)	Minimize Cost
0010 (2)	Maximize Reliability
0100 (4)	Maximize Throughput
1000 (8)	Minimize Delay

Table 23.1 *Well-known ports used with UDP (continued)*

Port	*Protocol*	*Description*
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

Example 23.1

In UNIX, the well-known ports are stored in a file called /etc/services. Each line in this file gives the name of the server and the well-known port number. We can use the *grep* utility to extract the line corresponding to the desired application. The following shows the port for FTP. Note that FTP can use port 21 with either UDP or TCP.

```
$ grep    ftp   /etc/services
ftp         21/tcp
ftp         21/udp
```

SNMP uses two port numbers (161 and 162), each for a different purpose, as we will see in Chapter 28.

```
$ grep    snmp /etc/services
snmp        161/tcp        #Simple Net  Mgmt Proto
snmp        161/udp        #Simple Net  Mgmt Proto
snmptrap    162/udp        #Traps for SNMP
```

User Datagram

UDP packets, called **user datagrams**, have a fixed-size header of 8 bytes. Figure 23.9 shows the format of a user datagram.

The fields are as follows:

- **Source port number.** This is the port number used by the process running on the source host. It is 16 bits long, which means that the port number can range from 0 to 65,535. If the source host is the client (a client sending a request), the port number, in most cases, is an ephemeral port number requested by the process and chosen by the UDP software running on the source host. If the source host is the server (a server sending a response), the port number, in most cases, is a well-known port number.

Table 23.2 *Well-known ports used by TCP*

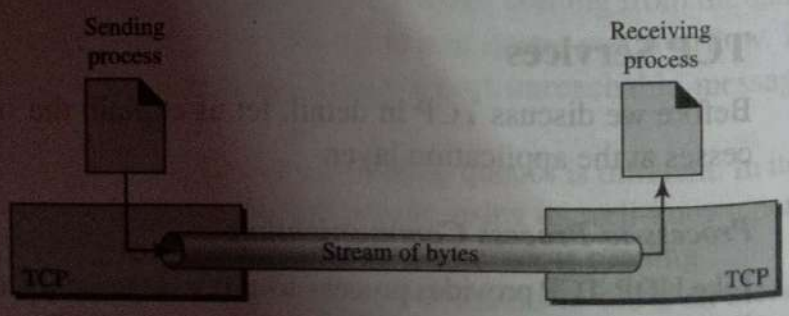
Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

Stream Delivery Service

TCP, unlike UDP, is a stream-oriented protocol. In UDP, a process (an application program) sends messages, with predefined boundaries, to UDP for delivery. UDP adds its own header to each of these messages and delivers them to IP for transmission. Each message from the process is called a user datagram and becomes, eventually, one IP datagram. Neither IP nor UDP recognizes any relationship between the datagrams.

TCP, on the other hand, allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their data across the Internet. This imaginary environment is depicted in Figure 23.13. The sending process produces (writes to) the stream of bytes, and the receiving process consumes (reads from) them.

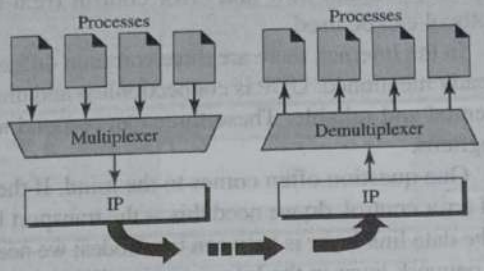
Figure 23.13 *Stream delivery*



Multiplexing and Demultiplexing

The addressing mechanism allows multiplexing and demultiplexing by the transport layer, as shown in Figure 23.6.

Figure 23.6 *Multiplexing and demultiplexing*



Multiplexing

At the sender site, there may be several processes that need to send packets. However, there is only one transport layer protocol at any time. This is a many-to-one relationship and requires multiplexing. The protocol accepts messages from different processes, differentiated by their assigned port numbers. After adding the header, the transport layer passes the packet to the network layer.

Demultiplexing

At the receiver site, the relationship is one-to-many and requires demultiplexing. The transport layer receives datagrams from the network layer. After error checking and dropping of the header, the transport layer delivers each message to the appropriate process based on the port number.

Connectionless Versus Connection-Oriented Service

A transport layer protocol can either be connectionless or connection-oriented.

Connectionless Service

In a **connectionless service,** the packets are sent from one party to another with no need for connection establishment or connection release. The packets are not numbered; they may be delayed or lost or may arrive out of sequence. There is no acknowledgment either. We will see shortly that one of the transport layer protocols in the Internet model, UDP, is connectionless.

Connection-Oriented Service

In a **connection-oriented service,** a connection is first established between the sender and the receiver. Data are transferred. At the end, the connection is released. We will see shortly that TCP and SCTP are connection-oriented protocols.

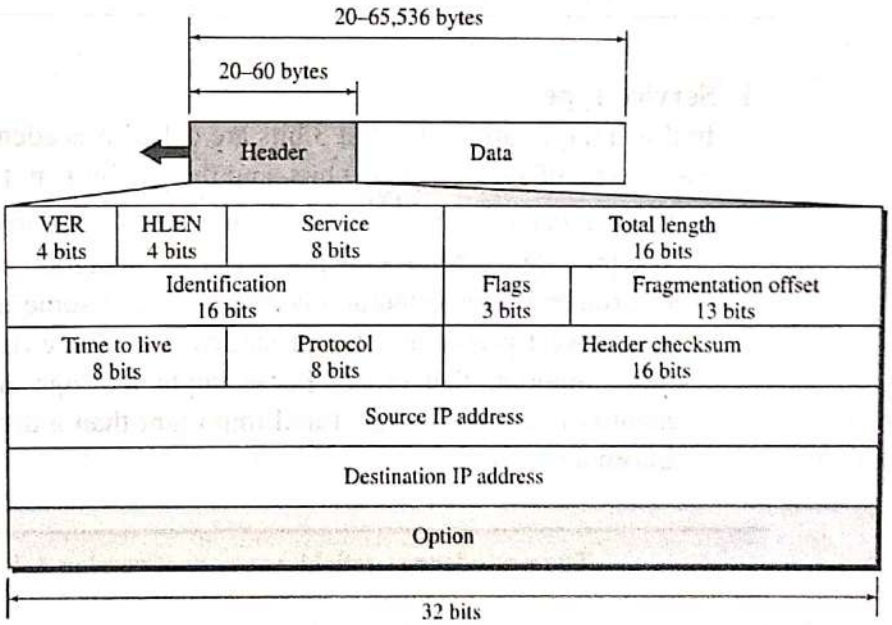
Classes and Blocks

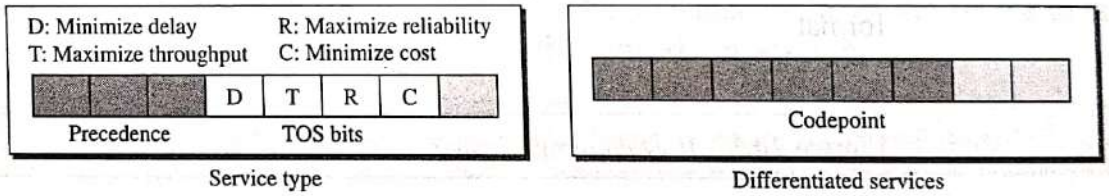
One problem with classful addressing is that each class is divided into a fixed number of blocks with each block having a fixed size as shown in Table 19.1.

Table 19.1 *Number of blocks and block size in classful IPv4 addressing*

Class	*Number of Blocks*	*Block Size*	*Application*
A	128	16,777,216	Unicast
B	16,384	65,536	Unicast
C	2,097,152	256	Unicast
D	1	268,435,456	Multicast
E	1	268,435,456	Reserved

Figure 20.5 *IPv4 datagram format*

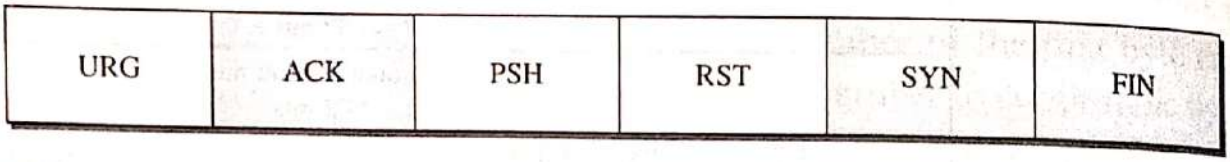




- **Flags.** This is a 3-bit field. The first bit is reserved. The second bit is called the *do not fragment* bit. If its value is 1, the machine must not fragment the datagram. If it cannot pass the datagram through any available physical network, it discards the datagram and sends an ICMP error message to the source host (see Chapter 21). If its value is 0, the datagram can be fragmented if necessary. The third bit is called the *more fragment* bit. If its value is 1, it means the datagram is not the last fragment; there are more fragments after this one. If its value is 0, it means this is the last or only fragment (see Figure 20.10).

Figure 20.10 *Flags used in fragmentation*

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

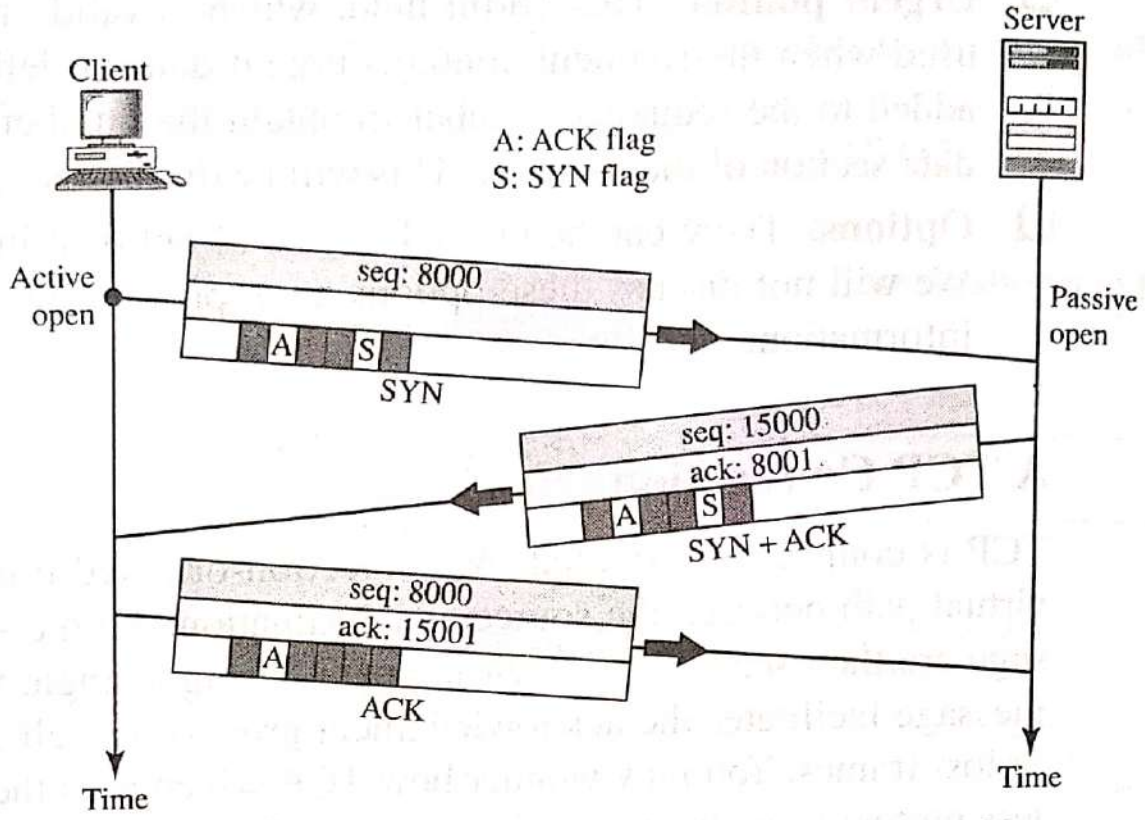


These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP. A brief description of each bit is shown in Table 23.3. We will discuss them further when we study the detailed operation of TCP later in the chapter.

Table 23.3 *Description of flags in the control field*

Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

Figure 23.18 *Connection establishment using three-way handshaking*



5c) Block 130.56.0.0/16

No. of subnets = 1024.

$2^{10} = 1024$

Block size = $2^{32-16} = 2^{16}$

Default is 16 bits for class B without subnetting
For subnetting we need extra 10 bits so
mask is /26

255.255.255.11000000

(a) → 255.255.255.192 // Subnet mask.

(b) No. of address in each subnet is nothing but no. of hosts

No. of hosts $= 2^h - 2$

$= 2^6 - 2 = 64 - 2 = 62$

(c) First & last in 1st subnet = 130.56.0.1

130.56.0.62

(d) First & last in last subnet = 130.56.255.193

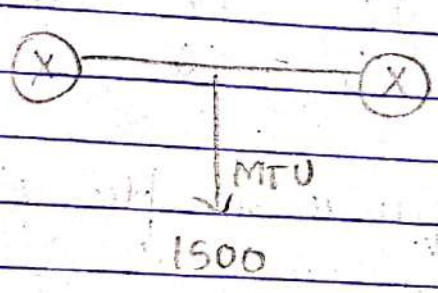
130.56.255.254

6a) No. of fragments = $\left\lceil \frac{\text{Datagram} - \text{header}}{\text{MTU} - \text{header}} \right\rceil$ ceil

Bytes Id # (same for all) fragno. flag frag offset (first byte / 8)

Various IP fields.

Bytes	ID	flag	frag offset
0-1479=[1480]	1	1	0
1480-2959	1	1	185
2960-4402	1	0	370



Theory

1) multiplexing & demultiplexing
2) ICMP ip4 , Congestion Control principles
3) Classfull addressing demerits
4) IP addressing . Differentiate classless & classfull
5) rdt FSM - rdt 1.0 rdt 2.0 rdt 3.0
6) Header formats - TCP, UDP, IP, ICMP
7) Pipelining - extra
8) 3 way handshake (timing diagram)
9) Compare virtual circuit network & datagram network
10) III dataplane & controlplane

1) * Simple & easy to implement
* Connection establishment time is reduced
* Data integrity or security not needed
* It is loss tolerable appl'n

3) 5-bit sequence no = $2^5 = 32$

0-31 (32)

0-31 (64)

0-31 (96)

97	98	99	100
0	1	2	3

4 is the sequence no after 100th

4) 0045 DF00 0058 FE20

Source Dest length checksum

(a) 69 = Source

(b) 56832 = Dest

(c) 88 = length

(d) Actual data = 80

(e) Server to client

(f) Trivial file transfer protocol

5) UDP protects boundaries of a msg not TCP
It doesn't fragment & reassemble like TCP

6) Not the case

7)

URG	ACK	PSH	RST	SYN	FIN

→ UAPRSF (Macro)

(i) all zeroes. The segment is part of data transmission without piggybacked acknowledgment

(ii) FIN set - FIN segment to request the termination of connection

(iii) ACK & FIN segment (iv) Request for resetting

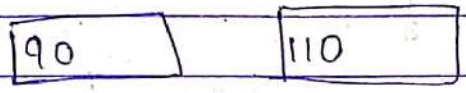
(v) SYN segment (vi) SYN + ACK segment

9)

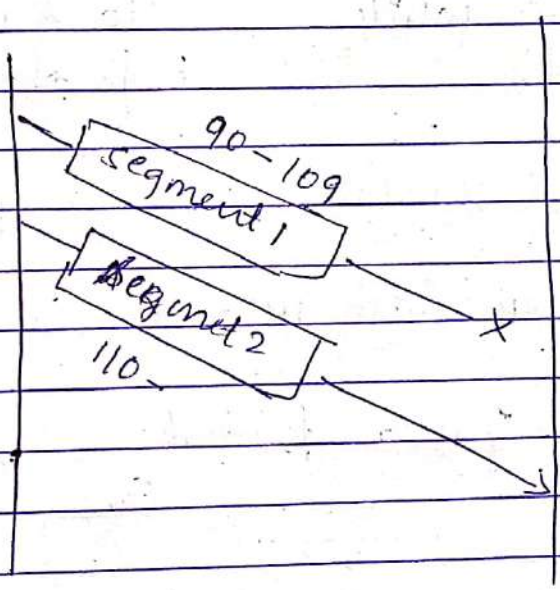
	UDP		TCP
App:	DNS	&	HTTP
	↓		↓
	to get the dest IP		to request data

10) rdt

11)



110 - 90 = 20 bytes



ACK = x, 90

12) Telnet port = 23 Client = Any random no

Above 1023 client A = 1467 client B = 1513

A → S	1467	23
B → S	1513	23
S → A	23	1467
S → B	23	1513