NP Unit 3,4

1) Explain lack of flow control with UDP.

```
#include "unp.h"
#define NDG 2000
#define DGLEN 1400
void dg_cli(FILE *fp, int sockfd, const SA *
        pservaddr, socklen_t, servlen)
{
  int i;
  char sendline[MAXLINE];
  for (i=0; i<NDG; i++) {
      sendto(sockfd, sendline, DGLEN, 0, pservaddr,
                servlen);
  }
}
```

~~dg_cli function that~~
dg_cli function that writes a fixed no. of datagram to server.

```
#include "unp.h"
static void recvfrom_int(int);
static int count;
void dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
{
  socklen_t len;
  char mesg[MAXLINE];
  signal(SIGHT, recvfrom_int);
  for(;;) {
    len = clilen;
    Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);
    count++;
  }
}
```

```
static void recvfrom_int(int signo)
{
    Printf("\n received %d datagram\n", count);
    exit(0);
}
```

* the interface's buffers were full or they could have been discarded by sending host
* The counter "dropped due to full socket buffers" indicates how many datagram were received by UDP but were discarded because the receiving socket's receiving queue is was full
* The number of datagrams received by the sender is this example is nondeterministic. It depends on many factors like network load, processing load on the client processing load on server.

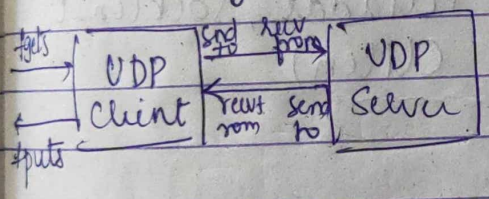
(2) Important functions of UDP echo server. stdin stout

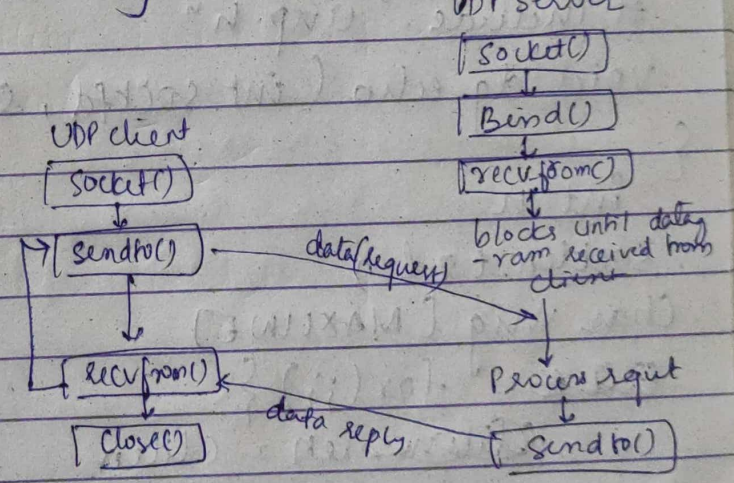
ans (i) main function (ii) dg-echo function.

```
#include "unp.h"
int main(int argc, char **argv)
{
    int sockfd;
    struct sockaddr_in servaddr, cliaddr;
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(SERV_PORT);
    bind(sockfd, (SA*) &servaddr, sizeof(servaddr));
    dg_echo(sockfd, (SA*) &cliaddr, sizeof(cliaddr));
}
```

```
#include "unp.h"
void dg_echo (int sockfd, SA * pcliaddr, socklen_t,
                                        clilen)
{
int n;
socklen_t len;
char msg [MAXLINE];
for(;;) {
    len = clilen
    n = Recvfrom (sockfd, msg, maxline, 0, Pcliaddr, &len);
    Sendto ( sockfd, msg, n, 0, Pcliaddr, len);
}
}
```

③ Steps involved in ~~echo~~ building an echo client-server appln using UDP





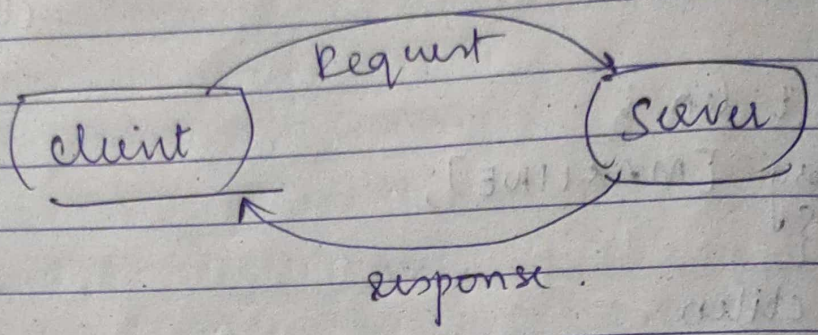
UDP client :-

* In UDP Echo ~~server~~ client a socket is created
* Then we bind socket
* After the binding is successful, we send msg ~~from~~ input from the user & display the data received from server using sendto() & recvfrom().

UDP Server :-

* In UDP Echo server, we create a socket & bind to a advertised port no.
* Then an infinity loop starts to process the client requests for connection

* The process receives client data using recvfrom() and echoes the data back using sendto()



④ The dg_echo function in C that is used in UDP echo Server appl^n

```
#include "unp.h"
Void dg_echo (int sockfd, SA * Pcliaddr, socklen_t,
                                        Clilen)
{
  int n
  Socklen_t * len
  Char msg [MAXLINE]
  for(;;) {
      len = Clilen
    n = recvfrom (sockfd, msg, maxline, 0, Pcliaddr, & len
      Sendto ( Sockfd, msg, n, 0, Pcliaddr, len)
}
```

10) Syslogd daemon. Indicate with code snippet, how to call the syslog function.

→ The common technique for logging message from a daemon is to call the syslog function.

Syntax

```
#include <syslog.h>
void syslog (int Priority, const char * message, ...);
```

Snippet

```
Static void
err_doit ( int errnoflag, int level, const char * fmt,
                                va_list ap)
{
    int errno_save, n;
    char buf [MAXLINE];
    errno_save = errno;
    Vsprintf (buf, fmt, ap); n = Strlen (buf);
    if (errnoflag)
        ~~snprintf (buf, fmt, ap);~~ |
        snprintf (buf+n, sizeof (buf)-n, ": %s" strerror
        Strcat (buf, "\n");                    (errno_save));

    if (daemon_proc) {
        syslog (level, buf);
    } else {
        fflush (stdout);
        fputs (buf, stdout);
        fflush (stdout);
    } return;
}
```

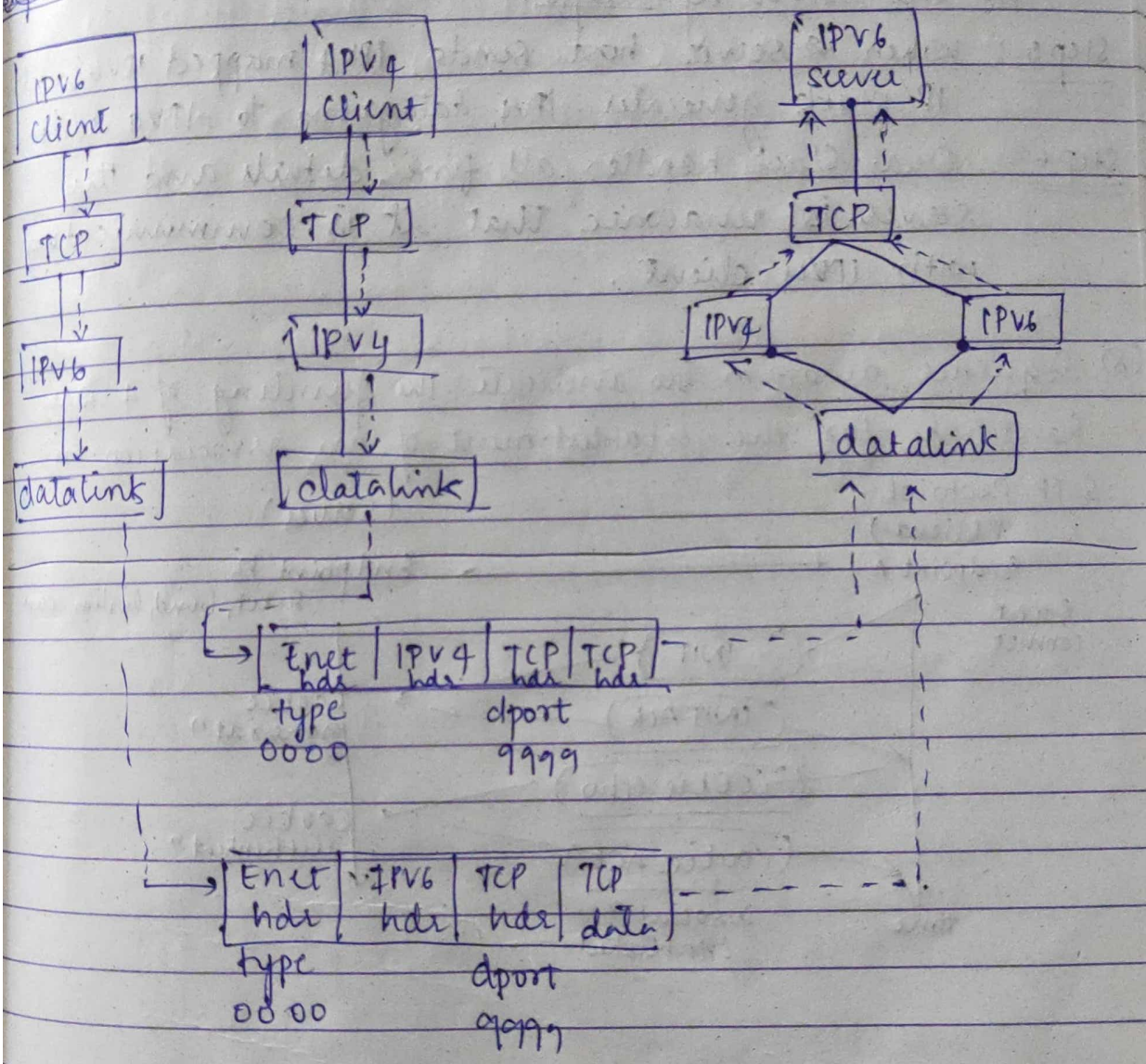
* Syslog stands for System logging Protocol and it is a standard Protocol used to send the system log or event msg to a specific server called a syslog server.

* Since a daemon does not have a controlling terminal, it needs some way to output the msg when something happens either normal infor. or emergency information that need to be handled by Administrator, & it sends the msg to syslog d daemon

2) Steps that allows an IPV4 TCP client to communicate with IPV6 server using dual stack.

Ans

Step 4:



Step 1: IPV6 Server starts, and creates a listening IPV6 socket & binds wildcard address to the socket.

Step 2: The IPV4 client calls gethostbyname & finds A record for the server. The server host have both A and AAAA record but IPV4 client as A record.

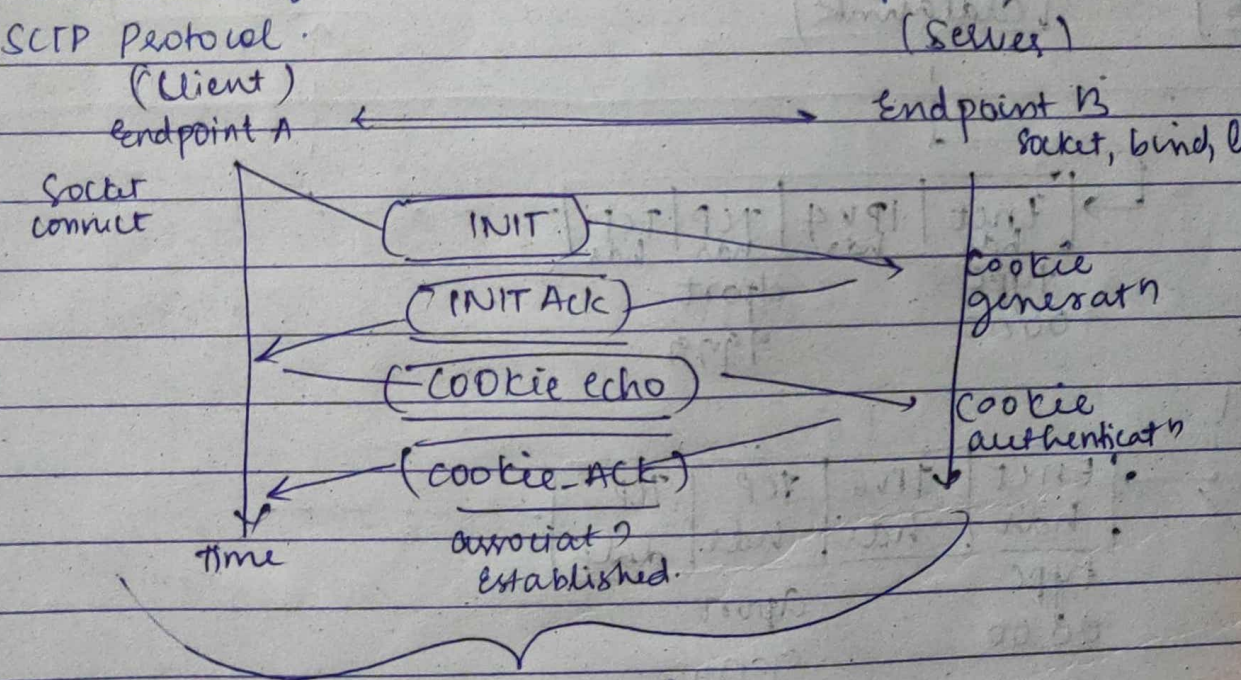
Step 3: The client calls connect and client's host sends IPV4 SYN to server.

Step4: The server host receive IPV4 SYN directed to IPV6 listening socket, sets a flag dividicating the connection is using IPV4 mapped IPV6 address & responds with IPV4 SYN/ACK.
When connection is established the address returned to the server by accept.

Step5: When server host sends IPV4 mapped IPV6 address IP stack generates IPV4 datagrams to IPV4 address.

Step6: Dual stack handles all finer details and the server is unaware that it is communicating with IPV4 client.

(8) Sequence diagram to indicate the functing of 4-way handchake for the establishment of an association in SCTP Protocol.



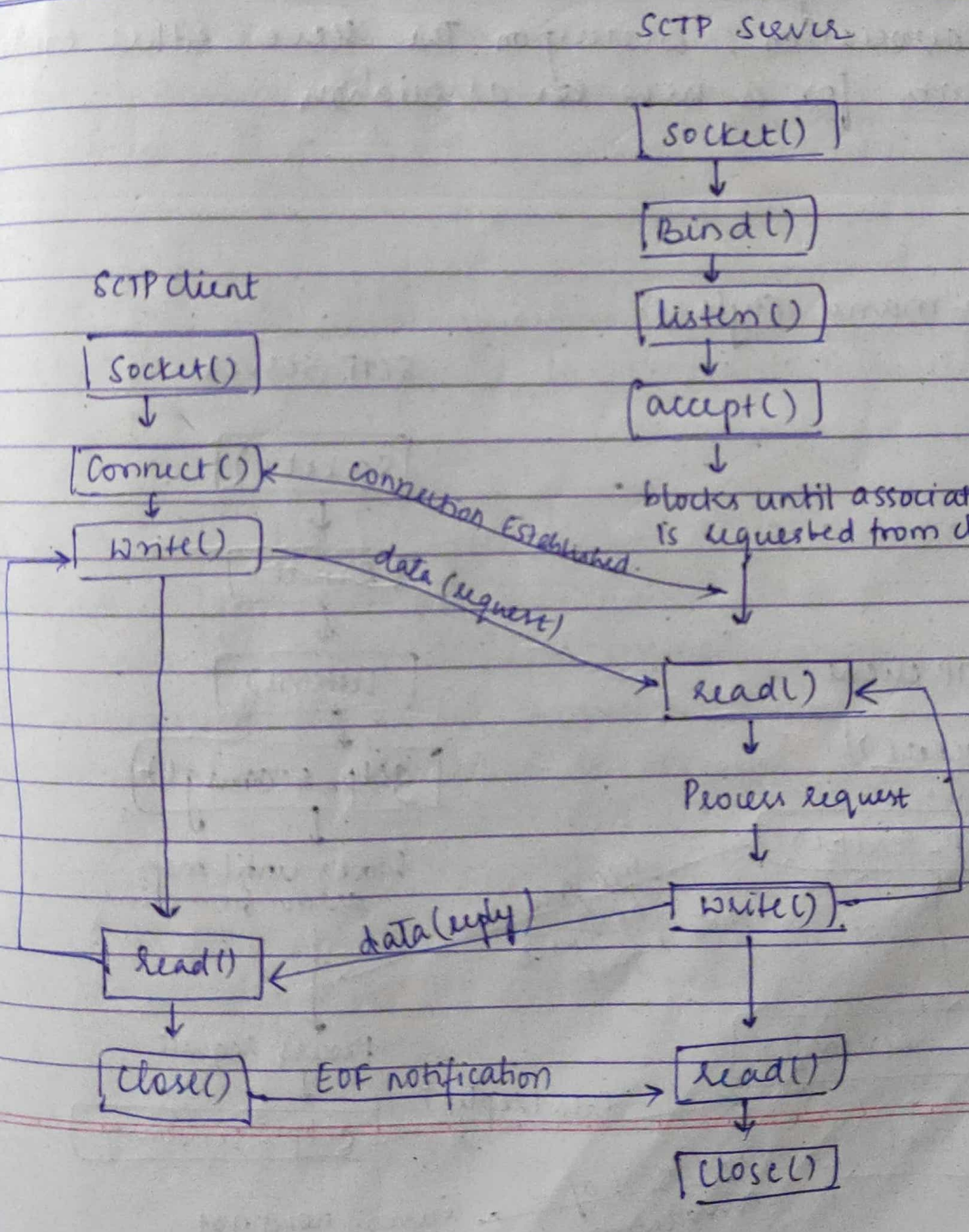
wrong

(4) Diffuent Interface models that are used in SCTP Protocol

Ans:

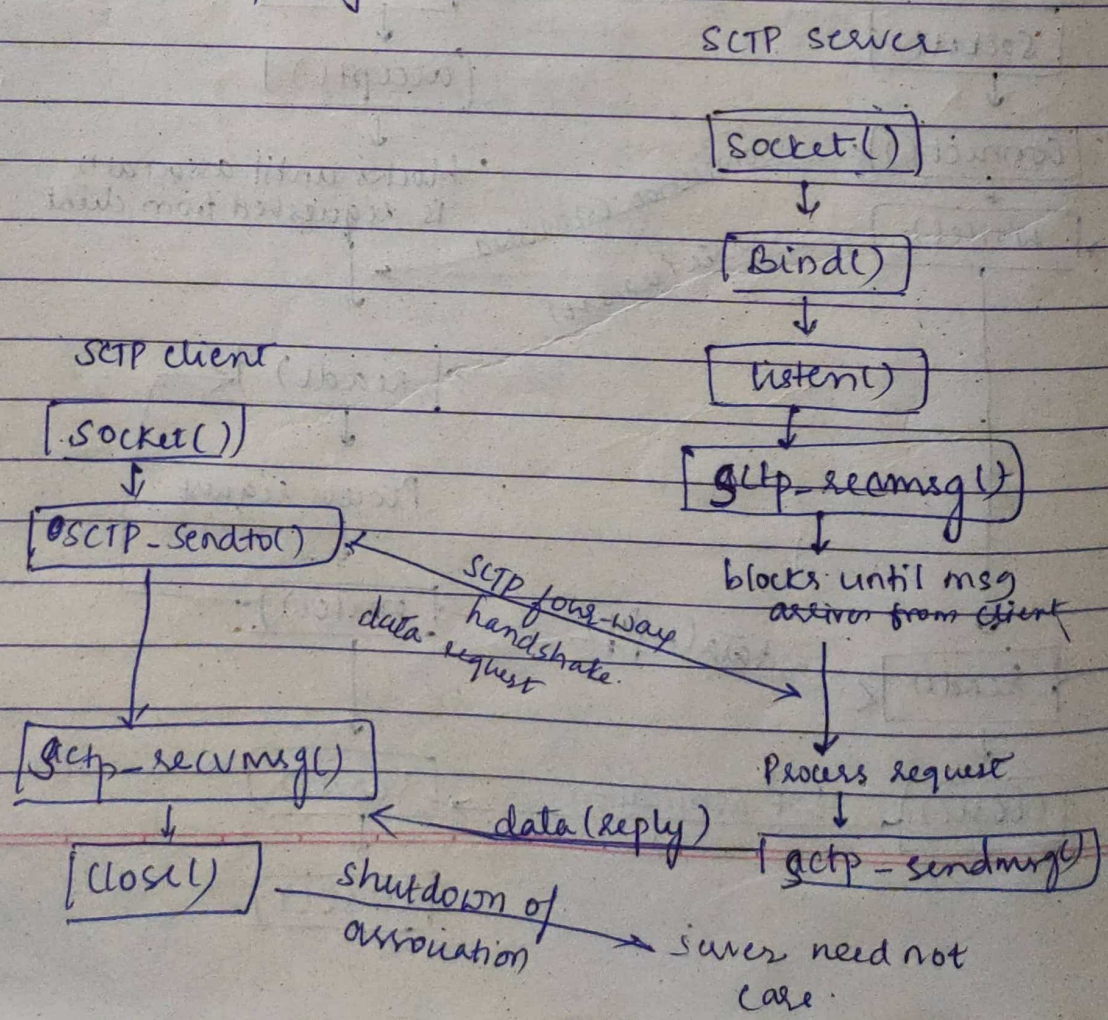
There are two types
- one-to-one
- one-to-many.

One-to-one:



* A typical user of the one-to-one style will follow timeline shown in fig.
* When the server is started, it open the socket, binds to an address & waits for client connection with accept system calls.
* Sometime later the client is started, it opens a socket & initiates an association with the server.
* We assume the client sends a request to server. The server process the request & server sends back a reply to client.
* This cycle continuous until the client initiates a shutdown of the associations. This action close the association, whereupon the server either exits or waits for a new association.

one-to-many style



* A typical one-to many style timeline is depicted:
* First, the server is started, creates a socket, binds to an address, calls listen to enable client associations, and calls sctp_recvmsg, which blocks waiting for the first mesg to arrive.
* A client opens a socket & calls sctp_sendto, which implicitly sets up the association & piggybacks the data request to the server on third packet of four-way handshake.
* The server receives the request, & processes & sends back the reply.
* The client receives the reply and closes the socket, thus closing the association.
* The server loops back to receive the next message.

(6) Identify the resulting changes with a connected UDP socket compared to default unconnected UDP socket?

Ans:

(1) We can no longer specify the destination IP address and port for an output operation. That is, we do not use sendto, but write or send instead. Anything written to a connected UDP socket is automatically sent to the protocol address specified by connect.

(2) Similar to TCP we can call sendto for a connected UDP socket, but we cannot specify a destination address.

(3) The fifth argument to sendto must be a null pointer & 6th argument should be 0.

The Posix specification states that when 5th argument is null pointer, the 6th argument is ~~ne~~ ignored.

(4) We do not need to use recvfrom to learn the sender of a datagram, but read, recv, or recvmsg instead.

(5) The only datagrams returned by the kernel for an input operation on a connected UDP socket are those arriving from the protocol address specified in connect.

(6) Technically, a connected UDP socket exchanges datagrams with only one IP address, because it is possible to connect to a multicast address.

(7) Asynchronous errors are returned to the process for connected UDP sockets.

NP Unit-5:

1) Explain Unicast, Multicast & Broadcast & anycast & differentiate them.

(or) 1) Explain scope of multicast address.

Ans:

→ IPV6 multicast address have an explicit 4-bit scope field that specifies how far the multicast packet will travel.

→ IPV6 packets also have a hop limit field that limits the number of times the packet is forwarded by routers.

→ The following values have been assigned to the scope field:

1 : Interface-local
2 : link-local
4 : admin-local
5 : site-local
8 : Organization-local
14 : global

→ The remaining values are unassigned or reserved. IPV6 multicast address that differ only in scope represent different groups.

→ The IPV4 doesnot have a seperate scope field for multicast packets.

→ The IPV4 TTL field in IP header are doubled as a multicast scope field.
The TTL of 0 means interface-local, 1 means link-local, up through 32 means site-local, up through 68 means region-local, up through 128 means continent-local.

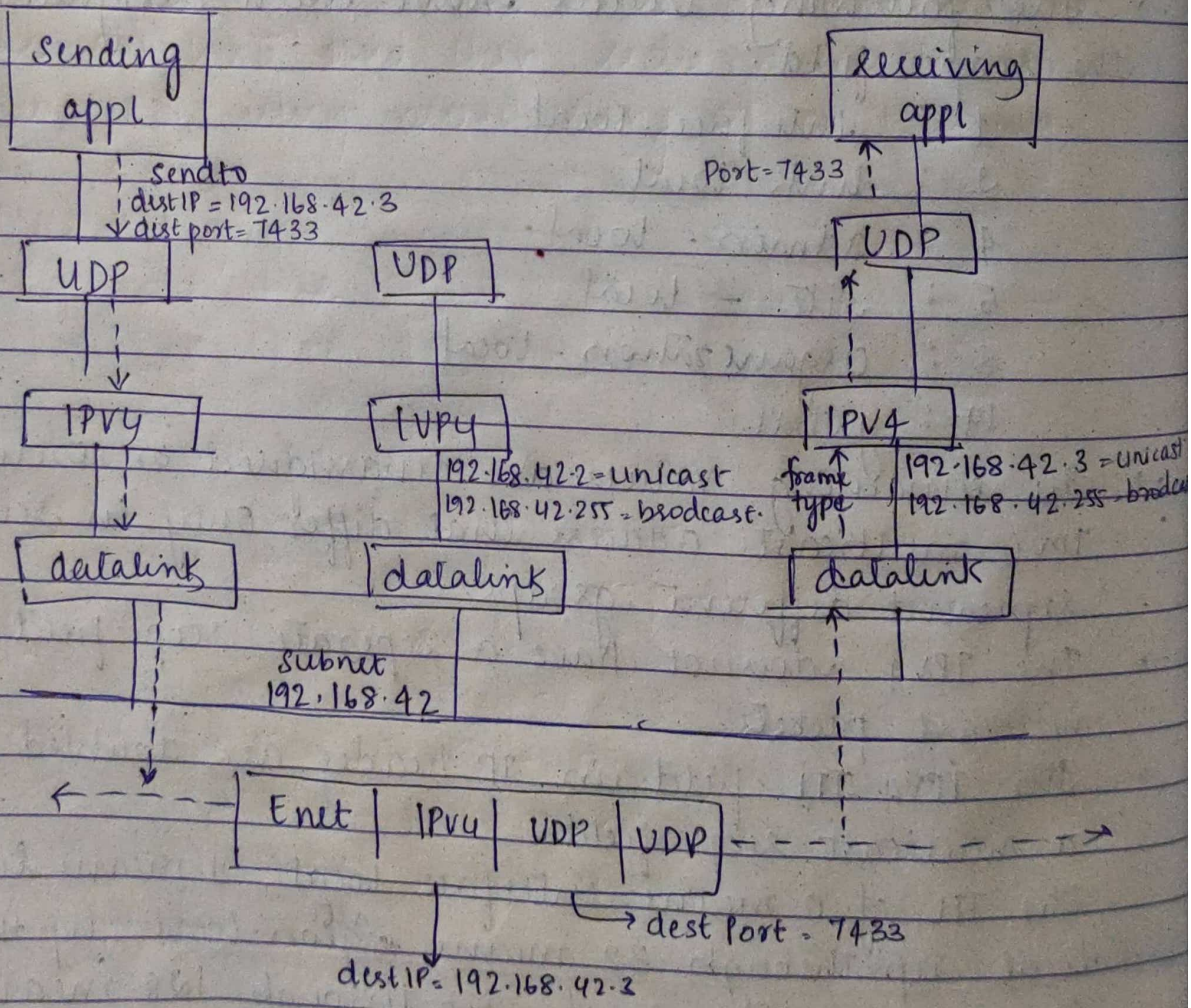
→ Although the use of IPV4 TTL field for scoping is accepted & recommended practice, Administrative scoping is preferred when possible.

(b) Describe scope of multicast addresses.

→ This defines the range 239.0.0.0 through 239.255.255.255.

→ Administratively scopped IPV4 multicast address is divided into local scope & organization-local scope the former being similar to IPV6 site local scope.

(2) Explain the Unicast example of a UDP datagram



→ The subnet address of Ethernet is 192.168.42/24 with 24 bits in network mask leaving 8 bits for host ID.

→ The application on left host calls sendto on a UDP socket, sending a ~~data~~ dest IP address 192.168.42.3 and port 7433.

→ The UDP layer prepends a UDP header and passes the UDP datagram to IP layer.

→ The IP prepends IPV4 header, determine outgoing interface.

→ In case of Ethernet, ARP is invoked to map destination IP address to its corresponding Ethernet address 00:0a:95:79:bc:b4.

→ The packet is then sent as a Ethernet frame with 48-bit address as destination Ethernet address.

→ The frame type field of Ethernet frame will be 0x0800, specifying IPV4 packet, and 0x86dd for IPV6 packet.

→ The Ethernet interface on host in middle compare the destination Ethernet address to its own Ethernet address 00:04:ac:17:bf:38. Since they are not equal it is ignored.

→ The Ethernet interface on host in right compares the destination Ethernet address to its own Ethernet address, they are equal.

It reads the entire frame, generate a hardware interrupt when frame is complete.

Since the frame type is 0x0800, the packet is placed on the IP input queue.

→ When the IP layer processes the packet, it first compare the destination IP address to its own IP address

since the destination address is one of the host's own IP address the packet is accepted.

28 30
27 29

26 25
23 22

(3) Explain SNTP protocol:-

→ NTP is a sophisticated protocol for synchronizing clocks across a WAN or a LAN, and can often achieve millisecond accuracy.

→ RFC 1305 discribes the protocol in detail & RFC 2030 describes SNTP, a simplified but protocol-compatible version intended for hosts that do not need the complexity of a complete NTP implementation.

→ It is common for a few hosts on a LAN to synchronize their clocks across the Internet to other NTP hosts and then redistribute this time on the LAN using broadcasting or multicasting.

```
Struct ntpdata {
        u_char status;
        u_char stratum;
        u_char ppoll;
        int precision:8;
        Struct s_fixedpt distance;
        Struct s_fixedpt dispersion;
        uint32_t refid;
        Struct l_fixedpt reftime;
        Struct l_fixedpt org;
        Struct l_fixedpt rec;
        Struct l_fixedpt xmt;
};
```

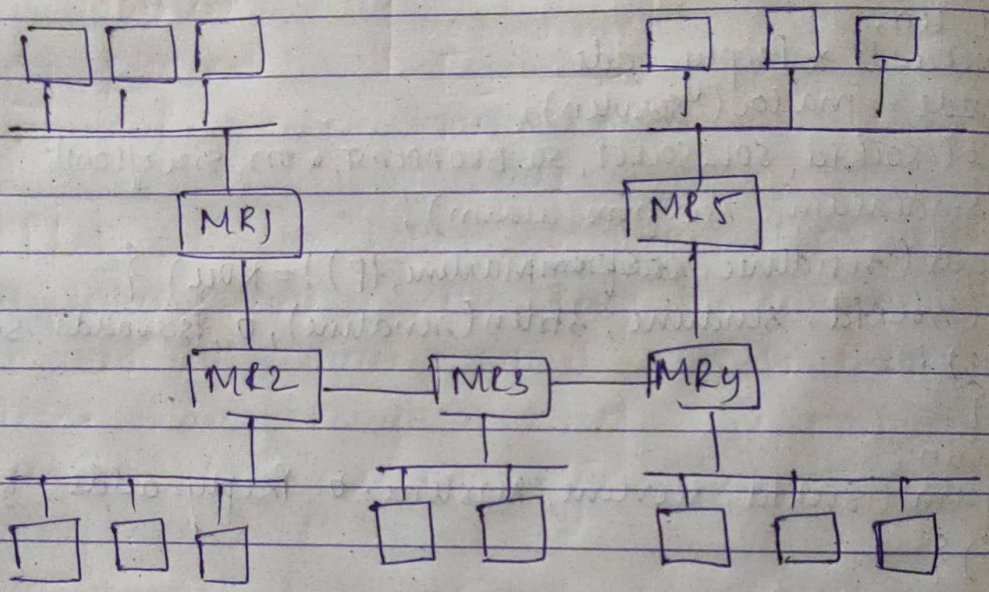
(4) Demonstrate with code, the use of dg_cli function that broadcasts.

```
include "unp.h"
static void recvfrom_alarm (int);
void
dg_cli (FILE* fp, int sockfd, const SA* Pservaddr, Socklen_t servlen)
{ int n;
const int on = 1;
char sendline [MAXLINE];
     recvline [MAXLINE+1];
socklen_t len;
struct sockaddr * Preply_addr;
Preply_addr = malloc(servlen);
setsockopt (sockfd, SOL_SOCKET, SO_BROADCAST, & on, sizeof(on));
Signal (SIGALARM, recvfrom_alarm);
while (Fgets (sendline, ~~exclfrom~~ Maxline, fp) != NULL) {
 Sendto (sockfd, sendline, strlen(sendline), 0, Pservaddr, Servlen);
alarm(5);
for(;;) {
len=servlen;
n= recvform (sockfd, recvline, Maxline, 0, Preply-addr, & len);
if (n<0) {
  if(errno== EINTR)
     break;
else
  err_sys("Recvform error");
} else {
   recvline[n] = 0
   Printf("form %s: %s", Sock_ntop_host (preply-addr, len)
                          recvline);
}}}
free (Preply-addr); }
static void
recvform_alarm (int signo)
{ return)
}.
```

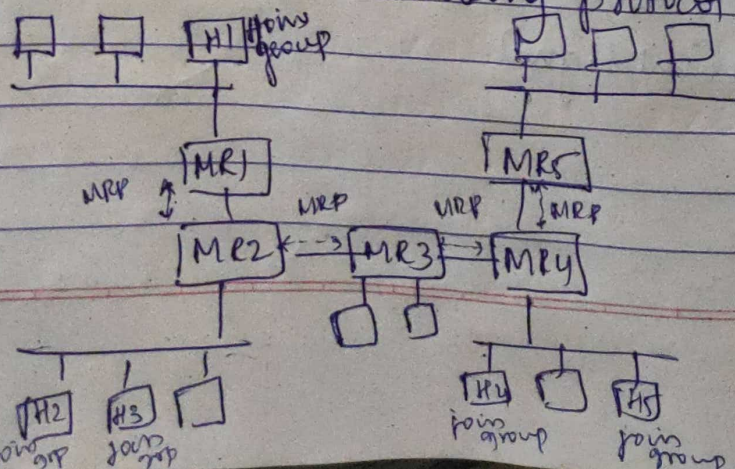
(5) Justify how beneficial multicasting on a WAN network.

Ans: Multicasting is also beneficial for WAN. Consider a WAN which shows five LANs connected with Five multicast routers.

Five LAN's connected with five multicast routers.



→ Next, assume that some program is started on five of hosts & those five programs join a given multicast group.
→ Each five host then joins the multicast group.
→ We also assume that the multicast routers communicate with all their neighbouring multicast routers using MRP multicast routing protocol.



When host joins a multicast group on a WAN

→ When a process on host joins the multicast group, the host sends an IGMP message to any attached multicast routers telling them that the host has just joined that group.

→ The multicast routers then exchange the informatn using MRP so that each multicast router knows what to do if it receives a packet destined to multicast address.

(8) Unit - 1 extra

(1) Explain OSI model