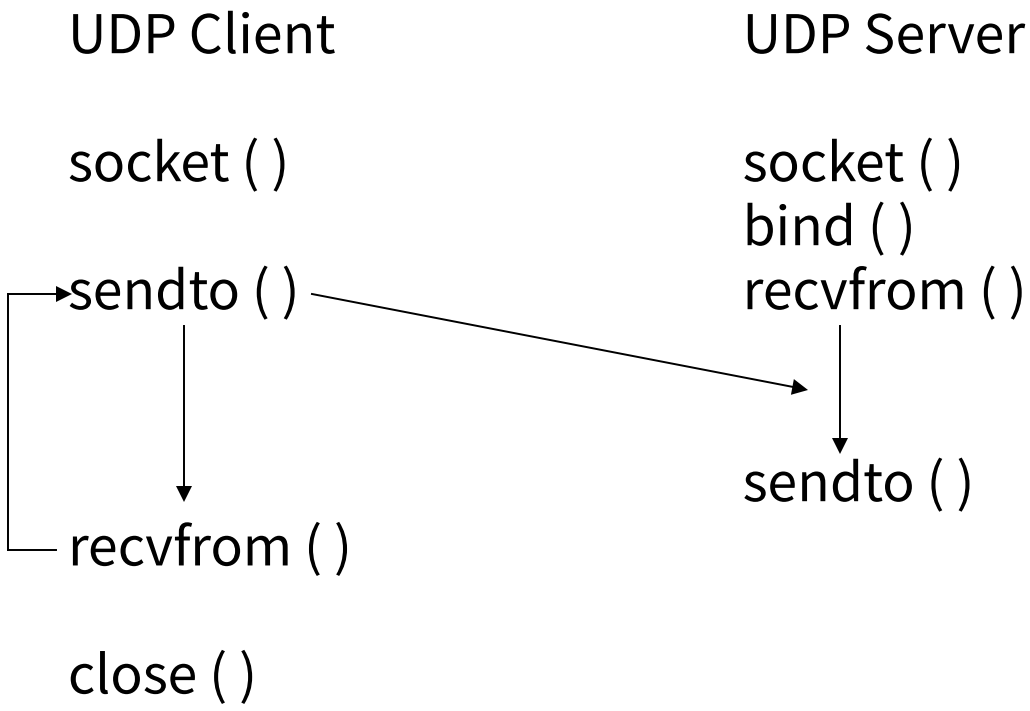
Elementary UDP Sockets

connectionless, unreliable, datagram

- *recvfrom* and *sendto* functions
- UDP echo server
- UDP echo client
- Verify received responses
- *connect* function with UDP
- Rewrite *dg_cli* function with *connect*
- Lack of flow control with UDP
- Determine outgoing interface with UDP
- TCP and UDP echo server using *select*

recvfrom and *sendto* Functions

```
#include <sys/socket.h>
ssize_t recvfrom (int sockfd, void *buff, size_t nbytes, int flags,
                  struct sockaddr *from, socklen_t *addrlen);
ssize_t sendto (int sockfd, const void *buff, size_t nbytes, int flags,
                  const struct sockaddr *to, socklen_t addrlen);
                  both return: number of bytes read or written if OK, -1 on erro
```



UDP Echo Server: main Function

udpcliserv/udpserv01.c

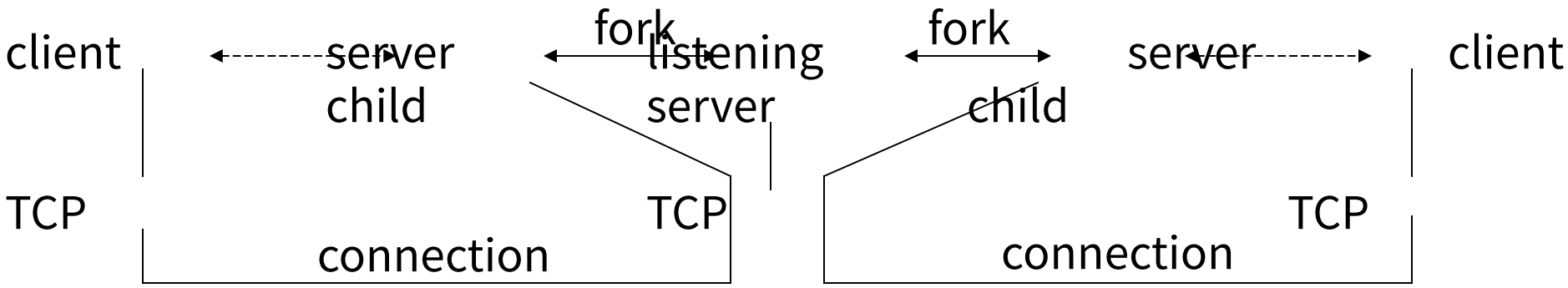
```
#include    "unp.h"
int
main(int argc, char **argv)
{
    int                 sockfd;
    struct sockaddr_in  servaddr, cliaddr;

    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));
}
```

UDP Echo Server: dg_echo Function

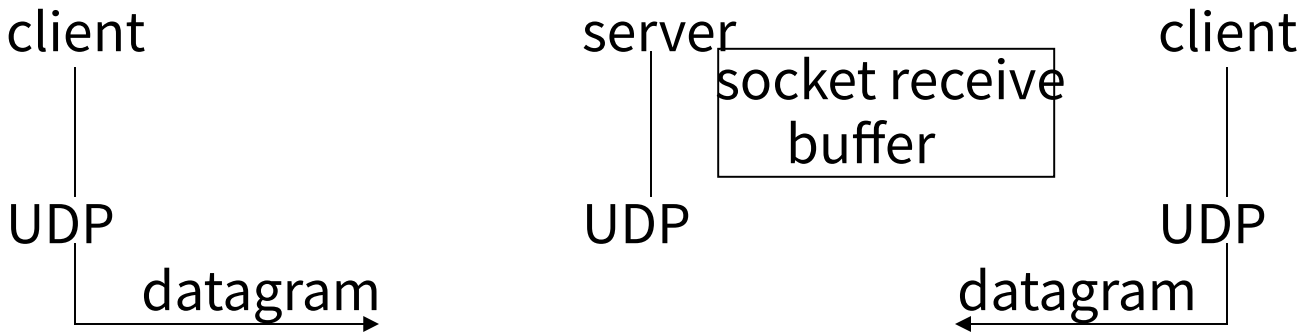
lib/dg_echo.c

```
#include      "unp.h"
void
dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
{
      int                n;
      socklen_t      len;
      char           mesg[MAXLINE];

      for ( ; ; ) {
            len = clilen;
            n = Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);

            Sendto(sockfd, mesg, n, 0, pcliaddr, len);
      }
}
```

Comparing TCP and UDP with Two Clients



TCP client-server with two clients



UDP client-server with two clients

Most TCP servers are concurrent and most UDP servers are iterative.

UDP Echo Client: main Function

udpcliserv/udpcli01.c

```
#include      "unp.h"
int
main(int argc, char **argv)
{
      int                        sockfd;
      struct sockaddr_in     servaddr;

      if (argc != 2)
            err_quit("usage: udpcli <IPaddress>");
      bzero(&servaddr, sizeof(servaddr));
      servaddr.sin_family = AF_INET;
      servaddr.sin_port = htons(SERV_PORT);
      Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
      sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

      dg_cli(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr));
      exit(0);
}
```

UDP Echo Client: dg_cli Function

lib/dg_cli.c

```
#include      "unp.h"
void
dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
{
      int    n;
      char   sendline[MAXLINE], recvline[MAXLINE + 1];

      while (Fgets(sendline, MAXLINE, fp) != NULL) {
            Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

            n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);

            recvline[n] = 0;      /* null terminate */
            Fputs(recvline, stdout);
      }
}
```

Problems with UDP Sockets

- Lost datagrams (client request or server reply):
  - recvfrom blocks forever
  - place a timeout, but don't know whether request or reply gets lost
- Malicious datagrams inter-mixed with server replies:
  - ignore any received datagrams not from that server
  - allocate another socket address structure and compare returned address

Problems with UDP Sockets (Cont.)

- For multi-homed server, verifying address may not work (server reply may go through another outgoing interface)
  - solution 1: verify server domain name, instead
  - solution 2: multi-homed UDP server creates one socket for every interface (IP addr), bind IP addresses to sockets, use *select* across all sockets

Problems with UDP Sockets (Cont.)

- Server not running:
  - ICMP port unreachable error (asynchronous error)
  - asynchronous errors not returned for UDP sockets unless the socket has been connected (reason?: considering a client sending 3 datagrams to 3 servers, recvfrom has no way to know the destination of the datagram causing the error)
  - recvfrom blocks forever
  - solution: call *connect* on a UDP socket

Problems with UDP Sockets (Cont.)

- Lack of flow control:
  - considering dg_cli in a client sendto 2000 1400-byte datagrams to the server
  - client may overrun the server (e.g. 96% loss rate: mostly lost due to receive buffer overflow, some due to network congestion)
  - use netstat -s to check the loss
  - solution: use SO_RCVBUF option to enlarge buffer, use request-reply model instead of bulk transfer

connect Function with UDP

- connect on UDP socket: no 3-way handshake, kernel only records the connected dest address
- For a connected UDP socket (compared to unconnected UDP socket):
  - no longer specify dest IP addr/port for output, use write/send instead of sendto
  - use read/recv instead of recvfrom
  - asynchronous errors are returned to the process for a connected UDP socket
- Used only if the client/server uses the UDP socket to communicate with exactly one peer

connect Function with UDP (cont.)

- *connect* multiple times for a UDP socket:
  - specify a new IP address/port to communicate
  - unconnect the socket (AF_UNSPEC)
- Call *sendto* for two datagrams on an unconnected UDP socket (temporary connecting):
  - steps in kernel: connect/output/unconnect, connect/output/unconnect (too much overhead)
  - solution: *connect* before *write* multiple datagrams

dg_cli Function That Calls *connect*

udpcliserv/dgcliconnect.c

```
#include      "unp.h"
void
dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
{
      int          n;
      char    sendline[MAXLINE], recvline[MAXLINE + 1];

      Connect(sockfd, (SA *) pservaddr, servlen);
      while (Fgets(sendline, MAXLINE, fp) != NULL) {

            Write(sockfd, sendline, strlen(sendline));
            n = Read(sockfd, recvline, MAXLINE);
            recvline[n] = 0;      /* null terminate */
            Fputs(recvline, stdout);
      }
}
```

connect to Determine Outgoing Interface with UDP

- No way to know the outgoing interface of an unconnected UDP socket
- Side effect of *connect* on UDP socket:
  - kernel chooses the local IP address by searching routing table
  - process calls getsockname to obtain the local IP addr and port

Combined TCP and UDP Echo Server Using *select*

- A single server using *select* to multiplex a TCP socket and a UDP socket:
  - create listening TCP socket
  - create UDP socket
  - establish signal handler for SIGCHLD
  - prepare for *select*
  - call *select*
  - handle new client connection
  - handle arrival of datagram

TCP and UDP Echo Server Using Select

udpcliserv/udpservselect.c

```
#include      "unp.h"
int
main(int argc, char **argv)
{
      int                     listenfd, connfd, udpfd, nready, maxfdp1;
      char                    mesg[MAXLINE];
      pid_t                   childpid;
      fd_set                  rset;
      ssize_t                 n;
      socklen_t               len;
      const int               on = 1;
      struct sockaddr_in      cliaddr, servaddr;
      void                    sig_chld(int);

            /* 4create listening TCP socket */
      listenfd = Socket(AF_INET, SOCK_STREAM, 0);
```

```
 bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family      = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port        = htons(SERV_PORT);

Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));

Listen(listenfd, LISTENQ);

      /* 4create UDP socket */
udpfd = 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(udpfd, (SA *) &servaddr, sizeof(servaddr));
```

```
Signal(SIGCHLD, sig_chld);/* must call waitpid() */

FD_ZERO(&rset);
maxfdp1 = max(listenfd, udpfd) + 1;
for ( ; ; ) {
        FD_SET(listenfd, &rset);
        FD_SET(udpfd, &rset);
        if ( (nready = select(maxfdp1, &rset, NULL, NULL, NULL)) < 0) {
                if (errno == EINTR)
                        continue;               /* back to for() */
                else
                        err_sys("select error");
        }
        if (FD_ISSET(listenfd, &rset)) {
                len = sizeof(cliaddr);
                connfd = Accept(listenfd, (SA *) &cliaddr, &len);
```

```
            if ( (childpid = Fork()) == 0) {      /* child process */
                Close(listenfd);      /* close listening socket */
                str_echo(connfd);     /* process the request */
                exit(0);
            }
            Close(connfd);            /* parent closes connected socket */
        }

        if (FD_ISSET(udpfd, &rset)) {
            len = sizeof(cliaddr);
            n = Recvfrom(udpfd, mesg, MAXLINE, 0, (SA *) &cliaddr, &len);

            Sendto(udpfd, mesg, n, 0, (SA *) &cliaddr, len);
        }
    }
}
```

Summary

- Lots of features in TCP are lost with UDP:
  - detecting lost packets, retransmitting, verifying responses as being from correct peer, flow control, etc.
- Some reliability can be added
- UDP sockets may generate asynchronous errors reported only to connected sockets
- Use a request-reply model