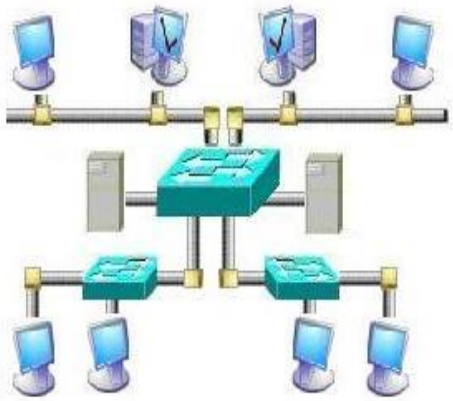
MC833
Programação em Redes de Computadores

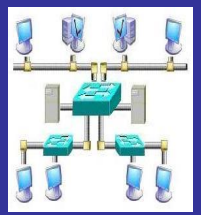
Primeiro Semestre 2019

Prof. Edmundo R. M. Madeira

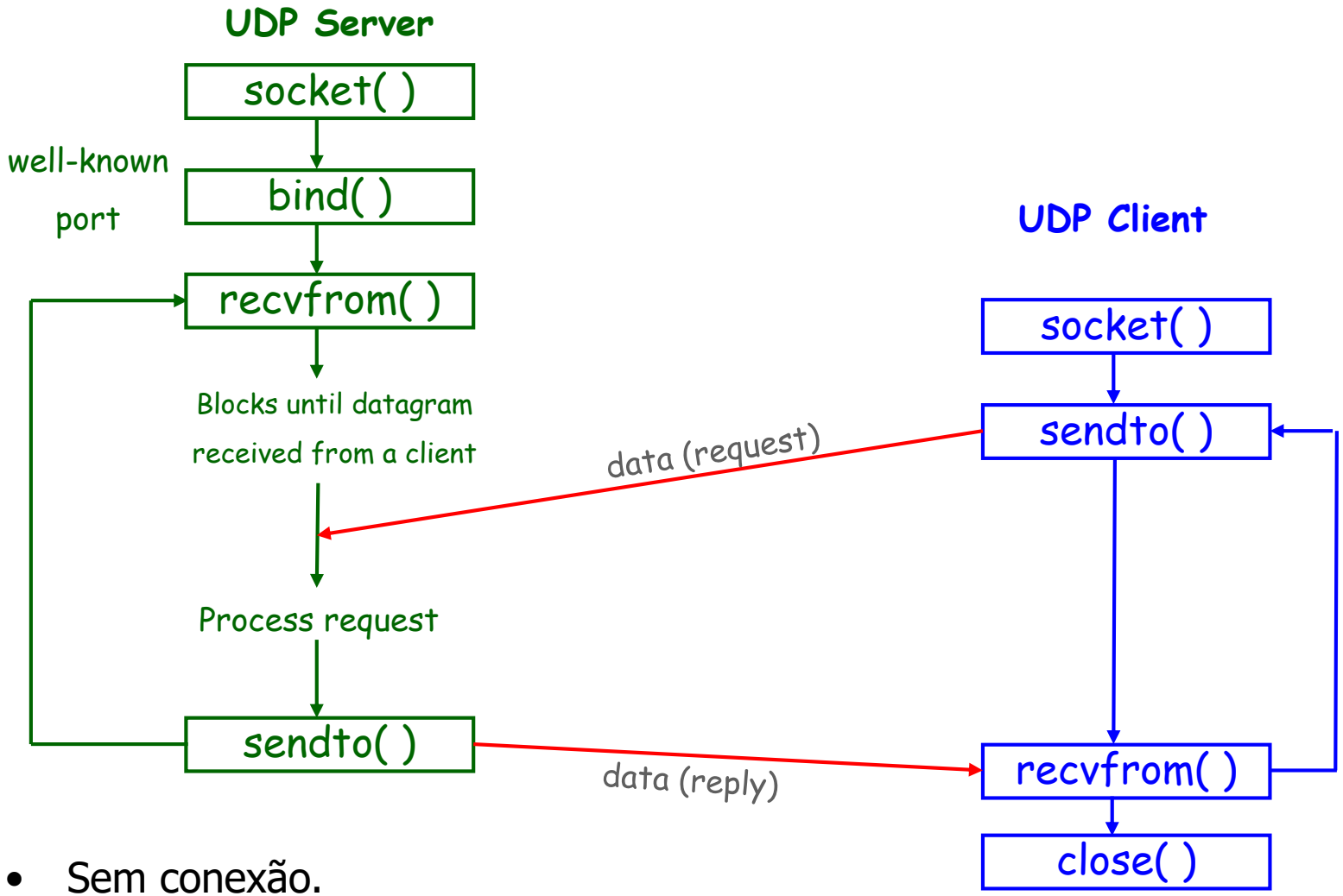


INSTITUTO DE COMPUTAÇÃO
Universidade Estadual de Campinas
UNICAMP

Sockets UDP

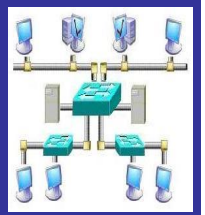


Funções para cliente-servidor UDP



- Sem conexão.
- Não implementa entrega confiável.
- DNS, NFS, SNMP.

Stevens Vol. 1 Fig. 8.1



Funções recvfrom e sendto

#include <sys/socket.h>

int recvfrom(int *sockfd,* void **buf*, int len, unsigned int *flags*, struct sockaddr **from*, int **fromlen*);

int sendto(int *sockfd,* const void **buf*, int *len*, Unsigned int *flags*, const struct sockaddr **to*, socklen_t **tolen*);

Retorno: número de bytes lidos ou escritos se OK, -1 para erro.

Argumentos:

sockfd: *descriptor*

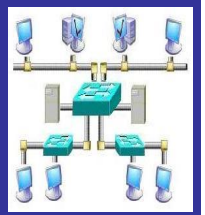
buf: buffer de leitura/escrita

len: quantidade de bytes leitura/escrita

flags: por enquanto = 0

from/to: ponteiro para *socket address structure*

fromlen/tolen: tamanho do *socket address structure*

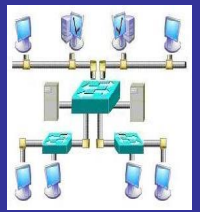


Funções recvfrom e sendto

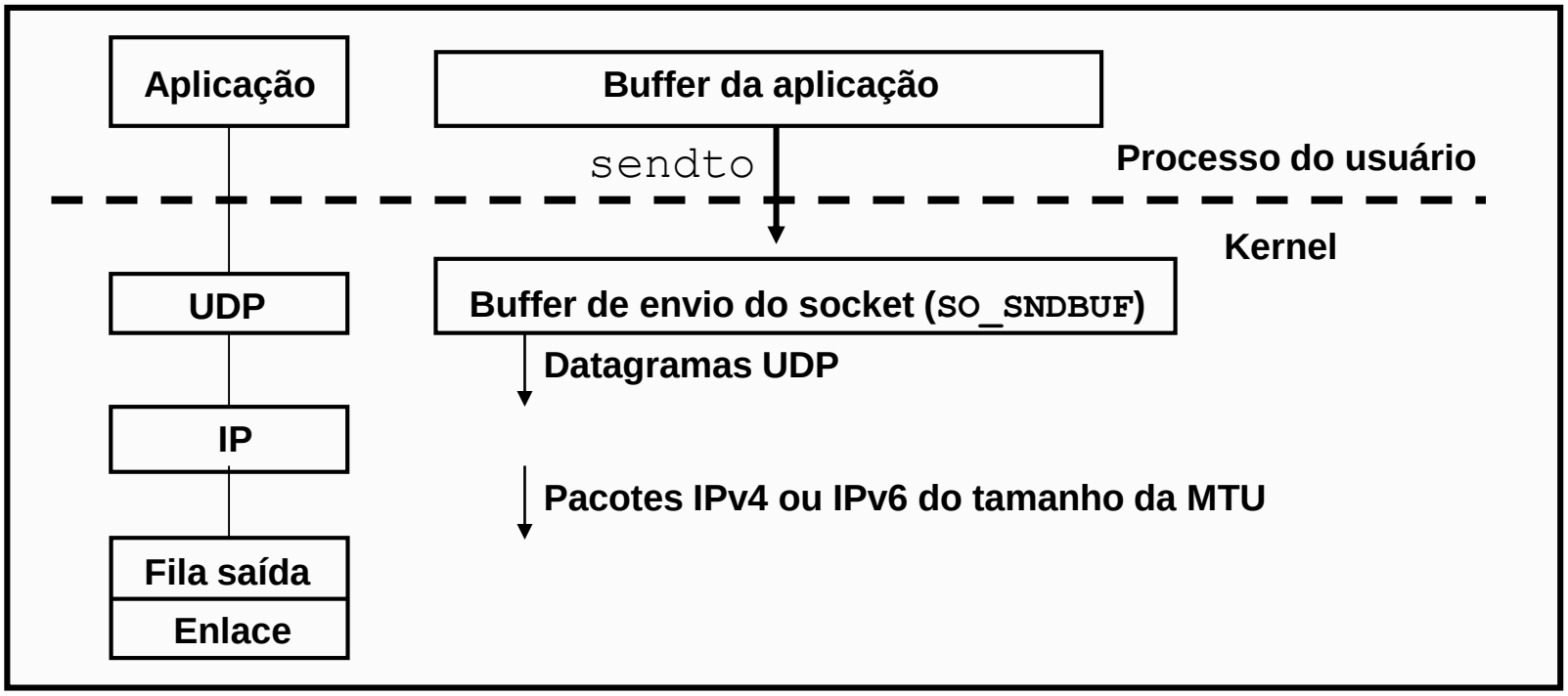
int recvfrom (int *sockfd,* void **buf*, int *len*, unsigned int *flags*,
struct sockaddr **from*, int **fromlen*);

int sendto (int *sockfd,* const void **buf*, int *len*,
Unsigned int *flags*, const struct sockaddr **to*, socklen_t **tolen*);

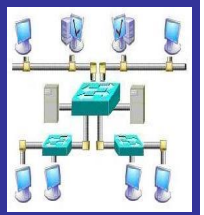
- Escrever datagrama tamanho zero é válido (20 bytes IPv4 + 8 bytes UDP header).
- **recvfrom** e **addrlen** podem ser ambos nulos. Não importa quem enviou.
- **recvfrom** e **sento** podem ser usados por TCP Exemplo: T / TCP – TCP for Transaction.



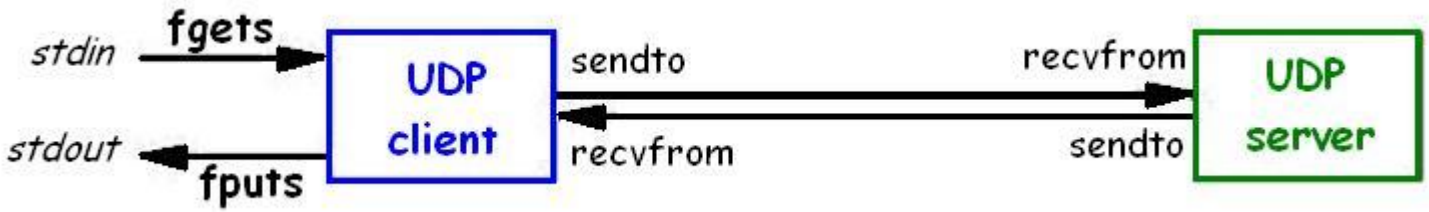
Função sendto



- Sucesso do write: *datagrama* foi colocado na fila de saída do enlace.

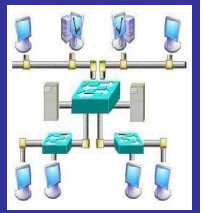


UDP Echo Server

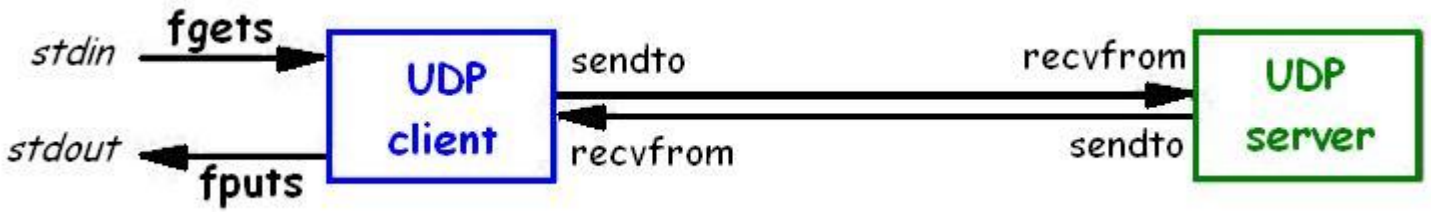


Stevens Vol. 1 Fig. 8.1

```
1 #include "unp.h"
2
3 int main(int argc, char **argv)
4 {
5         int sockfd;
6         struct sockaddr_in servaddr, cliaddr;
7         sockfd = Socket (AF_INET, SOCK_DGRAM, 0);
8         Bzero (&servaddr, sizeof(servaddr));
9         servaddr.sin_family = AF_INET;
10        servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
11        servaddr.sin_port = htons (SERV_PORT);
12        Bind (sockfd, (SA *) &servaddr, sizeof(servaddr));
13        dg_echo (sockfd, (SA *) &cliaddr, sizeof(cliaddr));
14 }
```

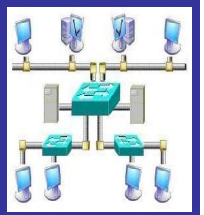


UDP Echo Server



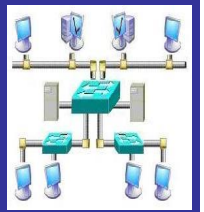
Stevens Vol. 1 Fig. 8.1

```
1 #include "unp.h"
2
3 void dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
4 {
5         int n;
6         socklen_t len;
7         char mesg[MAXLINE];
8         for ( ; ; ) {
9             len = clilen;
10            n = Recvfrom (sockfd, mesg, MAXLINE, 0, pcliaddr, &len);
11            Sendto (sockfd, mesg, n, 0, pcliaddr, len);
12        }
13 }
```

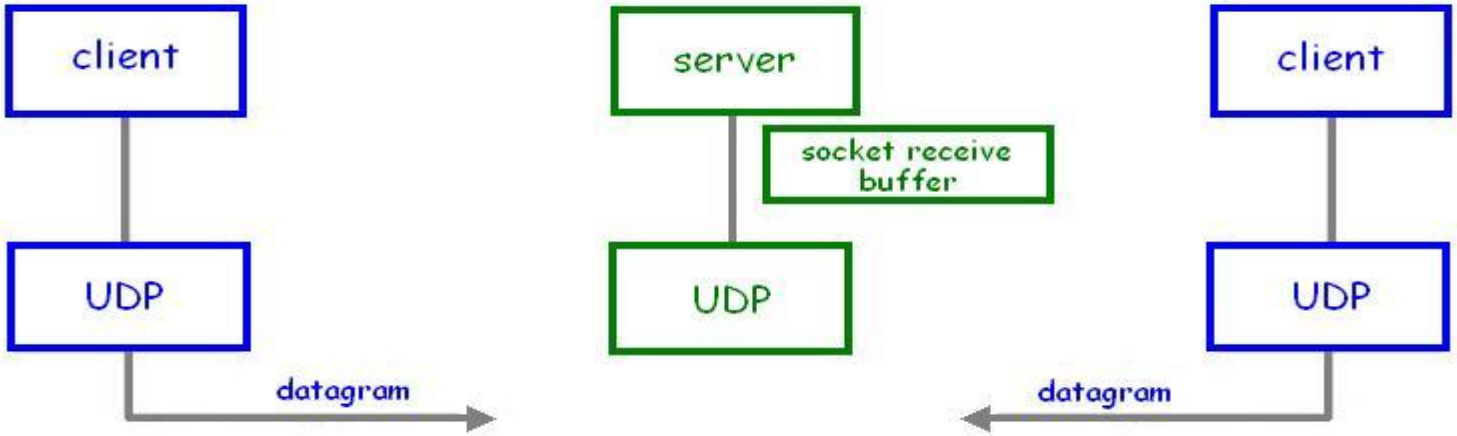


Servidor UDP

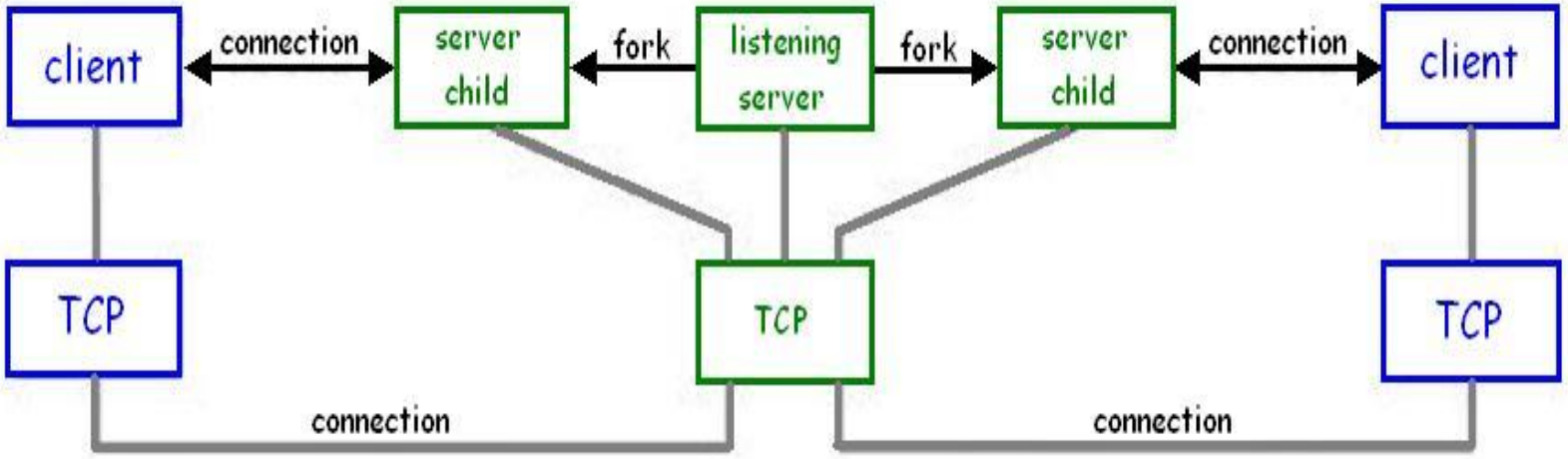
- Não há *end of file*
  - TCP ✹ concorrente.
  - UDP ✹ interativo.
- Apenas um processo servidor e único *socket.*
- Um único buffer no qual todos os datagramas chegam.



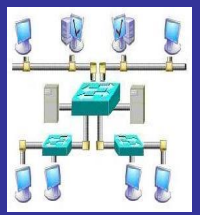
Servidor UDP x TCP



Stevens Vol. 1 Fig. 8.6

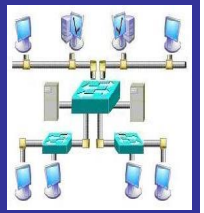


Stevens Vol. 1 Fig. 8.5



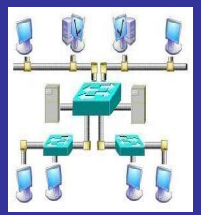
UDP Echo Client


```
1 #include "unp.h"
2
3   int main(int argc, char **argv)
4   {
5           int sockfd;
6           struct sockaddr_in servaddr;
7           if(argc != 2)
8                   err_quit("usage: udpcli <IPaddress>");

9           bzero(&servaddr, sizeof(servaddr));
10          servaddr.sin_family = AF_INET;
11          servaddr.sin_port = htons (SERV_PORT);
12          Inet_pton (AF_INET, argv[1], &servaddr.sin_addr);

13          sockfd = Socket (AF_INET, SOCK_DGRAM, 0);

14          dg_cli(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr));

15          exit(0);
16  }
```



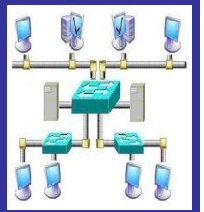
UDP Echo Client (função dg_cli)


```
1 #include "unp.h"
2
3 void dg_cli(FILE *fp,int sockfd,const SA *pservaddr,socklen_t,servlen)
4 {
5   int n;
6   char sendline[MAXLINE], recvline[MAXLINE + 1];

7   while (Fgets(sendline, MAXLINE, fp) != NULL) {

8       Sendto (sockfd,sendline,strlen(sendline),0,pservaddr,servlen);

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

10      recvline[n] = 0; /* null terminate */
11      Fputs(recvline, stdout);
12  }
13 }
```



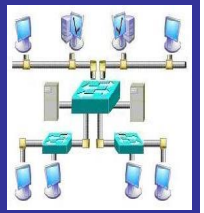
Datagramas perdidos

- Cliente – servidor UDP não são confiáveis.
- Se datagrama ou ACK de aplicação se perdem, cliente-servidor ficam bloqueados para sempre.



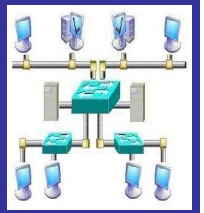
Verificando respostas

- Qualquer processo pode enviar datagramas para a porta efêmera do cliente.
- ***recvfrom*** retorna IP de quem enviou.
- Alterar o código para especificar a porta do servidor e alocar estrutura para salvar info sobre endereço IP.



Função *dg_cli* que verifica endereço


```
1 #include "unp.h"
2
3 void dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
4 {
5    int n;
6    char sendline[MAXLINE], recvline[MAXLINE + 1];
7    socklen_t len;
8    struct sockaddr *preply_addr;
9    preply_addr = Malloc(servlen);
10   while (Fgets(sendline, MAXLINE, fp) != NULL) {
11          Sendto (sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
12          len = servlen;
13          n = Recvfrom (sockfd, recvline, MAXLINE, 0, preply_addr, &len);
14          if (len != servlen || memcmp(pservaddr, preply_addr, len) != 0) {
15            printf("reply from %s (ignored)\n", Sock_ntop(preply_addr, len));
16            continue;
17          }
18          recvline[n] = 0; /* null terminate */
19          Fputs(recvline, stdout);
20   }
21 }
```



Servidor não responsivo

- tcpdump quando processo servidor não iniciou no servidor.

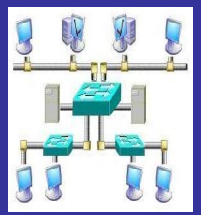
```
1 0.0 arp who-has freebsd4 tell macosx
2 0.003576 ( 0.0036) arp reply freebsd4 is-at 0:40:5:42:d6:de
3 0.003601 ( 0.0000) macosx.51139 > freebsd4.9877: udp 13
4 0.009781 ( 0.0062) freebsd4 > macosx: icmp: freebsd4 udp port    9877 unreachable
```

- sendto – retorna sucesso se houve espaço no buffer de envio para datagrama.

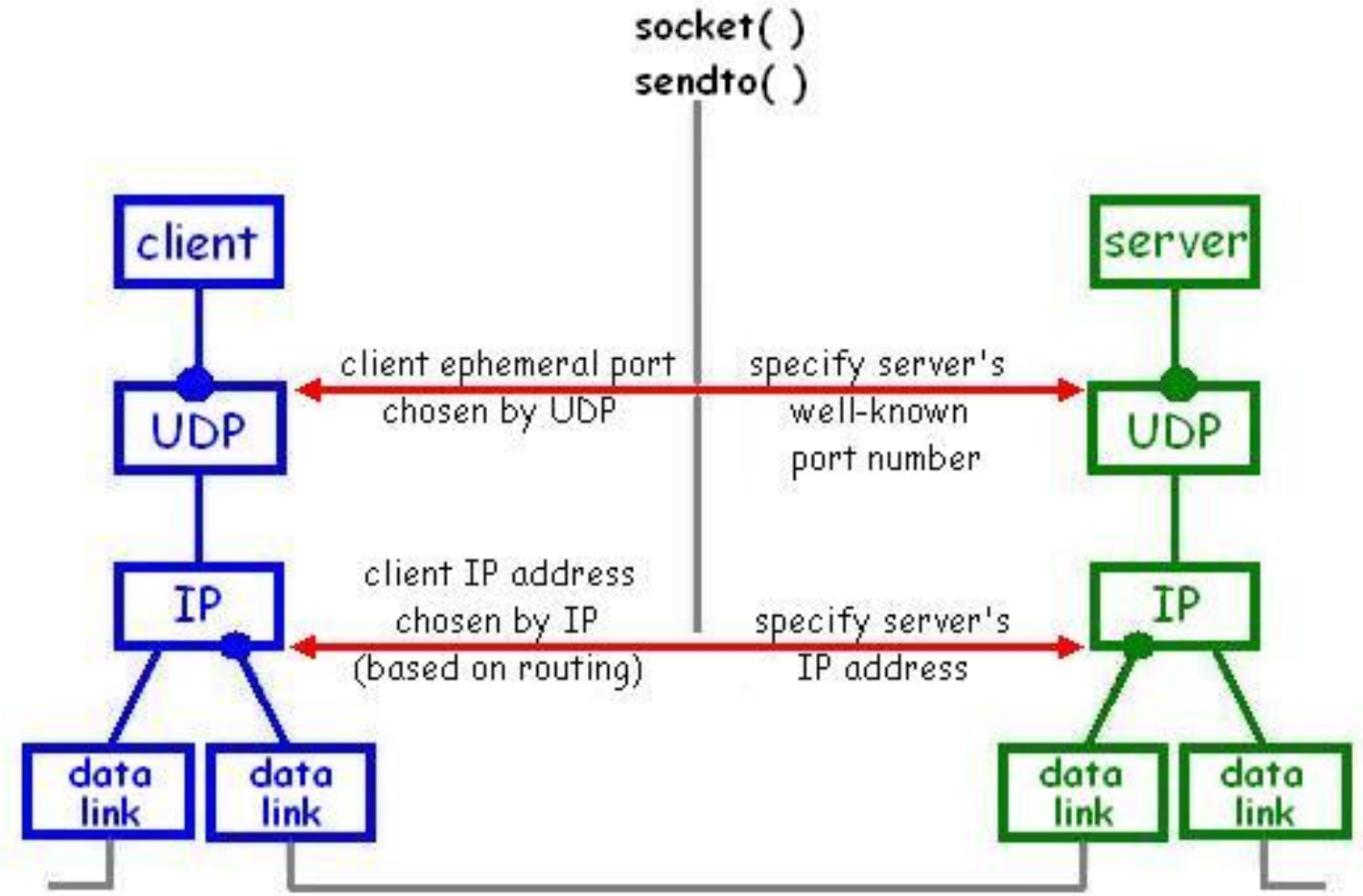
- Erro assíncrono – Não há como saber endereço IP e porta do servidor que não respondeu quando um único socket é usado para envio de datagrama para vários servidores.
  - ✹ Erro não é retornado para processo

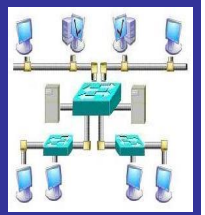
- connect ✹ socket UDP recebe de exatamente um *peer*.

- Erro retornado ao processo.



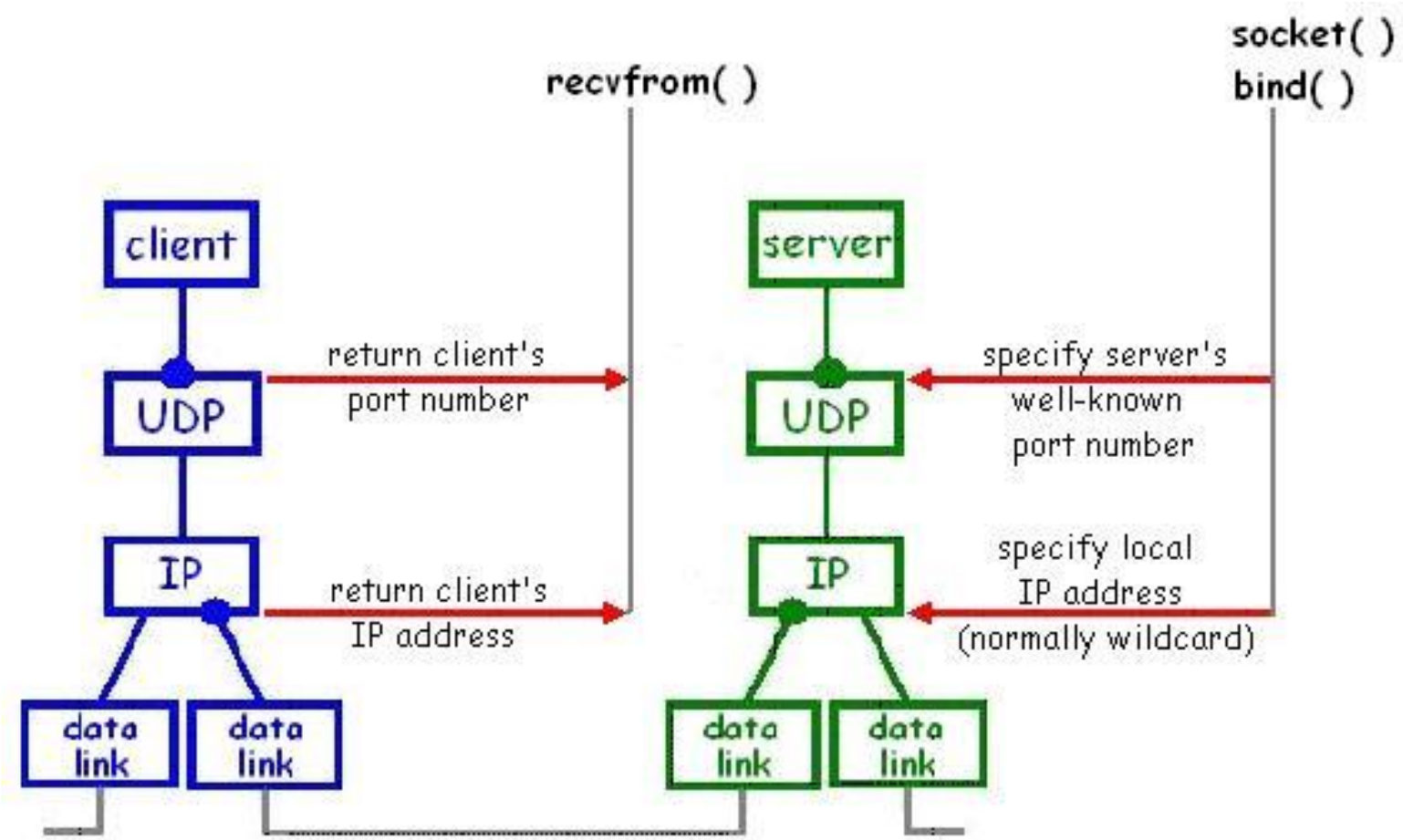
UDP Cliente-Servidor: visão cliente

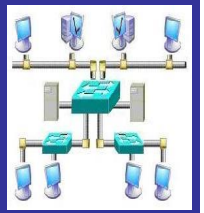




Stevens Vol. 1 Fig. 8.11

UDP Cliente-Servidor: visão servidor



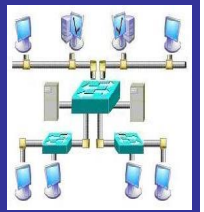


Stevens Vol. 1 Fig. 8.12

Função connect (UDP)


```
int connect ( int sockfd, const struct sockaddr *servaddr,
              socklen_t addrlen );
```

- Não há *three-way handshaking* – estabelecimento da conexão.
- Não é conexão TCP.
- O Kernel guarda o endereço IP e a porta do *peer*, armazenados na estrutura passada na chamada para **connect** (*servaddr*).
- Deve-se usar **write** ou **send** ao invés de **sento.**

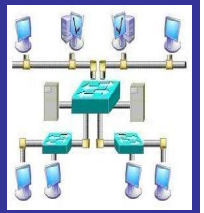


Função connect (UDP)


```
ssize_t sendto ( int sockfd, const void *buff, size_t nbytes,
int flags, const struct sockaddr *to, socklen_t *addrlen );
```

- Quando usar **sento** o ponteiro para a estrutura (**to*) e o seu tamanho (**addrlen*), devem ser nulos.

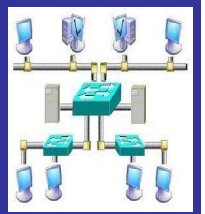
- Não se usa **recvfrom**. Usa-se **read** ou **recv**.



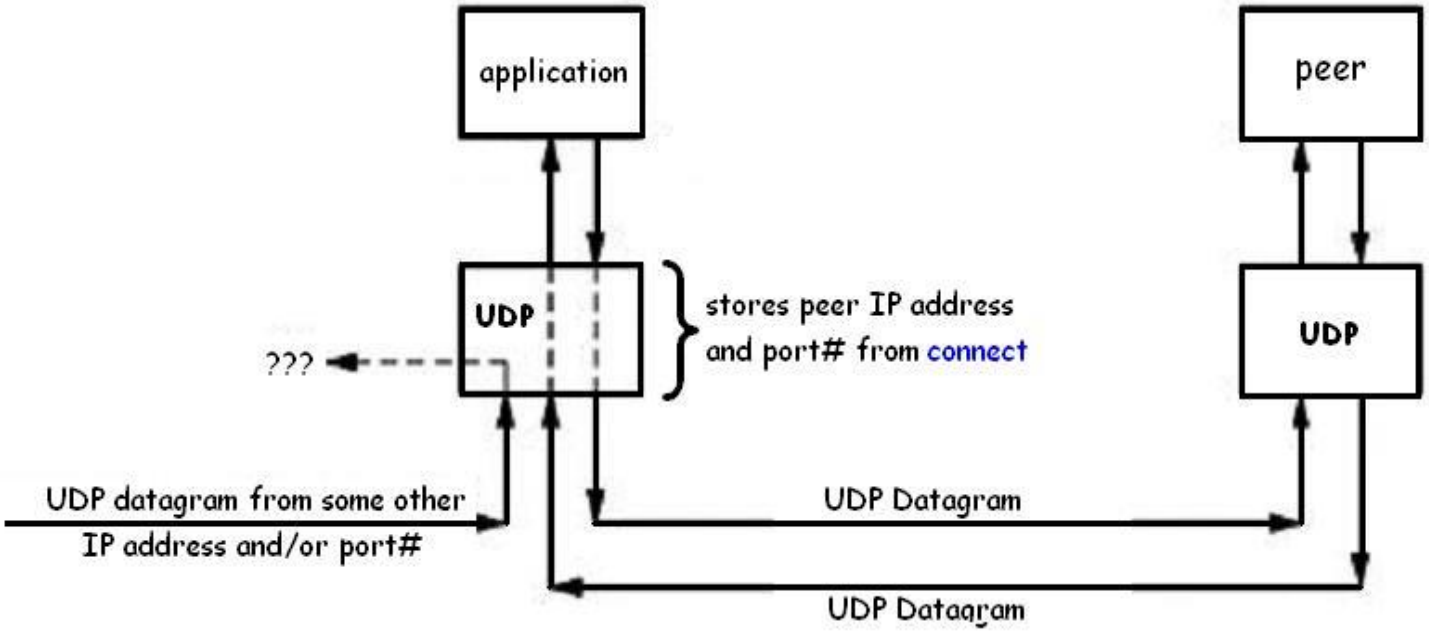
Função connect (UDP)


```
int connect ( int sockfd, const struct sockaddr *servaddr,
              socklen_t addrlen );
```

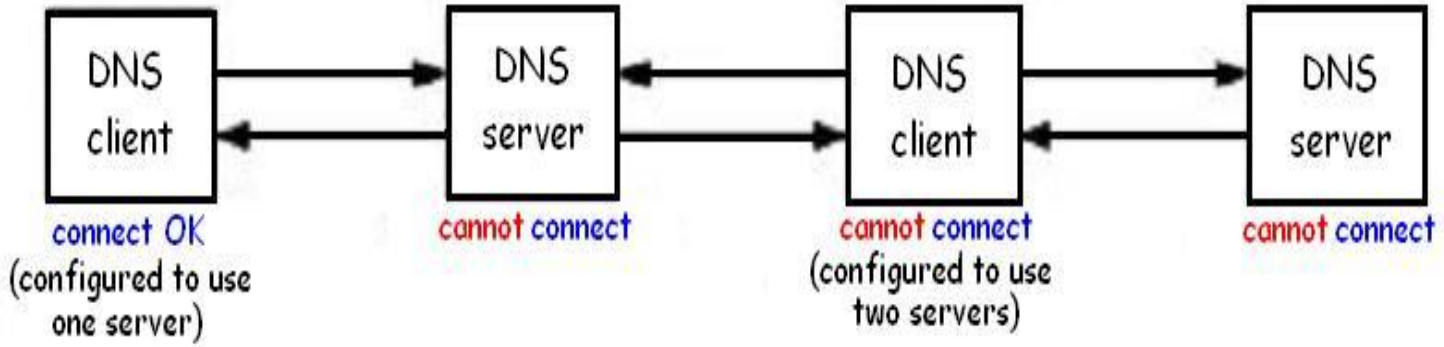
- Datagramas recebidos com endereço IP e porta diferentes do especificado em **connect** não são repassados.
  - Troca de informação com um peer somente.
- Erros assíncronos são retornados para processo **connect** em socket UDP.



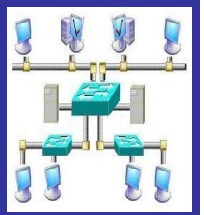
Função connect (UDP)



Stevens Vol. 1 Fig. 8.15

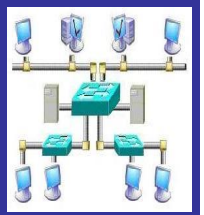


Stevens Vol. 1 Fig. 8.16



Função dg_cli com connect

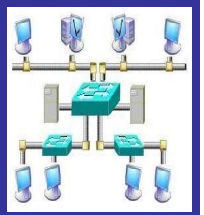

```
1 #include "unp.h"
2
3 void dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
4 {
5    int n;
6    char sendline[MAXLINE], recvline[MAXLINE + 1];


7    Connect(sockfd, (SA *) pservaddr, servlen);


8      while (Fgets(sendline, MAXLINE, fp) != NULL) {

9          Write (sockfd, sendline, strlen(sendline));

10         n = Read (sockfd, recvline, MAXLINE);

11         recvline[n] = 0; /* null terminate */
12         Fputs(recvline, stdout);
13   }
14 }
```



Flow Control

- Função dg_cli que escreve um número fixo de datagramas para o server.

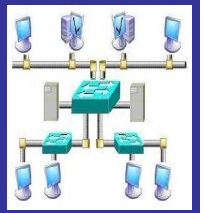
```
1 #include "unp.h"
2 #define NDG 2000 /* datagrams to send */
3 #define DGLEN 1400 /* length of each datagram */
4
5 void dg_cli(FILE *fp,int sockfd,const SA *pservaddr,socklen_t servlen)
6 {
7     int i;
8     char sendline[DGLEN];

9     for (i = 0; i < NDG; i++) {

10        Sendto ( sockfd, sendline, DGLEN, 0, pservaddr, servlen );

11    }
12 }
```



Flow Control

- Função dg_echo que conta os datagramas recebidos.

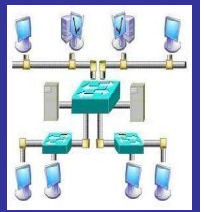
```
1 #include "unp.h"
2 static void recvfrom_int(int);
3 static int count;
4
5 void dg_echo ( int sockfd, SA *pcliaddr, socklen_t clilen )
6 {
7           socklen_t len;
8           char mesg[MAXLINE];
9           Signal ( SIGINT, recvfrom_int);
10          for ( ; ; ) {
11                      len = clilen;
12                      Recvfrom ( sockfd, mesg, MAXLINE, 0, pcliaddr, &len );
13                      count++;
14          }
15 }

16
17 static void recvfrom_int(int signo)
18 {
19          printf("\nreceived %d datagrams\n", count);
20          exit(0);
21 }
```



Flow Control

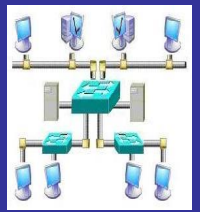
- Saída no *host* servidor.

```
freebsd % netstat -s -p udp
udp:
    71208 datagrams received
    0 with incomplete header
    0 with bad data length field
    0 with bad checksum
    0 with no checksum
    832 dropped due to no socket
    16 broadcast/multicast datagrams dropped due to no socket
    1971 dropped due to full socket buffers
    0 not for hashed pcb
    68389 delivered
    137685 datagrams output
```



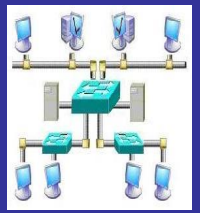
Servidor TCP e UDP usando select (1/4)


```
1 #include     "unp.h"

2
3 int main(int argc, char **argv)
4 {
5     int     listenfd, connfd, udpfd, nready, maxfdp1;
6     char    mesg[MAXLINE];
7     pid_t   childpid;
8     fd_set  rset;
9     ssize_t n;
10    socklen_t len;
11    const int on = 1;
12    struct sockaddr_in cliaddr, servaddr;
13    void    sig_chld(int);

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



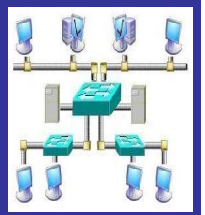
Servidor TCP e UDP usando select (2/4)


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

20      Setsockopt (listenfd,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));
21      Bind (listenfd, (SA *) &servaddr, sizeof(servaddr));
22      Listen (listenfd, LISTENQ);

23          /* create UDP socket */
24      udpfd = Socket (AF_INET, SOCK_DGRAM, 0);

25      bzero(&servaddr, sizeof(servaddr));
26      servaddr.sin_family = AF_INET;
27      servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
28      servaddr.sin_port = htons (SERV_PORT);

29      Bind (udpfd, (SA *) &servaddr, sizeof(servaddr));
...
```



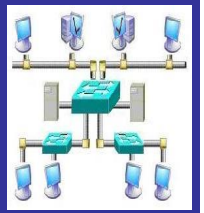
Servidor TCP e UDP usando select (3/4)


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

31      FD_ZERO(&rset);
32      maxfdp1 = max(listenfd, udpfd) + 1;
33      for ( ; ; ) {
34          FD_SET (listenfd, &rset);
35          FD_SET (udpfd, &rset);
36          if( (nready = select (maxfdp1, &rset, NULL, NULL, NULL)) <0) {
37              if (errno == EINTR)
38                  continue;      /* back to for() */
39              else
40                  err_sys ("select error");
41           }
...
```



Servidor TCP e UDP usando select (4/4)


```
42          if ( FD_ISSET (listenfd, &rset)) {
43              len = sizeof(cliaddr);
44              connfd = Accept (listenfd, (SA *) &cliaddr, &len);

45              if ( (childpid = Fork ()) == 0) { /*child process */
46                  Close (listenfd);   /* close listening socket */
47                  str_echo(connfd);  /* process the request */
48                  exit(0);
49               }
50               Close (connfd);   /* parent closes connected socket */
51          }

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

55              Sendto (udpfd, mesg, n, 0, (SA *) &cliaddr, len);
56          }
57      }
58 }
```



RMI

Remote Method Invocation