# Advanced I/O Functions

- Socket timeouts
- recv and send functions
- readv and writev functions
- recvmsg and sendmsg functions
- Ancillary data
- T/TCP: TCP for transactions

## Socket Timeouts

- Call alarm which generates SIGALRM before calling socket functions
- Block waiting for I/O in select which has a time limit built in
- Set SO_RCVTIMEO and SO_SNDTIMEO socket options by setsockopt

## Figure 14.1 `connect` with a timeout.

*lib/connect_timeo.c*

```
#include    "unp.h"

static void connect_alarm(int);

int
connect_timeo(int sockfd, const SA *saptr, socklen_t salen, int nsec)
{
    Sigfunc *sigfunc;
    int     n;

    sigfunc = Signal(SIGALRM, connect_alarm);
    if (alarm(nsec) != 0)
        err_msg("connect_timeo: alarm was already set");

    if ( (n = connect(sockfd, saptr, salen)) < 0) {
        close(sockfd);
        if(errno == EINTR)
            errno = ETIMEDOUT;
    }
    alarm(0);                       /* turn off the alarm */
    Signal(SIGALRM, sigfunc);       /* restore previous signal handler */

    return (n);
}

static void
connect_alarm(int signo)
{
    return;                         /* just interrupt the connect() */
}
```

**Figure 14.2 `dg_cli` function with `alarm` to timeout `recvfrom`.**

*advio/dgclitimeo3.c*

```
#include    "unp.h"

static void sig_alrm(int);

void
dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
{
    int     n;
    char    sendline[MAXLINE], recvline[MAXLINE + 1];

    Signal(SIGALRM, sig_alrm);

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

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

        alarm(5);
        if ( (n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL)) < 0) {
            if (errno == EINTR)
                fprintf(stderr, "socket timeout\n");
            else
                err_sys("recvfrom error");
        } else {
            alarm(0);
            recvline[n] = 0;    /* null terminate */
            Fputs(recvline, stdout);
        }
    }
}

static void
sig_alrm(int signo)
{
    return;                     /* just interrupt the recvfrom() */
}
```

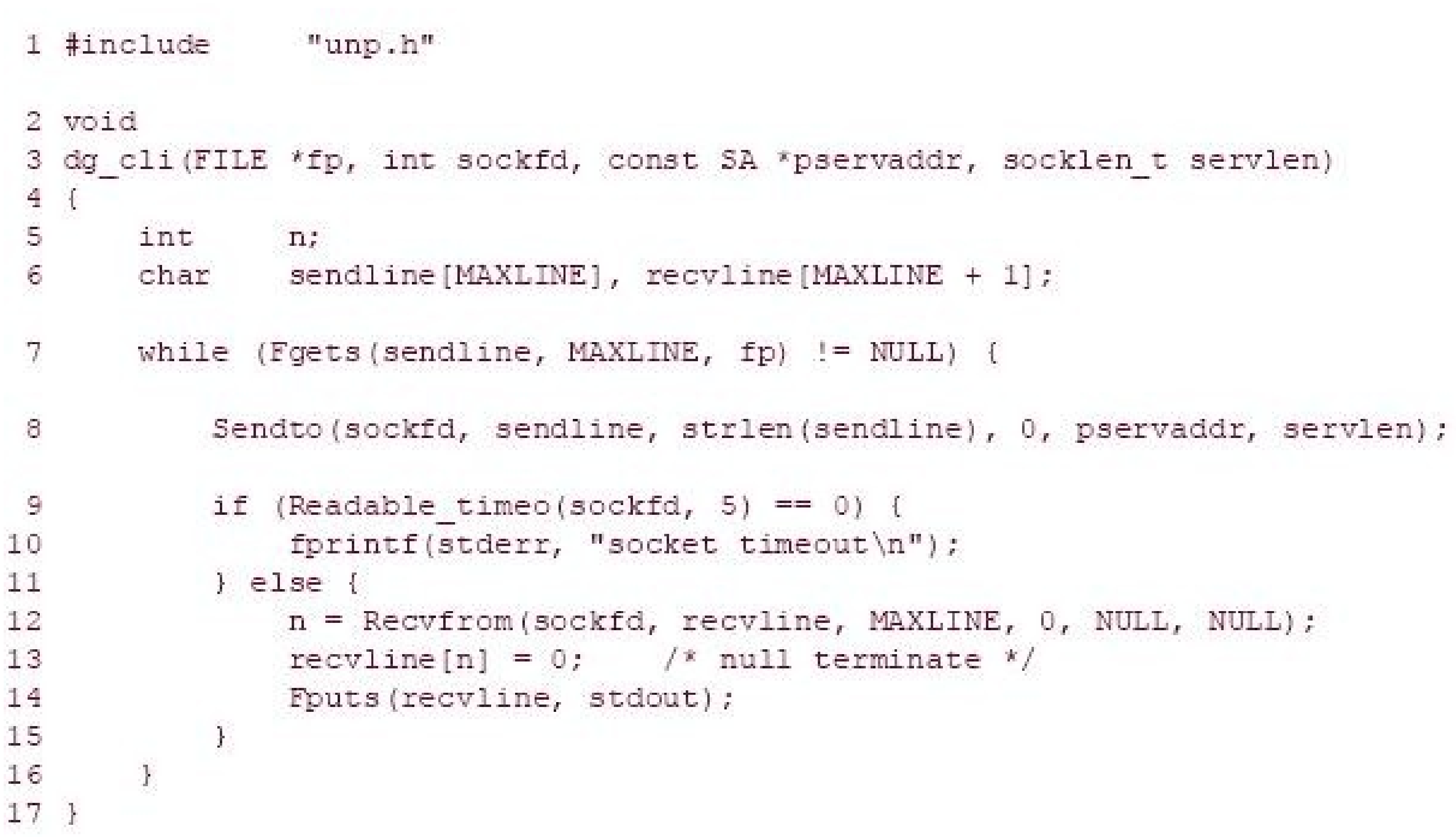
## Figure 14.3 `readable_timeo` function: waits for a descriptor to become readable.

*lib/readable_timeo.c*

```
#include      "unp.h"

int
readable_timeo(int fd, int sec)
{
    fd_set rset;
    struct timeval tv;

    FD_ZERO(&rset);
    FD_SET(fd, &rset);

    tv.tv_sec = sec;
    tv.tv_usec = 0;

    return (select(fd + 1, &rset, NULL, NULL, &tv));
        /* > 0 if descriptor is readable */
}
```

## Figure 14.4 `dg_cli` function that calls `readable_timeo` to set a timeout.

*advio/dgclitimeo1.c*



```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);

        if (Readable_timeo(sockfd, 5) == 0) {
            fprintf(stderr, "socket timeout\n");
        } else {
            n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
            recvline[n] = 0;    /* null terminate */
            Fputs(recvline, stdout);
        }
    }
}
```

**Figure 14.5 `dg_cli` function that uses the `SO_RCVTIMEO` socket option to set a timeout.**

*advio/dgclitimeo2.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];
    struct timeval tv;

    tv.tv_sec = 5;
    tv.tv_usec = 0;
    Setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));

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

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

        n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
        if (n < 0) {
            if (errno == EWOULDBLOCK) {
                fprintf(stderr, "socket timeout\n");
                continue;
            } else
                err_sys("recvfrom error");
        }

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

## recv and send Functions ~ read and write

```
#include <sys/socket.h>
    ssize_t recv (int sockfd, void *buff, size_t nbytes, int flags);
    ssize_t send (int sockfd, const void *buff, size_t nbytes, int flags);
      both return: number of bytes read or written if OK, -1 on error
```

| *flags* | Description | recv | send |
|---|---|---|---|
| MSG_DONTROUTE | bypass routing table lookup | | X |
| MSG_DONTWAIT | only this operation is nonblocking | X | X |
| MSG_OOB | send or receive out-of-band data | X | X |
| MSG_PEEK | peek at incoming message | X | |
| MSG_WAITALL | wait for all the data | X | |

# readv and writev Functions scatter read and gather write ~ read and write

```
#include <sys/uio.h>
ssize_t readv (int filedes, const struct iovec *iov, int iovcnt);
ssize_t writev (int filedes, const struct iovec *iov, int iovcnt);
  both return: number of bytes read or written, -1 on error
*iov: a pointer to an arrary of iovec structure
struct iovec {
  void   *iov_base;   /* starting address of buffer */
  size_t   iov_len;     /* size of buffer */
};
```

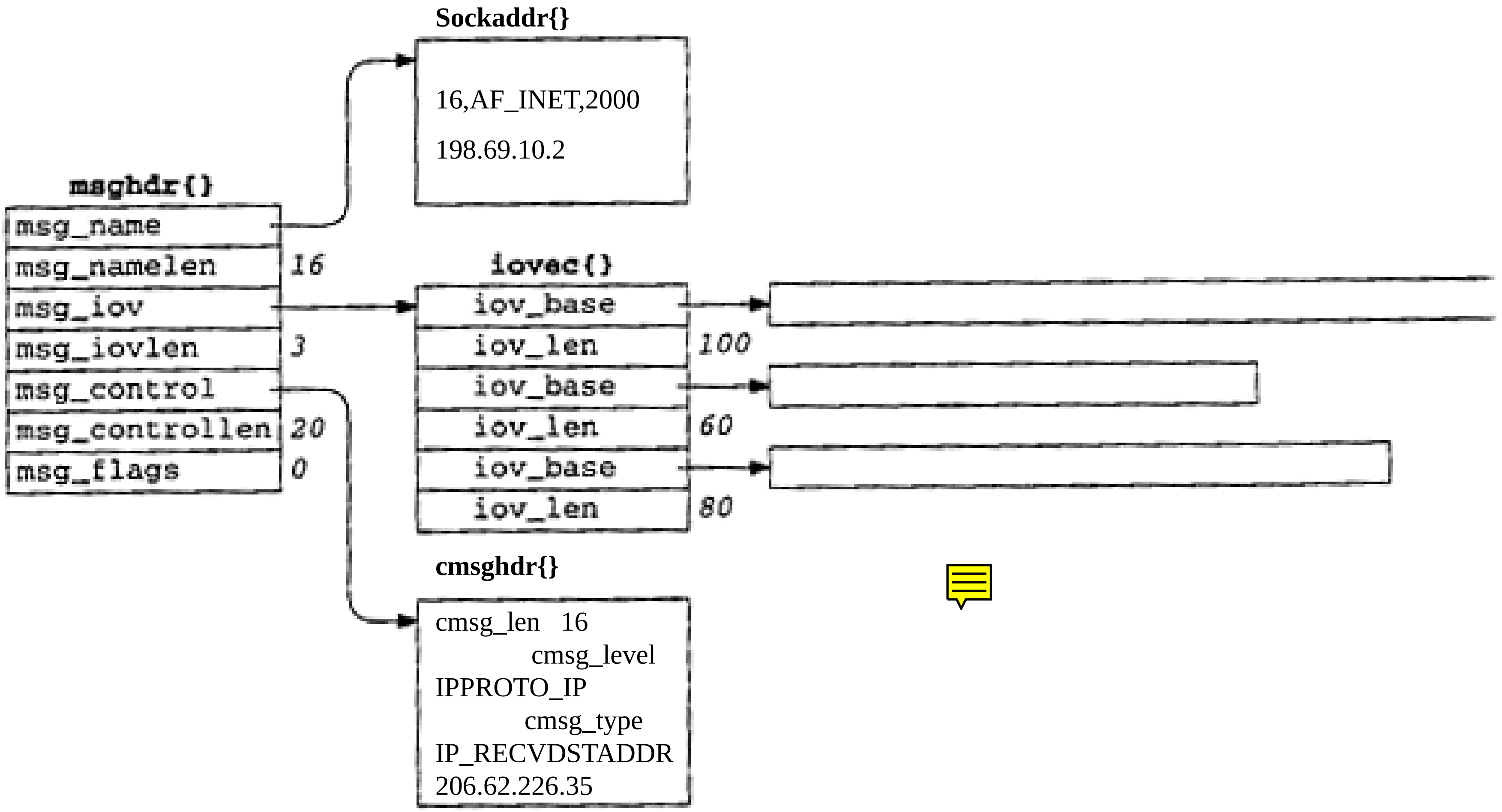
## recvmsg and sendmsg Functions the most general socket I/O functions

```
#include <sys/socket.h>
ssize_t recvmsg (int sockfd, struct msghdr *msg, int flags);
ssize_t sendmsg (int sockfd, struct msghdr *msg, int flags);
   both return: number of bytes read or written if OK, -1 on error
struct msghdr  {
   void     *msg_name;   /* protocol address */
   socklen_t    msg_namelen;   /* size of protocol address */
   struct iovec   *msg_iov;   /* scatter/gather array */
   size_t     msg_iovlen;   /* # elements in msg_iov */
   void      *msg_control;   /* ancillary data; must be aligned
           for a cmsghdr structure */
   socklen_t   msg_controllen;   /* length of ancillary data */
   int     msg_flags;   /* flags returned by recvmsg ( ) */
};
```

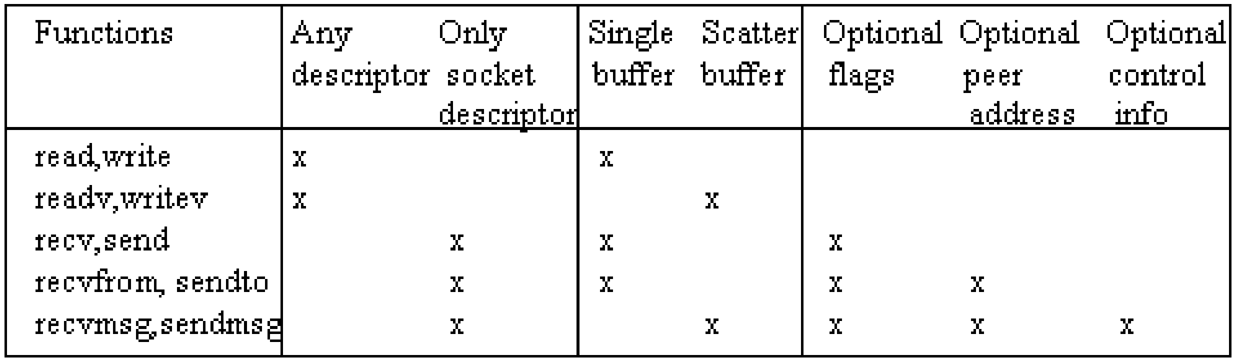
# Summary of I/O Flags by Various I/O Functions

| Flag | Examined by: *send flags* *sendto flags* *sendmsg flags* | Examined by: *recv flags* *recvfrom flags* *recvmsg flags* | Returned by: recvmsg msg_flags |
|---|---|---|---|
| MSG_DONTROUTE | X | | |
| MSG_DONTWAIT | X | X | |
| MSG_PEEK | | X | |
| MSG_WAITALL | | X | |
| MSG_EOR | X | | X |
| MSG_OOB | X | X | X |
| MSG_BCAST | | | X |
| MSG_MCAST | | | X |
| MSG_TRUNC | | | X |
| MSG_CTRUNC | | | X |

# msghdr When recvmsg Returns

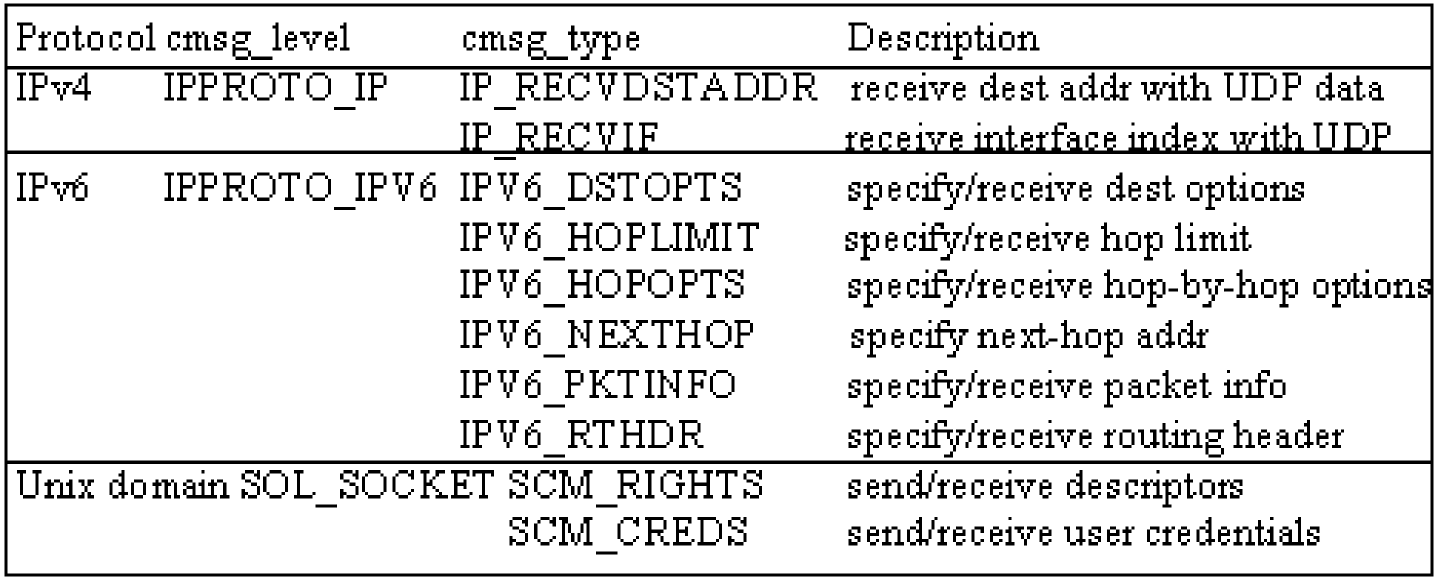


## Comparison of Five Groups of I/O Functions



| Functions | Any descriptor | Only socket descriptor | Single buffer | Scatter buffer | Optional flags | Optional peer address | Optional control info |
|---|---|---|---|---|---|---|---|
| read,write | x | | x | | | | |
| readv,writev | x | | | x | | | |
| recv,send | | x | x | | x | | |
| recvfrom, sendto | | x | x | | x | x | |
| recvmsg,sendmsg | | x | | x | x | x | x |

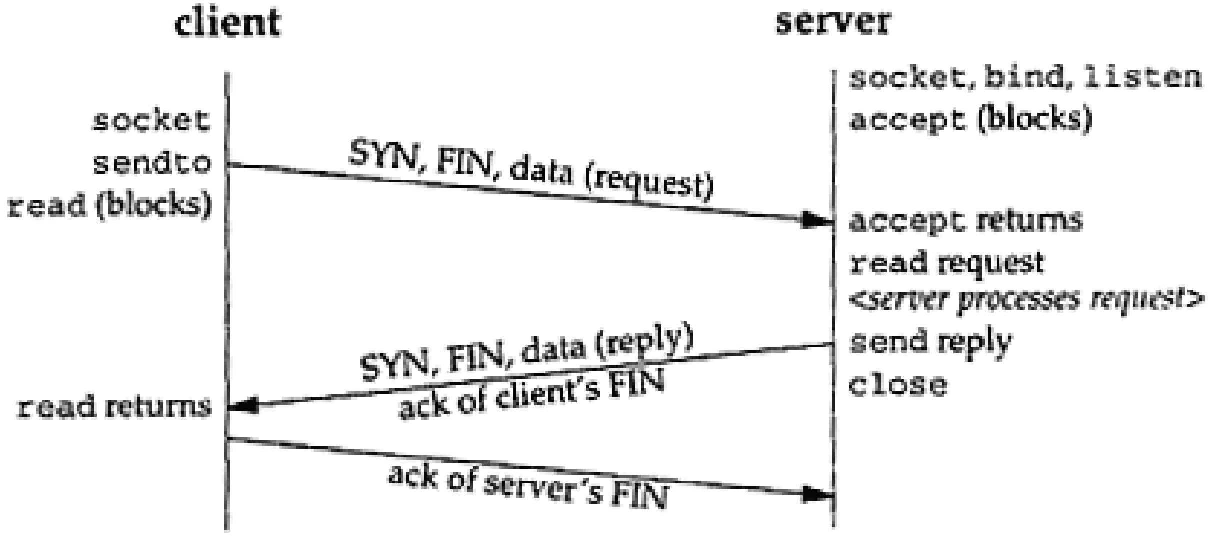
# Ancillary Data (Control Info)



| Protocol | cmsg_level | cmsg_type | Description |
|---|---|---|---|
| IPv4 | IPPROTO_IP | IP_RECVDSTADDR | receive dest addr with UDP data |
| | | IP_RECVIF | receive interface index with UDP |
| IPv6 | IPPROTO_IPV6 | IPV6_DSTOPTS | specify/receive dest options |
| | | IPV6_HOPLIMIT | specify/receive hop limit |
| | | IPV6_HOPOPTS | specify/receive hop-by-hop options |
| | | IPV6_NEXTHOP | specify next-hop addr |
| | | IPV6_PKTINFO | specify/receive packet info |
| | | IPV6_RTHDR | specify/receive routing header |
| Unix domain | SOL_SOCKET | SCM_RIGHTS | send/receive descriptors |
| | | SCM_CREDS | send/receive user credentials |

# T/TCP: TCP for Transactions

avoid three-way handshake between hosts that have communicated with each other recently



3 segments for T/TCP, compared to 10 for TCP and 2 for UDP