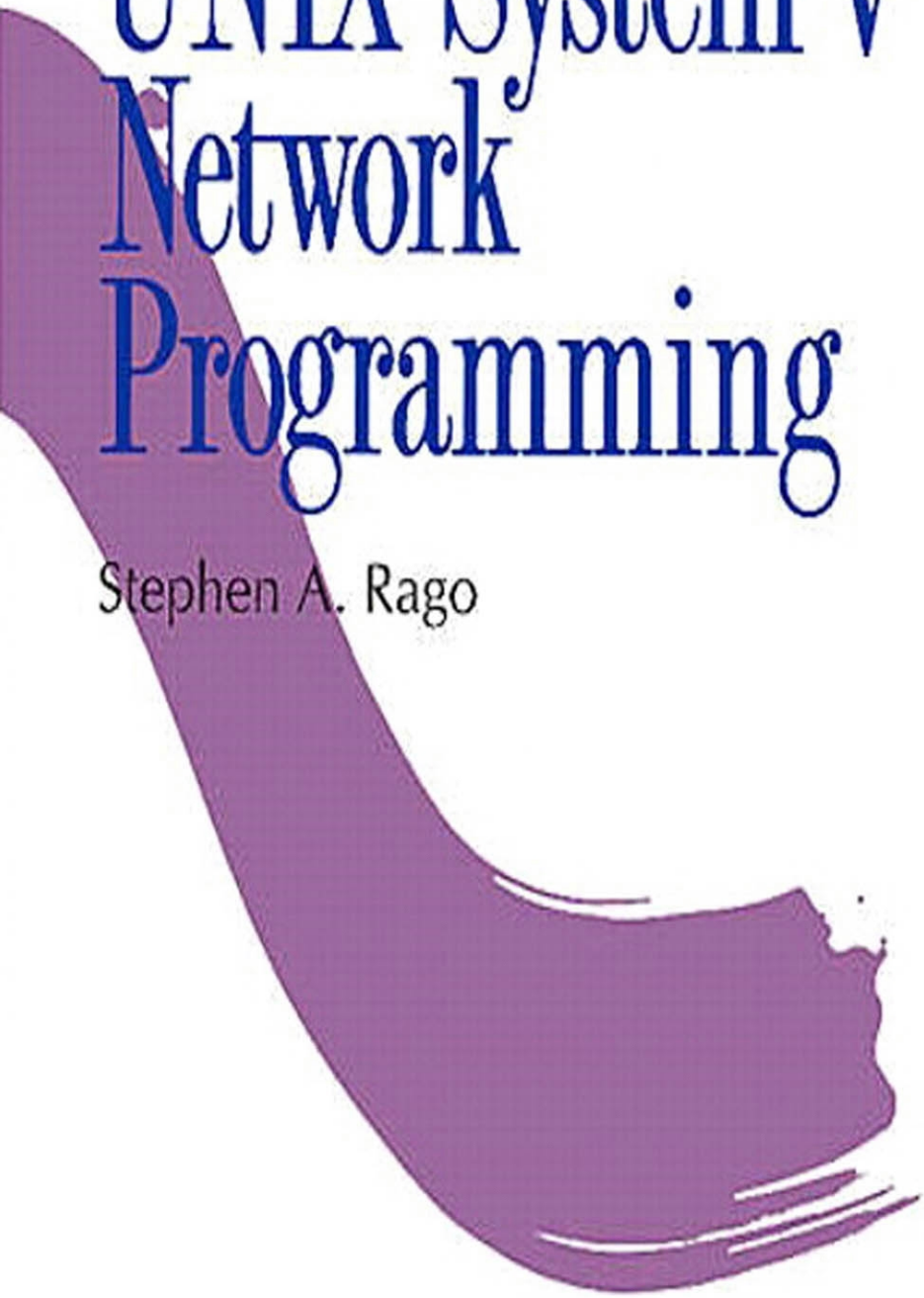
UNIX® System V Network Programming

Stephen A. Rago



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

UNIX® System V Network Programming

Stephen A. Rago

ADDISON-WESLEY PUBLISHING COMPANY
Reading, Massachusetts Menlo Park, California New York Don Mills, Ontario
Wokingham, England Amsterdam Bonn Paris Milan Madrid Sydney Tokyo
Seoul Taipei Mexico City San Juan

```
        protocol family = inet
        protocol = icmp
        device = /dev/icmp
        nlookups = 1
        directory lookup libraries:
            /usr/lib/tcpip.so
$
```

□

The NETPATH Library Routines

The second set of routines that access the network-configuration database is the NETPATH library [see getnetpath(3N)]. These routines provide users with a way of customizing the network search list by setting the NETPATH environment variable. Its format is similar to the shell's PATH variable: a list of names separated by colons. Unlike PATH, however, the names are network identifiers instead of pathnames.

The order of precedence of networks to select is from left to right. If a user included

```
NETPATH=osi_tp4:tcp:udp export NETPATH
```

in his or her .profile, any commands using the NETPATH library routines would use the OSI TP4 protocol and network before trying the Internet TCP or UDP protocol and network.

Note that protocols with different service types can be mixed together in the NETPATH. Applications are responsible for selecting protocols with the required service type by inspecting the nc_semantics field of the netconfig structure.

To enable applications to search /etc/netconfig based on NETPATH, setnetpath must first be called. It is similar to setnetconfig, performing data structure allocation and initialization and returning a handle. Just as endnetconfig should be called to clean up, endnetpath can be used to deallocate the data structures allocated by the call to setnetpath.

```
#include <netconfig.h>

void *setnetpath(void);
int endnetpath(void *handle);
```

Once setnetpath is called, applications can use getnetpath to obtain a netconfig structure for the next entry in the database file.

```
#include <netconfig.h>

struct netconfig *getnetpath(void *handle);
```

handle is the handle returned from a prior call to setnetpath. The next entry returned is determined by two criteria. First, if the caller's NETPATH environment variable includes the network ID for the next entry, then the entry is returned. Second, if the caller's NETPATH environment variable is uninitialized, the next entry is returned if it is visible in /etc/netconfig (that is, if the v flag is

present in the third column of the database).

Example 5.2.2. The following program is a rewrite of the one presented in Example 5.2.1, with the `netconfig` library routines replaced by the `NETPATH` routines.

```
/* netpath.c */
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>
#include <netconfig.h>

void
main()
{
     struct netconfig *ncp;
     void *p;
     int i;

     /*
      * Gain access to /etc/netconfig.
      */
     if ((p = setnetpath()) == NULL) {
         nc_perror("setnetpath failed");
         exit(1);
     }

     /*
      * Get each entry as specified by NETPATH,
      * and print the information.
      */
     while ((ncp = getnetpath(p)) != NULL) {
         printf("network: %s\n\tsemantics = ",
           ncp->nc_netid);
         switch (ncp->nc_semantics) {
         case NC_TPI_CLTS:
             printf("connectionless transport\n");
             break;

         case NC_TPI_COTS:
             printf("connection-oriented transport\n");
             break;

         case NC_TPI_COTS_ORD:
             printf("connection-oriented transport\n");
             printf("\t\t\twith orderly release\n");
             break;

         case NC_TPI_RAW:
             printf("raw transport\n");
             break;

         default:
             printf("unknown (%d)\n", ncp->nc_semantics);
         }
```

```
            printf("\tflags =");
            if (ncp->nc_flag == NC_NOFLAG)
                printf(" none\n");
            else if (ncp->nc_flag & NC_VISIBLE)
                printf(" visible\n");
            else
                printf(" unknown (0x%x)\n", ncp->nc_flag);
            printf("\tprotocol family = %s\n",
              ncp->nc_protofmly);
            printf("\tprotocol = %s\n", ncp->nc_proto);
            printf("\tdevice = %s\n", ncp->nc_device);
            printf("\tnlookups = %d\n", ncp->nc_nlookups);
            if (ncp->nc_nlookups > 0) {
                printf("\tdirectory lookup libraries:\n");
                for (i = 0; i < ncp->nc_nlookups; i++)
                    printf("\t\t%s\n", ncp->nc_lookups[i]);
            }
        }
        exit(0);
}
```

Although this example is almost the same as the last, its behavior is noticeably different. If the system has the SVR4 Internet protocol suite installed and we run the program without defining `NETPATH`, we get the following output:

```
$ unset NETPATH
$ ./netpath
network: ticlts
     semantics = connectionless transport
     flags = visible
     protocol family = loopback
     protocol = -
     device = /dev/ticlts
     nlookups = 1
     directory lookup libraries:
         /usr/lib/straddr.so
network: ticots
     semantics = connection-oriented transport
     flags = visible
     protocol family = loopback
     protocol = -
     device = /dev/ticots
     nlookups = 1
     directory lookup libraries:
         /usr/lib/straddr.so
network: ticotsord
     semantics = connection-oriented transport
                     with orderly release
     flags = visible
     protocol family = loopback
     protocol = -
     device = /dev/ticotsord
     nlookups = 1
     directory lookup libraries:
```

```
          /usr/lib/straddr.so
network: tcp
     semantics = connection-oriented transport
                 with orderly release
     flags = visible
     protocol family = inet
     protocol = tcp
     device = /dev/tcp
     nlookups = 1
     directory lookup libraries:
          /usr/lib/tcpip.so
network: udp
     semantics = connectionless transport
     flags = visible
     protocol family = inet
     protocol = udp
     device = /dev/udp
     nlookups = 1
     directory lookup libraries:
          /usr/lib/tcpip.so
$
```

The only networks found were those marked "visible" in `/etc/netconfig`. The order found is the order in which they appear in the database file.

If we set our `NETPATH` to a network that is not installed in the system, we get the following behavior:

```
$ NETPATH=starlan ./netpath
$
```

The command did not generate any output. This means that it could not find any network identifier equal to `starlan`. The `NETPATH` variable overrides the default network list found in `/etc/netconfig`. We get the same results if we set `NETPATH` to an empty string, as in

```
$ NETPATH=
$ ./netpath
$
```

To illustrate this point further, let's set our `NETPATH` variable to two network identifiers from `/etc/netconfig`, but in the opposite order from that in which they appear in the file. The output is

```
$ NETPATH=udp:tcp ./netpath
network: udp
     semantics = connectionless transport
     flags = visible
     protocol family = inet
     protocol = udp
     device = /dev/udp
     nlookups = 1
     directory lookup libraries:
          /usr/lib/tcpip.so
network: tcp
     semantics = connection-oriented transport
```

```
                        with orderly release
        flags = visible
        protocol family = inet
        protocol = tcp
        device = /dev/tcp
        nlookups = 1
        directory lookup libraries:
            /usr/lib/tcpip.so
$
```

Just as we expected, the `NETPATH` variable overrides the order of networks found in `/etc/netconfig`. Even though `tcp` appears before `udp` in `/etc/netconfig`, we find `udp` first, since it appears first in our `NETPATH` environment variable.

If we set the `NETPATH` variable to a network identifier that is in `/etc/netconfig` but is not marked visible, we get the information for the invisible entry. This behavior might surprise some users, but the absence of the visible flag does not prevent an entry from being returned if the caller's `NETPATH` variable explicitly includes that network's ID. (Thus, the visible flag is really a "default" flag — "visible" is a misnomer.) For example, ICMP is marked as being invisible in the database file. Using its network identifier, we get the following output:

```
$ NETPATH=icmp ./netpath
network: icmp
        semantics = raw transport
        flags = none
        protocol family = inet
        protocol = icmp
        device = /dev/icmp
        nlookups = 1
        directory lookup libraries:
            /usr/lib/tcpip.so
$
```

□

The `NETPATH` library routines are better suited to client-style applications, since this is usually the type of program run by users. Daemons (server-style applications), on the other hand, will probably use the network-configuration library routines because daemons are not usually associated with a particular user, and the `NETPATH` variable might not be set properly when the daemons are run. There are no fixed rules regarding usage, however. A client-style application that wants to run uninfluenced by a user's `NETPATH` environment variable could, for example, use the network-configuration routines.

5.3 NAME-TO-ADDRESS TRANSLATION

The name-to-address translation facility, also called *name-to-address mapping*, provides applications with a way to find an address corresponding to a service, and vice versa. The name-to-address translation facility relies on the network-selection library to provide information about the networks and protocols. Specifically, field 7

of /etc/netconfig contains the list of shared libraries that implement the network- and protocol-dependent portions of the generic name-to-address mapping library routines.

With SVR4, two shared libraries are provided. They both use information stored in ASCII files to perform name-to-address translation. The system administrator is responsible for maintaining these files.

/usr/lib/tcpip.so contains the routines that translate between services and addresses for the Internet protocol suite. It assumes that host names and their corresponding addresses are stored in /etc/inet/hosts, and service names and their corresponding port numbers are stored in /etc/inet/services. The addressing information returned is in the format of a sockaddr_in structure, defined in <netinet/in.h>.

The second library is /usr/lib/straddr.so. It can be used with any network whose addresses are character strings. It assumes host names and their corresponding addresses are stored in /etc/net/*transport*/hosts, and service names and their corresponding port numbers are stored in /etc/net/*transport*/services, where *transport* is the name of the transport provider for the network. The addressing information returned is in the form of *hostaddress.serviceport*. The loopback transport providers found in SVR4 use strings for addresses.

Each network product is expected to provide its own shared libraries that handle the translation details if the default two libraries do not suffice. The generic name-to-address translation library routines are provided in the network services library (linked with the -lnsl option). The generic routines provide the glue necessary to link the dynamic shared libraries into the address space of the calling process.

To convert from a host name and service name to an address, applications can use netdir_getbyname(3N) [see netdir(3N)]:

```
#include <netdir.h>

int netdir_getbyname(struct netconfig *ncp,
     struct nd_hostserv *hsp, struct nd_addrlist **alpp);
```

ncp is a pointer to a netconfig structure returned by one of the network-selection library routines. The nd_hostserv structure, which follows, defines the host and service whose address the caller is interested in obtaining.

```
struct nd_hostserv {
     char    *h_host;     /* host name */
     char    *h_serv;     /* service name */
};
```

The h_host field points to the name of the host computer, and the h_serv field points to the name of the service.

If successful, netdir_getbyname will return a list of addresses (through alpp) that can be used to connect to the host computer to obtain the desired service. The nd_addrlist structure is used to represent the addresses: