Name and Address Conversions

Ch. 11

Names vs. Numbers

- Use name for IP addresses and ports
- Easier to remember
- Names can stay the same while addresses and ports values could change
- Keep mapping:
  - Local: keep in a file on local host
  - Distributed: maintained by several hosts

DNS

- Domain Name System
- Distributed mapping and management
- Maps between names and IP addresses
  - Simple names (e.g., unix1)
  - Fully qualified domain name (FDQN), e.g., unix1.soe.ucsc.edu.
- Entries in DNS are known as resource records (RRs)

Domain Name System (DNS)

- Support user-friendly names for machines
  - Map logical names into IP addresses
- Early approach (i.e., /etc/hosts)
  - List all hostnames and IP addresses in a single file
  - Download regularly to other hosts
  - Could never handle the large set of hostnames in the Internet today, i.e., many more hosts and updates

DNS (Cont'd)

- Specifies name syntax and rules for delegating authority over names
  - Decentralized maintenance
- Specifies implementation of a distributed mapping of names to addresses
  - Wide-area distributed database
- Allows for mapping of additional object types
- RFC 1035

Decentralized Hierarchy

- Hierarchical namespace
  - Accommodates an arbitrarily large set of names with overwhelming a central site with administrative duties
  - Allows for growth of name space
  - Partitioned at the top level and authority for names and its subdivisions is passed to designated entities
  - E.g., local.site.top ('.' separator)
- Decentralizing the naming mechanism
  - Provides a resolution mechanism
  - Uses distributed databases
  - Servers operating at multiple sites cooperatively solve the mapping problem

DNS Name Hierarchy

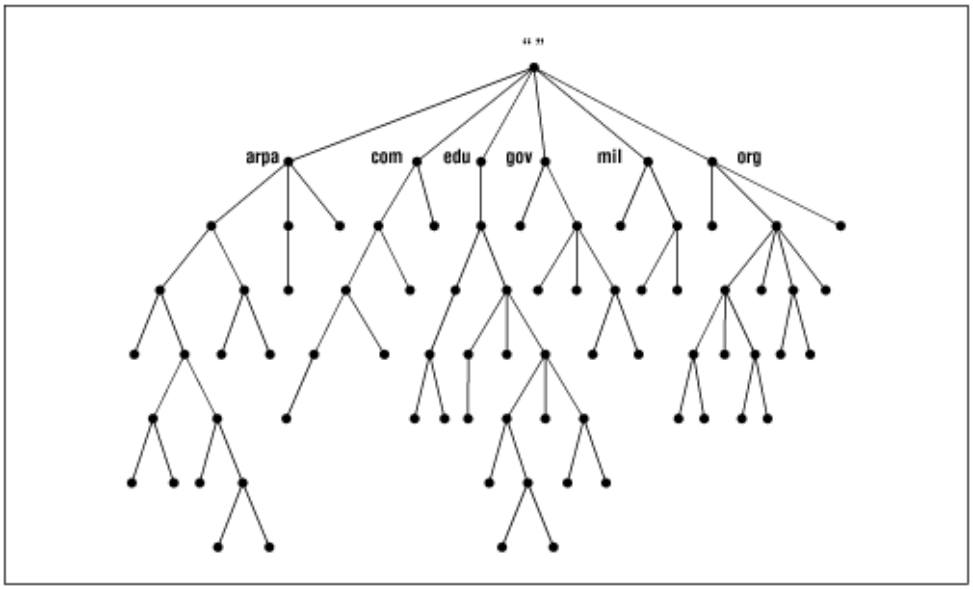
- DNS names don't generally follow the network topology
  - Independent of network routing hierarchy
  - Two machines attached to the same physical network may belong to two different administrative domains
- A domain (e.g., school.edu) can create its own subdomains (e.g., eng.school.edu)
- In practice, NS tree has few levels

Top-Level Domains (TLDs)

- Unnamed root "."
- Example official domain names

Domain Name	Meaning
COM	Commercial organizations
EDU	Educational institutions (4-year)
GOV	Government institutions
MIL	Military groups
NET	Major network support centers
ORG	Organizations other than those above
ARPA	Temporary ARPANET domain (obsolete)
INT	International organizations
country code	Each country (geographic scheme)

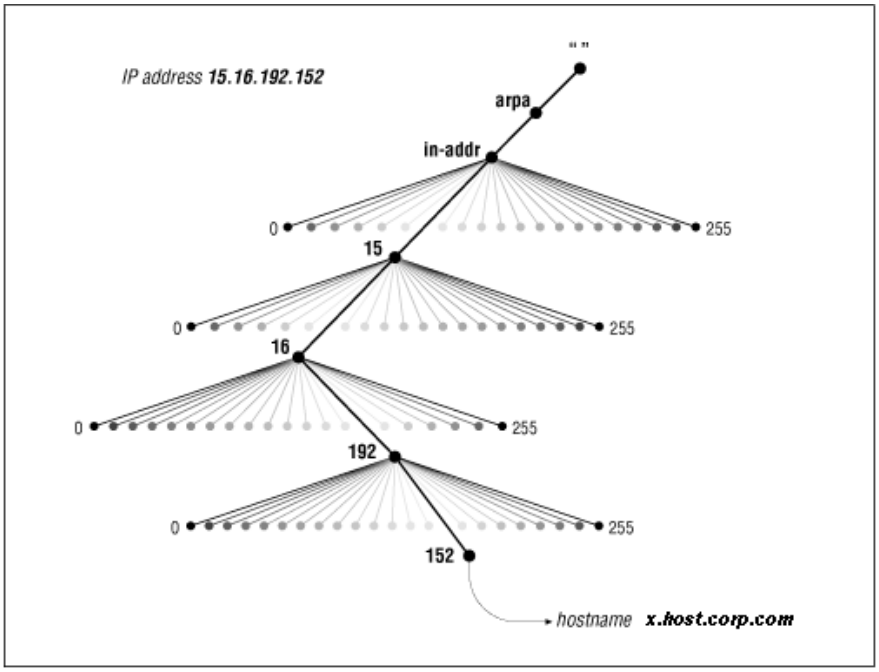
Name Hierarchy Tree



Inverse Mappings

- Name of address 15.16.192.152?
- Lookup 152.192.16.15.in-addr.arpa
  - Reversed decimals, why?
- Pointer queries

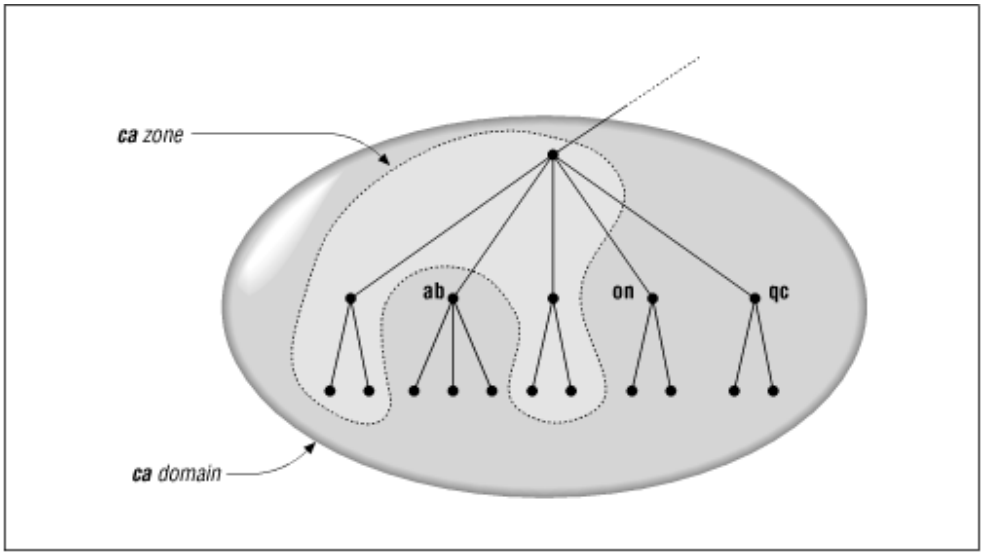
addr.arpa



DNS Zone

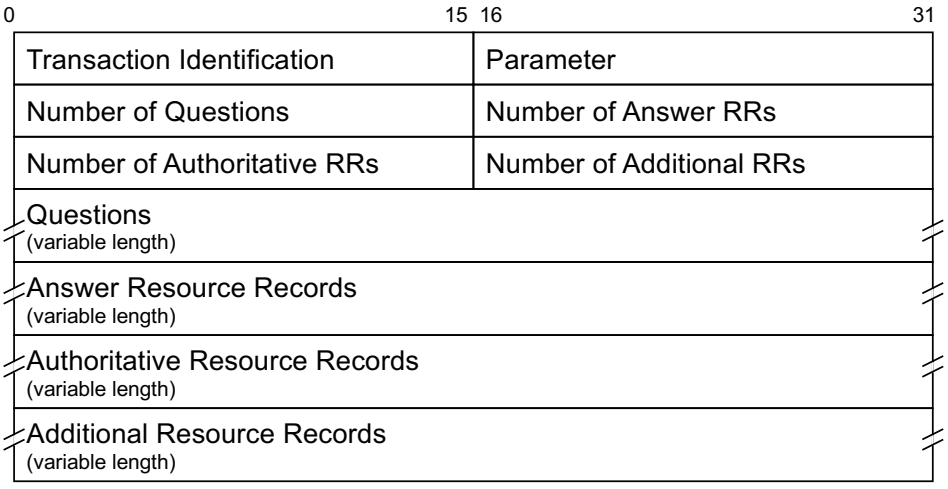
- One server can contain all the information for large parts of the naming hierarchy
- The hierarchy tree is partitioned into zones (sub-trees)
  - Independent administrative authorities
- Each zone has its own zone name server(s)
- Zone name server is responsible for resolving queries for all of its child nodes
- Actually, a hierarchy of name servers (NSs) and not of domains

Domain vs. Zone (E.g.)



Message Format

Variable length question, answer, authority and additional information sections



Message Format (Cont'd)

- Identification (2 bytes): match responses to queries
- Parameter (2 bytes): operation requested and response code
- Number of Questions (2 bytes)
- Number of Answers (2 bytes)
- Number of Authorities (2 bytes)
- Number of Additional (2 bytes)
  - Count of entries in corresponding sections
- Question section: contains queries for which answers are desired
- Other sections: contain a set of resource records

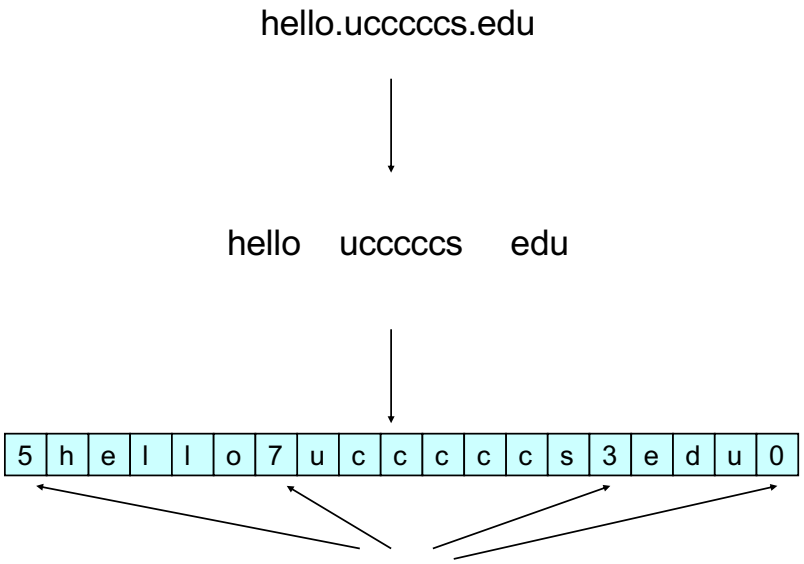
Parameter Field

- Bits numbered from left to right starting with zero

Bit of PARAMETER field	Meaning
0	Operation: 0 Query 1 Response
1-4	Query Type: 0 Standard 1 Inverse 2 Completion 1 (now obsolete) 3 Completion 2 (now obsolete)
5	Set if answer authoritative
6	Set if message truncated
7	Set if recursion desired
8	Set if recursion available
9-11	Reserved
12-15	Response Type: 0 No error 1 Format error in query 2 Server failure 3 Name does not exist

Encoding Query Names

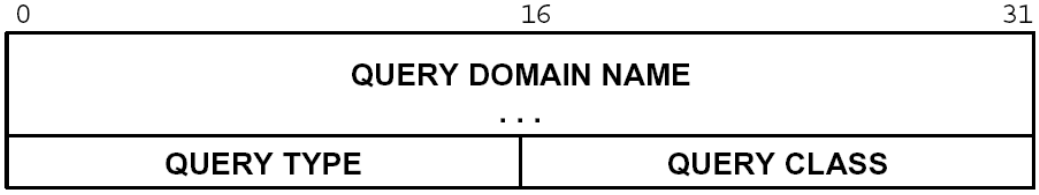
Example



NOTE: These count fields are not the ASCII characters “5”, “7”, “3” and “0”!!!

Question Section

- 1..N bytes: Query Domain Name
  - No padding used
  - E.g., label like example.com
- 2: Query Type: machine name or mail address (e.g., A)
- 2: Query Class (e.g., IN)



RR Types

- **A** maps a hostname to IPv4
- **AAAA** “quad A” maps hostname to IPv6
- **MX** “mail exchanger” can have preferences
  - Start with smallest value
- **CNAME** is “canonical name” for service names
  - Like an alias (e.g., ftp)
  - Allows for moving services between hosts

```
freebsd    IN    A      12.106.32.254
           IN    AAAA   3ffe:b80:1f8d:1:a00:20ff:fea7:686b
           IN    MX     5  freebsd.unpbook.com.
           IN    MX     10 mailhost.unpbook.com.
ftp        IN    CNAME  linux.unpbook.com.
www        IN    CNAME  linux.unpbook.com.
linux.unpbook.com.   IN    A 10.10.10.10
```

RR Types

- **PTR** “pointer records” maps IP into hostname
  - E.g., lookup 254.32.106.12.in-addr.arpa

```
1.100.0.192.in-addr.arpa.  IN PTR  lab1.school.edu.
2.200.0.192.in-addr.arpa.  IN PTR  lab2.school.edu.
```

Resolution

- Each host has a resolver
  - Typically a library that applications link in
- Local name server configured
  - Manually, i.e., /etc/resolve.conf
  - DHCP
- Client needs to know of at least one NS to contact

Resolution (Cont'd)

- Client "resolver" checks with the local NS
  - Queries consist of domain name to be resolved
  - Query class (e.g., internet)
  - Type of object (e.g., address)
- NS use well-known protocol ports for all communication with the clients (port=53)
- If not resolved, local NS refers to root NS
  - Well-known roots: [a-m].root-servers.net

Resolution (Cont'd)

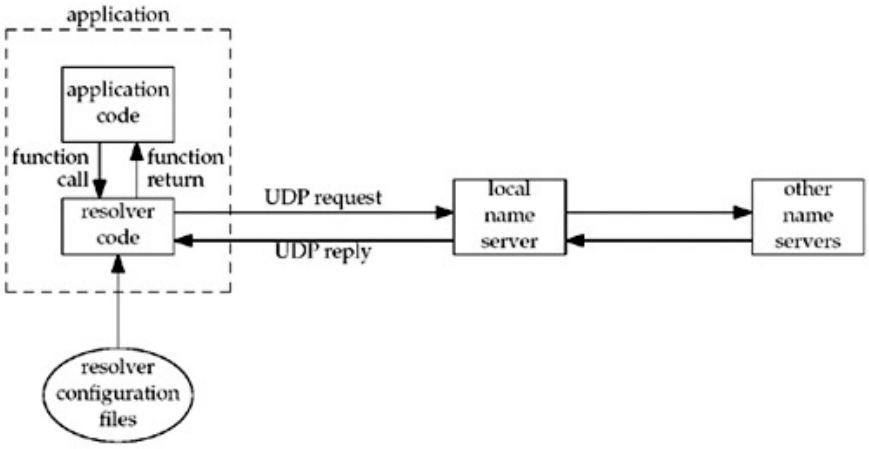
- NS checks to see if name lies in its subdomain for which it is an authority
- If not fully resolved by the NS, NS replies with the IP address of its child NS
- Can return multiple addresses

Lookup Methods

- Iterative
  - Server responds with its information for which NS to ask next
  - Partial answer
- Recursive
  - Server returns the complete information
  - Either final answer or "Not Found"
- Local NS typically does recursive lookup

Resolvers and DNS

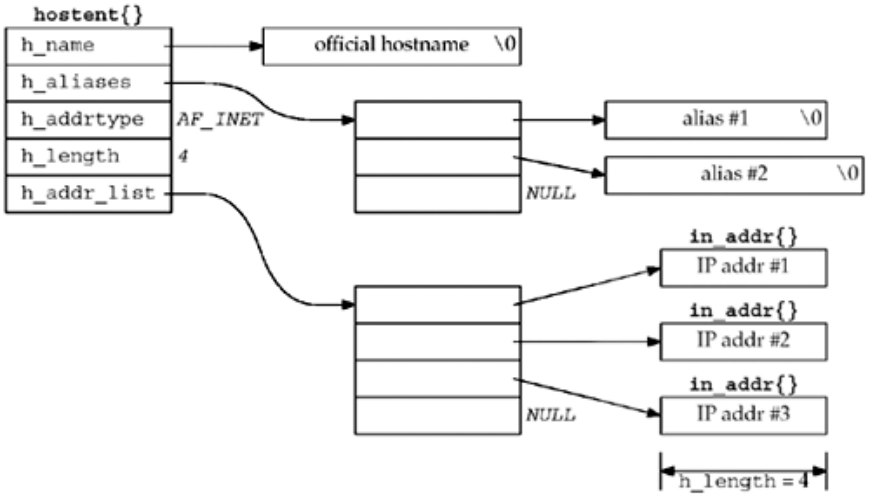
- Run one or more name servers (per domain)
- Resolver library for clients



gethostbyname()

Get canonical name, aliases and IP addresses

```
#include <netdb.h>
struct hostent *gethostbyname (const char *hostname);
```



(cont'd)

- Performs lookup for an A RR
- Null terminated strings for names

```
struct hostent {
  char  *h_name;      /* official (canonical) name of host */
  char **h_aliases;    /* pointer to array of pointers to alias names */
  int    h_addrtype;    /* host address type: AF_INET */
  int    h_length;       /* length of address: 4 */
  char **h_addr_list;  /* ptr to array of ptrs with IPv4 addrs */
};
```

- gethostbyaddr() does ptr lookup on an `in_addr` structure (*addr*)

```
struct hostent *
gethostbyaddr(const void *addr, socklen_t len, int type);
```

getservbyname()

- Services (e.g, 'ftp') known by name

```
#include <netdb.h>
struct servent *getservbyname (const char *name, const char *proto);
```

```
struct servent {
  char   *s_name;      /* official service name */
  char  **s_aliases;   /* alias list */
  int       s_port;       /* port number, network-byte order */
  char    *s_proto;     /* protocol to use */
};

struct servent *sptr;

sptr = getservbyname("domain", "udp"); /* DNS using UDP */
sptr = getservbyname("ftp", "tcp");         /* FTP using TCP */
sptr = getservbyname("ftp", NULL);         /* FTP matching any protocol */
sptr = getservbyname("ftp", "udp");        /* this call will fail */
```

getservbyport()

Reverse lookup given port number

```
#include <netdb.h>
struct servent *getservbyport (int port, const char *protoname);
```

```
struct servent *sptr;

sptr = getservbyport (htons (53), "udp");  /* DNS using UDP */
sptr = getservbyport (htons (21), "tcp");   /* FTP using TCP */
sptr = getservbyport (htons (21), NULL);   /* FTP using TCP */
sptr = getservbyport (htons (21), "udp");  /* this call will fail */
```

Posix Name/Address Conversion

- Traditionally, **gethostbyname()** and **gethostbyaddr()** were used
  - These depend on IPv4 AF_INET family
  - Not part of sockets library
- Posix includes protocol independent functions:
  - **getaddrinfo()**, **getnameinfo()**

getaddrinfo(), getnameinfo()

- These functions provide name/address conversions as part of the sockets library
- It's important to write code that can run on many protocols, i.e., IPv4, IPv6
- Handle protocol dependence in *library*
  - Same code can be used for many protocols
- Re-entrant function (vs. gethostbyname)
  - Important to threaded applications

getaddrinfo()

- Replaces both gethostbyname() and getservbyname()
- Returns linked lists of *addrinfo*

```
int getaddrinfo(
    const char *hostname,
    const char *service,
    const struct addrinfo *hints,
    struct addrinfo **result);
```

`getaddrinfo()` parameters

- *hostname* is a hostname or an address string (dotted decimal string for IP).
- *service* is a service name or a decimal port number string.

hints

- *hints* is an addrinfo* that can contain:
  - ai_flags (AI_PASSIVE, AI_CANONNAME)
  - ai_family (AF_XXX)
  - ai_socktype (SOCK_XXX)
  - ai_protocol (IPPROTO_TCP, etc.)
- If it is NULL, assumes value of AF_UNSPEC for ai_family and 0 for the rest in above

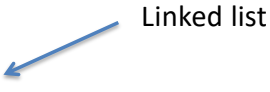
hints Uses

- If interested only TCP and not UDP, for example, then the *ai_protocol* member of the hints structure should be set to IPPROTO_TCP
- If only IPv4 and not IPv6, then the *ai_family* member of the hints structure should be set to AF_INET
- If hostname argument is NULL
  - If the AI_PASSIVE flag is given the returned address information shall be suitable for use in binding a socket for accepting incoming connections for the specified service, i.e., return INADDR_ANY for IPv4
  - Else, loopback address

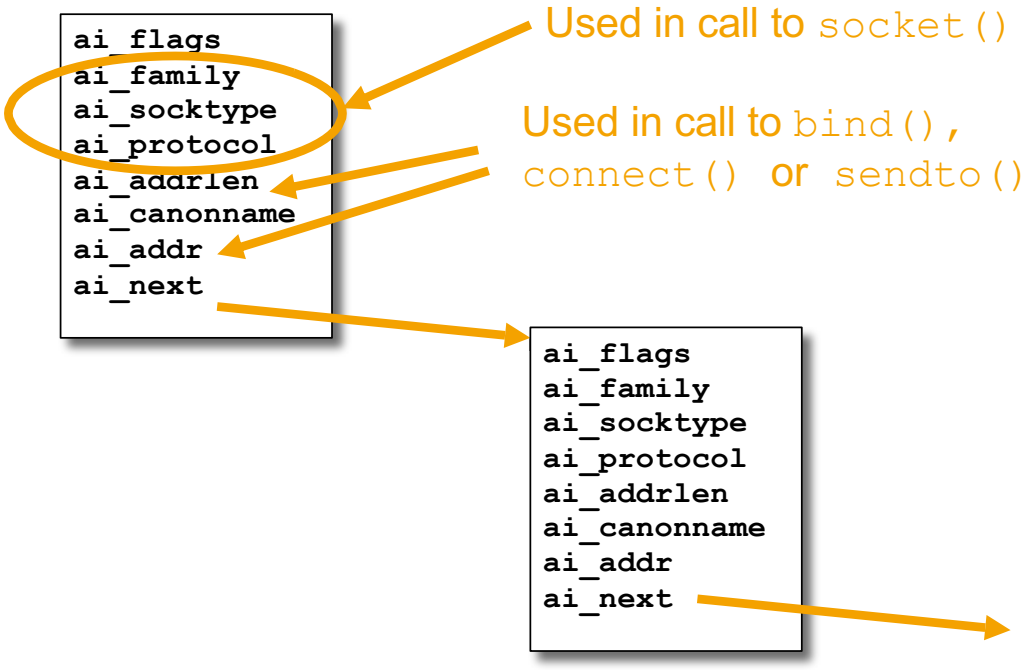
result

- Returned as a linked list by getaddrinfo

```
#include <netdb.h>
struct addrinfo {
  int         ai_flags;
  int         ai_family;
  int         ai_socktype;
  int         ai_protocol;
  socklen_t   ai_addrlen;
  char        *cannonname;
  struct sockaddr *ai_addr;
  struct addrinfo *ai_next;
};
```



addrinfo structure



freeaddrinfo()

Free the list of dynamic storage of *ai_addr* and *ai_canonname* returned by getaddrinfo()

```
#include <netdb.h>
void freeaddrinfo (struct addrinfo *ai);
```

```
struct addrinfo  hints, *res;

bzero(&hints, sizeof(hints) ) ;
hints.ai_flags = AI_CANONNAME;
hints.ai_family = AF_INET;

getaddrinfo("freebsd4", "domain", &hints, &res);
```

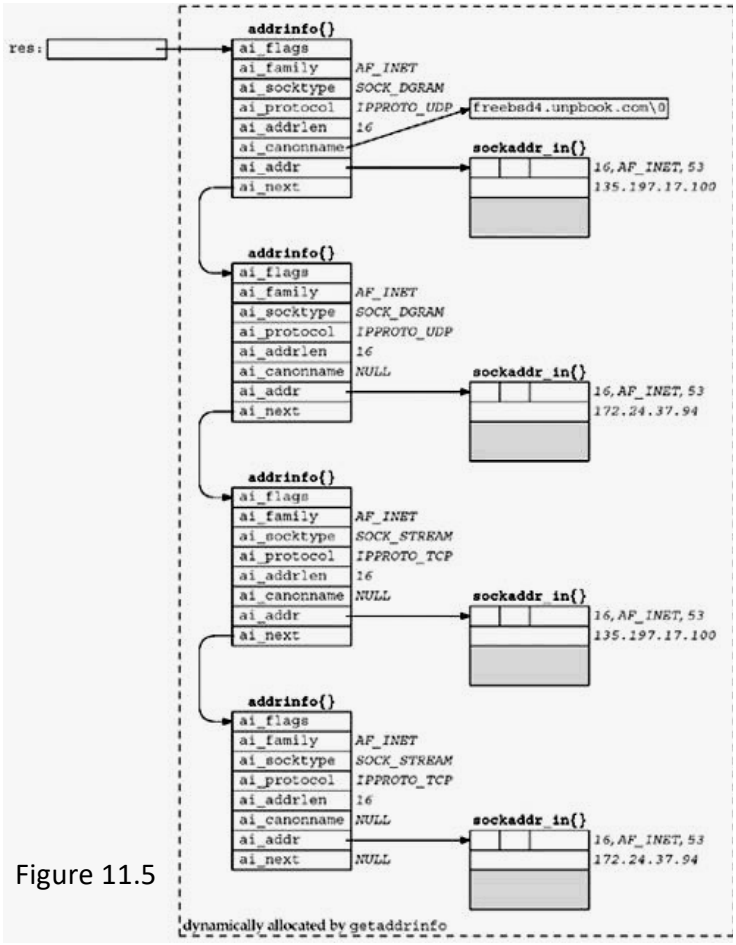


Figure 11.5

tcp_connect() Example

```
1  #include    "unp.h"

2  int
3  tcp_connect (const char *host, const char *serv)
4  {
5    int     sockfd, n;
6    struct addrinfo hints, *res, *ressave;

7    bzero(&hints, sizeof (struct addrinfo));
8    hints.ai_family = AF_UNSPEC;
9    hints.ai_socktype = SOCK_STREAM;

10   if ( (n = getaddrinfo (host, serv, &hints, &res)) != 0)
11      err_quit("tcp_connect error for %s, %s: %s",
12               host, serv, gai_strerror (n));
13   ressave = res;

14   do {
15      sockfd = socket (res->ai_family, res->ai_socktype, res->ai_protocol);
16      if (sockfd < 0)
17         continue;          /*ignore this one */

18      if (connect (sockfd, res->ai_addr, res->ai_addrlen) == 0)
19         break;             /* success */

20      Close(sockfd);         /* ignore this one */
21   } while ( (res = res->ai_next) != NULL);

22   if (res == NULL)           /* errno set from final connect() */
23      err_sys ("tcp_connect error for %s, %s", host, serv);

24   freeaddrinfo (ressave);

25   return (sockfd);
26 }
```

lib/tcp_connect.c
Fig 11.10

getnameinfo()

getnameinfo() looks up a hostname and a service name given a sockaddr

```
int getnameinfo(
   const struct sockaddr *sockaddr,
   socklen_t addrlen,
   char *host,
   size_t hostlen,
   char *serv,
   size_t servlen,
   int flags);
```