Adapting Your Network Code for IPv6

There's No Place like ::1/64

Mike Anderson

Chief Scientist

The PTR Group, Inc.

http://www.theptrgroup.com

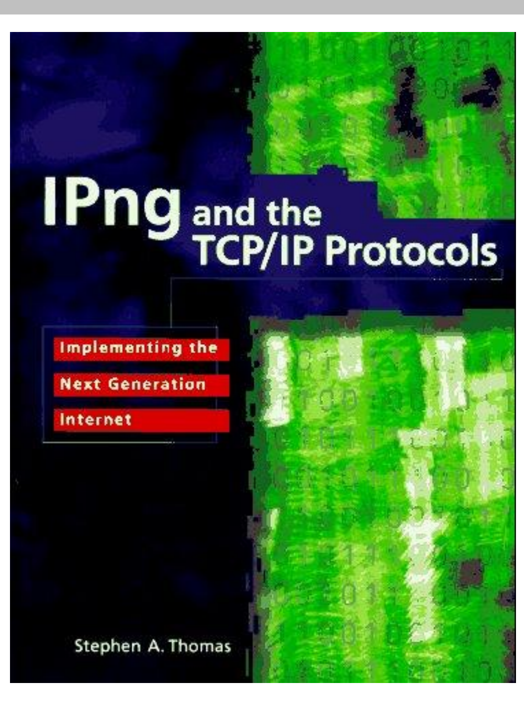
mailto: mike@theptrgroup.com

What We Will Talk About

- IPv6 history
- Why convert to IPv6?
- IPv6 Addressing
- Coexisting with IPv4
- IPv6 commands
- Typical server/client code flow
- IPv4 vs. IPv6 APIs
- Transitioning to IPv6 and testing your readiness
- Summary

Copyright 2012, The PTR Group, Inc.

IPv6 History



- Back in the early 1990s, the IETF foresaw the exhaustion of the 32-bit IPv4 address space
  - IPng project was born in 1994
- IPv6 was finalized in December of 1998
  - RFC 2460
- There actually was a test framework known as IPv5
  - But, it was never deployed

Copyright 2012, The PTR Group, Inc.

IPv4 Address Issues

- IPv4 (RFC 791) uses a 32-bit address space
  - Seemed like enough in 1981
- Originally split into different "class" addresses
  - Class A (7/24), B (14/16), C (21/8)
- As we started to run out, the IETF introduced CIDR
  - Addresses were expressed in addr/X format
    - E.g., 192.168.101.130/25 (255.255.255.128)
  - NAT became the rule of the day

Copyright 2012, The PTR Group, Inc.

Characteristics of the IPv4 Internet

- Today's IPv4-based Internet is a confusing jumble of middle devices
  - Firewalls, NAT boxes, load balancers, VPN tunnel servers and more
- It's almost impossible to get to a particular device on the Internet directly
  - This either a bug or a feature depending on your perspective
- Each middle device introduces latency in communications
  - Frequent rewriting of packets as they transit the 'net

Copyright 2012, The PTR Group, Inc.

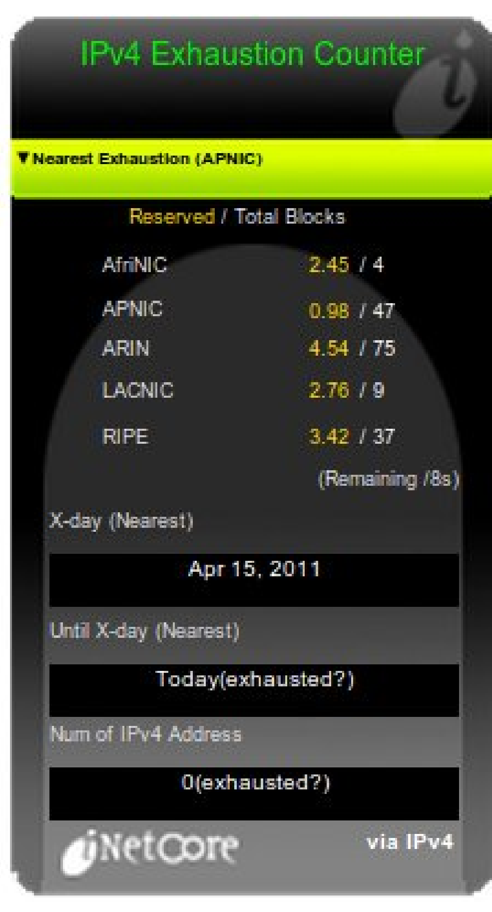
Reasons for Switching to IPv6

- We've run out of IPv4 addresses
- IPv6 is being mandated by most governments
  - We probably can't ignore this one forever ☺
- We want to regain end-to-end transparency
  - Reduction of latency is important for streaming media applications
- Core gateways are being over-burdened by address bloat
- IPv6 has security mechanisms built in
  - IPsec encryption

Copyright 2012, The PTR Group, Inc.

Whoops!

- After forecasting that we'd run out of addresses for the past decade, we finally did it!
- Did the Internet stop?
  - Nope
- However, the RIRs are getting aggressive about reclaiming unused address space
  - Not an issue if you're hiding behind a NAT box

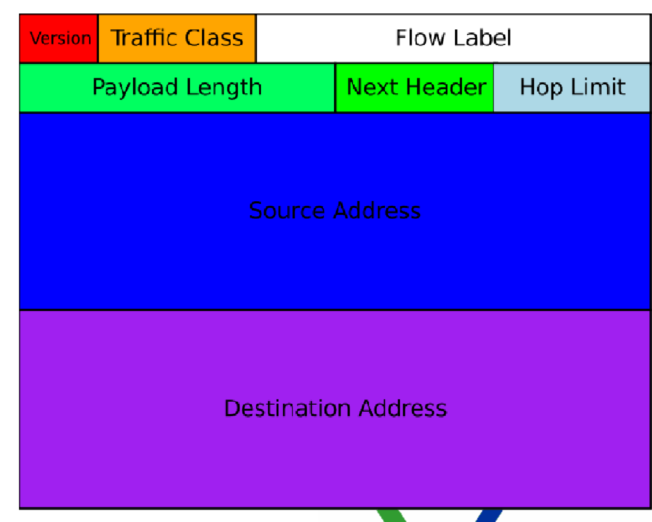


Copyright 2012, The PTR Group, Inc.

IPv6 is a Simpler Protocol

- IPv4 is a complex protocol
  - Many fields that need to be interrogated
- IPv6 has a fixed 40-octet length
  - IPv4 ranged from 20-60 octets
- IPv6 moved IPv4 options to additional headers

bit offset	0–3	4–7	8–13	14-15	16–18	19–31
0	Version	Header Length	Differentiated Services Code Point	Explicit Congestion Notification	Total Length	
32	Identification				Flags	Fragment Offset
64	Time to Live		Protocol		Header Checksum	
96	Source IP Address					
128	Destination IP Address					
160	Options (if Header Length > 5)					
160 or 192+	Data					



Copyright 2012, The PTR Group, Inc.
PTR

IPv6 Addresses

- IPv6 addresses are certainly more complex
  - 128-bit IPv6 vs. 32-bit IPv4
- Special addresses include:
  - ::1 (Loopback IPv4 127.0.0.1)
  - :: (unspecified a.k.a. 0.0.0.0/INADDR_ANY)
- IPv6 does not support broadcast
  - Only multicast
- IPv6 link-local addresses can be based on you hardware MAC address
  - MAC: 5c:26:0a:26:76:dc
  - IPv6: fe80::5e26:aff:fe26:76dc/64
    - EUI-64 address
- Auto assigned addresses via SLAAC or DHCP6

Copyright 2012, The PTR Group, Inc.

IPv6 Addresses #2

- Example (these are all equivalent):
  - 2008:0db8:0000:0000:0000:0000:1978:57ac
  - 2008:0db8:0000:0000:0000::1978:57ac
  - 2008:0db8:0:0:0:0:1978:57ac
  - 2008:0db8::1978:57ac
  - 2008:db8::1978:57ac
- An IPv6 address is enclosed in brackets
  - http://[2008:0db8::1978:57ac]/
  - https://[2008:0db8::1978:57ac]:443/
  - These things cry out for DNS
- Representation of IPv6 network in CIDR notation
  - 2008:0db8:1234::/48
    - 2008:0db8:1234:0000:0000:0000:0000:0000 through 2008:0db8:1234:ffff:ffff:ffff:ffff:ffff

Copyright 2012, The PTR Group, Inc.

IPv4/IPv6 Co-Existence

- For those O/Ses that support IPv6, most support "dual stack"
  - Both IPv4 and IPv6 are resident and can route packets
- If you have an IPv6 device and must route across IPv4, there are tunneling approaches
  - 6to4, Toredo, 6in4 and more
- There are also tunnel brokers
  - Tunnel endpoints to bypass IPv6-ignorant ISPs

Copyright 2012, The PTR Group, Inc.

IPv6 Commands

- Most of your favorite commands exist with a "6" appended
  - **`ping6`**, **`traceroute6`**, **`iptables6`**, etc.
- Many O/S variants already have IPv6 support
  - Linux, OS/X, Windows
- Some RTOSes support IPv6
  - VxWorks, ThreadX, QNX, OSE, LynxOS
  - However, many others do not…

Copyright 2012, The PTR Group, Inc.

Typical IPv4 Code Flow

- Server:
  - `socket(…)` – Opens a socket
  - `bind(…)` – Binds a local address to the socket
  - `listen(…)` – Advertise waiting on connections
  - `accept(…)` – Wait on the connections
  - If TCP `read(…)`/`write(…)` or `recv(…)`/`send(…)`
  - If UDP `recvfrom(…)`/`sendto(…)`
- Client:
  - `socket(…)` – Opens a socket
  - `connect()` – Connect to the server
  - If TCP `read(…)`/`write(…)` or `recv(…)`/`send(…)`
  - If UDP `recvfrom(…)`/`sendto(…)`

Copyright 2012, The PTR Group, Inc.

The Good News…

- The code flow for IPv6 is identical to that of IPv4
- The address structures in the API calls need to change to handle the 128-bit addresses
- The charges are related to those APIs that expose the size of the IP address or manipulate the address in some way
  - Especially, those that handle name to address resolution

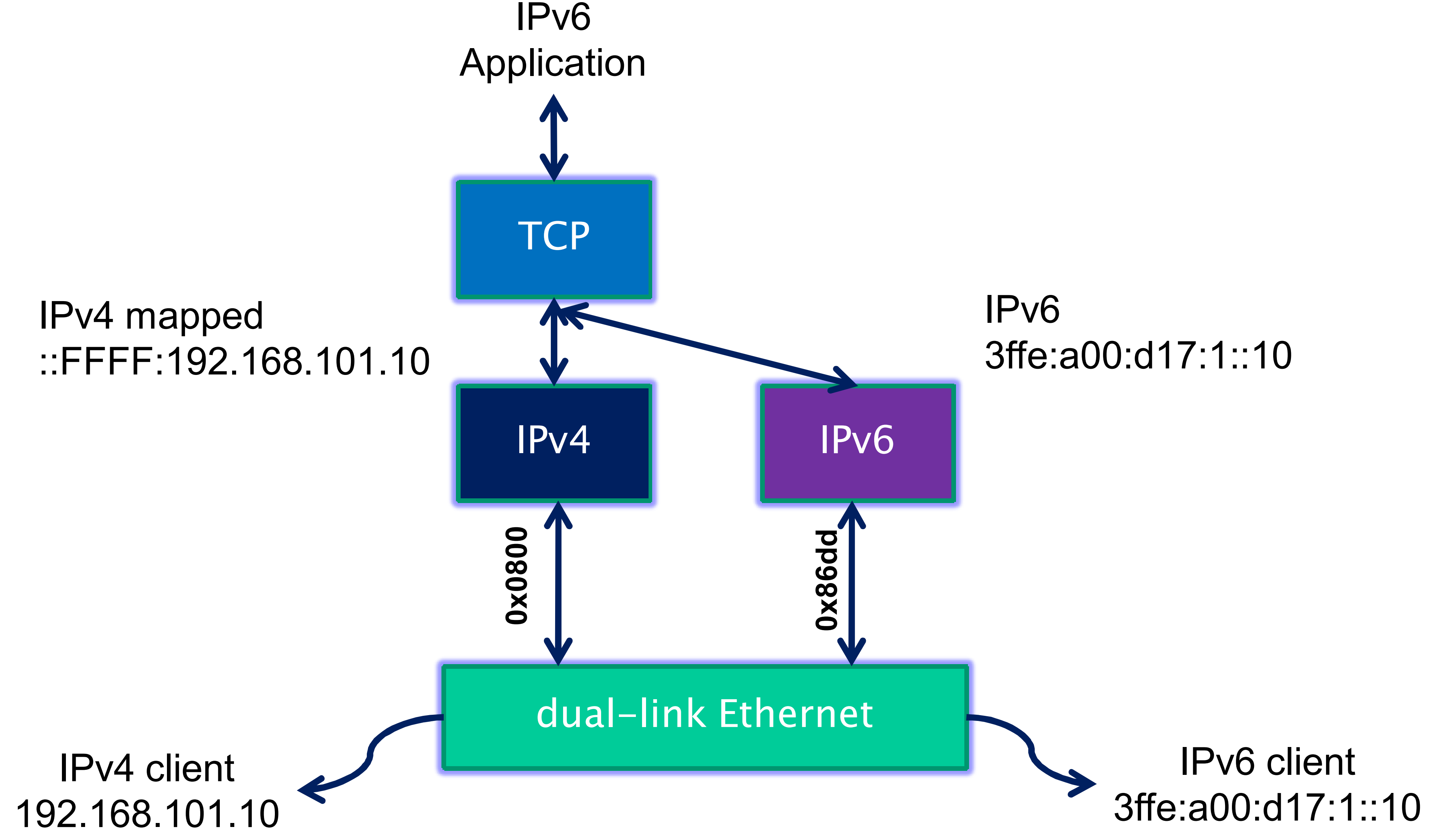
Copyright 2012, The PTR Group, Inc.

Strategies

- Since many O/Ses support dual stack, IPv4 code will continue to run for the foreseeable future
  - Therefore do nothing
- We could start developing IPv6-only code
  - The simplest conversion approach
- However, IPv4 is expected to still be with us for the next 15-20 years
  - So, we probably want to create IP-agnostic code
    - Can support either address type

Copyright 2012, The PTR Group, Inc.

Dual Stack Operation IPv6-Only



Copyright 2012, The PTR Group, Inc.

Porting Applications to IPv6-only

- As we've seen, IPv6 follows the same flow as IPv4 applications
  - The **`sockaddr_in`** structure becomes **`sockaddr_in6`**
  - Address family becomes **`AF_INET6`**/**`PF_INET6`**
  - Most of the rest of the calls stay the same
- If an application embeds the address in the protocol (e.g., FTP and NTPv3), then they need more rework

Copyright 2012, The PTR Group, Inc.

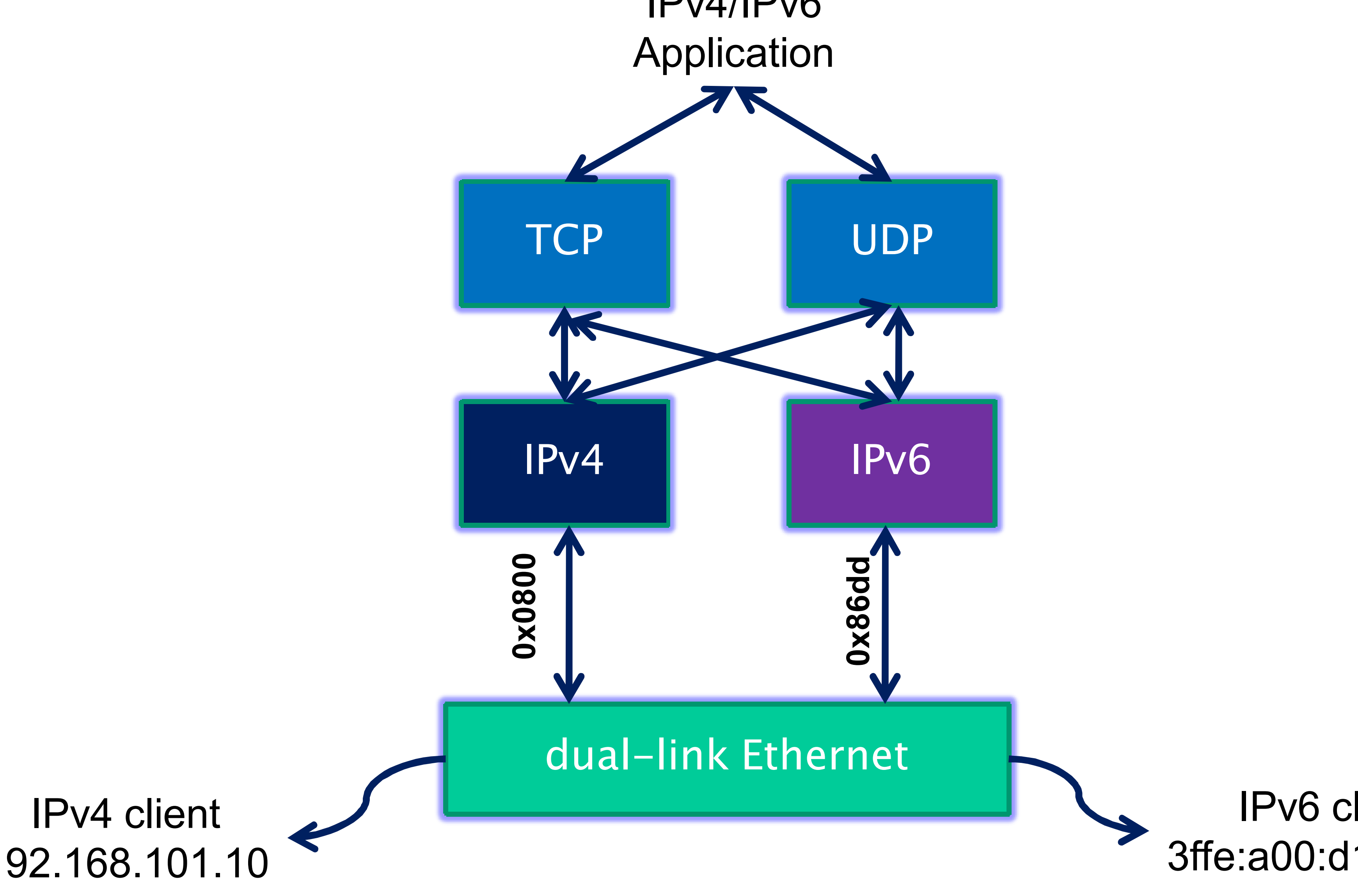
API Comparison

	IPv4 (AF_INET)	IPv6 (AF_INET6)
Data Structures	**PF_INET** **in_addr** **sockaddr_in** **sockaddr**	**PF_INET6** **in6_addr** **sockaddr_in6** **sock_storage**
Address conversion functions	**gethostbyname()** **gethostbyaddr()**	**getnameinfo()** **getaddrinfo()**
Name/address functions	**inet_aton()** **inet_addr()** **inet_ntoa()**	**inet_pton()** **inet_ntop()**

Red functions work with both IPv4 and IPv6

Copyright 2012, The PTR Group, Inc.

Dual Stack Operation IPv4/IPv6



Copyright 2012, The PTR Group, Inc.

PTR

IPv4 Structures

```
include <netinet/in.h>

// IPv4 AF_INET sockets:

struct sockaddr_in {
    short            sin_family;   // e.g. AF_INET, AF_INET6
    unsigned short   sin_port;     // e.g. htons(3490)
    struct in_addr   sin_addr;     // see struct in_addr, below
    char             sin_zero[8];  // zero this if you want to
};

struct in_addr {
    unsigned long s_addr;          // load with inet_pton()
};

// All pointers to socket address structures are often cast to pointers
// to this type before use in various functions and system calls:

struct sockaddr {
    unsigned short    sa_family;    // address family, AF_xxx
    char              sa_data[14];  // 14 bytes of protocol address
};
```


Copyright 2012, The PTR Group, Inc.

IPv6 Structures

```
// IPv6 AF_INET6 sockets:

struct sockaddr_in6 {
    u_int16_t       sin6_family;   // address family, AF_INET6
    u_int16_t       sin6_port;     // port number, Network Byte Order
    u_int32_t       sin6_flowinfo; // IPv6 flow information
    struct in6_addr sin6_addr;     // IPv6 address
    u_int32_t       sin6_scope_id; // Scope ID
};

struct in6_addr {
    unsigned char   s6_addr[16];   // load with inet_pton()
};


// General socket address holding structure, big enough to hold either
// struct sockaddr_in or struct sockaddr_in6 data:

struct sockaddr_storage {
    sa_family_t  ss_family;     // address family

    // all this is padding, implementation specific, ignore it:
    char      __ss_pad1[_SS_PAD1SIZE];
    int64_t   __ss_align;
    char      __ss_pad2[_SS_PAD2SIZE];
};
```


Copyright 2012, The PTR Group, Inc.

Example IPv4 Server Set Up

```
struct sockaddr addr;
int newFd;
int s = socket (PF_INET, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
struct sockaddr_in * ia = (struct sockaddr_in*) &addr;
ia->sin_family = AF_INET;
ia->sin_port = htons (5002);
bind (s, &addr, sizeof (struct sockaddr_in));
listen (s, 5);
while (1) {
  memset (&addr, 0, sizeof (addr));
  socklen_t alen = sizeof (struct sockaddr);
  newFd = accept (s, &addr, &alen);
  pthread_create (&pt, NULL, &process, (void *) &newFd);
}
```


Copyright 2012, The PTR Group, Inc.

Example IPv6 Server Set Up

```
struct sockaddr addr;
int newFd;
int s = socket (PF_INET6, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
struct sockaddr_in6 * ia = (struct sockaddr_in6*) &addr;
ia->sin6_family = AF_INET6;
ia->sin6_port = htons (5002);
bind (s, &addr, sizeof (struct sockaddr_in6));
listen (s, 5);
while (1) {
  memset (&addr, 0, sizeof (addr));
  socklen_t alen = sizeof (struct sockaddr);
  newFd = accept (s, &addr, &alen);
  pthread_create (&pt, NULL, &process, (void *) &newFd);
}
```


Copyright 2012, The PTR Group, Inc.

IPv4 Client Set Up

```
struct sockaddr addr;
struct sockaddr_in *ia;
int s = socket (PF_INET, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
ia = (struct sockaddr_in*) &addr;
ia->sin_family = AF_INET;
ia->sin_port = htons (5002);
ia->sin_addr.s_addr = htonl (INADDR_LOOPBACK);
connect (s, &addr, sizeof (struct
  sockaddr_in));
process(s);
close (s);
```


Copyright 2012, The PTR Group, Inc.

IPv6 Client Set Up

```
struct sockaddr addr;
struct sockaddr_in6 *ia;
int s = socket (PF_INET6, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
ia = (struct sockaddr_in6*) &addr;
ia->sin6_family = AF_INET6;
ia->sin6_port = htons (5002);
ia->sin6_addr.s6_addr = in6addr_loopback;
connect (s, &addr, sizeof (struct
  sockaddr_in6));
process(s);
close (s);
```


Copyright 2012, The PTR Group, Inc.

Name to Address Translation

- **getaddrinfo(...)**
  - Pass in string (address and/or port)
  - Optional hints for address family, type and protocol
    - Flags:
      - **AI_PASSIVE**, **AI_CANNONNAME**, **AI_NUMERICHOST**, **AI_NUMERICSERV**, **AI_V4MAPPED**, **AI_ALL**, **AI_ADDRCONFIG**
  - Returns a pointer to a linked list of **addrinfo** structures
    - Allocates memory for storing the returned addresses
- **freeaddrinfo(...)**
  - Frees memory allocated by **gettaddrinfo(...)**

Copyright 2012, The PTR Group, Inc.

Name to Address Translation #2

```
int getaddrinfo(const char *node,
                 const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);

struct addrinfo {
               int              ai_flags;
               int              ai_family;
               int              ai_socktype;
               int              ai_protocol;
               size_t           ai_addrlen;
               struct sockaddr *ai_addr;
               char            *ai_canonname;
               struct addrinfo *ai_next;
           };
```


Copyright 2012, The PTR Group, Inc.

Address to Name Translation

- **getnameinfo(…)**
  - You pass in v4 or v6 address and port
  - Size indicated by **salen** argument
  - Size for name and service buffers specified via **NI_MAXHOST**, **NI_MAXSERV**
  - Flags:
    - **NI_NOFQDN**, **NI_NUMERICHOST**, **NI_NAMEREQD**, **NI_NUMERICSERV**, **NI_DGRAM**
  - Returns name of host

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


Copyright 2012, The PTR Group, Inc.

Example Address Resolution

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

#define BUF_SIZE 500

int main(int argc, char *argv[])
    {
      struct addrinfo hints;
      struct addrinfo *result, *rp;
      int sfd, s;
      struct sockaddr_storage peer_addr;
      socklen_t peer_addr_len;
      ssize_t nread;
      char buf[BUF_SIZE];

      if (argc != 2) {
          fprintf(stderr, "Usage: %s port\n", argv[0]);
          exit(EXIT_FAILURE);
      }
```


Copyright 2012, The PTR Group, Inc.

Example Address Resolution #2

```
memset(&hints, 0, sizeof(struct addrinfo));
hints.ai_family = AF_UNSPEC;    /* Allow IPv4 or IPv6 */
hints.ai_socktype = SOCK_DGRAM; /* Datagram socket */
hints.ai_flags = AI_PASSIVE;    /* For wildcard IP address */
hints.ai_protocol = 0;          /* Any protocol */
hints.ai_canonname = NULL;
hints.ai_addr = NULL;
hints.ai_next = NULL;

s = getaddrinfo(NULL, argv[1], &hints, &result);
if (s != 0) {
   fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
   exit(EXIT_FAILURE);
}

/* getaddrinfo() returns a list of address structures.
   Try each address until we successfully bind(2).
   If socket(2) (or bind(2)) fails, we (close the socket
   and) try the next address. */

for (rp = result; rp != NULL; rp = rp->ai_next) {
   sfd = socket(rp->ai_family, rp->ai_socktype,
                rp->ai_protocol);
   if (sfd == -1) continue;

   if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0) break;  /* Success */
```


Copyright 2012, The PTR Group, Inc.

Example Address Resolution #3

```
    close(sfd);
  }

  if (rp == NULL) {                /* No address succeeded */
      fprintf(stderr, "Could not bind\n");
      exit(EXIT_FAILURE);
  }

  freeaddrinfo(result);            /* No longer needed */

  /* Read datagrams and echo them back to sender */

  for (;;) {
     peer_addr_len = sizeof(struct sockaddr_storage);
     nread = recvfrom(sfd, buf, BUF_SIZE, 0,
              (struct sockaddr *) &peer_addr, &peer_addr_len);
     if (nread == -1) continue;    * Ignore failed request */

     char host[NI_MAXHOST], service[NI_MAXSERV];

     s = getnameinfo((struct sockaddr *) &peer_addr,
                     peer_addr_len, host, NI_MAXHOST,
                     service, NI_MAXSERV, NI_NUMERICSERV);
```


Copyright 2012, The PTR Group, Inc.

Example Name Resolution #4

```
    if (s == 0)
        printf("Received %ld bytes from %s:%s\n",
                  (long) nread, host, service);
    else
        fprintf(stderr, "getnameinfo: %s\n",
                   gai_strerror(s));

    if (sendto(sfd, buf, nread, 0,
              (struct sockaddr *) &peer_addr,
               peer_addr_len) != nread)
         fprintf(stderr, "Error sending response\n");
   }
}
```


Copyright 2012, The PTR Group, Inc.

World IPv6 Day and Follow-On

- June 8, 2011 was World IPv6 Day
  - World-wide testing of IPv6 readiness
  - http://isoc.org/wp/worldipv6day/
  - Major vendors tested IPv6

- June 6, 2012 is the goal for permanently enabling IPv6 on major servers like Google, Yahoo!, Akamai, etc.

Copyright 2012, The PTR Group, Inc.

Testing Your IPv6 Readiness

- There is a test site: http://test-ipv6.com

Test your IPv6 connectivity.

Summary | Tests Run | Technical Info | Share Results / Contact

Your IPv4 address on the public internet appears to be 68.100.143.100

Your IPv6 address on the public internet appears to be 2001:0:53aa:64c:1835:61f6:bb9b:709b
Your IPv6 service appears to be: Teredo

(unknown result code: teredo-ipv4pref)

World IPv6 day is June 8th, 2011. **No problems are anticipated for you** with this browser, at this location. [more info]

Congratulations! You appear to have both IPv4 and IPv6 internet working. If a publisher publishes to IPv6, your browser will connect using IPv6. Note: Your browser appears to prefer IPv4 over IPv6 when given the choice. This may in the future affect the accuracy of sites who guess at your location.

Your DNS server (possibly run by your ISP) appears to have no access to the IPv6 internet, or is not configured to use it. This may in the future restrict your ability to reach IPv6-only sites. [more info]

Your readiness scores

10/10	for your IPv4 stability and readiness, when publishers offer both IPv4 and IPv6
9/10	for your IPv6 stability and readiness, when publishers are forced to go IPv6 only

Click to see test data

(Updated server side IPv6 readiness stats)

Copyright 2012, The PTR Group, Inc.

Summary

- For devices that are not connected to the Internet, embedded developers can probably ignore IPv6 for another few years
- For developers of middle boxes and mobile platforms, IPv6 will be of growing importance
  - Major carriers already mandate that any *new* device will have IPv6 required
- The use of dual-stacks represents the smoothest transition path
  - Albeit with the overhead of extra memory
- Fortunately, conversion of software to support IPv6 isn't likely to be a cliff

Copyright 2012, The PTR Group, Inc.