socket-programming

Implementations (C lang)

- IP Lookup
- Stream Sockets
    - Server
    - Receiver
- Datagram Sockets
    - Server
    - Receiver

What is a Socket?

- Socket is a way to speak to other programs using standard Unix file descriptors.

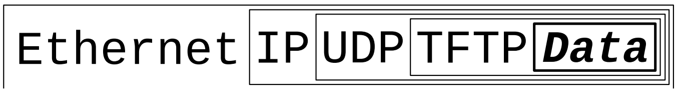
Types of Internet Socket

- Stream Sockets (`SOCK_STREAM`) => Two Way Connected Sockets [TCP/IP]
  ex: Telnet, HTTP
  IP(Internet Protocol) => Deals with Internet routing and not generally responsible for data integrity.
- Datagram Sockets (`SOCK_DGRAM`) => Connectionless Sockets [UDP]
  ex: tftp(trivial file transfer protocol), dhcpcd (DHCP client), multiplayer games, audio streaming, video conferencing
  this also uses IP for routing
  Connectionless

(More than two but only consider these two)

Low level and Network Theory

Data Encapsulation

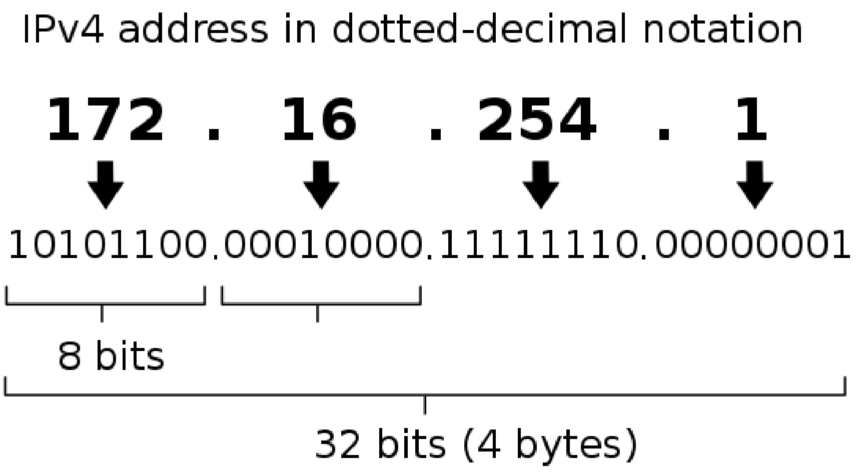


Layered Model

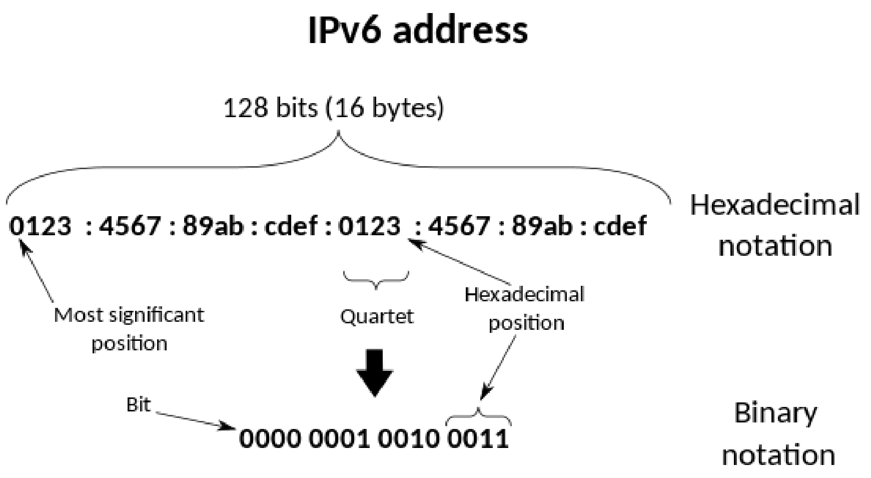
- Application Layer (telnet, ftp)
- Host to Host Transport Layer (TCP, UDP)
- Internet Layer (IP and routing)
- Network Access Layer (Ethernet, wi-fi)

IP Addresses, structs and Data Muning

IP Addresses (v4 and v6)



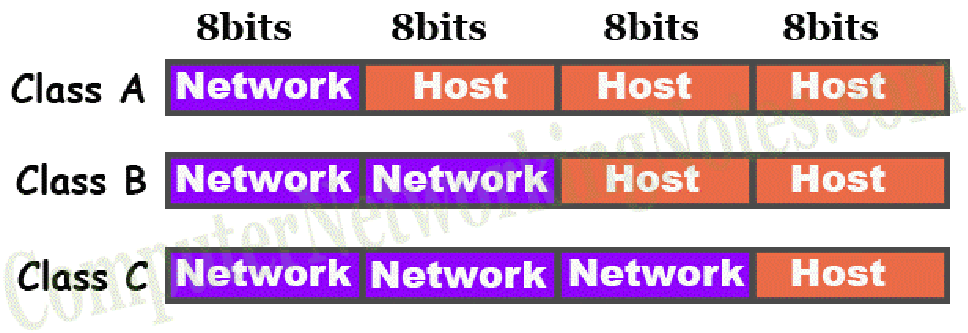
- **IPv4 => 32 bits (2^{32})**



- **IPv6 => 128 bits (2^{128})**
- IPv6 Representaions (Every pair is equal)
  - `2001:0db8:c9d2:0012:0000:0000:0000:0051 => 2001:db8:c9d2:12::51`
  - `2001:0db8:ab00:0000:0000:0000:0000:0000 => 2001:db8:ab00::`
  - `0000:0000:0000:0000:0000:0000:0000:0001 => ::1`
- Represent IPv4 in IPv6
  - `192.0.2.33 => ::ffff:192.0.2.33`

Subnets

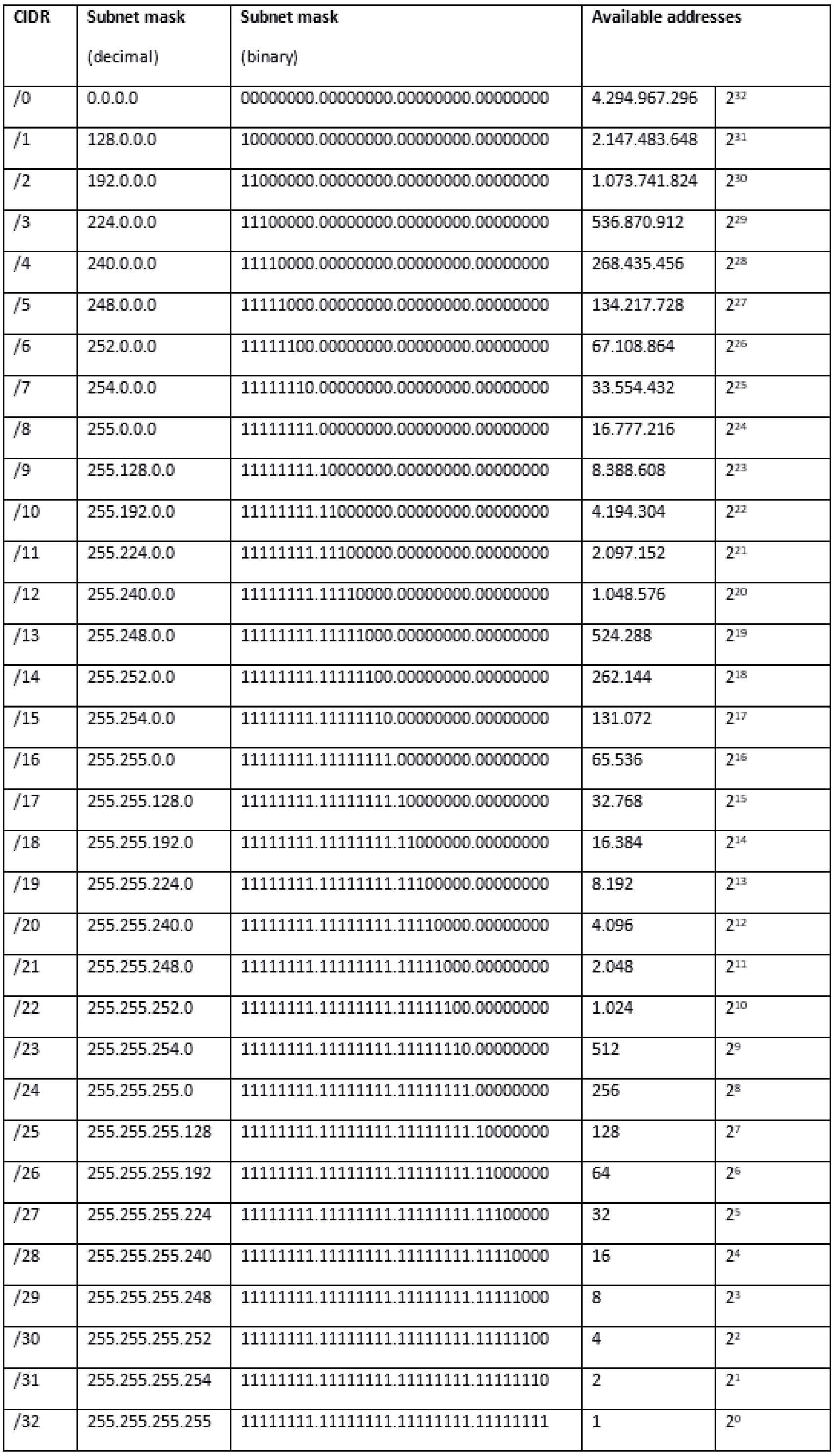
Classes in IPv4



Class	Network(bytes)	Host(bytes)	Range	Subnet Mask	No. of networks	No. of Hosts per network	Application
A	1	3	1.0.0.0 to 126.0.0.0	255.0.0.0	126	16,777,214	large no. of hosts
B	2	2	128.0.0.0 to 191.255.0.0	255.255.0.0	16,382	65,534	medium size networks
C	3	1	192.0.0.0 to 223.255.255.0	255.255.255.0	2,097,150	254	local area networks
D	N/A	N/A	224.0.0.0 to 239.255.255.255	N/A	N/A	N\A	Multicasting
E	N/A	N/A	240.0.0.0 to 255.255.255.255	N/A	N/A	N\A	Research/Reserved/Experimental

CIDR (Classless Inter-Domain Routing) Notation

- Representaion:
  - IPv4 => `192.0.2.12/30` - IPv6 => `2001:db8:5413:4028::9db9/64` or `2001:db8::/32`



CIDR	Subnet mask (decimal)	Subnet mask (binary)	Available addresses	
/0	0.0.0.0	00000000.00000000.00000000.00000000	4.294.967.296	2^{32}
/1	128.0.0.0	10000000.00000000.00000000.00000000	2.147.483.648	2^{31}
/2	192.0.0.0	11000000.00000000.00000000.00000000	1.073.741.824	2^{30}
/3	224.0.0.0	11100000.00000000.00000000.00000000	536.870.912	2^{29}
/4	240.0.0.0	11110000.00000000.00000000.00000000	268.435.456	2^{28}
/5	248.0.0.0	11111000.00000000.00000000.00000000	134.217.728	2^{27}
/6	252.0.0.0	11111100.00000000.00000000.00000000	67.108.864	2^{26}
/7	254.0.0.0	11111110.00000000.00000000.00000000	33.554.432	2^{25}
/8	255.0.0.0	11111111.00000000.00000000.00000000	16.777.216	2^{24}
/9	255.128.0.0	11111111.10000000.00000000.00000000	8.388.608	2^{23}
/10	255.192.0.0	11111111.11000000.00000000.00000000	4.194.304	2^{22}
/11	255.224.0.0	11111111.11100000.00000000.00000000	2.097.152	2^{21}
/12	255.240.0.0	11111111.11110000.00000000.00000000	1.048.576	2^{20}
/13	255.248.0.0	11111111.11111000.00000000.00000000	524.288	2^{19}
/14	255.252.0.0	11111111.11111100.00000000.00000000	262.144	2^{18}
/15	255.254.0.0	11111111.11111110.00000000.00000000	131.072	2^{17}
/16	255.255.0.0	11111111.11111111.00000000.00000000	65.536	2^{16}
/17	255.255.128.0	11111111.11111111.10000000.00000000	32.768	2^{15}
/18	255.255.192.0	11111111.11111111.11000000.00000000	16.384	2^{14}
/19	255.255.224.0	11111111.11111111.11100000.00000000	8.192	2^{13}
/20	255.255.240.0	11111111.11111111.11110000.00000000	4.096	2^{12}
/21	255.255.248.0	11111111.11111111.11111000.00000000	2.048	2^{11}
/22	255.255.252.0	11111111.11111111.11111100.00000000	1.024	2^{10}
/23	255.255.254.0	11111111.11111111.11111110.00000000	512	2^{9}
/24	255.255.255.0	11111111.11111111.11111111.00000000	256	2^{8}
/25	255.255.255.128	11111111.11111111.11111111.10000000	128	2^{7}
/26	255.255.255.192	11111111.11111111.11111111.11000000	64	2^{6}
/27	255.255.255.224	11111111.11111111.11111111.11100000	32	2^{5}
/28	255.255.255.240	11111111.11111111.11111111.11110000	16	2^{4}
/29	255.255.255.248	11111111.11111111.11111111.11111000	8	2^{3}
/30	255.255.255.252	11111111.11111111.11111111.11111100	4	2^{2}
/31	255.255.255.254	11111111.11111111.11111111.11111110	2	2^{1}
/32	255.255.255.255	11111111.11111111.11111111.11111111	1	2^{0}

Port Numbers

- 2 byte number that is like the local address for the connection
  - *(Think of the IP address as the street address of a hotel, and the port number as the room number.)*
- Details about different ports available on,
  - IANA Site
  - In Unix : `cat /etc/services`
- Some ports require special OS privileges to use - ex: port `1024`

Byte Order

- Two byte orderings
  - Network Byte Order (Big-Endian) -> big end first
    ex: b34f as [b3 | 4f]
  - Host Byte Order (Little-Endian) -> Little end first
    ex: b34f as [4f | b3]

- Programmer has to convert the numbers to **Network Byte** Order before they go out on the wire.
- Also convert them to **Host Byte Order** as they come in off the wire.

Function	Description
htons()	host to network short
htonl()	host to network long
ntohs()	network to host short
ntohl()	network to host long

structs

Name	Type	Usage
socket descriptor	`int`	-
`addrinfo`	`struct`	prep the socket address structures for subsequent use ([redacted])
`sockaddr`	`struct`	
`sockaddr_in`	`struct`	
`in_addr`	`struct`	
`sockaddr_in6`	`struct`	
`in6_addr`	`struct`	
`sockaddr_storage`	`struct`	

- **`addrinfo`**

```
struct addrinfo {
    int              ai_flags;     // AI_PASSIVE, AI_CANONNAME
    int              ai_family;    // AF_INET, AF_INET6, AF_UNSPEC
    int              ai_socktype;  // SOCK_STREAM, SOCK_DGRAM
    int              ai_protocol;  // use 0 for "any"
    size_t           ai_addrlen;   // size of ai_addr in bytes
    struct sockaddr *ai_addr;      // struct sockaddr_in or sockaddr_in6
    char            *ai_canonname; // full canonical hostname
    struct addrinfo *ai_next;      // linked list, next node
};
```

- **`sockaddr`**

```
struct sockaddr {
    unsigned short sa_family;   // address family, AF_xxx (AF_INET/ AF_INET6)
    char           sa_data[14]; // 14 bytes of protocol address
};
```

- **`sockaddr_in` (IPv4 only)**

```
/* for IPv4 only */
struct sockaddr_in {
 short int          sin_family;  // address family, AF_INET
 unsigned short int sin_port;    // port number
 struct in_addr     sin_addr;    // internet address
 unsigned char      sin_zero[8]; // same size as struct sockaddr;
};
```

- **`in_addr` (IPv4 only)**

```
 /* for IPv4 only, internet address (a structure for historical reasons) */
struct in_addr {
    uint32_t s_addr; // 32-bit int (4 bytes)
};
```

- **`sockaddr_in6` (IPv6 only)**

```
 /* IPv6 only */
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
};
```

- **`in6_addr` (IPv6 only)**

```
 /* IPv6 only */
struct in6_addr {
    unsigned char s6_addr[16]; // ipv6 address
};
```

- **`sockaddr_storage`**

```
 struct sockaddr_storage {
    sa_family_t ss_family; // address family

    // implementaion specific paddings
    char      __ss_pad1[__SS_PAD1SIZE];
    int64_t   __ss_align;
    char      __ss_pad2[__SS_PAD2SIZE];
};
```

IP Addresses, Part Deux

- `inet_pton` function (human readble mode to binary) (pton -> presentation to network / printable to network)

  NAME
  inet_pton - convert IPv4 and IPv6 addresses from text to binary form

  SYNOPSIS

  ```
   #include <arpa/inet.h>

  int inet_pton(int af, const char *src, void *dst);
  ```

  DESCRIPTION This function converts the character string src into a network address structure in the af address family, then copies the network address structure to dst. The af argument must be either `AF_INET` or `AF_INET6`. dst is written in network byte order.

  EXAMPLE

  ```
   struct sockaddr_in sa; // IPv4
  struct sockaddr_in6 sa6; // IPv6

  inet_pton(AF_INET, "10.12.110.57", &(sa.sin_addr)); // IPv4
  inet_pton(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr)); // IPv6
  ```

- `inet_ntop` function (binary to human readble mode)

  NAME inet_ntop - convert IPv4 and IPv6 addresses from binary to text form

  SYNOPSIS

  ```
   #include <arpa/inet.h>

  const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
  ```

  DESCRIPTION

This function converts the network address structure src in the af ad- dress family into a character string. The resulting string is copied to the buffer pointed to by dst, which must be a non-null pointer. The caller specifies the number of bytes available in this buffer in the argument size.

> All these `inet_pton()` and `inet_ntop()` will not do any DNS lookup on hostname. For that use `getaddrinfo()`.

System Calls or Busts

`getaddrinfo()`

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

- **onerror** :: `return non-zero integer` (if return 0, then it's successfull)

`socket()`

```
int socket(int domain, int type, int protocol);
```

- **onerror** :: `return -1`

`bind()`

```
int bind(int sockfd, struct sockaddr* my_addr, int addrlen);
```

- **onerror** :: `return -1`

`connect()`

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

- **onerror** :: `return -1`

`listen()`

```
int listen(int sockfd, int backlog);
```

- **onerror** :: `return -1`

`accept()`

```
int accept(int sockfd, struct sockaddr* addr, socklen_t* addrlen);
```

- **onerror** :: `return -1`

`send()` and `recv()`

```
 int send(int sockfd, const void* msg, int len, int flags);

int recv(int sockfd, void* buf, int len, int flags);
```

`sendto()` and `recvfrom()`

```
 int sendto(int sockfd, const void* msg, int len, unsigned int flags, const struct sockaddr *to, socklen_t tolen);

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

`close()` and `shutdown()`

```
 void close(int sockfd);

int shutdown(int sockfd, int how);
```

`getpeername()`

```
    int getpeername(int sockfd, struct sockaddr* addr, int* addrlen);
```

`gethostname()`

```
    int gethostname(const* hostname, size_t size);
```