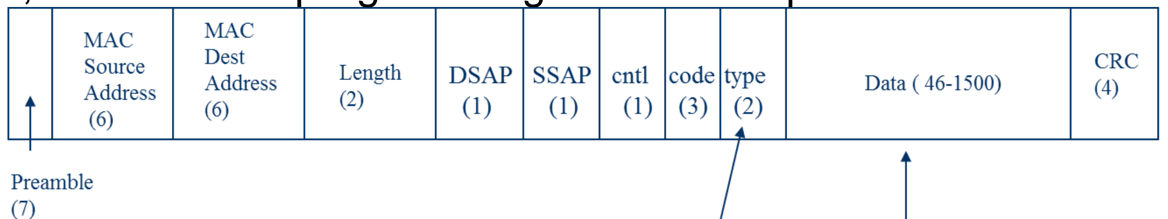
Tutorial on Addressing

- Machine address vs Network Addresses
  - Link / Phy layers use machine addresses
  - Network layer uses higher level addresses
- ARP: Address resolution protocol is used to allow a Host to learn the MAC address of another Host.
- There are several network communication paradigms:
  - One to one – unicast
  - One to all – broadcast
  - One to a few – multicast
- This tutorial focuses on IPV4 classful addressing, how the three paradigms are supported, and then the programming interface required.



Type indicates the type of protocol contained in the data …. This is how the LLC layer forwards received frames to the correct next layer (example - ARP or IP)

Payload : ARP frame, IP packet Maximum Transmission Unit (MTU) defined by the physical medium and the link layer

Host Identifiers

- What is the big issue we are dealing with?
  - A global network based on an internetworking model (like the Internet) requires a global name space.
  - A ‘Host’ is any computer that participates in the global or unified network.
  - Each Host must have a unique identifier which at an abstract level will require
    - Name: identifies what the object is
    - Address: identifies where it is
    - Route: identifies how to get there

Host Identifiers –what is the challenge ?

•Complexity as the issue is addressed at every layer in the layered network stack model with further complexity added at each layer to support mobile Hosts:

- •PHY: Each Host's physical connection requires a machine address. A Host with 2 interfaces will have 2 MAC addresses.
- •LINK: interfaces network addresses with machine addresses with the Address Resolution Protocol (ARP)
- •NETWORK: A network layer must understand the specifics related to network addresses, including required support for unicast, broadcast, multicast
- •TRANSPORT: a transport layer must translate between human readable names and network (binary) addresses. And more recently, transport protocols might support multiple paths or interfaces. For example Multipath TCP must be support names/addresses across multiple 'sub flows'
- •APPLICATIONS: Must be aware of the details at the transport layer interface but unfortunately also at the network (e.g., multiple versions like V4/V6) and any additional issues caused by mobility.

Programming View

- A network interface on a Host is assigned a network address
  - Static: a config file specifies the host's address
  - Dynamic: a protocol is used allowing a Host to learn its IP address
- A Host might have >1 interface - called multihomed.
  - In TCP/IP, a Host can serve as a router if it is multihomed AND if the IP stack is configured to forward packets between the two interfaces
- TCP/IP is a very specific network protocol….quite a bit of network architecture is built in to the addressing mechanism.
  - IP v4 vs IP v6 : 32 bit vs 128 bit network addresses (we focus on v4)
    - Might see a human readable domain name : www.cs.clemson.edu
    - Or the IP address in dotted decimal notation
      - » 130.127.49.4
- Programming perspective: we need to deal with this!!!!

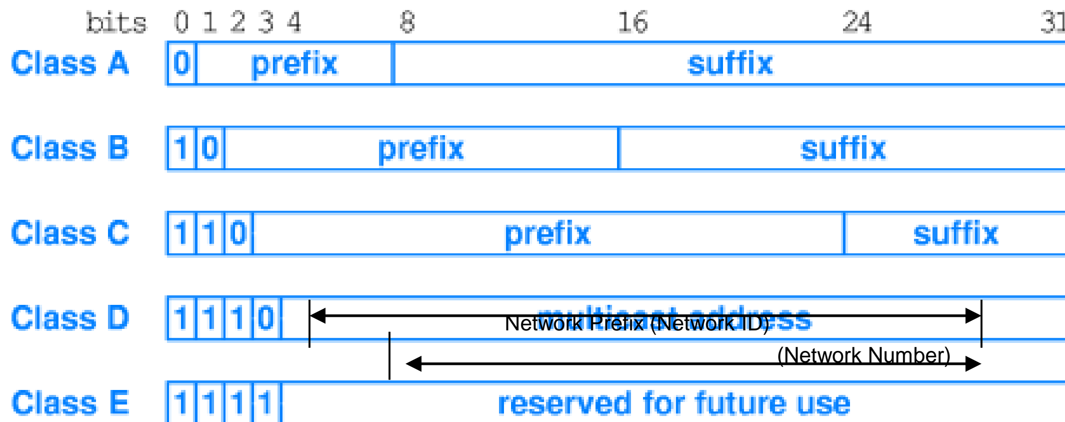
IP –s address

- Equivalent to the outdated ifconfig – lists all interfaces with stats
- eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
- link/ether 98:90:96:d7:64:24 brd ff:ff:ff:ff:ff:ff
- inet 130.127.48.105/23 brd 130.127.49.255 scope global eno1
- valid_lft forever preferred_lft forever
- inet6 2620:103:a000:401:9a90:96ff:fed7:6424/64 scope global mngtmpaddr dynamic
- valid_lft 2591959sec preferred_lft 604759sec
- inet6 fe80::9a90:96ff:fed7:6424/64 scope link
- valid_lft forever preferred_lft forever
- RX: bytes packets errors dropped overrun mcast
- 20771603924 42453437 0 0 0 259306
- TX: bytes packets errors dropped carrier collsns
- 29513098692 28510381 0 0 0 0

- Ifconfig:
- eno1 Link encap:Ethernet HWaddr 98:90:96:d7:64:24
- inet addr:130.127.48.105 Bcast:130.127.49.255 Mask:255.255.254.0
- inet6 addr: 2620:103:a000:401:9a90:96ff:fed7:6424/64 Scope:Global
- inet6 addr: fe80::9a90:96ff:fed7:6424/64 Scope:Link
- UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
- RX packets:42454903 errors:0 dropped:0 overruns:0 frame:0
- TX packets:28510593 errors:0 dropped:0 overruns:0 carrier:0
- collisions:0 txqueuelen:1000
- RX bytes:20771899399 (20.7 GB) TX bytes:29513150523 (29.5 GB)
- Interrupt:20 Memory:f7100000-f7120000

IP v4 Addresses

- IP v4 Addresses are 32 bit.
- On koala4.cs.clemson.edu, an ip address or ifconfig shows the v4 (and v6) address, and the subnet mask (which we will talk about more later….)
  - inet addr:130.127.48.105 Bcast:130.127.49.255 Mask:255.255.254.0
  - 130.127.48.105 is quad network notation – simply shows the value of each of the four octets, Most Significant first
- Write this 32 bit address in binary:
  - 100000010 0111111 00110000 0110 1001



Original address scheme was classful:

- Class A for large networks (>64K hosts)
- Class B for medium networks (>256 hosts)
- Class C for small networks (<256 hosts)
- Class D for multicast
- Class E reserved

- The first most significant bits specify the class…. In this case it is a class B
- Knowing the class gives us the number of octets used to specify the network prefix: class B , the first 2 most significant octets are the network prefix

IP v4 Addresses

- An IP v4 address is typically provided by either a domain name or by the dotted quad notation that effectively represents the address as a string of 4 numbers (separated by dots).
- Our example address of 130.127.48.105 is in dotted quad notation
- Note: it is possible to id the 32 bit unsigned value but this is not helpful
- We in fact prefer to assign a domain name to this address.
  - Koala4.cs.clemson.edu
- A bit part of the support for IP addresses is to
  - Convert an IP address in number form (32 bit unsigned int) to a name in dotted quad notation.
  - Convert a domain name to the IP address that the name is bound to

IP v4 Addresses - practice

- Given an IP address of 192.168.2.1

- What class?
  - The first octet in binary is: 1100 0000
  - So : Class “C” !!!!

- An address encodes the network address (aka network prefix) as well as the host id (network id, host id)
  - The network prefix is <192.168.2> and the host ID is 1.
  - For a class C address, one octect is used to specify the Host ID, there can be at most 255 Hosts directly connected to this network
    - Actually less as not all Host IDs are valid, some special meaning:
      - Think ‘all one’s and all zero’s for either the Host ID portion or the Network prefix have special meaning
- Host 1-----192.168.2/24 ----- Host 2 (assume Host 1 is .4, and Host 2 .1)
- For Host1 to communicate only to Host 2 requires unicast
- For Host 1 to communicate to all hosts on the network, limited broadcast – 255.255.255.255
- For Host 1 to communicate with a multicast group, it needs to be a class D address and all participants must join the group.

Broadcast Addresses

Two Types of broadcasts:

•A network or a directed broadcast
  •contains a valid network and host id.
  •class C example: 192.168.1.255
•A local or limited broadcast does not require knowledge of the network address.
  •Referred to as the all 1's broadcast:
    255.255.255.255
  •A limited broadcast useful for certain startup protocols:

Special Addresses

- Conventions…. The ‘this’ rule and the ‘all’ rule:
  - A netid or a host id of ‘0’ implies ‘this’
  - A netid or a host id of ‘1’ implies ‘all’

- Class C example: 192.168.1.0

The ‘this’ rule: ‘This host’ on the network

- Class C example: 192.168.1.255

The ‘1’s’ rule: ‘All’ hosts on the network

IP v4 Addresses - practice

- Given an IP address of 192.168.2.1

- Network prefix: 192.168.2/24
  - Classful addressing has been extended by subnetting and supernetting to remove the limits or structure imposed by the classful approach.

- Host 1-----192.168.2/24 ----- Host 2 (Host 1: 192.168.2.1, Host 2: 192.168.2.4)

- Host1 to communicate only to Host 2 -   requires a unicast socket (maps to unicast addressing)
  - Which really means when Host 1 sends to Host 2 – it needs to know Host 2's MAC address….hence ARP is involves

- For Host 1 to communicate to all hosts on the network a special address is required:
  - limited broadcast – 255.255.255.255

- For Host 1 to communicate with a multicast group, a special address is required
  - , it needs to be a class D address and all participants must join the group

Special Addresses

Host Addresses with Special Meaning

- All 0's and all 1's
- Loop Back : 127.x.x.x (e.g., 127.0.0.1)
  - What address class? Class A
- Private address space:
  - RFC 1918 defines certain address ranges for private use.
    - 10.0.0.0 - 10.255.255.255 (Class A space)
    - 172.16.0.0 - 172.31.255.255(Class B space)
    - 192.168.0.0 - 192.168.255.255 (Class C space)

Special Addresses – Private addresses

Let's say we have to networks

Net1 192/168/1/24 ==campus net=== Net2 192.168.1/24

- This is allowed!!! I would guess that on our campus we have dozens (perhaps hundreds) that are connected to a network using the same IPV4 private address (like 192.168.2.1).
- This works as long as packets that get forwarded over the common IP network (i.e., our campus network) do NOT contain private addresses.

Classful Address Ranges

Class	Lowest Address	Highest Address
A	1.0.0.0	126.0.0.0
B	128.1.0.0	191.255.0.0
C	192.0.1.0	223.255.255.0
D	224.0.0.0	239.255.255.255
E	240.0.0.0	247.255.255.255

Dotted Decimal Class Address Ranges

Classful Addresses

•Addresses do not specify computers, but rather connections to particular hosts.
•Multihomed: A host that has >1 physical connection.

Classful Addresses

Basic mechanism: A two level class hierarchy.

- requires a unique network prefix for each physical interface.
- Two additional schemes designed to conserve net addresses: subnet and classless addressing (CIDR).

Other issues:

- Mobility
- Flexibility
- Naming
- And of course the IP Address shortage problem...

Network Byte Order

•Network Byte Order: concept to isolate a network from machine architectures.

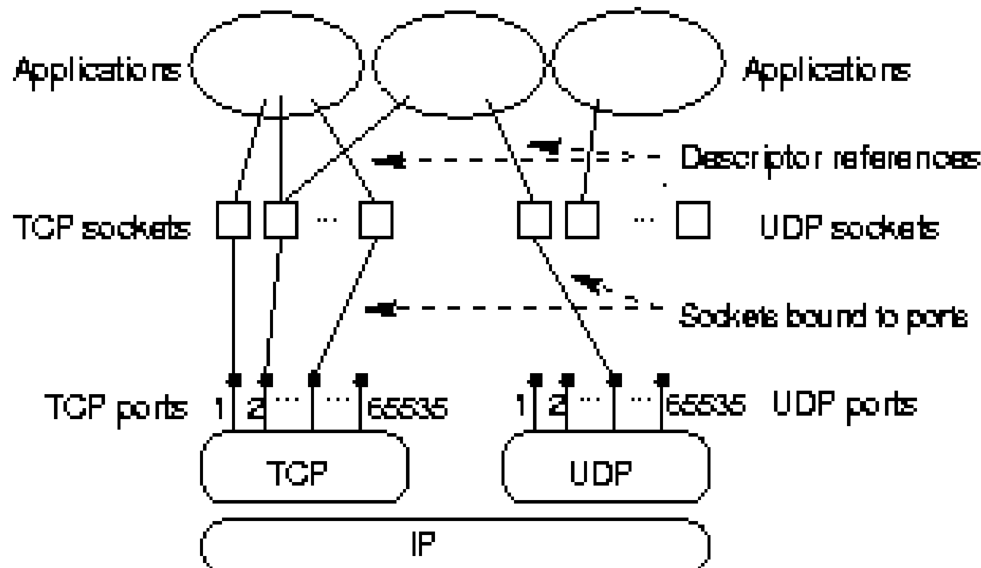
•Big Endian machines: lowest memory has high-order byte.
•Little Endian machines: lowest memory has low-order byte.
•Functions: htons(), htonl(), ntohs(), ntohl()

Network Byte Order is Big Endian. The MSB of an integer gets sent first.

Protocol Data must be in Network Byte Order but User Data does not have to be.

Sockets

- Identified by protocol and local/remote address/port
- Applications may refer to many sockets
- Sockets accessed by many applications



Abstract Socket Address Structure

Generic Address structure defined in <sys/socket.h>

```
struct sockaddr {
        u_char   sa_len;          /* total length */
        u_char   sa_family;       /* address family */
        char     sa_data[14];     /* actually longer; address value */
};
```

Address Structure IPv4

For sa_family = AF_INET use the following (defined in <netinet/in.h>):

```
struct in_addr {
        u_int32_t s_addr;
};
struct sockaddr_in {
        u_char     sin_len;
        u_char     sin_family;    //This is the address family- not the same as PF
        u_short    sin_port;
        struct     in_addr sin_addr;
        char       sin_zero[8];
};
```

//Note: the ‘in’ implies ‘Internet’

Address Structure IPv6

```
struct sockaddr_in6 {
        sa_family_t sin6_family; // Internet protocol (AF_INET6)
        in_port_t sin6_port; // Address port (16 bits)
        uint32_t sin6_flowinfo; // Flow information
        struct in6_addr sin6_addr; // IPv6 address (128 bits)
        uint32_t sin6_scope_id; // Scope identifier
}

struct in6_addr {
        unsigned char  s6_addr[16];
};
```

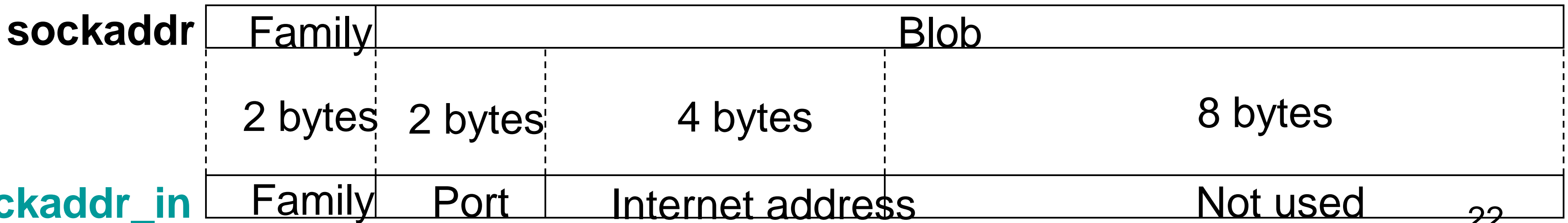
Note that an IPv6 address consumes more octets than what the generic InetAddr allows….that is ok as long as the code supports this.

Generic

```
struct sockaddr
{
        unsigned short sa_family;    /* Address family (e.g., AF_INET) */
        char sa_data[14];            /* Protocol-specific address information */
};
```

IP v4 Specific

```
struct sockaddr_in
{
        unsigned short sin_family;   /* Internet protocol (AF_INET) */
        unsigned short sin_port;     /* Port (16-bits) */
        struct in_addr sin_addr;     /* Internet address (32-bits) */
        char sin_zero[8];            /* Not used */
};
struct in_addr
{
        unsigned long s_addr;        /* Internet address (32-bits) */
};
```



Helper Function : to convert between IP addresses and dotted decimal

- Inet_pton and inet_ntop:
  - Replacements for inet_aton and inet_ntoa.
  - Convert between "presentation" (ascii) and "numeric" (binary)
  - Supports IPV4 and IPV6
- Example:
  - **int rtnVal = inet_pton(AF_INET, servIP, &servAddr.sin_addr.s_addr);**
  - if (rtnVal == 0)
    - DieWithUserMessage("inet_pton() failed", "invalid address string");
  - else if (rtnVal < 0)
    - DieWithSystemMessage("inet_pton() failed");
  - servAddr.sin_port = htons(servPort); // Server port

Helper Functions : writing network programs that work for either V4 or V6 addresses

- We live in a complex Internet world….
  - Hosts can support IPV4, (less likely only IPV6), or both which is referred to as 'dual stack'
  - Applications can chose to support only V4, only V6, or either depending on what the Host supports and the IP addresses assigned at that point in time.
- To support that flexibility, programs should use the routines:
  - *int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints,struct addrinfo **res)*
  - *int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, socklen_t hostlen, char *serv, socklen_t servlen, int flags);*

Helper Functions : writing network programs that work for either V4 or V6 addresses

- *int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints,struct addrinfo **res)*
- *int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, socklen_t hostlen, char *serv, socklen_t servlen, int flags);*

- The idea behind getaddrinfo is best explained from two possible calling contexts
  - As a client program that has been passed the server name (IPv4/6, DNS or address number format) and service (i.e., port).
    - Client passes the address and service strings as the first two parameters
    - Client fills in the hints structure – think of this as applying a filter to what the system might return.
    - The system returns all addresses it finds for the name/service with the caller's filter applied. This will likely be a list of possible addresses.
  - As a server program that is to bind an address/port to a socket:
    - The server will likely set the first parameter NULL and the second the desired port number
    - The server will like set a ai_flags field in the hints to indicate AI_PASSIVE. This will correctly tell the system to use the unspecified address allowing the socket to receive data from all addresses

Helper Function : getaddrinfo

```
//See code in ./Donahoo/TCPEcho/GetAddrInfo.c
char *addrString = argv[1];   // Server address/name
 char *portString = argv[2];   // Server port/service

  // Tell the system what kind(s) of address info we want
  struct addrinfo addrCriteria;                   // Criteria for address match
  memset(&addrCriteria, 0, sizeof(addrCriteria)); // Zero out structure
  addrCriteria.ai_family = AF_UNSPEC;             // Any address family
  addrCriteria.ai_socktype = SOCK_STREAM;         // Only stream sockets
  addrCriteria.ai_protocol = IPPROTO_TCP;         // Only TCP protocol

  // Get address(es) associated with the specified name/service
  struct addrinfo *addrList; // Holder for list of addresses returned
  // Modify servAddr contents to reference linked list of addresses
  int rtnVal = getaddrinfo(addrString, portString, &addrCriteria, &addrList);
  if (rtnVal != 0)
    DieWithUserMessage("getaddrinfo() failed", gai_strerror(rtnVal));

  // Display returned addresses
  for (struct addrinfo *addr = addrList; addr != NULL; addr = addr->ai_next) {
    PrintSocketAddress(addr->ai_addr, stdout);
```

Helper Functions : to convert between IP addresses and names : OLD !!!!

We are to use the naming service for translating between domain names, names based on numbers, and binary addresses : getaddrinfo and getnameinfo !!

gethostbyname, gethostbyaddr : convert between hostnames and IP addresses

```
struct hostent *  gethostbyname (const char *name)

struct hostent {
  char *h_name;   //official name of host
  char **h_aliases; //alias list
  int h_addrtype;   //host address type
  int h_length;     //length of address
  char **h_addr_list;  //list of addresses from name server
}
```

- getservbyname , getservbyport : converts between services and ports