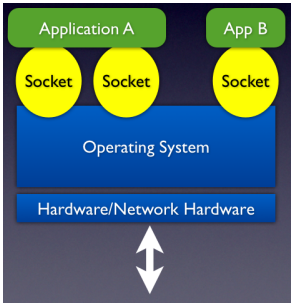
# How Applications Access the Network

**Socket** Network abstraction used for transferring data into and out of the network.




## Sockets

- Interfaces user programs with the operating system
- Allows Application Layer to access services of the Transport Layer

# Programming Sockets

Socket Programming Details

- Different families (IPv4 or IPv6)
- Different types (*stream*, *datagram*, *raw*, ...)
- Stores the IP Address and Port Number used for the communication

Socket Family Indicates whether you are using the IPv4 or IPv6 protocols for communication. (Network Layer)

Socket Type Indicates how the data will be sent across the network, in either a connection-based, or connection-less based manner. (Transport Layer)

Port Number The channel through which communication will travel on your computer. Somewhat analagous to opening a window on your house.

# Addresses...

Recall the addressing used at the different layers

- Application Layer $\rightarrow$ *Machine Names/Strings*
- Transport Layer $\rightarrow$ *Port Numbers*
- Network Layer $\rightarrow$ *IP Addresses*
- Link Layer $\rightarrow$ *MAC Addresses*

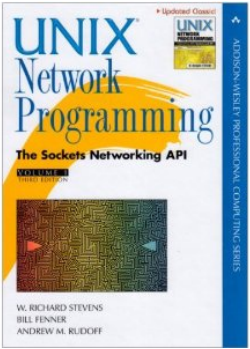
# C Socket API

## Socket API

We will program the network using the C Socket API. These are often low-level function calls that interface directly with the operating system to initiate, use, and terminate network connections.

## Good External References for the C Socket API

- UNIX Network Programming by Richard Stevens



# IPv4 Socket Structure

We will start by getting accustomed to the Socket structure:

## IPv4 Socket Structure

```
struct sockaddr_in
{
    __unint8_t      sin_len;
    sa_family_t     sin_family;
    in_port_t       sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];
}
```

- C structure that holds information necessary for creating and using a socket!
- What information seems important?

# IPv4 Socket Structure

We will start by getting accustomed to the Socket structure:

## IPv4 Socket Structure

```
struct sockaddr_in
{
    __unint8_t      sin_len;
    sa_family_t     sin_family;
    in_port_t       sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];
}
```

- C structure that holds information necessary for creating and using a socket!
- What information seems important?
- Certainly, family type, port number, and address..

# Generic Socket Structure

- The previous structure is specific to IPv4.
- Many Socket API functions take pointer to *generic* socket structures!
- Why????

# Generic Socket Structure

- The previous structure is specific to IPv4.
- Many Socket API functions take pointer to *generic* socket structures!
- Why????
- Primarily to allow generality across different socket families... IPv4, IPv6, etc...

# Generic Socket Structure

Generic Structure

```
struct sockaddr
{
    __unint8_t      sin_len;
    sa_family_t     sin_family;
    char            sin_data[14];
}
```

IPv4 Structure

```
struct sockaddr_in
{
    __unint8_t      sin_len;
    sa_family_t     sin_family;
    in_port_t       sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];
}
```

For instance:

```
struct sockaddr_in myIP4Addr;
...
randomSocketFunc( socketID, (struct sockaddr *)&myIP4Addr, sizeof(myIP4Addr) );
```

# IP Address Component

IPv4 Structure

```
struct sockaddr_in
{
    __unint8_t      sin_len;
    sa_family_t     sin_family;
    in_port_t       sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];
}
```

What does sin_addr contain?

- What are your ideas on what it contains?

# IP Address Component

IPv4 Structure

```
struct sockaddr_in
{
    __unint8_t      sin_len;
    sa_family_t     sin_family;
    in_port_t       sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];
}
```

What does sin_addr contain?

- What are your ideas on what it contains?
- The 32-bit number that represents the IP address!
- Where do the 32-bits come from???

# IP Address Component

IPv4 Structure

```
struct sockaddr_in
{
    __unint8_t      sin_len;
    sa_family_t     sin_family;
    in_port_t       sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];
}
```

What does sin_addr contain?

- What are your ideas on what it contains?
- The 32-bit number that represents the IP address!
- Where do the 32-bits come from???
- Each of the 4 decimal numbers in an IPv4 address can be stored in 8 bits!

# Dotted Decimal IPv4 Addresses

- Dotted Decimal IP Address: 131.212.7.1
- Converted to a Decimal 32-bit value: 2211710721

Each 8-bit value from the dotted decimal notation is packed into the 32-bit storage and we can interpret it as a number.

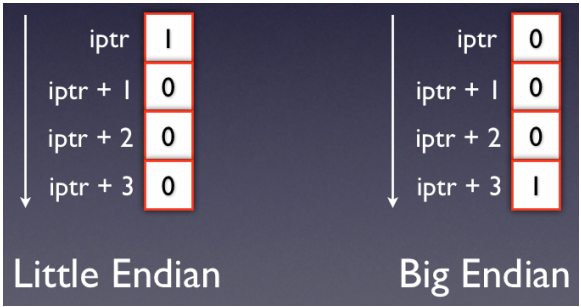
# Host Byte Order

Either Little Endian or Big Endian

Little Endian Least significant byte in starting address

Big Endian Most significant byte in starting address

```
int i = 1;
int *iptr = &i;
```



# Why does Byte Order Matter?

**Why?**

*You want your network apps to work between ALL types of machines!*

What is Network Byte Order?

- Communication on the network is done in Network Byte Order
- Network byte order is Big Endian

Fairly straight-forward to support (at least initially):

- Use the functions provided in the C Socket API to convert to and from *host* and *network* byte order:

```
uint16_t htons(uint16_t hostshort);
uint32_t htonl(uint32_t hostlong);
uint16_t ntohs(uint16_t netshort);
uint32_t ntohl(uint32_t netlong);
```

- Important to make conversions for IP addresses (32-bit, or 4-byte component for IPv4)
- Important for port numbers too!

# How can you compute byte order?

- Write a value to a multi-byte type and test the individual bytes!

# How can you compute byte order?

- Write a value to a multi-byte type and test the individual bytes!
- One Way:

```
unsigned int i = 1;
char *cptr = (char *) &i;
if (cptr[0] == 1)
    // Little Endian!
```

# How can you compute byte order?

- Write a value to a multi-byte type and test the individual bytes!
- One Way:

```
unsigned int i = 1;
char *cptr = (char *) &i;
if (cptr[0] == 1)
   // Little Endian!
```

- Another Way:

```
union EndianTestType {
   unsigned int i;
   char c[sizeof(unsigned int)];
};
EndianTestType endianTest;
endianTest.i = 1;
if (endianTest.c[0] == 1)
   // Little Endian!
```

# Back to Sockets!

Recall that sockets let you USE the network, so

- Start working to understand the socket address structure
- Start being aware of the C Socket API and related functions

What can you do with this information?

- Access *services* of the Internet
- Use *protocols* of the Internet
- Write *your* own services and protocols!

The following slides will provide some examples of use of these services.

# Net Tools You Can Use

## Domain Name Conversion Tools

Humans like strings, machines like numbers and this converts between the two. (*Hint:* Compare your own Programming Assignment 1 Code against the output of these tools.)

- These programs communicate with the DNS (Domain Name System) to perform various host name to IP conversions.
- nslookup - query Internet name servers
- dig - DNS Lookup utility
- host - another DNS Lookup utility

## Connectivity and Topology Tools

Determine is a host is online as well as the route to the host.

- ping - may determine if a machine is online
- traceroute - can help understand the path between hosts

# nslookup and dig

## host

Simple domain name string to number conversion utility.

## nslookup

- Queries the domain name servers for the information provided on the command line, or through nslookup's interactive shell
- Read the man page for more info: *man nslookup*
- Example: nslookup www.amazon.com

## dig

- Stands for Domain Information Groper
- Provides nice detail once you understand more about DNS!
- Read the man page for more info: *man dig*
- Example: dig amazon.com

# ping

- Determines if a host is online, or on the network
- Not always 100% reliable...
- Example:

```
$ ping shell.cs.utah.edu
PING shell.cs.utah.edu (155.98.65.55) 56(84) bytes of data.
64 bytes from shell.cs.utah.edu (155.98.65.55): icmp_seq=1 ttl=43 time=40.3 ms
64 bytes from shell.cs.utah.edu (155.98.65.55): icmp_seq=2 ttl=43 time=40.1 ms
64 bytes from shell.cs.utah.edu (155.98.65.55): icmp_seq=3 ttl=43 time=39.9 ms
64 bytes from shell.cs.utah.edu (155.98.65.55): icmp_seq=4 ttl=43 time=40.0 ms
```

- Sends small packets to destination host (uses the ICMP - Internet Control Message Protocol)
- Can also tell you the round trip time (RTT) between you and the destination!
- Other examples to try: ping localhost

# traceroute

- Provides you with information about topology between hosts
- Again, not always 100% reliable...
- Example:

```
$ traceroute marengo.d.umn.edu
traceroute to marengo.d.umn.edu (131.212.41.85), 30 hops max, 60 byte packets
 1  131.212.7.252 (131.212.7.252)  0.342 ms  0.404 ms  0.464 ms
 2  marengo.d.umn.edu (131.212.41.85)  0.270 ms  0.278 ms  0.275 ms
```

- Uses the time to live (TTL) and ICMP protocol to query the path and the individual RTTs
  - TTL - packets have parameters that specify how long they can exist!
  - TTL is decremented at each hop towards the destination host
  - When TTL == 0, an error is generated on the network core (ICMP TIME_EXCEEDED)

# How you write these tools?

The following slides detail some of the functions related to the Socket API and will provide you with the basis for building some of your own tools, for instance,

- Your own DNS conversion system

While the initial focus is on helper functions within the C Socket API as they relate to DNS services, you will use these functions throughout your own socket code.

# Important Socket API Functions - Name/IP Conversions

- getaddrinfo - Function that converts from a host name to a dotted decimal host IP address.

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

# Important Socket API Functions - Name/IP Conversions

- getaddrinfo - Function that converts from a host name to a dotted decimal host IP address.

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

- Example Usage:

```
std::string hostname = ``www.amazon.com'';
struct addrinfo hints, *res0;
memset(&hints, 0, sizeof(hints));
hints.ai_family = PF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;

getaddrinfo(hostname.c_str(), 0, &hints, &res0);
// Or, you can not specify the hints...
// getaddrinfo(hostname.c_str(), 0, 0, &res0);

struct sockaddr_in *tmpAddr = (struct sockaddr_in*)(res0->ai_addr);
```

# Important Socket API Functions - Name/IP Conversions

- getnameinfo - Function that converts from a dotted decimal IP address name to host name.

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

# Important Socket API Functions - Name/IP Conversions

- getnameinfo - Function that converts from a dotted decimal IP address name to host name.

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

- Example Usage:

```
std::string ipAddrStr = ''131.212.7.1'';

memset(&sockAddr, 0 sizeof(struct sockaddr_in));
sockAddr.sin_len = sizeof(struct sockaddr_in);
sockAddr.sin_family = AF_INET;   // only IPv4 for now

if (inet_pton(AF_INET, ipAddrStr.c_str(), (void *)&(sockAddr.sin_addr)) == -1)
  {
    std::cout << "bad inet_pton conversion!!!" << std::endl;
  }

char hostName[NI_MAXHOST];
getnameinfo((const sockaddr *)&sockAddr, sizeof(sockAddr), hostName, NI_MAXHOST, 0, 0, NI_NAMEREQD));
```

# Important Socket API Functions - IP Presentation

- inet_pton - Converts IP addresses from text to binary form

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

# Important Socket API Functions - IP Presentation

- inet_pton - Converts IP addresses from text to binary form

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

- Example usage (inet_pton):

```
inet_pton(AF_INET, ipAddrStr.c_str(), (void *)&(sockAddr.sin_addr));
```

# Important Socket API Functions - IP Presentation

- inet_pton - Converts IP addresses from text to binary form

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

- Example usage (inet_pton):

```
inet_pton(AF_INET, ipAddrStr.c_str(), (void *)&(sockAddr.sin_addr));
```

- inet_ntop - Converts IP addresses from binary to text form

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

# Important Socket API Functions - IP Presentation

- inet_pton - Converts IP addresses from text to binary form

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

- Example usage (inet_pton):

```
inet_pton(AF_INET, ipAddrStr.c_str(), (void *)&(sockAddr.sin_addr));
```

- inet_ntop - Converts IP addresses from binary to text form

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

- Example usage (inet_ntop):

```
inet_ntop(res0->ai_family, (const void *)&(tmpAddr->sin_addr), ipAddrStr, 1024);
```