Writing Client/Server Programs in C Using Sockets (A Tutorial) Part I

Session 5958

Greg Granger
grgran@sas.com

SAS/C & C++ Support
SAS Institute
Cary, NC

Part I: Socket Programming Overview

- Sockets (to me)
- Networking (or what's natural about natural logs)
- TCP/IP (and what it means to your life)
- More Sockets (we didn't get enough the first time)

What is "Sockets"

- An Application Programming Interface (API) used for InterProcess Communications (IPC). [A well defined method of connecting two processes, locally or across a network]
- Protocol and Language Independent
- Often referred to as Berkeley Sockets or BSD Sockets

Connections and Associations

- In Socket terms a connections between two processes in called an association.
- An association can be abstractly defined as a 5-tuple which specifies the two processes and a method of communication. For example:
  - ***{protocol, local-addr, local-process, foreign-addr, foreign-process}***
- A half-association is a single "side" of an association (a 3-tuple)
  - ***{protocol, addr, process}***

Networking Terms

- packet - the smallest unit that can be transferred "through" the network by itself
- protocol - a set of rules and conventions between the communicating participants
- A collection of protocol layers is referred to as a "protocol suite", "protocol family" or "protocol stack". TCP/IP is one such protocol suite.

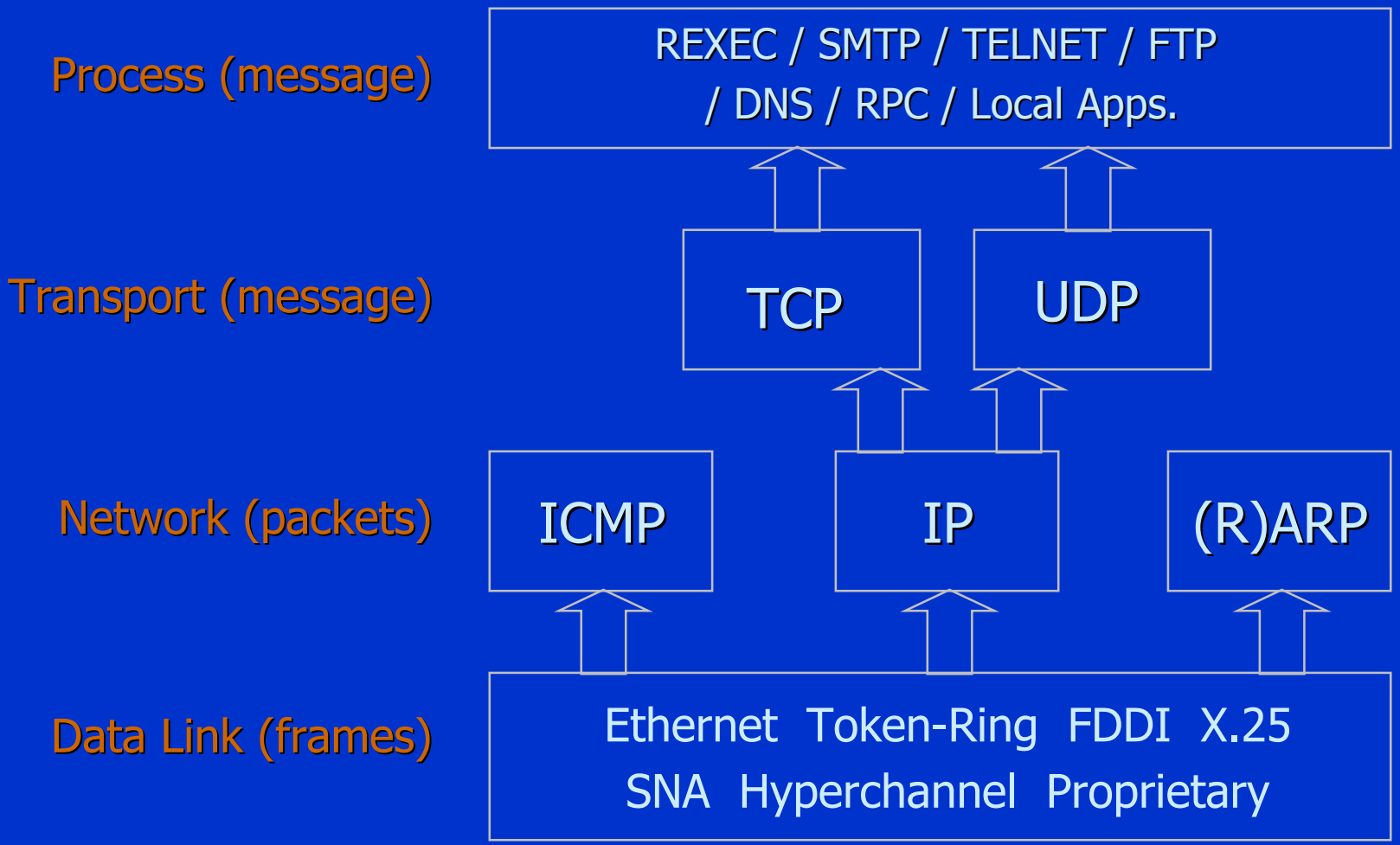
Introduction to TCP/IP

- What (the heck) is TCP/IP?
- Internet Protocol (IP)
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- TCP/IP Applications
- Name Resolution Processing
- TCP/IP Network Diagram

What is TCP/IP?

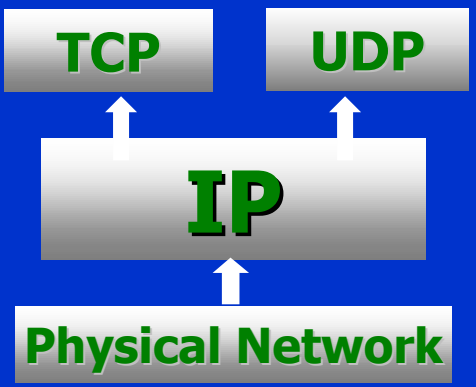
- Transmission Control Protocol/Internet Protocol
- A network protocol suite for interprocess communication
- The protocol of the Internet
- Open, nonproprietary
- Integrated into UNIX operating systems
- Many popular networking applications
  - telnet
  - X11 GUI
  - www
  - NFS (network file system)
  - SMTP (mail)
  - ftp (file transfer protocol)

TCP/IP Architectural Model



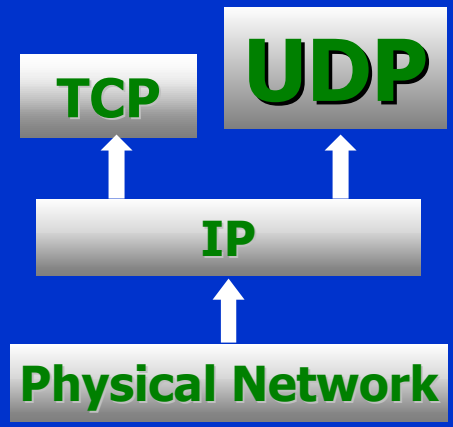
Internet Protocol (IP)

- Establishes a "virtual" network between hosts, independent of the underlying network topology
- Provides "routing" throughout the network, using IP addressing. For example: 149.173.70.9
- Features
  - Best-effort packet delivery
  - Connectionless (stateless)
  - Unreliable



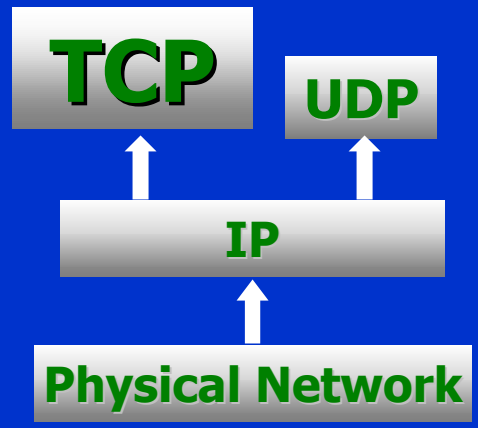
User Datagram Protocol (UDP)

- Application Interface to IP - Packet Oriented
- Establishes a "port", which allows IP to distinguish among processes running on the same host
- Features resemble IP semantics
  - Connectionless
  - Unreliable
  - Checksums (optional)



Transmission Control Protocol (TCP)

- Connection-oriented
- Stream Data Transfer
- Reliable
- Flow-Control
- Full-Duplex
- Suited for critical data transfer applications



The Importance of Ports

- Both the TCP and UDP protocols use 16 bit identifiers called ports to uniquely identify the processes involved in a socket.
- In UNIX the first 1024 ports for both protocols are called "well known ports" and are defined in the file /etc/services. Programs that bind to these ports require "root" access.
- These numbers are managed by the Internet Assigned Numbers Authority (IANA). A complete list of these assignments and more information about IANA can be found in RFC 1700

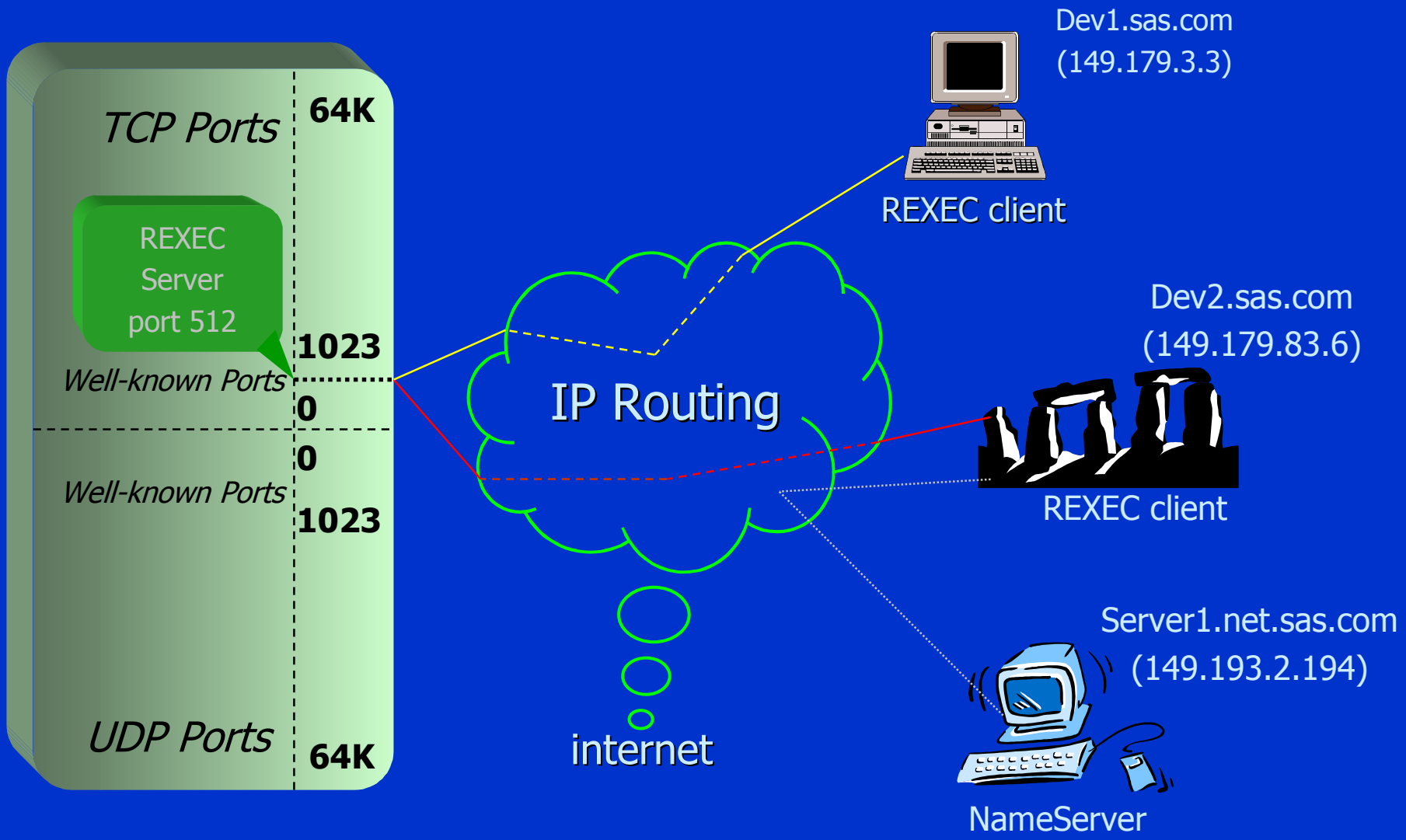
How stuff gets around (routing)

- TCP/IP packets are routed based on their destination IP address (ex: 10.24.2.123)
- Packets are passed from one network segment to another by machines called "routers" until the packet arrives at the network segment attached to the host with the destination IP address.
- Routers that act as gates to larger networks are called gateways.

Name Resolution Processing

- Associates an IP address to a "name" (hostname)
- Structured method of identifying hosts within an internet
- The Domain Name System (DNS) implements a hierarchical naming scheme which maps names like "mvs.sas.com" to an IP address
- DNS is implemented by a set of cooperating servers
- Machines that process DNS requests are called nameservers
- A set of library routines called "the resolver" provide the logic to query nameservers

TCP/UDP/IP Diagram



Back to Sockets

- Socket Definition and Components
- Socket Library Functions
- Primary Socket Header Files
- Sample Client/Server Dialog
- Ancillary Socket Topics
- Beyond Sockets

Definition and Components

- Socket - endpoint of communication
- Sockets - An application programming interface (API) for interprocess communication (IPC)
- Attributes:
  - Protocol Independent
  - Language Independent
  - Sockets implies (not requires) TCP/IP and C
- Socket and Connection Association
  - A local host can be identified by it's protocol, IP address and port.
  - A connection adds the IP address & port of the remote host.

Socket Library Function

- System calls
  - startup / close
  - data transfer
  - options control
  - other
- Network configuration lookup
  - host address
  - ports for services
  - other
- Utility functions
  - data conversion
  - address manipulation
  - error handling

Primary Socket Calls

- socket() - create a new socket and return its descriptor
- bind() - associate a socket with a port and address
- listen() - establish queue for connection requests
- accept() - accept a connection request
- connect() - initiate a connection to a remote host
- recv() - receive data from a socket descriptor
- send() - send data to a socket descriptor
- close() - "one-way" close of a socket descriptor

Network Database Administration functions

- gethostbyname - given a hostname, returns a structure which specifies its DNS name(s) and IP address(es)
- getservbyname - given service name and protocol, returns a structure which specifies its name(s) and its port address
- gethostname - returns hostname of local host
- getservbyname, getservbyport, getservent
- getprotobyname, getprotobynumber, getprotobyent
- getnetbyname, getnetbyaddr, getnetent

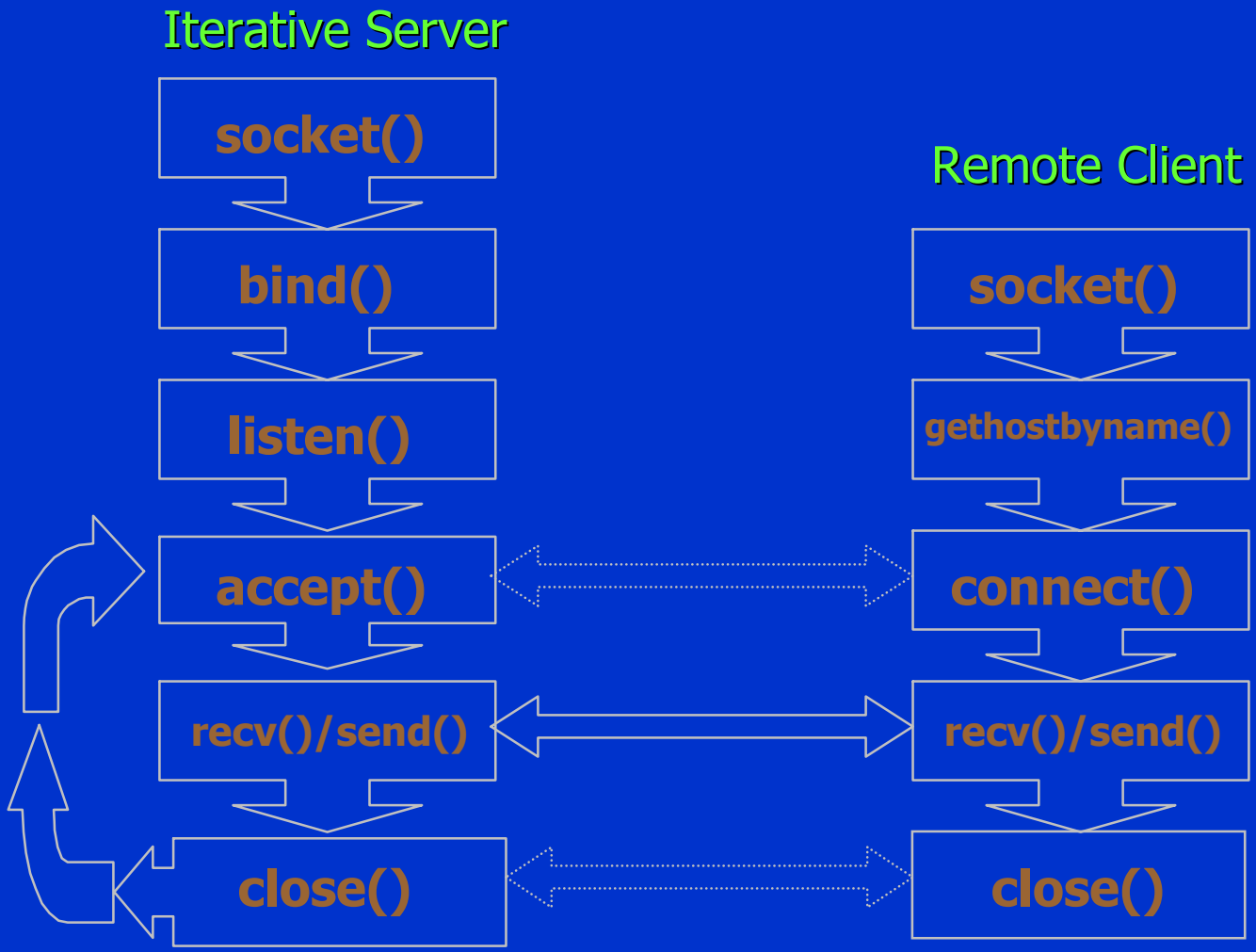
Socket Utility Functions

- ntohs/ntohl - convert short/long from network byte order (big endian) to host byte order
- htons/htonl - convert short/long from host byte order to network byte order
- inet_ntoa/inet_addr - convert 32-bit IP address (network byte order to/from a dotted decimal string)
- perror() - print error message (based on "errno") to stderr
- herror() - print error message for gethostbyname() to stderr (used with DNS)

Primary Header Files

- Include file sequence may affect processing (order is important!)
  - <sys/types.h> - prerequisite typedefs
  - <errno.h> - names for "errno" values (error numbers)
  - <sys/socket.h> - struct sockaddr; system prototypes and constants
  - <netdb.h.h> - network info lookup prototypes and structures
  - <netinet/in.h> - struct sockaddr_in; byte ordering macros
  - <arpa/inet.h> - utility function prototypes

Sample TCP Client / Server Session



Ancillary Socket Topics

- UDP versus TCP
- Controlling/managing socket characteristics
  - get/setsockopt() - keepalive, reuse, nodelay
  - fcntl() - async signals, blocking
  - ioctl() - file, socket, routing, interface options
- Blocking versus Non-blocking socket
- Signal based socket programming (SIGIO)
- Implementation specific functions

Design Considerations

- Data representation and conversion
- Server design alternatives
- Security Issues
- Portability Considerations

Data Representation

- Transport Protocols detail data exchange/movement; applications must interpret the data!
- Byte order affects data - not just addresses
- Text is often sent in ASCII, but ASCII versus EBCDIC is decided by the application-level protocol
- Structure alignment and floating point pose problems
- External Data Representation (XDR) can be used (even without RPC)

Server Design Alternatives

- Single Threaded
  - more complex code (must track multiple concurrent requests)
  - generally lower system overhead
  - crash of thread disables service
- Multi-Tasking
  - less complex code (written only for handling only one connection)
  - higher system overhead (each task requires it's own process space)
  - highly crash resistant (one or more tasks can fail without losing service)
- [Multi-]Threaded
  - shares less complex code of Multi-Tasking model
  - system overhead between Single-Threaded and Multi-Tasking model
  - crash resistant (but one badly behaved thread 'can' crash service)

Security Considerations

- Socket semantics do NOT address security problems, such as:
  - IP and adapter addresses
  - Userid and passwords
  - data encryption
  - traces
- UNIX systems require "root" privilege when a program binds a "reserved" (<1024) port
- getpeername() returns the peer's port and IP-address: determine "privileged" peers and "trusted" hosts
- The Kerberos protocol provides password and data encryption, along with service authentication

Portability Considerations

- Limit applications to "standard" socket routines, BSD 4.x
- Implement a portable transport module
- Mainframe Environment - Distribute existing applications
  - API Programmer's Reference - Details
  - SAS/C, C/370, Interlink, Open Connect, NSC
- OS/2 - REXX Sockets, Programmer's Toolkit
- MS Windows Sockets 1.1 - 2 WINSOCK.DLL (http://www.stardust.com ftp.stardust.com:/pub/winsock)

Summary

- Basic networking and features of TCP/IP protocols
- Socket library organization
- Socket library coding techniques
- Awareness of more advanced topics

What's Next

- Session 5959 - Part II - Client/Server Application

Bibliography

- Internetworking with TCP/IP: Volumes I, II & III, Douglas Comer, Prentice Hall, 1991 (ISBN Vol I: 0134685059, Vol III: 0138487146)
- The Whole Internet User's Guide & Catalog by Ed Kroll; O'Reilly & Associates
- UNIX Network Programming by W. Richard Stevens; Prentice Hall, 1990 (ISBN 0139498761)
- Socket API Programmer's Reference
- UNIX "man" pages
- TCP/IP Illustrated: Volumes 1 & 2, W. Richard Stevens (v2 with Gary R. Wright); Addison-Wesley Publishing Company, 1994