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

Session 5954

Gary T. Ciampa

sasgyc@unx.sas.com

SAS/C & C++ Support

SAS Institute

Cary, NC

Part 1: Socket Programming Overview

- TCP/IP Overview
- Introduction to Sockets
- Design Considerations

Part 2: Sample Client/Server Application

Introduction to TCP/IP

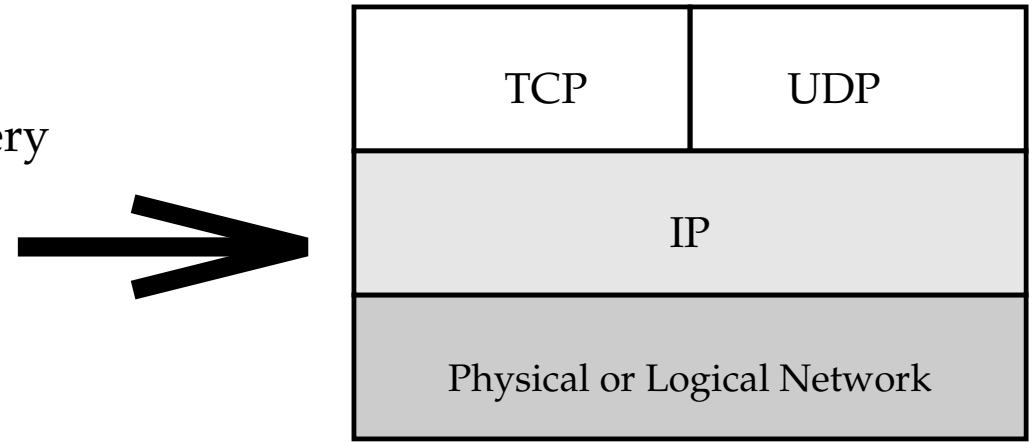
- What is TCP/IP?
- Internet Protocol (IP)
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- TCP/IP Applications
- Name Resolution Processing
- TCP/IP Network Diagram

What is TCP/IP?

- A network protocol suite: designed for "internetworking"
- 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)

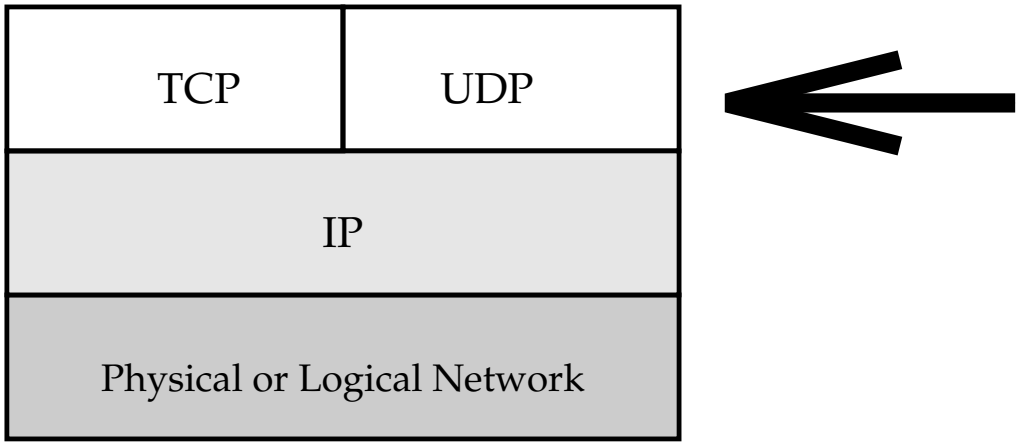
Internet Protocol (IP)

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



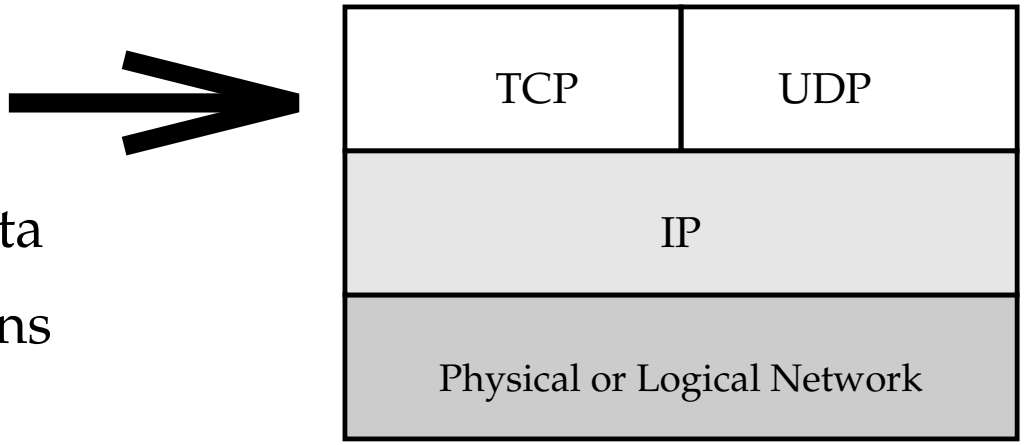
User Datagram Protocol (UDP)

- 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?



Transmission Control Protocol (TCP)

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



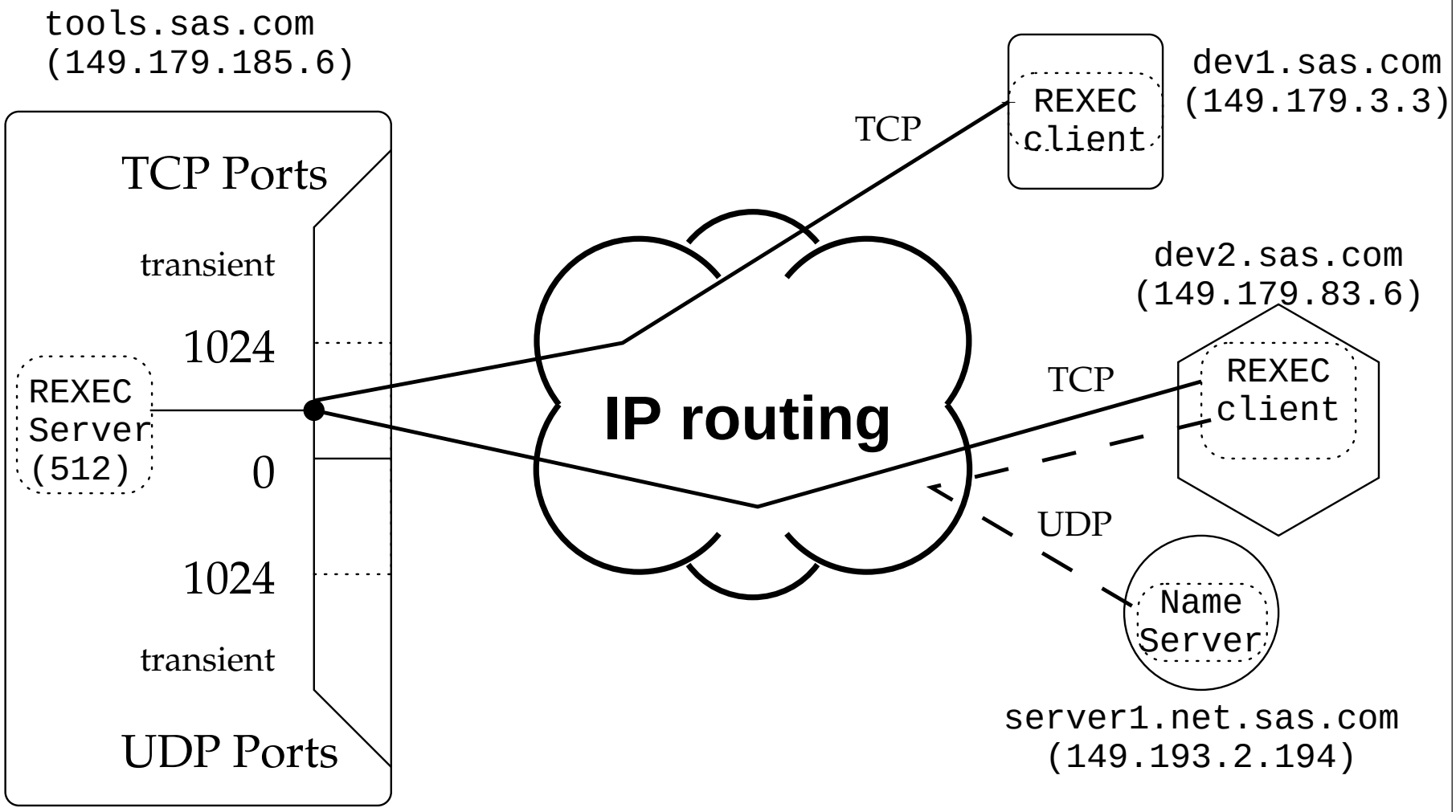
Client /Server Applications

- Peer-to-Peer Communication
- Extends applications to use network services
- A server establishes a "listen" socket on
  a "known" port (Listener/Daemon)
- A client application "connects" to the known port
  and requests a service or action
- A "service" may be iterative or concurrent

Name Resolution Processing

- Associates an IP address to a "name"
- 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
- A set of library routines called "the resolver" provide the logic to query name servers

TCP/UDP/IP Diagram



Introduction 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 for communication
- Sockets - An application programming interface (API) for interprocess communication (IPC)
- Attributes:

  Protocol Independent

  Language Independent

  Sockets implies TCP/IP and C
- Socket and Connection Association
- local host - { protocol, local-address, local-port }
- connection - { protocol, local-address, local-port , remote-address, remote-port }

Socket Library Functions

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

Primary socket calls

socket () - create 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

read()/write() - receive/send data to a socket descriptor

close () - one-way close of a socket descriptor

Network Database Administration functions

gethostbyname - given a host's name, 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 name of local host (this is actually
a system call)

getserv...(), getproto...(), getnet...(), ...

Socket utility functions

ntohs/ntohl - convert short/long from network byte order 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

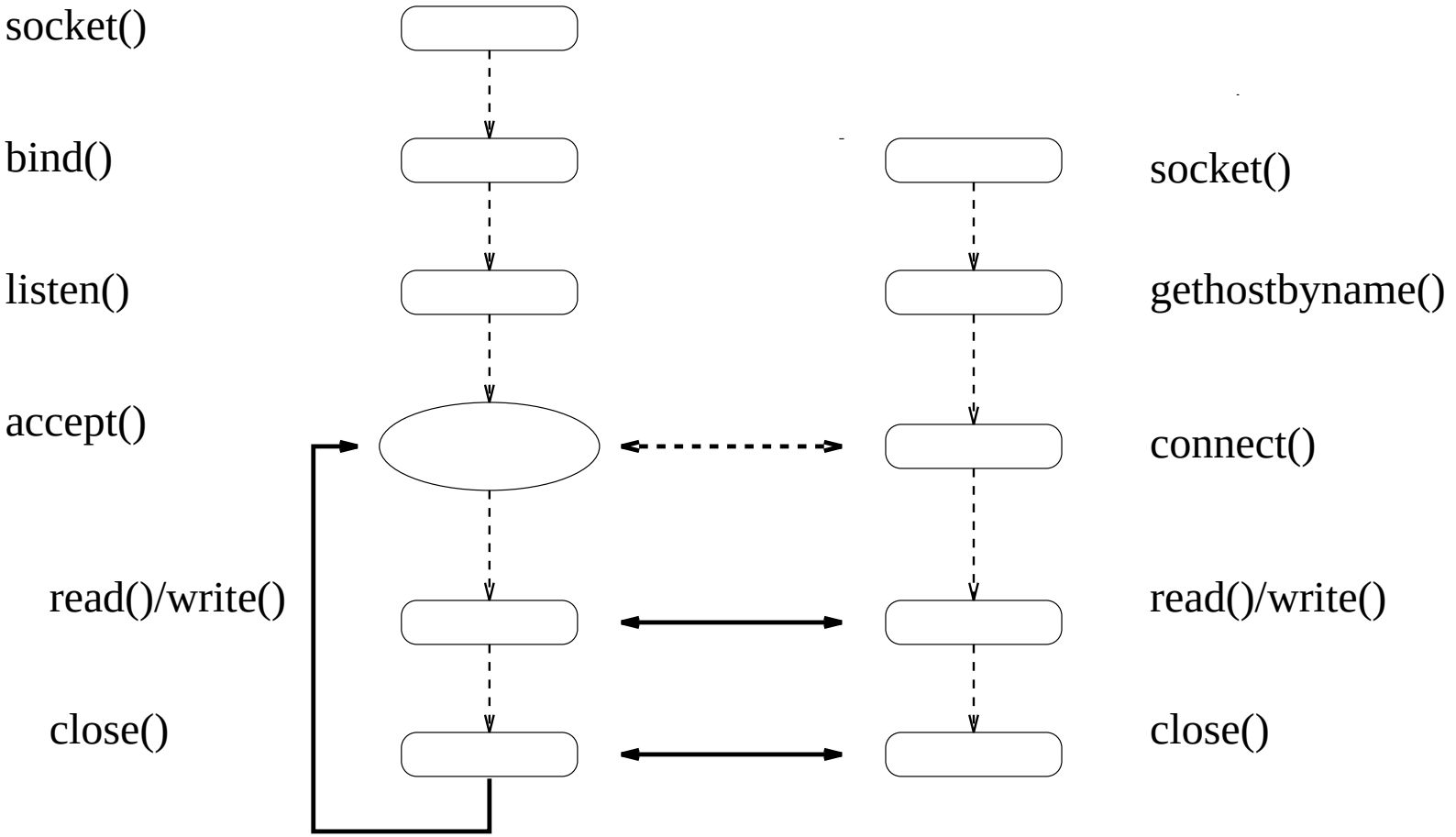
Primary Header Files

- Include file sequence may affect processing

```
<sys/types.h>   /* Prerequisite typedefs         */

<errno.h>       /* Names for "errno" values     */

<sys/socket.h>  /* struct sockaddr; system call*/
                /*   prototypes and constants  */

<netdb.h>       /* network information lookup   */
                /*   prototypes and structures */

<netinet/in.h>  /* struct sockaddr_in; byte     */
                /*   swapping macros            */

<arpa/inet.h>   /* utility function prototypes */
```

Sample TCP Client / Server Session

Iterative Server **Remote Client**



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

Beyond Sockets

- Unix System V Transport Layer Interface (TLI)
- Remote Procedure Call (RPC)
- Environment specific API's
  - Assembler, IUCV, REXX. WinSockets
  - Command Level API's
- Proprietary Protocols
- Application-oriented protocols (X, SNMP, NFS, etc.)

Design Considerations

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

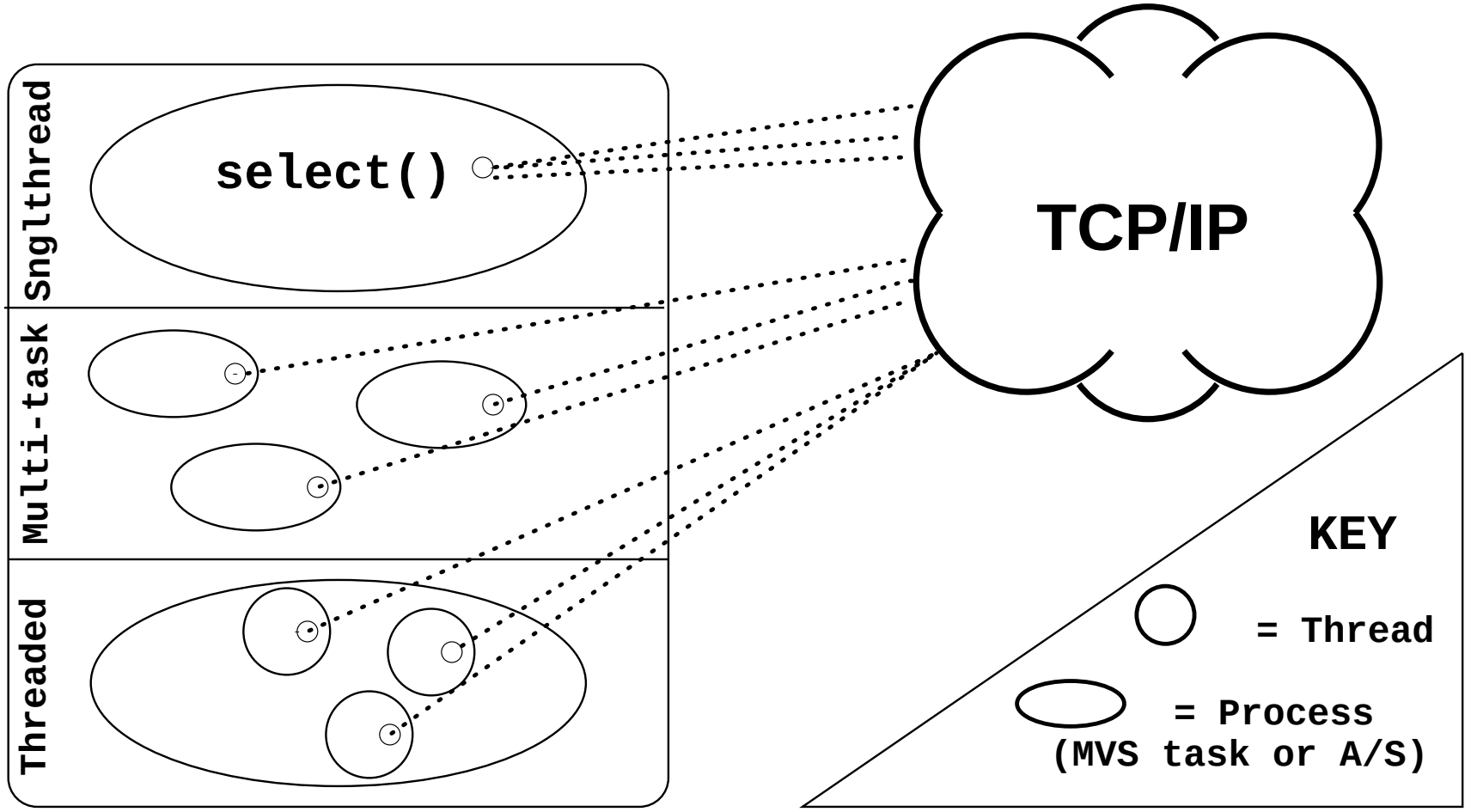
Data representation

- 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

- Reliability, data integrity-volume, transaction length
- A single-threaded design using select() allows multiple connections, test socket state
- A multi-tasking session oriented application, implements concurrency via, fork(), give/takesocket()
- A lightweight design provides concurrency using threads. The POSIX threads API is one way to accomplish this.

Server design alternatives



Security considerations

- Socket semantics do NOT address security problems:

  IP and Adapter addresses, userid/pass, 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

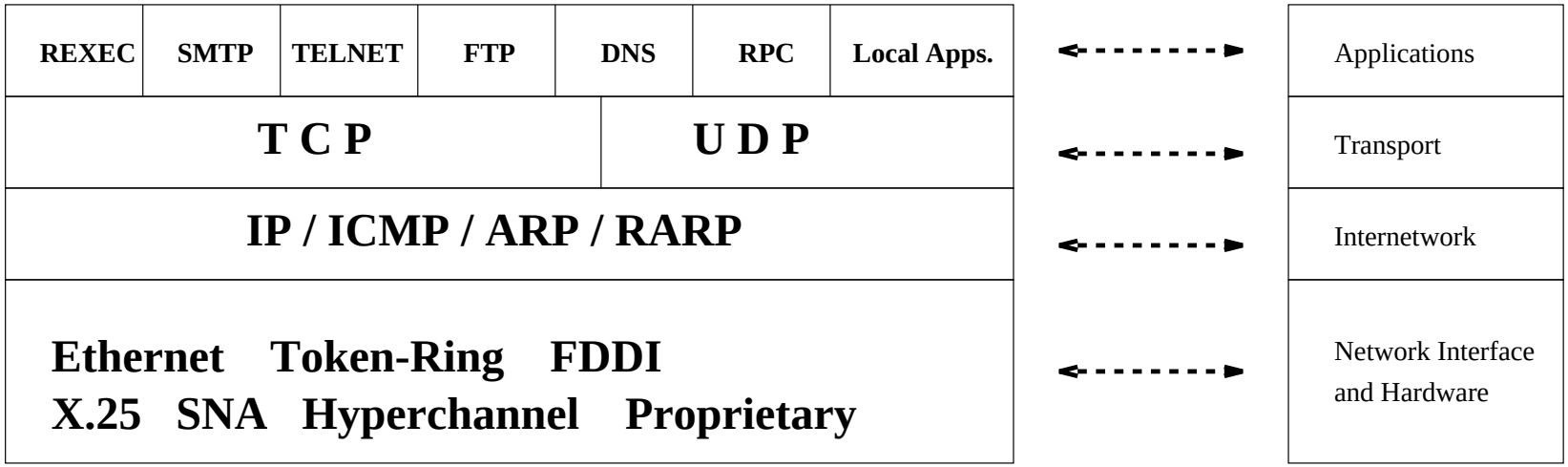
Summary

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

What's Next:

Session 5955 - Part II - Client/Server Application

Bibliography

- Internetworking with TCP/IP: Vol. I and Vol II,
  Douglas Comer, Prentice Hall, 1991
- The Whole Internet User's Guide & Catalog by Ed Kroll;
  O'Reilly and Associates.
- UNIX Network Programming by W. Richard Stevens;
  Prentice Hall, 1990.
- Socket API Programmer's Reference
- Unix "MAN" pages

TCPIP Architectural Model