1.7 OSI Model

* Open System Interconnection (OSI) Model is described by the International Organization for Standardization (ISO)
* 7 Layers- the first three layers from top are in user process space and controlled by user programs; and last four layers are in the system space and controlled by OS.

**ISO Model**

|  |
| --- |
| Applications  HTTP, Email, FTP, Telnet, Words, etc |
| Presentation   * Data encryption/decryption * Character/string conversion * Data compression * Graphic handling |
| Session  The main functions defined at the OSI transport layer include: **Tracking of communication** between the upper (application layer) and lower (network layer) from the source to the destination. This means that it separates the different applications from the packets it receives from the network layer. |
| Transport TCP/STCP/UDP   * Application identification * Client-side entity identification * Confirmation that the entire message arrived intact * Segmentation of data for network transport * Control of data flow to prevent memory overruns * Establishment and maintenance of both ends of virtual circuits * Transmission-error detection * Realignment of segmented data in the correct order on the receiving side |
| Network IP  The **network layer** is considered the backbone of the **OSI** Model. It selects and manages the best logical path for data transfer between nodes. This **layer** contains hardware devices such as routers, bridges, firewalls and switches, but it actually creates a logical image of the most efficient communication route and implements it with a physical medium. |
| Datalink Protocols  The data link layer provides the functional and procedural means to **transfer data between network entities** and might provide the means to detect and possibly correct errors that may occur in the physical layer. |
| Physical  Functions of the physical layer The physical layer is responsible for **sending computer bits from one device to another along the network**. |

Internet Protocol Suite

* Internet Protocol Suite – 4 layers: first layer contains applications, such as web server, email server, ftp, word processing. The last three layers contains transport, networking, device drivers, and hardware

|  |
| --- |
| Applications |
| TCP / UDP |
| IPv4, IPv6 |
| Device driver and hardware |

1.8 BSD Networking History

4.2 BSD (1983) first widely available release of TCP/IP and sockets API

43. BSD (1986) TCP performance improvements

4.3 BSD Tahoe (1988)

Slow start, congest avoidance, fast retransmit

BSD Networking Software release 1.0:Net/1

4.3 BSD Reno (1990) Fast recovery, TCP header prediction, SLIP header compression, routing table changes; length field added to sockaddr{}, control information added to mshdr{}

BSD Network Software Release 2.0(1991) Net/2

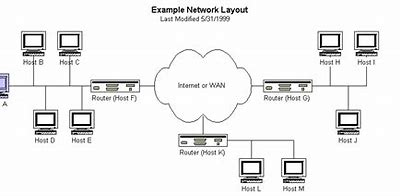
4.4 BSD (1993) Multicasting, long fat pipe modification

4.4 BSD-Lite (1994) Referred in text Net/3

4.4 BSD-Lite1 (1995)

BSD/OS, FreeBSD, NetBSD, OpenBSD

1.9 Test Networks and hosts



Notation: 172.24.37/24 means using 24 bits for networks, a host on this network could be 172.24.37.78

1. 0 – 127 – NHHH Hosts = 2^24
2. 128 – 191 – NNHH Hosts = 2^16
3. 192.223 – NNNH Hosts = 2^8

A router has at least two IP addresses, for instance:

LAN

172.24.37/24

LAN

192.164.42/24

Router connects to a network, the address connects to the LAN, another IP address in the router which is basically the same as the other LAN.

Discover Network Topology

1. Command: netstat -i or netstat -ni or netstat -nr shows network state information
2. Command: ifconfig show interface names and transmission information
3. Command ping shows host IP addresses in local network

1.10 Unix Standards

POSIX was developed by the IEEE and adopted by the ISO (International Organization for Standardization) and IEC (International Electrotechnical Company)

Open Group – X/Open Portability Guide, Issue3 (XPG3) 1989

1.11 64-bit Architecture

|  |  |  |
| --- | --- | --- |
| **Datatype** | **ILP32 model** | **LP64 model** |
| char | 8 | 8 |
| short | 16 | 16 |
| int | 32 | 32 |
| long | 32 | 64 |
| pointer | 32 | 64 |

### Exercises

|  |  |
| --- | --- |
| **1.1** | Go through the steps at the end of [Section 1.9](http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/ch01lev1sec9.html#ch01lev1sec9) to discover information about your network topology. |
| **1.2** | Obtain the source code for the examples in this text (see the Preface). Compile and test the TCP daytime client in [Figure 1.5](http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/ch01lev1sec2.html#ch01fig05). Run the program a few times, specifying a different IP address as the command-line argument each time. |
| **[1.3](http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/app05lev1sec1.html" \l "ch01a03)** | Modify the first argument to socket in [Figure 1.5](http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/ch01lev1sec2.html#ch01fig05) to be 9999. Compile and run the program. What happens? Find the errno value corresponding to the error that is printed. How can you find more information on this error? |
| **[1.4](http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/app05lev1sec1.html" \l "ch01a04)** | Modify [Figure 1.5](http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/ch01lev1sec2.html#ch01fig05) by placing a counter in the while loop, counting the number of times read returns a value greater than 0. Print the value of the counter before terminating. Compile and run your new client. |
| **[1.5](http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/app05lev1sec1.html" \l "ch01a05)** | Modify [Figure 1.9](http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/ch01lev1sec5.html#ch01fig09) as follows: First, change the port number assigned to the sin\_port member from 13 to 9999. Next, change the single call to write into a loop that calls write for each byte of the result string. Compile this modified server and start it running in the background. Next, modify the client from the previous exercise (which prints the counter before terminating), changing the port number assigned to the sin\_port member from 13 to 9999. Start this client, specifying the IP address of the host on which the modified server is running as the command-line argument. What value is printed as the client's counter? If possible, also try to run the client and server on different hosts. |