Chapter 1

Preliminaries

1.1 Introduction

All the discussions and exercises through out this course are based on the assumption that we are using Linux 2.6 Kernel or later versions based computers. We also assume that the CPU architecture is AMD64 (Intel x86_64) unless otherwise specified.

1.2 Processes

We know that a program is a sequence of instructions that we write using a programming language. A compiler or an interpreter then converts these instructions into equivalent machine code. A process is an instance of this program; in other words, a process is a program in execution. On a Linux system, we can execute a program from the shell or by performing a double-click on the executable file.

1.2.1 Process States

A process holds a set of resources such as:

1. Open files
2. Pending signals
3. Kernel data structures
4. Processor state

5. Memory address space and,
6. one or more threads

and it goes through various states from the time it is created until its termination. The various process states are:

1. New (The process creation is going on)
2. Running
3. Waiting
4. Stopped
5. Zombie

The below diagram depicts the process life cycle in a Linux system.

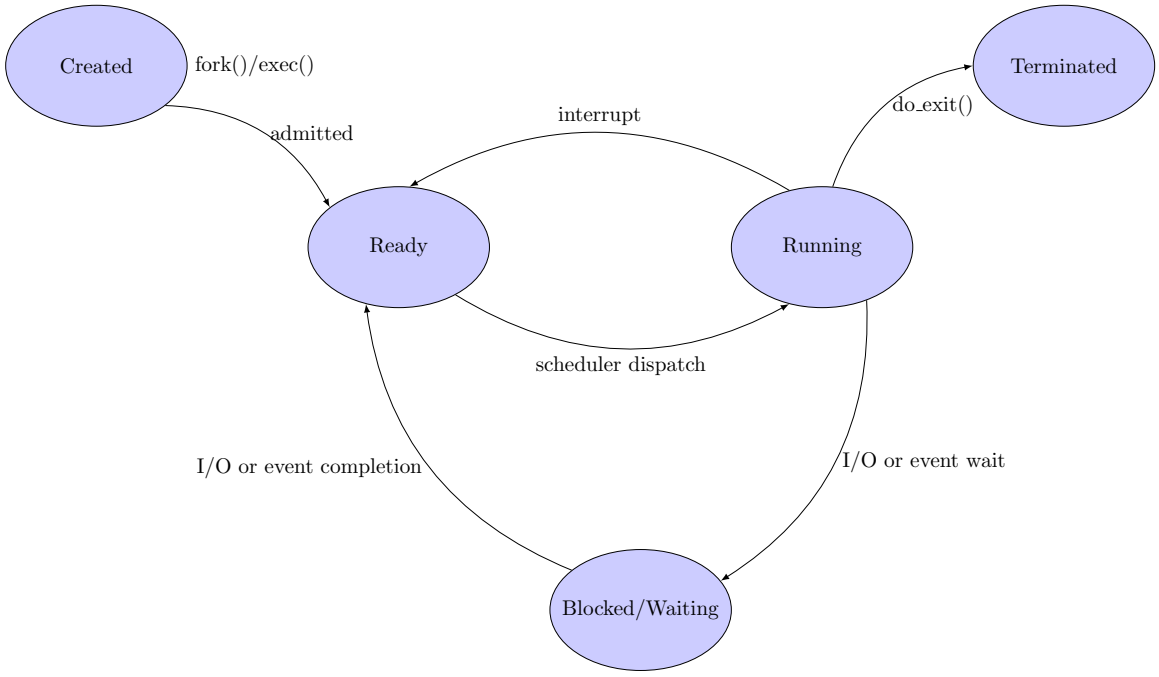

Figure 1: Process Lifecycle

Exercise 1.2.1. *Open a terminal and execute the below command. (copy/paste will not work because of a weird formatting which I need to fix.)*

```
ps -eo pid,ppid,cmd # there is no space between commas.
```

Prepare a detailed note on the output of the above command. Refer the man page of ps command. You can see the man page by executing the command manps on the terminal.

1.3 System Load Average

The most important resource in a computer is its CPU. The scheduler allocates CPU time to contending processes. In Linux, the load average has a CPU load average component and an IO load average component. The term "load" refers to the demand of a particular resource. Hence CPU load means the total number of tasks that are currently using CPU together with the number of tasks that are waiting for CPU time in the scheduler's run queue. IO load refers to the total number of tasks that are waiting for the completion of a disk operation (read/write) or other uninterruptible tasks. In Linux, there are three load average values which most tools show - last one minute average, last 5 minutes average and last 15 minutes average.

The system load average is a very critical metric when considering the scalability of a distributed system.

Exercise 1.3.1. *Open a terminal and execute the below command.*
top
Prepare a detailed note on the output of above command. Refer the man page of top command. You can see the man page by executing the command mantop on the terminal.

Exercise 1.3.2. *Save the below program into test_cpu.c (copy/paste will not work because of a weird formatting which I need to fix.)*

```
#include <stdio.h>
#include <math.h>

int main(){
while(1){
for(int i = -1; 1 > -10000; i--){
exp(i);
}
}
return 0;
}

```

To compile the above code, open a terminal and execute gcctest_cpu.c − lm.

Execute the binary file using: (copy/paste will not work because of a weird formatting which I need to fix.)

```
for i in \$(seq 1 10); do ./a.out & done
```

Execute top command.

Prepare a detailed note on the output.

1.4 Scalability

Consider a photo sharing web application. Assume this application is deployed in a server machine with a single-core processor and all the photos are stored in a disk drive local to the system. The deployment architecture of this application is given below:

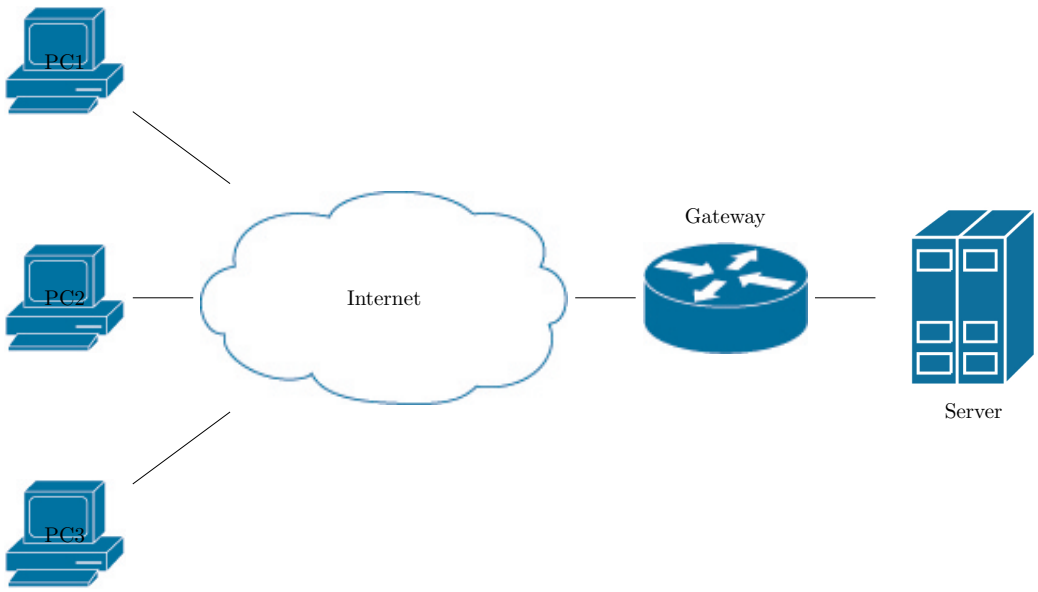

Figure 2: Single Server Web Application Architecture

Also assume this server can handle a maximum of 10 client connections (or 10 image download requests) in a second. That means each request will consume 100 millisecond of the CPU time. How do we modify our application architecture to handle more number of connections, let's say 100 connections per second? In other words, how do we make our system work reliably increased load? There are two common patterns in systems design to achieve this requirement.

1. Vertical Scaling or Scaling up
2. Horizontal Scaling or Scaling out

1.4.1 Scaling Up (Vertical Scaling)

In vertical scaling, we try to increase the number of CPU cores, RAM, disk, etc (a more powerful system). This type of scaling is not very complex, but it requires the engineers to bring down the service until the maintenance is over. Also vertical scaling is expensive as well. There is a limit on how much CPU and memory can be added to a single server. Amazon Web Services (AWS) EC2 (Elastic Compute Cloud) now provides a 128 virtual CPU , 3904GiB ***x1e.32xlarge*** instances.

x1e.16xlarge	64	179	1,952 GiB	1 x 1920 SSD	$13.344 per Hour
x1e.32xlarge	128	340	3,904 GiB	2 x 1920 SSD	$26.688 per Hour

Figure 3: x1e.32xlarge AWS EC2 instance

1.4.2 Scaling Out (Horizontal Scaling)

In this approach, we distribute the load across multiple systems. The scared out architecture of our photo sharing app is shown in Figure **??**). One can observe that the deployment architecture is more complex with the addition of a load balancer and the file server. The load balancer will distribute the incoming requests between the three back-end servers. In the single-server design, the images were served from the server machine's local disk, but in the scaled out version, we moved our images to a common file server to which all the back-end servers have access.

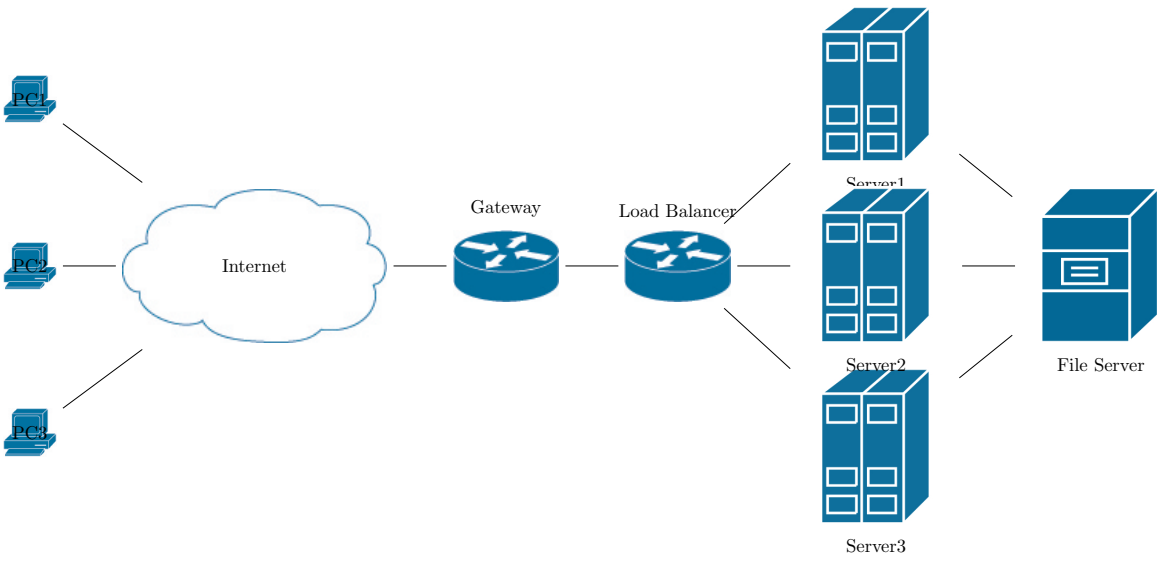

Figure 4: Horizontal Scaling

How do we implement such a file server? The rest of this chapter discusses two types of networked file systems namely NFS (Network File System) and AFS (Andrew File System). We will start with the review of Linux ***ext2*** file system.

1.5 File Systems

A file system dictates how data is stored and retrieved in a computing system. Some examples of file systems are:

1. Ext2/Ext3/Ext4 (extended-2/3/4) on Linux
2. UFS (Unix File System) on Solaris
3. FAT32 (Microsoft Windows)

When we first have a disk, we create ***partitions*** on it and then creates a file system into each partition. Once we setup the file system, we need to ***mount*** that partition in order to use it.

Unix/Linux file systems organize file in a tree structure and the root of this tree hierarchy is denoted as "/" and is called as the root of the file systems (Figure 5)). All Linux file systems support the following file types:

1. Directory - a directory is a container for other directories or non-directory files
2. Link files - soft links and hard links
3. Device files - character devices, block devices
4. pipes and FIFOs
5. Regular files - text files, binary executable files

Figure 5: Linux File System Hierarchy

File System Layout

When we create a file system on a partition, Linux creates a layout that determines how files and metadata are stored on the disk. A sample layout is below:

Boot Block	Super Block	INODEs block	Data Block

1. Boot Block - File System do not use boot block. It is used to store MBR (Master Boot Record).
2. Super Block - it stores information (metadata) about the entire file systems such as inode count, block count, free block count, etc
3. INODE Block - Metadata about every file such as file type, file mode, file user, file group, creation time, file size, etc in a special data structure called inode. INODE block contains all the inode data structures.
4. Data Block - it stores the real file content.

1.5.1 File System Operations

A file system provides APIs for managing all types of files. Some of the very important file system operations are:

1. open
2. read

3. write
4. close

Exercise 1.5.1. *Creating and mounting a file system - Steps (copy/paste will not work because of a weird formatting which I need to fix.)*

```
# 1. Install VirtualBox.
# 2. Create a new virtual machine and install any linux based OS.
# 3. Create a new disk and add it to the VM
# 4. get the details of the new disk
a1t20v21@a1t20v21-ubuntu1:~$ sudo fdisk -l | grep sd
Disk /dev/sda: 60 GiB, 64424509440 bytes, 125829120 sectors
/dev/sda1   *        2048    1050623    1048576   512M  b W95 FAT32
/dev/sda2         1052670 125827071 124774402 59.5G  5 Extended
/dev/sda5         1052672 125827071 124774400 59.5G 83 Linux
Disk /dev/sdb: 30 GiB, 32212254720 bytes, 62914560 sectors
a1t20v21@a1t20v21-ubuntu1:~$

# 5. create file system in the new disk
sudo mkfs.ext4 /dev/sdb
# 6. Create mount point
sudo mkdir /www
# 7. mount the disk to the mountpoint
sudo mount -t ext4 /dev/sdb /www
df -h
```

Exercise 1.5.2. *Network File System*

Now that we know what's a file system, we will work on setting up an NFS file system and get a feeling of how does it work. We will deep dive into the architecture of NFS in later sections. Try to work on the below exercise. Below steps illustrates how to run an NFS Server and two NFS clients. (copy/paste will not work because of a weird formatting which I need to fix.)

```
# On the Server

sudo apt update
sudo apt install nfs-kernel-server
sudo apt install nfs-common
sudo mkdir /www
sudo chmod 777 /www

add the below entry to /etc/exports
/www     192.168.68.130(rw,sync,no_subtree_check) 192.168.68.132(rw,sync,
no_subtree_check)

add the below entry to /etc/hosts.deny
```

```
rpcbind mountd nfsd statd lockd rquotad : ALL
sudo exportfs -a
sudo service nfs-kernel-server start

# On the Client

sudo apt-get install nfs-common
sudo mkdir /www
sudo mount -t nfs -o proto=tcp,port=2049 192.168.68.127:/www /www
```

1.6 OSI Model

OSI model is a conceptual 7-layer model for implementing network applications. The seven layers of the OSI model are given below:

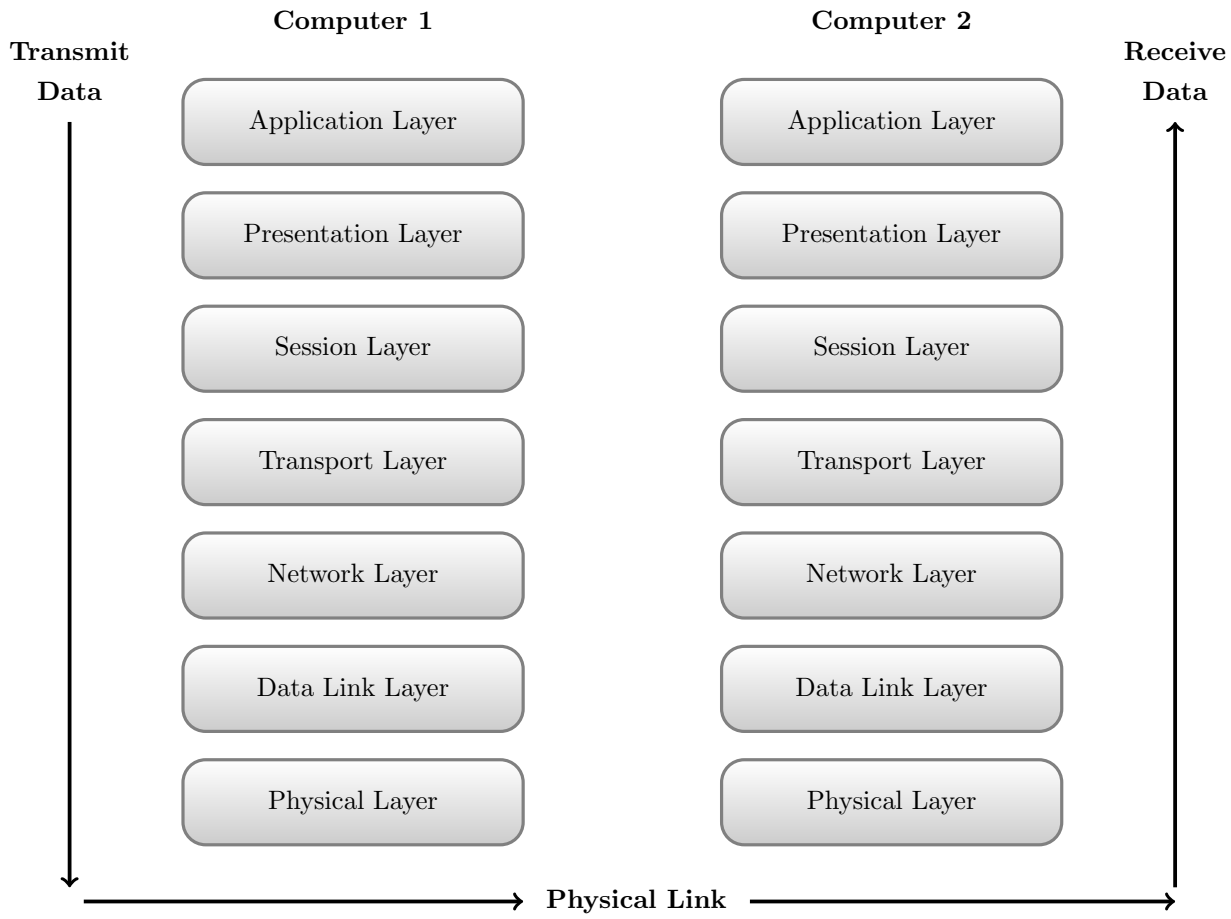

Figure 6: OSI Model

The top 6 layers define how data must be send over the wire and the physical layer defines networking hardware. Data move across the layers bidirectionally. Each layer modifies the data by adding new header information or removing headers.

1.7 TCP/IP 5-layer model

TCP/IP 5 layer model was created by researchers while a standards committee developed the OSI model.

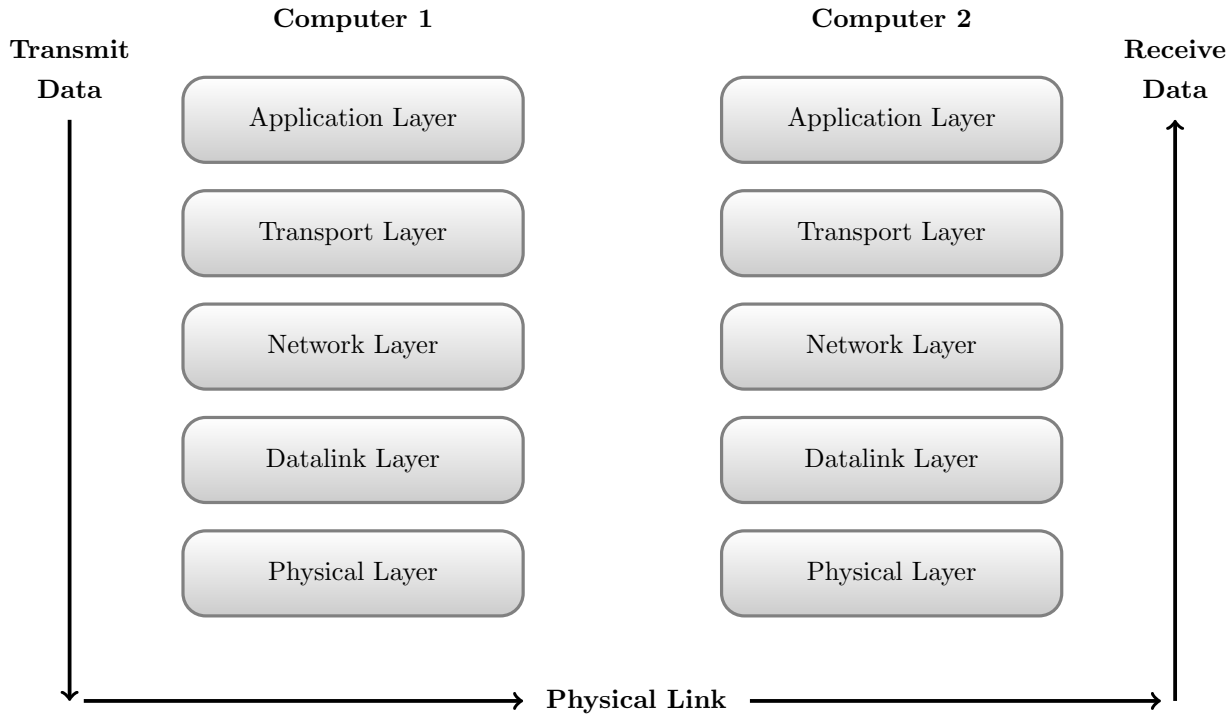

Figure 7: TCP/IP Model

1. Physical layer represents the networking hardware
2. Datalink layer (Network Interface Layer) is concerned with node to node data/packet delivery (L2 routing)
3. Network Layer (Internet layer) deals with the host to host data/packet delivery.
4. Transport layer is responsible for process to process data delivery.
5. Application layer implements user-space networking applications.
6. Data encapsulation happens at the sending machine. i.e., Data from Application layer is encapsulated with transport layer headers at the transport layer. The output of transport layer is called "Segments". Network layer encapsulate Segments with its headers to form "Packets". Datalink layer adds datalink layer headers to Packets and create Frames.
7. De-capsulation happens at the receiving machine.

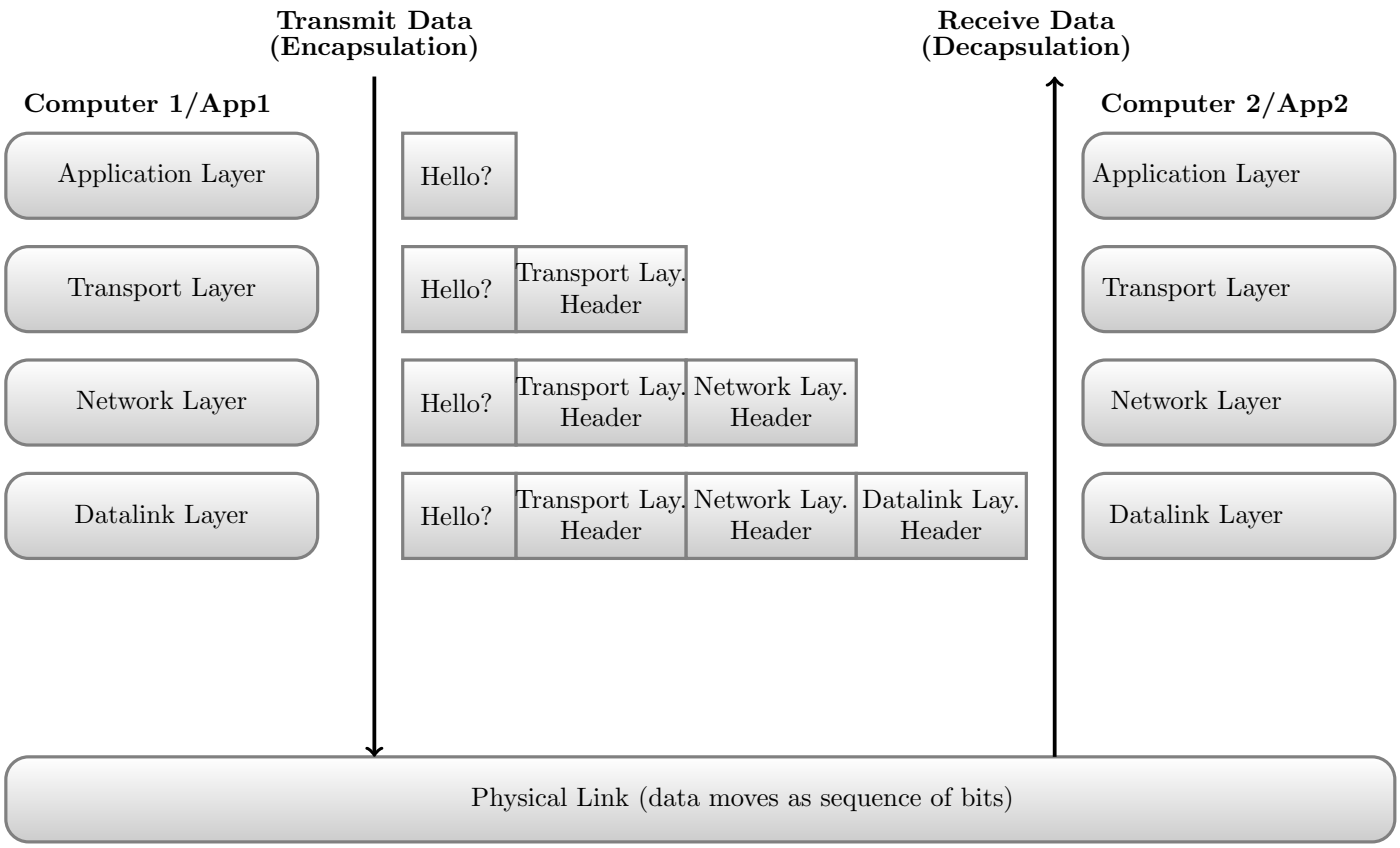

Figure 8: Data Encapsulation and decapsulation

1.7.1 IP Addressing and Subnetting

IP Address is a Network Layer concept. There are two versions of IP Addresses are available for use - IPv4 and IPv6. IPv4 addresses are 32-bits in size and IPv6 addresses are 128-bits. We are only discussing IPv4 addresses in section. Each machine connected to a network is uniquely identified by a 32-bit integer number called IP Address and we use a dotted-decimal form to represent these numbers for ease of use. Each IP address has 4 octets (1 Octet = 8bits). IP Addresses can be categorized into five classes - A, B, C, D and E. We will discuss about the first three classes below.

1. Class A Addresses:
   Range: 1.0.0.0 - 127.255.255.255
   Subnet Mask: 255.0.0.0
2. Class B Addresses:
   Range: 128.0.0.0 - 191.255.255.255
   Subnet Mask: 255.255.0.0
3. Class A Addresses:
   Range: 192.0.0.0 - 223.255.255.255
   Subnet Mask: 255.255.255.0

Exercise 1.7.1. *Convert IP Address "86.163.180.79" from dotted-decimal to integer format*

Step1: Let "86.163.180.79" = "a.b.c.d"

Step2: 86 = 01010110 = a

Step3: shift 'a' left 24 times, a = 01010110000000000000000000000000 = 1442840576

Step4: 163 = 10100011 = b

Step5: shift 'b' left 16 times, b = 10100011000000000000000 = 10682368

Step6: 180 = 10110100 = c

Step7: shift 'c' 8 times, c = 1011010000000000 = 46080

Step8: 79 = 1001111 = d

Step9: shift 'd' 0 times, c = 79

Step10: The IP in decimal = 1442840576 + 10682368 + 46080 + 79 = 1453569103

Subnets

We can divide a network into two or more smaller logical networks for better address space utilization and routing scalability. Subnetting is done using CIDR (Classless Inter-domain Routing). CIDR is based either VLSM (Variable Length Subnet Mask) or FLSM (Fixed Length Subnet Mask)

Exercise 1.7.2. *You can find this exercise at https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13788-3.html*

Given the Class C network of 204.15.5.0/24, subnet the network in order to create the network in Figure 3 with the host requirements shown

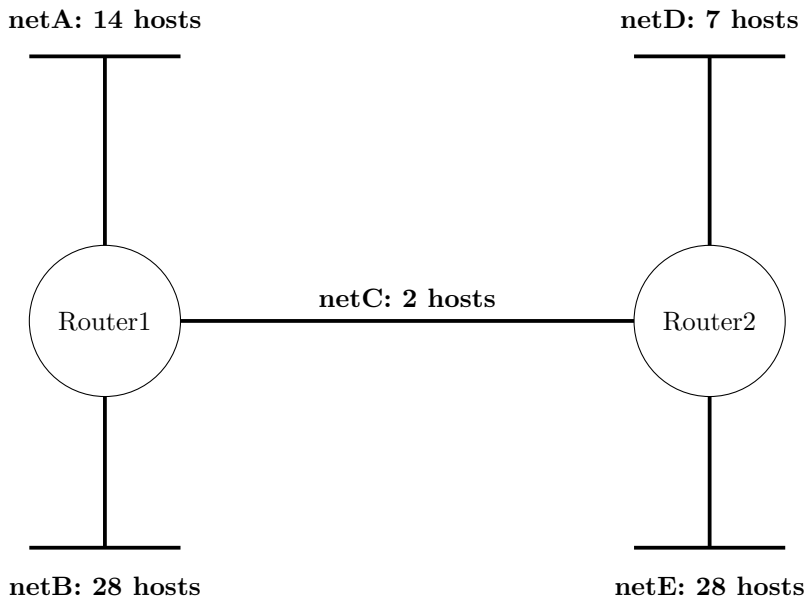

Figure 9: TCP/IP Model

References

[1] K C Wang. *Systems Programming in Unix/Linux*. Springer. ISBN: 978-3-319-92429-8.

[2] Brenden Gregg. *System Performance - Enterprise and the Cloud*. Prentice Hall.

[3] Douglas E. Comer *Internetworking with TCP/IP Volume One*. Pearson.

[4] https://www.sanfoundry.com/.

[5] https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13788-3.html