

Computer Networks

-An Introduction

Lecture-4
January 17, 2011

Rahul Banerjee, *PhD (CSE)*

Professor

Department of Computer Science

Birla Institute of Technology & Science, Pilani, INDIA

Email: rahul@bits-pilani.ac.in / Rahul.Banerjee.CSE@gmail.com

Home Page: <http://www.bits-pilani.ac.in/~rahul/>

Interaction Points

- Recollecting about need of a Computer Network / Internetwork
- Of Network Services, Service Types, Service Access Mechanisms, Service Interfaces, Service Access Points etc.
- Of Data/Instruction handling, encoding, formatting, encapsulation, decapsulation, multiplexing, demultiplexing, framing, packetization, messaging, transmitting, receiving, error-handling, forwarding, routing, translating etc.
- Revisiting Network Protocols, Protocol Stacks, Protocol Suites, Protocol Graphs, Protocol Representation, Protocol Analysis, Protocol Validation, Protocol Implementation etc.
- About Network Architectures & Network Reference Models
- Examples of Network Architectures & Reference Models
- Select References to the literature

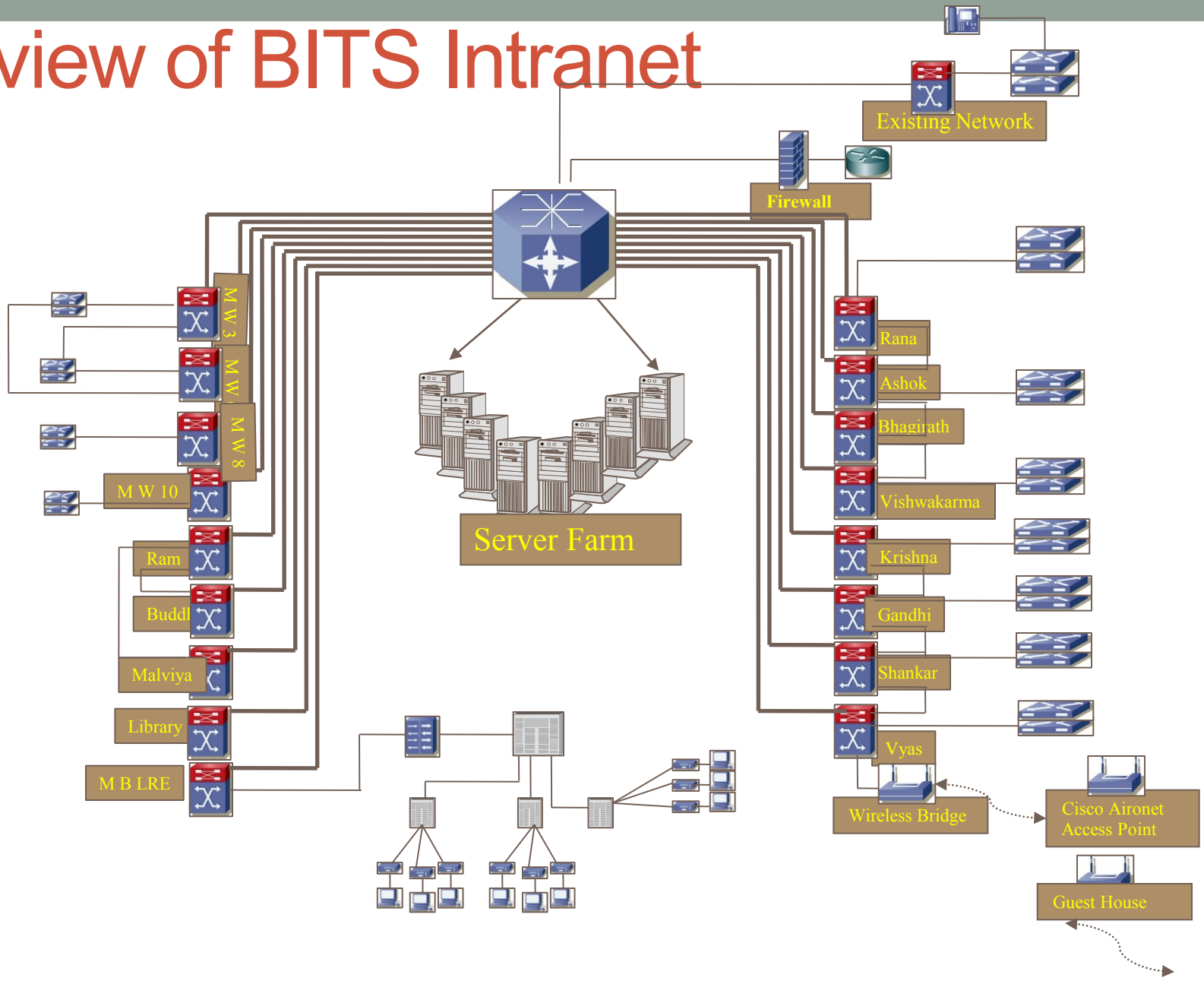
Why do we need a Computer Network / Internetwork? <recap>

- Due several reasons including one or more of the following:
 - Resource sharing (processing, storage, printing, media and much more),
 - Information exchange (data, mail, messages, audio, video and more),
 - Cost-effectiveness, ease of maintenance, availability and reliability (due to sharing and distributedness as compared to use of large single computers with several dumb terminals),
 - Support for services of common use / interest to a small or large number of people in a group or several groups

Examples of Types of Applications benefitting from Networking

- Types of applications & services:
 - hard real-time applications & services,
 - soft real-time applications & services,
 - non-real-time / best-effort / delay-tolerant applications / services
- About the significance of application-driven and economics-constrained nature of network system design approaches
- Overview of the BITS Intranet developed under *BITS Connect 1.0*
- *About the BITS Connect 2.0*

A view of BITS Intranet



(c) BITS-Pilani, INDIA

How do the Networks Handle Data?

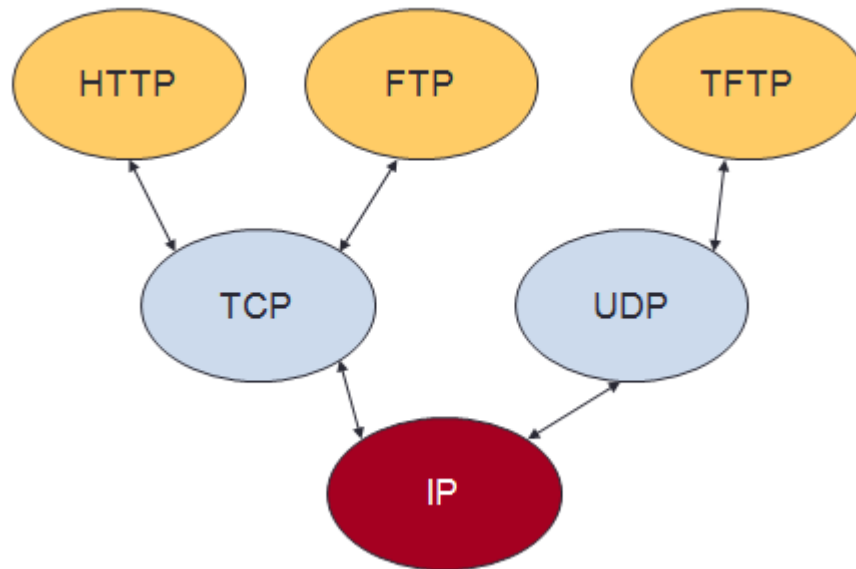
- Type and purpose of Data (synchronous data, asynchronous data, time-sensitive data, delay-tolerant data, control-data / Instruction)
- **Format of Data Representation / encoding for transmission, storage, manipulation etc.**
- About a Data Unit (character, block, frame, packet, message etc)
- **Multiplexing and Demultiplexing of Data / Instruction**
- Choosing mode, link, route / path of data transmission
- **Encapsulation and Decapsulation of data / instruction**
- Of modulation and demodulation of data over analog/digital signals
- **Error-handling (detection / correction), fragmentation, reassembly, translation and flow-control matters**

What do we mean by a Network Protocol?

- **Definition-1:** A Network Protocol is a set of rules and conventions leading to a set of pre-defined requests / commands and responses for any meaningful communication / exchange of control / information / both
- **Definition-2:** A Network Protocol may also be viewed as abstract interfaces in terms of operations associated with an interface definition in the context of one or more services on offer (along with the set of parameters, form, format, message types, meanings associated with the messages and even mechanism of handling select kinds of possible failures).
- **Definition-3:** A Network Protocol may be defined as a set of Commands / request and Responses
- **Definition-4:** A Network Protocol may be seen as a pair of modules implementing the interfaces between two layers or peers

Protocol Graphs

- A graph showing interrelation of various collaborating protocols at different levels is called a Protocol Graph.
- In a protocol graph, each protocol is represented as a node.



Protocol Representation

- There exist numerous schemes of representation of a given Protocol.
- One common way to specify a Protocol is to represent it as a State Transition Diagram.

Protocol Validation

- A Protocol needs to be proven correct before it is implemented.
- There exist quite a few ways of formal and semi-formal verification of Protocols.
- One common technique is to first represent a protocol a State Transition Diagram and then examine it for its 'completeness', 'reachability' and / or points of weakness etc.

An Example Protocol

- Let's take one of the simplest situations in which there exist two network nodes N1 and N2 interconnected through a bi-directional communication link.
- Let's assume that that it is required that primarily N1 would be sending the data to N2 using a simple mechanism over this link and only one unit of data (frame) would travel at a time over the link.

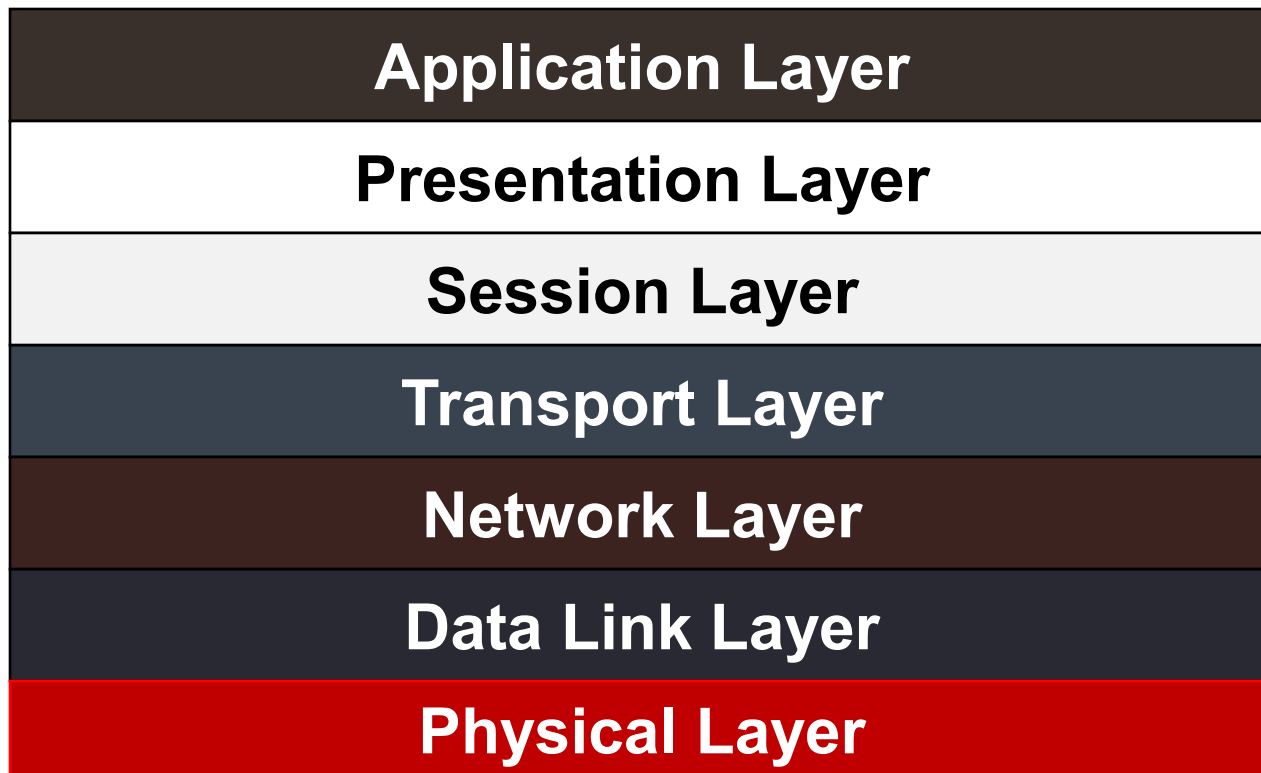
A Few Measures of Network Performance

- *Performance Measures*
 - *Available Performance*
 - *Measured Performance*
- *Bandwidth:*
 - *Width of the usable / allotted Frequency band*
 - *Rate of data transfer in bits per second*
 - *Throughput: Actual measured rate of achievable data transfer in bits per second*
 - *Bandwidth has often a value greater than that of the Throughput*
- *Round-Trip Time (RTT)*
- *Latency: Delays of various kinds*
- *'Delay x Bandwidth' metric*
- *Quality of Service*

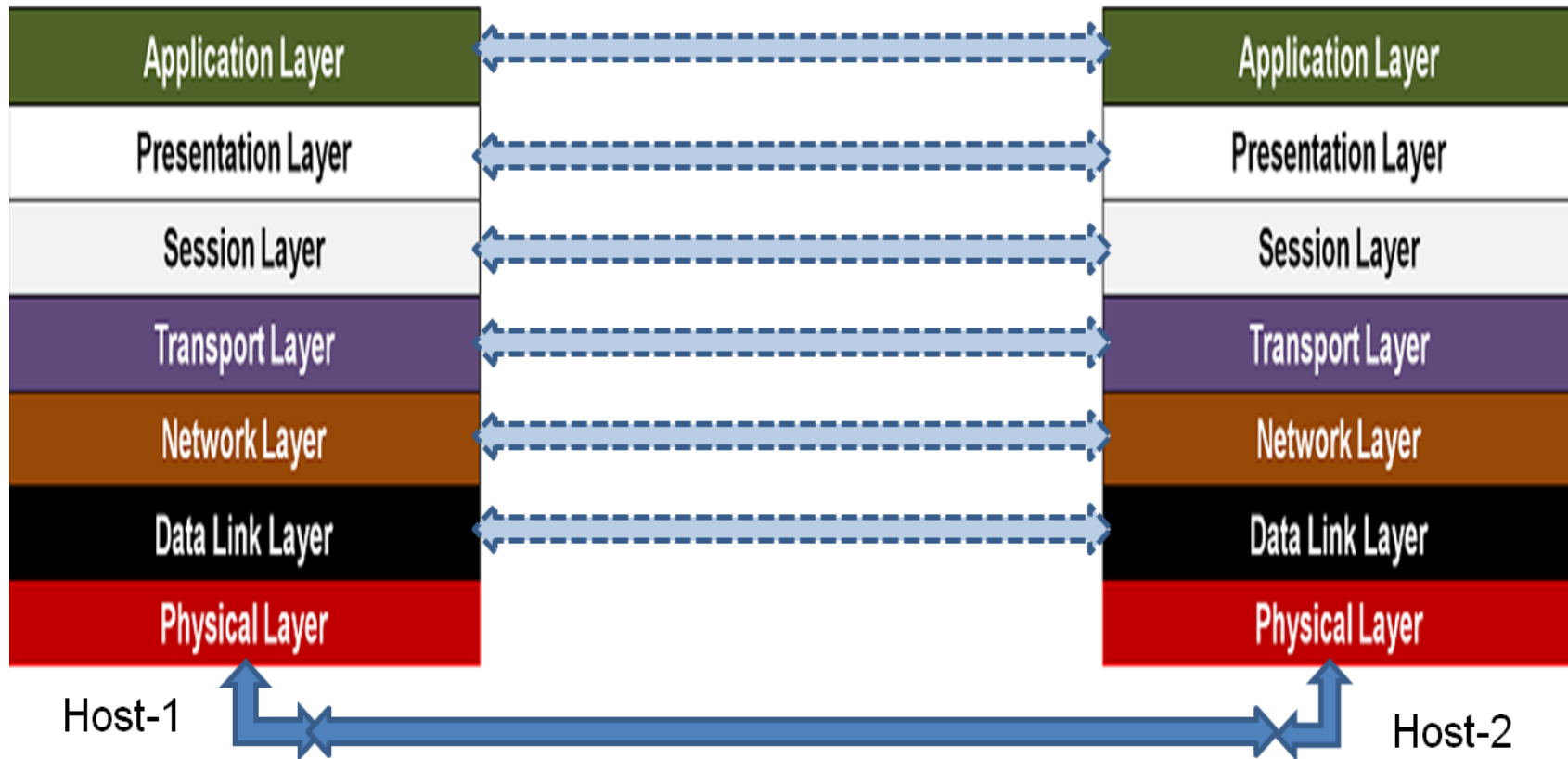
Network Architecture & Reference Models

- Architecture versus Reference Model: A simplistic perspective:
 - Network Architecture:
 - It may be seen as a detailed *generic blueprint* with unambiguous definitions of *services*, *interfaces*, *organization* and defined *protocols* that helps in design and implementation of a set of relevant protocol stack / suite based network / internetwork
 - Network Reference Model:
 - It is the same as the architecture minus the specifically defined readily usable protocols.
- Examples:
 - TCP/IP Architecture
 - OSI Reference Model & OSI Architecture

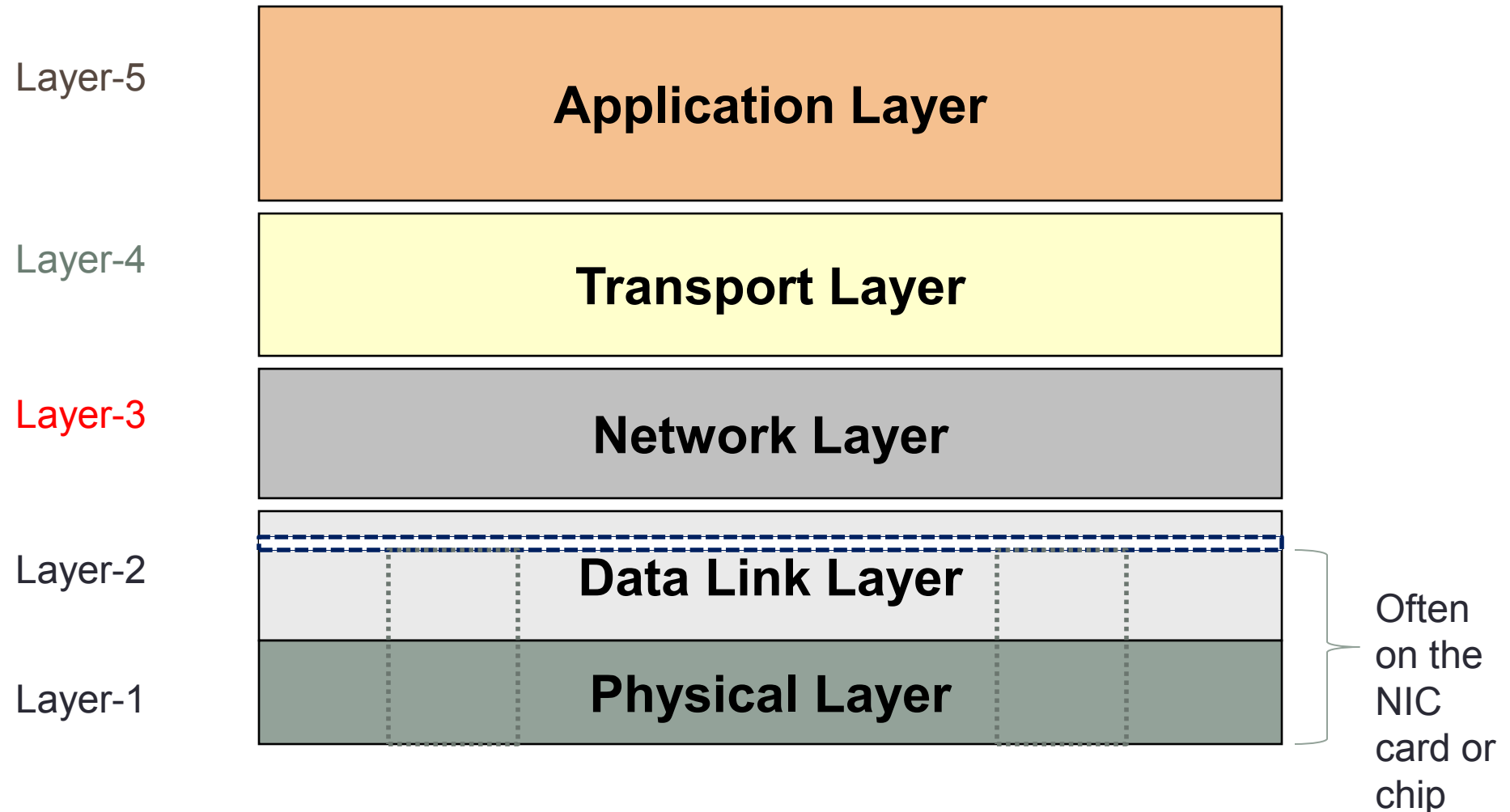
The ISO OSI Reference Model



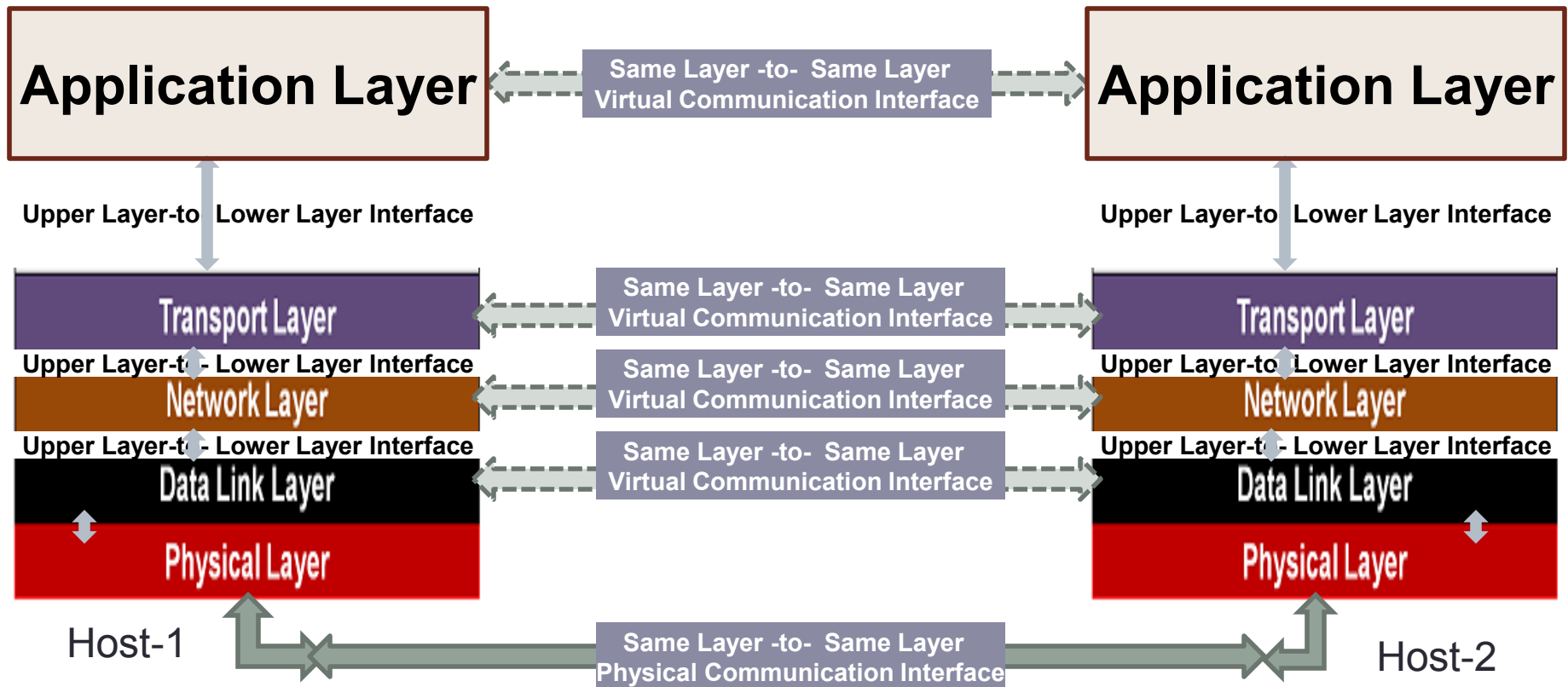
Interactions within and between the Nodes using the ISO OSI Reference Model



A 5-Layer Hypothetical Network Reference Model for Easy Conceptual Understanding



Interactions within and between the Nodes using the 5-Layer Hypothetical Network Reference Model <for Instruction>



Application Layer

- Application Layer is a layer of the Network Architecture that is primarily concerned with getting TPDUs from the lower layer (usually Transport Layer) and delivering it to the Application and vice-versa (with or without explicit presentation and session management support).
- Examples: HTTP, DHCP, DNS, SNMP, FTP (in the context of the TCP/IP Architecture).
- Web-services, Video-on-Demand over the network, Video/Voice-conferencing over the network etc. are examples of Applications that reside atop the protocols belonging to this layer.

Application Layer Responsibilities

- **It primarily deals with:**
 - Accepting messages from the Application Layer through the APIs
 - **Processing these messages and generating APDUs**
 - Deciding transport connection requirements (for further transmitting this DU after encapsulating it within an APDU)
 - **Passing this packet through the SAP to the lower layer (TL)**
- **It also deals with ...**
 - Accepting APDU from the lower layer through the SAP
 - **Processing the APDU**
 - Removing the encapsulation and passing the messages to the respective destination application
 - **Provide diagnostic support for network monitoring, configuration, management and trouble-shooting at the Application Layer or lower layer**

Transport Layer: What is it?

- Transport Layer is a layer of the Network Architecture that is primarily concerned with:
 - getting TPDU from the upper layer (usually Application Layer) and
 - delivering it to the same layer at the intended destination node (through the underlying Network Layer).
- Converse is also true of the targeted set of responsibilities of this layer.

Transport Layer Responsibilities

<another perspective>

- **It primarily deals with:**

- Accepting APDU from the Application Layer through the Service Access Point (SAP)
- **Processing these APDU**
- Deciding transport connection requirements (for further transmitting this DU after encapsulating it within a TPDU)
- **Passing this packet through the SAP to the lower layer (NL)**

- **It also deals with ...**

- Accepting TPDU from the lower layer through the SAP
- **Processing the TPDU**
- Removing the encapsulation and passing the messages to the respective destination application
- **Provide diagnostic support for network monitoring, configuration, management and trouble-shooting at the Application Layer or lower layer**

Network Layer

- Network Layer is primarily concerned with getting NLDU / Packets from the source node and delivering it to the intended destination node (through none or many intermediate nodes).
- Additional responsibilities of this layer include:
 - Providing support for connection-oriented / connectionless services as the case may be (depending upon the protocol stack and need)
 - Provide diagnostic support for network monitoring, configuration, management and trouble-shooting at the Network Layer or higher layer.
- Packet handling, packet management, Routing are its major responsibilities.
- In the context of packet routing, network layer structural design goals include:
 - Ensuring the shortest possible delay and thereby the highest throughput at the least possible cost
 - Ensuring acceptably reliable packet delivery (may be optional in some cases)
 - Ensuring secure packet delivery (may be optional in some cases)

Data Link Layer

- **Data Link Layer consists of two sub-layers:**
 - Media Access Control (MAC) sub-layer &
 - Logical Link Control (LLC) sub-layer.
- **Major Issues involved in the design of the Data Link Layer include:**
 - Which services are to be provided to each of the adjacent layers?
 - Exactly when to provide these services?
 - How to provide them?
 - To whom should they be provided?

Physical Layer

- Physical Layer deals with transmission of raw digital data using analog or digital signal.
- This layer is concerned with the logic type (negative or positive), amplitude of the signal, signal representation, bit-length, direction of transmission etc.
- It deals with connection-establishment and termination.
- This layer is, in a nutshell, a layer that deals with various electrical and mechanical characteristics of every physical component of a computer network.
- Exact electrical, mechanical and procedural Interface Definition is therefore its responsibility.
- Choice and use of the physical medium are the Physical Layer Design Issues.

Any question please?

Thank you for your kind attention!

For further details, you may contact at:

E-mail: rahul@bits-pilani.ac.in / rahul.banerjee.cse@gmail.com

or visit:

Home: <http://www.bits-pilani.ac.in/~rahul/>

References

- *Larry L. Peterson & Bruce S. Davie: Computer Networks: A Systems Approach, Fourth Edition, Morgan Kaufmann / Elsevier, New Delhi, 2007. <System design approach>*
- *S. Keshav: Computer Networking: An Engineering Approach, Pearson Education, New Delhi, 1997.*
- *A. S. Tanenbaum: Computer Networks, Fourth Edition, Pearson Education, New Delhi, 2003. <Conceptual Approach>*
- *Y. Zheng and S. Akhtar: Networks for Computer Scientists and Engineers, Oxford University Press, New York, 2002. <Structural approach>*
- *A. Leon Garcia and I. Widjaja: Communication Networks: Fundamental Concepts and Key Architectures, Second Edition, Tata McGraw-Hill, New Delhi, 2004.*
- *Mohammed G. Gouda: Elements of Network Protocol Design, Wiley Student Edition, John Wiley & Sons (Pte.) Ltd., Singapore, 2004.*
- *Thomas G. Robertazzi: Computer Networks and Systems: Queuing Theory and Performance Evaluation, Third Edition, Springer-Verlag, New York, 2000. <Analytical approach>*



Approaches to Writing Network Software

- **Define end points**
- Engage them based on assured availability or request without ensuring availability in advance
- **Send / Receive data (optionally verify integrity)**
- Terminate engagement if explicitly established, no need to terminate if no exclusive engagement was established in the first place
- **Reliability often has an associated price**
 - Example: reliable transport mechanisms
- **Unreliable communication is often faster, low-overhead but do not suit data transfers where information loss / change in arrival order is unacceptable**
 - Example: unreliable transport mechanisms
- **More later! ...**



About Application Client / Server Processes

Definition of an Application Server Process:

A process that provides any set of predefined services to one or more requesting clients is called a Server Process.

Types of Application Server Processes:

Concurrent Server Process

A process that simultaneously provides any set of predefined services to one or more requesting clients is called a Concurrent Server Process.

Iterative Server Process

A process that provides any set of predefined services to only one requesting client at any point of time is called an Iterative Server Process.

Definition of an Application Client Process:

A process that solicits any specific service from any designated server is called a Client Process.



The TCP/IP Access Points

In a TCP/IP network, services offered by any layer can only be used through parameter / data passing through the various Service Access Points (SAPs) located on Layer-boundaries.

TSAPs and NSAPs are two major examples of SAPs located at the AL-TCP/UDP Layer-boundary and TCP/UDP-IP Layer-boundary respectively.

- A typical example of an NSAP is an IP address.
- A typical example of a TSAP is an IP address + a 16-bit integer called as Port Number.

Port Numbers permit unique identification of several simultaneous processes using TCP / UDP.

IETF's RFC 1700 lists select IANA-suggested Port Number Assignments.

Network Programming in MS Windows environments

Methods of Network Programming in MS-Windows

- Windows Network Programming support exists for:
 - TCP/IP (IETF's)
 - Named Pipe <FIFO>, <server creates a pipe and awaits, allowing bi-directional data-flow>
 - Mailslots <uni-directional guaranteed or non-guaranteed data-flow, uses broadcast datagrams, can be created by any process>
 - IPX/SPX (Novell's Netware is based on this stack)
 - NetBEAU (NETBIOS Extended User Interface) <for PC-based networking, exposes nterfaces for reading / writing data, can be used for C/S or P2P, connection-oriented or connection-less way>
 - AppleTalk (Apple's)

Methods of Network Programming in MS-Windows

- **Methods available:**
 - **Windows NT Remote Procedure Call (RPC)** <uses a collection of libraries / tools with a focus on the Application Procedure alone, targeting execution on a remote machine, independent of transport protocols>
 - **Windows Socket (WinSock)** <uses Windows Socket interface, extension of BSD Sockets, allows bi-directional data-flow, upon completion of asynchronous I/O, exports functions which transmit messages to the relevant applications: feature fits within the message-driven interface model of MS-Windows>
 - **WinNet API** <for over-the-MS-Windows-network sharing of file-server / printer etc.>

Network Programming in Linux / UNIX and similar environments

The Sockets and Socket Pairs

- Socket is another name for the TSAP discussed above. Thus, it comprises of IP Address+Port Number.
- Sockets are often created by the `socket()` system call.
- A Socket Pair refers to a set of socket endpoints at the two communicating ends of a TCP/IP network.
- Typically, a Socket-Pair, for TCP as well as UDP, comprises of four components:
 - IP Address-1 & Port Address/Number-1
 - IP Address-2 & Port Address/Number-2
- In case of TCP, a Socket-pair typically uniquely represents a TCP-connection (logical).

Creating a Socket

- Every network protocol has its own definition of *Network Address*.
- In C, a protocol implementation provides a *struct sockaddr* as the elementary form of a *Network Address*.
- A sample definition of *struct sockaddr*

```
» #include <sys/socket.h>

» struct sockaddr {
» unsigned short sa_family;
» char sa_data [MAXSOCKADDRDATA]
» }
```

Creating a Socket

- In UNIX and Linux, Sockets are created by the `socket()` system call.
- This call returns a *file descriptor* for the Socket that is yet to be initialized.
- Then the socket is initialized by binding it to a particular protocol and address using the `bind()` system call.

```
#include <sys/socket.h>
```

```
int socket (int domain, int type, int protocol);
```

Here,

the parameter domain specifies the PF;

parameter type usually specifies either of SOCK_STREAM or SOCK_DGRAM;

parameter protocol specifies the protocol to be used (0=> default protocol associated).

```
int bind (int sock, struct sockaddr * my_addr, int addrlen);
```

- Here, the parameter `sock` is the socket-in-question; parameter `sockaddr` is the address of protocol; parameter `addrlen` is the length of the address for the local end-point.

Creating a Socket

- Next, *listen()* system call is executed for informing the system that the process is now ready to allow other processes establish a connection to this socket at the specified end-point.
- This step does not really establish a connection by itself, however!
- Now, the *accept ()* system call is made for accepting the connection requests.
- *accept ()* is a blocking call as it blocks until a process requests a connection. In case, the socket has been marked as 'non-blocking' by the *fcntl()* call, *accept()* would return an error if no process is requesting it for a connection.

```
#include <sys/socket.h>
```

```
int listen (int sock, int backlog);
```

Here,

the parameter `sock` is the socket-in-question;

parameter `backlog` is the number of connection requests that may be pending on the socket before any further connection requests are explicitly refused.

```
int accept (int sock, struct sockaddr * addr, int * addrlen);
```

- The *select ()* system call can also be made for determining if any connection request is currently pending to a socket.
- Similarly, a Client attempts to connect to a Server by creating a socket, binding it to an address (optionally) and making the *connect()* call to the Server at the known address.

A Glimpse of Address and Protocol Families for Various Stacks

- **Address Families:**
 - Unix / Linux Domain: AF_UNIX
 - TCP/IPv4 Domain: AF_INET
 - TCP/IPv6 Domain: AF_INET6
 - Novell NetWare Domain: AF_IPX
 - AppleTalk Domain: AF_APPLETALK
- **Protocol Families:**
 - Unix / Linux Domain: PF_UNIX
 - TCP/IPv4 Domain: PF_INET
 - TCP/IPv6 Domain: PF_INET6
 - Novell NetWare Domain: PF_IPX
 - AppleTalk Domain: PF_APPLETALK

The IP Socket Address Structure

```
struct in_addr {
    in_addr_t s_addr;          /* Big Endian 32-bit IP address */
};

struct sockaddr_in {
    uint8_t sin_len;          /* 16-byte structure length */
    sa_family_t sin_family;   /* AF_INET */
    in_port_t sin_port;       /* 16-bit Big Endian TL Port */
    struct in_addr sin_addr;   /* 32-bit Big Endian IP address*/
    char sin_zero [8];
};

/* This is defined in netinet/in.h file. */
```

On the Byte-ordering and Byte-manipulation Functions

- The term Byte-ordering refers to the manner in which a multi-byte / multi-octet number / field is stored in Lower-order (i.e. starting address) and Higher-order Memory Addresses.
- If the Lower-order Byte is stored in the Starting / Lower-order Memory Location and Higher-order Memory Location, then it is called the Little Endian scheme.
- If the storage is carried out in exactly opposite manner, it is called the Big Endian scheme.
- Different manufacturers may choose any one of these Byte-ordering Schemes. For instance, traditionally, Motorola processors have followed the Big Endian scheme whereas the Intel processors have used the Little Endian scheme.
- The Network Byte-order is always Big Endian, by convention whereas the Host Byte-order may be of either type.
- *None of these two schemes is superior or inferior to the other since they merely represent two possible ways in which Lower and Higher Order Bytes are stored into and retrieved from Memory or transferred over a network link.*

On the Byte-ordering and Byte-manipulation Functions (Contd.)

- Over a network, between any two nodes, data transfer takes place in Big Endian manner.
- There are standard function prototypes for 'Host-Byte-order to Network-Byte-order' conversion and 'Network-Byte-order to Host-Byte-order' conversion.
- These functions are defined in the `netinet/in.h` header file in the systems supporting Socket Programming in C and its variations. Their usage has been shown below:

```
#include <netinet/in.h>
uint16_t htons (uint16_t          16-bit-host-address)
uint32_t htonl (uint32_t          32-bit-host-address)
uint16_t ntohs (uint16_t          16-bit-network-address)
uint32_t ntohl (uint32_t          32-bit-network-address)
```

- There exist two categories of functions those are capable of operating on multi-byte / multi-octet numbers / fields.
 - The first category has BSD functions like `bcopy`, `bzero` etc. Functions belonging to this category begin with 'b' (b=> byte).
 - The second category has ANSI-C functions like `memset`, `memcpy` etc. Functions belonging to this category begin with 'mem' (mem=> memory). Both types are defined in the header files `strings.h` and `string.h` respectively.

On the Functions used for Address Conversions

- A few functions used for IPv4 Address conversion from / to ASCII string and Binary strings expressed in the Network Byte-order have been defined in `arpa/inet.h` header file. These include: `inet_pton`, `inet_ntop`, `inet_aton`, `inet_ntoa`, `inet_addr` etc.
- Here, `inet`=> internet, `a`=> ASCII, `n`=> network, `addr`=> address, `ntoa` and `aton` refer to network to ASCII and ASCII to network respectively.
- The first two of these functions can handle IPv4 as well as IPv6 addresses and are therefore commonly used these days since majority of implementations attempt to offer dual-stack compatibility.
- It is advisable to avoid use of `inet_addr`, `inet_aton` and `inet_ntoa` as these have known problems as well as are considered deprecated. In near future, they may not be supported.
- Using the preferred functions:

```
#include <arpa/inet.h>
int inet_pton ( int <addr-family>, const char *<strptr>, void *<addrptr> );
const char *inet_ntop ( <addr-family>, const void *<addrptr>, char *<strptr>, size_t <len> );
/* Here, addr-family may be AF_INET or AF_INET6 whichever needed. */
```

On the Functions used for Address Conversions (Contd.)

- As may be noticed in the syntax, `inet_ntop` requires *char *<strptr>*, this in turn requires to get the binary address that is stored as a part of a Socket Address Structure. This would mean some degree of protocol dependence of the function to be written since the programmer must know the said structure beforehand.
- ```
const char *inet_ntop (<addr-family>, const void *<addrptr>,
char *<strptr>, size_t <len>);
```
- This problem may be easily taken care of if the programmer writes his own function that extracts this data and its presentation format and supplies it to the standard function.
- There are numerous other similar situations in which such custom-built functions, if written by the programmer, increase the level of protocol independence of the resulting code. This, however, comes with its own overheads and may not be a preferable approach where speed and efficiency are the primary requirements.

# On the Functions used for I/O Operations on Stream Sockets and POSIX-compliant Status Information

- Two points are important whenever a programmer wishes to perform various read and write operations on Stream Sockets. (*Stream here refers to Byte Stream as in case of TCP Sockets.*)
  - A read operation or a write operation, even without an error, may read or write lesser number of bytes than explicitly mentioned particularly when a Buffer gets full. This does not mean that inaccurate result needs to be accepted; instead, it simply requires that the function is called again in order to read or write the rest of the bytes subsequently.
  - This may happen during all read as well as all non-blocking write operations.
- The `fstat` function has been used traditionally for getting the information about any specified descriptor. This function, along with others has a prototype in the `sys/stat.h` header file.
- Another function `isfdtype` has been more common in use of late. This is used as follows:

```
#include <sys/stat.h>
int isfdtype (int <sock-fd>, int <fd-type>);
Here, fd=> file descriptor.
```

# The TCP/IP Tips

- Syntax of the socket function that is required to create an end-point has been discussed earlier as:
  - `#include <sys/socket.h>`
  - `int socket (int <add-family or proto-family>, int <socket-type>, int <protocol>);`
- Unlike the Server, described earlier, the Client need not invoke the bind function call. In case of TCP over IPv4, this function, as discussed before, assigns a 32-bit+16-bit address to a Socket created by the socket call. Server processes normally prefer to be assigned a standard-convention-based port called Well Known Port. Whenever used, syntax of the function call is is:
  - `int bind (int <sock-fd>, const struct sockaddr *<assigned-addr>, socklen_t <addr-len>);`
- Syntax of the connect function that is needed by a Client to initiate a connection-request to a Server is:
  - `int connect (int <sock-fd>, const struct sockaddr *<server-addr>, socklen_t <addr-len>);`

# The TCP/IP Tips (Contd.)

- Just like bind, another function that is never invoked by the Client is the listen function. Syntax of the listen function that is required to make an unconnected Active Socket of a Server process behave as a Passive Socket entity has been discussed earlier as:
  - `#include <sys/socket.h>`
  - `int listen (int <sock-fd>, int <composite-connect-queue-len>);`  
Where, composite-connect-queue-len=> sum of complete+incomplete connection queues.
- Like bind and listen, another function that is called only by the Server is the function accept. As shown earlier, its syntax is:
  - `#include <sys/socket.h>`
  - `int accept (int <sock-fd>, struct sockaddr *<client-addr>, socklen_t *<addr--len>);`

# Tips on Programming a Concurrent Server

- Under UNIX and Linux environments, typically, the only available way to create a new process is by invoking the fork function that creates a Child Process.
- This function returns twice per invocation – once in the Parent Process that invoked it and thereafter in the created Child process. The first time it returns a value to the Parent that is actually the PID of the Child and second time it returns a value of zero to the Child.
- Syntax of its usage is:

```
#include <unistd.h>
pid_t fork(void);
```
- Interestingly, this function can be used for a different reason altogether, i.e. it can be used in association with the exec function whenever it is desired that a given process must invoke another process.
- *In this case, first a replica / child process is created by the fork call which then replaces itself with the process named as an argument to the exec function call. The exec has six variations (?).*
- Syntax of use of exec is:

```
int exec?(const char *<file-or-path-name>, <argument-list>, <...>);
```

# Tips on Programming a Concurrent Server (Contd.)

## Steps involved in writing a simple Concurrent Server are:

- Create a routine that creates a Socket, binds it to a well known address, changes it to listen mode and waits for a connection request at this address from a Client. (socket+bind+listen thus form Step One!)
- Once a connection request is received, based on the queuing status, this routine has to ensure that following events occur:
  - The accept call returns,
  - The fork call is used to spawn a Child,
  - Let the Child close the listening Socket, service the Client etc. While in the mean time, the Parent returns to its passive ('listening') status and awaits the next request.
  - Once the Child has serviced the Client, it closes all open descriptors and exits,
- Finally, the routine must ensure that the Parent closes the connected Socket. This completes the cycle.
- Syntax for a UNIX / Linux close is:

```
#include <unistd.h>
int close (int <sock-fd>);
```

# A Few More Functions of Importance

Certain other functions of relevance include:

- ioctl, fcntl, recvfrom, sendto, signal,
- select, poll, shutdown, pselect,
- getsockname, getpeername, getsockopt,
- setsockopt,
- gethostbyname, gethostbyname2, gethostbyaddr,
- uname,
- gethostname, getservbyname, getservbyport, getaddrinfo, getnameinfo
- gai\_strerror, freeaddrinfo

# Summary

- In TCP/IP setup, services specific to any layer can only be accessed through Service Access Points located at the layer-boundaries.
- A TSAP is an *IP address+ a 16-bit Port Number*.
- Typically, a Socket-Pair, for TCP as well as UDP, comprises of Server IP Address, Port Address / Number of the Server, Client IP Address, Port Address / Number of the Client.
- In case of TCP, a Socket-pair typically uniquely represents a TCP-connection.
- Each network protocol may have its own definition of *Network Address*.
- The Network Byte-order is always Big Endian, by convention whereas the Host Byte-order may be of either type.
- Functions used for IPv4 Address conversion from / to ASCII string and Binary strings expressed in the Network Byte-order have been defined in *arpa/inet.h* header file. These include: `inet_pton`, `inet_ntop`, `inet_aton`, `inet_ntoa`, `inet_addr` etc.

# Summary (Contd.)

- Under UNIX and Linux environments, typically, the only available way to create a new process is by invoking the fork function that creates a Child Process. This function returns twice per invocation – once in the Parent Process that invoked it and thereafter in the created Child process.
- The function fork can be used in association with the exec function whenever it is desired that a given process must invoke another process.
- Commonly used functions include socket, bind, listen, connect, accept, close, getsockname, getpeername, select, poll, shutdown, pselect, getsockopt, setsockopt, ioctl, fcntl, recvfrom, sendto, signal, gethostbyname, gethostbyname2, gethostbyaddr, uname, gethostname, getservbyname, getservbyport, getaddrinfo, gai\_strerror, freeaddrinfo, getnameinfo.
- If you know how to write a Client-Server pair for TCP-based application, we may easily identify the distinctly simple alterations that we may need to make in this code if we ever need to write a code for UDP-based application.

# Summary of a Quick tour to basics

- Every network protocol has its own definition of *Network Address*.
- In C, a protocol implementation provides a *struct sockaddr* as the elementary form of a *Network Address*.

- A sample definition of *struct sockaddr*

```
#include
 <sys/socket.h>

struct sockaddr {
 unsigned short
 sa_family;
 char sa_data
 [MAXSOCKADD
 RDATA]
}
```

# Summary of a Quick tour to basics

- In Linux, sockets are created by the `socket()` system call.
- This call returns a *file descriptor* for the socket that is yet to be initialized.
- Then the socket is initialized by binding it to a particular protocol and address using the `bind()` system call.

```
#include <sys/socket.h>
```

```
int socket (int domain, int type, int protocol);
```

Here,

the parameter domain specifies the PF; parameter type usually specifies either of SOCK\_STREAM or SOCK\_DGRAM; parameter protocol specifies the protocol to be used (0=> default protocol associated).

```
int bind (int sock, struct sockaddr * my_addr, int addrlen);
```

• Here,

- the parameter sock is the socket-in-question;
- parameter sockaddr is the address of protocol;
- parameter addrlen is the length of the address for the local end-point.

# Summary of a Quick tour to basics

- Next, *listen()* system call is executed for informing the system that the process is now ready to allow other processes establish a connection to this socket at the specified end-point.
- This step does not really establish a connection by itself, however!
- Now, the *accept ()* system call is made for accepting the connection requests.
- *accept ()* is a blocking call as it blocks until a process requests a connection. In case, the socket has been marked as 'non-blocking' by the *fcntl()* call, *accept()* would return an error if no process is requesting it for a connection.
- A Client attempts to connect to a Server by creating a socket, binding it to an address (optionally) and making the *connect()* call to the Server at the known address.

```
#include <sys/socket.h>
```

```
int listen (int sock, int backlog);
```

Here,

the parameter *sock* is the socket-in-question;

parameter *backlog* is the number of connection requests that may be pending on the socket before any further connection requests are explicitly refused.

```
int accept (int sock, struct sockaddr * addr, int * addrlen);
```

- The *select ()* system call can also be made for determining if any connection request is currently pending to a socket.



# Summary of the Concepts & Terms learnt so far



# Concluding remarks

- Networking support of some kind is already inside most of the operating systems we use today in variety of forms on Notebooks, Laptops, Workstations and Servers. All Smart-phones and several set-top boxes support it too.
- **Most multi-layer network switches from major vendors around the world can now support IP.**
- However, the degree of IP-readiness may vary.
- Internet Exchanges like the NIXI are already providing interconnectivity between Networks.
- **Subsequent lecture shall introduce you to the following topics:**
  - Performance
  - Quality of Service
  - **Reliability**
  - Security



Any question please?

*Thank you for your kind attention!*

For further details, you may contact at:

E-mail: [rahul@bits-pilani.ac.in](mailto:rahul@bits-pilani.ac.in) / [rahul.banerjee.cse@gmail.com](mailto:rahul.banerjee.cse@gmail.com)

**or visit:**

Home: <http://www.bits-pilani.ac.in/~rahul/>

# References

- *Larry L. Peterson & Bruce S. Davie: Computer Networks: A Systems Approach, Fourth Edition, Morgan Kaufmann / Elsevier, New Delhi, 2007. <System design approach>*
- IEEE 802 standards issued so far PLUS amendments like:
  - 802.3ap-2007: IEEE Standard for LAN/MAN — Specific Requirements Part 3: CSMA/CD Access Method and Physical Layer Specifications — Amendment 4: Ethernet Operation over Electrical Backplanes
  - 802.11-2007 IEEE Standard for LAN/MAN — Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications
  - 802.15.4a-2007 IEEE Standard for Telecommunications and Information Exchange Between Systems; PART 15.4: Wireless MAC and PHY Specifications for Low-Rate Wireless PANs (LR-WPANs) — Amendment 1: Add Alternate PHY
  - 802.1ag-2007 IEEE Standard for LAN/MAN — Virtual Bridged LANs — Amendment 5: Connectivity Fault Management

# References

- *A. S. Tanenbaum: Computer Networks, Fourth Edition, Pearson Education, New Delhi, 2003. <Conceptual Approach>*
- *Mohammed G. Gouda: Elements of Network Protocol Design, Wiley Student Edition, John Wiley & Sons (Pte.) Ltd., Singapore, 2004.*
- *Thomas G. Robertazzi: Computer Networks and Systems: Queuing Theory and Performance Evaluation, Third Edition, Springer-Verlag, New York, 2000. <Analytical approach>*
- *S. Keshav: Computer Networking: An Engineering Approach, Pearson Education, New Delhi, 1997.*
- *A. Leon Garcia and I. Widjaja: Communication Networks: Fundamental Concepts and Key Architectures, Second Edition, Tata McGraw-Hill, New Delhi, 2004.*
- *Baldwin, D.: Discovery Learning in Computer Science. In *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*, ACM, pp. 222-226, February 1996.*

# A Few More Networking Terms

- Repeaters / Repeater Hubs / Shared Hubs: where usually Physical layer / level exist with L1-protocol data unit (raw bits) regeneration and onward transmission
- Managed Hubs / Layer-2 Switching Hubs: where Physical and Data Link layers / levels exist with ability to handle and deliver Layer-2-protocol data unit (frame)
- Bridges: where Physical and Data Link layers / levels exist with L2-protocol data unit (frame) processing and forwarding
- Switches: where Physical and Data Link and / or Network (sometimes even higher) layers / levels exist with Layer-2 and / or Layer-3-protocol data unit (frame / packet) processing, switched routing / forwarding
- Routers: where Physical and Data Link and Network layers / levels exist with L3-protocol data unit (packet) processing, routing and forwarding
- Gateways: where two or more different networks meet and may require protocol / message translation capabilities

Clouds: abstraction of node connectivity in the networking context <details hidden>



# Summary

- **Intranet:** Completely private network of networks
  - Wireline
  - **Wireless**
    - Fixed
    - **Mobile**
  - Hybrid
- **The Internet:** Global public network of networks
  - Wireline
  - **Wireless**
    - Fixed
    - **Mobile**
  - Hybrid
- **Extranet:** Intranets interconnected via the Internet

# Concluding remarks

- Networking support of some kind is already inside most of the operating systems we use today in variety of forms on Notebooks, Laptops, Workstations and Servers. All Smartphones and several set-top boxes support it too.
- **Subsequent lectures shall introduce you to the following topics:**
  - Internetworks
  - Network Architectures
  - Performance
  - Quality of Service
  - Reliability
  - Security

# Next Interaction Points

- **Examples of Types of Applications benefitting from Networking**
  - hard real-time, soft real-time, non-real-time / best-effort / delay-tolerant applications / services <with examples>
  - **case-study movie**
  - Constituent networking components of a smart room setup
- **The Internet & its Evolution**
- About Internet Architecture
- **Who decides about the Internet?**
- The Internet versus the World-Wide Web
- **Protocols, Layers, Interfaces, Virtual Communication and Services**
- Select References to the literature
- **Questions and Answers / Summary**

# Introducing Some Terms Related to Networks

- Channel <application-level logical / virtual communication path>
- Services: Functionalities provided by a layer / protocol / entity
- Interfaces: Peer-to-Peer / Layer-to-Layer / entity-to-entity
- Service Access Points: defined addresses / ports through which data / parameters are passed
- Tunneling <Encapsulation & Decapsulation>