

# Computer Networks

By Ihsan Ul Haq

Lec-3

---

# Text Book

- **Computer Networks: A Systems Approach, 3rd Edition (The Morgan Kaufmann Series in Networking)** by Larry L. Peterson and Bruce S. Davie
- **Unix Network Programming, Vol. 1: The Sockets Networking API, Third Edition** -by W. Richard Stevens

## Wrap-up

- A network can be constructed from *nesting* of networks
- An *address* is required for each node that is reachable on the network
- Address is used to *route* messages toward appropriate destination

# Introduction

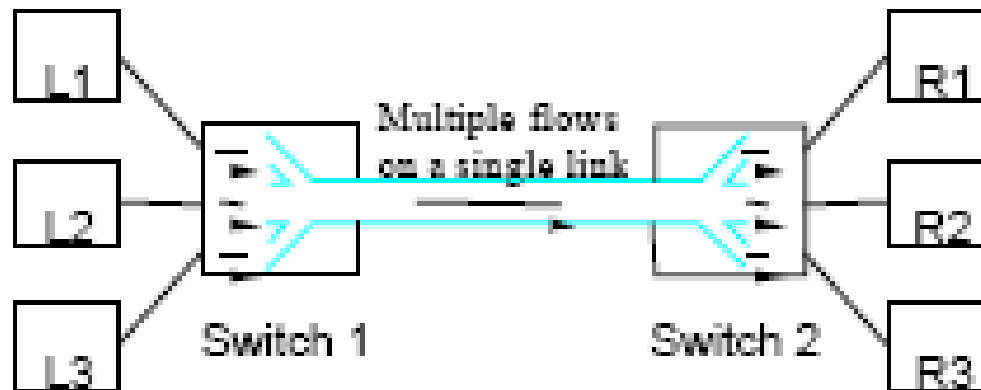
- Outline
  - Statistical Multiplexing
  - Inter-Process Communication
  - Performance Metrics
  - Network Architecture

## What next ?

- Hosts know how to reach other hosts on the network
- How should a node use the network for its communication ?
- All pairs of hosts should have the ability to exchange messages: **cost-effective** resource sharing for efficiency

# Multiplexing

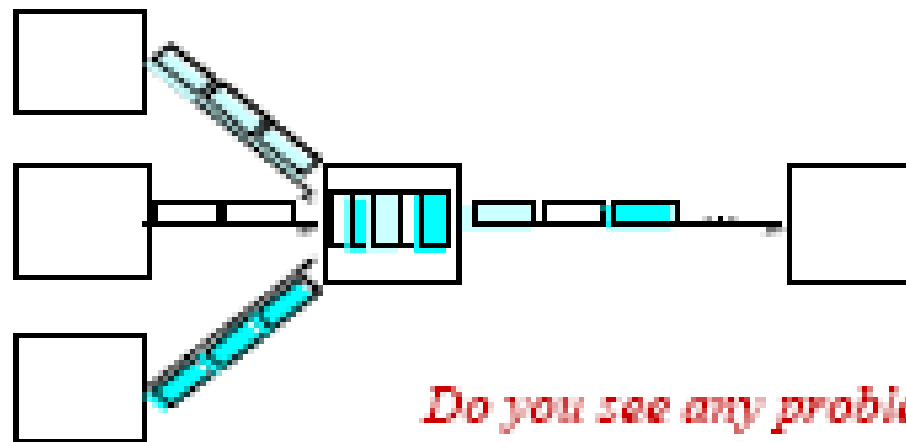
- Physical links/switches must be shared among users
  - (synchronous) Time-Division Multiplexing (TDM)
  - Frequency-Division Multiplexing (FDM)



*Do you see any problem with TDM / FDM ?*

# Statistical Multiplexing

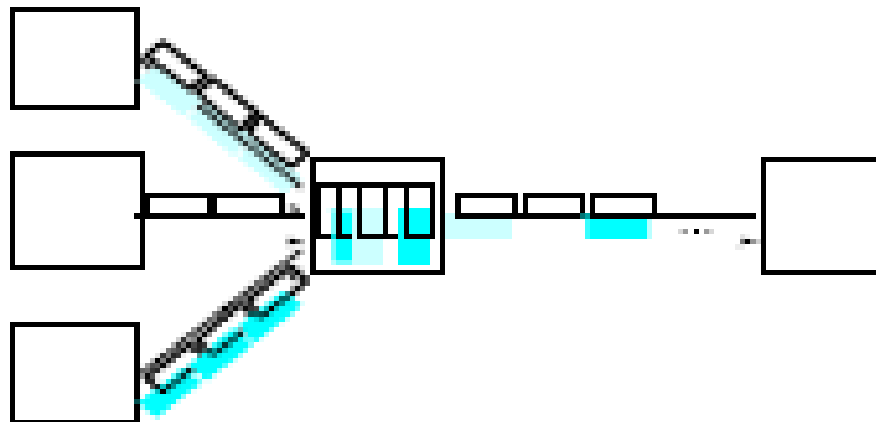
- On-demand time-division, possibly synchronous (ATM)
- Schedule link on a per-packet basis
- Buffer packets in switches that are *contending* for the link
- Packets from different sources interleaved on link



*Do you see any problem ?*

# Statistical Multiplexing

- An application needs to break-up its message in packets, and re-assemble at the receiver
- Fair allocation of link capacity: FIFO, round-robin or QoS
- If *congestion* occurs at a switch - buffer may overflow, packets may be lost

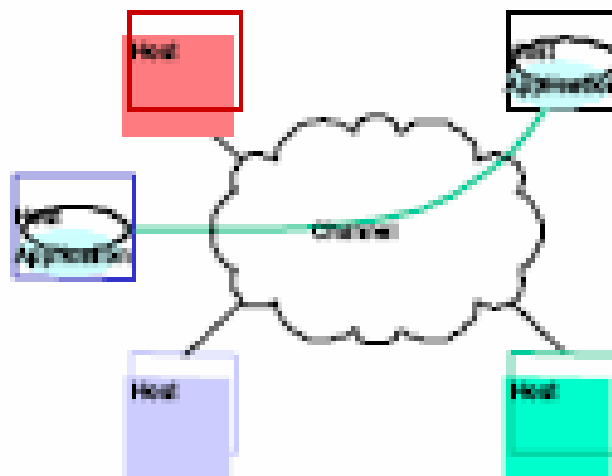


# Enough ... ?

- A network is delivering packets among a collection of computers
- How application processes communicate in a **meaningful** way ?
- Hide network complexity by implementing the common services once

# Inter-Process Communication

- Turn host-to-host connectivity into process-to-process communication, making the communication meaningful.
- Fill gap between what applications expect and what the underlying technology provides.



Abstraction for  
application-level  
communication

# IPC Abstractions

- Semantics and interface depend on applications
- Request/Reply
  - distributed file systems
    - file servers (FTP)
  - digital libraries / HTTP
    - information retrieval
- reliable ?
- prioritized ?
- delay/bandwidth guarantees ?
- Message stream
  - video on-demand
  - video conferencing
    - delay sensitive
    - two-way frame flow
    - 1/4 NTSC = 352x240 pixels
    - $(352 \times 240 \times 24)/8=247.5\text{KB}$
    - 30 fps = 7500KBps = 60Mbps
    - 10fps + compression < 10Mbps

# Abstract Channel Functionality

- What functionality a channel provides ?
  - Smallest set of abstract channel types adequate for largest number of applications
- Where the functionality is implemented ?
  - Network as a simple *bit-pipe* with all high-level communication semantics at the hosts
  - More intelligent switches allowing hosts to be “dumb” devices (telephone network)

# What Goes Wrong in the Network?

## Reliability at stake

- Bit-level errors (electrical interference)
- Packet-level errors (congestion)
  - distinction between lost and late packet
- Link and node failures
  - distinction between broken and flaky link
  - distinction between failed and slow node

# What Goes Undesirable in the Network?

Required performance at stake

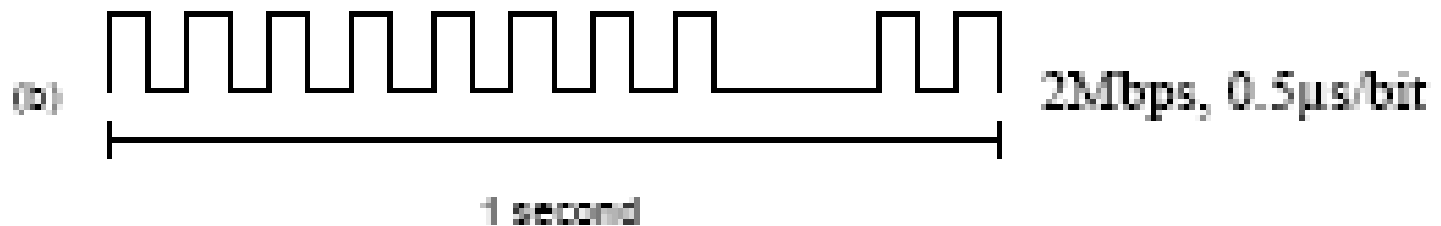
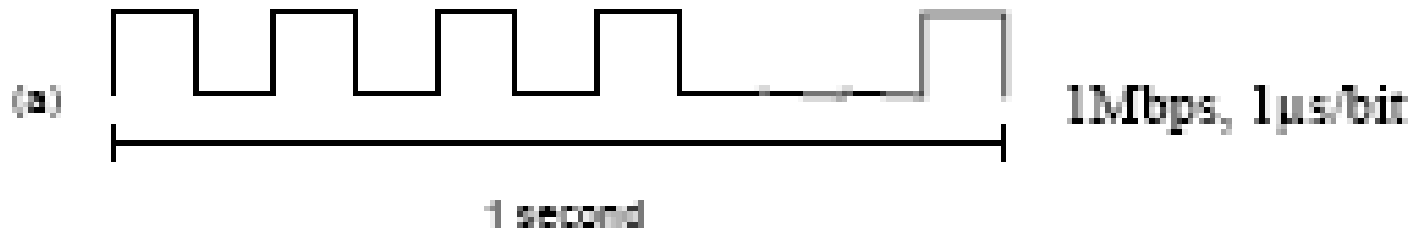
- Messages are delayed
- Messages are delivered out-of-order
- Third parties eavesdrop
  
- The challenge is to fill the gap between application expectations and hardware capabilities

# Performance Metrics

- • ...and to do so while delivering “good” performance
- Bandwidth (throughput)
  - data transmitted per unit time, e.g. 10 Mbps
  - link bandwidth versus **end-to-end** bandwidth
- notation
  - KB =  $2^{10}$  bytes
  - Kbps =  $10^3$  bits per second

# Performance Metrics

Bandwidth related to "bit width"



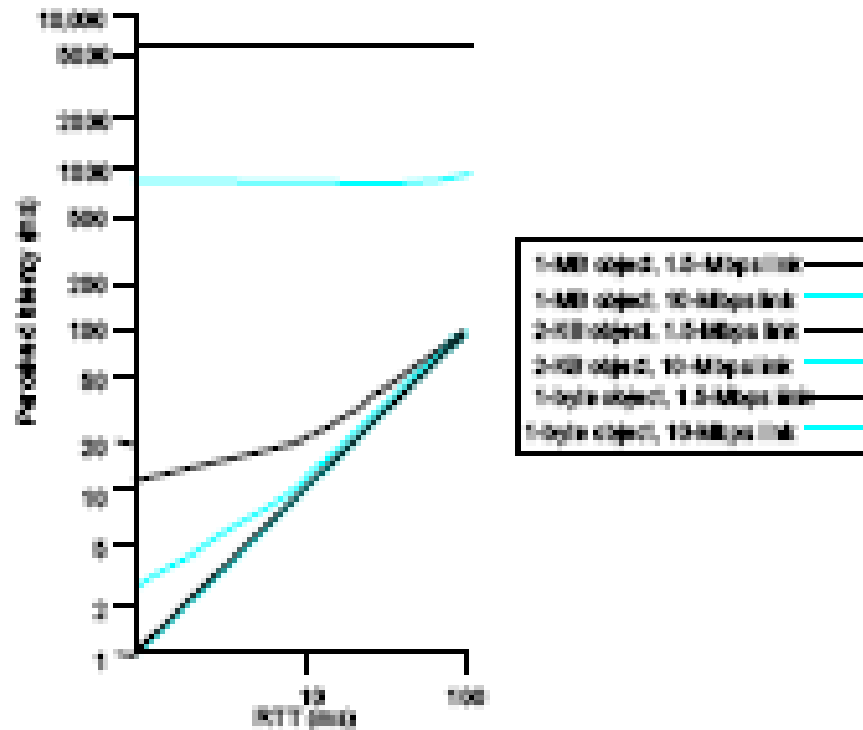
# Performance Metrics

- • Latency / delay
  - time to send message from point A to point B
  - one-way versus round-trip time (**RTT**)
  - components
    - Latency = Propagation + Transmit + Queue
    - Propagation = Distance / c
    - Transmit = Size / Bandwidth
- Note:
  - No queuing delay in direct (point-to-point) link
  - Bandwidth irrelevant if size = 1 bit
  - Process-to-process latency includes software processing overhead (dominates over short distances)

# Bandwidth versus Latency

- Relative importance, depends on application
- 1-byte character:
  - Choice of 1ms vs 100ms dominates 1Mbps vs 100Mbps
- 25MB file:
  - Choice of 1Mbps vs 100Mbps dominates 1ms vs 100ms
- Large data (file transfer) is bandwidth critical
- Small data (HTTP) is latency critical

# Bandwidth versus Latency



# Delay x Bandwidth Product

- Amount of data “in flight” or “in the pipe”
- Example: 100ms RTT  $\times$  45Mbps BW = 560KB
- This much data must be buffered before sender responds to slowdown request



# Infinite Bandwidth

- Latency (RTT) dominates instead of throughput
  - $\text{Throughput} = \text{TransferSize} / \text{TransferTime}$
  - $\text{TransferTime} = \text{RTT} + 1/\text{Bandwidth} \times \text{TransferSize}$
- 1 MB file over a 1 Mbps network takes around 8 sec
  - With RTT of 100ms, it corresponds to 80 RTTs
  - Effective throughput is  $1\text{MB}/8.1\text{s} = 0.987\text{Mbps}$
- 1 MB file over a 1 Gbps network takes 100ms + 8ms
  - Effective throughput is  $1\text{MB}/108\text{ms} = 74.1\text{ Mbps}$
- 1-MB *file* to 1-Gbps link appears like a 1-KB *packet* to 1-Mbps link

# Bandwidth Requirements

- Request/reply type applications may agree for as much bandwidth is available
- Message stream type applications often require a certain bandwidth: not more not less
- Average required bandwidth not always helps to design a network
  - Upper bound on a burst (peak-rate transfer)
- • Queuing delay introduces variation in latency (jitter)

# Wrap-up

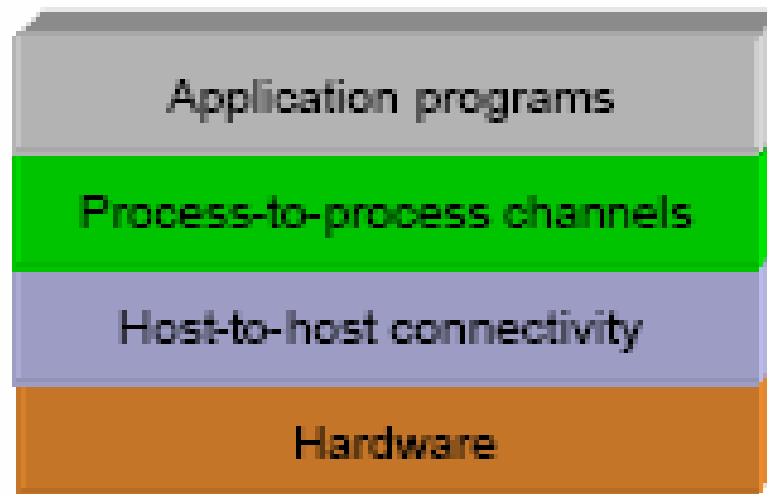
- Established a comprehensive set of requirements for network design
- Networks evolve to accommodate changes in underlying technologies and user demands
- However, hardware and user expectations are **moving targets** ...

# Network Architecture

- The challenge is to **fill the gap** between hardware capabilities and application expectations, and to do so while delivering “good” performance
- Designers cope with this complex task by developing a ***network architecture*** as a guideline
  - Layering, protocols, standards

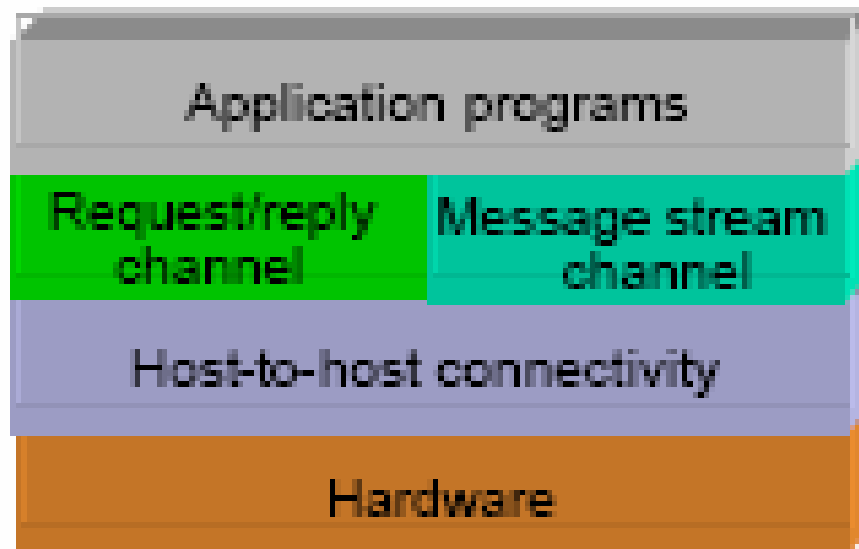
# Layering

- Use abstractions to hide complexity
- Abstractions naturally lead to layering
- Each layer provides some functionality



# Layering

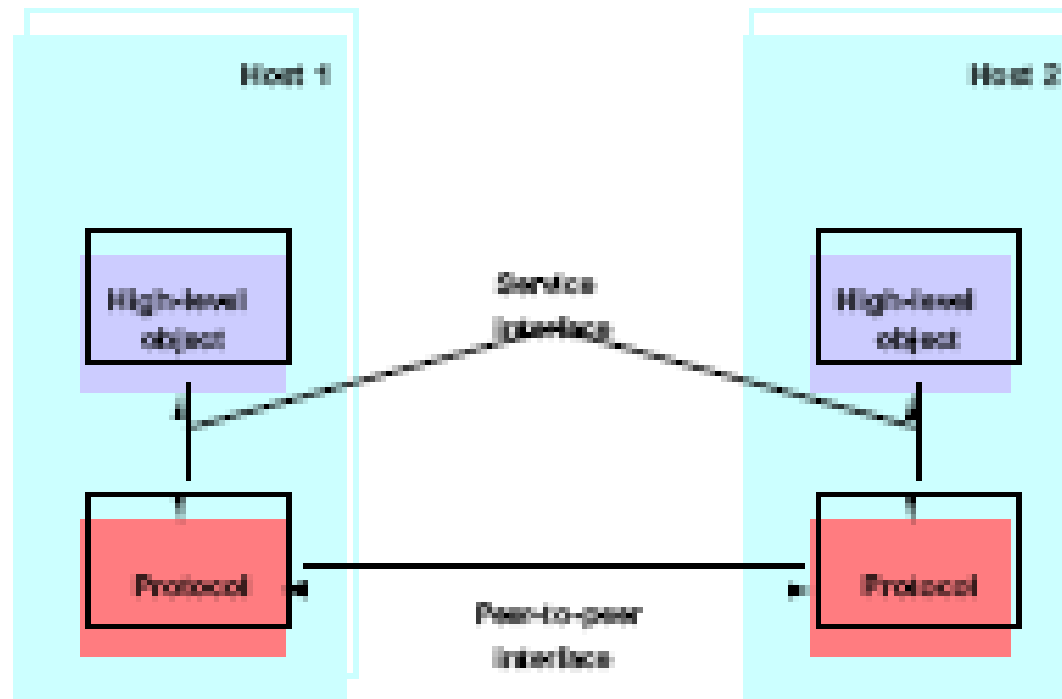
- Alternative abstractions at each layer
- Manageable network components
- Modify layers independently



# Protocols

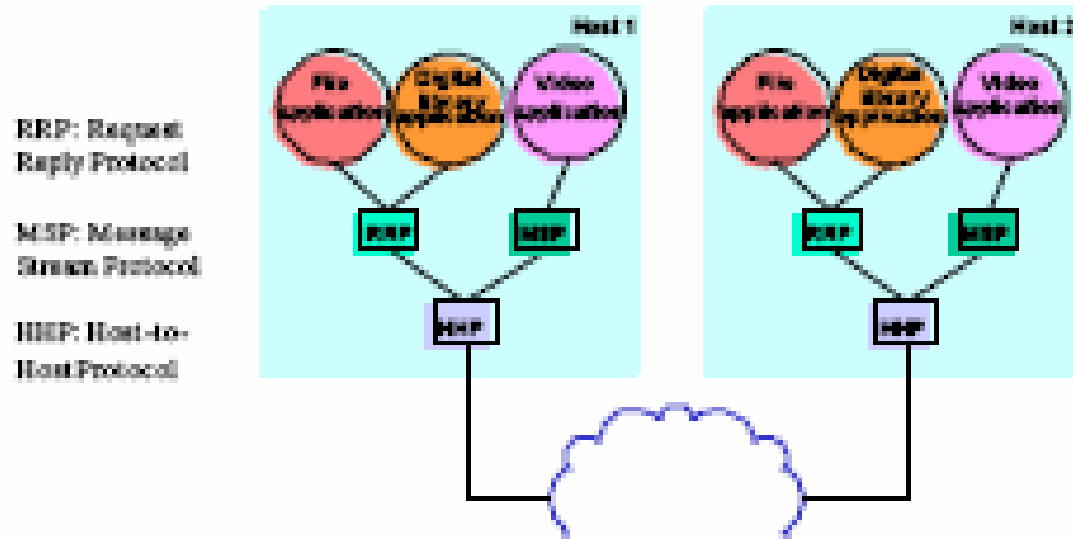
- Building blocks of a network architecture
- Each protocol object has two different interfaces
  - *service interface*: operations on this protocol
  - *peer-to-peer interface*: messages exchanged with peer
- Term “protocol” is overloaded
  - specification of peer-to-peer interface
  - module that implements this interface
  - peer modules are interoperable if both accurately follow the specifications

# Protocol Interfaces



# Protocol Graph – Network Architecture

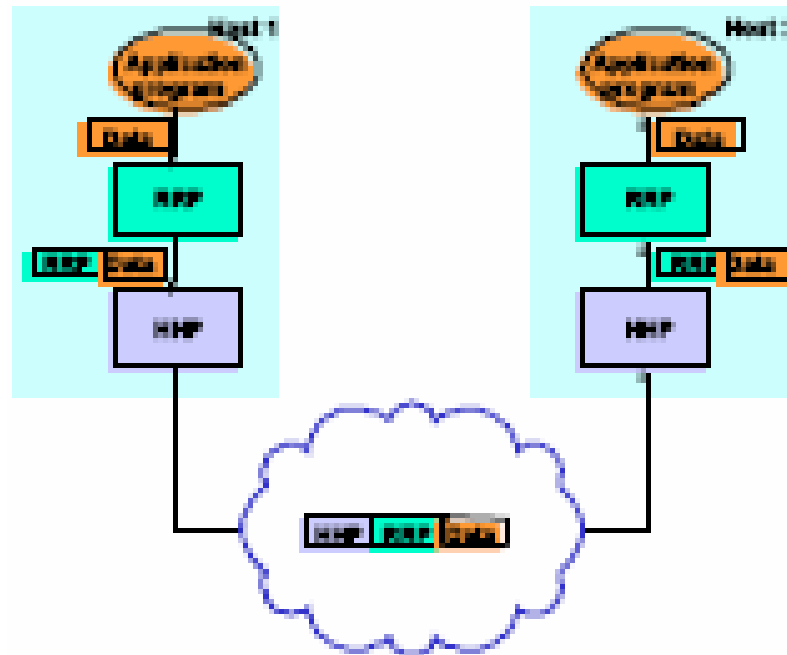
- Collection of protocols and their dependencies
  - most peer-to-peer communication is indirect
  - peer-to-peer is direct only at hardware level



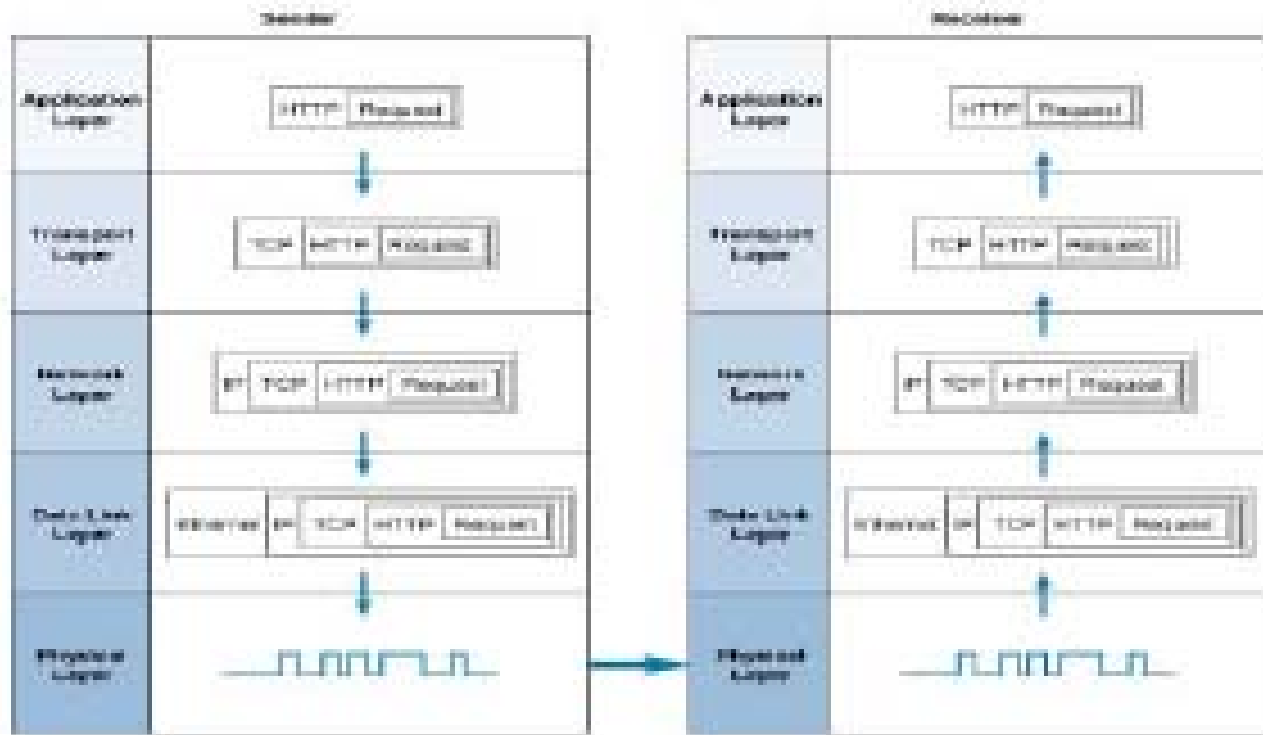
# Protocol Machinery

- Multiplexing and Demultiplexing (demux key)
- Encapsulation (header/body) in peer-to-peer interfaces
  - indirect communication (except at hardware level)
  - each protocol adds a header
  - part of header includes demultiplexing field (e.g., pass up to request/reply or to message stream?)

# Encapsulation



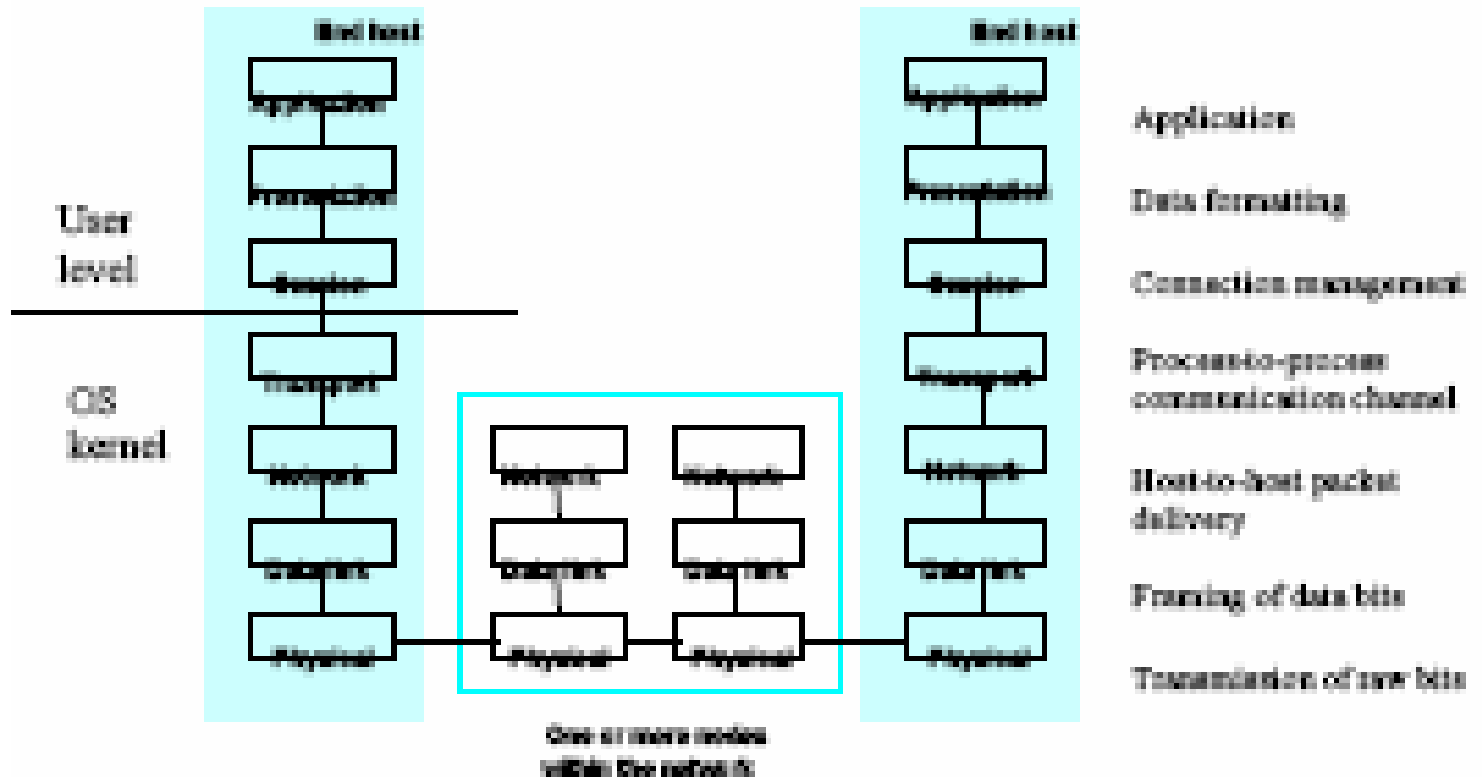
# Message Transmission Using Layers



# Standard Architectures

- • Open System Interconnect (OSI) Architecture
  - International Standards Organization (ISO)
  - International Telecommunications Union (ITU), formerly CCITT
  - “X dot” series: X.25, X.400, X.500
  - Primarily a reference model

# OSI Architecture

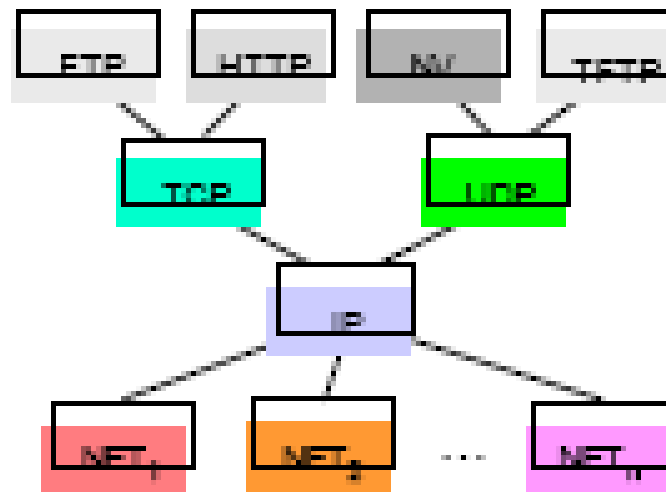


# Internet Architecture

- TCP/IP Architecture
  - Developed with ARPANET and NSFNET
  - Internet Engineering Task Force (IETF)
    - Culture: implement, then standardize
    - OSI culture: standardize, then implement
  - Became popular with release of Berkeley Software Distribution (BSD) Unix; i.e. free software
  - Standard suggestions traditionally debated publically through “Request For Comments” (RFC’s)

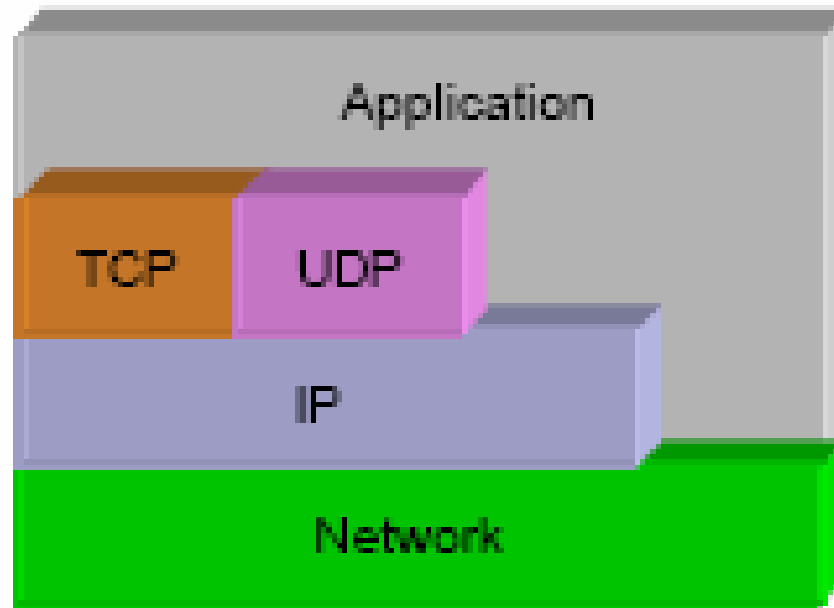
# Internet Architecture

- Implementation and design done together
- Hourglass Design (bottleneck is IP)
- Application vs Application Protocol (FTP, HTTP)



# Internet Architecture

- Layering is not very strict



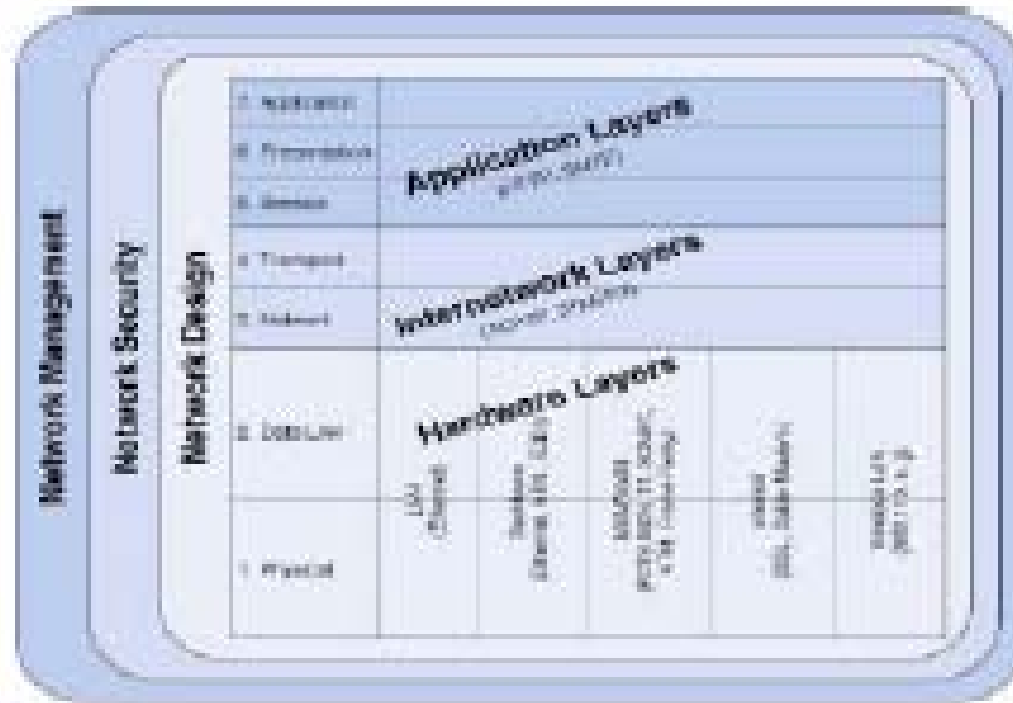
# Network Models

Groups of Layers	OSI Model	Early Internet Model
<i>Application Layers</i>	7. Application Layer	4. Application Layer
	6. Presentation Layer	
	5. Session Layer	
<i>Internetwork Layers</i>	4. Transport Layer	3. Transport Layer
	3. Network Model	2. Network Model
<i>Hardware Layers</i>	2. Data Link Layer	1. Hardware Layer
	1. Physical Layer	

# How Layers Fit Together in Practice

<b>Application Layers</b>	7. Application	Web (HTTP)	E-mail (SMTP)	Others (FTP, IM)
	6. Presentation			
	5. Session			
<b>Internetwork Layers</b>	4. Transport	Internet TCP		Novell SPPX
	3. Network	Internet IP		Novell IPX
<b>Hardware Layers</b>	2. Data Link	LAN (Ethernet)	Backbone (Ethernet, ATM, FDDI)	MAN/WAN (FDDI, SDH, T1, SONET, ATM, Frame Relay)
	1. Physical		Internet (DSL, Cable Modem)	Wireless LAN (802.11b, a, g)

# Networking in the Internet Age



# Protocol Acronyms

- (T)FTP – (Trivial) File Transfer Protocol
- HTTP – Hyper Text Transport Protocol
- NV – Network Video
- SMTP – Simple Mail Transfer Protocol
- NTP – Network Time Protocol
- TCP – Transmission Control Protocol
- UDP – User Datagram Protocol
- IP – Internet Protocol
- FDDI – Fiber Distributed Data Interface
- ATM – Asynchronous Transfer Mode

# Elements of a Protocol Implementation





- Outline

- Service Interface
- Process Model
- Common Subroutines
- Example Protocol

# Network Software

- Major factors for runaway success of the Internet:
  - most functionalities provided by software running on general-purpose computers
- new services can be added readily with just a small matter of programming
- Understanding how to implement network software is essential to understand computer networks

# Network Application Programming Interface (API)

- Interface that the OS provides to its networking subsystem
  - most network protocols are implemented in software
  - all systems implement network protocols as part of the OS
  - each OS is free to define its own network API
  - applications can be ported from one OS to another if APIs are similar
    - ● \*IF\* application program does not interact with other parts of the OS other than the network (file system, fork processes, display ...)

# Protocols and API

- Protocols provide a certain set of **services**
- API provides a **syntax** by which those services can be invoked
- Implementation is responsible for mapping API syntax onto protocol services

# Socket API

- Use sockets as “abstract endpoints” of communication
- Issues
  - Creating & identifying sockets
  - Sending & receiving data
- Mechanisms
  - UNIX system calls and library routines.



# Socket API

- Creating a socket
- `int socket(int domain, int type, int protocol)`
  - domain (family) = AF\_INET, PF\_UNIX, AF\_OSI
  - type = SOCK\_STREAM, SOCK\_DGRAM
  - protocol = TCP, UDP, UNSPEC
- return value is a ***handle*** for the newly created socket

# Sockets (cont)

- Passive Open (on server)

```
int bind(int socket, struct sockaddr *addr, int  
        addr_len)
```

```
int listen(int socket, int backlog)
```

```
int accept(int socket, struct sockaddr *addr, int  
          addr_len)
```

- Active Open (on client)

```
int connect(int socket, struct sockaddr *addr,  
           int addr_len)
```

## Sockets (cont)

- Sending Messages

int **send**(int **socket**, char \***msg**, int **mlen**, int **flags**)

- Receiving Messages

int **recv**(int **socket**, char \***buf**, int **blen**, int **flags**)