

# Computer Networks

## EE-5281

By Ihsan Ul Haq  
Lec-4

---



- 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**)

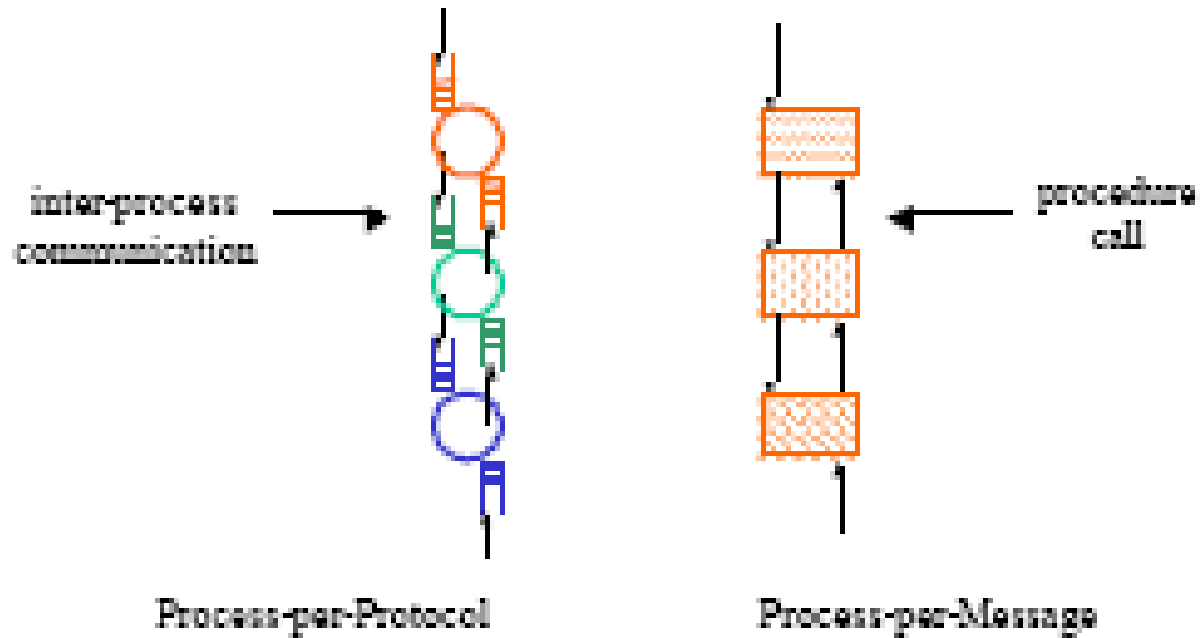
# Protocol-to-Protocol Interface

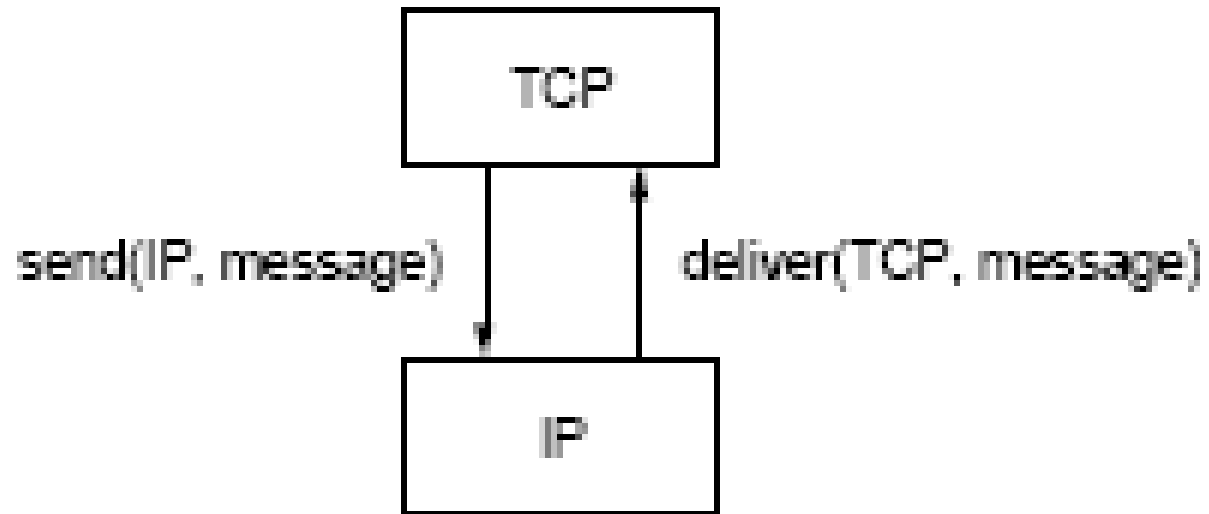
- A protocol interacts with a lower level protocol like an application interacts with underlying network
- Why not using available network APIs for PPI ?
  - Inefficiencies built into the socket interface
    - application programmer tolerate them to simplify their task
      - inefficiency at one level
  - protocol implementers do not tolerate them
    - inefficiencies at several layers of protocols

# Protocol-to-Protocol Interface Issues

- Configure multiple layers
  - static versus extensible
- Process Model
  - avoid context switches
- Buffer Model
  - avoid data copies

# Process Model

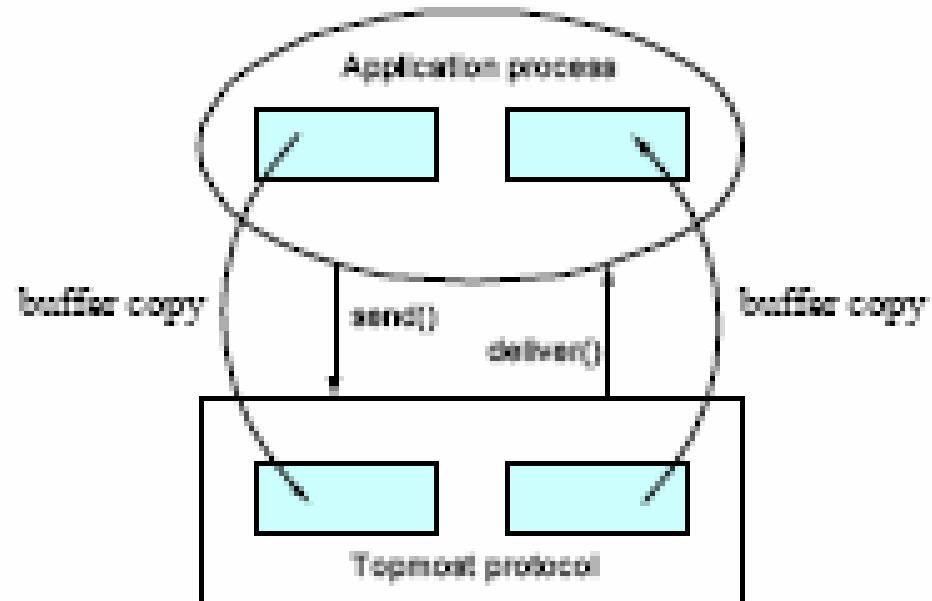




```
int send(Protocol lp, Msg *message)
```

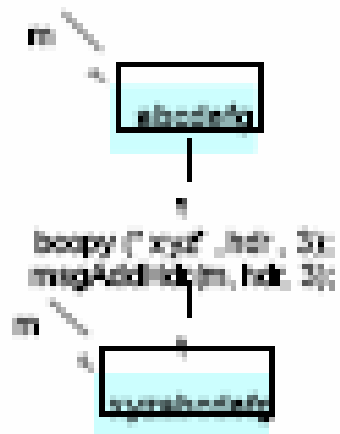
```
int deliver(Protocol hp, Msg *message)
```

# Buffer Model

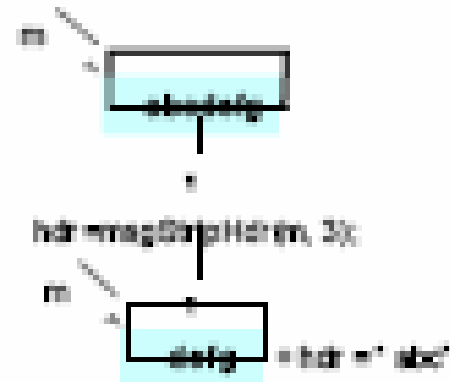


# Message Library

- Add header



- Strip header

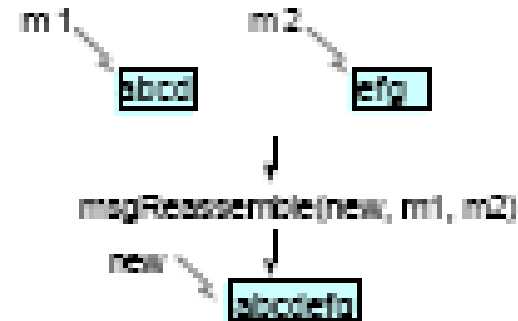


# Message Library (cont)

- Fragment message



- Reassemble messages

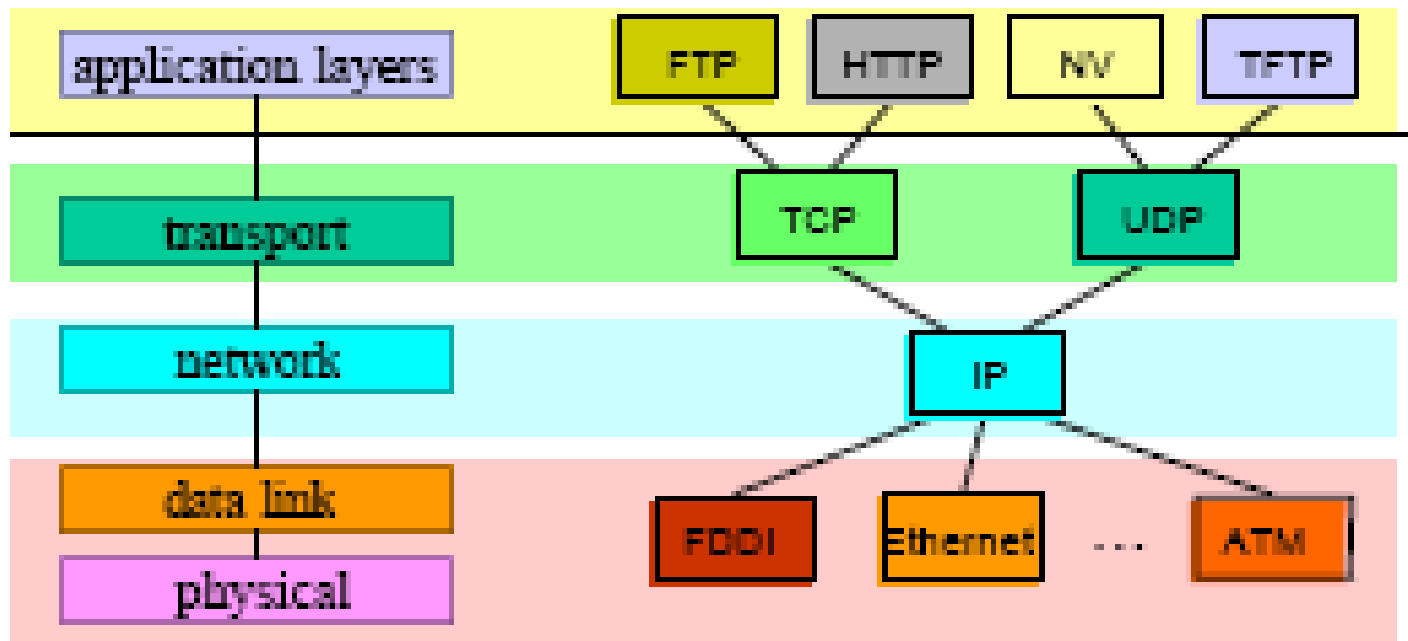


# Network Programming

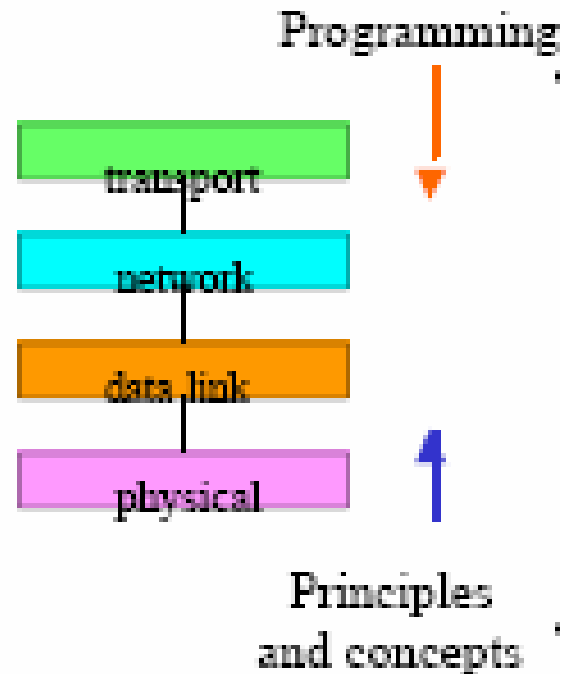
- Things to learn
  - Internet protocols (IP, TCP, UDP, ...)
  - Sockets API (Application Programming Interface)
- Why IP and sockets
  - IP (Internet Protocol) is standard
    - allows a common name space across most of Internet
    - reduces number of translations, which incur overhead
  - Sockets: reasonably simple and elegant Unix interface (most servers run Unix)

## OSI Model

## Internet Protocols



- learn to use Internet for communication (with focus on implementation of networking concepts)
- learn to build network from ground up



# Socket Programming

- • Reading: Stevens 2nd edition, Chapter 1-6
- • Sockets API: a transport layer service interface
  - introduced in 1981 by BSD 4.1
  - implemented as library and/or **system calls**
  - similar interfaces to TCP and UDP
  - can also serve as interface to IP (for super- user); known as “raw sockets”
  - Linux also provides interface to MAC layer (for superuser); known as “data- link sockets”

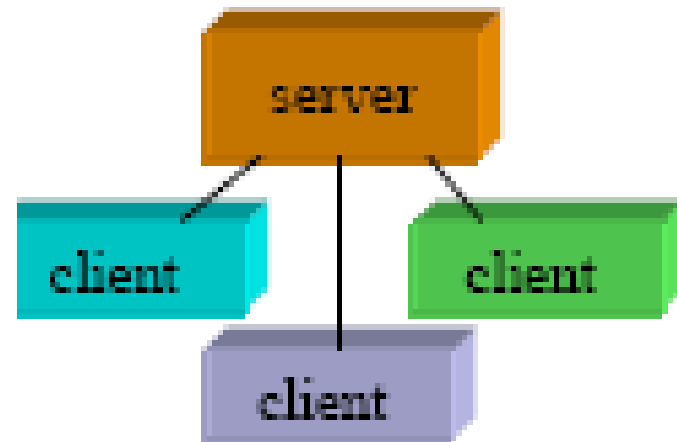
# Outline

---

- Client-server model
- TCP connections
- UDP services
- Addresses and data
- Sockets API
- Example of use

# Client-Server Model

- Asymmetric relationship
- Server/daemon
  - well-known name
  - waits for contact
  - process requests, sends replies
- Client
  - initiates contact
  - waits for response



# Client-Server Model

- • Bidirectional communication channel
- • Service models
  - **sequential**: server processes only one client's requests at a time
  - **concurrent**: server processes multiple clients' requests simultaneously
  - **hybrid**: server maintains multiple connections, but processes requests sequentially
- Server and client categories not disjoint
  - server can be client of another server
  - server as client of its own client (peer-to-peer architecture)

# TCP Connections

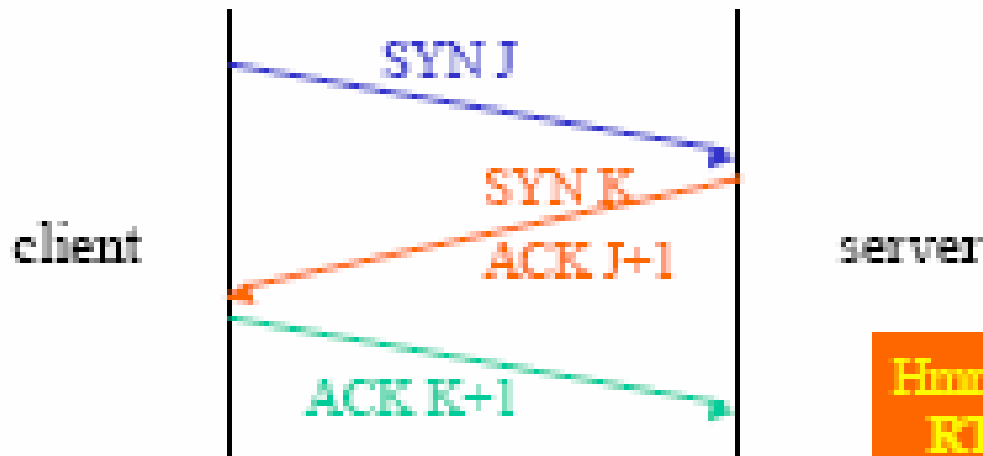
- Transmission Control Protocol, at OSI transport layer
- Recall: each protocol provides service interface
- Aspects of TCP service
  - transfers a stream of bytes (interpreted by application)
  - connection-oriented
    - set up connection before communicating
    - tear down connection when done
  - in-order delivery of data: if A sends M1 followed by M2 to B, B never receives M2 before M1

# Aspects of TCP Service

- Reliable
  - data delivered at most once
  - exactly once if no catastrophic failures
- Flow control
  - prevents senders from wasting bandwidth
  - reduces global congestion problems
- Full-duplex: send or receive data at any time
- 16-bit **port** space allows multiple connections on a single host

# TCP Connections

- TCP connection setup via 3-way handshake
  - J and K are sequence numbers for messages



Hummm ...  
RTT is  
important!

# TCP Connections

- TCP connection teardown (4 steps) (either client or server can initiate connection teardown)

