

# CS348: Computer Networks

---



## Application Layer

Dr. Manas Khatua  
Assistant Professor  
Dept. of CSE, IIT Guwahati  
E-mail: [manaskhatua@iitg.ac.in](mailto:manaskhatua@iitg.ac.in)

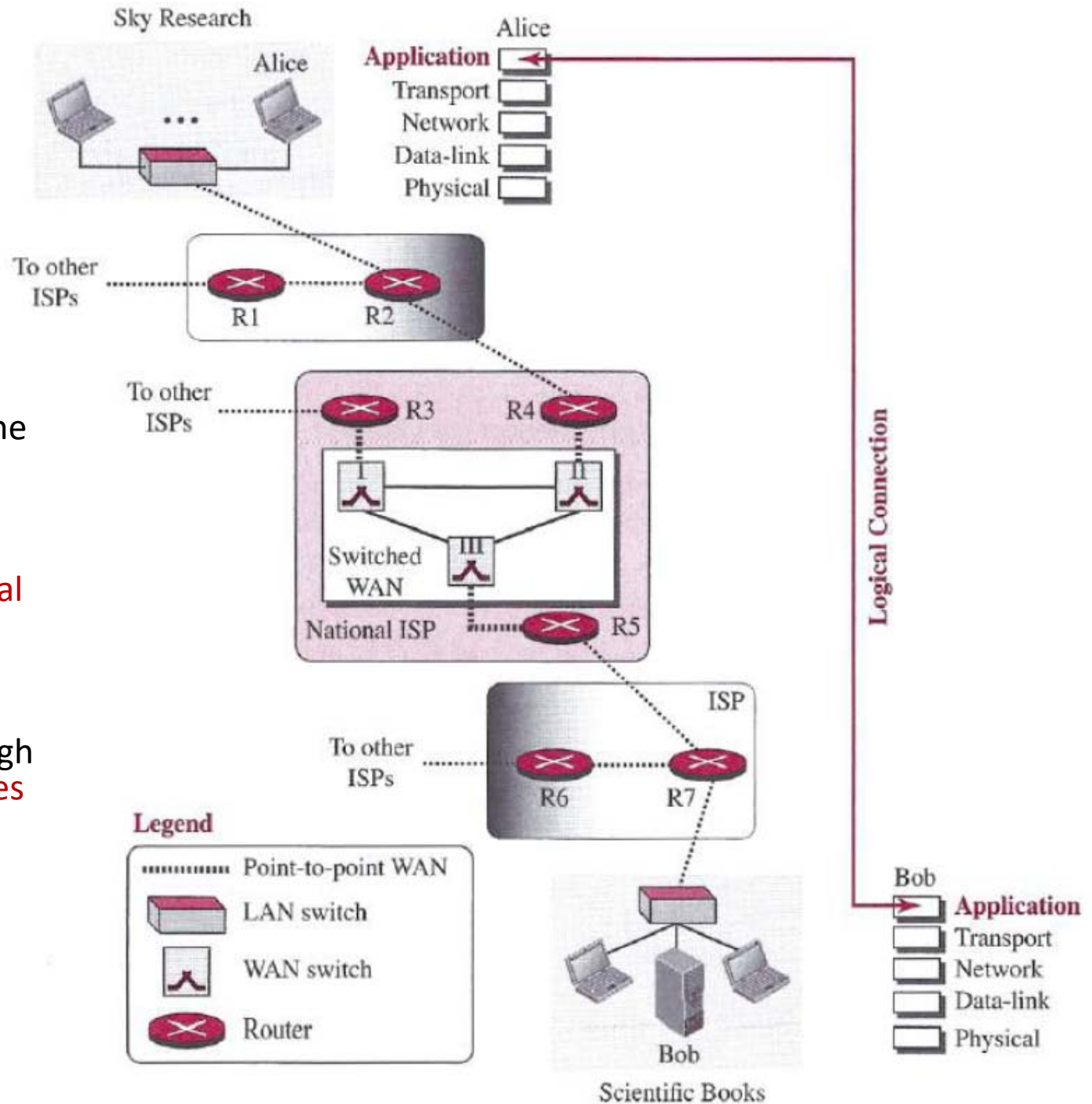
---

# Basic

Application layer  
**provides services** to the  
users

Communication is  
provided using a **logical  
connection**.

The actual  
communication through  
several **physical devices**  
(Alice, R2, R4,...) and  
**channels**.



# Providing Services

---

- Application layer **provides services** to the user, and **takes services** from Transport layer.
- The application layer, however, is somewhat **different** from other layers.
- The **protocols can be removed** from this layer easily
  - as they only **receives services** from Transport layer but **does not provide service** to that layer.
- The protocols used in the **first four layers** of the TCP/IP suite **need to be standardized and documented**.
  - So, normally comes with Operating Systems (OS).
- Several application-layer protocols that have been standardized and documented by the Internet authority.
  - e.g. DHCP, SMTP, FTP, HTTP, TELNET

# Popular Network Applications



- The **network applications** have been the driving force behind the Internet's success
- **1970s and 1980s**: classic **text-based applications** - text email, remote access to computers, file transfers, newsgroups.
- **mid-1990s**: World Wide Web, encompassing Web surfing, search, and **e-commerce**
- **end-1990s**: **instant messaging** and **P2P file sharing**
- **Since 2000**: voice and video applications
  - voice-over-IP (**VoIP**)
  - video conferencing over IP such as **Skype**;
  - user-generated video distribution such as **YouTube**;
  - movies on demand such as **Netflix**.
- **Recently**: multi-player **online games** (e.g. World of Warcraft), social networking applications (e.g. **Facebook** and **Twitter**)

# N/W Application vs Application Layer Protocol



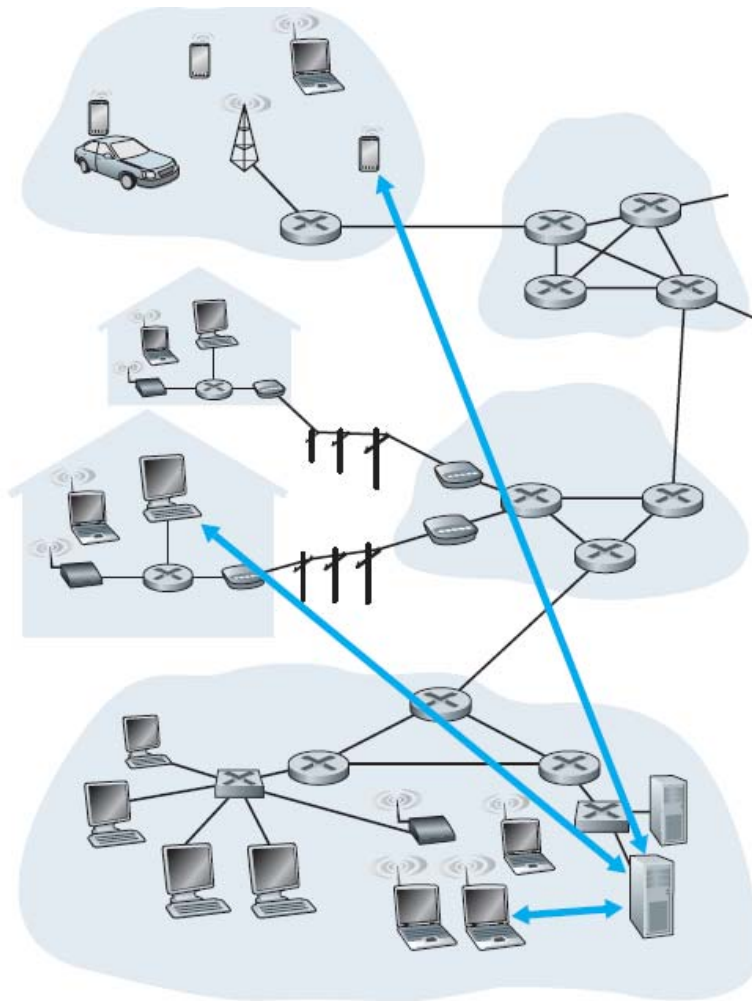
- An **application-layer protocol** is only one piece of a **network application**.
- **Example:**
  - **Web application** consists of many components, including
    - a standard for document formats (e.g. HTML),
    - Web browser (e.g. Firefox, Chrome),
    - Web server (e.g. Apache, Microsoft Server), and
    - **application-layer protocol** (e.g. HTTP) which defines the format and sequence of messages exchanged between browser and Web server.
  - Internet **e-mail application** also has many components, including
    - mail servers that house user mailboxes;
    - mail clients (such as Microsoft Outlook, Gmail) that allow users to read and create messages;
    - a standard for defining the structure of an e-mail message; and
    - **application-layer protocol** (e.g. SMTP) that define how messages are passed between servers, how messages are passed between servers and mail clients, and how the contents of message headers are to be interpreted.

# N/W Application Architecture

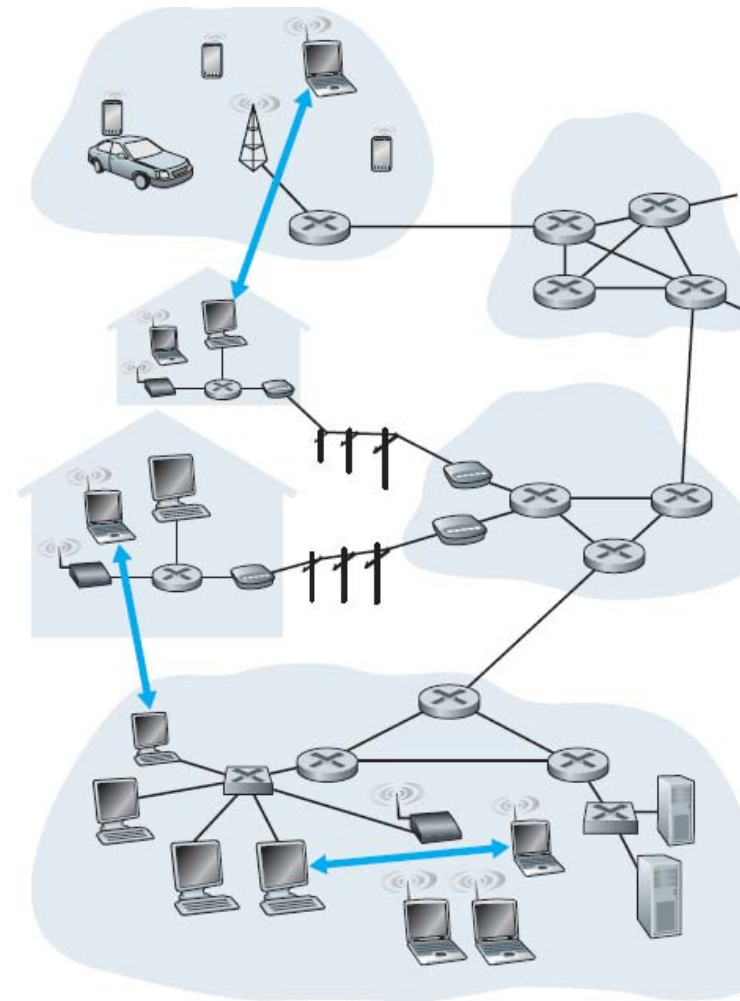
---

- Application protocols are designed by
  - the **application developer**
- It dictates how the application is structured over the various end systems
  - What the **relationship** should be between these two programs?
- We use two predominant **architectural paradigms**
  - **client-server**
    - there is an **always-on** host, called the **server**, which services requests from many other hosts, called **clients**.
    - Classic **example**: Web, FTP, Telnet, SSH, and e-mail.
  - **peer-to-peer** (P2P)
    - the application exploits direct communication between pairs of intermittently connected hosts, called **peers**
    - Classic **example**: Internet Telephony (e.g., Skype), file sharing (e.g., BitTorrent), and IPTV

# Client-Server & P2P architectures



a. Client-server architecture



b. Peer-to-peer architecture

**Figure 2.2** ♦ (a) Client-server architecture; (b) P2P architecture

# Client-Server Paradigm

---

- the **service provider** is an application program, called the **server process**;
- Server process **runs continuously**, waiting for another application program, called the **client process**, to make a connection through the Internet and ask for service.
- The **server process must be running all the time**;
- the **client process starts when the client needs** to receive service.
- Several traditional services are still using this paradigm, e.g., WWW, FTP, SSH, E-mail, and so on.
- **Problems:**
  - the server should be a **powerful** computer
  - there should be a service provider willing to accept the **cost** and create a powerful server for a specific service
  - Often in a client-server application, a **single-server host is incapable** of keeping up with all the requests from clients.



# Peer-to-Peer Paradigm

---

- There is **no need** for a **server process** to be running all the time and waiting for the client processes to connect.
- The responsibility is **shared between peers**.
  - e.g.: Internet telephony (Skype), BitTorrent, IPTV
- **Advantages:**
  - easily **scalable** and **cost-effective** by eliminating the need for expensive servers to be running and maintained all the time
- **Challenges:**
  - **ISP Friendly**: Most residential ISPs follow “**asymmetrical**” bandwidth usage (different upstream and downstream speed) controlled by server application. But, P2P does not have control server!
  - **Security**: more difficult to create **secured communication** between distributed services
  - **Incentives**: success also depends on convincing users to volunteer bandwidth, storage, and computation resources to the applications
- Few instant messaging application **use both** client-server and P2P

# Processes Communicating

---

- A **process** can be thought of **as a program** that is running within an end system.
- When processes are **running** on the **same end system**,
  - they can communicate with each other with *inter-process communication*
- When processes are **running** on two **different end systems**
  - communicate with each other by *exchanging messages* across the network.

## Client-Server Programming

- Runs two processes: a **client** and a **server**
- A client is a running program that **initializes the communication** by sending a request;
- A server is another application program that **waits for a request** from a client.
- A **client program** is started and stopped by the user whenever it requires.
- A service provider continuously runs the **server program**

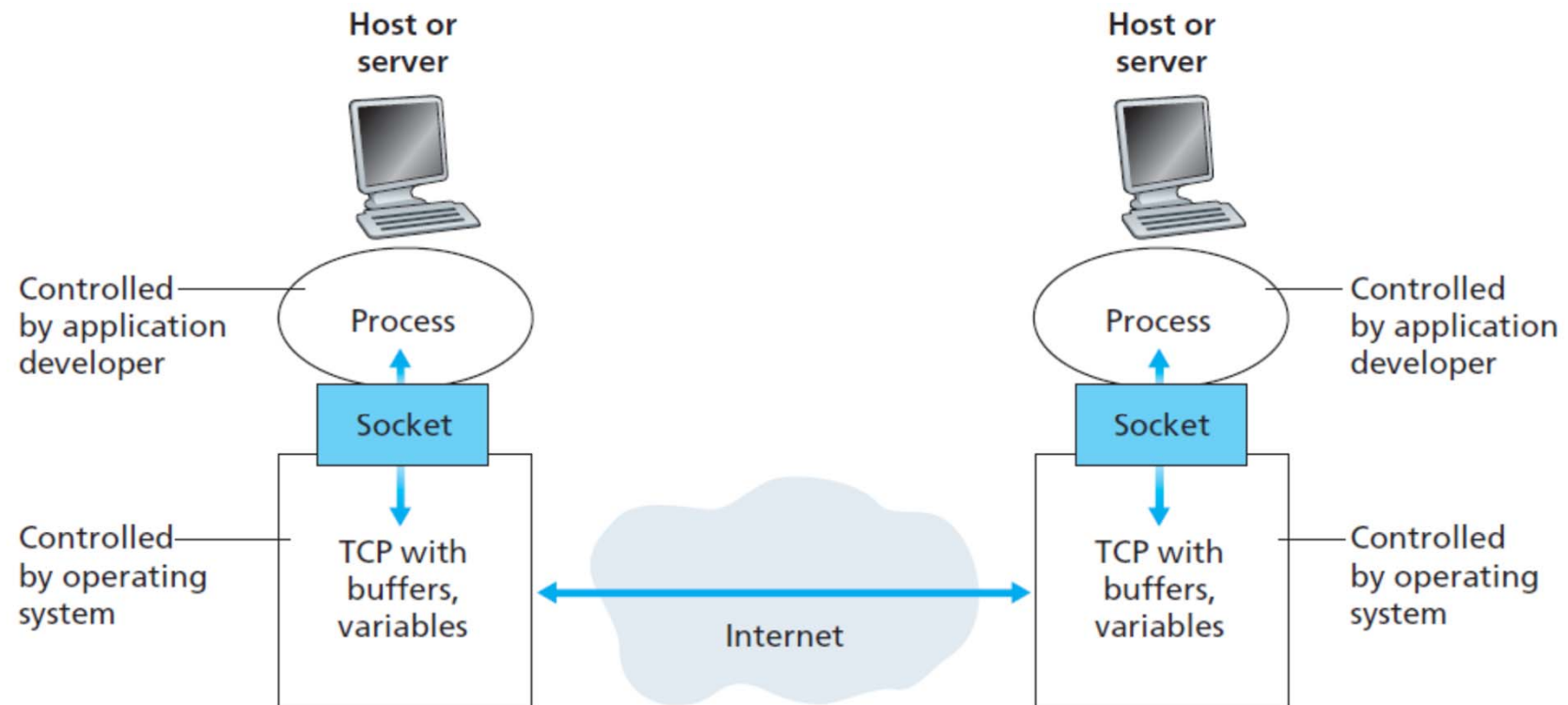
# Interface Between a Process & Network



- most applications consist of pairs of communicating processes
- Any message sent from one process to another must **go through the underlying network**.
- Application Programming Interface (API)
  - Set of instructions **to talk with** the lowest four layers (in OS)
  - instructs to **open** a connection, **send** and **receive** data, **close** the connection
  - Set of instruction of this kind is **API**
- Several APIs have been designed for communication. Three most **common APIs**
  - **Socket interface**
  - Transport Layer Interface (TLI)
  - STREAM

# Socket

- A process sends messages into, and receives messages from, the network **through a software interface** called a **socket**.



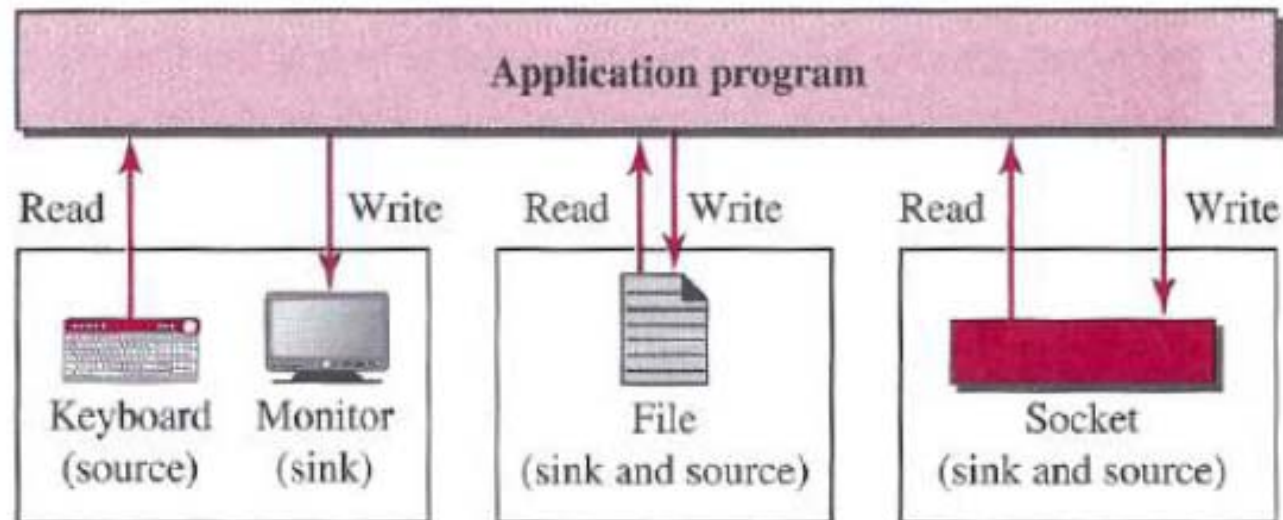
**Figure 2.3** ♦ Application processes, sockets, and underlying transport protocol

# Socket Interface

---

- **Socket interface** started in the early 1980s at UC Berkeley as part of a UNIX environment.
- The socket interface is **a set of instructions** that provide communication between the **application layer** and the **OS**.
- The idea of **socket** allows us to use the set of all instructions already designed in a programming language for other sources and sinks.
- For example,
  - in **C, C++, or Java**, we have several instructions that **can read and write data to other sources and sinks**;
  - a keyboard (a source), a monitor (a sink), or a file (source and sink).
- We are adding only **new sources and sinks** to the programming language without changing the way we send or receive data.

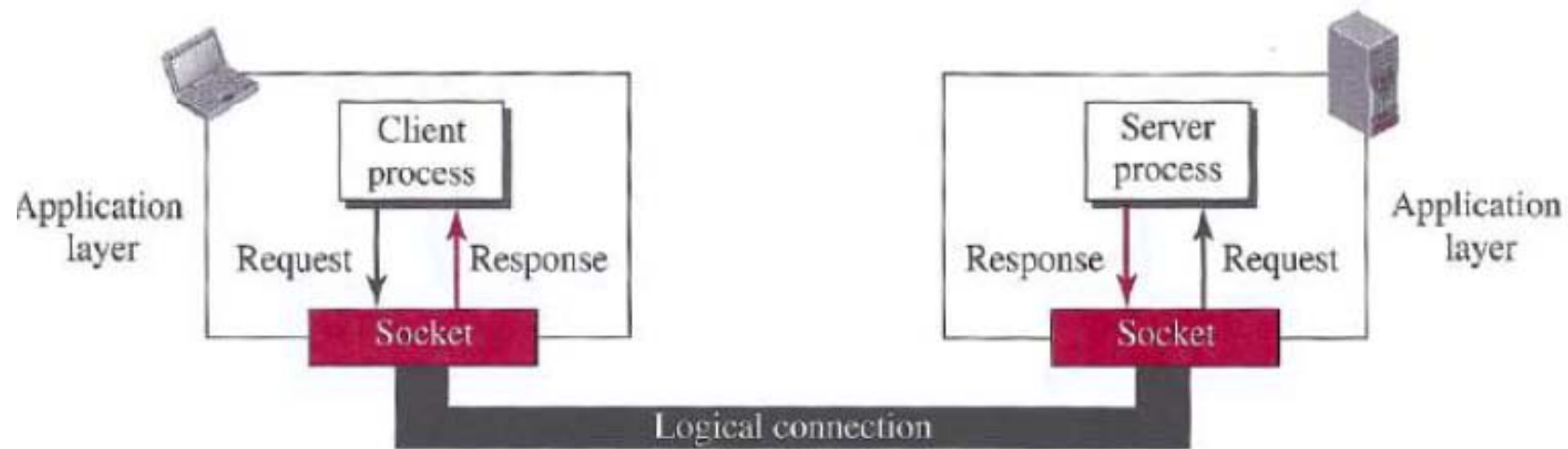
# Cont...



- Socket is not a physical entity like files, keyboard, etc.; **it is an abstraction**

# Cont...

- Communication between a client process and a server process is nothing but **communication between two sockets**



- Need a pair of **socket addresses** for communication:
  - a **local socket address** and a **remote socket address**.

# Cont...

- A **socket address** should
  - **first** **define the computer** on which a client or a server is running.
    - a computer in the Internet is uniquely defined by its **IP address**
  - **Then**, we need another identifier to **define the specific client or server** involved in the communication
    - an application program can be defined by a **port number**
    - Popular applications have been assigned specific port numbers
    - Few port numbers: Web server=80, Mail Server=25
- So, **socket address** = {IP address, port number}





# Finding Socket Addresses

---

- In Server Site:
  - The server needs a **local** (**server**) and a **remote** (**client**) socket address for communication.
- In Client Site:
  - The client also needs a **local** (**client**) and a **remote** (**server**) socket address for communication.
- How can a client / server find a pair of socket addresses for communication?

# Cont...



- In Server Site
  - *Local Socket Address (server)*: Provided by the OS; IP and Port number needs to be defined. For standard services, port numbers are well-known.
  - *Remote Socket Address (client)*: The server can find this socket address from the REQ packet when a client tries to connect to the server.
- In Client Site
  - *Local Socket Address (client)*: Provided by the OS; The port number is assigned to a client process each time the process needs to start the communication; the ephemeral port numbers are assigned to client
  - *Remote Socket Address (server)*: We know well-known port-number of standard application, but don't know IP. We know only URL (e.g. [www.gmail.com](http://www.gmail.com)) , and DNS gives server socket address corresponding to URL.

# Transport Layer Services

---

- The **choice of the transport layer protocol** seriously affects the capability of the application processes.
- broadly classify the possible **transport layer services** along four dimensions:
  - Reliable data transfer
  - Throughput
  - Timing
  - Security
- Application **uses UDP**
  - if it is sending small messages
  - if the simplicity and speed is more important for the application than reliability
  - for lightweight transport protocol, providing minimal services
- Application **uses TCP**
  - if it needs to send **long messages** and require **reliability**
  - for providing **security** it use SSL (Secure Socket Layer)

# Cont...



Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

**Figure 2.5** ♦ Popular Internet applications, their application-layer protocols, and their underlying transport protocols

# Thanks!