- **Application layer protocols**
  - Noticing that the application layer protocols in this Chapter is NOT the entire application protocols. In another word, we only focus on the communication aspect. For example, when we discuss web applications, we do not discuss the web server, browser software and HTML, instead we only cover the HTTP protocol. Similarly, we do not cover the mail server and mail client. In general, an application layer protocol is just one layer of protocol on the Internet protocol stack, which itself is not the entire application protocol.
  - **Client/server vs. Peer-to-Peer.** These are the two approaches structuring a network application. In the *client/server* paradigm, a client process requests a service by sending one or more messages to a server process. The server process implements a service by reading the client request, performing some actions (for example, in the case of an HTTP server, locating the requested Web objects), and sending one or more messages in reply (in the case of HTTP, returning the requested object). In the peer-to-peer approach, the two ends of the protocol are equals, either side can request and serve if desired.

- **Process** and **socket**
  - A network application usually involves two or more **processes** exchanging messages. In the client-server model, we refer them as **client process** and **server process.** Each pair of communication is uniquely determined by the two processes (identified by the **port numbers**) and by the **IP addresses** of the two hosts. In another word, the **4-tuples** (*source IP address, destination IP address, source process port number* and *destination process port number*) uniquely determines one pair of communication. E.g., a client can open up multiple web applications to the same web server, the client side port number of the different web applications must be different (other three are the same).
  - The socket is essentially an interface between application layer and transport layer. This is also referred as socket **API** (application programming interface) between the application and the network, since application developer has control of everything on the top of the socket. In another word, process communication only deals with this interface, it will not see any actual packet or message exchange between any hosts (hidden by the protocol stack).
  - **Pull versus push.** How does one application process get data to or from another application process? In a pull system (such as the Web), the data receiver must explicitly request ("pull") the information. In a push system, the data holder sends the information to the receiver without the receiver's explicitly asking for the data (as in SMTP, when an email is "pushed" from sender to receiver).

- **HTTP**
  - HTTP utilizes TCP (transport protocol), in which a HTTP client first initiates a TCP connection with a HTTP server (see Slides 2-20); once the connection is established, the browser (HTTP client) communicates with the Web server

through their socket interfaces on top of TCP.

- Once a message is sent to the socket interface at a client or server, the message is "in the hand" of TCP, which provides reliable data transfer (details in Chapter 3). In another word, HTTP client and server do not handle the possible loss of packet over the Internet, and how that is recovered by TCP.
- HTTP is a **stateless** protocol, in which client or server does not maintain any state information about the request and response or HTTP connection. If a client asks for the same object from a server twice, the server does not respond by saying that it has just sent this object; instead the server responds with the object each time.
- An interesting point is that HTTP as a stateless protocol relies on TCP, which is a connection-oriented protocol (details in Chapter 3). A connection-oriented protocol is always a state-based protocol, as both sides have to maintain the state of the connection.

- **Nonpersistent** and **Persistent** HTTP Connections
  - A nonpersistent HTTP connection requires that each object be delivered by an individually established TCP connection. In order to improve the performance of nonpersistent http, a HTTP client can possibly establish multiple TCP connections (**parallel** TCP connections) to request multiple objects simultaneously. Each object, however, still requires 2RTT plus the transmission time of the object at the server side, even though overall time can be reduced significantly at the expense of multiple TCP connections.
  - A persistent HTTP connection can send multiple objects over a single TCP connection. This eliminates the overhead in establishing and maintaining multiple TCP connections (we will see discussions in Chapter 3 that the establishment of TCP connections can be time consuming, and maintenance of TCP connections consume resources).
  - The major difference between non-persistent and persistent protocol is the number of TCP connections that are required for transmitting the objects. Nonpersistent HTTP connection requires that each object be delivered by an individually established TCP connection. Usually there is one RTT delay for the initial TCP connection.
  - Please note, however, it always requires two RTT for initial TCP connection and the acquirement of the base-html file (object), since only after receiving the base html file, the client (browser) would know how many objects (excluding the base html) that this web page contains.
  - The **parallel TCP connections** are used by non-persistent HTPP to improve the performance, i.e., to speed up the response time. Ideally, the shortest time that it can receive all objects is two RTT (one for initial TCP, one for request and response) plus the transmission time of the *largest* objects assuming that all the TCP connections can be established simultaneously.
  - Notice that the above calculation is very simplified. As there still can be discrepancies other than the fact there each RTT can be different. For example,

usually one RTT for initial TCP connection, the base html has to be requested first, under both non-persistent HTTP and persistent HTTP; since only after the client receives the base html file and parses that, it can then determine how many objects that this web page contains for subsequent request(s).

- HTTP Request format
  - **GET**, most commonly used request to web server for objects
  - **POST**, to fill out a form. In this case, the message format on Slide 2-31 contains the content in the field of "Entity Body".
  - **HEAD**, similar to GET command except HTTP responds with no requested objects. This is used for debugging. Noticing in both GET and HEAD, the "entity body" in the request message is empty.
  - **PUT**, upload files to web servers.
  - **DELETE**, delete files on web servers, specified in the URL field in the header.

- **Web Caching**, Caching is the act of saving a local copy of a requested piece of information (for example, Web document) that is retrieved from a distant location (server), so that if the same piece of information is requested again, it can be retrieved from the local cache, rather than having to retrieve the information again from the distant location. Caching can improve performance by decreasing response time (since the local cache is closer to the requesting client) and avoiding the use of scare resources (for example access link).

- **Cookies**, Cookies provide a way for a Web server to keep track of users given the stateless nature of a HTTP server. Cookies has 4 components embedded in HTTP request/response header line, Web client (browser) and server database. Cookies can be used for authorization, shopping carts, recommendation and user session state (such as Web-based e-mail).

- **FTP**
  - FTP is an **out-of-band** protocol in that the control and data (file transfer) are exchanged over two different TCP connections. Other protocols such as HTTP, SMTP are **in-band** protocols.
  - The client initiates the ftp session to the well known port number 21 at the ftp server. After the authentication and the exchange of the control info such as directory, whether upload (from client to server) or download a file (from server to client), the ftp server uses the well known port number 20 to establish a new TCP connection for actual file (data) transfer. One TCP connection (data) per file transfer, i.e., nonpersistent in nature.
  - The FTP server maintains the **state**, in particular all the control info about the user (control TCP) such as user account, directory on both sides. This limits the number of simultaneous ftp sessions that FTP can maintain.

- Basic Components in Electronic Mail
  - *User agent*: mail reader composing, editing and reading mails. E.g., Eudora, outlook, elm (for unix), mail based mail (imail.cse.ust.hk, webmail.ust.hk, hotmail, gmail).
  - *Mail servers*: mail box for each registered user and message queues for outgoing e-mails.
  - **SMTP**, transfer messages from sender mail server to recipient mail server, also used to transfer message from user agent (local machine) to the user mail server. This is a **push** protocol, as it "pushes" message to a server.
  - POP3 or IMAP: user to retrieve message from its mail server. This is a **pull** protocol, as it "pulls" a message from a server.

- More on Mail
  - SMTP is an old protocol (much older than HTTP). It restricts the mail message to be in simple 7-bit ASCII format. This poses a problem in attaching large files, esp. media files. This requires binary multimedia data to be encoded to ASCII before being sent over SMTP and it requires the corresponding ASCII message to be decoded back to binary after SMTP transport. HTPP never needs to do that since each object is sent separately in its own format.
  - In a Web-based mail, SMTP is used between sender and recipient mail servers, but HTTP is used for a user to send message to its mail server, and user uses HTTP to retrieve mails from its mail server.

- **Root** DNS server, Top-Level Domain (**TLD**) servers, **authoritative** DNS servers
  - There are 13 root servers (labeled A to M) in the Internet, mostly in North America. 13 root DNS servers operate like a single server, and each of the 13 root DNS server is actually a cluster of replicated servers for security and reliability purposes.
  - TLD servers are responsible for top-level domains such as .com, .org, .net, .gov, .edu, .cn, .hk and etc. There are companies maintaining various servers such as *Network Solutions* maintaining the TLD servers for the com top-level domain, *Educause* maintains the TLD servers for the edu top-level domain. Government usually responsible for country's top-level domains.
  - Each organization with publicly accessible hosts (such as Web servers and mail servers) on the Internet must provide publicly accessible DNS records that map the names of those hosts to IP addresses. An organization's **authoritative DNS server** maintains those DNS records for all hosts within its organization or domain.
  - The root, TLD and authoritative DNS servers all belong to the hierarchy of DNS servers (Slide 2-63).
  - Local DNS servers, as default name servers, with ISP, companies and

institutions. Each host on the Internet is configured to know the IP address of one or more of its local DNS servers. Local DNS servers do not strictly belong to the hierarchy, rather act as proxy for forwarding DNS query to the hierarchy.

- **DNS caching**, DNS caching is extensively used to improve the delay performance and to reduce the number of DNS messages circulating around the Internet. For example, a local DNS server can cache the IP addresses of TLD servers (this is often the case), thereby allowing the local DNS server to bypass the root DNS server in a query chain.

- **Peer-to-Peer or P2P Applications**
  - Note that uploading capacity $\mathbf{u_s}$ from a server or $\mathbf{u_i}$ from a client specifies the bandwidth that a server or a client can transmit (upload) onto the Internet, while download rate $\mathbf{d_i}$ is used to specify the rate that a client can receive (download or retrieve) from the Internet. Recall in ADSL (chapter one) that usually $\mathbf{d_i} > \mathbf{u_i}$. Usually we can use $\mathbf{d_{min}}$ to specify the minimum of the downloading rates of all peers, i.e., $\min\{\mathbf{d_i}\}$.
  - The fundamental benefit (Slide 2-79) in P2P applications is to utilize the uploading bandwidth (capacity) from each peer ($\mathbf{u_i}$), which helps to facilitate the content dissemination, thus help ***scale*** to large population.
  - $D_{c\text{-}s}$ and D_{p2p} (on slides 2-78/79) are the lower bound of the time that the file of size F can be distributed to N peers in client-server and P2P, respectively.
  - The easy part of P2P file sharing is that, once a peer is located with the content, it can be used to transfer the objects from the peer with the content to the requested peer, typically using HTTP or ftp in some cases
  - The central difficulty is to how locate which peer(s) currently on the Internet that has the content. In BitTorrent, **trackers** keep track of peers that shares content (called **torrents**).