Application Layer

- **Required Reading:** Chapter 2 of the text book.

- **Outline of Chapter 2**
  - Network applications
  - HTTP, protocol for web application
  - Protocols for email application
  - DNS, domain name system
  - Protocols for peer-to-peer applications
  - Video streaming
  - Socket programming

- **Application Programs**
  Programs for network applications run on end-systems and communicate over networks. For example, a web server program runs on a server machine and communicates with a browser program running on a client machine. Network application programs use service of transport layer protocols for communication via API (Application Program Interface). Socket API is a de facto API.

- **Application Architectures**
  Major architectures used by network application programs include the client-server model, the peer-to-peer model and a hybrid of the two. In the client-server model, there is a server end-system and a client end-system. A server provides services, is always on duty and has a permanent IP address. A client communicates with a server to get service, may have a dynamic IP address, may be intermittently connected, and does not have interaction with other clients. Web application is an example of the client-server paradigm. In the peer-to-peer (P2P) model, there is no always-on server. Each end system works as both a server and client, communicates with each other. End systems in the P2P model are called peers. Peers may have dynamic address and may be intermittently connected. A network application may use a hybrid of client-server and P2P models. Skype is such an example. In a phone call by Skype, there are connection set-up, conversation and connection termination. The programs for connection set-up follow the client-server model, a centralized server is used. The conversation (voice-over-IP) uses the P2P model.

- **Process**
  A process is a program in running on an end-system. Network applications are realized by processes on end-systems. Communication between processes within a same end-system is realized by operating system (inter-process communication defined by OS). Communication between processes on different end-systems are realized by by exchange messages over a network connecting the end-systems. In the client-server model, a client process initiates the communication and a server process reacts to the request from the client. In the P2P model, each end-system has both client and server processes.
• **API**

Most network applications are in application layer and use transport layer protocols to transmit messages for communication. An operating system (OS) in an end-system provides an interface between application programs and transport layer protocols such as TCP/UDP, known as Application Program Interface (API). Socket API is a result of transporting TCP/IP into UNIX, a *de facto* standard and basis of API. Figure 1 gives a graphical explanation on the socket API.

• **API’s Functionality**

An application program uses TCP/IP via the API for communication. TCP/IP standards do not specify the form of the API (it may depend on operating systems) but suggest its functionality which includes: allocate local resources for communication, specify local and remote communication endpoints, initiate a connection (client side), send a datagram (client side), wait for an incoming connection (server side), send or receive data, determine when data arrives, generate urgent data, handle incoming urgent data, terminate a connection gracefully, handle termination from remote site, abort communication, handle error conditions or a connection abort, and release local resources when communication finishes.

• **Service Required by Applications**

Major requirements for network applications include data loss, data transfer delay, throughput, and security. Different network applications may have different requirements on communications. Some applications has very strict requirements on data loss, for example, a file transfer application may require 100% reliable data transfer but can tolerate a long transfer delay. While some other some applications (e.g., audio) can tolerate occasional data loss but require low and uniform transfer delay. Some applications (e.g., video) require a minimum throughput all the time. Some other applications (e.g., file transfer) have elastic throughput requirement. Some applications require the privacy and integrity for the data transferred.

• **Service from Transport Layer**

In TCP/IP protocol suit, TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are two major protocols provide services to protocols in application layers.
TCP provides reliable data delivery network service which has the following properties.

Virtual circuit connection: Protocols on two end-systems communicate with each other by sending messages across an internet to make an agreement (virtual connection) on transfer data. It looks like to set-up a dedicated path between the two end-systems, although no dedicated physical path is created. When the transfer is finished, the connection is terminated.

TCP uses positive acknowledgment with retransmission to realize reliable data delivery. The destination sends source an ACK message when it receives a packet. The source waits for an ACK for the sent message. If no ACK (use a timer to check), the source retransmits the packet. When data loss occurs, large transfer delay may happen due to retransmission.

TCP has flow control and congestion control.

Applications on TCP: E-mail (SMTP), Web (HTTP), etc.

UDP provides unreliable connectionless (best effort) data transfer between applications. There is no connection set-up and no flow/congestion control in UDP. An application program uses UDP for data transfer usually handles the reliability issues itself. UDP has a small delay.

Applications on UDP: VoIP (e.g., Skype)

- **Web and HTTP**

  HTTP is an application layer protocol for web applications. A web application follows the client-server model. A web server provides services in a form of web pages. A web page has an HTML-file which includes objects. Objects can be an HTML file, JPEG image, audio file, Java applet, etc. Each object is addressed by a URL (Uniform Resource Locator). An example of URL: `www.cs.sfu.ca/`. Web clients use a browser to request, receive and display web objects. Servers send objects in response to client’s requests. Notice that HTTP does not define how a web object is displayed on a client end-system but only deals with the data transfer.

- **HTTP**

  HTTP uses TCP for data transfer. A web client initiates a TCP connection to a web server, using port number 80 (in the Internet, there are two levels of addressing to identify a process on an end-system: an IP address for identifying an end-system and a TCP/UDP port number for identifying the process in the end-system using TCP/UDP). A web server reacts to the TCP connection request from the client. If the request is accepted, a TCP connection is created and HTTP messages are exchanged between client (browser) and server using the TCP connection. Either side can terminate the TCP connection (usually client side initiates the termination). An HTTP server does not maintain info on previous client requests. HTTP can be classified into non-persistent HTTP, at most one object is sent over a TCP connection, and persistent HTTP, multiple objects can be sent over a TCP connection.

- **Non-Persistent HTTP**

  A non-persistent HTTP is simple but less efficient. To see this, let RTT be the round trip time, the time for a packet from a client to a server and back, and let $t_m$ be the transmit time for sending $m$ objects to a network. The non-persistent HTTP response time for one object is one RTT (time to create a TCP connection) plus one RTT (time for requesting and receiving
the object) plus $t_1$ which gives $2\text{RTT}+t_1$, and the non-persistent HTTP response time for the last of $n$ objects by one TCP connection is $2n\text{RTT}+nt_1$ (see Figure 2).

The delay of non-persistent HTTP is large. The delay can be improved by parallel TCP connections. The non-persistent HTTP response time for the last of $n$ objects by $m$ parallel TCP connections is $2\left\lceil \frac{n}{m} \right\rceil \text{RTT}+\left\lceil \frac{n}{m} \right\rceil t_m$. A web browser may support up to 5-10 parallel TCP connections. The OS overhead is large, one TCP connection for every object.

- **Persistent HTTP**

  A persistent HTTP is more complex but more efficient. A server responds to a client’s request to open a TCP connection. Subsequent HTTP objects between same client/server are sent over the same TCP connection. The delay for one object is $2\text{RTT}+t_1$. The delay for the last of $n$ objects over one TCP connection without pipelining is $\text{RTT}+n(\text{RTT}+t_1)$, and the delay for the last of $n$ objects over one TCP connection with pipelining is $2\text{RTT}+nt_1$ (see Figure 3). The OS overhead is small (one TCP connection for all objects). Persistent with pipelining is the default mode of HTTP.

- **HTTP Request Message**

  An HTTP request message has a request line, header lines, and an entity body. There are a method field, a URL field, and a version field in the request line (see Figure 4). The method field specifies how to get an object from a server. It can take the following values:

  - **GET**, the requested object is specified by the link in the URL field (v1.0/1.1/2). Most HTTP requests use GET. Entity body is empty when GET is used.
  - **POST**, the requested object is specified in entity body (v1.0/1.1/2). For example, a client may send a request for an object specified by key words to a search engine. In this case, the key words are included in entity body.
HTTP allows multiple header lines in a request message. One well used header line is for cookies. Entity body may contain additional info about the request such as key words for a search engine or an object for uploading.

- **HTTP Response Message**
HTTP response message is similar to that of HTTP request message and has a status line, header lines and an entity body. The status line contains HTTP/version, protocol status code
and status phrase/data. For example, the status line may contain `HTTP/1.1 200 OK`. The following are well used status codes and phrases:

- 200 OK, request succeeded, requested object in the response.
- 301 Moved Permanently, requested object removed, new location specified in the response.
- 400 Bad Request, request message not understood by server.
- 404 Not Found, requested object not found at the server.
- 505 HTTP Version Not Supported.

• Cookies

To improve the efficiency of communications, a web server may want to keep the client-server interaction history and state. Cookies are designed for this purpose. A cookie has four components: A cookie header line in the HTTP response message; A cookie header line in the HTTP request message; a cookie file kept in user’s host and managed by user’s browser; and a back-end database at server (web site). Cookies can be used to improve the efficiency of web applications but introduce security problems.

• Web Cache

Cache is well used in Web application to meet client’s request without a connection to an original server. The idea is that when a client downloads an object from a server, the object is cached at client’s machine or a proxy server. When the client or others want the object again, the cache provides the object. Cache technique reduces the response time and traffic significantly.

• A Cache Example

Assume that clients at an institution have 20 requests/second and the object size of each request is 100,000 bits. The round trip between the gateway G of the institution and the original server is 2 seconds. The round trip between a client and G is 10 milliseconds.

If no cache is used then 2Mbps from G to clients, 2Mbps from server to G, and average delay is 2seconds+milliseconds.

If 50% of the objects are cached at G then 2Mbps from G to clients, 1Mbps from server to G, and the average delay is 1seconds+milliseconds.

• Electronic Mail

Email is a major network application. There are three major components (user agent, mail server and simple mail transfer protocol, SMTP) in an email system. A user agent or mail reader is used to compose, edit and read mail messages. A mail server has mailboxes to keep incoming messages and message queues to keep messages to be sent. SMTP protocol transfers mails between mail servers.

• SMTP
SMTP uses TCP (port 25) to transfer messages between mail servers. Messages are transferred directly from sending server to receiving server without going through intermediate servers. There are three phases of transfer (a typical TCP connection): handshake to create TCP connection, transfer of messages, and connection closure.

- **Email Example**

  1. A composes a message and send to bob@someschool.edu.
  2. A’s user agent sends message to A’s mail server, message placed in message queue.
  3. SMTP at A’s server opens a TCP connection with B’s mail server.
  4. SMTP transfers message from A’s server to B’s server.
  5. B’s mail server puts message in B’s mailbox.
  6. B uses user agent to read message.

- **SMTP Message Format**

  An SMTP message has a header (the contents in the header are not SMTP commands) which include a **to:** line, a **From:** line, and a **Subject:** line. After the header, there is a message body which is ASCII codes only.

- **Comparison between HTTP and SMTP**

  Both use persistent TCP connections for file transfer. HTTP is a mainly a pull protocol, clients download information from a Web server. SMTP is a push protocol, a sending mail server pushes files to a receiving mail server. HTTP supports data in different formats. SMTP can only transfer data in ASCII form (data not in ASCII is converted to ASCII codes). HTTP encapsulates each object in one response message. SMTP puts all objects of a file into one message.

- **Mail Access Protocols**

  SMTP is a protocol for transferring mail messages from one mail server to another. Other protocols are used to retrieval messages from mail server. Commonly used protocols include POP, IMAP and HTTP. POP (Post Office Protocol) is a simple protocol to access emails. It has authorization (agent<→server) and download functions. IMAP (Internet Mail Access Protocol) is more complex and have more features. HTTP (gmail, hotmail, yahoo, SFU connect, etc.) are now commonly used as email access protocols.

- **Domain Name System (DNS)**

  In the Internet, hosts/routers are identified by IP addresses. However, it is easier to identify hosts/routers by names for humans. Domain name system is a scheme for (1) assigning meaningful high-level names for machines and (2) mapping between the names and the IP addresses of the machines. The scheme handles both the translation from names to IP addresses and from IP addresses to names.

  DNS supports host aliasing. A host in the Internet is usually identified by a unique name (called canonical name) which may be long/complex and difficult to memorize. In such a case,
the host is often given simpler alias names. DNS provides the binding between the canonical name and alias names. Similarly, DNS also supports mail server aliasing.

DNS can be used to support load balancing. An example, an Internet service is provided by multiple servers with a same alias name; DNS binds the alias name to the canonical name of a server with the lightest load.

- **Naming a Machine**

  When the Internet was small, the name-address business was handled by a central site, the Network Information Center (NIC). When the Internet grew and became large, a distributed system was developed for the name-address mapping. The system partitions the name space to give hierarchical machine names and delegate authorities over the partitions.

  In a TCP/IP internet, hierarchical machine names are assigned based on the structure of organizations that have the authority for parts of the namespace, not necessarily based on the structure of the physical network interconnection.

  The Domain Name System (DNS) is the mechanism which implements a machine name hierarchy for a TCP/IP internet. DNS specifies the name syntax and rules for delegating authorities over names and the implementation of a distributed computing system that maps the names to addresses.

- **Domain Name Space**

  To have a hierarchical name space, a *domain name space* is designed. In the domain name space, the names for machines are defined in an inverse-tree structure with the root at the top. Each node in the tree has a *label* which is a string of characters. The label is called a *domain name*. The root has a null label. Each child node of the root is called a *Top-Level Domain* (TLD). Figure 5 give an example of the domain name space.

  A machine is assigned a *full domain name* which is a sequence of domain names separated by dots (.). The domain names are always read from the node up to the root.

  A *domain* is a subtree of the domain name space. The name of the domain is the domain name of the root of the subtree.

- **Internet Domain Names**

  The TLDs are partitioned into three groups: geographic domains, organizational domains and inverse domain.

  Geographic domains: The top-level domains are grouped by countries/regions. Each country/region is given a 2-letter top-level domain name. For example, machines in Canada fall in the top-level domain *ca*.

  Organizational domains: The top-level domains are groups by organizational type. For example *com* for commercial organizations, *edu* for educational institutions (4 years), *gov* for government institutions, etc.

  Inverse domain: The inverse domain is used to map an address to a name. The 1st level node in the domain has label *arpa* and the 2nd level node has label *in-addr*. The rest of the
A name for a machine (full domain name) is a sequence of the domain names from a leaf to a TLD, separated by dot.

• Map Domain Name to Address

In addition to the rules for assigning domain names to machines, the domain name scheme includes mapping names to addresses. To do so, the information in the domain name space should be stored. But it is inefficient and unreliable to store the information at a single site (machine). The solution for these problems is to distribute the information among many machines called DNS (or name) servers which provide name-to-address translation, mapping domain names to IP addresses. Each name server can be responsible for a specific domain (subtree). A client uses a name resolver (a client program) which uses one or more name servers to find the name-to-address mapping.

A DNS domain is a complete subtree in the DNS name tree. A zone is a domain managed by one server (one authority), minus any delegated subtrees. The server for a zone keeps a database called zone file to store all the information in the zone and some partial information on the delegated subdomains. An administrative authority may have control of several zones, e.g., a zone for the domain the authority controls and a zone for the corresponding .in-addr.arpa domain.

A root server is a server whose zones cover the whole DNS name tree. A root server usually does not keep any information about domains but delegates its authority to other servers and keeps references to those servers.
A primary server is a server which is responsible for creating, maintaining, and updating the zone file.

A secondary server is a server which transfers the complete information about a zone from another server. It neither creates nor updates the zone files.

- **Domain Name Resolution**
  Domain name resolution includes mapping names to IP addresses and mapping IP addresses to names. The DNS is designed as a client-server application. A client which needs a name resolution calls the DNS client (resolver). The resolver sends the query to the nearest name server.

  The resolution can be recursive and iterative. In *recursive resolution*, if the server has the answer, it replies with the answer. Otherwise, it forwards the query to another server (usually the parent) and waits for response.

  In *iterative resolution*, if the server has the answer, it replies with the answer. Otherwise, it returns the client an IP address of a server that it thinks can resolve the query.

  DNS requires each name server knows its parent server and at least one root server.

- **DNS Message**
  DNS has two types of messages: query and response. The query message has a header and the question records; the response message has a header, question records, answer records, authoritative records, and additional records.

- **Peer To Peer Application**
  A basic architecture for P2P model is that each end system acts both a client and a server, there is no always-on server, and end systems (peers) are intermittently connected. P2P model may improve the communication efficiency over client-server model for some applications. Let us see this by an example on the time to distribute a file of $F$ bits from a server to $N$ clients. Let $u_s$ be the server upload bandwidth, $u_i$ be the client $i$ upload bandwidth, and $d_i$ be the client $i$ download bandwidth.

  By the client-server model:
  
  - Server takes $(N \times F)/u_s$ time to send $N$ copies of $F$
  - Client $i$ takes $F/d_i$ time to download $F$
  - Total time $\max\{(N \times F)/u_s, \frac{F}{\min_{i=1}^{N} \{d_i\}}\}$, increase linearly in $N$

  By P2P model:
  
  - Server takes $F/u_s$ time to upload one copy of $F$
  - Client $i$ takes $F/d_i$ time to download $F$
  - Total time $\max\{F/u_s, \frac{F}{\min_{i=1}^{N} \{d_i\}}, \frac{N \times F}{u_s + \sum_{i=1}^{N} u_i}\}$
• P2P File Distribution
A key issue in P2P application is how to distribute a file to peers with minimum delay and bandwidth. BitTorrent is a well used protocol for P2P file distribution. Torrent and tracker are two important concepts in BitTorrent. In BitTorrent, a file is partitioned into chunks and each peer may hold a subset of the chunks. Torrent is a group of peers exchanging chunks of file. Tracker is a scheme to track peers participating in a torrent.

• BitTorrent
In BitTorrent, a file is divided into *chunks*, each has 256K Bytes. A peer who wishes to receive a file joins a torrent, registers with the tracker to get a list of peers in the torrent, and connects to a subset of peers (neighbors) to download chunks of the file from neighbors. While a peer is downloading chunks of a file, it may respond to other peers’ requests to upload chunks of the file it holds to other peers. Peers can join and leave torrent dynamically.

Each peer in a torrent has a subset of chunks of a file. Each asks neighbors for list of chunks they have, sends requests to neighbors for chunks the peer does not have. This is known as pulling chunks.

Peers use tit-for-tat policy to send chunks. A peer sends chunks to four neighbors currently sending itself chunks at the highest rate (re-evaluate top 4 in every 10 seconds). In every 30 seconds, a peer randomly selects another peer and sends the selected peer chunks.

• Video Streaming and Content Distribution Networks
One major application in the Internet is video streaming system. In this system pre-recorded videos are placed at servers and clients view (download) the videos on demand. There are many well known video streaming services such as Netflix, Youtube, Amazon, Youku, etc. A major challenge in video streaming is how to provide millions videos to billions clients of different capabilities with satisfactory qualities. Main quality requirements here include the end-to-end delay and the quality of the pictures in the video.

Major approaches to address the challenge includes: (1) Compress the video into multiple quality levels and provide a client the video with a quality level which matches the capability of the client. This can avoid the long end-to-end delay caused by a limited bandwidth at the server-to-client’s path and also reduces the bandwidth used for the video transfer. (2) Use content distribution networks (CDN)s to provide services. In a CDN, servers are placed all over the world and close to clients to provide better services.

• HTTP Streaming and DASH
HTTP streaming is a simple technique to provide video distribution service. In HTTP streaming, a video is placed at HTTP servers identified by a URL, a client sends a request for the URL and a server sends the video to the client. HTTP streaming is simple but may provide only one quality level for clients with different capabilities. Especially, HTTP streaming is not adaptive to dynamic changes of bandwidth on the server-to-client path.

Dynamic adaptive streaming over HTTP (DASH) is an improvement over HTTP streaming. In DASH, a video is compressed into multiple quality levels and the video of each quality level is divided in multiple chunks. A client requests the video in unit of chunks (a few seconds)
with a quality level subject to the available bandwidth dynamically. DASH provides videos of multiple quality levels to satisfy different requirements of clients and is adaptive to dynamic bandwidth changes. But DASH is more complex and introduces more overhead in the Internet (needs to measure the bandwidth of the server-to-client path).

- **Content Distribution Networks (CDNs)**

For a better distribution of videos to clients, service providers usually place multiple servers close to clients in the Internet and connect the servers by a network called CDN. Videos are placed at servers and a client gets videos from a nearby server. A CDN can be a dedicated network (owned by a service provider, e.g., Google owns its own CDNs) or a virtual network based on the service of the Internet.

Approaches for placing servers include: *enter deep* and *bring home*. In the enter deep approach, servers are placed at/close to the edges of the Internet. In this approach, the servers are close to clients (advantage) but it requires a large number of servers and is difficult to manage the servers.

In the bring home approach, servers are placed at the Internet Exchange Points (IXPs). An IXP is a point which connects multiple peer Internet service providers (ISPs). Two ISPs are called peer if they use the service from each other but do not charge for the service provided. This approach may require a small number of servers, making the server management easier. But the servers may not be close to clients and high capability of servers may be required.

- **A CDN Example**

Google owns a dedicated CDN which has three tiers of servers. There are 14 mega data centers (tier 1 servers), the number of servers in each center is in the order of $10^5$. The centers provide dynamic content service like search results and Gmail messages. There are about 50 clusters (tier 2 servers) in IXPs, each cluster has hundreds of servers. The clusters provide static content service like Youtube videos. There are several hundreds enter-deep clusters at the Internet edges, each enter-deep cluster has dozens servers. The enter-deep clusters provide static content service like static portions of Web pages.

- **CDN Operations**

When a client makes a request for a video (URL), the CDN will first find a server which contains the video identified by the URL and is close to the client (or has the lightest work load), and then redirects the client to the server. Here is an example:

1. A client visits the Web page of NetCinema.
2. The client sends a DNS query for *video.netcinema.com*.
3. A local DNS server $A$ forwards the DNS query to a DNS server $B$ of NetCinema and $B$ returns $A$ a domain name *a1105.kingcdn.com*.
4. $A$ sends a DNS query for *a1105.kingcdn.com*, the query is forwarded to a DNS server $C$ in KingCDN system, and $C$ returns the IP address of a server which has the video.
5. $A$ forwards the IP address to the client.
6. The client uses the IP address to download the video.
• **Socket Programming**

An OS provides an interface between application programs and the transport layer protocols (TCP/UDP), known as Application Program Interface (API). API’s functionality includes the following:

- Allocate local resources for communication
- Specify local/remote communication endpoints
- Initiate connection (client side)
- Wait for incoming connection (server side)
- Send/receive data
- Terminate connection, handle termination from remote side
- Generate urgent data, handle incoming urgent data
- Abort communication, handle errors or connection abort
- Release resources when communication finishes

• **The Socket API**

The socket API is the result of transporting TCP/IP software into UNIX operating system by a group at the University of California at Berkeley. The system is known as BSD UNIX (Berkeley Software Distribution UNIX). The socket API is now a *de facto* standard and forms the basis of the API for many operating systems. The socket API adds a new abstraction, *socket*, for network communication. Like a file descriptor, a socket is identified by a *socket descriptor* which is an integer.

• **Socket Programming with TCP**

In a client-server model, a server process has created a socket *sd* that waits for client’s contact. A client contacts the server to create a TCP connection. To do so the client creates a client-local TCP socket, specifies the server process (IP address, port number), and the client TCP initiates a connection to the server TCP. When contacted by a client TCP, the server TCP creates a new socket for the server process to communicate with the client (the original socket *sd* is used to wait for other clients’ contacts), port numbers are used to distinguish clients. Figure 6 gives an explanation on socket programming with TCP.
Figure 6: Socket programming with TCP.