**Application architecture**: designed by the application developer and dictates how the application is structured over the various end systems.

**Server**: an always on host that services requests from many other hosts.

**Client**: sends a request to a server. Clients don’t directly communicate with each other in client-server architecture.

**Client-Server architecture:** clients don’t directly communicate with each other and the server has a fixed address.

**IP address**: the fixed address of a server (32 bits)

**Data center**: houses many hosts to mimic a virtual server which makes it easier to serve multiple requests from multiple clients.

**P2P architecture**: there is minimal (or no) reliance on dedicated servers in data centers. The application exploits direct communication between pairs of connected hosts, peers. Is self-scalable.

**Challenges of P2P architecture**: they aren’t ISP friendly because most ISPs can’t handle a lot of upstream traffic, not as secure, the success of the P2P application depends on how many people are using it.

**Differences between client/server and P2P**: in P2P, there is no concept of dedicated hosts serving others. Each host has the same capabilities and responsibilities as the next host.

**Process**: a program running within a host/end system. Processes can communicate each other whether they’re on the same host or not. They communicate by sending messages to each other.

**Network application**: consists of pairs of processes that send messages to each other over a network. Each process in the pair is labeled either a client or a server depending on what they are doing.

**Client (process communication)**: the process that contacts the other process at the beginning of the session

**Server (process communication)**: the process that waits to be contacted to begin the session.

In the Web, a browser process initializes contact with a Web server process; hence the browser process is the client and the Web server process is the server. In

P2P file sharing, when Peer A asks Peer B to send a specific file, Peer A is the client and Peer B is the server in the context of this specific communication session.

**Examples of network applications**: email, p2p file sharing, web, text messaging, YouTube

**Socket**: a software interface that a process uses to send and receive messages through the network. It is the interface between the application layer and the transport layer within a host. When a process wants to send a message to another process on another host, it shoves the message out its door (socket). This sending process assumes that there is a transportation infrastructure on the other side of its door that will transport the message to the door of the destination process. Once the message arrives at the destination host, the message passes through the receiving process’s door (socket), and the receiving process then acts on the message.

**Application Programming Interface (API)**: a socket also acts like an interface between the application layer and the network since the socket is the programming interface where network applications are built.

**Information needed for a process to send packets to a process in a different host**: the address of the host and port number of the process on that host.

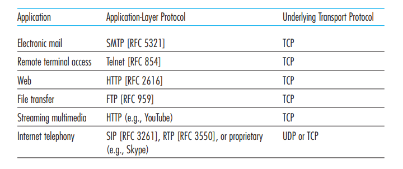
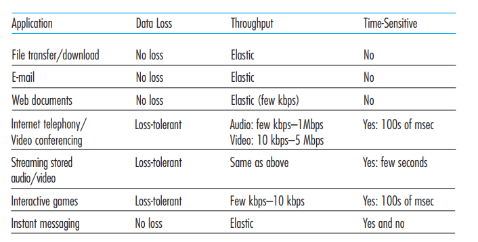
**Popular port numbers**: 80 – web server, 25 – mail server

**Services that a transport service can provide** – data integrity, throughput, security and timing guarantees.

**Reliable data transfer** – occurs when a protocol provides a guaranteed data delivery service.

**Loss-tolerant applications** – applications where some data loss is acceptable, such as conversational audio/video.

**Throughput (for processes communicating with each other)** - the rate at which the sending process can deliver bits to the receiving process. A transport protocol could offer the service of providing a guaranteed available throughput at some rate.

**Bandwidth-sensitive application** – an application that has throughput requirements such as a multimedia application (video).

**Elastic application** – an application that doesn’t have a throughout requirement, such as email or file transfers.

**Timing -** Interactive real-time applications would be concerned with making sure that whatever the sender pumps into the socket, the data is received no longer than a set amount of time.

**Security** – a transport protocol could encrypt all data transmitted and the receiving host would have to decrypt the data before delivering it to the correct process.

**TCP Service** – a connection-oriented service that offers reliable data transfer. TCP has the client and server exchange transport layer control information with each other before the application-level messages begin to flow. This so-called handshaking procedure alerts the client and server, allowing them to prepare for an onslaught of packets. After the handshaking phase, a TCP connection is said to exist between the sockets of the two processes. Doesn’t provide encryption (SSL is TCP with security). The communicating processes can rely on TCP to deliver all data sent without error and in the proper order. When one side of the application passes a stream of bytes into a socket, it can count on TCP to deliver the same stream of bytes to the receiving socket, with no missing or duplicate bytes.

**Congestion-control mechanism** – TCP will throttle a sending process (client or server) when the network is congested between a sender and receiver.

**UDP Service**: provides just the bare minimum of services. It is connectionless, so there is no handshaking before the two processes start to

communicate. It provides unreliable data transfer service and doesn’t include congestion control.

Neither protocol offers timing or throughput guarantees.

**Application-layer protocol**: defines how an application’s processes, running on different end systems, pass messages to each other. It defines the type of messages exchanged, the syntax of the various message types, the semantics of the fields and the rules for determining when and how a process sends and responds to messages. This is only one piece of a network application.

An Internet e-mail application has many components, including mail servers that house user mailboxes;

mail clients (such as Microsoft Outlook) that allow users to read and create messages; a standard for defining the structure of an e-mail message; and application-layer protocols 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. The principal application-layer protocol for electronic mail is SMTP (Simple Mail Transfer Protocol) Thus, e-mail’s principal application-layer protocol, SMTP, is only one piece of the e-mail application.

**What is the difference between network architecture and application architecture?** From the application developer’s perspective, the network architecture is fixed and provides a specific set of services to applications. The application architecture, on the other hand, is designed by the application developer and dictates how the application is structured over the various end systems. Network architecture: how your network is built, i.e. servers, routers, PC's etc. Application architecture: how your software is built.

**For a P2P file-sharing application, do you agree with the statement, “There is no notion of client and server sides of a communication session”? Why or**

**why not?** No. In a P2P file-sharing application, the peer that is receiving a file is typically the client and the peer that is sending the file is typically the server.

**Suppose you wanted to do a transaction from a remote client to a server as fast as possible. Would you use UDP or TCP? Why?** You would use UDP. With UDP, the transaction can be completed in one roundtrip time (RTT) - the client sends the transaction request into a UDP socket, and the server sends the reply back to the client's UDP socket. With TCP, a minimum of two RTTs are needed - one to set-up the TCP connection, and another for the client to send the request, and for the server to send back the reply.

**Does SSL operate at the transport layer or the application layer? If the application developer wants TCP to be enhanced with SSL, what does the developer have to do?** SSL operates at the application layer. The SSL socket takes unencrypted data from the application layer, encrypts it and then passes it to the TCP socket. If the application developer wants TCP to be enhanced with SSL, she has to include the SSL code in the application.

**Why do HTTP, FTP, SMTP, and POP3 run on top of TCP rather than on UDP?** The applications associated with those protocols require that all application data be received in the correct order and without gaps. TCP provides this service whereas UDP does not.