**::OSI LAYERS::**

**1. Physical Layer** :-- The physical layer is responsible for the physical cable or wireless connection between network nodes. It defines the connector, the electrical cable or wireless technology connecting the devices, and is responsible for transmission of the raw data, which is simply a series of 0s and 1s, while taking care of bit rate control.

**2. Data Link Layer** :-- The data link layer establishes and terminates a connection between two physically-connected nodes on a network. It breaks up packets into frames and sends them from source to destination. This layer is composed of two parts—Logical Link Control (LLC), which identifies network protocols, performs error checking and synchronizes frames, and Media Access Control (MAC) which uses MAC addresses to connect devices and define permissions to transmit and receive data.

**3. Network Layer** :-- The network layer has two main functions. One is breaking up segments into network packets, and reassembling the packets on the receiving end. The other is routing packets by discovering the best path across a physical network. The network layer uses network addresses (typically Internet Protocol addresses) to route packets to a destination node.

**4. Transport Layer** :-- The transport layer takes data transferred in the session layer and breaks it into “segments” on the transmitting end. It is responsible for reassembling the segments on the receiving end, turning it back into data that can be used by the session layer. The transport layer carries out flow control, sending data at a rate that matches the connection speed of the receiving device, and error control, checking if data was received incorrectly and if not, requesting it again.

**5. Session Layer :--**The session layer creates communication channels, called sessions, between devices. It is responsible for opening sessions, ensuring they remain open and functional while data is being transferred, and closing them when communication ends. The session layer can also set checkpoints during a data transfer—if the session is interrupted, devices can resume data transfer from the last checkpoint.

**6. Presentation Layer :--**The presentation layer prepares data for the application layer. It defines how two devices should encode, encrypt, and compress data so it is received correctly on the other end. The presentation layer takes any data transmitted by the application layer and prepares it for transmission over the session layer.

**7. Application Layer :--**The application layer is used by end-user software such as web browsers and email clients. It provides protocols that allow software to send and receive information and present meaningful data to users. A few examples of application layer protocols are the [Hypertext Transfer Protocol](https://www.imperva.com/learn/performance/http2/) (HTTP), File Transfer Protocol (FTP), Post Office Protocol (POP), Simple Mail Transfer Protocol (SMTP), and Domain Name System (DNS).

::: [OSI vs. TCP/IP Model](https://www.imperva.com/learn/application-security/osi-model/#osi-vs-tcpip-model) :::

The [Transfer Control Protocol/Internet Protocol](https://www.imperva.com/learn/application-security/tcp-transmission-control-protocol/) (TCP/IP) is older than the OSI model and was created by the US Department of Defense (DoD). A key difference between the models is that TCP/IP is simpler, collapsing several OSI layers into one:

* OSI layers 5, 6, 7 are combined into one Application Layer in TCP/IP
* OSI layers 1, 2 are combined into one Network Access Layer in TCP/IP – however TCP/IP does not take responsibility for sequencing and acknowledgement functions, leaving these to the underlying transport layer.

Other important differences:

* TCP/IP is a functional model designed to solve specific communication problems, and which is based on specific, standard protocols. OSI is a generic, protocol-independent model intended to describe all forms of network communication.
* In TCP/IP, most applications use all the layers, while in OSI simple applications do not use all seven layers. Only layers 1, 2 and 3 are mandatory to enable any data communication.

-----------------------------------------------------------------------------------------------------------------------------------------

Network : Interconnection of computers is called network LAN WAN MAN  
Client : The computer that receive services  
Server : The computer that provide services  
  
  
  
3 requirements to establish network :  
 Hardware : Includes Computers,cables,modems,routers,hubs  
 Software : Includes programs to communicate between server and client  
 Protocol : Representing way to establish connections and  
 helps to send and receive data in standard format

**What is DNS?**

The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through [domain names](https://www.cloudflare.com/learning/dns/glossary/what-is-a-domain-name/), like nytimes.com or espn.com. Web browsers interact through [Internet Protocol (IP)](https://www.cloudflare.com/learning/network-layer/internet-protocol/) addresses. DNS translates domain names to [IP addresses](https://www.cloudflare.com/learning/dns/glossary/what-is-my-ip-address/) so browsers can load Internet resources.

Each device connected to the Internet has a unique IP address which other machines use to find the device. DNS servers eliminate the need for humans to memorize IP addresses such as 192.168.1.1 (in IPv4), or more complex newer alphanumeric IP addresses such as 2400:cb00:2048:1::c629:d7a2 (in IPv6).

Python Internet modules

A list of some important modules in Python Network/Internet programming.

|  |  |  |  |
| --- | --- | --- | --- |
| **Protocol** | **Common function** | **Port No** | **Python module** |
| HTTP | Web pages | 80 | httplib, urllib, xmlrpclib |
| NNTP | Usenet news | 119 | nntplib |
| FTP | File transfers | 20 | ftplib, urllib |
| SMTP | Sending email | 25 | smtplib |
| POP3 | Fetching email | 110 | poplib |
| IMAP4 | Fetching email | 143 | imaplib |
| Telnet | Command lines | 23 | telnetlib |
| Gopher | Document transfers | 70 | gopherlib, urllib |

Reserved port numbers and associated services :  
-----------------------------------------------  
Socket : Logical connecting point between client and server  
 Each socket is given unique identification number called "port number"  
 Port number can take 2 bytes(0 to 65535 bits) 2 bytes = 8 bits + 8 bits  
Socket Programming : Establishing connection between client and server is called "Socket Programming"

What is Sockets?

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The *socket* library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Sockets have their own vocabulary −

|  |  |
| --- | --- |
| **Sr.No.** | **Term & Description** |
| 1 | **Domain**  The family of protocols that is used as the transport mechanism. These values are constants such as AF\_INET, PF\_INET, PF\_UNIX, PF\_X25, and so on. |
| 2 | **Type**  The type of communications between the two endpoints, typically SOCK\_STREAM for connection-oriented protocols and SOCK\_DGRAM for connectionless protocols. |
| 3 | **Protocol**  Typically zero, this may be used to identify a variant of a protocol within a domain and type. |
| 4 | **Hostname**  The identifier of a network interface −   * A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation * A string "<broadcast>", which specifies an INADDR\_BROADCAST address. * A zero-length string, which specifies INADDR\_ANY, or * An Integer, interpreted as a binary address in host byte order. |
| 5 | **Port**  Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service. |

The *socket* Module

To create a socket, you must use the *socket.socket()* function available in *socket* module, which has the general syntax −

s = socket.socket (socket\_family, socket\_type, protocol=0)

Here is the description of the parameters −

* **socket\_family** − This is either AF\_UNIX or AF\_INET, as explained earlier.
* **socket\_type** − This is either SOCK\_STREAM or SOCK\_DGRAM.
* **protocol** − This is usually left out, defaulting to 0.

Once you have *socket* object, then you can use required functions to create your client or server program. Following is the list of functions required −

Server Socket Methods

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.bind()** : This method binds address (hostname, port number pair) to socket. |
| 2 | **s.listen()** : This method sets up and start TCP listener. |
| 3 | **s.accept()** : This passively accept TCP client connection, waiting until connection arrives (blocking). |

Client Socket Methods

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.connect()** : This method actively initiates TCP server connection. |

General Socket Methods

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.recv()** : This method receives TCP message |
| 2 | **s.send()** : This method transmits TCP message |
| 3 | **s.recvfrom()** : This method receives UDP message |
| 4 | **s.sendto()** : This method transmits UDP message |
| 5 | **s.close()** : This method closes socket |
| 6 | **socket.gethostname()** : Returns the hostname. |

**::: TO KNOW THE IP ADDRESS OF A WEBSITE :::::**

**import** socket  
*# Take server name*host = input(**"Enter website address "**)  
  
**try**:  
 *# To know IP Address of any website* addr = socket.gethostbyname(host)  
 print(**"IP Address of website "**, addr)  
 print(**"Host Address of website "**, socket.gethostbyaddr(host))  
 print(**"Host HostName ex of website "**, socket.gethostbyname\_ex(host))  
 print(**"Host Name of website "**, socket.gethostname())  
**except** socket.gaierror:  
 print(**"Website doesnot exist"**)