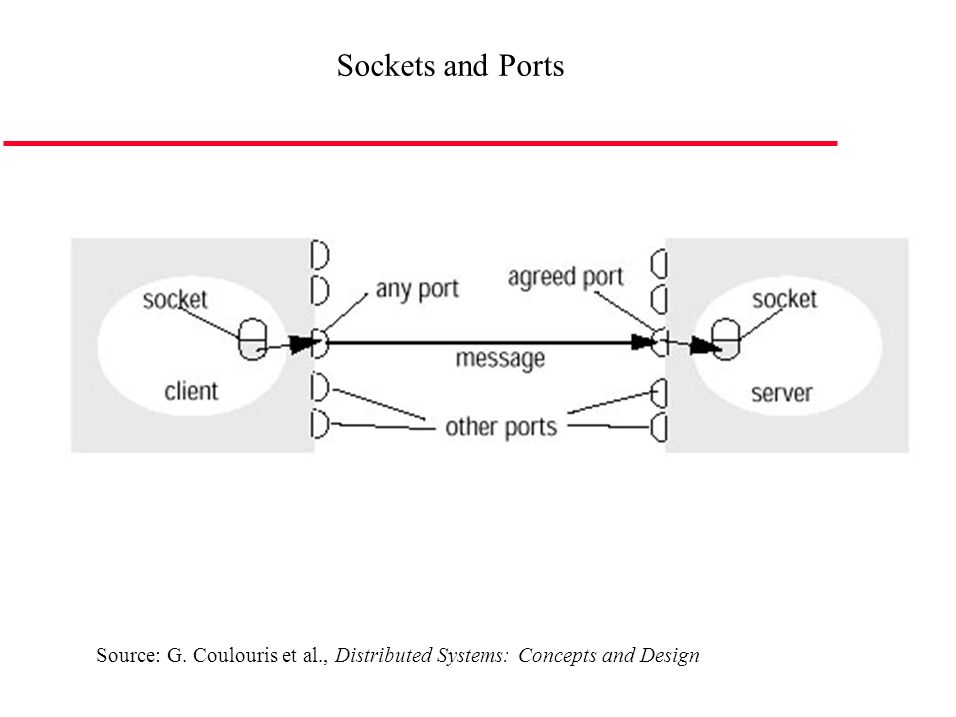
**FAQ’s:**

1. **What is the difference between client- server and peer-to-peer architectures?**

|  |  |
| --- | --- |
| peer to peer | Client - server |
| Decentralized form of networking architecture | Centralized form of networking architecture |
| The network access, tasks and workload are divided and shared amongst the various members. It is an "everyone pulls their own weight" sort of relationship | Working is based on a resource provider or storehouse (server) and the entities that require the resources (clients). The clients make requests to the server to access the resources. It is a "make a request and it will be granted" sort of service |
| Supply and consumption of resources is carried out by the peers, there is no higher body or "boss" and no separate entity exists to dole out resources. All peers in a network can request for resources as well as grant them | Two members of such a system are servers and clients. Servers contain the resources in the form of information or data. Certain resources like say printers, can be connected to the server and the client has to request access to the server, to use the printer |
| Members are called peers, they have the same privileges and rights and enjoy the same access to various data sources and devices. There is no difference between them in any manner. Peers communicate with each other directly, no need for a median in the middle | Clients are the respective workstations or computers that do not share their resources but work on their own and makes requests to the server for data or resources or functions |
| The Peer-to-Peer network paradigm is commonly used in P2P file sharing programs like Napster and Bitorrent | Email, banking services, even the HTTP protocol are all examples of client server model |

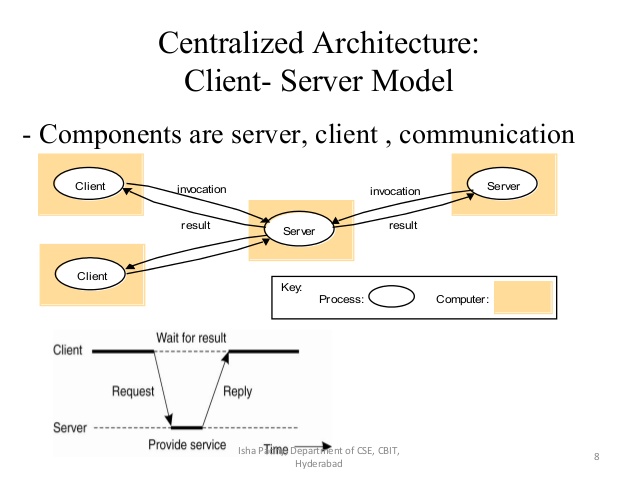
1. **What is socket?**

Both forms of communication (UDP and TCP) use the socket abstraction, which provides an endpoint for communication between processes. Sockets originate from BSD UNIX but are also present in most other versions of UNIX, including Linux as well as Windows and the Macintosh OS. Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process, as illustrated in Figure . For a process to receive messages, its socket must be bound to a local port and one of the Internet addresses of the computer on which it runs. Messages sent to a particular Internet address and port number can be received only by a process whose socket is associated with that Internet address and port number.

Processes may use the same socket for sending and receiving messages. Each computer has a large number (2 16 ) of possible port numbers for use by local processes for receiving messages. Any process may make use of multiple ports to receive messages, but a process cannot share ports with other processes on the same computer. (Processes using IP multicast are an exception in that they do share ports) However, any number of processes may send messages to the same port. Each socket is associated with a particular protocol – either UDP or TCP.

1. **What is client server communication?**

This is the architecture that is most often cited when distributed systems are discussed. It is historically the most important and remains the most widely employed.



In particular, client processes interact with individual server processes in potentially separate host computers in order to access the shared resources that they manage. Servers may in turn be clients of other servers, as the figure indicates. For example, a web server is often a client of a local file server that manages the files in which the web pages are stored. Web servers and most other Internet services are clients of the DNS service, which translates Internet domain names to network addresses. Another web-related example concerns *search engines*, which enable users to look up summaries of information available on web pages at sites throughout the Internet. These summaries are made by programs called *web crawlers*, which run in the background at a search engine site using HTTP requests to access web servers throughout the Internet. Thus, a search engine is both a server and a client: it responds to queries from browser clients and it runs web crawlers that act as clients of other web servers. In this example, the server tasks (responding to user queries) and the crawler tasks (making requests to other web servers) are entirely independent; there is little need to synchronize them and they may run concurrently. In fact, a typical search engine would normally include many concurrent threads of execution, some serving its clients and others running web crawlers.

1. **Which package is used in java for socket programming?**

The Java interface to TCP streams is provided in the classes ServerSocket and Socket:

* **ServerSocket:**

This class is intended for use by a server to create a socket at a server port for listening for connect requests from clients. Its accept method gets a connect request from the queue or, if the queue is empty, blocks until one arrives. The result of executing accept is an instance of Socket – a socket to use for communicating with the client.

* **Socket:**

This class is for use by a pair of processes with a connection. The client uses a constructor to create a socket, specifying the DNS hostname and port of a server. This constructor not only creates a socket associated with a local port but also connects it to the specified remote computer and port number. It can throw an UnknownHostException if the hostname is wrong or an IOException if an IO error occurs.

The Socket class provides the methods getInputStream and getOutputStream for accessing the two streams associated with a socket. The return types of these methods are InputStream and OutputStream, respectively – abstract classes that define methods for reading and writing bytes. The return values can be used as the arguments of constructors for suitable input and output streams. DataInputStream and DataOutputStream, allow binary representations of primitive data types to be read and written in a machine-independent manner.

1. **Can we do multicasting in case of socket programming? How?**

Yes we can do multicasting in case of socket programming. At the application programming level, IP multicast is available only via UDP. An application program performs multicasts by sending UDP datagrams with multicast addresses and ordinary port numbers. It can join a multicast group by making its socket join the group, enabling it to receive messages to the group. At the IP level, a computer belongs to a multicast group when one or more of its processes has sockets that belong to that group. When a multicast message arrives at a computer, copies are forwarded to all of the local sockets that have joined the specified multicast address and are bound to the specified port number.

1. **Which are java API’s used to support multicasting in socket?**

The Java API provides a datagram interface to IP multicast through the class *MulticastSocket*, which is a subclass of *DatagramSocket* with the additional capability of being able to join multicast groups. The class *MulticastSocket* provides two alternative constructors, allowing sockets to be created to use either a specified local port or any free local port. A process can join a multicast group with a given multicast address by invoking the *joinGroup* method of its multicast socket. Effectively, the socket joins a multicast group at a given port and it will receive datagrams sent by processes on other computers to that group at that port. A process can leave a specified group by invoking the *leaveGroup* method of its multicast socket.

The arguments to the *main* method specify a message to be multicast and the multicast address of a group (for example, *"228.5.6.7").* After joining that multicast group, the process makes an instance of *DatagramPacket* containing the message and sends it through its multicast socket to the multicast group address at port 6789. After that, it attempts to receive three multicast messages from its peers via its socket, which also belongs to the group on the same port. When several instances of this program are run simultaneously on different computers, all of them join the same group, and each of them should receive its own message and the messages from those that joined after it.

The Java API allows the TTL to be set for a multicast socket by means of the *setTimeToLive* method. The default is 1, allowing the multicast to propagate only on the local network. An application implemented over IP multicast may use more than one port. For example, the MultiTalk [mbone] application, which allows groups of users to hold textbased conversations, has one port for sending and receiving data and another for exchanging control data.