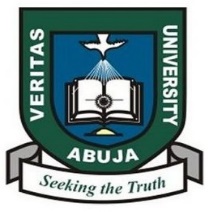
**VERITAS UNIVERSITY, ABUJA**

**FACULTY OF NATURAL AND APPLIED SCIENCES**

**Computer And Information Technology Department**

Mr Thomas Enesi Baidoo.

Email: [baidoot@veritas.edu.ng](mailto:baidoot@veritas.edu.ng).

Course: Distributed Systems.

**Module DFS\_B.**

**DISTRIBUTED SYSTEM**

# What is a Distributed System?

A distributed System is a collection of autonomous computer systems that are physically separated but are connected by a centralized computer network that is equipped with distributed system software. The autonomous computers will communicate among each system by sharing resources and files and performing the tasks assigned to them.

**Types of Distributed Systems:**

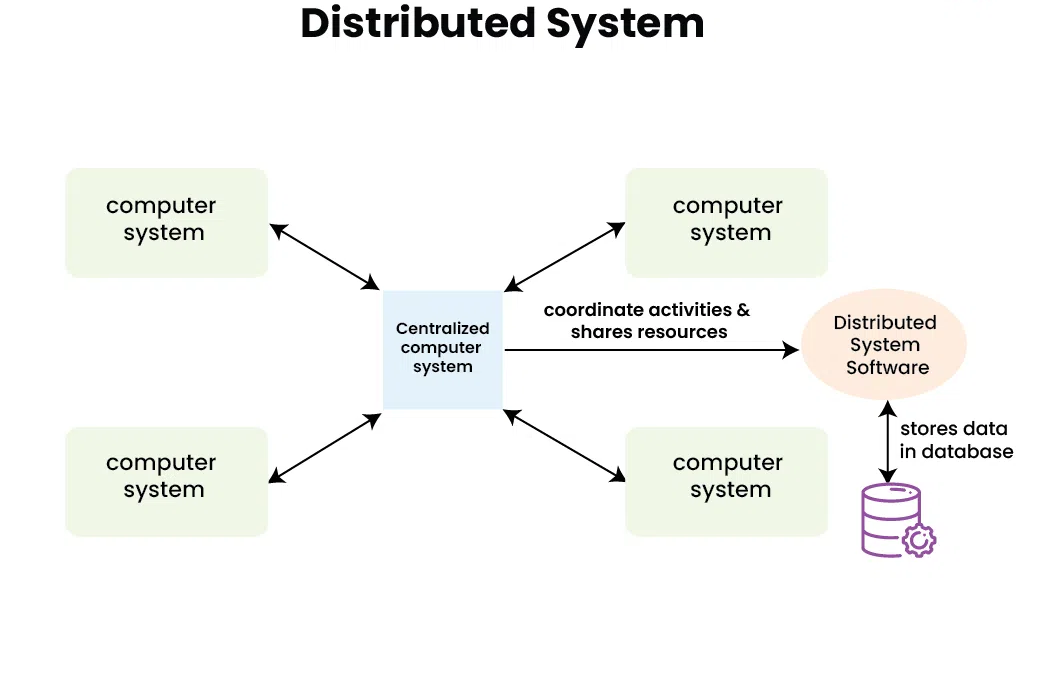
There are many models and architectures of distributed systems in use today.

* **Client-server systems:** The most traditional and simple type of distributed system, involves a multitude of networked computers that interact with a central server for data storage, processing, or other common goal.
* **Peer-to-peer networks:** They distribute workloads among hundreds or thousands of computers all running the same software.
* **Cell phone networks:** It is an advanced distributed system, sharing workloads among handsets, switching systems, and internet-based devices.

“*The most common forms of distributed systems today operate over the internet, handing off workloads to dozens of cloud-based virtual server instances that are created as needed, and then terminated when the task is complete.”*

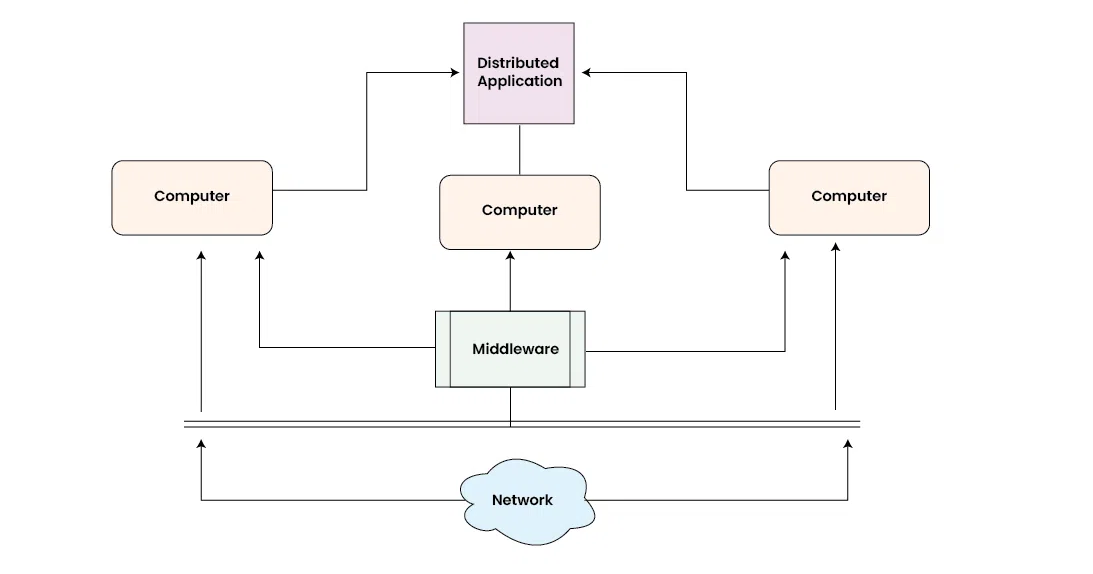
**Example of a Distributed System**

Any social media can have its Centralized Computer Network as its Headquarters and computer systems that can be accessed by any user and using their services will be the Autonomous Systems in the Distributed System Architecture.



* **Distributed System Software:** This Software enables computers to coordinate their activities and to share the resources such as Hardware, Software, Data, etc.
* **Database:** It is used to store the processed data that are processed by each Node/System of the Distributed systems that are connected to the Centralized network

**Working of Distributed System**

****

* As we can see that each Autonomous System has a common Application that can have its own data that is shared by the Centralized Database System.
* To Transfer the Data to Autonomous Systems, Centralized System should be having a Middleware Service and should be connected to a Network.
* Middleware Services enable some services which are not present in the local systems or centralized system default by acting as an interface between the Centralized System and the local systems. By using components of Middleware Services systems communicate and manage data.
* The Data which is been transferred through the database will be divided into segments or modules and shared with Autonomous systems for processing.
* The Data will be processed and then will be transferred to the Centralized system through the network and will be stored in the database.

**Characteristics of Distributed System**

* **Resource Sharing:** It is the ability to use any Hardware, Software, or Data anywhere in the System.
* **Openness:** It is concerned with Extensions and improvements in the system (i.e., How openly the software is developed and shared with others)
* **Concurrency:** It is naturally present in Distributed Systems, that deal with the same activity or functionality that can be performed by separate users who are in remote locations. Every local system has its independent Operating Systems and Resources.
* **Scalability:** It increases the scale of the system as several processors communicate with more users by accommodating to improve the responsiveness of the system.
* **Fault tolerance:** It cares about the reliability of the system if there is a failure in Hardware or Software, the system continues to operate properly without degrading the performance the system.
* **Transparency:** It hides the complexity of the Distributed Systems to the Users and Application programs as there should be privacy in every system.
* **Heterogeneity:** Networks, computer hardware, operating systems, programming languages, and developer implementations can all vary and differ among dispersed system components

**Advantages of Distributed System**

* Applications in Distributed Systems are Inherently Distributed Applications.
* Information in Distributed Systems is shared among geographically distributed users.
* Resource Sharing (Autonomous systems can share resources from remote locations).
* It has a better price performance ratio and flexibility.
* It has shorter response time and higher throughput.
* It has higher reliability and availability against component failure.
* It has extensibility so that systems can be extended in more remote locations and incremental growth.

**Disadvantages of Distributed System**

* Relevant Software for Distributed systems does not exist currently.
* Security possesses a problem due to easy access to data as the resources are shared to multiple systems.
* Networking Saturation may cause a hurdle in data transfer i.e., if there is a lag in the network then the user will face a problem accessing data.
* In comparison to a single user system, the database associated with distributed systems is much more complex and challenging to manage.
* If every node in a distributed system tries to send data at once, the network may become overloaded.

**Use cases of Distributed System**

* **Finance and Commerce:** Amazon, eBay, Online Banking, E-Commerce websites.
* **Information Society:** Search Engines, Wikipedia, Social Networking, Cloud Computing.
* **Cloud Technologies:** AWS, Salesforce, Microsoft Azure, SAP.
* **Entertainment**: Online Gaming, Music, YouTube.
* **Healthcare:** Online patient records, Health Informatics.
* **Education:** E-learning.
* **Transport and logistics:** GPS, Google Maps.
* **Environment Management:** Sensor technologies.

**Challenges of Distributed Systems:**

While distributed systems offer many advantages, they also present some challenges that must be addressed. These challenges include:

**Network latency:** The communication network in a distributed system can introduce latency, which can affect the performance of the system.

**Distributed coordination:** Distributed systems require coordination among the nodes, which can be challenging due to the distributed nature of the system.

**Security:** Distributed systems are more vulnerable to security threats than centralized systems due to the distributed nature of the system.

**Data consistency:** Maintaining data consistency across multiple nodes in a distributed system can be challenging.

Features of Distributed Operating Systems

There are various important goals that must be met to build a distributed system worth the effort. A distributed system should easily connect users to resources, it should hide the fact that resources are distributed across a network, must be open, and must be scalable.

**Connecting Users and Resources:**

The main goal of a distributed system is to make it easy for users to access remote resources, and to share them with other users in a controlled manner. Resources can be virtually anything, typical examples of resources are printers, storage facilities, data, files, web pages, and networks. There are many reasons for sharing resources. One reason is economics.

**Transparency:**

An important goal of a distributed system is to hide the fact that its process and resources are physically distributed across multiple computers. A distributed system that can present itself to users and applications such that it is only a single computer system is called transparent.

**Different Forms of Transparency –**

* **Openness:**

Another important goal of distributed systems is openness. An open distributed system is a system that offers services in standards that describable the syntax and semantics of those service instances, standard rules in computer networks control the format, content, and meaning of messages sent and received. Such rules are formalized in the protocols. In distributed systems, services are typically specified through interfaces, often called interface definition languages (IDL). Interface definitions written in IDL almost always capture only the syntax of services. They accurately specify the names of functions that are available with the types of parameters, return values, possible exceptions that can be raised and so on.

* **Scalability:**

The uncertain trend in distributed systems is towards larger systems. This observation has implications for distributed file system design. Algorithms that work well for systems with 100 machines can work for systems with 1000 machines and none for systems with 10, 000 machines. for starters, the centralized algorithm does not scale well. If opening a file requires contacting a single centralized server to record the fact that the file is open, then the server will eventually become a bottleneck as the system grows.

* **Reliability:**

The main goal of building distributed systems was to make them more reliable than single processor systems. The idea is that if some machine goes down, some other machine gets used to it. In other words, theoretically the reliability of the overall system can be a Boolean OR of the component reliability. For example, with four file servers, each with a 0.95 chance of being up at any instant, the probability of all four being down simultaneously is 0.000006, so the probability of at least one being available is (1-0.000006) = 0.999994, far better than any individual server.

* **Performance:**

Building a transparent, flexible, reliable distributed system is useless if it is slow like molasses. Application on a distributed system, it should not deteriorate better than running some application on a single processor. Various performance metrics can be used. Response time is one, but so are throughput, system utilization, and amount of network capacity consumed. Furthermore, the results of any benchmark are often highly dependent on the nature of the benchmark. A benchmark involves many independent highly CPU-bound computations which give radically different results than a benchmark that consists of scanning a single large file for same pattern.

# Communication in Distributed Systems

**Features of Good Message Passing in Distributed System:**

Message passing is the interaction of exchanging messages between at least two processors. The cycle which is sending the message to one more process is known as the sender and the process which is getting the message is known as the receiver.

In a message-passing system, we can send the message by utilizing send function and we can receive the message by utilizing receive function. Following are the general syntaxes for send function and receive function.

Send ()

Receive ()

Send (receiver, message)

Receive (sender, message)

Message passing is possible at whatever point the processors are in communication. The communication of a message can be established in distributed in two ways.

* Inter-process communication (IPC).
* Remote methodology call (RPC).

**Inter-process communication**: Inter-process communication is a process of trading information between two independent processes in a distributed environment. Inter-process communication can be achieved using two strategies or approaches.

* Original sharing (or) shared data approach.
* Copy sharing (or) Message passing approach.

**Remote Procedure Call:** A remote procedure call is a strong procedure for building distributed client server-based applications.

* In RPC there is a compelling reason need to change every one of the processors in an equivalent memory area or address space.
* RPC is utilized fully to communicate the processors at whatever point they are in the various memory area. In the same system or in a different system in the distributed system.
* RPC is generally helpful for developing client server-based applications.
* The process which needs to access the service is known as the client cycle or “caller”. The process which provides the services is known as the server process, or “callee”.

**Features of Good Message Passing System:**

Following are some of the features of a good message passing System.

**Simplicity:**  A message-passing system should be straightforward, simple, and easy to use. It should be straightforward to build the applications and to speak with existing applications &new applications by utilizing primitives provided by the message passing system. It should also be feasible for a developer to assign various modules of the distributed application and to send and receive the message between them in a way as simple as possible without the need to stress over the system and are network aspects.

**Uniform Semantics:** In a distributed system the message can be passed in two ways.

* Local communication
* Remote communication

Whenever we are utilizing the remote procedure call system, we ought to utilize 2 semantics one is at:

* Client machine
* Server machine both the semantics ought to be like get good communication among the process.

**Efficiency:** Efficiency is a critical task in the distributed message-passing system. If the event that the passing system is inter-process communication. The whole message passing system is collapsed.

**Reliability:** The message passing system ought to be reliable. This feature can be accomplished when the message passing system is following the right convention. Generally, the communication link failures may interrupt the communication cycle. A reliable IPC protocol can manage failure issues and guarantee of delivery of a message.

**Correctness:** A message-passing system ought to confirm the clients regardless of whether they are the right clients. If the right client isn’t found delivery of the message will become celled and again send the status to the sender.

**Flexibility:** The message passing system should be flexible for the different services like general SMS, file transfer, audio, and video sharing, and so on.

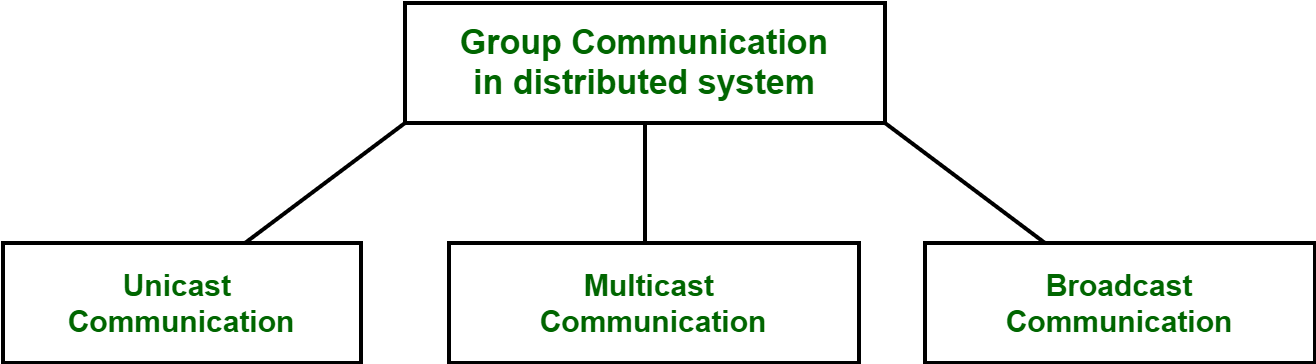
**Security:** A good message passing system must also be capable to provide secure end-to-end communication. to provide the security we ought to utilize the following:

* Authentication system
* Encryption system
* Decryption system

**Portability:** The message passing system ought to be portable itself.

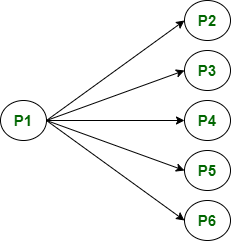
**Group Communication in distributed Systems.**

Communication between two processes in a distributed system is required to exchange various data, such as code or a file, between the processes. When one source process tries to communicate with multiple processes at once, it is called Group Communication. A group is a collection of interconnected processes with abstraction. This abstraction is to hide the message passing so that the communication looks like a normal procedure call. Group communication also helps the processes from different hosts to work together and perform operations in a synchronized manner, therefore increasing the overall performance of the system.

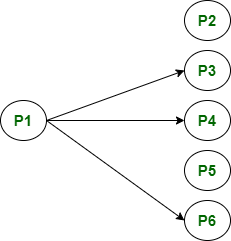


**Types of Group Communication in a Distributed System:**

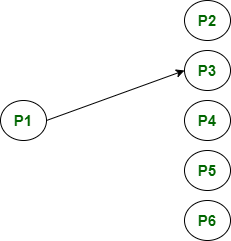
**Broadcast Communication:**When the host process tries to communicate with every process in a distributed system at same time. Broadcast communication comes in handy when a common stream of information is to be delivered to each process in most efficient manner possible. Since it does not require any processing whatsoever, communication is very fast in comparison to other modes of communication. However, it does not support many processes and cannot treat a specific process individually.



**Multicast Communication:** When the host process tries to communicate with a designated group of processes in a distributed system at the same time. This technique is mainly used to find a way to address problem of a high workload on host system and redundant information from process in system. Multitasking can significantly decrease time taken for message handling.



**Unicast Communication:** When the host process tries to communicate with a single process in a distributed system at the same time. Although, same information may be passed to multiple processes. This works best for two processes communicating as only it must treat a specific process only. However, it leads to overheads as it must find exact process and then exchange information/data.



# Remote Procedure Call in Distributed System

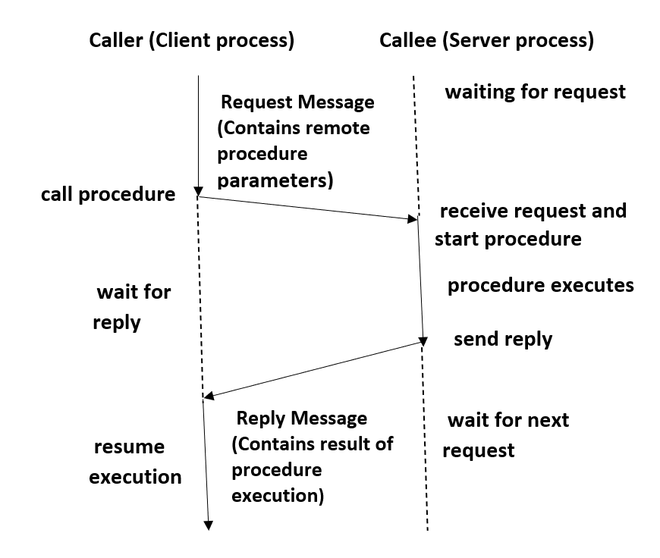
**What is RPC Mechanism in Distributed System?**

Remote Procedure Call (RPC) is a communication technology that is used by one program to make a request to another program for utilizing its service on a network without even knowing the network’s details. A function call or a subroutine call are other terms for a procedure call.

It is based on the client-server concept. The client is the program that makes the request, and the server is the program that gives the service. An RPC, like a local procedure call, is based on the synchronous operation that requires the requesting application to be stopped until the remote process returns its results. Multiple RPCs can be executed concurrently by utilizing lightweight processes or threads that share the same address space. Remote Procedure Call program as often as possible utilizes the Interface Definition Language (IDL), a determination language for describing a computer program component’s Application Programming Interface (API). In this circumstance, IDL acts as an interface between machines at either end of the connection, which may be running different operating systems and programming languages.

**Working Procedure for RPC Model:**

* The process arguments are placed in a precise location by the caller when the procedure needs to be called.
* Control at that point passed to the body of the method, which is having a series of instructions.
* The procedure body is run in a recently created execution environment that has duplicates of the calling instruction’s arguments.
* At the end, after the completion of the operation, the calling point gets back the control, which returns a result.
  + The call to a procedure is possible only for those procedures that are not within the caller’s address space because both processes (caller and callee) have distinct address space and the access is restricted to the caller’s environment’s data and variables from the remote procedure.
  + The caller and callee processes in the RPC communicate to exchange information via the message-passing scheme.
  + The first task from the server-side is to extract the procedure’s parameters when a request message arrives, then the result, send a reply message, and finally wait for the next call message.
  + Only one process is enabled at a certain point in time.
  + The caller is not always required to be blocked.
  + The asynchronous mechanism could be employed in the RPC that permits the client to work even if the server has not responded yet.
  + To handle incoming requests, the server might create a thread that frees the server for handling consequent requests.



**Types of RPC:**

**Callback RPC**: In a Callback RPC, a P2P (Peer-to-Peer) paradigm opts between participating processes. In this way, a process provides both client and server functions which are quite helpful. Callback RPC’s features include:

* The problems encountered with interactive applications that are handled remotely.
* It provides a server for clients to use.
* Due to the callback mechanism, the client process is delayed.
* Deadlocks need to be managed in callbacks.
* It promotes a Peer-to-Peer (P2P) paradigm among the processes involved.

**RPC for Broadcast:** A client’s request that is broadcast all through the network and handled by all servers that possess the method for handling that request is known as a broadcast RPC. Broadcast RPC’s features include:

* You have an option of selecting whether the client’s request message ought to be broadcast.
* It also gives you the option of declaring broadcast ports.
* It helps in diminishing physical network load.

**Batch-mode RPC:** Batch-mode RPC enables the client to line and separate RPC inquiries in a transmission buffer before sending them to the server in a single batch over the network. Batch-mode RPC’s features include.

* It diminishes the overhead of requesting the server by sending them all at once using the network.
* It is used for applications that require low call rates.
* It necessitates the use of a reliable transmission protocol.

**Local Procedure Call Vs Remote Procedure Call:**

* Remote Procedure Calls have disjoint address space i.e. different address space, unlike Local Procedure Calls.
* Remote Procedure Calls are more prone to failures due to possible processor failure or communication issues of a network than Local Procedure Calls.
* Because of the communication network, remote procedure calls take longer than local procedure calls.

**Advantages of Remote Procedure Calls:**

* The technique of using procedure calls in RPC permits high-level languages to provide communication between clients and servers.
* This method is like a local procedure call but with the difference that the called procedure is executed on another process and a different computer.
* The thread-oriented model is also supported by RPC in addition to the process model.
* The RPC mechanism is employed to conceal the core message passing method.
* The amount of time and effort required to rewrite and develop the code is minimal.
* The distributed and local environments can both benefit from remote procedure calls.
* To increase performance, it omits several of the protocol layers.
* Abstraction is provided via RPC.  To exemplify, the user is not known about the nature of message-passing in network communication.
* RPC empowers the utilization of applications in a distributed environment.

**Disadvantages of Remote Procedure Calls:**

* In Remote Procedure Calls parameters are only passed by values as pointer values are not allowed.
* It involves a communication system with another machine and another process, so this mechanism is extremely prone to failure.
* The RPC concept can be implemented in a variety of ways, hence there is no standard.
* Due to the interaction-based nature, there is no flexibility for hardware architecture in RPC.
* Due to a remote procedure call, the process’s cost has increased.

**Communication Protocols for RPCs**

The concept of Communication protocols for Remote Procedure Calls (RPCs) in Distributed Systems in detail.

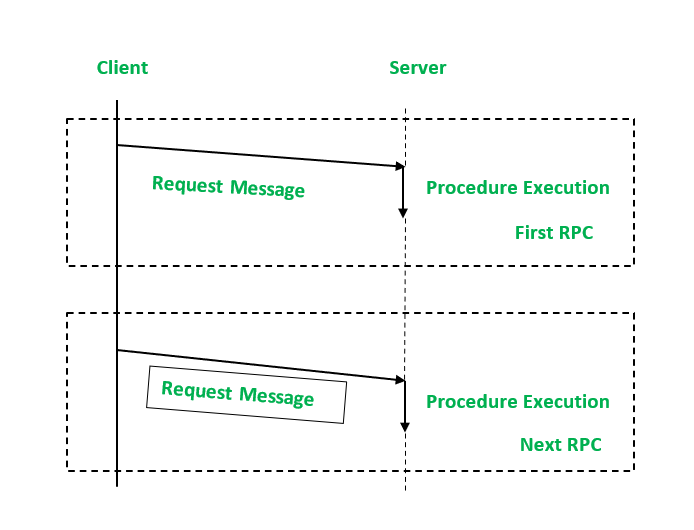
**Communication Protocols for Remote Procedure Calls:**

The following are the communication protocols that are used:

* Request Protocol
* Request/Reply Protocol
* The Request/Reply/Acknowledgement-Reply Protocol

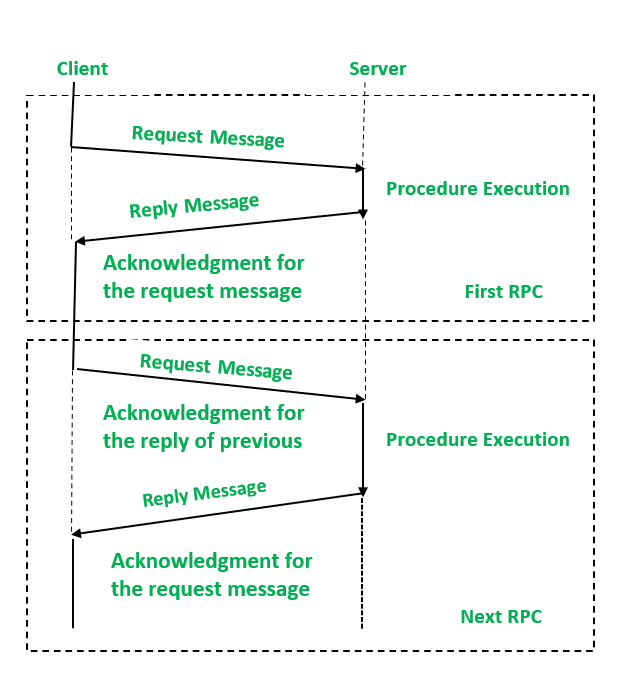
**Request Protocol:**

* The Request Protocol is also known as the R protocol.
* It is used in Remote Procedure Call (RPC) when a request is made from the calling procedure to the called procedure. After execution of the request, a called procedure has nothing to return and there is no confirmation required of the execution of a procedure.
* Because there is no acknowledgement or reply message, only one message is sent from client to server.
* A reply is not required so after sending the request message the client can further proceed with the next request.
* May-be call semantics are provided by this protocol, which eliminates the requirement for retransmission of request packets.
* Asynchronous Remote Procedure Call (RPC) employs the R protocol for enhancing the combined performance of the client and server. By using this protocol, the client need not wait for a reply from the server and the server does not need to send that.
* In an Asynchronous Remote Procedure Call (RPC) in case communication fails, the RPC Runtime does not retry the request. TCP is a better option than UDP since it does not require retransmission and is connection oriented.
* In most cases, asynchronous RPC with an unstable transport protocol is utilized to implement periodic update services. One of its applications is the Distributed System Window.



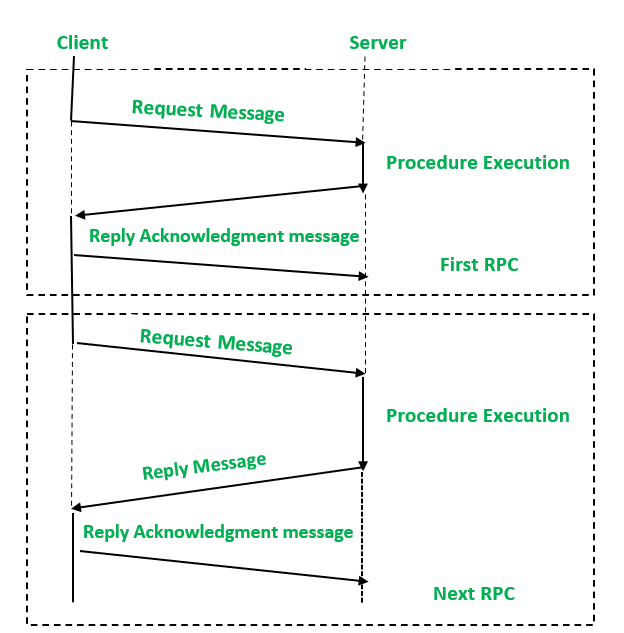
**Request/Reply Protocol:**

* The Request-Reply Protocol is also known as the RR protocol.
* It works well for systems that involve simple RPCs.
* The parameters and result values are enclosed in a single packet buffer in simple RPCs. The duration of the call and the time between calls are both briefs.
* This protocol has a concept base of using implicit acknowledgements instead of explicit acknowledgements.
* Here, a reply from the server is treated as the acknowledgement (ACK) for the client’s request message, and a client’s following call is considered as an acknowledgement (ACK) of the server’s reply message to the previous call made by the client.
* To deal with failure handling e.g. lost messages, the timeout transmission technique is used with RR protocol.
* If a client does not get a response message within the predetermined timeout period, it retransmits the request message.
* Exactly once semantics is provided by servers as responses get held in reply cache that helps in filtering the duplicated request messages and reply messages are retransmitted without processing the request again.
* If there is no mechanism for filtering duplicate messages, then at least-call semantics is used by RR protocol in combination with timeout transmission.

****

**The Request/Reply/Acknowledgement-Reply Protocol:**

* This protocol is also known as the RRA protocol (request/reply/acknowledge-reply).
* Exactly once semantics is provided by RR protocol which refers to the responses getting held in reply cache of servers resulting in loss of replies that have not been delivered.
* The RRA (Request/Reply/Acknowledgement-Reply) Protocol is used to get rid of the drawbacks of the RR (Request/Reply) Protocol.
* In this protocol, the client acknowledges the receiving of reply to messages and when the server gets back the acknowledgement from the client then only deletes the information from its cache.
* Because the reply to acknowledgement message may be lost at times, the RRA protocol requires unique ordered message identities. This keeps track of the acknowledgement series that has been sent.

****

**Complicated RPCs**

* RPCs that involve long-duration calls or large gaps between calls.
* RPCs that involve parameters(arguments) and/or result in values that are too large to fit in a single datagram packet.

**RPCs that involve long-duration calls or large gaps between calls:**

* The client probes the server regularly: After the submission of a request message from a client to the server, the client continuously sends a probe packet which a server needs to acknowledge. The exception status is communicated to the corresponding user if a communication failure occurs. Each probe packet contains the message identifier from the initial request message.
* The server generates an acknowledgement regularly: If the generation of the next packet by the server gets delayed then the predicted retransmission time interval, then it generates an acknowledgement on its own. Hence, during a long-duration call, many acknowledgements may be generated from the server as several acknowledgements directly correspond to the call duration. If within the set interval of time the response or acknowledgement from the server has not been received by the client, then it concludes that either server has crashed, or the failure occurs on the client-side or in case of communication failure user is alerted about the exception condition.

**RPCs that involve parameters/arguments and/or result in values that are too large to fit in a single datagram packet:**

* RPCs with Long Messages: To handle such an RPC, employ many physical RPCs for a single logical RPC. The sending of data in each physical RPC is made in the size of a single datagram packet. This technique is inefficient since each RPC incurs a set amount of overhead regardless of the quantity of data transmitted.
* Multi-datagram Messages: Multi-datagram messages are another approach for dealing with sophisticated RPCs in this category. It involves the division of long RPC parameters(arguments) or result into many packets and then sent in multiples. All the packets in a multi datagram message utilize a single acknowledgement packet for enhancing communication performance.

**Stub Generation in Distributed System**

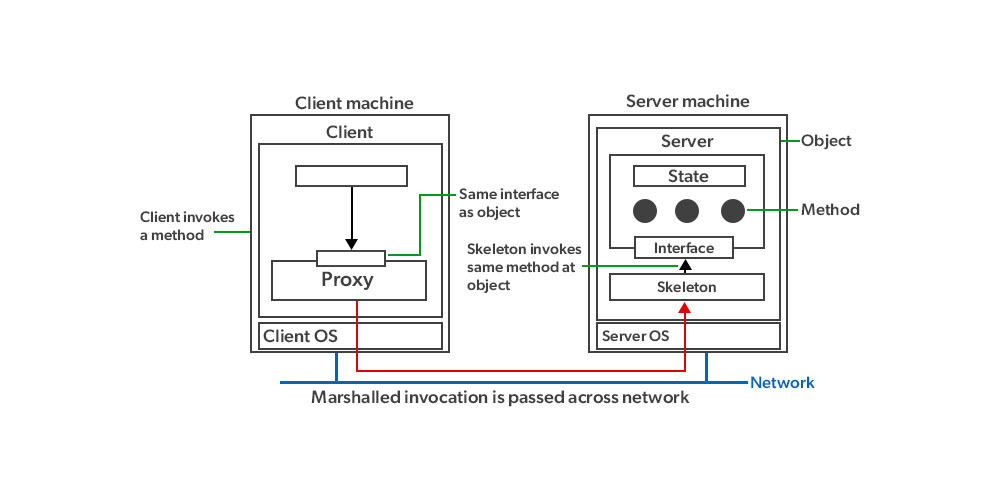
A stub is a piece of code that translates parameters sent between the client and server during a remote procedure call in distributed computing. An RPC’s main purpose is to allow a local computer (client) to call procedures on another computer remotely (server) because the client and server utilize distinct address spaces. That is the reason parameters used in a function (procedure) call must be translated; otherwise, the values of those parameters would be useless because pointers to parameters in one computer memory would point to different data on the other computer.

In most cases, stub libraries are installed on both the client and the server. Client stubs transform (marshalling) parameters used in function calls and then reconvert the result returned by the server when the function is completed. On the other side, client arguments are reconverted by server stubs, and results are converted back after function execution.

**Stub Generation:**

Stubs can be created in two different ways:

* Manual Generation of Stub: In the manual generation of stubs, the RPC implementer provides a collection of translation functions from which a user can create their own stubs using this way. This method is easy to use and can handle a wide range of argument types.
* Automatic Generation of Stub: In the automatic generation of stubs, client and server interfaces are defined using Interface Definition Language (IDL).  An interface specification, for example, contains information indicating whether each argument is input, output, or both; only input arguments must be passed from client to server, while only output elements must be copied from server to client. This is the most popular way of generating stubs.

****

* Import an Interface: A client program that calls procedures from an interface is said to import the interface.
* Export an Interface: A server program that implements a procedure in an interface is said to export the interface.
* Interface Definition Language (IDL): When building a distributed application, a programmer first creates an interface definition in IDL, then writes the client program that imports the interface and the server program that exports it. An IDL compiler is used to process the interface definition and generate components that may be coupled with client and server programs without requiring any changes to existing compilers. The compiler generates the required marshalling and unmarshalling operations in each stub procedure and a header file that supports the data types in the interface specification from an interface for each procedure in the interface. Both the client and server programs contain the header file, and the client stub procedures are compiled and linked with the client program, while the server stub procedures are compiled and linked with the server program. An IDL compiler can be programmed to interpret interface definitions for a variety of languages, allowing clients and servers written in different languages to communicate via Remote Procedure Calls (RPC).

To achieve semantic transparency, designers used the concept of stubs to make RPC (Remote Procedure Call) look like LPC (Local Procedure Call). Stubs mask the real RPC implementation from the applications that interface with the underlying RPC system.