ABSTRACT
This paper describes the working and features of Jini technology with references from various papers. It is just an attempt to sum up the different views and discussions given by various professors and researchers in their papers. Jini technology is a service oriented architecture that describes a programming scenario which extends Java technology to enable the building of secure, distributed systems which consist of federations of ordered network services and clients. Jini technology can be used to build adaptive network systems that are scalable, evolvable and flexible as is the need in dynamic computing environments [1].

1. INTRODUCTION
Jini systems provide mechanisms for service construction, lookup, communication, and use in a distributed system. Examples of services include printers or disks and users of the system. The heart of the Jini system is a trio of protocols called discovery, join, and lookup. Discovery occurs when a service is looking for a lookup service with which to register. Join occurs when a service has located a lookup service and wishes to join it. Lookup occurs when a client or user needs to locate and invoke a service described by its interface type [2]. Jini network technology is basically a middleware that provides a set of application programming interfaces (APIs) as well as network protocols that can meet home network requirements. It establishes a software platform enabling all devices that form the network to talk to each other, irrespective of their operating systems or interface constraints. In a Jini environment, each device provides a service to other devices in the network. Each device publishes its own interfaces, which other devices can use to communicate with it and thereby access its particular service. This approach ensures compatibility and standardized access among all devices [3].

2. JINI ARCHITECTURE
The purpose of the Jini architecture is to merge groups of devices and software components into a single, dynamic distributed system. Jini connection technology consists of an infrastructure and a programming model which address the fundamental issues of how devices connect with each other to form a spontaneous community. The outline of the Jini architecture is shown in Figure 1.

Based on Java technology, Jini technology uses Java Remote Method Invocation protocols to move code around the network. Network services run on top of the Jini software architecture. Jini technology assumes a changing network, in terms of both the components that make up the network and the way these components interact. Networks are generally long-lived entities that, as they grow to include ever-larger populations of users and machines, become increasingly difficult to upgrade as a single entity. Jini architecture is designed around support for incrementally upgrading network components. Jini allows anything with a processor, some memory, and a network connection to offer services to other entities on the network. This class of devices includes all the things we traditionally think of as computers but also most of the things we think of as peripherals, such as printers, storage devices, and specialized hardware. In the near future, the definition will also encompass a host of other devices, such as cell phones, personal digital assistants, and microprocessor controlled devices, such as televisions and stereo components. Making the network central requires a design that allows updates and changes to individual components without the wholesale shutdown of the network. The Jini system allows upgrades and updates to be installed and used by the components being networked without requiring that the network be shut down or all individual components be updated. An additional result of building around the network is that the data and code running on any device in the network cannot be assumed by users or developers to have been built especially for that device. Indeed,
given the longevity of networks and the rapid rate of change in small devices, the code and the information used on a particular processor is often constructed or gathered long before the processor is designed or built. Users of a Jini-based network should be able to add or remove member components without having to update other member components in the network community. A final goal for Jini’s designers was imposed by the size of current networks and how rapidly they are growing. If we have all the embedded systems that could possibly be given access to a network as part of our system, Jini technology has to be able to scale to very high levels [4].

Jini technology provides a distributed environment for devices to communicate with each other. Each device provides a set of services to the network, creating a federation of services for the constituent devices. No central authority controls federation services. A Jini environment implements the connection technology below the network application layer, building on top of the Java platform. The technology accepts all sorts of devices, including electronic home appliances, musical instruments, and other devices that are not part of a conventional computer network. Each device can act as a client or server depending on whether it is requesting a service or providing one. A Java object represents each device. The object’s interfaces expose the services offered to the network. Accessing a particular service involves using the published interfaces to invoke a remote procedure on the appropriate device object. Jini uses Java’s remote method invocation for accessing a particular service. RMI, the Java equivalent of a remote procedure call, enables clients to obtain handles to the desired remote objects. Jini can also use RMI to pass objects as arguments and to return values, making it easy to move code as well as data across the network [3]. Jini connection technology is graphically shown below in Figure 2.

Some of the main features of the Jini architecture are described below.

2.1 LOOKUP

The lookup service can be viewed as a directory service, in that services are found and resolved through it. In a Jini community, services register their proxy objects with a lookup service through discovery and join process, and clients query the lookup service to find out the services they want. Jini uses three related discovery protocols, useful in different situation [3]. Multicast Request Protocol is used when an application or service first becomes active, and needs to find lookup services in the vicinity. It is used by lookup services to announce their presence to the services that may have interest in the community. Unicast Discovery Protocol is used to establish communications with a specific lookup service known to it over a wide-area network. But a Jini lookup service does much more than a simple name server. Client sees a service as an interface, including methods that the client will invoke to execute the service, along with any other descriptive attributes. The lookup service maps interfaces seen by clients to a set of service proxy objects. Client downloads the service proxy, which is actually an RMI stub that can communicate back with the server. This proxy object enables client to use the service without knowing anything about it. Although service proxy object is a typical scenario of service invocation, i.e. accessing services through RMI method invocation, the downloaded service object can be the service itself or a smart object capable of speaking any private communication protocol.

2.2 LEASING

Access to services in the Jini system is granted on a lease basis. A service is requested for a time period and then granted for a negotiated period between the service user and provider. This lease must be renewed before its expiration. Otherwise, the resources associated with the services are released. The lookup service grants lease to a service registration and the service should continue to renew the lease. A device can leave the community or fail abruptly without having a chance to deregister itself. There is a set of interfaces and classes that define the notion of leasing, developed specifically for problems in resource allocation and reclamation in distributed systems [4]. In Jini, service or event registrations must be leased. A lease is also time-restricted i.e. interest must be renewed periodically in order to remain valid. Lease times are determined by the lease grantor. Un-renewed leases are interpreted as expired registrations and must be handled consequentially. Leases provide a consistent picture of the system components when entities join and leave unpredictably. Jini deploys the notion of leasing to allow clients and services to leave easily without disrupting other members. Jini’s leasing model sets time limits that services are available to clients. A Jini member offering a resource does so through a lease, which represents a period of time which the service is available. As a result, a service usually periodically renews its lease with the Lookup service [5].

2.3 DISCOVERY

The discovery protocol, allows an entity wishing to join a Jini network to find a lookup service. A proxy is a local object that stands in for the remote object. While presenting the same programmatic interface to the local code, the proxy deals with any network-related functions, transmitting any parameters to the remote service and receiving any return values from that service. A service that wants to join a Jini federation sends out a packet,
multicast over the LAN to a well-known port, asking for any lookup service to respond. The packet might specify that only lookup services within a particular group respond, but in the simplest case, any and all lookup services on the local network would respond. The packet also contains the information necessary for any lookup service to respond to the requester. Upon receipt of such a request, a lookup service responds by sending the requester a local proxy to the lookup service. This proxy, when loaded into the Java virtual machine running on the requester, contains enough information that, if the code needed for the proxy is not present at the site of the requester, it can be downloaded over the network. Therefore, the proxy returned to the requester is always matched to the lookup service that sent it. Upon receipt of a proxy for a lookup service, a service can register to offer itself for use to other members of the Jini federation by placing a proxy object of its own in the lookup service. If the service has received a response to more than one lookup service, it can register itself in any or all of these services. Clients looking for a service find the lookup service by multicasting a discovery request. Upon receipt of a proxy from any available lookup service, a client can request a service. Such a request takes the form of asking for an object implementing a particular Java language type.

3. CONCLUSION
This paper discusses the JINI network technology and its related architectural features. The Jini approach provides a number of methods and tools for building distributed applications for decentralized and unprepared network environments.

4. REFERENCES
[1].http://www.jini.org/wiki/Main_Page
[8] Pollu Huang, Jini for Ubiquitous Devicesa