PEER-TO-PEER DIRECTORY SERVICE IN RESOURCE AREA NETWORK

A thesis presented to

the faculty of

the Russ College of Engineering and Technology of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

Chitra Nedunchelliyan

November 2007

This thesis titled

PEER-TO-PEER DIRECTORY SERVICE IN RESOURCE AREA NETWORK

by

CHITRA NEDUNCHELLIYAN

has been approved for

the School of Engineering and Computer Science

and the Russ College of Engineering and Technology by

Chang Liu

Assistant Professor of Electrical Engineering and Computer Science

---

Dennis Irwin

Dean, College of Engineering and Technology

ABSTRACT

NEDUNCHELLIYAN, CHITRA, M.S., November 2007, Computer Science

PEER-TO-PEER DIRECTORY SERVICE IN RESOURCE AREA NETWORK (129pp.)

Director of Thesis: Chang Liu

This research effort involves designing and implementing a resource directory (RD) for a Resource Area Network (RAN). The resource directory is an application that performs resource and peer management in a RAN. This Java-based implementation enables the RAN to support disparate resource types such as memory, files, file chunks, and web services as shared resources. The RD also allows for dynamic definition and modification of the resource types actively supported in the RAN. In addition, the search efficiency is enhanced by implementing a balanced binary tree algorithm within the peer network structure in the RAN. The algorithm reduces the height of the tree hence reducing the hop number for the search messages. The experiments conducted demonstrate RAN's ability to support and dynamically manage multiple resource types. The tests simulated on the sample networks demonstrate the reduction in hop number for search messages in RAN when compared with an unstructured network like Gnutella.

Approved: _____

Chang Liu

Assistant Professor of Electrical Engineering and Computer Science

ACKNOWLEDGMENTS

I would like to express my gratitude and many thanks to my advisor Dr. Liu for his patience and generous support for this research effort. In addition, I am thankful for his long hours in reviewing the numerous versions in developing this document.

I would also like to thank my committee members for their time and effort in supporting my work and most importantly for being part of my thesis committee. I would also like to thank my fellow associates for their contribution in this area of research.

I would like to thank my family and friends for their encouragement. A special thanks to my husband for being there. Last but not the least; I would like to express my gratitude to Tammy and Valerie for helping me through the graduation process.

Table of Contents

Page

List of Tables

List of Figures

List of Acronyms

RAN	Resource Area Network
RD	Resource Directory
P2P	Peer-to-Peer
RE	Resource Exporters
RI	Resource Importers
TTL	Time to Live
UUID	Universally Unique Identifier
UDDI	Universal Description, Discovery, and Integration
RDSS	RAN-based Distributed Storage System
API	Application Programming Interface

1. Introduction

This chapter includes a brief background of the evolution of the computer systems and its present day usage. The inception and the limitations of a centralized network structure are discussed. In addition, the final sections of this chapter introduce the need for RAN, RAN Resource directory and the scope of the project.

1.1. Evolution of Computer Systems

It is now more than half a century that the computers have served humanity while becoming an indispensable part of our everyday lives – not limited to just the technologist. Today, humans rely, knowingly or otherwise, on computers that are ever more powerful and faster utilizing a multitude of software technologies. Computers are a critical part of a wide variety of utilities that range from everyday appliances like a microwave oven to large nuclear power stations. According to [1], "in the year 2000 alone about 60% of US households owned at least one personal computer, that's about 168.4 million computers. About 99% of the public schools are equipped with Internet access according to the US Census bureau." The world has seen an explosive growth of the personal computers (PC) since the early days of IBM's innovation. The evolving microchip manufacturing technology has only assisted this growth, as it not only helped to make the computers smaller but also resulted in cheaper and affordable computers. This explosive growth of personal computers has spawned further research and developmental efforts that have led to affordable computers and computer paraphernalia.

Though today's computers have become much smaller and faster than what existed a few decades ago, there has been no proverbial 'technological leap' that would help today's technology to stay abreast with the growing demands. Rather, what can be

observed is indicative of improvisation of existing technologies to meet the short-term growth requirements. In other words, technology has come to a point where the growth has become less than linear when compared to the fast-paced growth that was witnessed some years ago. For example, in the distant past the scientific community was able to relish the birth of something as powerful and novel as Artificial Intelligence (AI). The advent of AI opened a completely new realm of machine-enabled computing to address what was previously considered possible only by the human brain.

1.2. Permeation of Computer Systems

'Technological leaps' do not happen overnight and it takes time to transition through the conceive-design-develop-deploy cycle. Additionally, user acceptance of the new technology happens over a period – especially, as the personal computers are in widespread use around the world, in contrast to the past. To meet today's needs, the developers work on improvising the existing technology to stay abreast with the users' demand. For example, the advent of computers that embraced the concepts like time slicing and parallel processing where hugely helpful in increasing the computing efficiency when the processors were much slower. One area where current technologies have not been able to meet the not-so-idealistic technological evolution is 'large unmonitored networks.'

Most of the theories and improvisations in place are well tuned for networks that are well planned and maintained. However, these implementations do not effectively handle large networks that comprise of several distributed, disparate resources – these resources could range from mainframes, personal computers to pocket PCs. Today's networks are mostly over-crowded due to large number of users; voluminous

communication between the different network resources; and most importantly, the increased ‘depth-of-use’ of computers to access the information available in the web.

1.3. Limitations of Centralized Networks

The explosive growth in the use of computer systems resulted in a need to create networks that allowed the sharing of information amongst the various computing systems. The most prevalent type of network topology implemented is the centralized network. In this type of network topology, there is one central node and several client nodes. The central node performs all the processing in the network while the client nodes interact with the central node for all its operations. Some of the key benefits of this network topology are increased security and ease of data and system management. However, some limitations are inherent due to the very nature of this network structure. These limitations are as follows:

- Single point of failure
- Expensive servers
- Scalability

As the network size and density grew, the centralized network topology approach resulted in constraints such as efficiency and scalability. It is not possible to overcome these issues by increasing the speed of the systems or by going for high performance systems. It may be possible to address these issues on large networks by improving the efficiency of the network components, and/or through effective communication between the network resources, and/or through efficient management of network resources.

1.4. Peer-to-Peer Networks

In large centralized networks, the maintenance of the network information like a list of available network resources, gateways, resources' current state etc, overwhelm the central server leaving little room for actual data processing or computation among the network resources. This limitation has paved way to the concept of 'resource sharing' among systems but with limited knowledge of the entire network. This approach ensures that the network systems are not overloaded with duplicative data about the all network components. The advent of peer-to-peer systems has become an inviting alternative to the crowded centralized networks. The concept of systems with limited and localized information about the network seemed to prove useful in many situations. The efficiency of the networks improved drastically as the system resources where spent on the computation and not on the processing of meta-data about the resident network. Thus, the problems of the overcrowded networks were circumvented to a considerable extent by the peer-to-peer networks.

According to [10], "The peer-to-peer networks offer redistribution of popular content and massive redundancy of content that, in some cases, prove useful as there could be more than one resource for the required information." More importantly, content serving moves from the 'Center' to the 'Edge' systems thus reducing the traffic to the central system. Likewise, in peer-to-peer computing, the computing resources move from centralized, capital-intensive, high-maintenance systems, to edge systems owned and maintained by the common user.

Today's Internet, as perceived by many, is an aggregation of millions of computers that communicate freely over the wires. One reason for this explosive growth is the fact

that the Internet has a large amount of useful information – to the extent where one may even consider the 'Internet' to be omniscient.

1.5. Limitations of Peer-to-Peer Systems

In this section, some of the limitations of the peer-to-peer systems are discussed.

1.5.1. Multiple Resource Types

In addition to the traditional resources like files and memory, the evolving trend in the networking world considers components like programming components (webservices), CPU time, and file chunks as resources. In most peer-to-peer systems, only a specific resource type is shared. For example, in Napster only files are considered as resources and shared across the network. However, in SETI@Home, processing time and bandwidth are shared as resources. Thus, the peers participating in such a network are confined to share only specific resource types as allowed by the peer-to-peer network. If a peer is allowed to share more than one type of resource in the same peer-to-peer network, this could drastically increase the shared content offered by the network and hence improve the productivity of the members.

1.5.2. Resource Type Modifiability

One of the peer-to-peer systems that shares multiple resource types, GRUB, explained in detail in section 9.5, shares processing time, bandwidth, and disk space as resources among the peers [36]. Each peer contributes to the 'web crawler' process that can be used to feed other search engines in the Internet. Though multiple resource types are shared in GRUB, the peers lack the flexibilty to share additional resource types. In addition, the peers will not be able to inhibit sharing a specific resource type that is

already shared. Thus all the peers contribute in performing only a single operation like web crawling or sharing files in the network.

To provide flexibilty to the operations that can be perfomed in the network, the network should be designed to allow the peers to modify the resource types that can be shared. Thus the peers can use its different resource types in more than one operation of the network like being able to share files and also perform web crawling. Such a network will consists of peers that participate in more than one operation of the network.

1.5.3. Need for Efficient Search

The key to the 'usefulness' of any 'knowledge holding' system is the ease with which the users can access the information contained within. Hence, the ability to find the information governs the usefulness of the actual data. In some of the unstructured peer-to-peer systems, like Gnutella, finding a resource involves massive distribution of the search query among the peers. The resoponse time for such a query is dependent on a number of factors that include the

- number of hops the query travels before it finds the resource
- number of query messages generated and distributed
- network density (number of connections per peer)

The search complexity in an unstructured peer-to-peer network can be $O(n)$ where n is the total number of peers. Any modification to the peer-to-peer implementation that would result in lowering the response time would increase the efficiency of the network.

1.5.4. Unpredictable Peer Life

In any peer-to-peer system, there is no control over the peer's life in the network. The peers can join or the leave the network at will. The ability of the network to handle such a

situation determines the fault tolerence of the network. In general, it would be desirable that the peer-to-peer network be designed to handle instances where a loss of any peer may not distrupt the network functions. New peers should be able to join the network or the existing peers should be able to resume its operations, even when designated superpeers or seed peers are offline.

1.6. Motivation for RAN and Resource Directory

Through the implementation of a Resource Area Network (RAN) [8] and its Resource Directory (RD), some of the limitations of the existing peer-to-peer systems can be overcome. The peer-to-peer system limitations related to multiple resource types, resource type modifiability and need for efficient search are addressed in RAN.

The focus of this thesis is to implement a RAN, a peer-to-peer network, and a Resource Directory that can

- Support multiple resource types as shared network resources
- Allow for dynamic modification of the resource types
- Reduce the number of hops for a search message

1.7. Project Scope

The scope of this research project is to develop a directory service for a RAN system that can use multiple resource types as shared resources. This directory service will allow for a dynamic and flexible management (addition or removal) of active resource types in the network. Additionally, this directory service will also include a functionality to ensure that the hop number for the search messages is reduced in the RAN by managing the network structure of the peers.

2. RAN - Resource Area Network

Advancements in network technology have improved to an extent where it is possible to achieve greater efficiency in speed while performing operations on the network than on a dedicated system. Significant cost reduction in implementing higher-end network systems has resulted in a situation where one can achieve network speeds that exceed the performance of stand-alone systems. When considering the systems prevalently utilized by the common user it may be efficient, in terms of speed, to read a data from the network rather than to read it from a secondary storage device. An experiment [2] conducted by the SLAC'ers proved that it is possible to send 6.7 billion bytes of data at 1 gigabit per second over 10,000 kilometers. If this trend continues, in the near future, one can expect even greater improvements in the arena of network technology. This advancement in the network speed is not limited to data transfer as files. This improved network speed could allow the use of the network to share 'non-traditional' resources like Memory, CPU time, Monitor, etc.

It is known that loading a file from a local secondary file system into memory takes a significant amount of time; this is described as file latency. The time delay caused by file latency is a limiting factor in the performance of any operating system though they may be connected to high-speed networks. The time taken to access files from any data source, usually measured in terms of milliseconds, translates to several million clock-cycles where the system is idle waiting for the data to be loaded onto the memory. When utilizing secondary storage devices, the file access time [3] could vary anywhere from 3 seconds to 5 minutes – this, mostly, is determined by the size and type of the file. With the current trends in network technology, network latency is highly reduced to an

insignificant amount when compared to the file latency induced due to the use of secondary storage devices in a lower end system. Hence, it is more efficient to have the files located on a higher-end system and have the lower-end systems connect to the files through today's modern network technologies. In doing so, the total latency in accessing the file from the perspective of the user on a lower-end system is highly reduced. Thus, based on the network speed, it may sometimes be more efficient to get the data from the network rather than from the local secondary file systems.

Several studies show that resources, like memory and processing time, can be shared among the computers that are connected to a high-speed network. RAN uses this concept and allows sharing of disparate resources such as memory, file, file-chunk, CPU time, web services etc. The main goal of the RAN Resource Directory development bed is to make the low level and intricate details of peer-to-peer application development transparent to the programmer. As RAN supports several different resource types, it stands out when compared to other resource sharing networks that handle a limited variety of resources. RAN system allows addition of new types of resources and provides an efficient way of sharing them among the peers in the network.

A classic example of one such shared network system is UC Berkeley's SETI @ Home [5] Project where one specific resource type – processing time for the project – is shared among the users connected to the Internet so that every user contributes some processing time for this project. The details of this project are further discussed in the later sections.

2.1. RAN Architecture

RAN Architecture comprises of three major types of components: resource exporters (REs), resource importers (RIs), and resource directories (RDs). The following figure shows the different types of peers that can be part of RAN system.

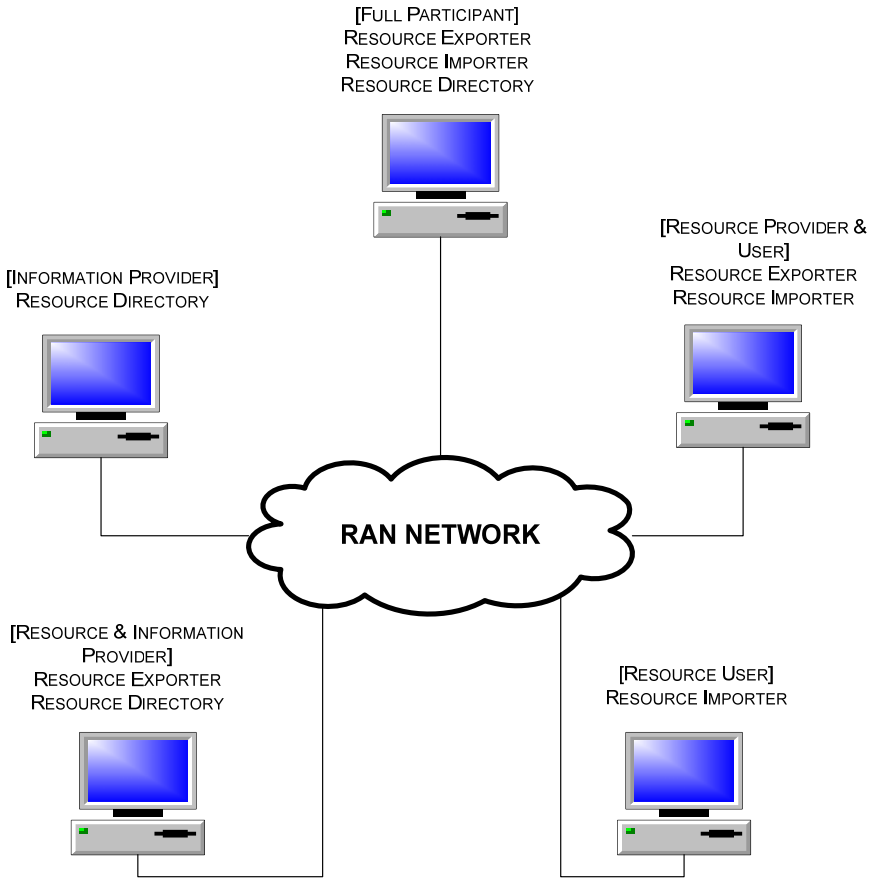


Figure 2.1.1 Resource Area Network

2.1.1. Resource Exporters:

One of the important advantages of any peer-to-peer system is its ability to share resources. The RAN, in particular, extends this ability to share resources not just in file format but also diverse resources like memory, processing time, etc. The key component

in RAN architecture that allows sharing of resources is the Resource Exporter. Resource exporters register local shared resources in the resource directories. Thus, the resources are made available to other peers, which are part of the RAN system. Once the resources are registered, they are available for use by any computer that is part of the RAN. The registration requires capturing all the data required for the resource to be available remotely.

The exporters can be stand-alone application that registers the existing resources into the system. They can accept requests to allow remote utilization of the shared resources. The REs also work as gatekeepers that decide if the received request can be serviced completely by allowing the external application to access the requested resource. When the resources' registration is complete in the resource directories, the peers can request the resource for their use. If the resource directory were able to locate a resource, it would return the information on how to access that resource to the requesting peer. The requesting peer then sends a request to access the resource to the associated resource exporter. The handshaking is done between the requesting peer and the resource exporter to determine how the resource needs to be made available to the peer.

2.1.2. Resource Directories

The peer-to-peer systems, as opposed to centralized systems, are atypical when it comes to the allowing registration of participants in the network. This is one of the advantages of the peer-to-peer networks as it is easy for any system to become part of peer-to-peer network and the overhead required for maintaining the network is less compared to centralized systems. No one node carries the information about the network in its entirety and hence it can be difficult to locate a resource in such an un-orderly

system if it is not maintained properly. The resource directories perform the function of a 'book-keeper' and help in maintaining the network information. They perform the grunt work of managing the information required to locate resources in the peer-to-peer systems. They store all the information required to locate the resource in the form of resource descriptors. They also answer queries about its stored resource descriptors. The peer-to-peer systems' core concept of limited information in any peer could be a limitation on the efficiency in locating resources. The RAN resource directory also exchanges resource descriptors between the resource directories connected in the RAN.

2.1.3. Resource Importers:

If a local application in a peer-to-peer system is in a need of RAN resources, the request is sent to the resource importer, which communicates with the RAN resource directory. The RAN resource directory provides the list of available resource descriptors for the requested resource. Then a handshake is performed between the resource importer and the respective resource exporter in the remote peer. Thus, the resource importer enables the local resources to be accessible by remote peers.

2.1.4. Multi-role Peers:

Most of the peers in the RAN fall into one of the three categories described above. Some of the peers take on a dual role. For example, some peers which the user interacts with, can act as both resource exporter submitting local resource for remote usage and also as resource importer consuming the available remote resources. Such peers are designated as 'Resource Provider and User'. RAN resource directory acts as an 'Information Provider' and it can be coupled with either the exporter, importer or both.

Resources, which as described earlier can be of any type such as file, file chunks, memory, etc., are registered in the form of resource descriptors. The current implementation of the RAN supports several different resources such as memory, file, file chunks and web-services. One of the most important constraints that govern the efficiency in a peer-to-peer system is the ease at which the peers communicate with each other and maintain the structure of the network so that more peers participate in the network. The network structure in the current RAN implementation is managed by using a balancing algorithm that connects the peers in the form of a binary tree structure. The details of this algorithm are covered in the later sections.

2.2. Resource Directory

A resource directory is one of the main part of the RAN infrastructure. The resource directory contains the list of all known and available resources in a RAN. In addition, the contents of this list are categorized based on specific resource types. The information about the disparate resources is contained in 'resource descriptors.' The resource directory, in addition to performing a registry-like service, also responds to queries from resource exporters. These functions are discussed in detail in the later sections.

Resource directory stores resource descriptors, answers queries about its stored resource descriptors, and allows registration of new resource descriptors. In addition, a resource directory would pass queries around to its nearby resource directories if the resources are not readily available in its system. When the resource directory obtains the results, the result is passed back to inquirer. To avoid infinite hops of queries among resource directories, each query carries a value that limits the number of hops.

A resource descriptor is a piece of structured information that describes in detail a particular resource, including resource category id, resource type id, resource id, resource access point, resource checksum, resource signature, and other resource-specific information.

2.2.1. Types of Resource Directories:

Resource directories, based on the source of the registered resources, can be categorized as a limited resource directory and an extended resource directory. Limited Resource Directory**:**

When a resource directory is established, it will listen for messages from one or more resource exporters to register the available resources – this is commonly referred to as the 'limited resource directory.' Limited resource directories are needed when a resource provider may wish to run a resource directory such that its resource exporter can register resources locally. This is especially the case when a resource provider decides against investing additional system resources required to run an extended resource directory.

Extended Resource Directory:

When a resource directory, in addition to registering resources from resource exporters, also registers resources from other nearby resource directories, it is referred to as an 'extended resource directory.' Extended resource directories contain all functionalities of limited resource directories and are further classified into two subtypes – a *passive resource directory* and an *active resource directory*.

A passive extended resource directory learns about resources registered in nearby resource directories through queries and corresponding query results that it passes around

to other peers. Whereas an active extended resource directory sends out original queries to nearby resource directories and actively seeks information on resources that were registered elsewhere. In addition, an active resource directory may seek resource information from outside the RAN. For example, in a RAN of software components, an active resource directory may query an UDDI (Universal, Description, Discovery and Integration) directory to find information regarding XML web services registered in the UDDI directory. All limited resource directories are passive resource directories.

3. Comparing Peer-to-Peer Systems

There is a variety of peer-to-peer systems available in the industry. In this chapter, the general characteristics of a variety of peer-to-peer systems are compared [7, 9, 11]. Some important attributes of peer-to-peer systems used for this comparison are architecture, network structure, efficiency and scalability.

3.1. Architecture:

Based on architecture, most peer-to-peer systems can be classified into two major types – Pure peer-to-peer and Hybrid peer-to-peer.

- **Pure peer-to-peer** – Pure peer-to-peer systems treat all peers as a node and lack a central router. The routing system is either a distributed catalog for the resources, which uses indices as parameters, or direct messaging systems, where the message is routed until the resource is located. Gnutella, Limewire [12], Bearshare [14] are some of the peer-to-peer systems that follow this type of architecture.

- **Hybrid peer-to-peer –** Hybrid peer-to-peer systems either support heterogeneous peers or have a centralized system. The centralized system will provide the catalog for the location of different peers. A classical example of this kind of peer-to-peer system is Napster.

- **RAN** Architecture – RAN follows a modified version of the hybrid architecture. The resource directory is a database that stores the peer and the resource catalogs. Every peer registers the resource to one or more resource directories enabling the peer to manage the resources that are available for sharing. Since the complete list of peers are

not saved in any of the peers, it helps in enhancing the security of the system. Though currently RAN does not support any authentication, this feature would be a great addition to the existing system.

3.1.1. Comparison

In this section, the various attributes of the peer-to-peer systems are compared.

3.1.1.1. Virtual Horizon

Pure/Hybrid peer-to-peer Systems: The TTL (Time To Live), which is the limit to the number of hops the messages can travel between the peers, creates a virtual network boundary for an individual peer and its messages. This effectively divides the network into subnets and hence limits the range of the messages.

RAN Systems: In RAN systems each resource includes the hopNumber or TTL as part of the metadata. This value can be adjusted based on the level of participation required for the search queries in the network. With this feature and by the inherent network structure used for the RAN system, a message may not often run into a situation where the message is not propagated further as the TTL is exceeded. Hence, RAN systems are not always limited by TTL.

3.1.1.2. Network Traffic

Pure/Hybrid peer-to-peer Systems: The way the peers are connected forces the messages to be transmitted to all the peers that are connected to it and hence each message is duplicated and dispersed throughout the network. This results in huge amount of network traffic and hence each message increases the load due to unwanted messages on the network.

RAN Systems: The network traffic can be reduced within the system, as the nodes are arranged in a binary tree structure and the messages are transmitted between the nodes traversing the tree structure. If the message is encoded with the value of the node where the message was transmitted from, while sending the message, the nodes can avoid sending it to the same node and hence stops flooding the network.

3.1.1.3. Network Security:

Pure/Hybrid peer-to-peer Systems: The metadata, data about the resources, of the available resources are stored in multiple peers as in pure peer-to-peer systems or in a database as in hybrid peer-to-peer systems. In systems where the metadata is stored in a database, the security of the system may be comprised. A new peer can join the network and obtain the metadata of the available resources. With this information, the new peer can turn hostile and sabotage all other peers in the network.

RAN Systems: If the authentication is developed within the RAN system, the chances of malicious attacks can be reduced. Though the list of nodes and their information are stored in the database, un-trusted peers always contact a trusted peer to join the network and if the authentication fails, none of the information will be shared to the new peer.

3.2. Network Structure:

Network structure is the protocol by which the peers are connected to each other in any network. It is a key factor to a peer-to-peer system's efficiency. The ease at which the data can be located in a network is related to the ease at which the peer, containing the data, can be located in the network. In some networks, there is a direct relation between the peer location in the network and the data's location whereas in other networks, the data's location is irrelevant to the peer's location in the network. Based on their network

structure, the peer-to-peer systems can be classified as unstructured or structured networks.

Unstructured Networks:

Generally, in a peer-to-peer system, the peers join the network by first communicating with a known or an existing peer, which is already part of the network. The selection of the peer can be either predefined or randomly chosen. This type of organizational protocol, where the peers join the network in a random order or in an order that is irrelevant to the data, is commonly known as unstructured networks.

In such a network, the location of the data is unrelated to the topology of the peers connected in the system. The data can be available in any of the peers and the peer location is invisible to the user who requests the data. The protocol dictates how each peer routes the request to the network. Mostly such communications are in the form of messages either broadcasted or back propagated. Broadcasted messages are forwarded to all the peers the sender is connected too. Back-propagated messages are routed through a specific path on the network that traces the reverse of the path taken by an original message.

RAN is an unstructured network, as the peer location in the network is not related to peer's resource content, where the peers connect to a known seed peer and the seed peer determines the new peer's location in the RAN system. The peers can be functionally different based on the system parameters. However, RAN uses a hybrid form of broadcasting where it does not forward query messages to the source path.

Structured Networks:

In a structured network, the location where the data is present is tightly coupled with the peer location in the network. Thus, the files are located precisely in specific location often dictated by the network overlay. An Example of such a network is the Tapestry [15] peer-to-peer network, which uses Plaxton algorithm [16]. Table 3.2.1 below presents the association of network structures and extent of centralization of some peer-to-peer networks.

	Un-Structured Networks	**Structured/Partly Networks**
Centralized/Partly Centralized	KaZaa, RAN	
De-Centralized	Gnutella	Freenet /Tapestry

Table 3.2.1 Structure vs. Centralization in peer-to-peer Networks

3.3. Network Peer Types

The peer-to-peer networks can be classified based on the 'peer-type' of the peers that participate in the network. The peers in a network can be considered as equals, as they may have functional parity, or they can be functionally different, that establishes a hierarchy amongst peers.

Equal Peers:

If all the participating peers are functionally the same and if the unavailability of a peer does not affect the network performance, then such peers are called equal peers. In networks such as Gnutella v0.4 [9], all the peers are considered equal and there is no distinction between them.

Hierarchical Peers:

If some of the participating peers perform more functions when compared to other peers then such a network is said to contain hierarchical peers. The hierarchically superior peers – super peers – have a higher availability and are powerful systems [18]. The availability of these peers affects the network performance. For example, in Kazaa [19], the peers can be of two types. One type has limited functionality known as the leaf peer. The other type known as super peers, are resource rich and have more functionality when compared to the leaf peers.

In RAN, the peers are considered equal but they can be functionally different based on the system parameter set during start up. A more detailed discussion on the peers-types in RAN is available in chapter 2.

3.4. Efficiency/Scalability:

The primordial premise of the peer-to-peer network is the ability of the peers to share the disparate resources thus moving away from the traditional client-server networks. The key to this setup is to have the peers function effectively as one system – efficiently communicating with one another. The ease at which the request for any data, stored in any of the peers, can be found and routed to the requester in a timely manner defines the network efficiency – this, to the most extent, is dependent on the search mechanism. For a given peer-to-peer network, being able to maintain the same level of efficiency regardless of the network size defines the scalability of the network. For example, in Gnutella, the system has a limitation towards high scalability due to the large volume of search messages generated and propagated throughout the network.

4. The Design of RAN Resource Directories

This chapter describes the resource directory parameters that are required during start up of any RAN system. These parameters determine the operational mode for the RAN system. Based on the operational mode selected during RAN start up, a whole suite of functional ASCII commands are available to perform the RAN related operations. Some of these commands are unique as their functionality may vary based on the RAN's start up mode. This chapter also describes the core database structures that are used for the RAN operations.

4.1. RAN System Parameters

The RAN resource directory uses a number of parameters that allow the system to be defined to operate under different functional modes. These parameters are initialized during RAN system startup – these are refered to as RAN system parameters.

This section provides a list of all RAN system parameters. Parameters specified within"[ ]" are optional and those specified within "< >" are required.

[-passive]

This parameter defines the RAN resource directory to be either an active or aassive directory. The default value for this parameter is to create an active RAN resource directory. This parameter can be changed to create a Passive RAN resource directory.

[-standalone]

RAN resource directory can either participate in the peer-to-peer network or operate in a standalone mode. This parameter, if set, creates a RAN resource directory which operates in a standalone mode.

[-peer-to-peer]

This parameter, if set, creates a RANresource directory which operates in a peer-to-peer mode. This responds to queries and adds resources and participates in all the peer-to-peer operations.

[<-mysql ipaddress -u username -p password>]

This is a required parameter that provides the IP address of the RAN-resource directory where the database is running, the user name and the password. The RAN resource directory uses MySQL database as its backbone and all the interaction to the database is through the resource directory interface.

Eg. [<-mysql 132.235.15.108 -u cs456_556 -p cs456_556>]

[<-seed ipaddress>]

This parameter sets the ipaddress of the seed node.

The seed node is a known node in any ran network. When a RAN node starts up, it requires a seed node to join the network. During the RAN node start up, the node communicates to this seed node and issues the request to join the network. It is the seed node that determines if the RAN node can be part of the RAN system.

[< -asciimsg portnumber>]

The RAN node also has the ability to process ascii messages. This parameter sets the port address on which the RAN node would be listening for ASCII messages from other nodes or other systems.

[-help]

This is an optional parameter that, when set, prints out all the acceptable RAN start up parameters.

4.2. RAN Database Structure Definition

The backbone of the RAN resource directory is a database that stores all the information about the resource types and the registered resources. The resource importer and the resource exporter communicate with resource directory which performs the required operation with the database. As part of setting up a RAN peer, these databases need to be initialized along with setting up the Resource Importer (RI) and the Resouce Exporter (RE). The database can be restricted to be accessed only by RI and RE, thus improving the security around the data in the peer. By restricting the direct access to the database, we would also restrict security breaches, sql injection or any other kind of database attack.

Creating Database and Basic Table Structure:

RAN resource directory is developed using MySQL as the database and it is assumed that the peers joining the RAN system also have MySQL installed in it with a valid username and password to connect to the database. The initialization for RAN resource directory starts with setting up the database. The package includes InitDB executable file that requires IP address, username and password for the database as parameters.

This executable creates a database ‘Resource’ and creates three other system tables provided in Table 4.2.1 below. Each of the tables schema and its functionality is described below.

Resource
resource_directories
resource_type_property
resource_types

Table 4.2.1 Resource System Tables

4.2.1. RESOURCE_DIRECTORIES

The table ‘RESOURCE_DIRECTORIES’ is one of the system tables that is created as part of the initialization of RAN resource directory. This table is used when the peer is operating in the peer-to-peer mode. It is used in maintaining the peers’ heirarchial structure in the RAN. The table maintains a list of other peers that are connected to the local peer. It also monitors information about each of the peers which determines its reliability in the network. Some of the basic information captured about other peers are its IP address and the port at which it can communicate with it. In RAN, the resource directory is considered as a resource and hence can be shared. When a peer joins a RAN system, a resource directory is created and self registers in the directory as a resource of type ‘Resource directory’ thus enabling the resource directory to be shared with other peers.

Some of the parameters of a resource directory, provided in Table 4.2.1.1 below, are ‘IP’ and ‘port’ of the peer-to-peer RAN resource directory. Additional parameters, like the ‘reputation’ and ‘hopNumber’, specified are used to determine the reliability of the data that is received from the other RAN resource directories. The parameters like ‘isNearby’ and ‘isWellknown’ can be used in the query process to indentify preferred RAN resource directories.

Field	Type	Null	Key	Default	Extra
Ip	varchar(4)	YES		NULL	
Port	int(11)	YES		NULL	
isWellKnown	int(11)	YES		NULL	
timeStamp_FirstStatusCheck	timestamp(14)	YES		NULL	
timeStamp_LastStatusCheck	timestamp(14)	YES		NULL	
Reputation	int(11)	YES		NULL	
isNearby	int(11)	YES		NULL	
hopNumber	int(11)	YES		NULL	

Table 4.2.1.1 Attributes of a Resource Directory

4.2.2. RESOURCE_TYPES

The table 'RESOURCE_TYPES' is used in managaing the disparate resource types that are registered by the resource exporter or by other applications in the RAN peers. The resources can be related or totally be unconnected. For example, the resource type 'file' can be stored as several 'file chunks' dispersed through out the network. Hence, each of the 'file chunk' can also be registered as a resource in RAN. RAN also provides the ability to track the relationship between the resources. This feature is used in one of the RAN application which is discussed in the later chapters.

Each resource type has an unique identifier in the form of UUID (Universally Unique Identifier). The UUID [23] is a 16 byte (128 bit) string that supports hexadecimal numbers. This UUID is used to provide a unique key, with high reliability, to all the peers in the peer-to-peer system without the possibility of duplication. In any network, which

has central supervision, the central server manages the responsibility of creating uniqueness among the network elements. In a peer-to-peer network, as all the peers operate independent of each other, such supervision cannot be implemented without much overhead. The use of UUID greatly reduces any chance of duplication. Table 4.2.2.1 provides the schema of the 'RESOURCE_TYPES' table.

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
res_id	varchar(128)		PRI	NULL	
res_name	varchar(128)	YES		NULL	
parent_restype_id	varchar(128)	YES		NULL	
res_key	varchar(100)	YES		NULL	

Table 4.2.2.1 Attributes of Resource Types

The res_id is the primary key for the table and it uniquely identifies the resource types. The res_name field contains the name of the resource type. The parent restype ID indentifies the UUID of the parent resource. This is used in managing the resources with parent and child relationship.

The RAN system comprises of several peers that operate independent of each other and communicate with each other for resources. There is no co-ordination or control over naming of the resources types or resources. They can have resources that is of the same type and name and still be different with respect to the content. The RAN resource directory also captures information on properties that uniquely indentify the resource. The res_key is an important property which specifies which parameter uniquely identifies

the registered resource. For examples, the resource type file, uses ‘md5checksum’ or a combination of ‘filename’ and ‘md5checksum’ as the res_key. If the requesting application has the res_key property value of the file it is looking for, it can validate the returned result with this value.

4.2.3. RESOURCE_TYPE_PROPERTY

The table ‘RESOURCE_TYPES_ PROPERTY’ is used in managaing the list of properties for all the disparate resource types that are registered by the resource exporter or by other applications in the RAN peers. The number of properties for each resource type may vary based on the resource type, the application that is registering the resource and the application that will be consuming the resource. The aaplicable attributes are listed in 4.2.3.1 below.

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
id	varchar(16)	YES		NULL	
property_name	varchar(16)	YES		NULL	
property_type	varchar(16)	YES		NULL	

Table 4.2.3.1 Attributes of Resource Type Property

When a new resource type is added in the RAN resource directory, a new table is created to hold all the resources that will be registered for the resource type. The properties for the resource type that are captured in the table are used in resource specific table. RAN resource directory currently supports 2 types of datatypes – Varchar and

Integers. As an enhancement, RAN resource directory can be extended to allow properties with other database datatypes.

4.3. RAN ASCII Commands

As a RAN system can comprise of disparate systems, there is flexibility to allow communication between these systems in several forms which range from a simple ASCII message to webservice calls. In this implementation, the RAN resource directory is designed to support commands in the form of ASCII messages. The RAN resource directory ASCII interface uses standard TCP/IP port which is defaulted to 6767. This default can be changed by identifying any other port number through the system startup parameter. All RAN resource directory functionalities are supported through this ASCII interface. All these commands are implemented on a peer-by-peer basis. The list of ASCII commands supported by RAN resource directory are provided below.

4.3.1. System Commands

Version*:* RAN resource directory supports a system command called '*Version'* that returns the current version of the RAN resource directory. This command is primarily used by other applications that interact with RAN resource directory to check their version compatabilty.

Usage: Version;

Result: Version 0.5

Quit: The system command '*Quit*' closes its connection with the resource directory. Even though RAN supports multiple clients, as an enhancement, a timeout can be implemented as part of the RAN system to close the connection with the clients that are inactive for a

perdetermined time period. This feature would also improve the security around the client system application.

Usage: Quit;

Result: The client connection is closed.

GetUUID: As discussed earlier, RAN resource directory uses UUID as the unique identifier in all its database operation. All the resources in the RAN resource directory are associated with an UUID. This system command '*GetUUID*' retrieves the UUID of the resource specified.

Usage : GetUUID <Resource Name>

Sample: GetUUID *file*;

Result: 4442dfd5-2163-4077-9c30-a15926480510

Clear: RAN resource directory supports a system command '*Clear*' that is used to roll back all the changes and to revert to the original state. It unregisters all the resource types that are registered to the RAN resource directory and also deletes the individual resources for all the resource types. This is used during RAN resource directory reset process.

Usage:Clear;

Result: Clears the entire database on a peer and unregisters the resource types and deletes the .resources associated with the resource types

Database resource: 4 table(s)

Table	Records	Size	Created	U...	Engine	Comment
conn_stat	0	33 KB	2006-03-29 12:47:26	N/A	InnoDB	InnoDB free: 11264 kB
resource_directories	0	17 KB	2006-03-29 08:47:33	N/A	InnoDB	InnoDB free: 11264 kB
resource_type_property	2	17 KB	2006-03-29 08:47:33	N/A	InnoDB	InnoDB free: 11264 kB
resource_types	4	17 KB	2006-03-29 08:47:33	N/A	InnoDB	InnoDB free: 11264 kB

Figure 4.3.1.1 Snapshot of an Initialized or Cleared Database

4.3.2. System Parameter Commands

These commands return the value of the system parameter that is set duing RAN resource directory start up.

Ispeer-to-peer: The command '*Ispeer-to-peer*' returns a value of true if the RAN resource directory is operating in peer-to-peer mode, else the returned value is false

Usage: Ispeer-to-peer;

Result: True

Getip: The command '*Getip*' returns the ip address of a peer that was set during the start up.

Usage: Getip;

Result: 192.168.2.4

Getport: The command '*Getport*' returns the ASCII port value that was set during start up

Usage: *Getport*;

Result: 6767

4.3.3. Functional Commands

GetNumOfResourceType: The RANresource directory supports several different types of resources. The '*GetNumOfResourceType'* command retrieves the total number of resource types that are registered in the RANresource directory.

Usage: *GetNumOfResourceType*;

Result: GetNumOfResourceType: 3

AddResourceType: The different types of resources that can be shared in the RAN system are registered to the resource directory by using the command '*AddResourceType'* When

a peer unit is being initialized to join the RAN system, the RE, which is part of the peer, registers all the resource types the peer is capable of sharing with other peers in the network. When the peer is fully functional as part of the network, other applications can also register new types of resources.

Usage : addt uuid:<Resource UUID> <Resource Type Name> uuid:<Parent UUID> <Key Identifier>;

Sample:

addt uuid:4442dfd5-2163-4077-9c30-a15926480510 File uuid:00000000000000000000 filename+md5checksum;

addt uuid:63ace7be-4526-11dc-8314-0800200c9a66 FileChunks uuid:4442dfd5-2163-4077-9c30-a15926480510 filename+md5checksum+index;

Result: Adding ResourceType ...Success

Illustration of '*AddResourceType*' command:

The three distinct process that implemented for the '*AddResourceType*' command. The following figures show the snapshots of the database after '*AddResourceType*' command. Figure 4.3.3.1 shows that a new entry is added in the resource_types. The following figure, Figure 4.3.3.2, shows the actual entry in the resource_types table. Figure 4.3.3.3 shows the new table created for the newly added resource and it contains the system property Id.

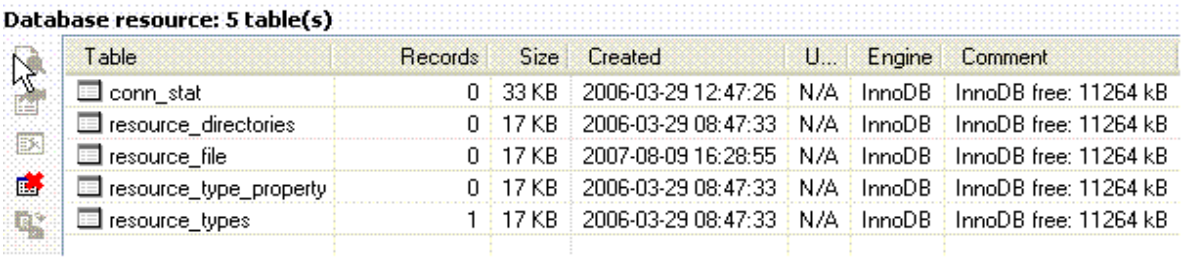


Figure 4.3.3.1 New Resource Type Entry

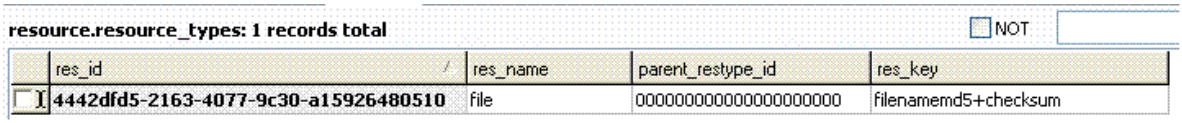


Figure 4.3.3.2 Data for New Resource Type Entry

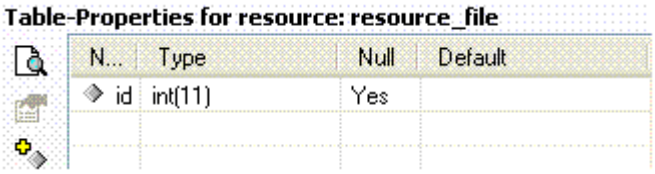


Figure 4.3.3.3 Table for New Resource Type Entry

GetResourceTypes : The command *'GetResourceTypes'* lists the different types of resources that are registered in the resource directory. For every resource that is registered in RAN, an unique table is created using the registered resource properties. When a new resource is registered, the table specific to the resource is populated with the new resource data.

Usage: *GetResourceTypes*;

Result: webservice, file, filechunks, diskspace

RemoveResourceType: Any resource type that is registered in the RAN can be unregistered. This allows flexibility for applications to add or remove a resource types. This command '*RemoveResourceType*' removes the resource type from the list of resources and also removes the properties specific to that resource type from the RAN resource directory. This also unregisters all registered resources of its kind.

Usage : del uuid:<Resource UUID>

Sample: del uuid:4442dfd5-2163-4077-9c30-a15926480510;

Result: removing resource … Success

AddResourceProperty: Each resource that is registered in RAN system has set of properties associated with it. These properties enable the resource directory to garner specific information about the resource type. When a new resource is registered for that type of resource, their properties are also stored in both the resource type property repository and also in the table for the resource type. The properties are very important for a resource type as they enable the resource exporter and other applications to search for registered resources using specific property value.

The properties are unique to a resource type and can be either of type string or integer.

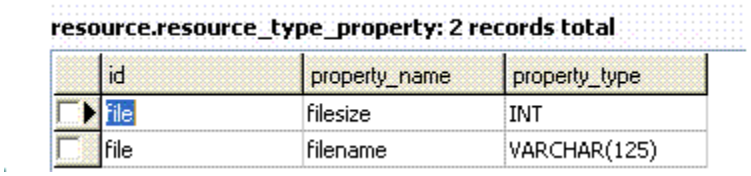
Usage: : addp uuid:<UUID of the resource type> <Prop Name> <Prop Value>;

Sample : addp uuid:4442dfd5-2163-4077-9c30-a15926480510 fileSize 11;

addp uuid:4442dfd5-2163-4077-9c30-a15926480510 fileName VARCHAR;

Result: Adding resource property...Success

Figure 4.3.3.4 below shows the effect of addp command in the database. Two new entries are created in resource_type_property table and the Figure 4.3.3.5 below shows the new columns that are added in the resource_file table.



resource.resource_type_property: 2 records total

	id	property_name	property_type
▶	file	filesize	INT
	file	filename	VARCHAR(125)

Figure 4.3.3.4 Effect of ADDP Command on Database

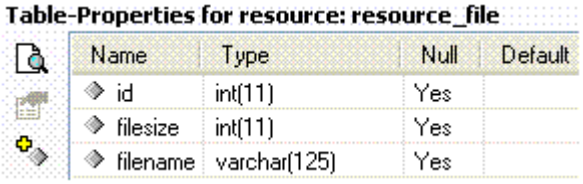


Table-Properties for resource: resource_file

Name	Type	Null	Default
id	int(11)	Yes	
filesize	int(11)	Yes	
filename	varchar(125)	Yes	

Figure 4.3.3.5 New Columns in Resource File Table

GetResourceTypeProp : This command *'GetResourceTypeProp'* returns the list of properties registered for a specific resource type.

Usage: getprop <Resource Name>

Sample: getprop file;

Result: fileName, fileSize

RemoveResourceProperties: This command '*RemoveResourceProperties*' is used to remove all the properties that are registered for a resource type or can also be used to remove specific properties. When this command is invoked, it removes the property from the resource type property repository and also from the table for the resource types.

Usage: : delprop <Resource Type Name> List of [<Prop Name>]

Sample : delprop file;

delprop file fileSize;

Result: removing all resource property...Success

removing specific resource properties...Success

Figure 4.3.3.6 shows the effect of '*RemoveResourceProperties*' on the resource_file table when specific columns are dropped. If all the coumns are dropped, the table would be re-initialized to its original start state.

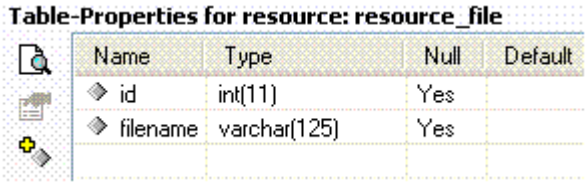


Figure 4.3.3.6 Column Drops from Resource File Table

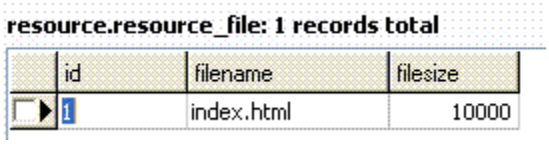
RegisterResource: After the RE adds the resource types that the peer unit is willing to share with other peers in the RAN, it registers the specific resources for the resource types. Every registered resource type can have one or more registered resource. The registered resource is addded to the specific table for that resource type along with its properties.

Usage: : reg uuid:<UUID of the resource > List of [<Prop Name> <Prop Value>];

Sample : *reg uuid:4442dfd5-2163-4077-9c30-a15926480510 fileName index.html fileSize 10000;*

Result: Adding resource property...Success

As shown below in Figure 4.3.3.7, when a new resource is registered, a new record is inserted in the resource_file. This resource is now available for the resource importer and other applications.


resource.resource_file: 1 records total

	id	filename	filesize
▶	1	index.html	10000

Figure 4.3.3.7 New Resource File Record

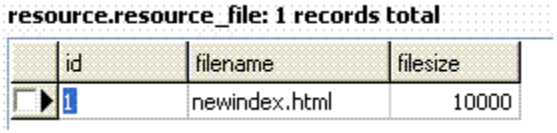
UpdateRegisteredResource: A registered resource can update its information using the command '*UpdateRegisteredResource*'. All the properties, except the id which is system generated field, can be updated using this command.

Usage: up uuid:<UUID of the resource > List of [<Prop Name> <Prop Value>];

Sample: up uuid:4442dfd5-2163-4077-9c30-a15926480510 filename index.html; filename newindex.html;

Result: Updating Resource… Success

In Figure 4.3.3.8 shown below, the registered resource is updated with the new file name after the '*UpdateRegisteredResource*' command.


resource.resource_file: 1 records total

	id	filename	filesize
▶	1	newindex.html	10000

Figure 4.3.3.8 Updated Filename for New Resource

GetRegisteredResourceNum: : This command *'GetRegisteredResourceNum* returns the list of registered resource for a specific resource type.

Usage: getnr uuid:<UUID of the resource >

Sample: getnr uuid:4442dfd5-2163-4077-9c30-a15926480510;

Result: : NumberOfRegisteredResources: 4

UnregisterResource: This command '*UnregisterResource* is used to remove registered resources for a specific resource type specified in the command. The command removes the registered resource that is selected based on the search criteria which is also specified in the command. The number of parameter in the search criteria has to be atleast one and can have upto all the properties for that resouce type. All the records matching the crieteria would be deleted.

Usage: *unreg* uuid:<UUID of the resource > List of [<Prop Name> <Prop Value>];

Sample: *unreg* uuid:4442dfd5-2163-4077-9c30-a15926480510 fileName index.html; fileSize 10000;

Result: Removing Registered Resource… Success

Query: The resource directory unit can be either a standalone or actively partcipate in the RAN system. As discussed earlier, this is initialized as system parameter. In both cases, the resources that are registered in the RAN resource directory should be searchable. This command '*Query',* which is one of the most important functionality of RAN resource directory, queries the resource directory for a specific registered resource. Based on the mode in which the resource directory is operating, the command parameters may vary.

The RI in the peer is where all the applications would query for resource. If the peer, is participating in RAN, the RI would also be receiving requests for the resources from other peers. The RI queries the local resource directory and if the search returns resources that match the criteria in the query, it would send the result to the requester. If the query does not fetch a result from the local resource directory, the RI then can query other peers

that it is connected to. The details of which peer it would send this request to is covered in the follwing chapters.

The parameter ‘hopnumber’ in the query comand is optional and is used only if the peer is operating in peer-to-peer mode. This value is very similar to TTL in network messages and is used to specify the life time of a query in RAN. The query lifetime in RAN is the number of different peers a message can be routed to before it becomes inactive.

Usage: que uuid:<UUID of the resource > List of [<Prop Name> <Operator> <Prop Value>]; <Number of results> < Result property> [<Hop Number>];

Samples:

que uuid:4442dfd5-2163-4077-9c30-a15926480510 fileName = file3 ; 3 file 5;

que uuid:4442dfd5-2163-4077-9c30-a15926480510 name = file3 ; 8 createddate 5;

Result: SendQuery: , file2, file21, file22

SendQuery: , file2, file21, file22, file23, file24, file25

5. Qualities of RAN Resource Directory

This chapter provides a discussion on the general qualities of the RAN resource directory. The key qualities such as modifiability, performance, usability, scalability and accessibility are discussed in detail.

5.1. Modifiability

The peer-to-peer architecture is new and its potential is just starting to be realized. The RAN directory service has a high degree of configurability. The most important feature of RAN resource directory is its capability to handle different types of resources. One of the applications developed based on the RAN system RDSS*, a Resource Area Network (RAN)-based Distributed Storage System [5],* is capable of downloading files in parts, interrupting and later resuming these downloads.

Several features, like support of disparate resources and ASCII listener, have been made available in order to improve the extendibility of the peer-to-peer systems. Since RAN resource directory is developed on a Java platform, it is flexible for interoperability with other systems. In addition, it is possible to add future enhancements, one like peer authentication, to make the RAN resource directory implementation a more robust peer-to-peer system.

5.2. Performance

Peer-to-peer systems' usability primarily relies on its architecture and performance. The time taken to join the existing network and locate the resources governs the usability of the system. In RAN, relatively longer time is spent for a new peer to join the existing network. This is because the seed node attempts to ensure a balanced network structure during the join process. As explained in previous chapters, a balanced network is

effective in performing searches as the height of the tree is reduced. However, the time taken for registering a resource into the network is minimal. As the tree is balanced, the time taken to search for a resource is highly reduced.

5.3. Usability

As with any peer-to-peer system, the general usability of the RAN resource directory should be such that a common user can easily interact with the system. The RAN resource directory has an ASCII interface and a Java API that the user can select to interact with the RAN application layer. As a future enhancement, other interfaces like webservices and http can be implemented to accommodate a broader range of systems.

5.4. Scalability

In a peer-to-peer network, the scalability is the ability to handle large number of peers and still operate in an efficient manner. In RAN resource directory, a peer only stores information about its resource directory and the network information of its immediate peers. There is no central node to hold the information about all the peers in the network; each peer does not carry data on peers that it is not directly connected to. This limited network information helps the peer to manage its position in the network and still be completely functional as any other peer in the network. This feature helps RAN resource directory to support a large number of peers.

5.5. Accessibility

As with any peer-to-peer system, the RAN resource directory allows peers from other networks to connect to it. The peer comes packaged with the resource importer, resource exporter and resource directory. For a peer to connect to the RAN system, it is required to have a seed peer's ip address. The seed peer determines where in the network

to connect the peer to expedite the tree structure balancing process. As an enhancement, an authentication can be developed to protect the data in the network.

6. Search in RAN Resource Directory

In this chapter, the search mechanism in RAN resource directory is discussed in detail. As explained in the previous chapters, the RAN system can comprise of hybrid peers connected to each other in a physically unstructured network topology, as the peer location is not determined by its resource content. There are several protocol messages in the RAN resource directory, as discussed in chapter 5, that are used either to maintain the peer structure as a balanced binary tree to increase the efficiency of search or to search and add resources from other peers to the RAN resource directory in the local peer.

For a tree of n nodes, its balanced binary tree will be of the tree of shortest height when compared to all the tree structures that can be created from n nodes. The peers in RAN are always connected and maintained in the form of balanced binary tree. The balancing is performed by several protocol messages between the peers. The messages related to the directory service also follow the same format as the protocol messages. They are sent as ASCII messages between the peers. Chapter 5 describes the messages in the RAN. The search for the resources is straightforward in RAN and the following section outlines the search mechanism in RAN system.

The search mechanism in RAN involves sending simple ASCII messages between the peers. In this section, the several steps involved in the search mechanism are illustrated using a sample peer-to-peer network as shown in Figure 6.1.

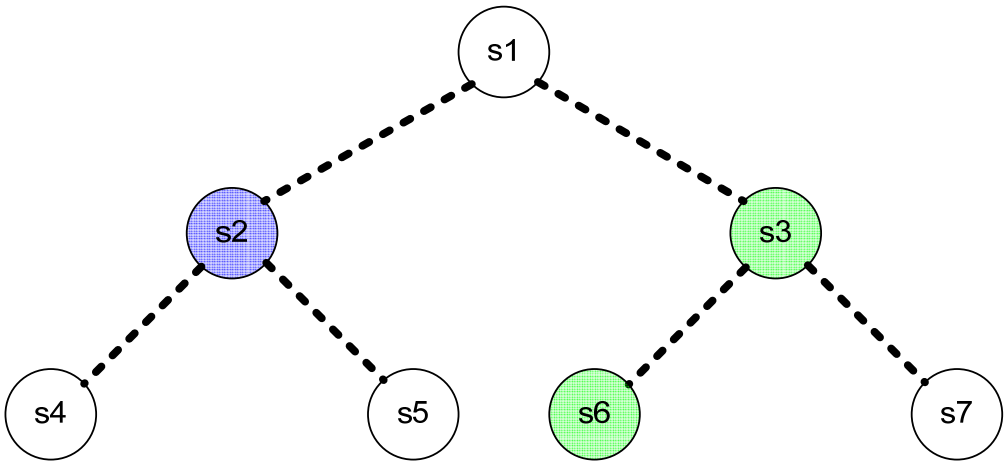


Figure 6.1 Sample peer-to-peer Network

Description of Sample Peer-to-Peer Network:

The sample peer-to-peer network consists of seven peers – the term peer used interchangeably to refer to a node or a peer. The node colored in blue is the location where the search query message is initiated; the green colored nodes are the peers where the data associated with the search is located. To keep the focus limited to the search mechanism, it is assumed that the nodes in the peer-to-peer system are already balanced. The node-balancing algorithm is not part of the search mechanism discussion and is (assumption) performed independent of the node search activity.

6.1. Building the Query Message

At node s2, the search process creates the search query message. The RAN resource directory supports two types of query messages based on the operational mode (peer-to-peer or standalone).

Peer-to-Peer Operational Mode:

For a peer-to-peer operational mode, the message used – 'query_resource UUID <property list>; max; <return property list>;hopNo' – runs the query for a resource in the

entire peer-to-peer network (only limited by the ‘hopNo’) and returns the result in an array.

Standalone Operational Mode:

For a standalone operational mode, the message used – ‘query_resource UUID <property list>; max; <return property list>;’ – runs the query for a resource (locally in the peer) and returns the result in an array.

The parameters for the query message are as follows:

UUID: The universally unique identifier for every resource registered in the RAN resource directory.

Property list: This contains the list of properties of the resource prop1 value1 cond1 prop2 value2 cond2

Max: This attribute defines the maximum number of results that can be returned per query.

Return property list: This list contains the parameters of the resource that are required by the caller application. As the RAN resource directory can interact with several applications, each application may require different response parameters.

hopNo: This property is very specific to the peer-to-peer systems. This determines the maximum number of hops that the peer-to-peer resource directories would perform to complete a query. A value of ‘0’ for this attribute indicates standalone or non-peer-to-peer mode.

Command	**UUID**	**Property List** (*Variable Length*)	**Max** (*4 Byte*)	**Return Property List** (*Variable Length*)	**hopNo** (*4 bytes*)

Figure 6.1.1 Query Message Structure

6.2. Request Initiation

When the 'search query message' build process is complete, node s2 initiates the request. Upon request initiation, the messages are sent to the nodes directly connected to node s2. Blue arrows in Figure 6.2.1 represent the message propagation. Note: If the search is performed on a standalone or non-peer-to-peer mode, the search is completed on the local node, s2, and the search results returned without any network propagations.

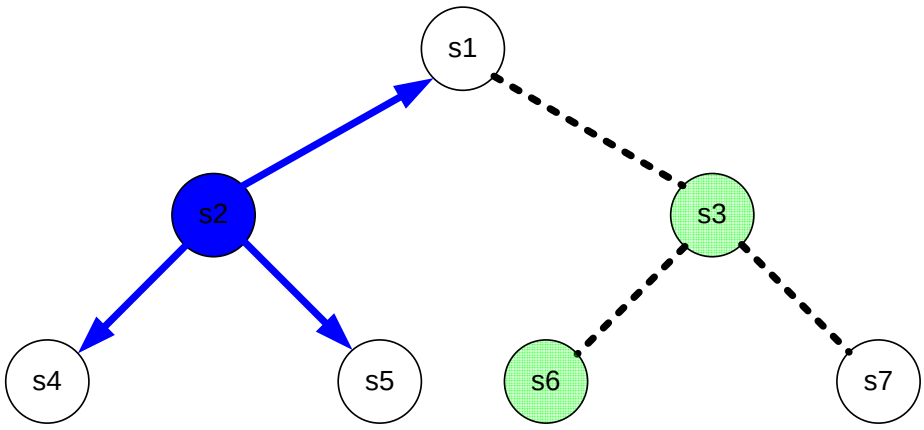


Figure 6.2.1 Request Initiation

6.3. Request Processing

When the nodes, s1, s4 and s5, receive the search request from their adjacent node, they begin to process the request – depending on node connectivity, this could be a two-part process.

6.3.1. Request Processing – Part 1

When processing the request, if a node has other adjacent nodes, then the node first (sets off a cascading chain by) propagates the request to other adjacent nodes – other than the originator of the request. However, if no adjacent nodes are available, the resources local to the node, as in nodes s4 and s5, are searched and the results are returned to node s2. This initial process is illustrated in Figure 6.3.1 below where nodes s4 and s5 only perform a search of their local resources and return a negative result – colored in red. Node s1, however, propagates the message onwards to node s3 before performing the second step in processing the message received from node s2.

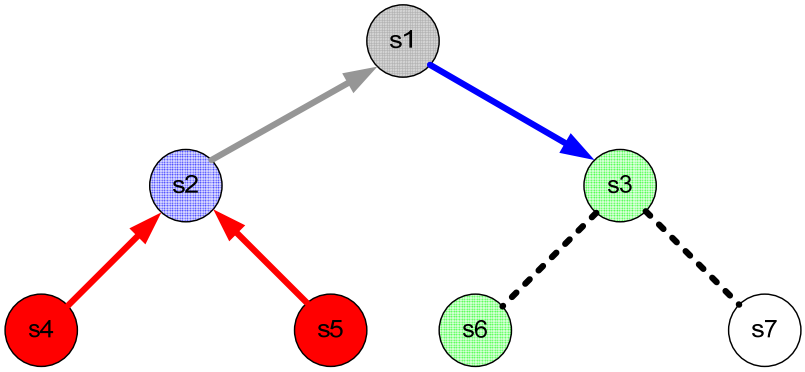


Figure 6.3.1 Request Processing – Part One

6.3.2. Request Processing – Part 2

Subsequent to propagating the message onwards to node s3, node s1 performs local search for the data requested by node s2 and returns a negative result – colored in red. In addition, while this happens, node s3 processes the request from node s1 and has started to process its two-part request process. These steps, covering nodes s1 and s3, are illustrated in Figure 6.3.2 below.

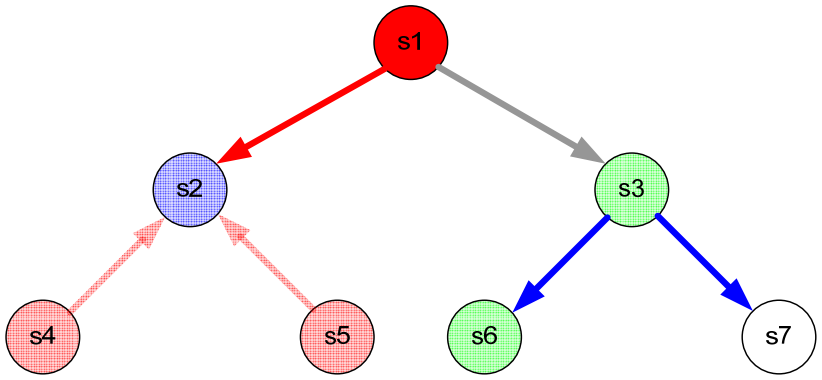


Figure 6.3.2 Request Processing – Part Two

6.4. Successful Search Return – First

After propagating the message onward to nodes s6 and s7, node s3 performs a search of the local resources for the data requested from node s1. As the requested resource is available at node s3, the search query message initiated by node s2 has obtained its first successful search return. When the search returns a positive result – colored in green, the requested data is broadcast to the originator, node s2, of the search request. The data broadcast is implemented by 'back tracking' the message propagation route. In this example, the successful search result from node s3 is broadcasted to node s2 after being routed through node s1. The original blue color used for node s2 is replaced with color green to represent that the data requested is now available as part of node s2's resource. This process is illustrated in Figure 6.4.1 below.

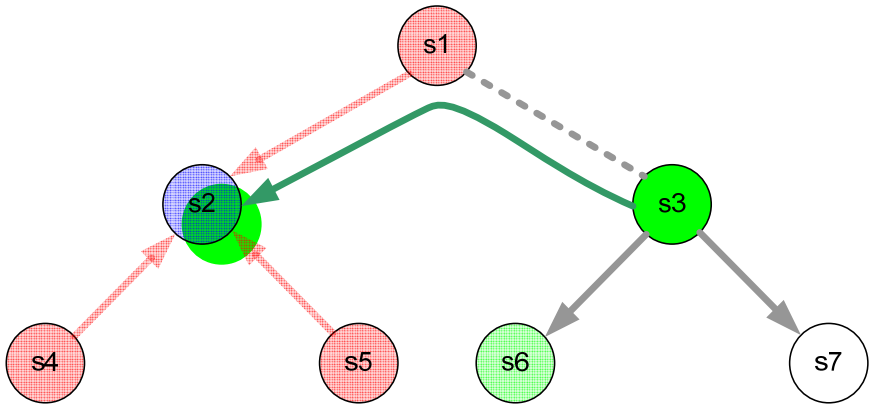


Figure 6.4.1 Successful Search Return – First

6.5. Successful Search Return – Subsequent

Even though, node s3 returned a successful result for the query, the search continues to propagate through the peers. When node s6 receives the request, it has no knowledge of the prior success in query and processes the search request. As the search returns a successful result, node s6 broadcasts the requested data back to the originator, node s2, by 'back tracking' the initial request. However, (as shown in Figure 6.5.1) at node s2, this and other subsequent results received are not used.

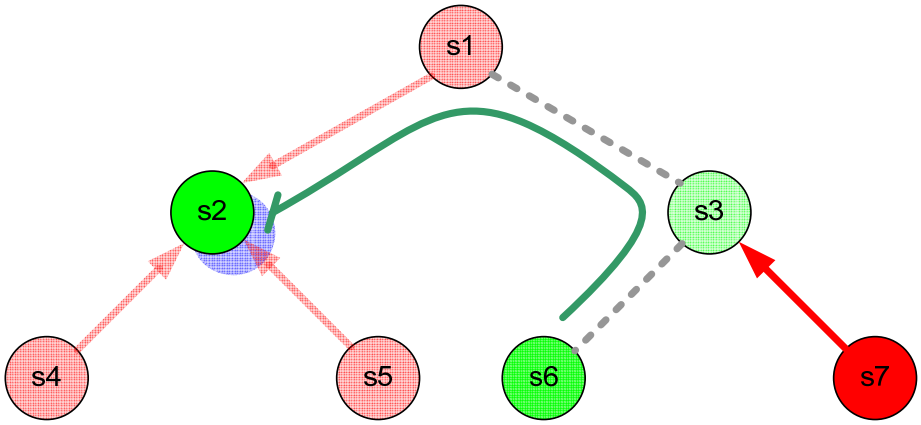


Figure 6.5.1 Successful Search Return – Subsequent

7. Balancing in RAN

The search response time in a peer-to-peer system is one of the key factors in measuring the efficiency of the network. In this chapter, the factors affecting search, the need for network balancing, and the details of the balancing algorithm used are discussed.

7.1. Factors Affecting Search

In this section, some of the factors affecting the search performance, the possbile methods to mitigate these factors, and the specific factor addressed through the RAN resource directory implementation are discussed.

Some of the factors that determine the response time for search of resources in the RAN are:

- Size of Network/ Network Density

This is the number of active peers in the network that can participate in the search. In general, the greater the number of active peers in a network, the larger the search time for resources in the network. Due to its inherent scalable nature, the the peer-to-peer networks, like RAN, have a large number of active peers. Limiting the number of active peers in the network will be counterintutive to the scalable naure of the RAN system. Hence the size of the network cannot be limited to ensure reduced search time in the RAN. Also, the network density of a peer-to-peer system, the number of connection for a peer, determines the time taken to propogate the search messages. However, for a network with higher density, the number of duplicative xsearch generated and propagated is very large.

- Number of Knowledge Peers

This is the number of peers that contain the specfic information that is sought after through the search query. In networks that are geared towards serving popular content, it may be possible to find several peers that contain the same information. However, in RAN, as the peers are designed to share multiple resource types and as RAN is degined to be a multi-functional network, the number of knowledge peers for specfic information may vary. Though there is a specific type of resource directory that actively gathers information from other resource directories, there is no control over the type of peers that join to form the network or the number of active peers during a search operation.

- Depth of Search/ Number of Hops

This refers to the number of peers through which the search query is propagated. Each hop of the search message entails some cost for each peer in terms of processing time and bandwidth. Every peer in the message route, uses its own resources and network bandwidth to process the request and propagate it further in the network. The overall serach response time is the aggregate time taken by each peer in the message route to process the request and return the response back to the source.

The number of hops encountered, for a specific search, to return a successful response depends on the relative location of peers and the length of the message route in the network. In RAN, as all peers are equal, there is no control over the relative location of the source and knowledge peers. However, the length of the message route can be minized by managing the network structure and thus reducing the search resoponse time.

7.2. Balancing Algorithms

A tree structure is considered to be balanced when every peer has the same number of children and all leaf peers are equidistant from the root peer. Several balancing algorithms are available to maintain a network in a balanced state. In most cases, the balanced state of a network can be defined in terms of the height or weight of the network peers. In a weight balanced tree structure, each peer is aligned in the network based on its key value with regard to its parent's key value. A key value could be a ranking or a probsbility of the peers based on one or more factors [38]. For example, in a tree where the peers are ranked between values 1 and 100, the root peer could have a rank of 50 and all peers that have a rank of greater than 50 are connected to the right; peers that have a rank of less than 50 are connected to the left of the root peer. Such a tree is said to be a weight balanced tree structure.

In a height balanced tree structure, the difference in height between the sub-trees are not greater than one. One example of a height balanced tree is an AVL tree. In an AVL tree, the balancing is based on the key value of the nodes. The AVL tree maintains the tree structure such that the values on the left subtree is always less than the root or parent node and the values on the right subtree is always greater than the root or parent node [26,38]. In RAN implementation, the balancing algorithm implemented maintains the network structure as a height balanced binary tree structure. In this implementation, rather than using a key value for the balancing process only the size of the subtree is used.

7.3. Why Balanced Binary Tree

In the RAN system, the peer units are connected to each other to form a binary tree – a network tree is a graph where the peers are represented by the vertices and the peer connections are represented by the edges. The Binary tree [26] is a structure that can have a maximum of two children at any given node. Though this is one of the simplest forms of tree data structures, this is beneficial for the RAN implementation as it is easier to perform balancing and, on a continuous basis, maintain the network in a balanced state. The two children are referred to as the left and the right child. The children can in turn be a binary tree. The root of the tree is a node with no parents. In RAN, these are usually the seed node whose existence is known in the RAN system. The peer units, when joining the network, communicate with these seed nodes and the seed nodes direct the peer on how to join the network.

RAN uses a binary tree network structure for its features like fullness, density and the ability of being balanced. A binary tree can be described as a full binary tree if all the nodes that have less than two children are in the two lower levels of the tree structure. The tree can be converted into a full tree by making sure that the new nodes are added in an order so that the current level is full before a new level is introduced in the tree. In RAN, depending on the balanced state of the peers' interconnected structure, the seed directs the new peer to join either its left or right child. As the decision to join the structure is governed by the current state-of-balance of the peers, the tree is maintained as a full tree to certain extent. In order to reduce the number of messages in the network and also considering the fact that the RAN system can consist of peers that have limited life time in the network, the structure is allowed to deviate from its pure full form.

The density of a binary tree is defined as its 'leftness' based on the distribution of the nodes in the tree structure. The nodes that have two children are found to the left of all the nodes which have one child or less. The nodes with one child are found to the left of all the nodes with no children. However, this maintenace of density requires more messages in the system and more bandwidth. In RAN, leftness is not maintained as the structure is balanced. By not maintaining the density of the structure, the available bandwidth are allocated for other purposes such as balancing the nodes, querying a resource and maintaining the structure of the network.

A completely balanced tree is a tree structure in which the height and size of the two children of any node is the same. In RAN, the peer units' interconnected structure is a height-balanced binary tree structure resulting from using the 'balancing algorithm' explained in the following sections.

In a structure where the nodes connect without any predetermined order, the running time of the search operation is dependent on the number of nodes in the structure. Based on computational theory, the big O notation (O) is used to describe the asymptomatic upper bound of any computation [26]. In searching an un-indexed dataset, which can be a linear tree structure, of size 'n' with values that are in random order, it will take a maximum of n steps to find a data from the tree structure. The worst-case running time of a search of the unordered tree structure is O (n). In the case of the tree with node values, the search operation depends upon the height of the tree and the search is said to be O ($\log_2$ n) [25]. Among the various types of possible binary tree structures, the height of the tree is the least with the balanced tree implementation. The number of nodes that can be in a balanced binary tree is 2^h-1, where h is the height of the tree.

In RAN resource directory, as it is very important to reduce the search time for resources, a balanced binary tree structure has been incorporated. The balancing algorithm always tries to keep the height of the tree to a minimum at all times. With a balanced tree implementation in the RAN resource directory, the search time is reduced as it takes the shortest time to look for a resource compared to any other structure with the same number of nodes.

7.4. Design Requirements for Balancing

The table 'CONN_STAT' is used in managaing the list of peers that are connected to each other in the RAN. Some of the key attributes are described in this section. Each peer unit consists of conn_uuid, which uniquely identifies the peer unit. This identifier is in the form of an UUID. It captures information about the other peers that it is directly connected to. The information on its parent and its children are captured as 'conn_parent', 'conn_right_uuid' and 'conn_left_uuid' respectively. A snap-shot of the 'CONN_STAT' table schema is provided in Figure 7.3.1.

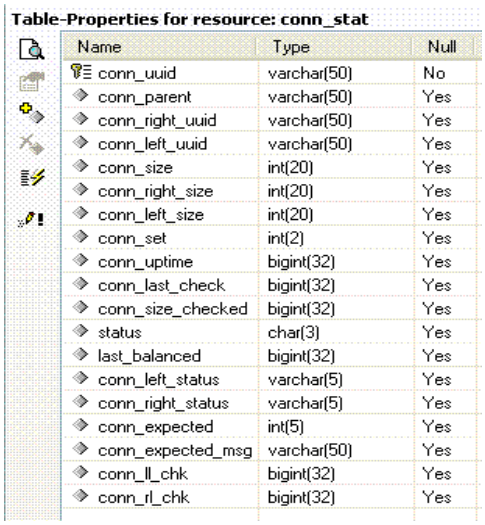


Figure 7.3.1 Conn_Stat Schema

For the balancing algorithm, the CONN_STAT table also maintains the size of the peer unit and its child sub-tree sizes in the variables ‘conn_size’, ‘ conn_right_size’ and ‘conn_left_size’ respectively. When the peer connects to another peer it sets the peer’s parent value in ‘conn_set’.

The table also manages several attributes that denotes the peer’s lifetime in the network. The ‘conn_uptime’ denotes the total uptime of the peer unit, ‘conn_last_check’ denotes the time of the last status check and ‘conn_size_checked’ denotes the time of last size check. This table also contains other attributes that relate to the balancing algorithm. The algorithm specifies where each node can be added so that the underlying structure remains balanced.

7.5. Resource Protocol

Several messages are sent between the peer units for maintaining the structure of the network and checking the status of the other peer units. The list of messages sent for maintaining the network structur are described in the following sections.

7.5.1. Docking Messages

The ‘Docking Messages’ are the preliminary information exchange between an un-networked peer unit and RAN peer units. The docking messages are used during the initial join process and during the balancing process.

RTJ (Request to join)

The child peer unit sends a RTJ message to the parent with its defining parameters. The peer verifies if this peer could be added as its child while still maintaining the balanced nature of the entire network. If it can accept the peer as its child, it verifies if the parameter ‘total size’, which is part of the message, is valid by

checking it against the sum of the ‘left size’ and ‘right size’ parameters from the message. If the message is verified to be valid, it sends a JME (Join Me) message to the child peer.

Usage: RTJ : PEER UUID : Right Size : Left Size : Total Size

Sample: RTJ: 132.235.85.150#4545#4546: 1: 2: 3

JME (Join Me)

The parent peer sends this message back to the requesting child and the child verifies if the child UUID specified in the message matches with its own UUID. Upon successful verification, the child peer sends the JSC (Join Success) message back to the parent.

Usage: JME : Parent UUID : Child UUID

Sample: JME: 132.235.87.150#4543#4542: 132.235.85.150#4545#4546

JSC (Join Success)

The child sends this message back to the parent after verifying the ‘Join Me’ message. Upon receiving this message, the parent updates its connection data structure and sets the child UUID and the size of the sub-tree. Only on the receipt of this message does the child peer become the child of the parent peer unit.

Usage: JSC: Child UUID

Sample: JSC: 132.235.85.150#4545#4546

JST (Join Sub-tree)

In instances where the parent may already have two child peer units, the parent sends out the message “Join Sub-tree“ (JST) with its left and right child identifiers. The order of the child is based on the size of the left and right sub-tree. It sends the sub-tree

with least number of children as the primary parameter and then the other one as the secondary parameter. On receiving this message, the child creates a new socket with the primary UUID sent by the parent and the operation continues.

Usage: JST: Primary UUID : Secondary UUID

Sample: JST : 132.235.88.150#4541#4542 : 132.235.89.150#4542#4543

7.5.2. Alive Messages:

Each peer unit keeps track of the status of its children peer units. They send the 'Alive' messages so that the changes in the network are correctly updated throughout the network.

ICA (Is Child Alive)

The child sends this message to the parent periodically and checks if the parent has the updated information about the child. If the parent has the correct information then it sends a response back to the child.

Usage: ICA : Parent UUID : Child UUID : Total Size of Sub-tree

Sample: ICA: 192.168.2.4#4543#4544#6768: 192.168.2.4#4542#4543#6769: 1

SCA (Success Child Alive)

The parent sends this message to the child in response to an ICA message. If the parent contains the correct information about the child then the parent updates the 'conn_last_check' time to keep track of the last time the children were alive.

Usage: SCA : Parent UUID : Child UUID

Sample: SCA: 192.168.2.4#4544#4545#6767: 192.168.2.4#4543#4544#6768

SSS (Send Sub-tree Size)

If, through the ICA messages from the child, the parent detects that the size of the child sub-tree has been changed, it sends the child the SSS message. Upon receipt of this message, the child responds with the updated size information to its parent through an ICA message.

Usage: SSS : Parent UUID : Child UUID

Sample: SSS:192.168.2.4#4544#4545#6767:192.168.2.4#4543#4544#6768

7.5.3. Balancing Messages

In this section, the several messages exchanged during the balancing process are discussed in detail. These messages enable the peers to inform all the other peers about the balancing process. The peers that participate in the balancing process are protected from being part of multiple balancing acts by their statuses. At any given time, multiple peers can initiate the balancing process. They would have to wait for confirmation from all its peers to change its status to the acceptable value before it can start balancing.

CIB (Check If Balanced)

The peer sends a CIB message to the child to indicate that the balancing is underway. The status is set to ‘U’ if the peer is unbalanced for more than the threshold time. In addition, when the peer status is changed to ‘S’, it sends out CIB messages to its children. Any peer that receives the CIB message checks if it is a leaf peer, if so, it sets its status to ‘N’ indicating that it is ready for balancing. If it is not a leaf peer, it sets its status to ‘C’ indicating that it needs to alert its children. When the status updates are received from the children, it sets its status to ‘W’ as it is waiting for its turn to update its parent.

The CIB is the controlling message with which the nodes that are participating in the balancing process are alerted to the changes. The peer and its sub-tree are locked until the balancing process is complete. This is implemented so that same subset of peers are not participating in multiple balancing act.

Usage: CIB

Sample: CIB

SCIB (Success – Check if Balanced)

The SCIB message is a notification message for the parent peers that the child has received the CIB message. Upon receipt of this message, the peer resets it corresponding child's status with respect to balancing. This message is used to get confirmation from the child peer that it is willing to participate in the balancing act. It is also used to verify if the child is not already participating in other balancing operation.

Usage: SCIB <child UUID>

Sample: SCIB:192.168.2.4#4543#4544#6768

GCIB (Got – Check If Balance)

The GCIB message is sent from the Child to its parent peer indicating that it is waiting to hear from its children on the status change. This message indicates that the child peer has received the CIB message, but has not performed any operation to update its children. This message is a response to the CIB message until the child gets confirmation from its children on their willingness to participate in the balancing process.

Usage: GCIB <parent UUID>

Sample: GCIB:192.168.2.4#4543#4544#6768

Balanced

The peer that is identified during the balancing process establishes the connection with the unbalanced peer. If the peer was successful in establishing the connection with the new peer, it sends the 'Balanced' message to the peer indicating that the connection was established; the parent is set to the new peer. It also resets the other statuses that were set during the balancing process. This is the last step in the balancing process.

Usage: Balanced

Sample: Balanced

7.6. The Balancing Algorithm

This section details the various steps involved in the balancing process implemented as part of the RAN resource directory.

7.6.1. Initiation of the Balancing Process

The peer-to-peer balancing algorithm mainly works on messages, discussed in the previous sections, which are passed between the peers. The status of the peers governs the initiation of the balancing algorithm. Table 7.5.1.1 provides a list of valid status values and its description of the peers.

Status	**Description**
B	This value indicates that the peer is in a 'Balanced' state.
U	This value indicates that the peer is in an 'Unbalanced' state.
C	'Can I Balance' indicates that the peer is waiting on its child to reset their status.
W	'Waiting to be balanced' indicates that the peer is waiting for its turn to update

	its parent on its status change.
S	'Source of the request' indicates that the peer is initiator of the balancing process.
N	'Balancing the tree' indicates that the peer is ready to for the balancing process.

Table 7.5.1.1 Status of Peer Units

The following is the sequential process flow of the status changes during the balancing algorithm illustrated using a sample RAN system of five peer units as shown in Figure 7.5.1.1.

Balanced State

By default, when a peer unit initializes, the status is set to B (Balanced).

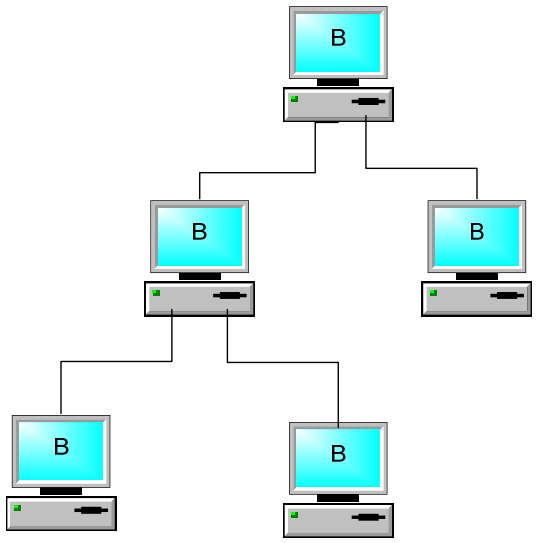


Figure 7.5.1.1 Balanced State

Un-Balanced State

When the imbalance detection logic determines the difference index (δ) between the left and right sub-tree size to be greater than threshold ($\delta_T = 3$), the peer unit status is set to U (Unbalanced) as shown in Figure 7.5.1.2 below.

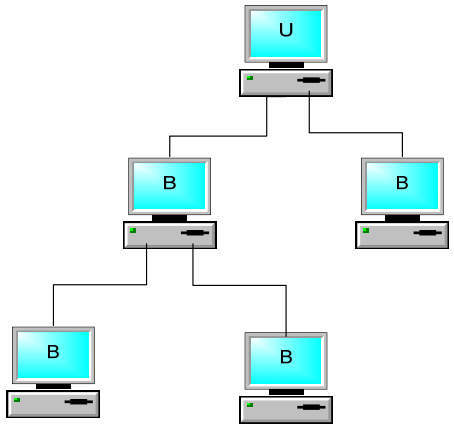


Figure 7.5.1.2 Unbalanced State

Determining Source Node

When the peer status is set to 'U', the built-in time delay logic ensures that the balancing algorithm is not triggered if the peer unit is in an unbalanced state for less than a predetermined time (5 minutes).

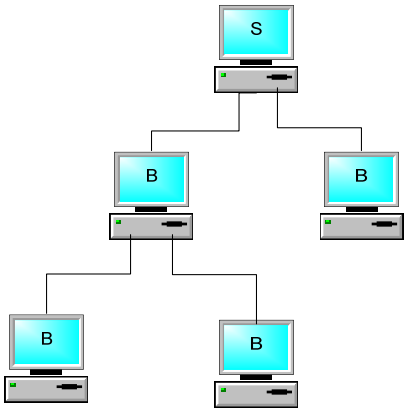


Figure 7.5.1.3 Source State

This time delay is provided to ensure that the balancing algorithm, which is processor intensive, is not activated for momentary unavailability of the peer units or temporary unbalance caused during the balancing process. However, if the unbalanced state of a peer unit persists for more than 5 minutes (or a variable time), the balancing algorithm is activated by setting the status of the peer unit to 'S' denoting that the peer unit is the source that initiated the current active balancing process – this is illustrated in Figure 7.5.1.3.

Status Change for Child Nodes

When the status of the source node is set to 'S', it sends out messages to its children with a request to change their status to N (Balancing status). The child node status is immediately set to 'N' if it has no children. However, if the child node is not a leaf node and in turn has children, then the request to change the status of its children is sent and the peer unit's status is set to 'C' (Can I Balance), as shown in Figure 7.5.1.4 below, while it waits for its children's status change.

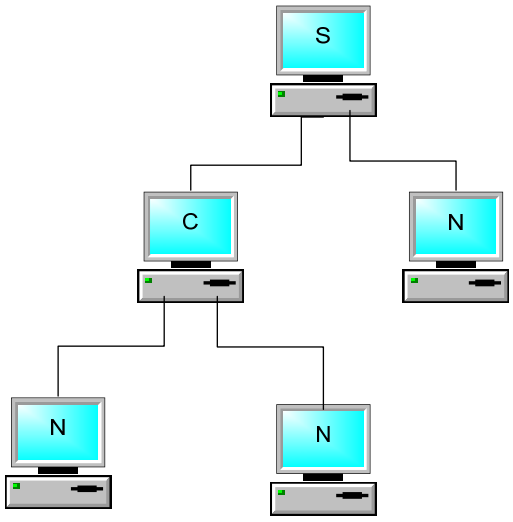


Figure 7.5.1.4 Child Node Status

When the peer unit receives a confirmation from all its children about their status change to ‘N’ then its status is also set to ‘N’. There is an intermediate state ‘W’ (Waiting to be Balanced), between states ‘C’ and ‘N’, for peer units with children when the peer unit has received confirmation from its children and is yet to communicate the new status to its parent. The status messages are then back propagated to the source until all the source’s children status are set to ‘N’, (shown in Figure 7.5.1.5) starting the ‘Balancing algorithm’.

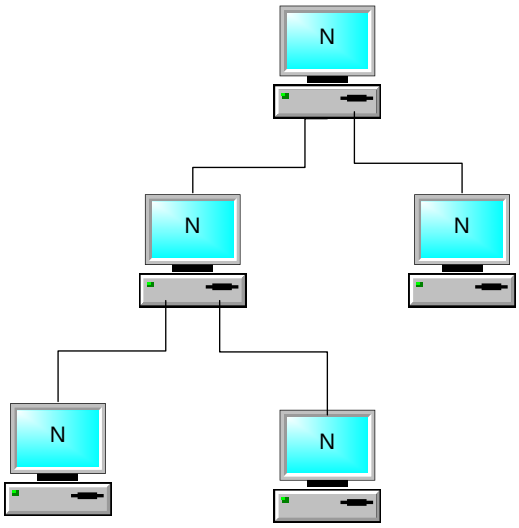


Figure 7.5.1.5 Balancing Status

7.6.2. Balancing Process Flow

The unbalance in the network can be of two types based on the nature of the sub-tree size. When the degree of unbalance in the network, determined by the index value, has an exact match with a sub-tree size, this unbalance is of type ‘Perfect Match’. However, when the degree of unbalance in the network does not have an exact match with a sub-tree size, the unbalance is of type ‘Imperfect Match’. In this section, both the types of unbalance are illustrated with sample networks.

Perfect Match Unbalance

The sample network used to illustrate this unbalance type is provided in Figure 7.5.2.1 below.

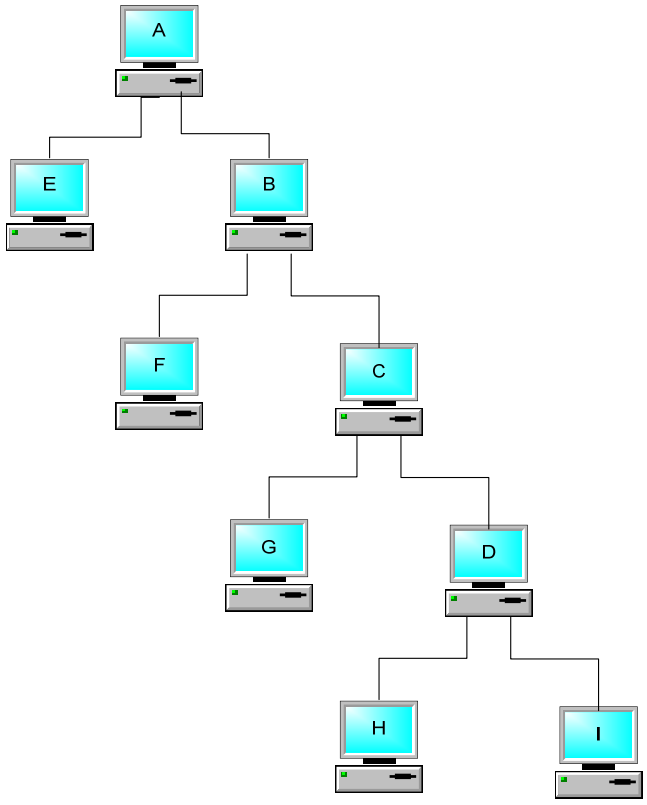


Figure 7.5.2.1 Perfect Match Unbalance

For this sample network, node A initiates the balancing algorithm when its status and all its children's status are set to 'N'. As the initial step in the balancing process the index (δ_n) value for source node A is computed using the function

$$\delta_n = \text{Ceil } \{[(\text{Right Child Size}) \sim (\text{Left Child Size})] / 2\}.$$

For the sample network, based on the equation above, the value of δ_A is computed as 3. The balancing algorithm searches the branch with the largest tree size to identify a sub-tree whose size matches the index value δ_A (3). Node C detects that one of its child, node D, is a tree of the size that matches the index value δ_A (3). Node C sends a message to its child node D requesting it to join the left child of the source node, node E. The network structure after this round of balancing in presented in Figure 7.5.2.2 below.

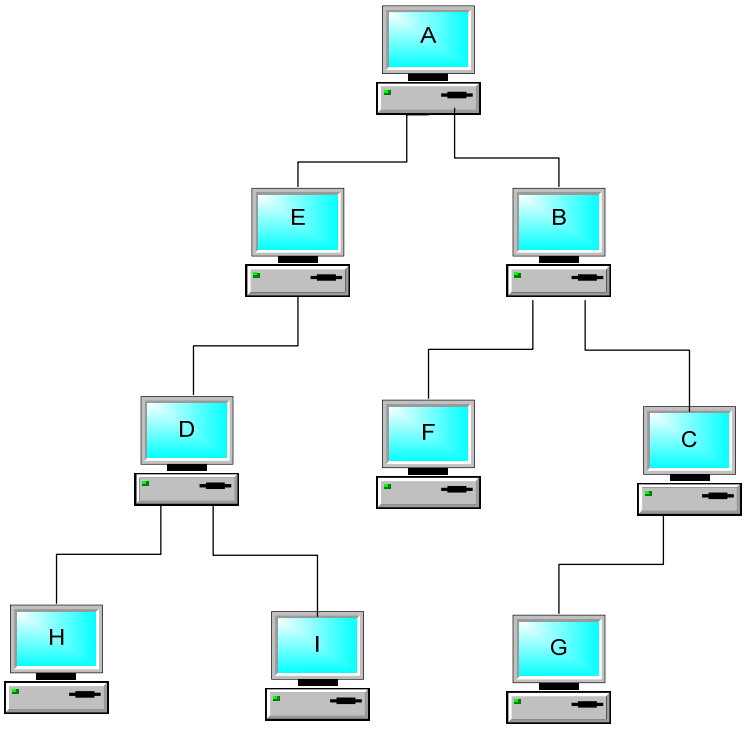


Figure 7.5.2.2 Perfect Match Balanced Network

After this initial round of balancing, the network is in a 'near' balanced state with respect to node A. However, upon cursory observation it may appear that the network is still unbalanced with respect to node E. The new unbalance index computed for node E, using the equation specified above, results in a δ_E value of 2. As the threshold for the index value δ_T is set to 3, no further balancing is performed in this network.

Imperfect Match Unbalance

The sample network used to illustrate this unbalance type is provided in Figure 7.5.2.3 below.

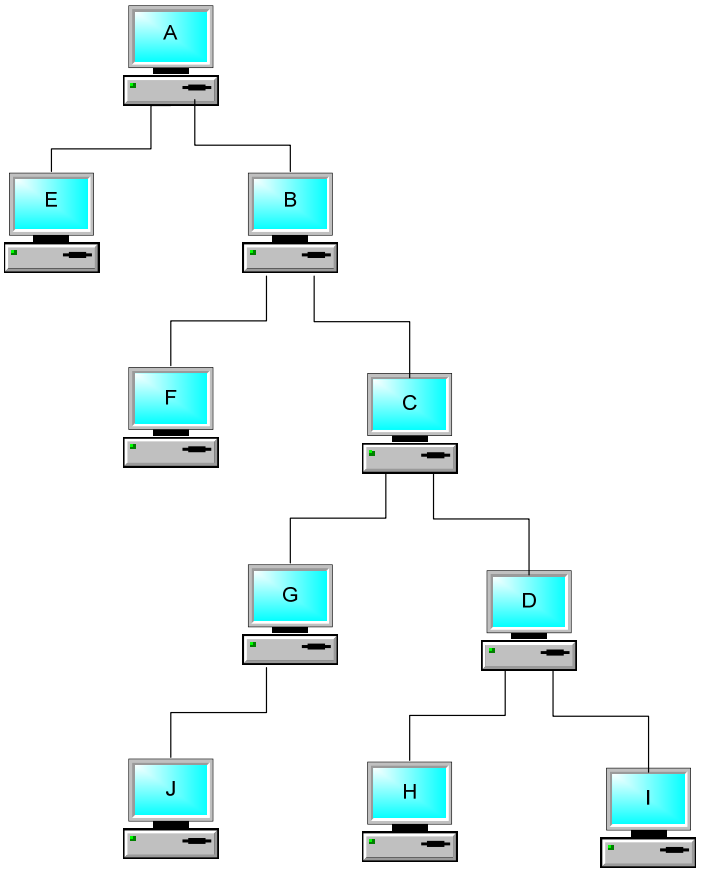


Figure 7.5.2.3 Imperfect Match Unbalance

As explained in the previous section, node A initiates the balancing algorithm and the value of δ_A is computed as 4. The balancing algorithm searches the branch with the largest tree size to identify a sub-tree whose size matches the index value δ_A (4). Node C detects that its children have tree of size less than the index value δ_A (4). Node C sends a message to its child node D, the node with the largest tree size, requesting it to join the left child of the source node, node E. The network structure after this round of balancing in presented in Figure 7.5.2.4 below.

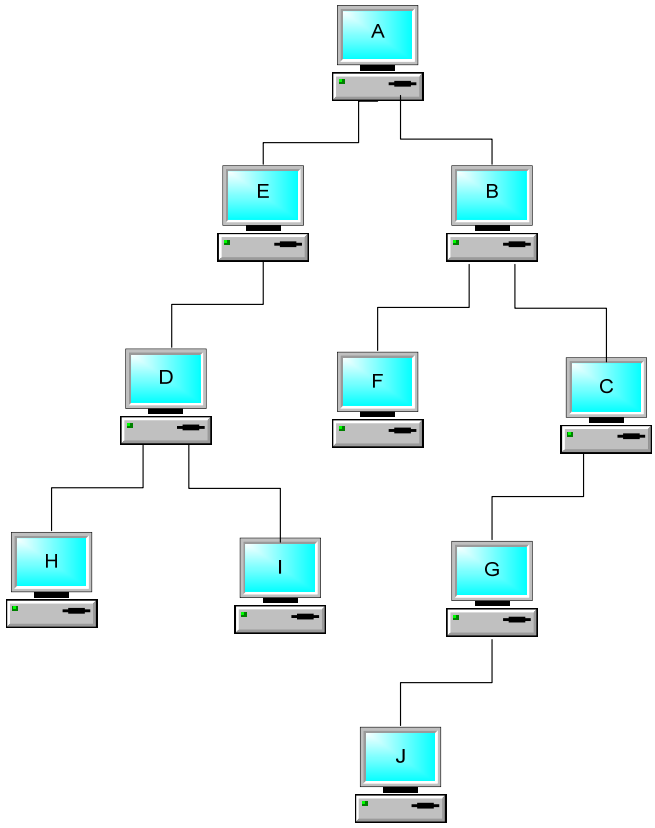


Figure 7.5.2.4 Imperfect Match Balanced Network

After this initial round of balancing, the network is in a 'near' balanced state with respect to node A. However, upon cursory observation it may appear that the network is still unbalanced with respect to nodes E, B and C. The new unbalance index computed for these nodes, using the equation specified above, results in a $\delta_{(E, B, \text{and } C)}$ value of either 1 or 2. As the threshold for the index value δ_T is set to 3, no further balancing is performed in this network.

8. Results

The scope of this research effort and the results obtained are presented in this chapter. To better illustrate the results achieved, the test setup is also discussed in this chapter followed by the test results. The search results obtained from a balanced peer-to-peer RAN resource directory system are compared with the results obtained from an unbalanced network.

8.1. Implementation Scope

In order to accomplish the goal of this research effort the following were implemented.

- Designed a RAN resource directory system that was incorporated into the peer units (individual peers) to form a RAN system that supports disparate resource types
- Incorporated flexibility in the resource directory to enable dynamic management (addition, removal, and modification) of active resource types shared in RAN system
- Implemented a balanced binary tree structure in the RAN so as to reduce the hop number for the search messages

8.2. Disparate Resource Types

In this section, the different resource types used to demonstrate the RAN's ability to manage various resource types are presented. All the resource types listed were used in one or more test setups described in the following three sections. The different resource types used in this project are as follows:

- File
- File-Chunks

- Memory
- Storage
- Free-Storage
- Used-Storage
- Webservice

Figure 8.2.1 below provides a snapshot of the database associated with one peer unit in which multiple resource types were registered.

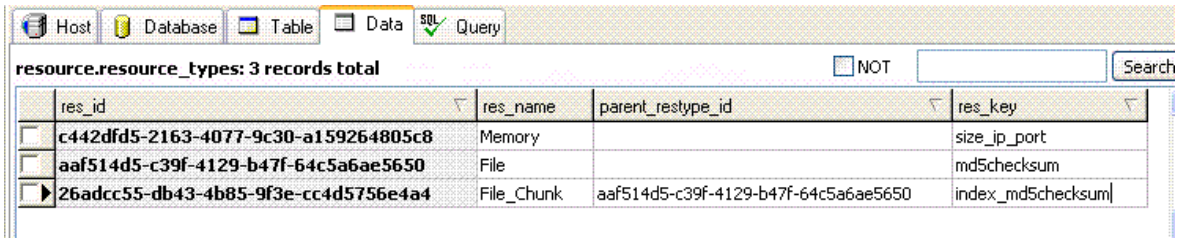


Figure 8.2.1 Database Snapshot of a Peer Unit

Each resource type has a set of defining attributes that can be modified. For example, for a resource type ‘Memory’, the defining attributes are ‘IP’, ‘Port’, ‘Speed’ and ‘Block size’. In addition to these attributes, new attributes can be included to this definition. Figure 8.2.2 shows the attributes that define all the resource types for the peer unit depicted in Figure 8.2.1.

resource.resource_type_property: 14 records total

id	property_name	property_type
Memory	ip	VARCHAR(125)
Memory	port	INT
Memory	speed	INT
Memory	block_size	INT
File	ip	VARCHAR(125)
File	port	INT
File	md5checksum	VARCHAR(125)
File	date	VARCHAR(125)
File	owner	VARCHAR(125)
File	name	VARCHAR(125)
File_Chunk	ip	VARCHAR(125)
File_Chunk	port	INT
File_Chunk	md5checksum	VARCHAR(125)
File_Chunk	index	INT

Figure 8.2.2 Attributes Set for Resource Types

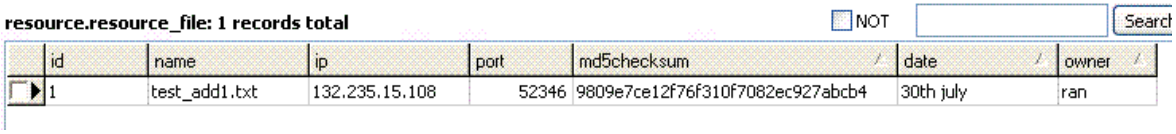
When these three resource types were added to the RAN resource directory, the resource directory created tables to hold the information about the registered resources for every resource type. Figure 8.2.3 shows the new tables created when the resource types were added to the resource directory.

Database resource: 7 table(s)

Table	Records	Size	Created	U...	Engine	Comment
conn_stat	0	33 KB	2006-03-29 12:47:26	N/A	InnoDB	InnoDB free: 11264 kB
resource_directories	0	17 KB	2006-03-29 08:47:33	N/A	InnoDB	InnoDB free: 11264 kB
resource_file	0	17 KB	2007-09-04 15:07:43	N/A	InnoDB	InnoDB free: 11264 kB
resource_file_chunk	1	17 KB	2007-09-04 15:07:43	N/A	InnoDB	InnoDB free: 11264 kB
resource_memory	0	17 KB	2007-09-04 15:07:43	N/A	InnoDB	InnoDB free: 11264 kB
resource_type_property	11	17 KB	2006-03-29 08:47:33	N/A	InnoDB	InnoDB free: 11264 kB
resource_types	3	17 KB	2006-03-29 08:47:33	N/A	InnoDB	InnoDB free: 11264 kB

Figure 8.2.3 Resource Type Tables

Figures 8.2.4 shows the table structure of the resource type ‘File’ while Figure 8.2.5 shows the table structure of the resource type ‘Memory’. The respective defining attributes are the columns shown in the figures.

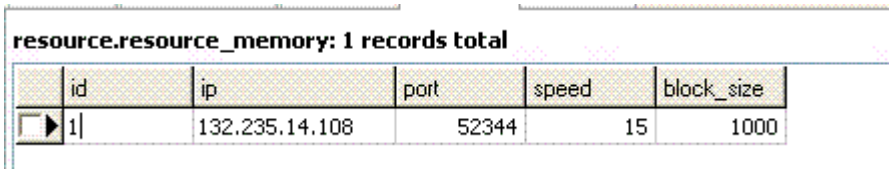


resource.resource_file: 1 records total

NOT Search

	id	name	ip	port	md5checksum	date	owner
▶	1	test_add1.txt	132.235.15.108	52346	9809e7ce12f76f310f7082ec927abcb4	30th july	ran

Figure 8.2.4 Resource Type Table for File



resource.resource_memory: 1 records total

	id	ip	port	speed	block_size
▶	1	132.235.14.108	52344	15	1000

Figure 8.2.5 Resource Type Table for Memory

By creating a RAN with these disparate resource types it has been possible to demonstrate that different resource types could be shared among peers in the RAN system.

8.3. Three System RAN Resource Directory Service

In this section, the several stages in the balancing algorithm involving three systems are presented. The three systems are as follows:

- 132.235.15.108#4544#4545#6767 – System A
- 132.235.15.108#4543#4544#6768 – System B
- 132.235.15.108#4542#4543#6769 – System C

During start up, these systems are connected to form an unstructured network and it is not possible to control to order in which they join the network. In the test scenario, the

three systems are connected as shown in Figure 8.3.1 below. The system are referred as ‘System A’, ’System B’ and ‘System C’ for ease of explanation.

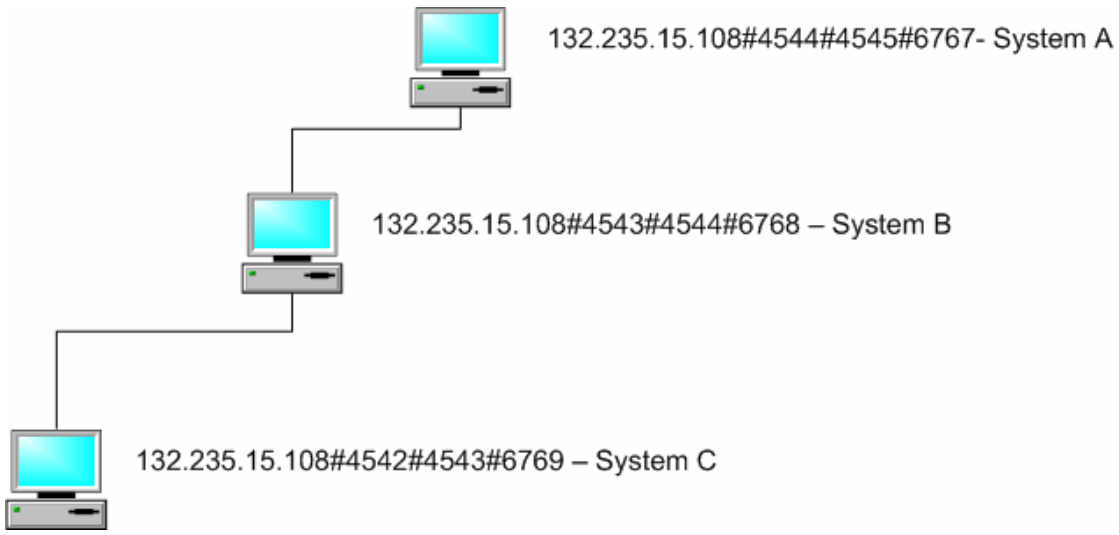


Figure 8.3.1 Three System RAN RD – Unbalanced Network

The snap shot of the connection status of the three systems in the database is presented in Table 8.3.1 below. The database holds all the information that is relevant for all the active RAN resource directories in the network. Some of the high-level attributes are captured in the following snap shot.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A			System B	3	0	2	**B**
System B	System A		System C	2	0	1	**B**
System C	System B			1	0	0	**B**

Table 8.3.1 Three System RAN RD – Connection Status

Table 8.3.1 indicates that the peers are connected in a linear manner and their status is B (Balanced) when they join the network. However, some of the information is shared

between the peers, the global status of all the peers in the network not known to the individual peers. The CONN_SIZE, CONN_RIGHT_SIZE, CONN_LEFT_SIZE are the attributes based on which the balancing algorithm determines the peer status to be either balanced or unbalanced.

When every peer is connected to each other and transfers its local knowledge to the parent peer, the system is updated with the correct size information. In the snapshot provided in Table 8.3.2 below, the peers have updated their information and hence System A's status is set to 'U' (Unbalanced). After the status is set, the algorithm determines the peer or peers that need to be moved to restore the network balance.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A			System B	3	0	2	**U**
System B	System A		System C	2	0	1	**B**
System C	System B			1	0	0	**B**

Table 8.3.2 Three System RAN RD – Updated Connection Status

The balancing algorithm explained in the chapter 5 details the several statuses the peer can be in. In the following scenario, the status of the source peer is set to 'S' (Source) and the other nodes are set to 'N' or 'W' based on their current status. After the peers communicate their status to the source, their status is reset to 'B' or 'N'.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A			System B	3	0	2	**S**
System B	System A		System C	2	0	1	**W**
System C	System B			1	0	0	**N**

Table 8.3.3 Three System RAN RD – Transitory State 1

In the following stage, peer ‘System C’ is moved to ensure that the network structure is balanced. The transitory state snap shot, provided in Table 8.3.4, is the stage where the move was performed, but the information has not been transmitted to the source node. As these transactions are not monolithic, there can be a lag between state transitions encountered by the peers and their corresponding status in the database. This could cause ‘Phantom Network States’ where the state of a peer in the table is inconsistent to its actual state. To minimize this condition, a time-delay logic is incorporated into the balancing algorithm. For the test scenario, the time delay was set to 5 minutes. Hence, for the balancing algorithm to be activated the peers need to maintain the unbalanced state for more than the time-delay period.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System C	System B	4	1	2	**B**
System B	System A			2	0	0	**B**
System C	System A			1	0	0	**B**

Table 8.3.4 Three System RAN RD – Transitory State 2

Thus, the final state of the peers is as presented in Table 8.3.5. The peers form a balanced network as portrayed in Figure 8.3.2.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System C	System B	3	1	1	**B**
System B	System A			2	0	0	**B**
System C	System A			1	0	0	**B**

Table 8.3.5 Three System RAN RD – Balanced State

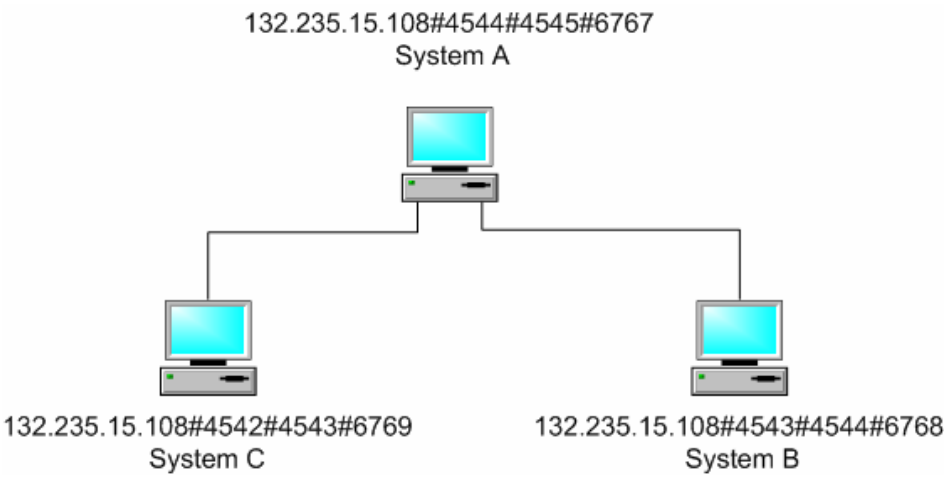


Figure 8.3.2 Three System RAN RD – Balanced Network

8.4. Five System RAN Resource Directory Service

In this section, the several stages in the balancing algorithm involving five systems are presented. The five systems are as follows:

- 132.235.15.108#4544#4545#6767 - System A
- 132.235.15.108#4543#4544#6768 - System B
- 132.235.15.108#4541#4543#6770 - System C
- 132.235.15.108#4540#4543#6771 - System D
- 132.235.15.108#4542#4544#6769 - System E

As explained in the previous section, these systems are connected to form an unstructured network. In the test scenario, they are connected as shown in Figure 8.4.1 below. The snapshot of the connection status and the high-level status attributes of the five systems in the database are presented in Table 8.4.1 below.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A				1	0	0	**B**
System E				1	0	0	**B**
System B				1	0	0	**B**
System D				1	0	0	**B**
System C				1	0	0	**B**

Table 8.4.1 Five System RAN RD – Connection Status

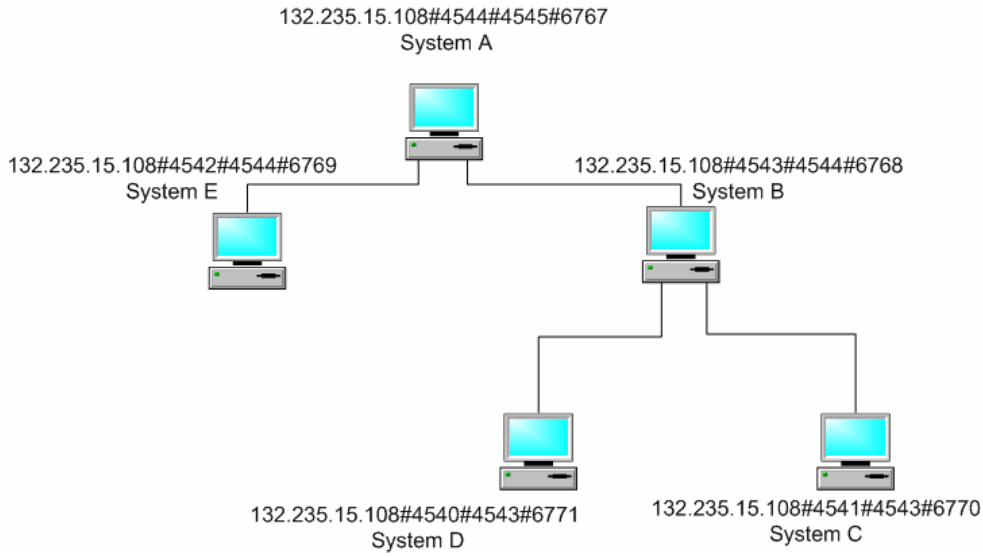


Figure 8.4.1 Five System RAN RD – Unbalanced Network

Table 8.4.2 is the snapshot of the peers' connection status in the database after all systems have connected and updated the connection information. 'System A' detects that it is unbalanced and sets its status to 'U'.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System E	5	3	1	**U**
System E	System A			1	0	0	**B**
System B	System A	System C	System D	3	1	1	**B**
System D	System B			1	0	0	**B**
System C	System B			1	0	0	**B**

Table 8.4.2 Five System RAN RD – Updated Connection Status

When the status of ‘System A’ is set to U, the balancing algorithm waits for the preset time and starts the balancing process. The status of the source peer is set to ‘S’ (Source) and the other nodes are set to ‘N’ or ‘W’ based on their current status. After the peers communicate their status to the source, their status is reset to ‘B’ or ‘N’ as shown in Table 8.4.3 below.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System E	5	3	1	**N**
System E	System A			1	0	0	**N**
System B	System A	System C	System D	3	1	1	**N**
System D	System B			1	0	0	**N**
System C	System B			1	0	0	**N**

Table 8.4.3 Five System RAN RD – Initiation of Balancing

When the status of all the participating peers’ is set to N, the balancing algorithm is initiated. The leaf peer’s status remains ‘N’ until the balancing is progressively completed in a top-down approach. At each parent level, the tree is checked for balance and if the tree is not balanced, the balancing logic is implemented on the sub-tree. During the balancing process, the peers communicate their status to the source and then reset their status to either ‘B’ or ‘N’. Table 8.4.4 shows the status of the peers during balancing.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System E	5	3	1	**S**
System E	System A			1	0	0	**B**
System B	System A	System C	System D	3	1	1	**W**
System D	System B			1	0	0	**N**
System C	System B			1	0	0	**N**

Table 8.4.4 Five System RAN RD – Transitory State 1

In the following stage, ‘System D’ was moved to ensure that the network structure is balanced. The transitory state snap shot, provided in Table 8.4.5, is the stage where the move was performed, but the information has not been transmitted to the source node ‘System A’. To minimize ‘Phantom Network States,’ a time-delay of 5 minutes is incorporated into the balancing algorithm.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System E	6	3	2	**B**
System E	System A		System D	2	0	1	**B**
System B	System A	System C		2	1	0	**B**
System D	System B			1	0	0	**N**
System C	System B			1	0	0	**N**

Table 8.4.5 Five System RAN RD – Transitory State 2

Thus, the final state of the peers is as presented in Table 8.4.6. The peers form a balanced network as portrayed in Figure 8.4.3.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System E	5	2	2	B
System E	System A		System D	2	0	1	B
System B	System A	System C		2	1	0	B
System D	System B			1	0	0	B
System C	System B			1	0	0	B

Table 8.4.6 Five System RAN RD – Balanced State

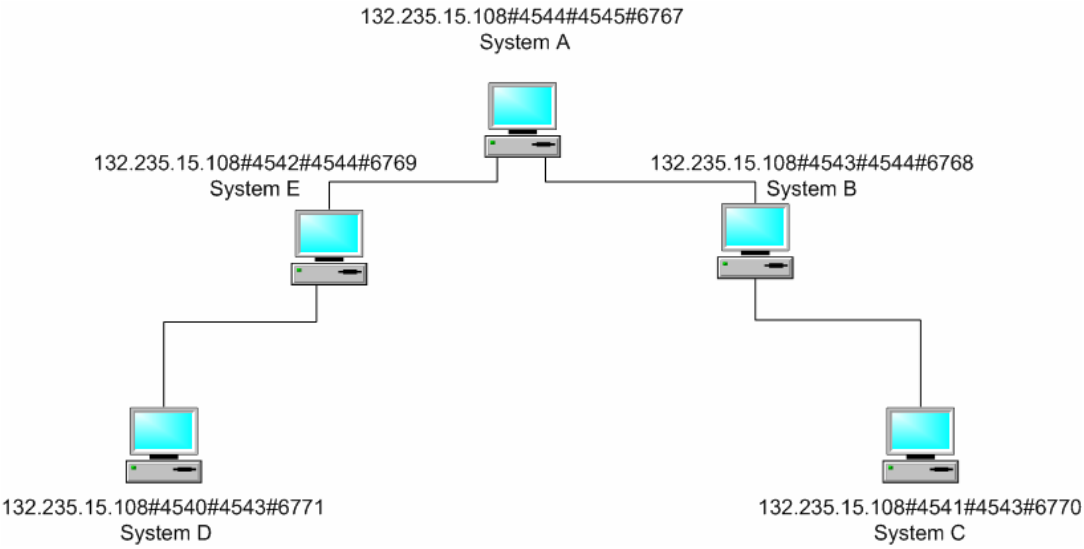


Figure 8.4.2 Five System RAN RD – Balanced Network

8.5. Seven System RAN Resource Directory Service

In this section, the several stages in the balancing algorithm involving seven systems are presented. The seven systems are as follows:

- 132.235.15.108#4544#4545#6767 - System A
- 132.235.15.108#4543#4544#6768 - System B
- 132.235.15.108#4542#4544#6769 - System C
- 132.235.15.108#4541#4543#6770 - System D
- 132.235.15.108#4540#4543#6771 - System E
- 132.235.15.108#4539#4541#6772 - System F
- 132.235.15.108#4538#4541#6773 - System G

As explained in section 8.1, these systems are connected to form an unstructured network. In the test scenario, they are connected as shown in Figure 8.5.1 below.

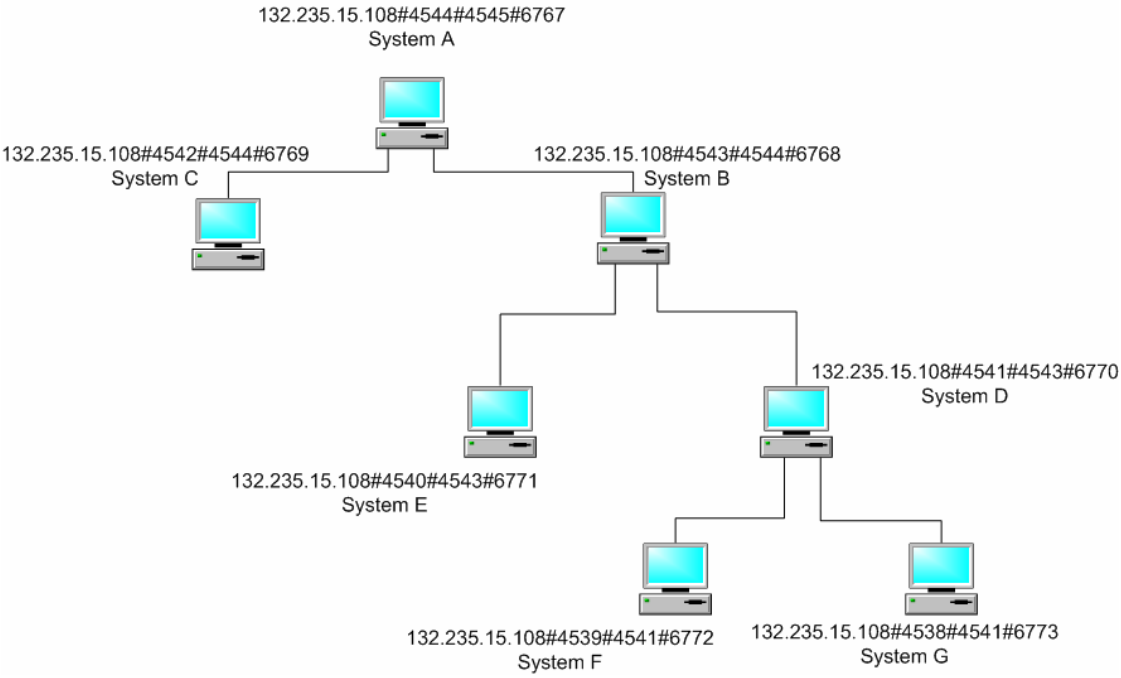


Figure 8.5.1 Seven System RAN RD – Unbalanced Network

The snapshot of the connection status and the high-level status attributes of the seven systems in the database are presented in Table 8.5.1 below.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A				1	0	0	**B**
System C				1	0	0	**B**
System B				1	0	0	**B**
System E				1	0	0	**B**
System D				1	0	0	**B**
System F				1	0	0	**B**
System G				1	0	0	**B**

Table 8.5.1 Seven System RAN RD – Connection Status

Table 8.5.2 is the snapshot of the peers' connection status in the database after all systems have connected and updated the connection information. 'System A' and 'System B' detect that they are unbalanced and set their status to 'U'.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System C	7	5	1	**U**
System C	System A			1	0	0	**B**
System B	System A	System D	System E	5	3	1	**U**
System E	System B			1	0	0	**B**
System D	System B	System G	System F	3	1	1	**B**
System F	System D			1	0	0	**B**
System G	System D			1	0	0	**B**

Table 8.5.2 Seven System RAN RD – Transitory State 1

In this test case, as two subsystems are unbalanced, the balancing algorithm on the higher-level peer is on-hold until the lower-level peer completes balancing and is ready to change state to initiate further balancing. With this approach, the balancing will work with a subsystem that is already balanced and hence it is easier to resolve which peers can be moved to balance the network state.

Table 8.5.3 shows the lower peers are ready for balancing. However, the status on 'System A' would be set to 'N' only after the state of 'System B' changes to 'W' and sends a message to 'System A' indicating its availability for balancing. Subsequent to this event, the status of both 'System A' and 'System B' are set to 'N'.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System C	7	5	1	**S**
System C	System A			1	0	0	**N**
System B	System A	System D	System E	5	3	1	**N**
System E	System B			1	0	0	**N**
System D	System B	System G	System F	3	1	1	**N**
System F	System D			1	0	0	**N**
System G	System D			1	0	0	**N**

Table 8.5.3 Seven System RAN RD – Transitory State 2

Table 8.5.4 is a snap shot of the peers' network state where the lower sub-tree is balanced. For this experiment, the balancing threshold (δ_T) was set to one. However, the balancing threshold (δ_T) could be set to higher value so that under certain circumstances, where the balancing is costly for the operation of the system, the balancing can be effectively minimized or turned off.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System C	7	5	1	**S**
System C	System A			1	0	0	**N**
System B	System A	System D	System E	5	2	2	**B**
System E	System B		System F	2	0	1	**B**
System D	System B	System G		2	1	0	**B**
System F	System E			1	0	0	**N**
System G	System D			1	0	0	**N**

Table 8.5.4 Seven System RAN RD – Transitory State 3

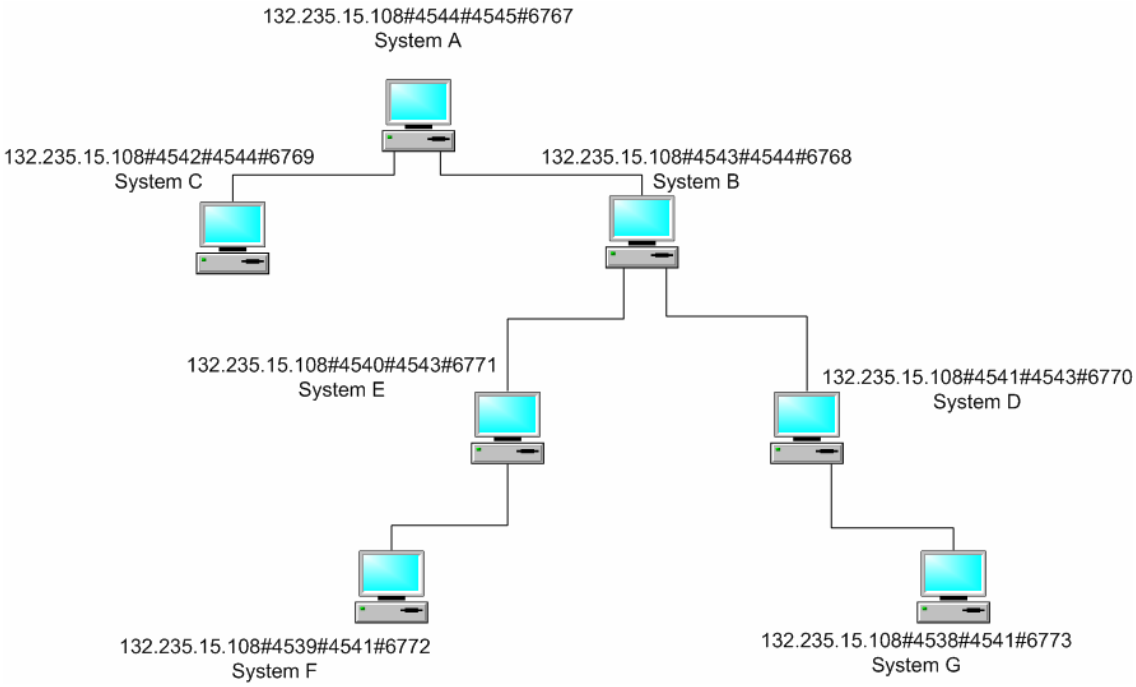


Figure 8.5.2 Seven System RAN RD – Balancing State 1

Through this experiment, it is illustrated that when multiple peers are unbalanced, the order of balancing is dependent on the first unbalanced node that locks down the child peers. When balancing is complete on the smaller subsets, the balancing algorithm balances the peers placed hierarchically above the balanced peer.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System C	7	5	1	N
System C	System A			1	0	0	N
System B	System A	System D	System E	5	2	2	N
System E	System B		System F	2	0	1	N
System D	System B	System G		2	1	0	N
System F	System E			1	0	0	N
System G	System D			1	0	0	N

Table 8.5.5 Seven System RAN RD – Transitory State 4

As indicated in Table 8.5.5, a snapshot of system states at the end of the first round of balancing, the revised network continues to be in an unbalanced state. Hence, the balancing algorithm is triggered and cycle repeats till the network reaches complete balance with respect to the triggering node ‘System A’ as shown in Table 8.5.6 and Figure 8.5.3 below.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System C	7	3	3	**B**
System C	System A		System E	3	0	2	**B**
System B	System A	System D		3	2	0	**B**
System E	System C		System F	2	0	1	**B**
System D	System B	System G		2	1	0	**B**
System F	System E			1	0	0	**B**
System G	System D			1	0	0	**B**

Table 8.5.6 Seven System RAN RD – Balancing State 1

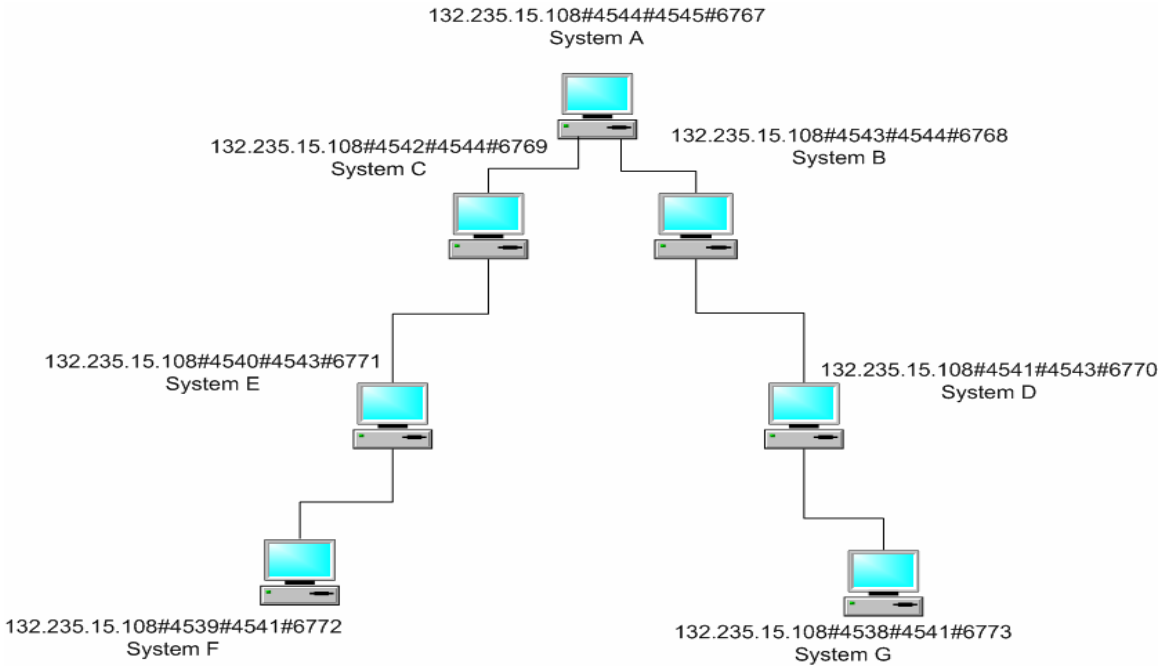


Figure 8.5.3 Seven System RAN RD – Balancing State 2

When the network is balanced with respect to ‘System A’, the balancing algorithm progresses the peers hierarchy and the cycle of balancing repeats until the network settles in a state of complete balance as indicated in Figure 8.5.4 and Table 8.5.7.

CONN_UUID	CONN_PARENT	CONN_R_UUID	CONN_L_UUID	CONN_SIZE	CONN_R_SIZE	CONN_L_SIZE	STATUS
System A		System B	System C	7	3	3	**B**
System C	System A	System F	System E	3	1	1	**B**
System B	System A	System D	System G	3	1	1	**B**
System E	System C			1	0	0	**B**
System D	System B			1	0	0	**B**
System F	System C			1	0	0	**B**
System G	System B			1	0	0	**B**

Table 8.5.7 Seven System RAN RD – State of Complete Balance

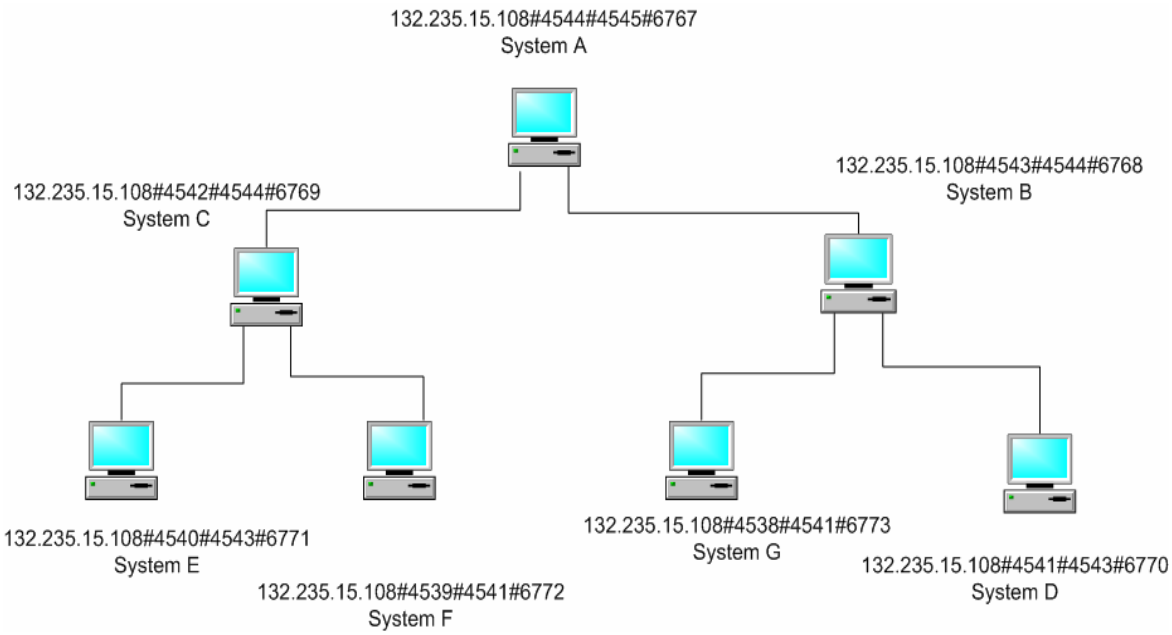


Figure 8.5.4 Seven System RAN RD – Balancing State 2

8.6. RAN Test Setup

In this section, the systems and the setup used to construct the RAN system for the experiments are described.

All peers for the experiments were created by instantiating the resource directory with a designated set of socket entities (server and client) that are used to receive and send the messages between the peers. During resource directory initialization, all peers except the knowledge peer were initialized using a common MySql database. The knowledge peer was initialized using a separate MySql database that contained the results of the search queries used in the experiments. In Figure 8.6.1 below, the shaded portion represents the peers that are on a common MySql database.

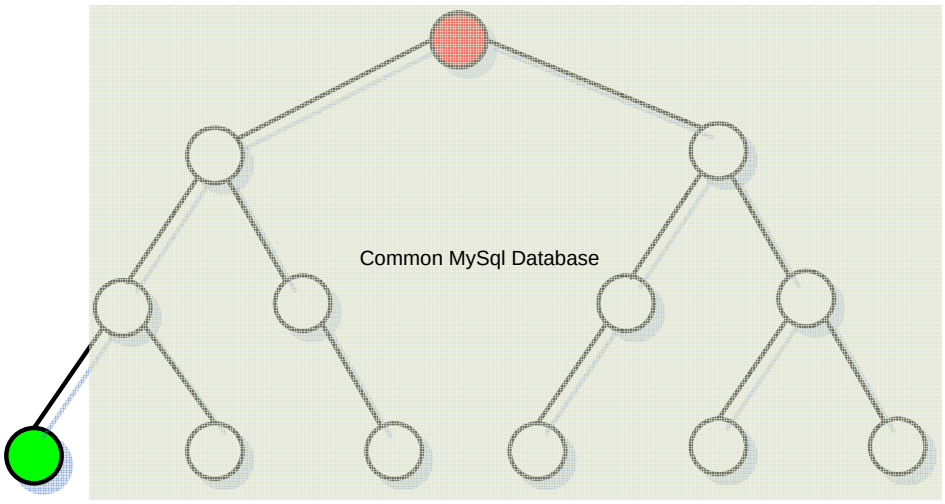


Figure 8.6.1 Test Case Peer Structure

The test cases used for the worst-case network structure were constructed by connecting the peers in a pure linear fashion. The unbalanced test cases were created by instantiating the peers in random order and connecting them to form the network. This case

is representative of the Gnutella, an unstructured network, where the peers are connected in no specific order with a peer connectivity of 3 or less. Though these test cases were created using the same application as that of the balanced test case, the balancing algorithm was inactivated by setting the time to trigger the balancing process to a large value. The test cases used for the balanced network structure were created by following the same process as that of the unbalanced test cases and, additionally, activating the balancing process. The network structure created at the end of the balancing process was used for the experiments.

The general system specifications used for the experiments conducted are presented in Table 8.6.1 below.

Operating System	Windows XP
Processor	M Processor 1.7 GHz, 598 MHz,2.00 GB RAM M Processor 1200 MHz, 789 MHz, 376 MB
Database version	MySql version 5.0.1
Network	Wireless/Ethernet 54 Mbps/100 Mbps
Java version	1.4.2 / 1.5.0

Table 8.6.1 System Specifications for Experiments

8.7. Search Results

The test setups described in the previous sections were used to demonstrate the benefits of having a balanced binary tree structure for the peer units in the RAN implementation. To produce quantifiable results, specific locations in the test network were assigned to the source node (where the query is initiated) and the knowledge node (where the resource to the query is resident). The tests were performed on networks of four different sizes; these were 3-, 5-, 7-, and 13-node networks. For every network, three

different configurations were selected to simulate network structures that would result in a performance variation. With this approach, the results obtained are based on analyzing the performance of the RAN resource directory on a sample size of 12.

Figures 8.7.1 through 8.7.3 below are used to provide the reader with an illustration of the samples that were used for the different configurations tested on the 13-node network. In all the following figures, the node colored in red is indicative of the source node and the green-colored node is the knowledge node.

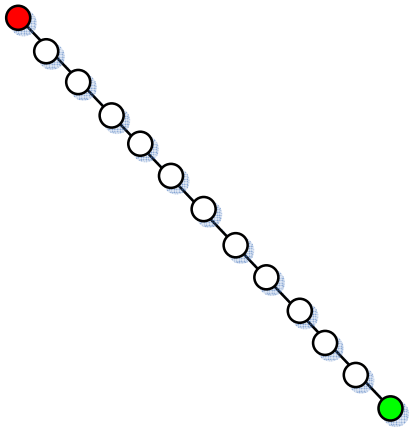


Figure 8.7.1 13-Node Network – Worst Case

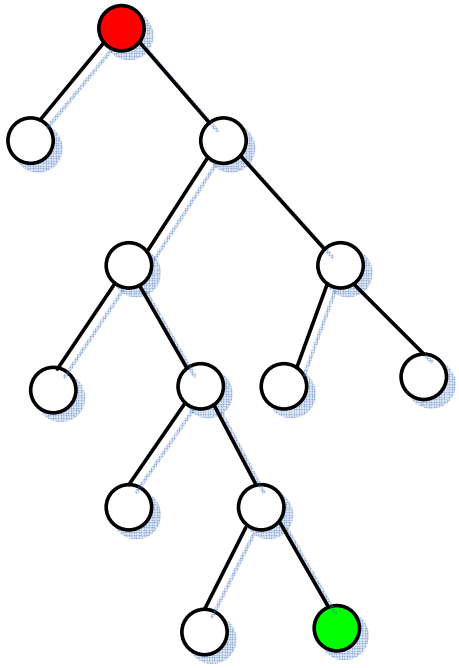


Figure 8.7.2 13-Node Network – Unbalanced Test Case (Gnutella)

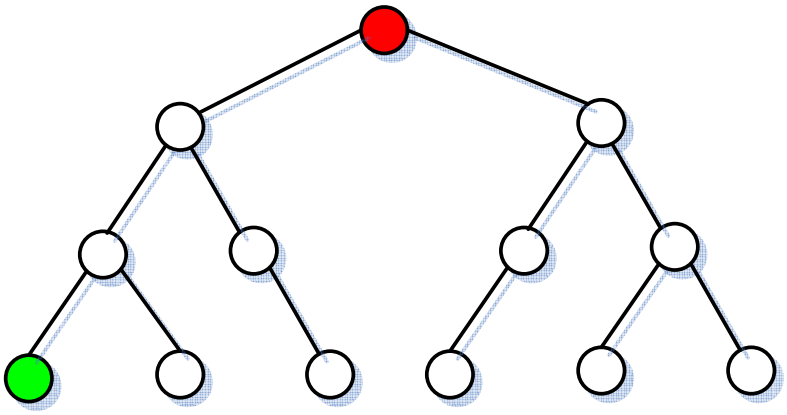


Figure 8.7.3 13-Node Network – Balanced Test Case (RAN)

The ‘Worst Case’ is the network configuration where each node has exactly one child and the maximum number of nodes separates the source and knowledge nodes. The ‘Test Case (Unbalanced)’ is the network configuration where the nodes were allowed to join and no balancing logic was implemented. The ‘Test Case (Balanced)’ is the network

configuration where the nodes were allowed to join and the balancing logic was implemented on these nodes. Table 8.7.1 and Table 8.7.2 are the data captured during the test runs on the 12 samples.

No. of Nodes	**Hop No.**		
	Worst Case	**Test Case Unbalanced (Gnutella)**	**Test Case Balanced (RAN)**
3	2	2	1
5	4	2	2
7	6	3	2
13	12	5	3

Table 8.7.1 Comparison of Search Results – Hop Number

No. of Nodes	**Time Taken For Search (ms)**		
	Worst Case	**Test Case Unbalanced (Gnutella)**	**Test Case Balanced (RAN)**
3	70	66	30
5	144	64	72
7	228	120	74
13	600	225	123

Table 8.7.2 Comparison of Search Results – Time Taken

In Figure 8.7.4, the trend colored in red is indicative of the test results for the 'Worst Case' configuration whereas colors blue and green are indicative of test results of the 'Gnutella like Test Case (Unbalanced)' and the 'RAN Test Case (Balanced)' configurations respectively.

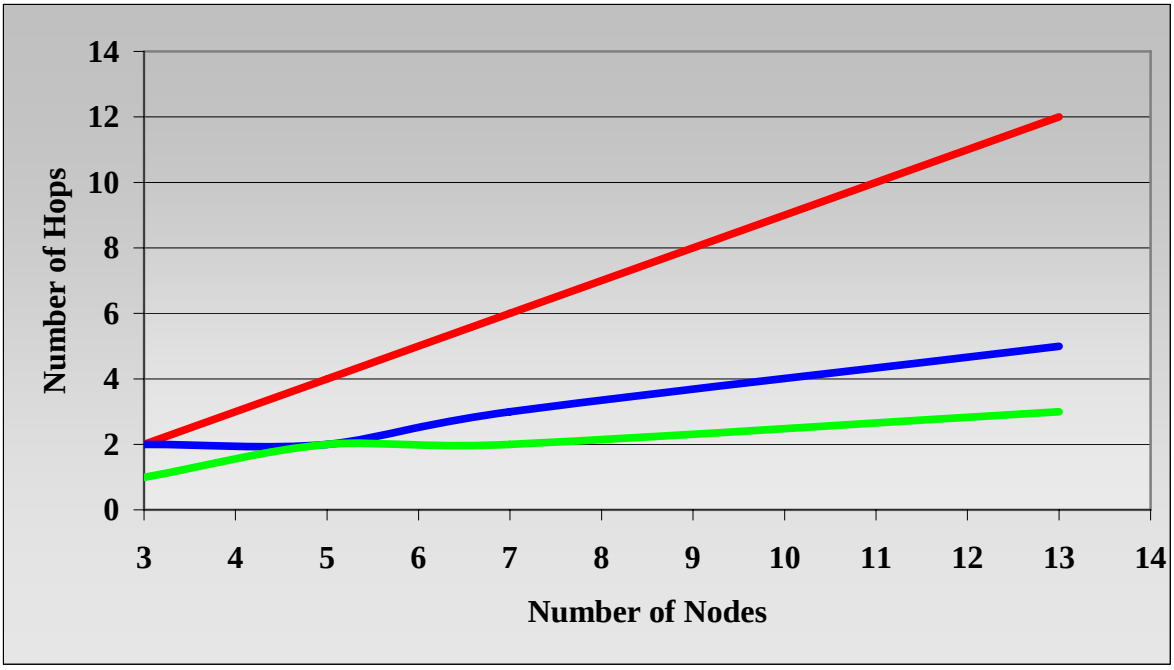


Figure 8.7.4 Number of Nodes vs. Number of Hops

Figure 8.7.4 indicates that as the number of nodes in the network increases, the number of hops taken for a search may rapidly increase in a linearly structured network. In addition, when comparing the results obtained for the number of hops taken for a search in an unbalanced network (comparable to Gnutella) with that of a balanced network, the hop number in an unbalanced network tends to be mostly greater than the balanced network. In addition, the difference in the hop number between the two network configurations tends to increase as the network size increases.

By extrapolating the results, it is reasonable to expect that for large networks, the hop numbers can be large and the number of messages can differ by orders of magnitude. For a realistic peer-to-peer system, it is common for the network to have a large number

of nodes (possibly, this can be several hundred). Implementing a balanced binary tree algorithm in a RAN like network can improve the search performance.

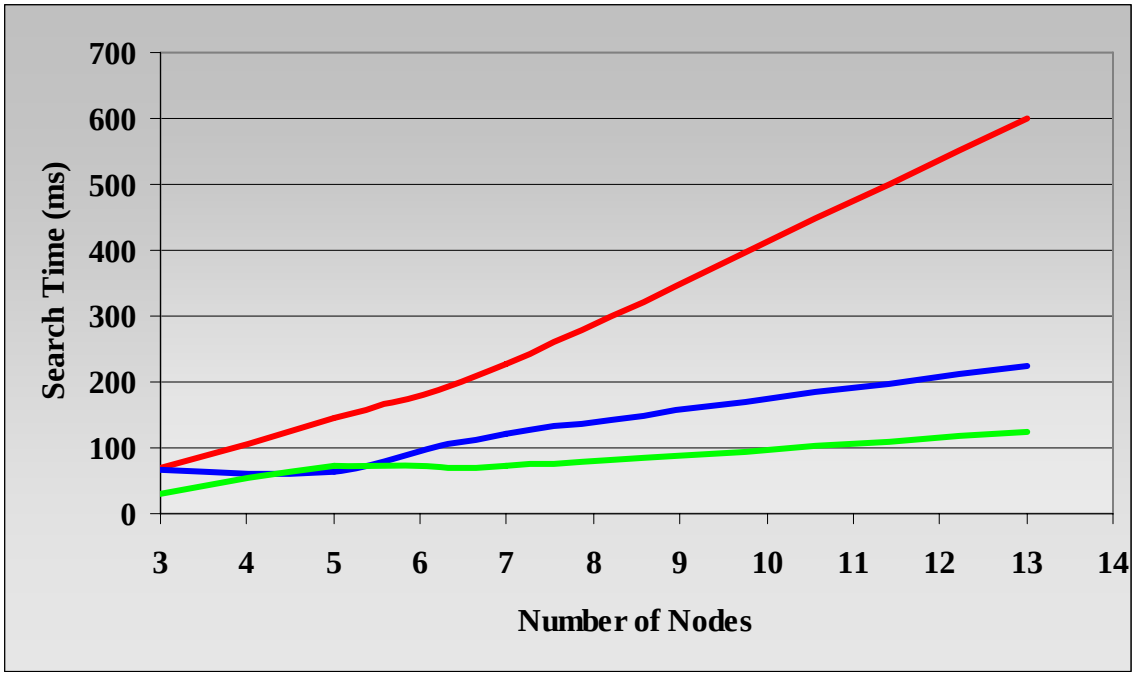


Figure 8.7.5 Number of Nodes vs. Search Time

Figure 8.6.5 uses the same color code as that of the previous figure. By analyzing this chart, it can be concluded that as the network complexity increases the time taken to complete a search also increases. However, the rate of this increase with increase in network size is relatively less for a balanced network when compared with an unbalanced or poorly structured network. It is also observed that the increase in search time is nonlinear – this could be due to the fact that the number of message exchanges in the network increases at a higher rate as the network size increases. The messages are mostly protocol messages that are exchanged between the peers. These messages, discussed in

detail in section 5.5, are essential for the proper functioning and maintenance of the RAN system.

From an end-user perspective, the search performance of a peer-to-peer system is, in most instances, associated with the time taken to return the results. This search time is proportional to the number of nodes traversed before the query message reaches the knowledge node. As the height of the network structure is a minimum for a balanced network structure, the number of hops taken is minimal. Hence, the search time on a balanced network structure is less when compared to that of an unstructured or unbalanced network structure.

8.8. Summary of Results

As discussed in sections 8.2 and 8.7 above, it has been possible to demonstrate the following:

- RAN resource directory has the ability to support disparate resource types and dynamically manage the resource types or revise existing resource type definitions
- By implementing a balanced binary tree algorithm for the peer units in the RAN, the search is improved

9. Related Work

In this chapter, the various peer-to-peer network implementations are discussed and the general characteristics of these peer-to-peer systems are compared with RAN.

9.1. NINJA SDS

SDS – Service Discovery Service – is a service that provides a listing of all the available services in a network to the clients. NINJA SDS [28] functions as a 'middleman' where the service providers can list or publish all their available or existing services and the clients can formulate search queries to locate the required services. Additionally, NINJA SDS also ensures that the security of the clients and the providers are maintained. This is performed by controlling the services that are listed based on the providers service capability. Security is further enhanced by encrypting all messaging in this platform in addition to client and provider authentication. In this platform, services are applications that can interact with the clients to perform the client-requested operations.

The information stored in NINJA SDS is of two types: the first is the information related to all services already running and their locations, and the second is the list of all available services. To ensure that the messages are platform independent and to be able to validate the service descriptions against a default schema, all messages, including service descriptions and client queries, are exchanged using XML data format. The schemas in the form of DTD (Document Type Definition) allows for changes in the service description format. The SDS uses announcement based messaging, as the system can comprise of several thousand services and servers. By using this technique periodically, it is possible to ensure that system is responsive to service or components faults. The

platform is designed to be scalable by allowing the SDS server to spawn a child server if the parent server is overloaded.

The SDS platform consists of clients, providers and SDS servers. The SDS servers send multicast service announcements in the network while the providers listen to these messages to identify their SDS server. Upon identifying its SDS server, the provider's description is multicast to the server. The client uses ARMI – Authenticated Remote Method Invocation – to communicate with the servers. The client submits its query to the SDS servers using an XML message that also includes its access rights. Based on the query, the server returns either the service information that is an exact match or the list of possible services. If the server fails to find the requested services, the message is forwarded on to other SDS servers.

9.1.1. Comparison with RAN:

This section discusses the various advantages and disadvantages of the RAN over NINJA SDS.

Advantages of RAN:

NINJA SDS uses intermediate brokers to communicate between the client and service provider whereas in RAN there are no intermediaries. The peers are directly connected to each other and hence the search can be efficient in RAN due to the less number of peers that the query interacts with to obtain the results. Any service that has an interface to connect with other systems is considered as a resource in NINJA SDS; however, this feature is a subset of RAN's ability to support multiple resource types. The RAN peers are equal peers and they can perform similar operations. However, in NINJS

SDS there are specific peers that act as SDS Servers and have unique capabilities thus increasing the vulnerability of the system to server failures.

Disadvantages of RAN:

In RAN, any peer can be a seed peer and performs the important task of initial balancing by directing the new peer to join the appropriate peers. In the advent of seed peer failures, new peers cannot join the network through the failed seed peer. However, this is not a huge disadvantage as there are several seed peers in RAN system. Ninja SDS uses RMI (JAVA SSL) as its security and hence the messages are authenticated and secured whereas RAN lacks this security.

9.2. JESA

JESA – Java Enhanced Service Architecture – is a service discovery protocol for ad hoc networks comprising of disparate systems with limited resources such as embedded or mobile devices [35]. The fundamental concept behind this service platform is sharing of services amongst disparate devices with limited computational capability. This service discovery process, based on certain entity (akin to nodes in RAN resource directory) attributes like initial knowledge, count, and its network activity, can be classified into different categories. Based on the entity's initial knowledge this service can be classified as preconfigured or non-configured. In the pre-configured discovery service, every entity has prior knowledge that is required to join the ad hoc networks. In the non-configured setup, the entities have no prior knowledge about the network services. Most devices in the ad hoc networks belong to this category.

In addition, JESA can be classified based on the client-provider relationship. In the 'location-aware' and 'immediate' types, there is a direct relation between the provider

and the client. The provider interacts directly with the client and supplies all the required information. In the 'mediated' type of client-provider relationship, both the client and the provider deals with the brokers, a special type of service provider, during the discovery process and does not directly interact with each other. Some of these discovery process types can be further classified based on request origin and level of provider knowledge.

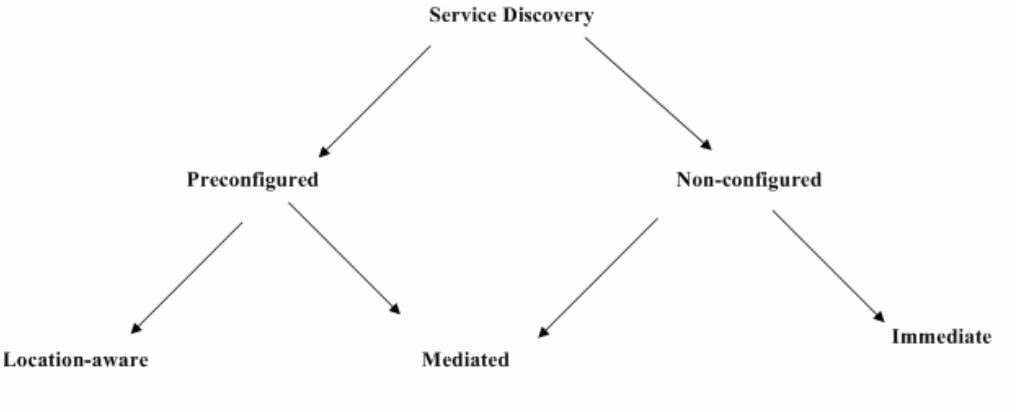


Figure 9.2.1 JESA Service Discovery Classification

The first step in the discovery process is to identify the provider and to obtain the data associated with the provider location. This is accomplished by use of PLP – Provider Location Protocol – messages. In this step the PARP – Proxy/Attribute Request protocol – messages are used obtain more information about the service proxy or service attributes. Through these steps, the client is able to identify the provider, connect to, and use the service from the providers.

9.2.1. Comparison with RAN:

This section discusses the various advantages and disadvantages of the RAN over JESA.

Advantages of RAN:

In JESA, the peers are mediated with the service providers through brokers in the network. In RAN, however all peers are equal and can easily communicate with each other; after a query, the location of the resource is known for further transactions between the peers. The lack of brokers creates a transparency in the RAN system. The resource that is shared in a JESA network is limited to mobile ad-hoc services; this cannot be revised. Whereas, RAN supports multiple resource types and has the ability to add or remove resource types. JESA uses UDP messages to communicate between the peers whereas RAN uses ASCII messages. As the UDP messages are inherently unreliable, RAN's approach of using ASCII over socket connection is reliable.

Disadvantages of RAN:

In RAN, the discovery process is inherently more complex as the primary focus of the RAN system is to reduce the time taken to search for a resource in the network. However, in JESA, the focus is on the discovery of the devices in the ad hoc network and attributes' exchange.

9.3. GNUTELLA

Gnutella is a decentralized peer-to-peer file-sharing network that uses Gnutella protocol for its operations. In the Gnutella network, every peer is equal and is considered both a client and a server. These are referred to as 'Gnutella servents' [9]. These servents communicate with each other using the Gnutella protocol. The protocol consists of a descriptor that are used for communicating data among the servents and the rules that govern the descriptor exchange between the servents.

The different Protocol Messages in Gnutella are as follows:

9.3.1. Broadcast Messages

Ping: These messages are used to find other servents in the network and these messages may receive either one or more responses from other servents in the network.

Query: These messages are used to search the network for specific resources. The search criteria are provided in the query message in the form of strings. Additionally, the query message also contains specifications on the minimum response rate of the servents.

9.3.2. Back-Propagated Messages

Pong: These messages are created by the servents in response to a 'Ping' message. This message also includes the servent unique identifier, address of the servent and the information on the resources it shares in the network. This information contains the number of files and the file sizes of the shared resources.

QueryHit: These messages are created by the servent in response to 'Query' messages only when they meet the criteria specified in the 'Query' message. These messages contain the descriptors from the original 'Query' message, the IP address of the servent, its response rate and the search result. The search result consists of information that uniquely identifies the requested resources and its attributes such as file name and size.

9.3.3. Node-to-Node Messages

Push: These messages are generated by a servent in response to a 'QueryHit' message from a servent that is behind a firewall and does not allow incoming connections. This contains requesting servent's information that is needed to establish a TCP/IP connection. Upon receiving this information, the servent behind the firewall, tries to establish a TCP connection with the requesting servent.

Get: These messages are issued in response to a ‘QueryHit’ and are used to initiate the file download by establishing a direct connection between the servents. This file download is not part of the Gnutella protocol.

9.3.4. Comparison with RAN:

This section discusses the various advantages and disadvantages of the RAN over Gnutella.

Advantages of RAN:

As the Gnutella network is restricted to resources of type ‘File’ it is limited to RAN in the types of resources that can be shared in the network. In Gnutella, all clients, when installed in the system, participate in the queries and hence have little flexibility to change their role in the network. Hence, a portion of the system’s bandwidth is allocated for searches from other peers. However, in RAN, as the peers have the ability to decide their role in the network, only peers with resource directories that choose to participate in searches have bandwidths allocated. In addition, as Gnutella is an unstructured network and its search mechanism is based on 'flooding' the network with a search message, a large number of messages (large volume) are generated and propagated throughout the network. This results in a limitation on scalability on the Gnutella network. However, in RAN due to the balanced structure, the hop number is reduced.

Disadvantages of RAN:

In Gnutella, the peers are completely decentralized and failure of any node does not affect the network. Whereas in RAN, any peer can be a seed peer and performs the important task of initial balancing by directing the new peer to join the appropriate peers. In the advent of seed peer failures, new peers cannot join the network through the failed

seed peer. However, this is not a huge disadvantage as there are several seed peers in RAN system. In Gnutella, all peers have the freedom to connect to any peer based on the IP. In RAN, the seed peer and the current state of the peer network structure determines where in the network the new peer can connect.

9.4. SETI @ Home

The SETI @ HOME (Search for Extraterrestrial Intelligence) is a large-scale distributed project using BOINC [24] platform, which uses computers connected to the Internet. BOINC (Berkeley Open Infrastructure for Network Computing) is a non-commercial volunteer-based distributed computing platform. This commuting uses the processing power of the millions of personal computers connected to the network as a resource. This resource is used to perform the lengthy computation that is required by some of the projects.

The main goal of SETI@ Home project is to determine the existence of intelligent alien life forms by detecting the radio signals from space. These signals can be noises made from man-made electronics and hence huge processing power is needed to detect the narrow band intelligent signal from the huge data sent from the Arecibo Observatory. Due the vastness of the space and other factor, it becomes difficult to process the data without involving huge processing time and power.

The data collected from the telescope is divided into smaller work units that are then sent to the users who are requesting work units to process. The client application is mostly in a screen saver mode, where the processing is done when the PC is ideal. If the client machine finds any potential signal in the work unit it received, the work unit is sent

to the server. The server updates the scientific database with the results and also updates the user information.

The current SETI@ Home project only processes data from a small gamut of radio frequency from selected parts of the sky. The processing power of the voluteer computers have accounted for 437,000 years of CPU time for a total 4.3 x 1020 flop[4]. Thus making SETI @ Home the largest distributed computing project.

9.4.1. Comparison with RAN:

This section discusses the various advantages and disadvantages of the RAN over SETI@Home.

Advantages of RAN:

SETI@Home is a centralized peer-to-peer system sharing 'processing time' as its resource between the peers. As any centralized network, it is vulnerable to the single point failure. As SETI@Home is a huge project in distributed computing, backup servers are available to safe guard against single point failures. The peers that participate in the network cannot be authenticated and the work performed needs to be verified by processing the same work unit with another trusted peer. However in RAN, a preliminary form of authentication is in place, and the data is protected by controlling the access of the data through resource directories alone.

Disadvantages of RAN:

Unlike SETI@Home, the search results in RAN are not verfied and the credibility of any query result relies on the peers that registered the resource descriptor.

9.5. GRUB

The net information worth of the Internet is close to 500 thousand terabytes as of 2003 with a decadal growth of about 1000 times. This growth can be attributed to the large number of users and the size of the network. The ability to find the resources governs the usability of the system. The survey by Pew Internet and American Life Project in 2003 estimates that there were more than 72 million users online in a day [22] of which 40- 50 % search the Internet for information.

Several applications have been developed to find information from the Internet – these are popularly known as 'Search Engines' [35]. They help in minimizing the time taken to search the vast Internet for information. These search engines are one of the most visited sites in the Internet. To perform a search efficiently the search engines collect the information and categorize them. This information is then indexed for fast retrieval when the user searches for them. The search engines need to be able to perform extreme processor intensive operations, as they have to be responsive to the user while, in the background, they collect and index other information from the Internet. This resulted in the search engines that were 'slow' to respond and took 'longer' to perform the search.

To make the search engines more effective, the search application was spawned into two separate processes - one limited to the search function and a new type of application called 'web crawler.' In this approach the search engines performs the task of interacting with the user and returns the results from an indexed data source. The 'web crawler' performs the task of skimming the Internet and indexing the data. There are about 90 million registered domains with the volume of documents contained being close to several hundred billion. As most of these documents are publicly available, the web

crawlers have to create an index of several billion web links and also periodically update as the web content changes. Even if the current day crawler indexes a website at least once a day, the data can become outdated. Some of the websites, like newspapers, update several times in a day thus making the indexed pages in the search engines inaccurate.

GRUB is a peer-to-peer distributed web crawler in which the processing time, bandwidth, and the disk space are the resources that are shared between the peers [36]. Each peer contributes to the 'web crawler' process that can be used to feed other search engines in the Internet thus increasing the accuracy of the search by reducing the chances of data becoming inaccurate due to the time lag between the search and web cawling. The GRUB peers would help in sharing bandwidth, processing time and its diskspace with all the peers in the network. For the search engines, it is very important to keep the search accurate even though the number of web pages increases on a daily basis.

9.5.1. Comparison with RAN:

This section discusses the various advantages and disadvantages of the RAN over GRUB.

Advantages of RAN:

The servers in the GRUB network allot a set of websites for each peer where the only function performed by the peer is web crawling. This is huge limitation when compared with RAN, where the peers can share multiple resource types and perform a multitude of operations with each other.

Disadvantages of RAN:

In GRUB, the master server controls the access to the database where the indexed pages are stored to increase security. However, in RAN, though the resource directory controls local data source, there is no master controller.

9.6. Comparison of RAN with Related Systems

In this section, the summary of the general characteristics of the various peer-to-peer systems and their comparison with RAN, as presented in the previous sections, are tabulated in Table 9.6.1 below.

Peer-to-peer system	**Characteristics**
Category: Search	
RAN	▪ Controlled search – limited by HopNo. ▪ Limited to connected peers.
NINJA SDS	▪ Uses XSet XML search Engine ▪ Uses intermediate systems to connect the clients and service providers
JESA	▪ Uses Provider Location Protocol (PLP)
Gnutella	▪ Exhaustive ▪ Limited to connected peers
SETI@HOME	▪ No search between peers
GRUB	▪ No search between peers
Category: Topology	
RAN	▪ Partly centralized ▪ Can exist as sub nodes
NINJA SDS	▪ Partly centralized
JESA	▪ Partly centralized
Gnutella	▪ Decentralized
SETI@HOME	▪ Centralized
GRUB	▪ Centralized
Category: Reliability	
RAN	▪ Susceptible to seed node failure ▪ Resilient to peer failure ▪ Can exist as sub nodes ▪ Can have multiple seed peers
NINJA SDS	▪ Fault-tolerant Scalable ▪ Continuous availability

	▪ Directory like service
JESA	▪ Works with or without central service brokers ▪ Scalable
Gnutella	▪ Scalable ▪ Fault-tolerant
SETI@HOME	▪ Susceptible to central server failure ▪ Resilient to client failure ▪ Back up servers available
GRUB	▪ Susceptible to seed node failure ▪ Resilient to peer failure ▪ Can exist as sub nodes ▪ Can have multiple seed peers
Category: Search Accuracy	
RAN	▪ Supports partial and exact matches ▪ Substring searches ▪ Depends on accuracy of resource descriptor
NINJA SDS	▪ Supports partial and exact matches ▪ Depends on accuracy of service descriptor in SDS server
JESA	▪ Supports matches on service type
Gnutella	▪ Supports partial and exact matches ▪ Substring searches ▪ Accuracy based on 'master list'
SETI@HOME	▪ Search not supported
GRUB	▪ Supports partial and exact matches ▪ Substring searches ▪ Depends on accuracy of resource descriptor
Category: Resource Types	
RAN	▪ Disparate resource types ▪ Allows for addition of new resource types
NINJA SDS	▪ Services with interface capable of performing computation
JESA	▪ Services in ad-hoc network
Gnutella	▪ All file types
SETI@HOME	▪ Processing time
GRUB	▪ Disparate resource types ▪ Allows for addition of new resource types
Category: Peer Connection	
RAN	▪ Connected as binary tree structure ▪ Balanced structure for increased search performance
NINJA SDS	▪ Local clients and the providers are connected to the SDS server ▪ SDS server are interconnected
JESA	▪ Connection based on the client and provider initial knowledge and relationship ▪ Client – provider connection can be direct or mediated.

Gnutella	▪ All the peers are connected to one another.
SETI@HOME	▪ All peers connected to one central server.
GRUB	▪ Connected to the centralized server
Category: Clients and Messaging	
RAN	▪ Any participating peer Participates in resources' search ▪ Acts as conduit for message propagation ▪ ASCII messages
NINJA SDS	▪ Any participating work station ▪ Uses ARMI to communicate with the servers ▪ XML messages
JESA	▪ Embedded and mobile systems ▪ UDP messages
Gnutella	▪ Any participating peer ▪ Performs look up functionality ▪ Gnutella packets over TCP/IP
SETI@HOME	▪ All peers connected to one central server.
GRUB	▪ Connected to the centralized server

Table 9.6.1 Comparison of peer-to-peer systems and RAN

10. Evaluation and Future Work

This chapter provides a reference to the related research effort that resulted in proving the concept of the RAN's ability to support files and file chunks as resources. Also discussed are future enhancements that could improve the performance.

10.1. RAN-based Application

As described in the earlier chapters in this literature, RAN resource directory architecture could be used to design several systems that support disparate resources. One such system, designed based on the RAN architecture, is the RDSS, a Resource Area Network (RAN)-based Distributed Storage System, "which is designed for high scalability, long-term reliability, and operational efficiency of storage system" [5]. The RDSS system is an Internet-scale storage system in a physically distributed network that can operate with high reliability using the RAN architectural pattern [1].

The goal of RDSS is to utilize the huge amount of unused storage capacity among the disparate nodes in a distributed network, to provide efficient storage services. The life of a peer in a network can vary drastically in a network with equal peers. There is no central server to monitor the peers joining or leaving the network.

In most of the peer-to-peer networks, the trust determination of the peers determines the security of the peer-to-peer network. In a network with no restriction on the peers that are willing to participate in the resource sharing, the probability of huge number of un-trusted peers is high. The peers' index of trustworthiness cannot be

computed easily. There can be nodes that join the network with the intention of maliciously accessing the storage system.

The RDSS performance relies on its ability to support ‘short lived peers’ and the ability to manage ‘un-trusted’ peers. ‘Short lived peers’ are peers whose life time in the network is very low. RDSS is designed to provide consistent reliability with ‘Short lived peers’. By detecting and managing the network activity of ‘un-trusted’ peers, the security in the network can be enhanced. In order to provide an efficient solution for RDSS, the following assumptions were made. The RAN resource directory is expected to be trustworthy even though it may consider all peers as ‘low trust’ peers in the network. RDSS considers malicious nodes act separately and not in conjunction with RAN resource directory or other peers.

This system is an example of how resource directory plays an important role in the systems based on RAN architecture. The RDSS uses an active resource directory as its directory service, in which when a resource is registered the information is actively distributed throughout the network. These types of resource directories are used in an environment where the time taken for a search needs to be minimal. When an RDSS server joins the RAN system, the resource exporter registers all its available storage resources to the resource directory. The resource directory is responsible for forwarding these registrations to its peers. In order to reduce the duplication of data, the resource directory forwards the registration request to all its peers other than the peer from where the request came from. The resource directory is responsible for locating these registered

storage resources when an application requests for the resources. The NRS (Node ranking system) that was implemented for RDSS improves the system reliability by ranking the nodes based on their past performance in the network.

10.2. Limitations of RAN Resource Directory Implementation

This section provides a brief discussion on the limitations related to the current implementation of the RAN Resource Directory. In future resource directory implementations these limitations could be overcome by developing on some of the items discussed under future work.

10.2.1. Seed Node Failure

Though RAN implementation may not be susceptible to single point failures, this network is however unprotected against failures of the seed peers. Every peer, as part of startup system parameter, has a designated seed peer IP address. These seed peers are the first point of contact for any peer that joins the network. The seed peer performs the initial balancing and routes the peer to the docking location. The seed peer failure can result in two conditions discussed below.

10.2.1.1. New Peer Inclusion

When the seed peer of any peer that attempts to join the network fails, the new peer can no longer join the network. The new peer repeatedly attempts to join the seed peer and can be included in the network only when the seed peer is available.

10.2.1.2. Sub-tree Coalescing

When the failure of the seed peer splits the network into two sub-trees, both sub-trees can independently operate. Though this can be viewed as an advantage of the RAN resource directory, the sub-trees do not automatically join to form a single network. There is no dynamic substitution of the seed peer designation.

10.3. Future Work

In this section, some enhancements related to the security and performance of the RAN resource directory implementation are discussed.

10.3.1. Dynamic Seed Peer Substitution

The limitation related to seed peer failure can be overcome by dynamically designating an existing peer as a seed peer. These peers can actively inform their status to other peers in the network. When the peer authentication is implemented, the peers can actively multicast its status information in a channel that other peers or subset of peers listen to. This would reduce the existence of independent network and will make the RAN system a more robust peer-to-peer system.

10.3.2. Authentication

To increase the reliability of the RAN resource directory, peer authentication can be implemented in the system. However, in the current resource directory data is protected as it can be accessed only through the resource directory interface. The peer authentication would prevent malicious peers from participating in the RAN system.

10.3.3. Blind Messages

During the query search process, even though a node returns a successful result for the query, the search continues to propagate through the network, as the other peers are not aware of the successful find. All successful results found are back propagated to the originator of the search query. However, at the originator node the secondary successful results received are ignored. As most peers are blind to the fact that the search query has resulted in a successful find, they continue to send subsequent successful results back to the originator. In future implementations this process can be modified to where, based on search query ID, all peers in a network are informed of the first successful find and hence resulting in the stoppage of further search query propagation or back casting of subsequent successful results.

10.4. Conclusion

Through this project, it has been able to demonstrate that it is possible to create a RAN system with disparate resource types. By using a resource directory, it is possible to dynamically manage the information contained within the peers and manage the resource types that are shared in the RAN system. In addition, the search time in the RAN can be reduced by implementing a balancing algorithm to maintain the peer structure.

11. Bibliography

[1] E.Glenn, "Number of Personal Computers in the US," [Online document], [cited Jun 2004], Available HTTP: http://hypertextbook.com/facts/2004/DianeEnnefils.shtml

[2] T. Ray, "Who Cares About the Fastest Internet Ever?," [Online document], March 13, 2003, [cited Jun 2004], http://www.newsfactor.com/story.xhtml?story_id=20986&page=1

[3] E. Drakopoulos, M.Merges, "Performance study of client-server storage systems," AT&T Bell Lab, Naperville, IL, 1991

[4] E. Korpela, D.Werthimer, D.Anderson, J.Cobb, M.Leboisky, "SETI@HOME—massively distributed computing for SETI,"Computer Science. Eng. 3," 1 (Jan. 2001), 78-83

[5] X. Li, "RDSS: A Reliable and Efficient Distributed Storage System," Master Thesis, School of Electrical Engineering and Computer Science, Ohio University, August 5, 2004

[6] X. Li, C. Liu, "Towards a Reliable and Efficient Distributed Storage System," the 38th Hawaii International Conference on System Sciences (HICSS) Software Technology Track, Peer-to-Peer Infrastructures and Applications, Big Island, Hawaii, January 3-6, 2005

[7] Gnutella, [Online document], [cited Apr 2004], http://gnutella.com

[8] C. Liu, L. Welch, D. Juedes, "The Resource Area Network Architectural Pattern," The 10th Conference on Pattern Languages of Programs, Urbana, IL, USA, September 8-12, 2003

[9] "The Gnutella Protocol Specification v0.41" (Document Revision 1.0), [Online document], http://www.clip2.com/GnutellaProtocol04.pdf

[10] Oram, "Peer-To-Peer: Harnessing the Benefits of a Disruptive Technology: O'Reilly," 2001. ISBN: 059600110X

[11] Napster, [Online document], [cited Apr 2004], http://www.napster.com

[12] Limewire, [Online document], [cited Jun 2004], http://www.limewire.com

[13] C. Rohrs, "LimeWire Design," [Online document], 2001, [cited Aug 2007], http://www.limewire.org/techdocs/design.html

[14] Bearshare, [Online document], [cited May 2005], http://www.bearshare.com/

[15] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, J. Kubiatowicz, "Tapestry: A Resilient Global-scale Overlay for Service Deployment," IEEE Journal on Selected Areas in Communications, January 2004, Vol. 22, No. 1

[16] C. Greg Plaxton, Rajmohan Rajaraman, Andrea W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," In Proceedings of ACM SPAA. ACM, June 1997

[17] "What is Freenet?", [Online document], [cited Apr 2004], http://freenetproject.org/index.php?page=whatis

[18] L. Garc es-Erice, E. W. Biersack, P. A. Felber, K. W. Ross, G. Urvoy-Keller, "Hierarchical peer-to-peer systems," In Proc. of ACM/IFIP Intl. Conf. on Parallel and Distributed Computing (Euro-Par 2003)

[19] "Kazaa," [Online document], [cited May 2005], http://www.kazaa.com

[20] E. Adar, B. A. Huberman, "Free riding on gnutella," First Monday, vol. 5, Oct. 2000.

[21] "Understanding Peer-to-Peer Networking and File-Sharing," [Online document], [cited May 2005], http://www.limewire.com/about/peer-to-peer.php

[22] "How Much Information? 2003", [Online document], [cited Apr 2004], http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/Internet.htm

[23] RFC 791, "INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION," http://tools.ietf.org/html/rfc791

[24] D. P. Anderson , J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, 2002, "SETI@home: an experiment in public-resource computing," ACM 45, Nov 2002), 56-61

[25] R. Penton, “Trees” , [Online document], [cited July 2006], http://www.gamedev.net/reference/programming/features/trees2/page2.asp

[26] A. V. Aho, J. D. Ullman, J. E. Hopcroft, “Data Structures and Algorithms,” Addison Wesley, 1983

[27] M. Sipser, “Introduction to the Theory of Computation 2ed. PWS Publishing,” 2006, ISBN 0-534-94728-X. Part Two: Computability Theory, chapters 3–6, pp.123–222

[28] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, R. H. Katz, “An architecture for a secure service discovery service,” In Proceedings of the 5th Annual ACM/IEEE international Conference on Mobile Computing and Networking, Seattle, WA, MobiCom '99. ACM Press, New York, NY, 24-35

[29] F. Zhu, M. W. Mutka, L. M. Ni, "Service Discovery in Pervasive Computing Environments,” IEEE Pervasive Computing, vol. 04, no. 4, pp. 81-90, Oct-Dec, 2005

[30] Y.Chawathe, S. Ratnasamy, L.Breslau, N. Lanham, S. Shenker, 2003. “Making gnutella-like peer-to-peer systems scalable,” In Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols

[31] T. Mengotti, “GPU, a framework for distributed computing over Gnutella,” ETH Zürich, Switzerland

[32] L. Garc´es-Erice, E.W. Biersack, P.A. Felberl, K.W. Ross, G. Urvoy-Keller,"Hierarchical Peer-to-peer Systems,"Polytechnic University,Brooklyn, NY 11201, USA

[33] S. Preu, E. Gregori, M. Conti, A. T. Campbell, G. Omidyar, M. Zukerman, ” JESA Service Discovery Protocol,”Proceedings of Networking 2002, volume 2345 of LNCS, pages 1196--1201, Pisa, Italy, May 2002

[34] S. Preu, “JESA Service Discovery Protocol - Efficient Service Discovery in Ad-Hoc Networks,” University of Rostock

[35] “INTERNET GROWTH STATISTICS”, [Online document], [cited July 2006], http://www.Internetworldstats.com/emarketing.htm

[36] Grub, [Online document], [cited Aug 2007], http://www.grub.org/

[37] K. K. Ramakrishnan, J. S. Emer, “ Performance Analysis of Mass Storage Service Alternatives for Distributed Systems,” IEEE Trans. Software Eng. 15(2): 120-133 (1989)

[38] T.M. Cormen, C. E. Leiserson, R. L. Rivest , C. Stein, "Introduction to Algorithms", MIT Press, 2001