

# **RADIUS SIMULATOR**

## **(Remote Authentication Dial-In User Service)**

### **MINOR PROJECT REPORT**

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD  
OF THE DEGREE OF

### **BACHELOR OF ENGINEERING**

**(Computer Science and Engineering)**



Submitted By:

Siddharth Kumar     UE143093  
Kunal Taneja         UE143053  
Haritima Manchanda UE143038

Submitted To:

Mr. Makhan Singh

**Department of Computer Science and Engineering**  
**University Institute of Engineering and Technology (UIET)**  
**Panjab University, Chandigarh**

**Abstract:**

The project is aimed at developing a tool which simulates the RADIUS Client functionality. It enables centralized authentication, authorization and accounting for all the dial-in users who wants to connect and use network services. RADIUS facilitates centralized user administration, which is important for several of these applications. Many ISPs have tens of thousands, hundreds of thousands, or even millions of users. Users are added and deleted continuously throughout the day, and user authentication information changes constantly. Centralized administration of users in this setting is an operational requirement. The distributive nature of RADIUS effectively separates the security processes (carried out on the authentication server) from the communications processes (implemented by the modem pool or the Network Access Server (NAS)), allowing for a single centralized information store for authorization and authentication information. This centralization can significantly lessen the administrative burden of providing appropriate access control for a large number of remote users. If ensuring high availability is not a priority, then redundancy is not required; this centralization can thus be maximized, since all RADIUS-compatible hardware on a LAN can derive authentication services from a single server.

In this project we are going to develop a RADIUS client tool which will be able to authenticate multiple dial-in users (user equipment) with centralized RADIUS server and authorize them to use network services and along with that it will account all the users for the services provided to them. We are going to use FreeRadius open source tool as a RADIUS server. FreeRadius server will maintain the central database of user profiles. On the basis of the profiles maintained in the DB, server will authenticate the user that whether the incoming request from our client is for valid user or not. Only after successful authentication, server will provide access to network services to the users.

## **Acknowledgement**

I express our deep sense of gratitude towards Mr. Makhan Singh, Assistant Professor, Department of Computer Science and Engineering, Panjab University, for giving the team an opportunity to experience dynamic professional environment during the project. We are much obliged to the college for giving us an opportunity and a congenial atmosphere to learn while working without bound. We will be failing in our duty if we don't express our heartiest thanks to our respected parents for providing us every kind of support during the completion of this project. Last but not the least we feel very honourable to thank all our friends for their encouragement and moral support for making this project and at last how can we forget god the almighty who is helping everybody.

“World acknowledge those who acknowledge the world.”

Thanks to you all.

### **Certificate of Originality**

This is the Certificate of Originality to certify that Haritima Manchanda (UE143038), Kunal Taneja (UE143053) and Siddharth Kumar (UE143093) have successfully completed the project on:

#### **RADIUS Simulator (Remote Authentication Dial-In User Service)**

Under the mentorship of Mr. Makhan Singh, Assistant Professor, Department of CSE, UIET, Panjab University. This project is authentic and original development of the students under the guidance of their mentor.

Haritima Manchanda UE143038  
Kunal Taneja UE143053  
Siddharth Kumar UE143093

Mr. Makhan Singh  
Asst. Professor  
Department of CSE

**Table of Figures**

<b>S.no</b>	<b>Figure Title</b>	<b>Page No.</b>
1.	SDLC of proposed system	13
2.	RADIUS design	14
3.	High Level architecture of RADIUS	15
4.	Use case diagram	16
5.	RADIUS authentication	16
6.	RADIUS accounting	17
7.	User Interface Design	17
8.	Methodology	18
9.	Authentication and authorization sequence	19
10.	Client server communication	21
11.	Gantt chart	24
12.	RADIUS server	27
13.	Running server	28
14.	GUI	28
15.	File tab of GUI	29
16.	User specific credentials	29
17.	Generate tab	30
18.	Wireshark	31
19.	Input file	32
20.	Users file	32

## Table of Contents

Serial no.	Contents	Page no.
1.	<b>Introduction</b> 1.1 Introduction to Project 1.2 Project Category 1.3 Objectives 1.4 Problem Formulation 1.5 Identification/Reorganization of Need 1.6 Existing System 1.7 Proposed System	7-9
2.	<b>Requirement Analysis and System Specification</b> 2.1 Feasibility study (Technical, Economical, Operational) 2.2 Software Requirement Specification Document 2.3 Validation 2.4 SDLC model	10-13
3.	<b>System Design</b> 3.1 Design Approach (Function oriented or Object oriented) 3.2 Detail Design 3.3 System Design using Flowcharts or UML 3.4 User Interface Design 3.5 Methodology	14-19
4.	<b>Implementation, Testing, and Maintenance</b> 4.1 Introduction to Languages, IDE's, Tools and Technologies 4.2 Project Scheduling using GANTT charts. 4.3 Testing Techniques and Test Plans	20-25
5.	<b>Results and Discussion</b> 5.1 User Interface Representation 5.2 Snapshots of system 5.3 Backend Representation	26-32
6.	<b>Conclusion and future scope</b>	33
7.	<b>References</b>	34

## **Chapter 1 – Introduction**

### **1.1 Introduction to Project**

The project is based on developing a tool that facilitates the radius client functionality. The aim is to identify the shortcomings in the existing tool and develop an improved version of the radius protocol tool.

RADIUS is an acronym for "Remote Authentication Dial in User Service".

RADIUS is a networking protocol developed as an AAA (Authentication, Authorization & Accounting) protocol. It is based on Client-Server model. RADIUS protocol yields an open and scalable client/server security system. The RADIUS client is considered as a NAS (Network Access Server), and the RADIUS server is usually a daemon process running on UNIX/Windows NT machine. RADIUS protocol provides a connectionless service. So the communication between NAS (network access server) and RADIUS server uses UDP (User Datagram Protocol). This protocol carries all authentication, authorization and configuration related information between the NAS, which is responsible to authenticate its links, and central authentication RADIUS server.

In this project we developed a RADIUS client tool which will be able to authenticate multiple dial-in users (user equipments) with centralized RADIUS server and authorize them to use network services and along with that it will account all the users for the services provided to them.

There are multiple methods for authentication of users in Radius protocol. We used the CHAP protocol for authentication. In a CHAP scheme, the following process establishes a user identity:

1. After the link between the user machine and the authenticating server is made, the server sends a challenge message to the connection requester. The requester responds with a value obtained by using a one-way hash function.
2. The server checks the response by comparing it against its own calculation of the expected hash value.
3. If the values match, the authentication is acknowledged; otherwise the connection is terminated.

We used FreeRADIUS open source tool as a RADIUS server. FreeRADIUS server will maintain the central database of user profiles. On the basis of the profiles maintained in the DB, server will authenticate the user that whether the incoming request from our client is for valid user or not. Only after successful authentication, server will provide access to network services to the users.

### **1.2 Project Category**

The project is an industry based project that tends to cater the needs of the organisations that require centralised Authentication, Authorisation and Accounting (AAA) management for users that connect and use a network service. It is often used by Internet service providers (ISPs) and enterprises to manage access to the Internet or internal networks, wireless networks, and integrated e-mail services. These networks may incorporate modems, digital subscriber line (DSL), access points, virtual private networks (VPNs), network ports, web servers, etc.

### 1.3 Objectives

Out of all the protocols available today for providing AAA services, Radius is the most widely used protocol. To implement the Radius protocol FreeRADIUS is campaigned as the most used Radius tool in the world. The popularity is explained, with an easy-to-use approach, no cost required to use (downloadable from an official website), user-friendliness, quick and easy installation and comparable security to the payable servers. However FreeRADIUS comes with its own set of limitations. Our objective was to study those limitations and develop a tool that incorporates all the desirable properties of the FreeRADIUS while overcoming the limitations of the FreeRADIUS tool.

The main objective of our project was to improve the rate at which the Radius Protocol handles request that is its scalability. FreeRADIUS handles only one request at a time therefore its scalability is limited. We intended to extract the vulnerabilities of the existing system and to improve upon them.

### 1.4 Problem Formulation

The problem can be divided into the following modules:

- **FreeRADIUS Server:** It supports all common authentication protocols, and the server comes with a PHP-based web user administration tool called dialup admin. The FreeRADIUS Suite includes a RADIUS server, a BSD-licensed RADIUS client library, a PAM library, an Apache module, and numerous additional RADIUS related utilities and development libraries. In this project we have made use of it as a free open source server.
- **RADIUS Client-Side program written in C:** It is here we develop the AAA structure and scale our program such that the whole RADIUS environment can support multiple client requests simultaneously. Various concepts of C such as Multithreading, Socket Programming (UDP), Message IPC, and Message Synchronisation are used to develop the complete architecture.
- **Graphical User Interface:** In order to provide flexibility and ease of access to the services provided by the simulator our project supports a GUI which is a simple menu driven interface through which the user can provide the credentials and create the input file, the server after accessing the input file runs the C program at the back end which takes care of the AAA services.

### 1.5 Identification/Reorganisation of Need

With the increasing usage of wireless devices a need for a protocol that supports wireless environment is widely increasing. We required a system that is more flexible, scalable and interoperable. We need a system that can handle many requests at a time and is secure enough to handle the threats to the vulnerabilities of the system.



## **1.6 Existing System**

Diameter protocol is a planned successor of the Radius protocol but until now Radius is the widely deployed protocol. The basic operation of RADIUS and Diameter is similar. They both carry authentication, authorization and configuration information between a Network Access Server (NAS) and a shared Authentication Server. On the transport layer RADIUS uses connectionless UDP, while Diameter utilizes either SCTP or TCP. Both protocols provide acknowledged, error-free, non-duplicated transfer of user data. Though Diameter protocol by the use of TCP protocol offers more reliable services but this also makes the services heavier to implement and operate.

The existing Radius tool does not provide any support for error messages. The entities which do not receive any replies to their requests have no way of knowing whether the other party drops messages or if there is for example network congestion while diameter supports error messages. One of the main reasons for creating Diameter is the poor scalability of RADIUS. But as FreeRADIUS is used intensively in companies nowadays it is important to make improvisations in the existing system and provide as much as increased scalability and congestion control as possible.

Some of the shortcomings that we were able to spot out in existing FreeRADIUS tool supporting Radius Protocol are as follows:

- FreeRADIUS allows accounting to start before the access request has been processed. So before the user has been authenticated accounting.
- FreeRADIUS can process one request at a time.
- In FreeRADIUS a user has to type long statements to request access to a server and start accounting request.

## **1.7 Proposed System and Its Benefits**

The proposed system targets an increase in scalability and congestion control by allowing multiple user requests to be handled at a time. It also allows accounting to take place only after the user has been authenticated or the request for access has been processed. The system uses CHAP protocol for authenticating the users.

## **Chapter 2 – Requirement Analysis and System Specification**

### **2.1 Feasibility study**

A feasibility study is an evaluation and analysis of the potential of the proposed project which is based on extensive investigation and research to give full comfort to the decision makers. Feasibility studies aim to objectively and rationally uncover the strengths and weakness of an existing business or proposed venture, opportunities and threats as presented by the environment, the resources required to carry through, and ultimately the prospects for success.

In its simplest terms, the two criteria to judge feasibility are the cost required and value to be attained. As such, a well-designed feasibility study should provide a historical background of the business or project, description of the product or service, accounting statements, details of the operation and management, marketing research and policies, financial data, legal requirements and tax obligations. Generally, feasibility studies precede technical development and project implementation.

When a new project is proposed, it normally goes through feasibility assessment. Feasibility study is carried out to determine whether the proposed system is possible to develop with available resources and what should be the cost consideration. Facts considered in the feasibility analysis were

- Technical feasibility
- Economic feasibility
- Operational feasibility

#### **Technical feasibility:**

Technical feasibility centres on the existing computer system (hardware/software) and to what extent it can support the proposed addition also the organization already has sufficient high end machines to serve the processing requirements of the proposed system.

The project is technically feasible as the technology involved in the project is easily available. This project is technical feasible because in this project add information, search result, all these things are technically feasible. In our project, if we want to add new data, then simply we click on add menu and store into database.

**Economic feasibility:**

Economic feasibility is the most frequently used method for evaluating the effectiveness of the candidate system. More commonly known as cost/benefit analysis, the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with the costs. If benefits outweigh the costs, then the decision is made to design and implement the system. The project is economically feasible as we use Linux as platform which is an open source operating system and the server used (FreeRADIUS Server) in project is also free of cost. For the users to access the application, the only cost involved will be getting access to the network over Linux based operating system.

**Operational feasibility:**

Operational feasibility is evaluation is to determine whether system is operationally is acceptable. During this study it was determine whether the system will operate in the way that user wants or not. This project is also operational feasible because in this project we provide the graphical user interface (GUI) which is easy to understand & operate. It also provides the user friendly interface. The user will easily use the system. In this project we use the buttons, text box, images which is easily understandable for end user.

The system will be used if it is developed well skill then be resistance for users that undetermined

1. It will help in the time saving and fast processing and applications.
2. New product will provide all the benefits of present system with better performance.
3. Improved information, better management and collection of the reports.

**2.2 Software Requirement Specification Document****Data Requirements**

- Client credentials like Client IP Address, Client Port Number, Calling Station id etc.
- Port numbers for Authentication and Accounting requests.
- RADIUS Server's database containing credentials corresponding to variety of clients.

**Functional Requirements**

- To build fully functional components of the system.
- To build a user friendly client side application
- To ensure the proper authentication, authorisation and accounting services.

### **Performance Requirements**

- To ensure system reliability.
- To ensure system accuracy.
- To ensure system flexibility.
- To ensure system maintainability.
- To improve the efficiency of system.

### **Maintainability Requirements**

- To ensure that bug fixing in the system takes place properly.
- To ensure that the quality of the system doesn't degrade with time.
- To ensure that the system is easy to maintain and update.
- To provide ease of operations.
- To provide easy adaptability.

### **Security Requirements**

- To maintain the integrity of system.
- To ensure system recovery in case of system crashes.

### **Look and Feel Requirements**

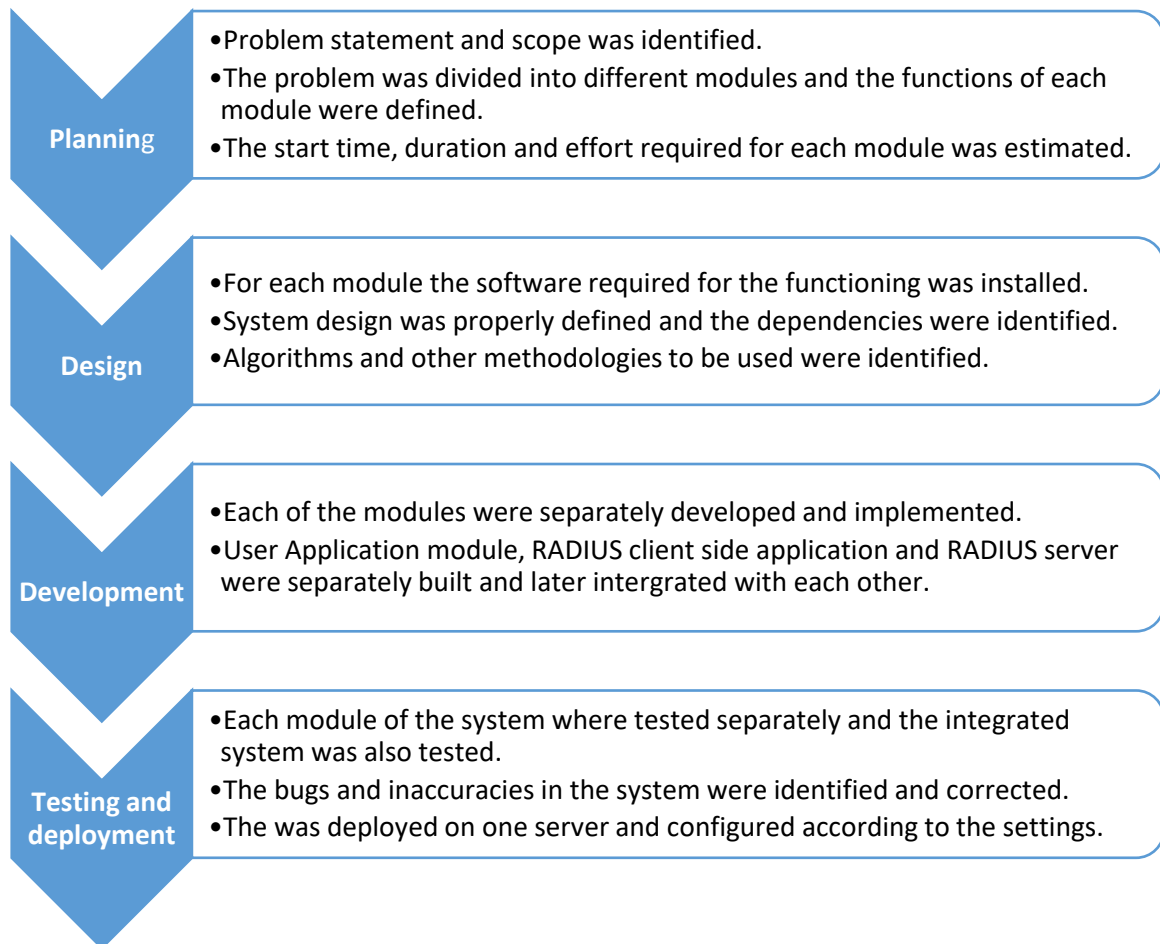
- The application should be easy and understandable for the users.

## **2.3 Validation**

RADIUS is a client/server protocol. The client passes user information to designated RADIUS servers and acts on the response that is returned. RADIUS servers receive user connection requests, authenticate the user, and then return the configuration information necessary for the client to deliver service to the user. A RADIUS server can act as a proxy client to other RADIUS servers or other kinds of authentication servers. The system is reliable and efficient as it provides a one-time investment. It also reduces a large amount of human effort and time required. The system uses easily available softwares and technology and hence leads to cost reduction. The system provides a user friendly application that is easily understandable to an individual. The application built is easy to update and maintain.

## 2.4 Software Development Life Cycle

The software development life cycle of the system would consist of the planning, design, development, testing and deployment phase.



*Fig 1: SDLC of the proposed system*

## Chapter 3- System Design

### 3.1 Design Approach

The RADIUS server is an authentication and authorization information server for a Network Access Server (NAS). A NAS is a device that provides an access point to the network for remote users connecting using SLIP, PPP, or any other remote access protocol. The NAS transmits the information provided in the connection request from the remote user to the RADIUS server. The RADIUS server checks this information against the entry for the remote user in the directory. It then returns to the NAS an authorization or denial for the remote user connection. It can also provide the appropriate connection parameters for the remote user connection. The **user** is the entity requesting access to network resources. In the directory database, a user is identified by a unique **uid**. The uid attribute, and all other attributes describing a remote user, are defined in the remote User object class.

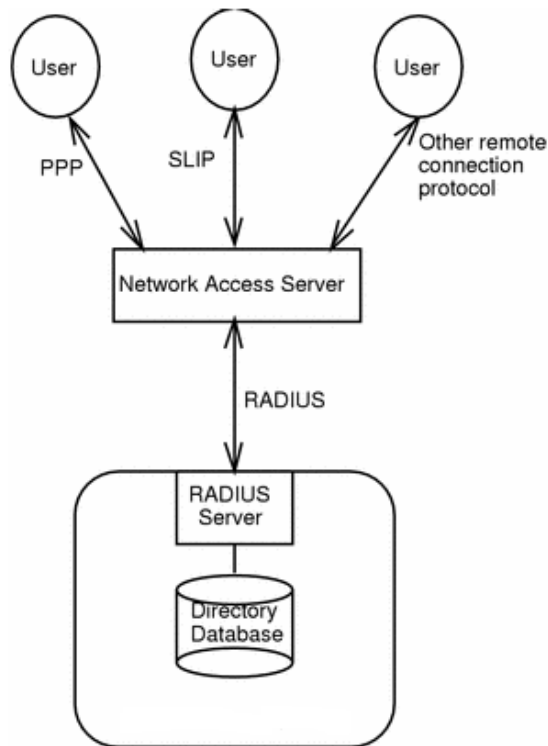


Figure 2: RADIUS Design

The Network Access Server, also called a client, is the device to which remote users connect. The client queries the RADIUS server for authentication status, user profiles and authorizations. In the directory database, a client is identified by a unique ipHostNumber. The ipHostNumber attribute, and all other attributes describing a RADIUS client are defined in the NAS object class. The RADIUS server authenticates the NAS, then checks the remote user's identity and authorization in the directory database. It returns the user's status and configuration information to the NAS. If the RADIUS server is unable to authenticate the NAS, the request from the NAS is ignored. The RADIUS server does not respond, even with a connection rejection.

### 3.2 Detail Design and Methodology.

In this project, keeping in mind the AAA services of RADIUS protocol we designed a multi-threaded system in which to each service, single thread was devoted. Moreover, concurrency control such as mutual exclusion and serializability were enforced.

- **1st thread (Authentication)** - Responsible for fetching first request of each user of respective client and authenticating it against the entries present in server.
- **2nd thread (Authorization)** - After successful authentication, this thread manages the communication between the server and user. For example, user might request accounting services from the server, and the latter after validating the state of the user authorises the user for respective services.
- **3rd thread (Accounting)** - This thread monitors the resources used by clients and for what duration these resources are used.

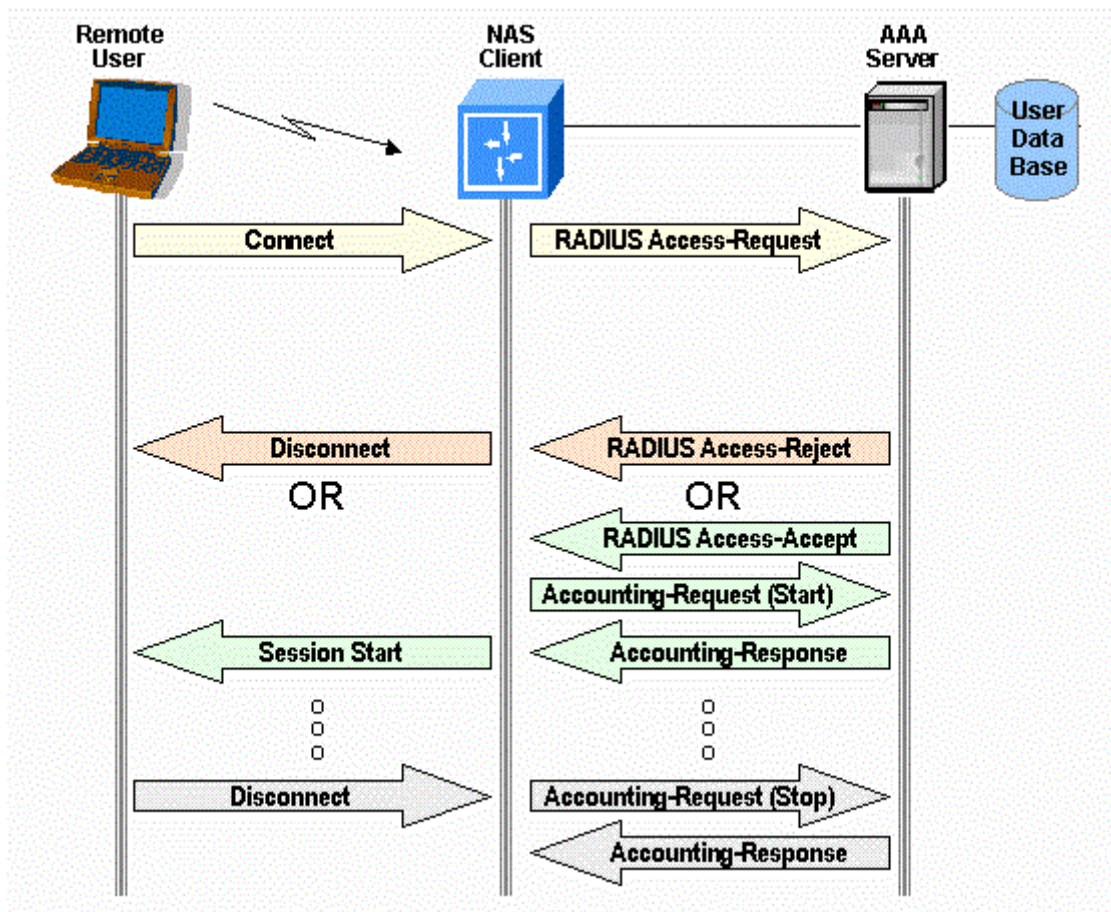


Figure 3: High Level architecture of RADIUS protocol

### 3.3 System Design using various Structured analysis and design tools:

#### 3.3.1. Use Case Diagram:

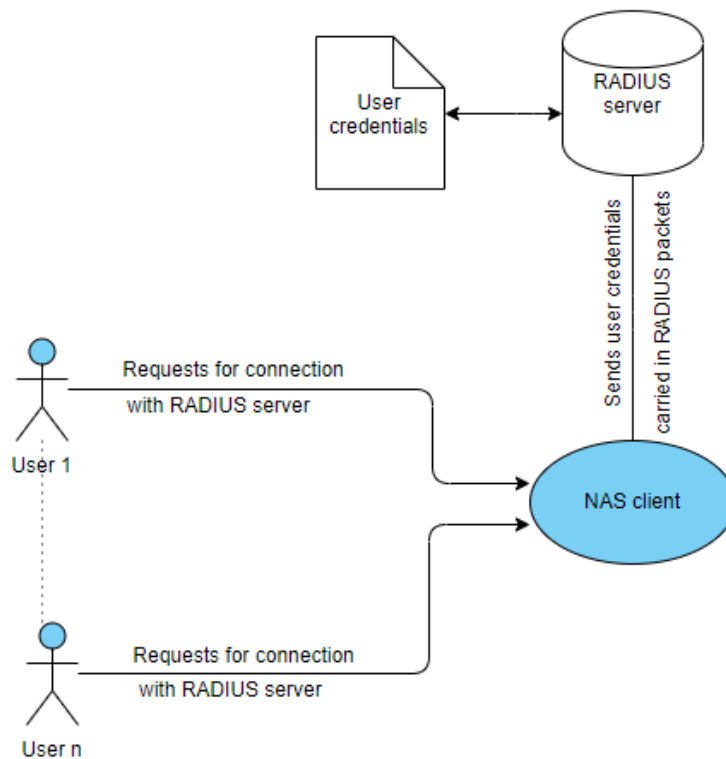


Figure 4: Use case Diagram

#### 3.3.2. Sequence Diagram:

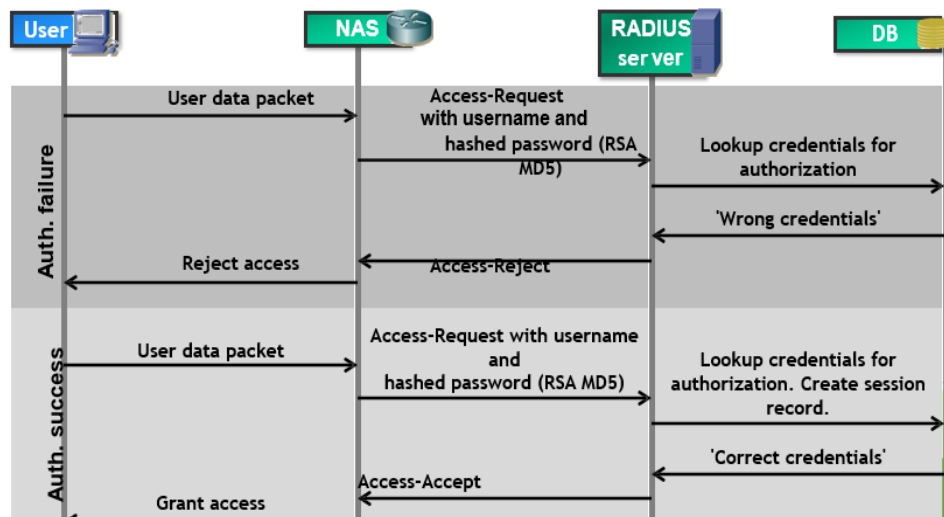


Figure 5: RADIUS Authentication



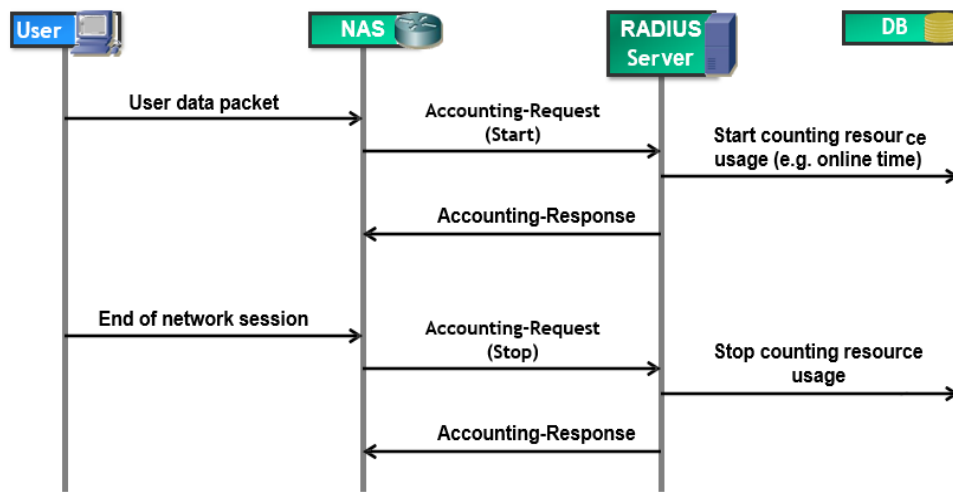


Figure 6: RADIUS Accounting

### 3.4 User Interface Design

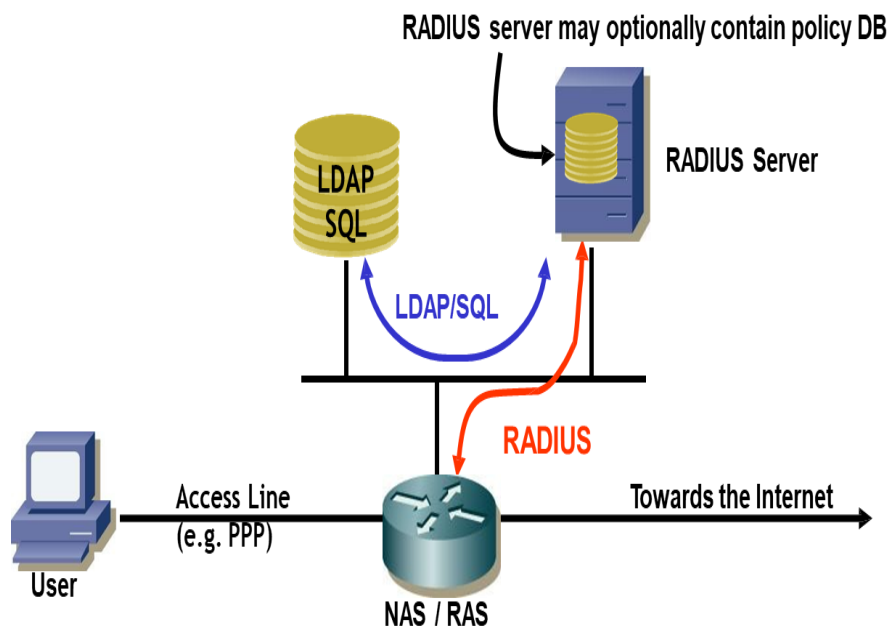


Figure 7: User Interface Design

In this, a front-end NAS (network access server) or RAS (remote access server) performs authentication of a user with a backend RADIUS server. The NAS/RAS sends user information (credentials) to the RADIUS server carried in RADIUS packets. The RADIUS server implements the access policy (who is granted access with what authorizations) or may retrieve policies from a database through LDAP (Lightweight Directory Access Protocol).

### 3.6 Methodology

Communication between a network access server (NAS) and a RADIUS server is based on the User Datagram Protocol (UDP). Generally, the RADIUS protocol is considered a connectionless service. Issues related to server availability, retransmission, and timeouts are handled by the RADIUS-enabled devices rather than the transmission protocol.

RADIUS is a client/server protocol. The RADIUS client is typically a NAS and the RADIUS server is usually a daemon process running on a UNIX or Windows NT machine. The client passes user information to designated RADIUS servers and acts on the response that is returned. RADIUS servers receive user connection requests, authenticate the user, and then return the configuration information necessary for the client to deliver service to the user. A RADIUS server can act as a proxy client to other RADIUS servers or other kinds of authentication servers.

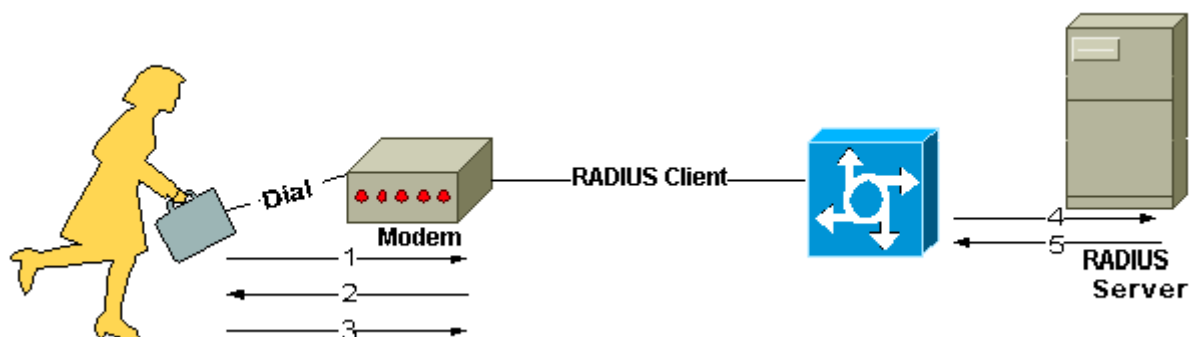


Figure 8: Methodology

1. User initiates PPP authentication to the NAS.
2. NAS prompts for username and password (if Password Authentication Protocol [PAP]) or challenge (if Challenge Handshake Authentication Protocol [CHAP]).
3. User replies.
4. RADIUS client sends username and encrypted password to the RADIUS server.
5. RADIUS server responds with Accept, Reject, or Challenge.
6. The RADIUS client acts upon services and services parameters bundled with Accept or Reject.

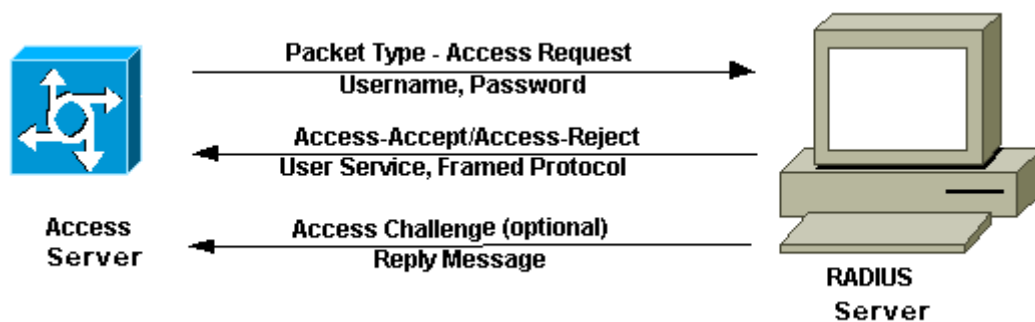
The RADIUS server can support a variety of methods to authenticate a user. When it is provided with the username and original password given by the user, it can support PPP, PAP or CHAP, UNIX login, and other authentication mechanisms.

Typically, a user login consists of a query (Access-Request) from the NAS to the RADIUS server and a corresponding response (Access-Accept or Access-Reject) from the server. The Access-Request packet contains the username, encrypted password, NAS IP address, and port. The early deployment of RADIUS was done using UDP port number 1645, which conflicts with the "data metrics" service. Because of this conflict, RFC 2865 officially assigned port

number 1812 for RADIUS. Most Cisco devices and applications offer support for either set of port numbers. The format of the request also provides information about the type of session that the user wants to initiate. For example, if the query is presented in character mode, the inference is "Service-Type = Exec-User," but if the request is presented in PPP packet mode, the inference is "Service Type = Framed User" and "Framed Type = PPP."

When the RADIUS server receives the Access-Request from the NAS, it searches a database for the username listed. If the username does not exist in the database, either a default profile is loaded or the RADIUS server immediately sends an Access-Reject message. This Access-Reject message can be accompanied by a text message indicating the reason for the refusal.

In RADIUS, authentication and authorization are coupled together. If the username is found and the password is correct, the RADIUS server returns an Access-Accept response, including a list of attribute-value pairs that describe the parameters to be used for this session. Typical parameters include service type (shell or framed), protocol type, IP address to assign the user (static or dynamic), access list to apply, or a static route to install in the NAS routing table. The configuration information in the RADIUS server defines what will be installed on the NAS. The figure below illustrates the RADIUS authentication and authorization sequence.



*Figure 9: Authentication and Authorization Sequence*

The accounting features of the RADIUS protocol can be used independently of RADIUS authentication or authorization. The RADIUS accounting functions allow data to be sent at the start and end of sessions, indicating the amount of resources (such as time, packets, bytes, and so on) used during the session. An Internet service provider (ISP) might use RADIUS access control and accounting software to meet special security and billing needs.

Transactions between the client and RADIUS server are authenticated through the use of a shared secret, which is never sent over the network. In addition, user passwords are sent encrypted between the client and RADIUS server to eliminate the possibility that someone snooping on an insecure network could determine a user's password.

## **Chapter 4 - Implementation, Testing and Maintenance**

### **4.1 Introduction to Languages, Tools and Technologies and standards used for Implementation**

#### **4.1.1. LINUX**

Linux is a computer operating system originally developed by Linus Torvalds as a research project. Linux runs on Intel, Mac, Sun, Dec Alpha, and several other hardware platforms.

##### **4.1.1.1. Linux Features**

- Linux is a full-featured, 32-bit multi-user/multi-tasking OS.
- Linux adheres to the common (POSIX) standards for UNIX .
- Native TCP/IP support.
- A mature X Windows GUI interface.
- Complete development environment. C, C++, Java, editors, version control systems.
- Open Source.

##### **4.1.1.2. Multi-User Operation**

In UNIX and Linux, all interactions with the OS are done through designated \"users\", who each have an identification ID (login name) and a password. UNIX allows different users to co-exist simultaneously and allows for different levels of users.

The most powerful user is called super user or \"root\", and has access to all files and processes. The super user does many of the system management tasks like adding regular users, file backups, system configuration etc.

Common users' accounts, which perform non-system type tasks, have restricted access to system-sensitive components to protect Linux from being accidentally or purposely damaged. In a moment you will enter a user account and start exploring the Linux file system.

##### **4.1.1.3. Why Linux**

Linux can operate as a web, file, smb (WinNT), Novell, printer, ftp, mail, SQL, masquerading, firewall, and POP server to name but a few. It can act as a graphics, C, C++, Java, Perl, Python, SQL, audio, video, and documentation, development workstation etc. Linux is a good solution for developers that need a stable and reliable platform that has open source code. It's not a good system for beginning developers that want a simple GUI interface to a programming language, although Linux has many GUI software development interfaces. Linux is ideal as a workstation also, and offers many customizable features not found in any other platform. It makes a good platform for dedicated workstations that have limited functions like in an educational or laboratory environment. It may not be ideal as a workstation for beginning users who want an instantly customizable universal WYSIWYG interface. Other systems provide solutions for this need. Still, Linux becomes easier to use on a daily basis. It's only a matter of time until Linux is accessible by everyone.

#### 4.1.1.4. Conventions

The following is a list of conventions supported in this manual:

##### <bash>

It indicates a command entered in a terminal by the user. When you see this sign, you are expected to enter these commands exactly as indicated and check that the results are consistent with what is written. If you see a problem, please ask the instructor to elaborate or clarify.

##### <tcsh>

It indicates commands in the tcsh shell. Most of these commands are to be completed after hours or at home. Since Linux advocates freedom of choice, we wish to make students aware of this option to bash.

### 4.1.2. Socket Programming

#### 4.1.2.1. About socket

An interface between application and network which is used for communication between processes. Once configured the application can – pass data to the socket for network transmission –receive data from the socket. To the kernel, a socket is an endpoint of communication. To an application, a socket is a file descriptor that lets the application read/write from/to the network. Clients and servers communicate with each by reading from and writing to socket descriptors. Uniquely identified by an internet address an end-to-end protocol (e.g. TCP or UDP) a port number. Two types of (TCP/IP) sockets Stream sockets (e.g. uses TCP) provide reliable byte-stream service. Datagram sockets (e.g. uses UDP) provide best-effort datagram service messages up to 65.500 bytes. Socket extend the convectional UNIX I/O facilities file descriptors for network communication extended the read and write system calls.

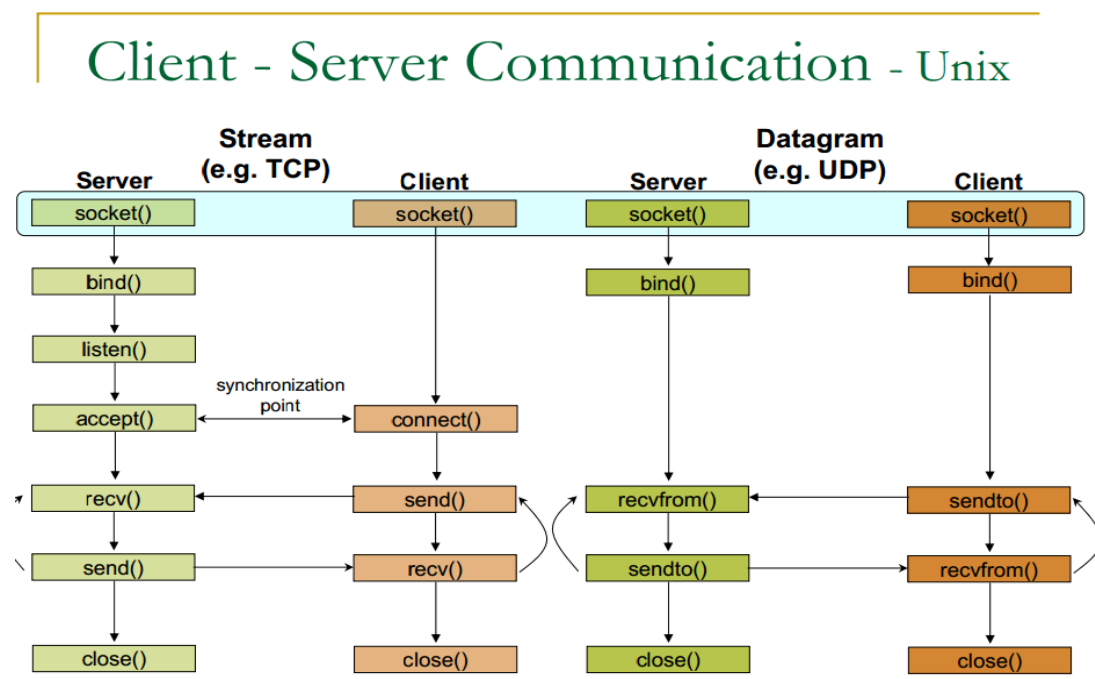


Figure 10: Client Server Communication

#### 4.1.2.2 Creating a Socket

- `int sockid = socket(family, type, protocol)`
- `sockid`: socket descriptor, an integer (like a file-handle)
- `family`: integer, communication domain
- `PF_INET`, IPv4 protocols, Internet addresses (typically used)
- `PF_UNIX`, Local communication, File addresses
- `type`: communication type
- `SOCK_STREAM` - reliable, 2-way, connection-based service
- `SOCK_DGRAM` - unreliable, connectionless, messages of maximum length
- `protocol`: specifies protocol
- `IPPROTO_TCP` `IPPROTO_UDP`
- usually set to 0 (i.e., use default protocol)
- upon failure returns -1

#### 4.1.2.3 Binding a socket

- `int status = bind(sockid, &addrport, size);`
- `sockid`: integer, socket descriptor
- `addrport`: struct `sockaddr`, the (IP) address and port of the machine
- for TCP/IP server, internet address is usually set to `INADDR_ANY`, i.e., chooses any incoming interface.
- `size`: the size (in bytes) of the `addrport` structure
- `status`: upon failure -1 is returned

#### 4.1.2.4 Connecting a Socket

- The client establishes a connection with the server by calling `connect()`
- `int status = connect(sockid, &foreignAddr, addrlen)`
- `sockid`: integer, socket to be used in connection
- `foreignAddr`: struct `sockaddr`: address of the passive participant
- `addrlen`: integer, `sizeof(name)`
- `status`: 0 if successful connect, -1 otherwise
- `connect()` is blocking

#### 4.1.2.5 Accepting a Socket

- The server gets a socket for an incoming client connection by calling `accept()`
- `int s = accept(sockid, &clientAddr, &addrLen)`
- `s`: integer, the new socket (used for data-transfer)
- `sockid`: integer, the orig. socket (being listened on)
- `clientAddr`: struct `sockaddr`, address of the active participant filled in upon return
- `addrLen`: `sizeof(clientAddr)`: value/result parameter must be set appropriately before call adjusted upon return

#### 4.1.2.6 Exchanging Data With Stream Socket

- `int count = send(sockid, msg, msgLen, flags)`
- `msg: const void[],` message to be transmitted
- `msgLen: integer,` length of message (in bytes) to transmit
- `flags: integer,` special options, usually just 0
- `count: # bytes transmitted` (-1 if error)
- `int count = recv(sockid, recvBuf, bufLen, flags)`
- `recvBuf: void[],` stores received bytes
- `bufLen: # bytes received`
- `flags: integer,` special options, usually just 0
- `count: # bytes received` (-1 if error)

#### 4.1.2.7 Exchanging Data With Datagram Socket

- `int count = sendto(sockid, msg, msgLen, flags, &foreignAddr, addrlen)`
- `msg, msgLen, flags, count:` same with `send()`
- `foreignAddr: struct sockaddr,` address of the destination
- `addrlen: sizeof(foreignAddr)`
- `int count = recvfrom(sockid, recvBuf, bufLen, flags, &clientAddr, addrlen)`
- `recvBuf, bufLen, flags, count:` same with `recv()`
- `clientAddr: struct sockaddr,` address of the client
- `addrlen: sizeof(clientAddr)`

#### 4.1.3. Tools to be used:

- Open source FreeRadius Tool (To use as a RADIUS Server)
- Wireshark (Network Packet Analyzer Tool)

#### Platform to be used:

- Linux

#### Language to be used:

- C language
  - Socket Programming
  - Multithreading
  - IPC (Shared Memory)
- Tcl/Tk

For developing a GUI which we show the stats related to successful and failed authentication of all the Dial-In users.

## 4.2 Gantt Chart (Project Schedule)

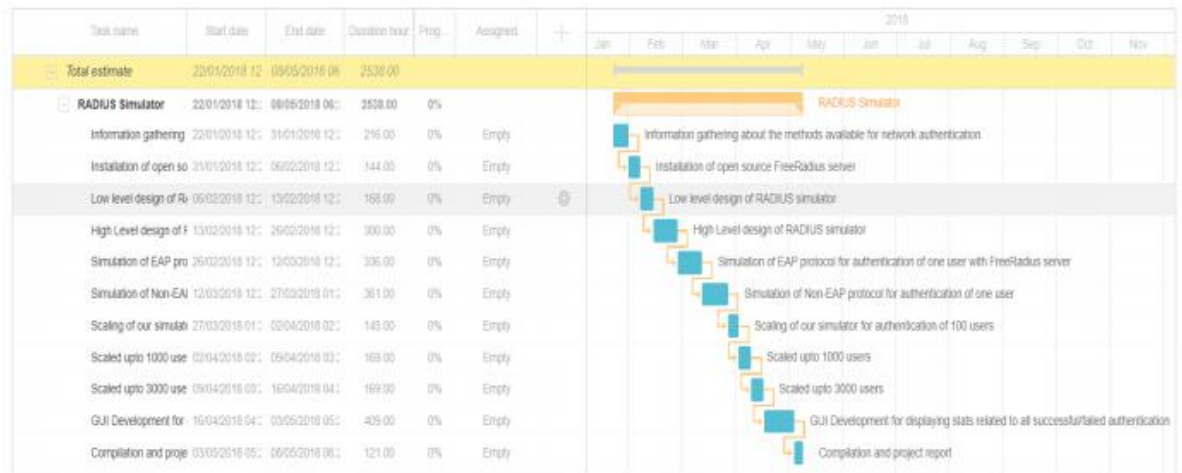


Figure 11: Gantt Chart

## 4.3 Testing

### System Testing

Software testing is a crucial element of software quality and represents the ultimate review of specialization design and coding. No program or design is perfect, communication between the user and the designers not always complete or clear, and time is usually short, this result is errors. The number and nature of error in a new design depend on several features:

- Communication between the user and the designer.
- The programmer's ability to generate a code that reflects exactly the system specifications.
- The time frame for the design.

Testing is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. Inadequate testing or non-testing leads to error, that may not appear until months later but it can be fatal in future. This creates two problems:

- The time lag between the cause and the appearance of the problem.
- The effect of the system errors on files and records within the system.

Effective testing early in the process translates directly into long-term cost saving from the reduced number of errors.



Some major types of testing methods are:

1. White box testing
2. Black box testing
3. Stress testing
4. Performance testing

**White box testing:**

White box sometimes called “glass box testing” is a test case design uses the control structure of the procedural design to derive test case.

Using white box testing methods, the following tests were made on the system.

- A) All independent paths within a module have been exercised once. In our system, ensuring that case was selected and executed checked all case structures. The bugs that were prevailing in some part of the code were fixed.
- B) All logical decisions were checked for the truth and falsity of the values.

**Black box testing:**

Black box testing focuses on the functional requirements of the software. This is black box testing enables the software engineering to derive a set of input conditions that will fully exercise all functional requirements for a program. Black box testing is not an alternative to white box testing rather it is complementary approach that is likely to uncover a different class of errors that white box methods like..

- 1) Interface errors
- 2) Performance in data structure
- 3) Performance errors
- 4) Initializing and termination errors

**Performance testing:**

In this type of testing, we check for the performance of our project. Performance testing is the major part of our testing. In this type of testing all the modules performance is checked out, how it behave under various conditions. In this overall performance of the system is checked out and it gives variances, if any, so the developer can take the appropriate action in response to the variances.

## Chapter 5 - Results and Discussions

### 5.1 User Interface Representation

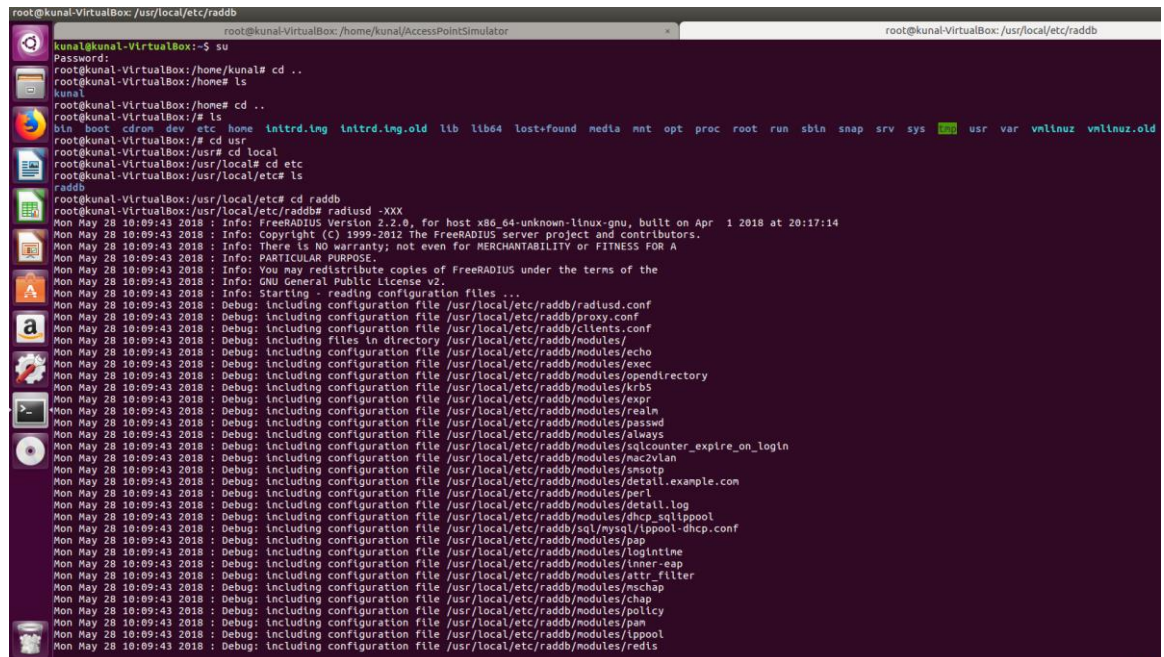
#### 5.1.1 Brief Description of Various Modules of the system

- **FreeRADIUS Server:** FreeRADIUS is the most popular open source RADIUS server and the most widely deployed RADIUS server in the world. It supports all common authentication protocols, and the server comes with a PHP-based web user administration tool called dialupadmin. The FreeRADIUS Suite includes a RADIUS server, a BSD-licensed RADIUS client library, a PAM library, an Apache module, and numerous additional RADIUS related utilities and development libraries. In this project we have made use of it as a free open source server.
- **RADIUS Client-Side program written in C:** The heart of our project lies in this module, as in this we design and formulate the client-side architecture of the RADIUS. It is here we develop the AAA structure and scale our program such that the whole RADIUS environment can support multiple client requests simultaneously. Various concepts of C such as Multithreading, Socket Programming (UDP), Message IPC, Message Synchronisation are used to develop the complete architecture.
- **Graphical User Interface:** In order to provide flexibility and ease of access to the services provided by the simulator our project supports a GUI which is a simple menu driven interface through which the user can provide the credentials and create the input file, the server after accessing the input file runs the C program at the back end which takes care of the AAA services.
- **Wireshark:** Wireshark is the world's foremost and widely-used network protocol analyzer. It lets you see what's happening on your network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational institutions. Wireshark lets the user put network interface controllers into promiscuous mode (if supported by the network interface controller), so they can see all the traffic visible on that interface including unicast traffic not sent to that network interface controller's MAC address.

## 5.2 Snapshots of system with brief detail of each module

### 5.2.1 FreeRADIUS Server

Server can be executed over the machine by using the radiusd command in the terminal. This command is found under the path: `/usr/local/etc/raddb`



```
root@kunal-VirtualBox: /usr/local/etc/raddb
kunal@kunal-VirtualBox:~$ su
Password:
root@kunal-VirtualBox: /home/kunal# cd ..
root@kunal-VirtualBox: /home# ls
kunal
root@kunal-VirtualBox: /home# cd ..
root@kunal-VirtualBox: /# ls
bin boot cdrom dev etc home initrd.img initrd.img.old lib lib64 lost+found media mnt opt proc root run sbin snap srv sys usr var vmlinuz vmlinuz.old
root@kunal-VirtualBox: /# cd usr
root@kunal-VirtualBox: /usr# cd local
root@kunal-VirtualBox: /usr/local# cd etc
root@kunal-VirtualBox: /usr/local/etc# ls
radius
root@kunal-VirtualBox: /usr/local/etc# cd raddb
root@kunal-VirtualBox: /usr/local/etc/raddb# radiusd -XXX
Mon May 28 10:09:43 2018 : Info: FreeRADIUS Version 2.2.0, for host x86_64-unknown-linux-gnu, built on Apr 1 2018 at 20:17:14
Mon May 28 10:09:43 2018 : Info: Copyright (C) 1999-2012 The FreeRADIUS server project and contributors.
Mon May 28 10:09:43 2018 : Info: There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
Mon May 28 10:09:43 2018 : Info: PARTICULAR PURPOSE.
Mon May 28 10:09:43 2018 : Info: You may redistribute copies of FreeRADIUS under the terms of the
Mon May 28 10:09:43 2018 : Info: GNU General Public License v2.
Mon May 28 10:09:43 2018 : Info: Starting - reading configuration files ...
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/radiusd.conf
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/proxy.conf
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/clients.conf
Mon May 28 10:09:43 2018 : Debug: including files in directory /usr/local/etc/raddb/modules/
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/echo
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/openssl
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/krb5
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/expr
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/exec
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/realn
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/passwd
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/always
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/sqlcounter_expire_on_login
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/smsotp
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/detail.example.com
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/perl
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/detail.log
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/dhcp_sqlpool
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/sql/mysqlippool-dhcp.conf
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/pap
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/logintime
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/inner-eap
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/attr_filter
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/mchap
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/chap
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/pam
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/ippool
Mon May 28 10:09:43 2018 : Debug: including configuration file /usr/local/etc/raddb/modules/redis
```

Figure 12: RADIUS server

- **-X option:** Debugging mode. Equivalent to "-sfixx -l stdout". When trying to understand how the server works, ALWAYS run it with "radiusd -X". For production servers, use "raddebug". Finer-grained debug mode. In this mode the server will print details of every request on it's stdout output. You can specify this option multiple times (-x -x or -xx) to get more detailed output.

```

root@kunal-VirtualBox: /home/kunal/AccessPointSimulator
Mon May 28 10:09:44 2018 : Debug: callerId = yes
Mon May 28 10:09:44 2018 : Debug: Module: Checking post-proxy [...] for more modules to load
Mon May 28 10:09:44 2018 : Debug: Module: Checking post-auth [...] for more modules to load
Mon May 28 10:09:44 2018 : Debug: Module: Instantiating module "attr_filter.access_reject" from file /usr/local/etc/raddd/modules/attr_filter
Mon May 28 10:09:44 2018 : Debug: attr_filter attr_filter.access_reject {
Mon May 28 10:09:44 2018 : Debug:   attrfile = "/usr/local/etc/raddd/attrs.access_reject"
Mon May 28 10:09:44 2018 : Debug:   key = "%{User-Name}"
Mon May 28 10:09:44 2018 : Debug:   relaxed = no
Mon May 28 10:09:44 2018 : Debug: }
Mon May 28 10:09:44 2018 : Debug: reading patrlist file /usr/local/etc/raddd/attrs.access_reject
Mon May 28 10:09:45 2018 : Debug: } # modules
Mon May 28 10:09:45 2018 : Debug: Server inner-tunnel { # from file /usr/local/etc/raddd/sites-enabled/inner-tunnel
Mon May 28 10:09:45 2018 : Debug:   modules {
Mon May 28 10:09:45 2018 : Debug:     Module: Checking authenticate [...] for more modules to load
Mon May 28 10:09:45 2018 : Debug:     Module: Checking authorize [...] for more modules to load
Mon May 28 10:09:45 2018 : Debug:     Module: Checking session [...] for more modules to load
Mon May 28 10:09:45 2018 : Debug:     Module: Checking post-proxy [...] for more modules to load
Mon May 28 10:09:45 2018 : Debug:     Module: Checking post-auth [...] for more modules to load
Mon May 28 10:09:45 2018 : Debug:   } # modules
Mon May 28 10:09:45 2018 : Debug: } # server
Mon May 28 10:09:45 2018 : Debug: radlud: ### Opening IP addresses and Ports ###
Mon May 28 10:09:45 2018 : Debug: listen {
Mon May 28 10:09:45 2018 : Debug:   type = "auth"
Mon May 28 10:09:45 2018 : Debug:   ipaddr = 10.0.2.15
Mon May 28 10:09:45 2018 : Debug:   port = 1812
Mon May 28 10:09:45 2018 : Debug: }
Mon May 28 10:09:45 2018 : Debug: listen {
Mon May 28 10:09:45 2018 : Debug:   type = "acct"
Mon May 28 10:09:45 2018 : Debug:   ipaddr = 10.0.2.15
Mon May 28 10:09:45 2018 : Debug:   port = 1813
Mon May 28 10:09:45 2018 : Debug: }
Mon May 28 10:09:45 2018 : Debug: listen {
Mon May 28 10:09:45 2018 : Debug:   type = "control"
Mon May 28 10:09:45 2018 : Debug:   listen {
Mon May 28 10:09:45 2018 : Debug:     socket = "/usr/local/var/run/radlud/radlud.sock"
Mon May 28 10:09:45 2018 : Debug:   }
Mon May 28 10:09:45 2018 : Debug: }
Mon May 28 10:09:45 2018 : Debug: listen {
Mon May 28 10:09:45 2018 : Debug:   type = "auth"
Mon May 28 10:09:45 2018 : Debug:   ipaddr = 127.0.0.1
Mon May 28 10:09:45 2018 : Debug:   port = 18120
Mon May 28 10:09:45 2018 : Debug: }
Mon May 28 10:09:45 2018 : Debug: ... adding new socket proxy address * port 40831
Mon May 28 10:09:45 2018 : Debug: Listening on authentication address 10.0.2.15 port 1812
Mon May 28 10:09:45 2018 : Debug: Listening on accounting address 10.0.2.15 port 1813
Mon May 28 10:09:45 2018 : Debug: Listening on command file /usr/local/var/run/radlud/radlud.sock
Mon May 28 10:09:45 2018 : Debug: Listening on authentication address 127.0.0.1 port 18120 as server inner-tunnel
Mon May 28 10:09:45 2018 : Debug: Listening on proxy address 10.0.2.15 port 1814
Mon May 28 10:09:45 2018 : Info: Ready to process requests.

```

Figure 13: Running Server

## 5.2.2 Graphic User Interface

- The GUI is a simple menu based structure which performs specific back end operations corresponding to each and every option of the menu.

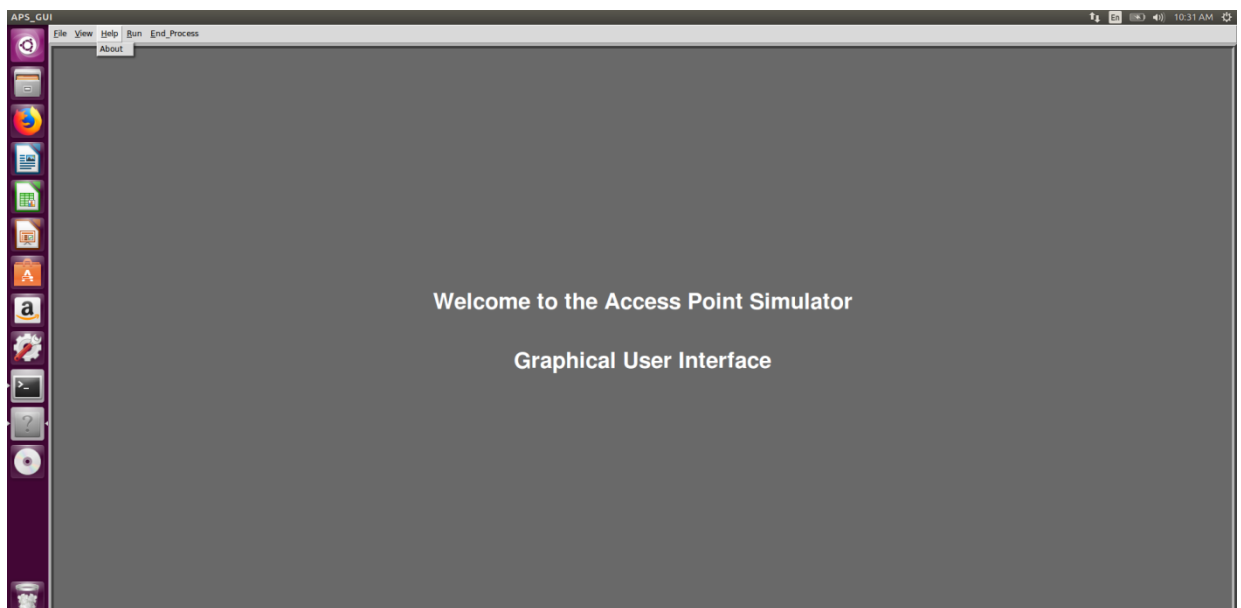


Figure 14: GUI

- File tab:** Under this tab we find option 'New' and after clicking on this we're presented with a new window which asks for the client information.

top

Clients Info Users Info Generate Files

Enter the Client IP 10.0.2.15

Enter the Client Port 34799

Enter the Server IP 10.0.2.15

Figure 15: File Tab of GUI

- After entering the client information, next we're asked for the user specific credentials.

top

Clients Info Users Info Generate Files

Enter the no. of Users per Client 1

Enter the Start IMSI 1234567890123456

Enter the timeout value 5

Enter the Framed IP 10.10.20.0

Figure 16: User specific credentials

- Finally after entering all the credentials we go and click on the generate files button to populate our database of users and clients which can be later on cross verified in the file 'inputfile'.

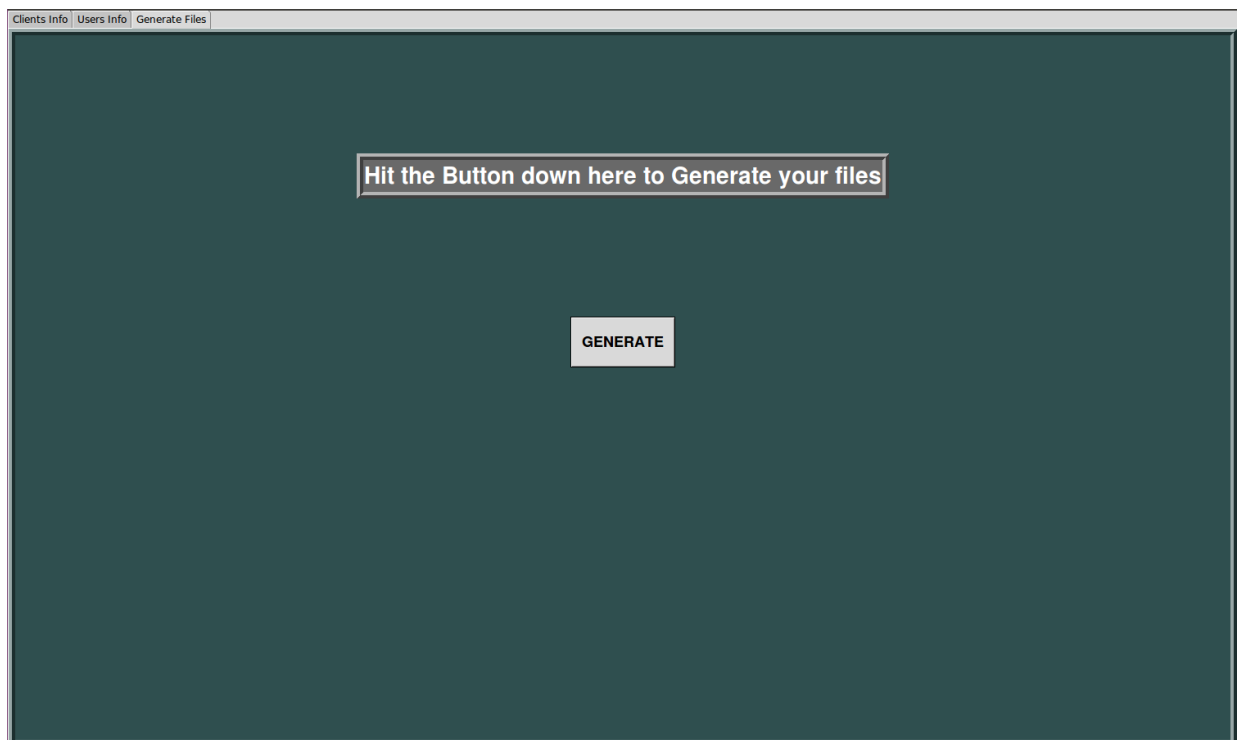


Figure 17: Generate tab

- Executing the backend C program: After all the credentials are loaded, we can execute our client side code over this data. To do this we simply click on the 'myradclient' option under the run tab of the GUI parent window.

### 5.2.3 Wireshark

This is a packet analyser tool which lets us see what's happening on our network at a microscopic level. It gives a complete description of every RADIUS message along with its attributes. We can easily see the Access Request, Access Accept, Accounting Request, Accounting response packets between the client and the server IP.

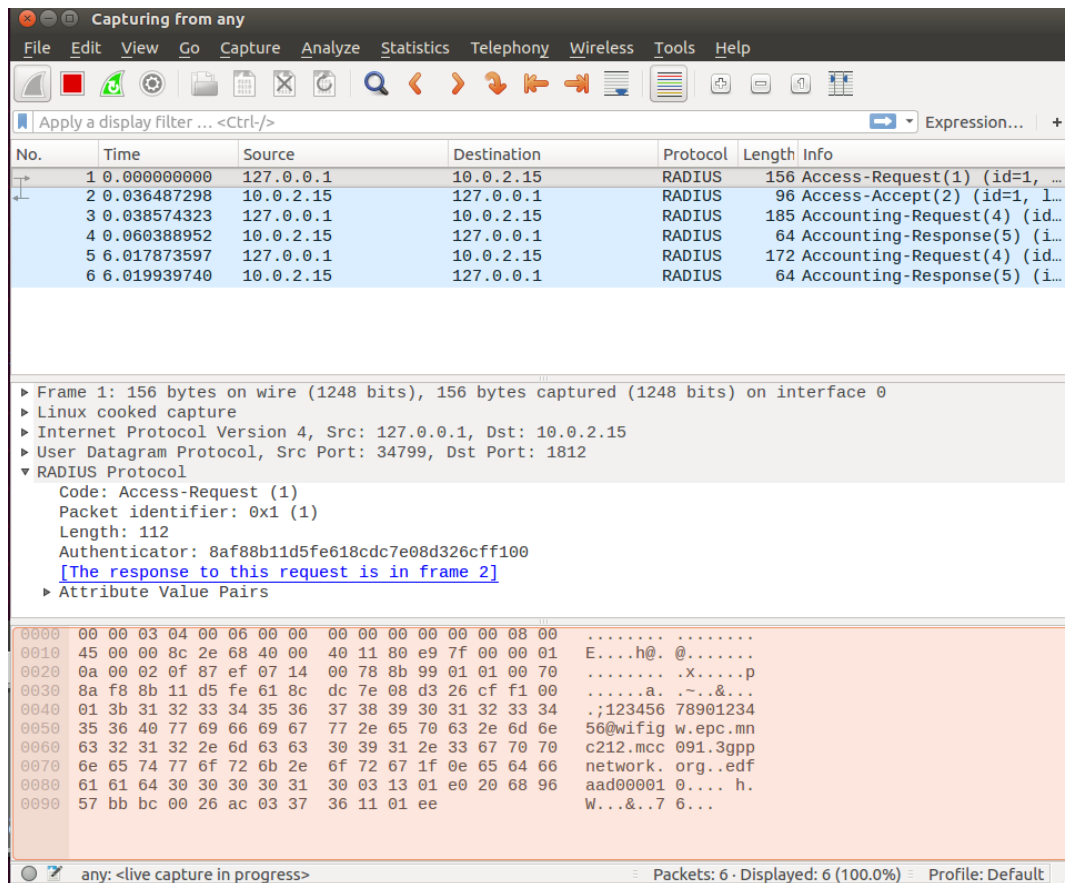


Figure 18: Wireshark

## 5.3 Back Ends Representation

### 5.3.1 Snapshots of Database Tables with brief description

- Inputfile:** All the information that GUI accepts from the user is stored under this file which is further used by the server for authentication mechanisms. Inside this file, line starting with "G:" contains information about client IP, client port, user profile name and secret(which should be same as configured in 'client.conf' file at server side). Next line starts with "US#" contains information about all the UE's that a particular client simulate. Each UE contains info about maximum number of UE which are going to be authenticated, NAI, mask value (it decides which AVP needs to go inside the packet),framed protocol, framed-IPaddr, Nas-ID, vendor Specific Location name and nonEapType.

### Mask value description : -

- \* 1st bit : NAS\_PORT
- \* 2nd bit : NAS\_IP\_ADDRESS
- \* 3rd bit : USER\_NAME
- \* 4th bit : EAP\_MESSAGE
- \* 5th bit : MESSAGE\_AUTHENTICATOR
- \* 6th bit : STATE
- \* 7th bit : called\_station\_id
- \* 8th bit : calling\_station\_id
- \* 9th bit : chap\_passwd
- \* 10th bit : FRAMED\_PROTOCOL
- \* 11th bit : NAS\_ID
- \* 12th bit : FRAMED\_IP\_Address
- \* 13th bit : Acct\_Status\_Type



Figure 19: Input file

- **users:** This file contains the user specific information that is information like User-name, User-Password, Class, Session-ID which is provided by the RADIUS server automatically when it successfully authenticates the client.

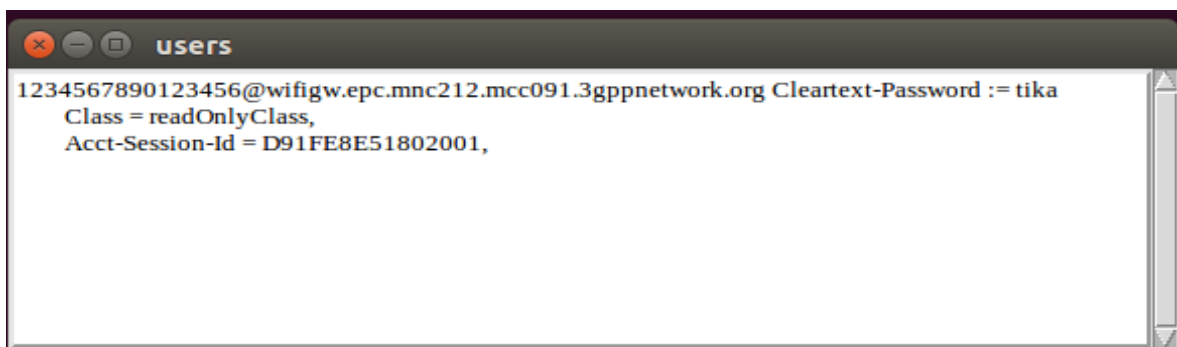


Figure 20: Users file



## **Chapter 6- Conclusion and future scope**

The developed system successfully implements the desirable properties of the existing system while overcoming its shortcomings. The problem of low scalability was improved to some extent. Here packets are not that heavy and they are handled more easily by networking devices. It eases the use and limits the authentication time response. Also, if the request to primary authentication server fails, the server does not need to wait for the reply packets (UDP is connectionless). It provides enhanced reporting and tracking based on client usernames.

There are some areas of the protocol that we need to work upon. Until now due to the limited resources available we have been able to extend the scalability to 40 users, i.e. we are able to process upto 40 requests at a time. So scalability can be increased more to some extent. The present tool for Radius doesn't define exact retransmission behaviour. Therefore more work could be done on the retransmission of messages. With RADIUS only clients can make re-authentication requests. This is done by simply sending a new authentication request to the server. However, the server cannot demand re-authentication on demand, as it cannot initiate messages. This behaviour can also be worked on.

## References:

1. John Vollbrecht (2006). "The Beginnings and History of RADIUS" (PDF). Interlink Networks. Retrieved 2009-04-15
2. RFC 2865 Remote Authentication Dial In User Service .
3. RFC 2866 RADIUS Accounting.
4. *Brien Posey (2006-08-31). "SolutionBase: RADIUS deployment scenarios" (PDF). TechRepublic. Retrieved 2009-04-15.*
5. *"How Does RADIUS Work?". Cisco. 2006-01-19. Retrieved 2009-04-15..*
6. Jonathan Hassell (2003). *RADIUS: Securing Public Access to Private Resources*. O'Reilly Media. pp. 15–16.
7. C. Rigney, S. Willens, A. Rubens, W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", IETF RFC 2865, June 2000.
8. *"RFC 6733 - Diameter Base Protocol". PROPOSED STANDARD. Standards Track. ISSN 2070-1721. Retrieved 12 October 2014.*
9. "FreeRADIUS: The world's most popular RADIUS Server - About". freeradius.org. Retrieved 2014-11-12.
10. *"Wireshark - About". The Wireshark Foundation. Retrieved January 30, 2018*