Implementing TCP/IP Stack in C

Software Requirements Specification Document

Software Engineering Project (UCS503)

Report Evaluation

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Submitted To:

Mr. Abhishek Jain

Prepared By:

Dhairya 101916016

Parth 101916015

Shrey 101916017

23/04/2021

TABLE OF CONTENTS

Revision History

Chapter	Topic	Page
1.	**Introduction**	4
1.1	Purpose	4
1.2	Document Conventions	4
1.3	Intended Audience and Reading Suggestions	4
1.4	Project Scope	5
1.5	References	5
2.	**Overall Description**	8
2.1	Product Perspective	8
2.2	Product Features	9
2.3	User Classes and Characteristics	9
2.4	Operating Environment	9
2.5	Design and Implementation Constraints	10
2.6	User Documentation	10
2.7	Assumptions and Dependencies	10
3.	**System Features**	11
3.1	System Feature 1	11-12
3.2	System Feature 2 (and so on)	11-12

4. **External Interface Requirements** 12

4.1 User Interfaces 13

4.2 Hardware Interfaces 13

4.3 Software Interfaces 13

4.4 Communications Interfaces 14

5. **Other Nonfunctional Requirements** 14

5.1 Performance Requirements 14

5.2 Safety Requirements 14

5.3 Security Requirements 14

5.4 Software Quality Attributes 15

6. **Other Requirements**

Appendix A: Analysis Model 16-38

Appendix B: Glossary 39

Appendix C: Issues List --

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

Since the internet has become a popular platform routers, switches, hubs, and servers have become very important to manage. In the core of the Internet, routers owned by various Internet Service Providers will coordinate with one another using the Border Gateway Protocol to exchange routing information. This helps them direct traffic from particular destination hosts to particular routers, helping packets make their way to their final destination. **Our main purpose is to mimic the network topology and Implement TCP/IP stack in C.**

1.2 Document Conventions

Typeface	**Indicates**
Font	Open Sans
Bold	Mainly for headings
Italics	Mainly URLs in References
Blue-Underline	Used for URLs

1.3 Intended Audience and Reading Suggestions

This document is intended for developers, project managers, marketing staff, users, testers, and documentation writers. This SRS contains information related to the Functional requirements and non-functional requirements. You can start reading from the purpose heading and proceed as further with no specific path needed.

1.4 Project Scope

This C/C++ Linux Project will be an implementation-based network project which consists of Network Socket Programming. The admin console will allow the user to see the practical implementation of the OSI Model.

The **main purpose** of this software is that one can mimic the network topology and the developer as well as the end user can see the packet Journey Upwards and Downwards through layers of TCP/IP Stack (= OSI Model). Understand End-To-End Architecture and Design of Network Application and TCP/IP Stack. The developer of the course will be creating nodes, links connecting nodes, configuring network parameters on nodes, sending and receiving traffic streams.

1.5 References

https://www.quora.com/How-does-the-Internet-work

https://en.wikipedia.org/wiki/OSI_model

https://www.geeksforgeeks.org/tcp-ip-model/

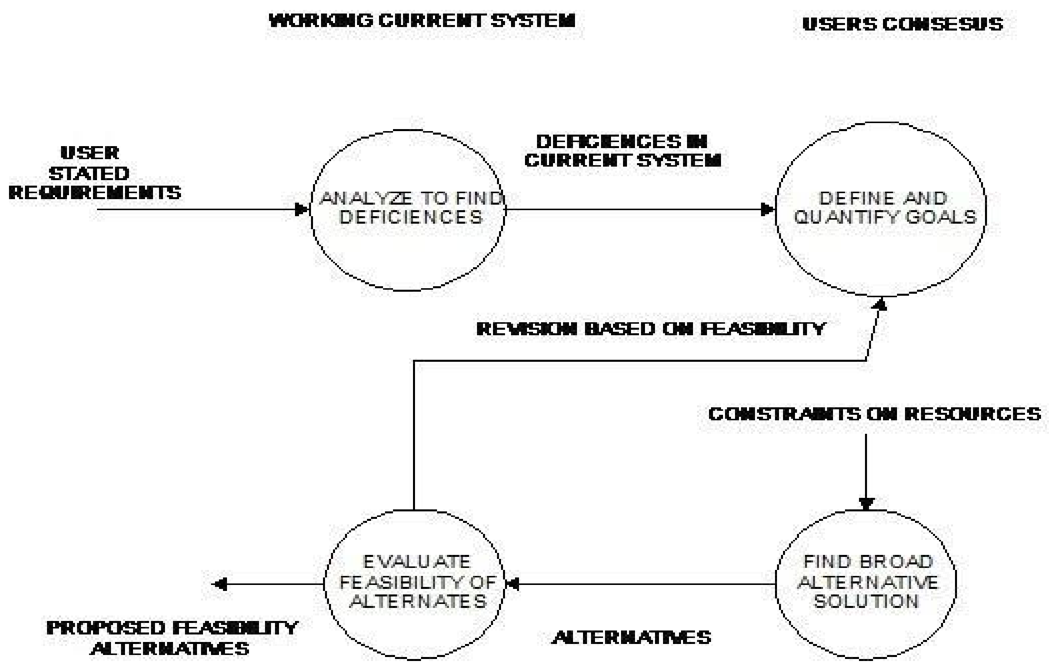
https://www.cisco.com/c/en_in/solutions/small-business/resource-center/networking/network-switch-vs-router.html

FEASIBILITY STUDY

A feasibility study is the process of determination of whether or not a project is **worth doing**. The contents and recommendations of this feasibility study helped us as a sound basis for deciding how to precede the project. It helped in making decisions such as which software to use, hardware combinations, etc.

The following is the process diagram for feasibility analysis. In the diagram, the feasibility analysis starts with the user set of requirements.

The next step is to check for the deficiencies in the existing system. By evaluating the above points a fresh idea is conceived to define and quantify the required goals. User consent is very important for the new plan. Along with, implementing the new system, the ability of the organization is also checked. Besides that, a set of alternatives and their feasibility is also considered in case of any failure in the proposed system. Thus, **the feasibility study** is an important part of software development.


WORKING CURRENT SYSTEM
USERS CONSESUS
USER STATED REQUIREMENTS
ANALYZE TO FIND DEFICIENCES
DEFICIENCES IN CURRENT SYSTEM
DEFINE AND QUANTIFY GOALS
REVISION BASED ON FEASIBILITY
CONSTRAINTS ON RESOURCES
EVALUATE FEASIBILITY OF ALTERNATES
FIND BROAD ALTERNATIVE SOLUTION
PROPOSED FEASIBILITY ALTERNATIVES
ALTERNATIVES

PROCESS DIAGRAM FOR FEASIBILITY ANALYSIS

In the SDLC (Systems Development Life Cycle) of our project we maintained a number of feasibility checkpoints between the two phases of the SDLC.

These checkpoints indicate that the management decision to be made after a phase is complete.

We conducted three tests for Project feasibility namely, **Technical, Economical, and Operational feasibilities.**

Technical Feasibility

Technical feasibility determines whether the work for the project can be done with the existing equipment, software technology, and available personnel. Technical feasibility is concerned with specifying equipment and software that will satisfy the user requirement.

This project is feasible on technical remarks. The proposed system can run on any machines supporting **Linux** and **Internet** services and works on the best software and hardware that has been used while designing the system so it would be feasible in all technical terms of feasibility.

Economical Feasibility

Economical feasibility determines whether there are sufficient benefits in creating to make the cost acceptable, or if the cost of the system is too high.

We classified the costs of our project according to the phase in which they occur. As we know that the system development costs are usually one-time costs that will not recur after the project has been completed. For calculating the Development costs we evaluated certain cost categories viz.

(i) Personnel costs

(ii) Computer usage

(iii) Training

(iv) Supply and equipment costs

(v) Cost of any new computer equipment and software

Operational Feasibility

Operational feasibility is a measure of how people feel about the system. It measures the acceptability of a solution. It refers to projecting whether the system will operate and be used once it is installed. Our Project is operationally feasible since there is no need for special training of staff member and whatever little instructing on this system is required can be done so quite easily and quickly as it is essentially This project is being developed keeping in mind the general people who one have intermediate knowledge of computer operation and they can easily stimulate OSI model on the console.

2. Overall Description

2.1 Product Perspective

The purpose of this project is to provide a simple C implementation of a full TCP/IP (layer 2 and layer 3 of the OSI model) stack for educational purposes. The entire design of the project is based on an abstraction between the layers and separation into independent layers while breaking down each part so that it can be understood by the end-users.

2.2 Product Features

- The end user can build a MultiNode Topology Emulation of Routers and Switches.

- End users can also Implement DataLink Layer (L2 routing), including ARP.

- Implement Network Layer (L3 routing)

- The user can also execute commands to troubleshoot the network built.

- All users are authenticated to avail of the services.

- **Help documents are** also included for end-users benefit.

2.3 User Classes and Characteristics

The **user** must have the basic knowledge regarding networking so that they can use and understand the implementation of the OSI Model and can use the features of the product to understand more on the practical working of the network.

2.4 Operating Environment

The application will run over the command line interface on some kind of Operating system (Unix, Linux, and Mac which uses UNIX kernel).

2.5 Design and Implementation Constraints

- The end-user of the product must have intermediate knowledge of networking so that he/she can use the product to stimulate the OSI Model and run commands to troubleshoot the network.

- Network layers 1, 6, and 7 are not going to be implemented in this project due to the lack of host servers to transfer packets.

- The project has been entirely built on Command-line Interface instead of Graphical Interface so the user must be familiar with the basics of cmd.

2.6 User Documentation

Help documents will be available for the user. Apart from this, some basic video links will also be provided to understand the basic fundamentals of the working of a network.

2.7 Assumptions and Dependencies

- The end-user of the product must have basic knowledge of L2 and L3 routing.

- The end-user must be operating the product on a Linux-based machine.

3. System Features

3.1 Build a Multi-Node Topology Emulation of Routers & Switches

3.1.1 Description and Priority

The user would be able to construct and build networking emulations, using the logics of topology, adding/deleting the routers and switches in the network created.

3.1.2 Stimulus/Response Sequences (Functional requirement)

- The user searches for the topologies through which they want to construct the Network.
- The user inputs the required values to construct the graph that further constructs the topology through the CLI.
- The user can edit the topology and network structure anytime during the project.

3.2 Implement Data Link and Network Layers (L2 & L3 routing), including ARP

3.2.1 Description and Priority

The user will be able to transfer the data frames between the Hosts connected to the nodes. The ARP will help map the Logical Address of each node to the Physical address of each interface connected to the nodes.

3.2.2 Stimulus/Response Sequences (Functional requirement)

- The user will be entering the required inputs for the type of network topology needed.

- The encapsulation of network-layer data packets into frames happens and the User is able to see the Topology constructed.
- For further clarity, the user can easily edit the values/topology as per the needs.

3.3 Constructing routing table and ARP table for each Topology that User enters

3.4.1 Description and Priority

The user would be able to get all the entries as well as the full Routing and ARP tables of the desired topology they chose.

3.4.2 Stimulus/Response Sequences (Functional requirement)

- The topology and all the required inputs have been received by the user.
- The user will be able to see and analyze the entries of the Routing and ARP tables.
- Users can always change/edit the topology according to their needs and requirements.
- The earlier work can be stored, if the user wants to look back at the previously created network.

4. EXTERNAL INTERFACE REQUIREMENTS

The interface ought to be command-line based.

System features should be improved for more features.

4.1 User Interface

It is designed to be functional and minimal in its styling. All products will be displayed in a text format. Command-line will be used to set up the dashboard of the project where users can run commands and build a network topology.

- Simple Command Line Interface (CLI) for running commands in the program.
- Easy execution of commands via just terminal of UNIX.
- Dynamically configurable interface
- Help wizard can be used for displaying manuals of functions.

4.2 Hardware Interface

The System must run over a LINUX-based operating system, all the hardware shall be required to work and may or may not be connected to the internet, this all will be a hardware interface for the system.

- Machine Used: Ubuntu 19.04, GCC compiler
- Processor speed of 2 GHz or more
- Ram of 512MB and above
- Disk space: 1500 MB or more for components

4.3 Software Interface

The product is implemented using C language so a GCC compiler for Linux is mandatory to build the C files. LibCli Library will also be required to implement the command-line interface to the product. Graph Data Structure is also required to implement the network topology.

4.4 Communication Interface

Command Line Interface/Terminal

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- The device to be used to implement the TCP/IP stack needs to be a UNIX-based Operating System.
- This software will assume that C/C++ build Libraries and GCC compiler is available on the machine, and so the software won't contain any compiler files.
- The User interface will be the Command Line Interface/Terminal.
- Users need no internet access on their devices.

5.2 Safety Requirements

The user data/application is stored automatically so if the user system crashes or faces any bug issues, the previous work won't get destroyed from the memory.

5.3 Security Requirements

Random Hash Codes are generated to get unique MAC addresses for each interface connected to each and every node.

5.4 Software Quality Attributes

- **Reliability**: The system is completely reliable as it has been checked for several trial runs before it is made available to the users.

- **Adequacy**: The input required of the user is limited to only what is necessary.

- **Maintainability:** The system is completely suitable for debugging (localization and correction of errors) and for modification and extension of functionality.

- **Readability:** The system is made in such a way that it can be easily used by everyone.

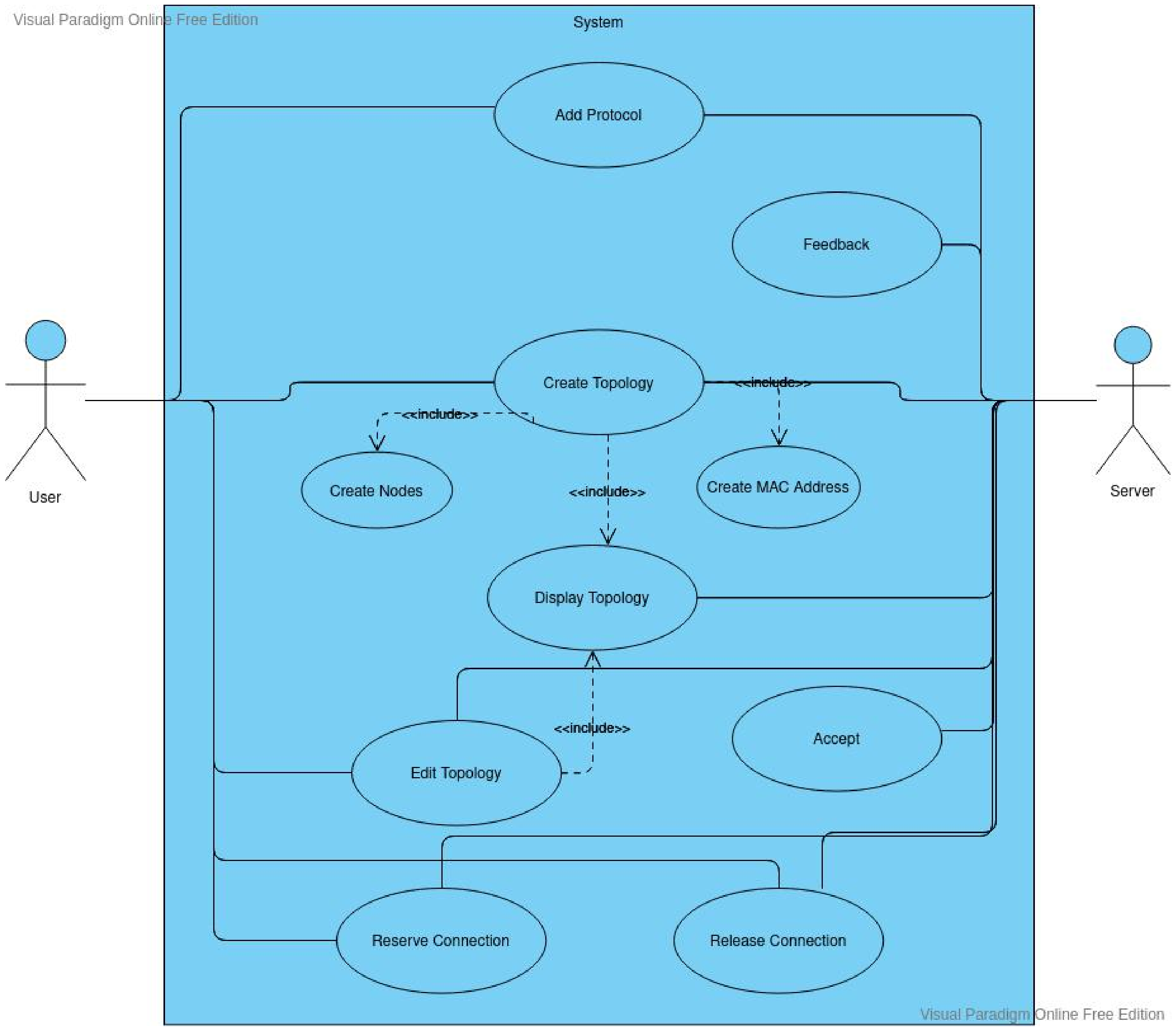
- **Extensibility**: Required modifications at the appropriate locations can be made without undesirable side effects.

- **Efficiency**: the ability of a software system to fulfill its purpose with the best possible utilization of all necessary resources (time, storage, transmission channels, and peripherals).

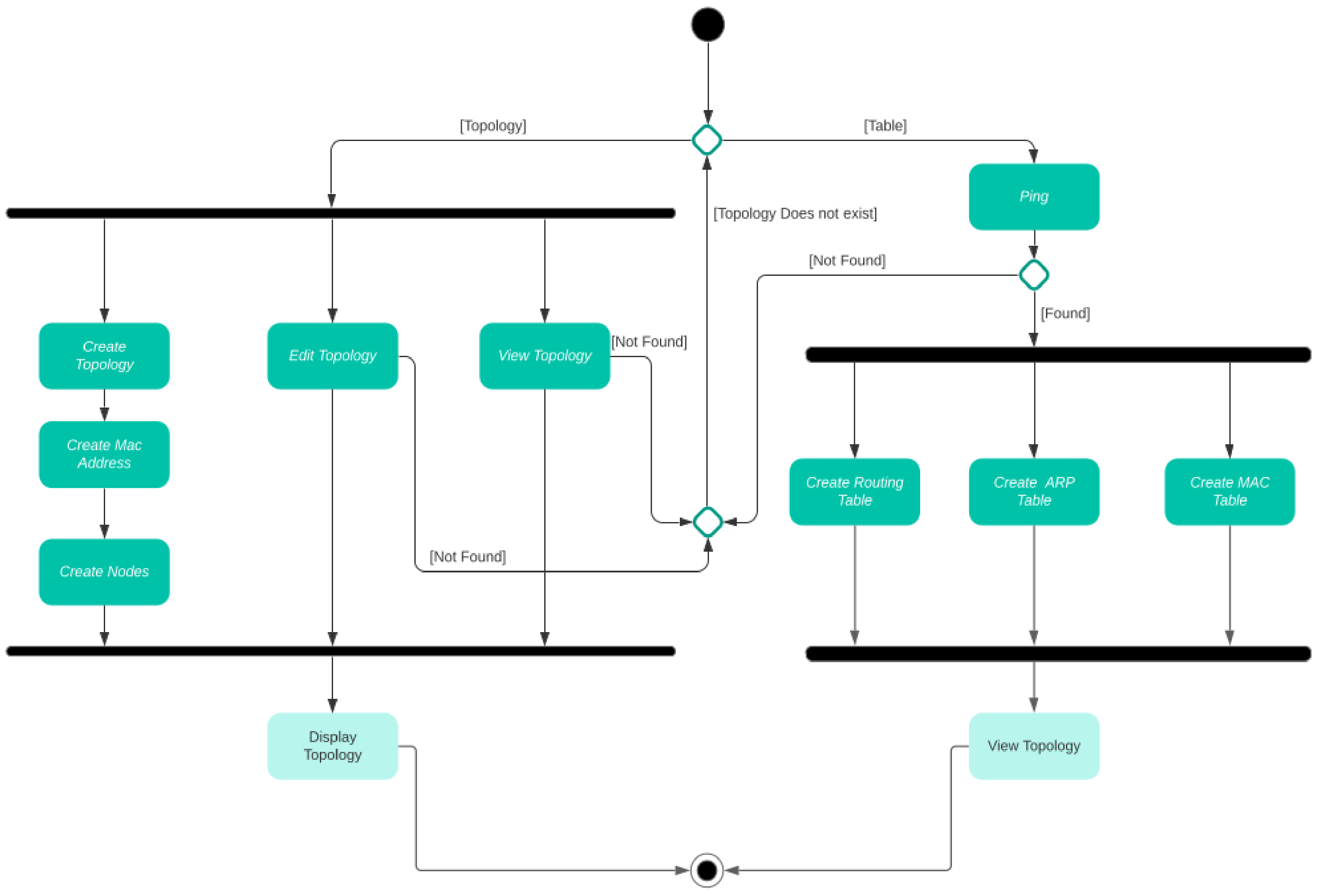
- **Portability:** the ease with which a software system can be adapted.

Appendix A: Analysis Model

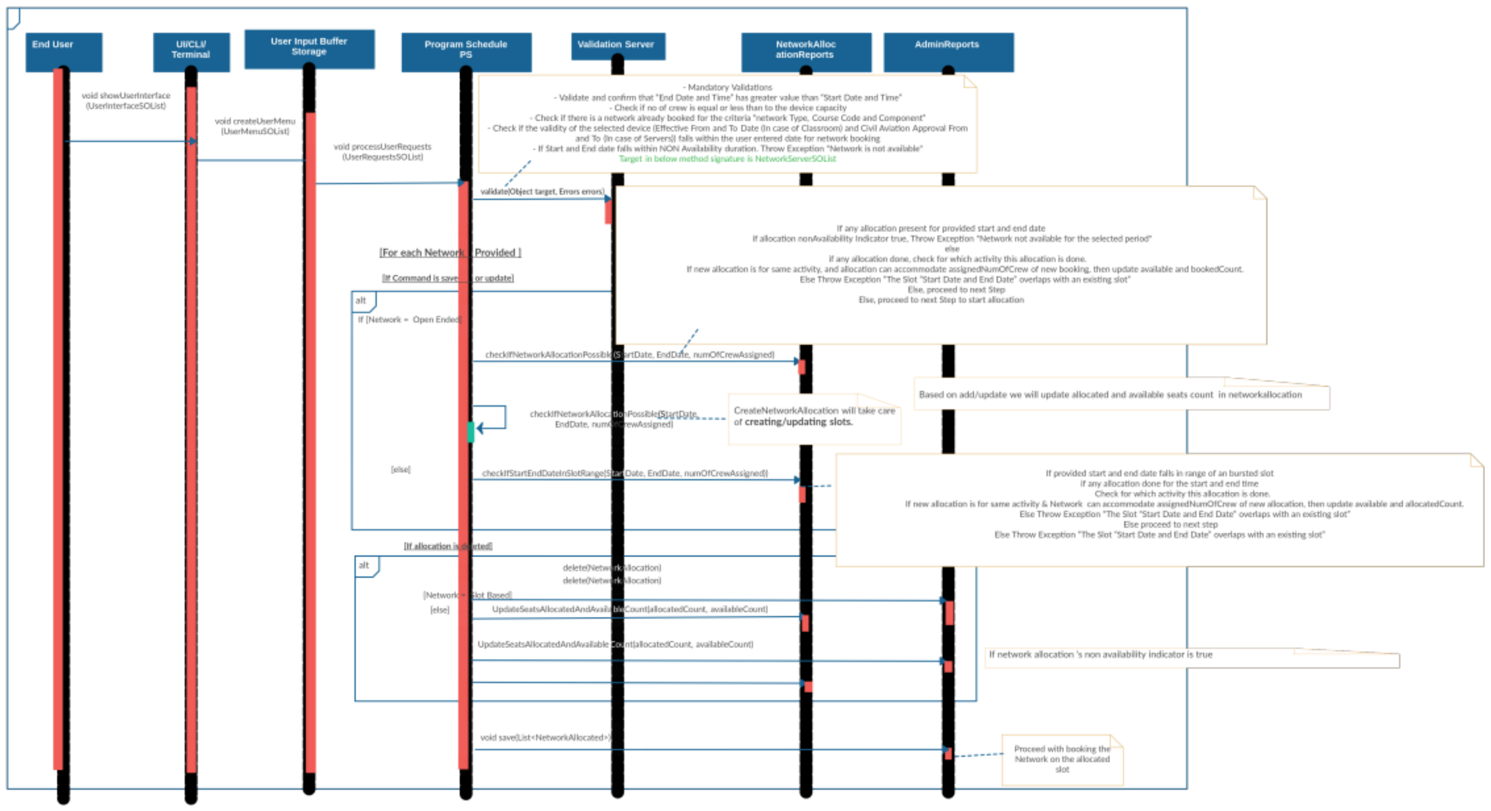
USE CASE DIAGRAM



ACTIVITY DIAGRAM

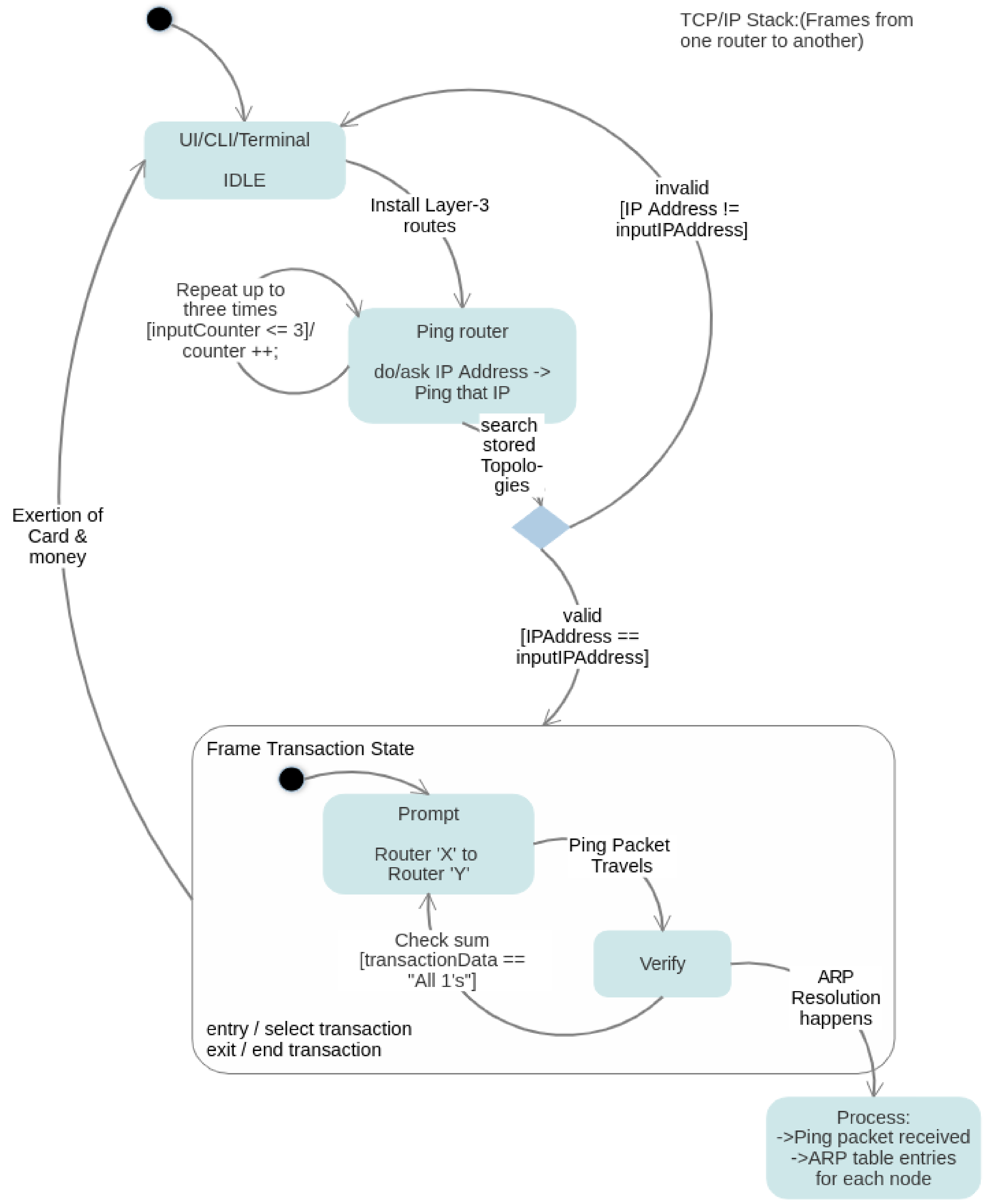


SEQUENCE DIAGRAM

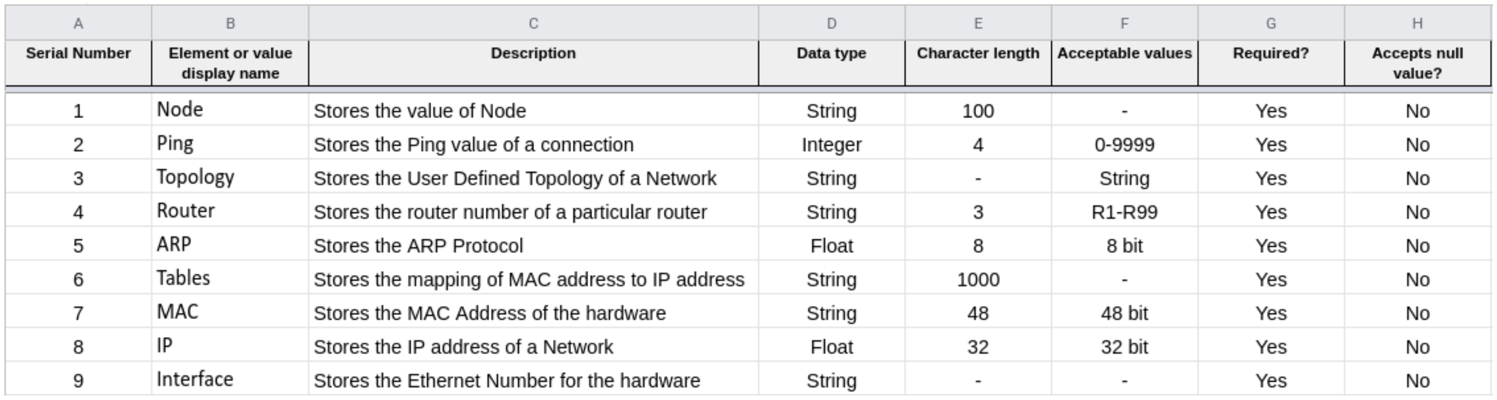

End User
UI/CLI/ Terminal
User Input Buffer Storage
Program Schedule PS
Validation Server
NetworkAllocationReports
AdminReports
void showUserInterface (UserInterfaceSOList)
void createUserMenu (UserMenuSOList)
void processUserRequests (UserRequestsSOList)
- Mandatory Validations
- Validate and confirm that "End Date and Time" has greater value than "Start Date and Time"
- Check if no of crew is equal or less than to the device capacity
- Check if there is a network already booked for the criteria "network Type, Course Code and Component"
- Check if the validity of the selected device (Effective From and To Date (In case of Classroom) and Civil Aviation Approval From and To (In case of Servers)) falls within the user entered date for network booking
- If Start and End date falls within NON Availability duration. Throw Exception "Network is not available"
Target in below method signature is NetworkServerSOList
validate(Object target, Errors errors)
If any allocation present for provided start and end date
If allocation nonAvailability Indicator true, Throw Exception "Network not available for the selected period"
else
if any allocation done, check for which activity this allocation is done.
If new allocation is for same activity, and allocation can accommodate assignedNumOfCrew of new booking, then update available and bookedCount.
Else Throw Exception "The Slot "Start Date and End Date" overlaps with an existing slot"
Else, proceed to next Step
Else, proceed to next Step to start allocation
[For each Network Provided]
[If Command is save or update]
alt
If [Network = Open Ended]
checkIfNetworkAllocationPossible(StartDate, EndDate, numOfCrewAssigned)
Based on add/update we will update allocated and available seats count in networkallocation
checkIfNetworkAllocationPossible(StartDate, EndDate, numOfCrewAssigned)
CreateNetworkAllocation will take care of creating/updating slots.
[else]
checkIfStartEndDateInSlotRange(StartDate, EndDate, numOfCrewAssigned)
If provided start and end date falls in range of an bursted slot
If any allocation done for the start and end time
Check for which activity this allocation is done.
If new allocation is for same activity & Network can accommodate assignedNumOfCrew of new allocation, then update available and allocatedCount.
Else Throw Exception "The Slot "Start Date and End Date" overlaps with an existing slot"
Else proceed to next step
Else Throw Exception "The Slot "Start Date and End Date" overlaps with an existing slot"
[If allocation is deleted]
alt
delete(NetworkAllocation)
delete(NetworkAllocation)
[Network = Slot Based]
[else]
UpdateSeatsAllocatedAndAvailableCount(allocatedCount, availableCount)
UpdateSeatsAllocatedAndAvailableCount(allocatedCount, availableCount)
If network allocation 's non availability indicator is true
void save(List<NetworkAllocated>)
Proceed with booking the Network on the allocated slot

Full Size Version of Sequence Diagram

STATE DIAGRAM

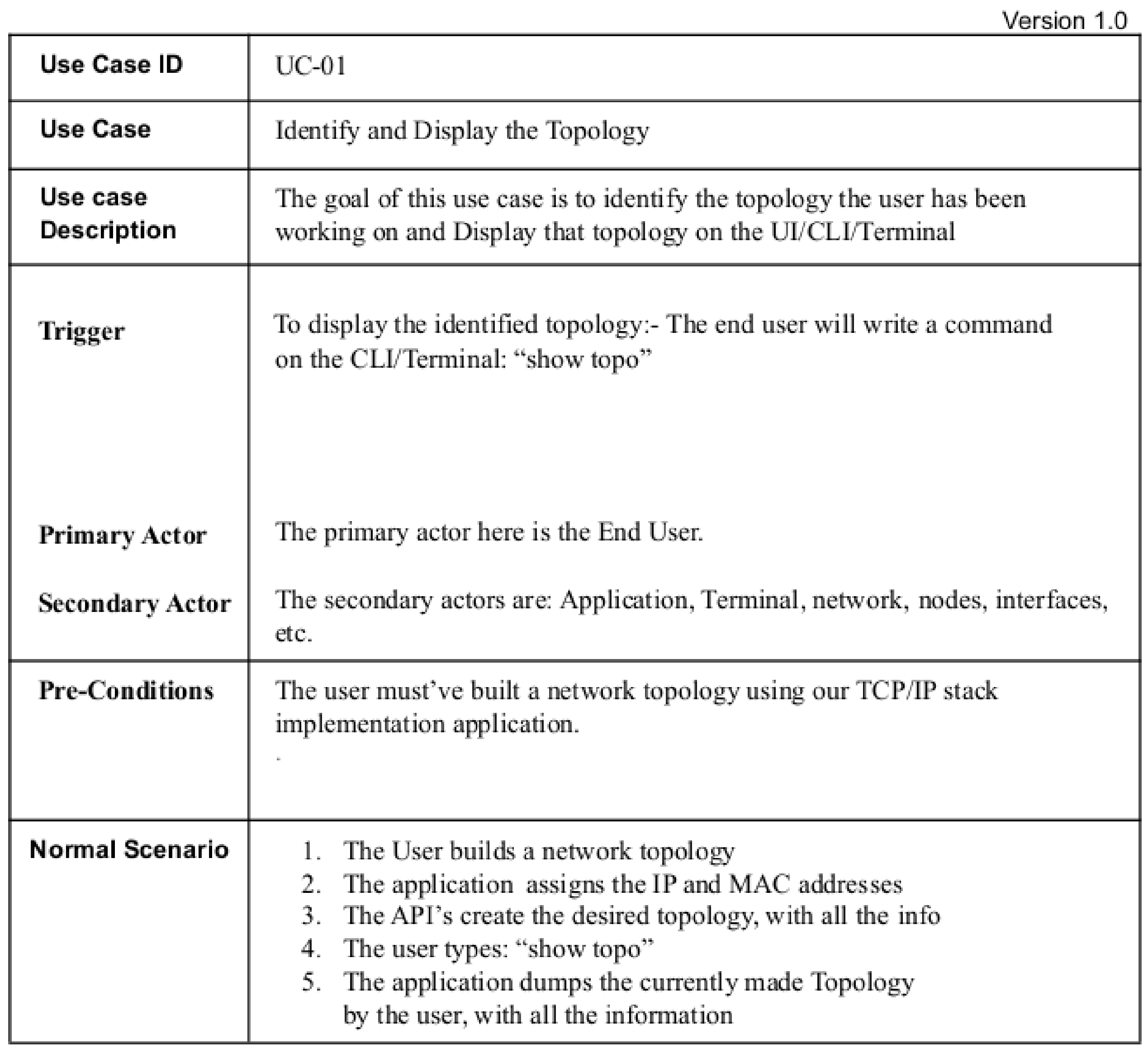

TCP/IP Stack:(Frames from one router to another)
UI/CLI/Terminal
IDLE
Install Layer-3 routes
invalid [IP Address != inputIPAddress]
Repeat up to three times [inputCounter <= 3]/ counter ++;
Ping router
do/ask IP Address -> Ping that IP
search stored Topolo- gies
Exertion of Card & money
valid [IPAddress == inputIPAddress]
Frame Transaction State
Prompt
Router 'X' to Router 'Y'
Ping Packet Travels
Check sum [transactionData == "All 1's"]
Verify
ARP Resolution happens
entry / select transaction
exit / end transaction
Process: ->Ping packet received ->ARP table entries for each node

DATA DICTIONARY



A	B	C	D	E	F	G	H
Serial Number	**Element or value display name**	**Description**	**Data type**	**Character length**	**Acceptable values**	**Required?**	**Accepts null value?**
1	Node	Stores the value of Node	String	100	-	Yes	No
2	Ping	Stores the Ping value of a connection	Integer	4	0-9999	Yes	No
3	Topology	Stores the User Defined Topology of a Network	String	-	String	Yes	No
4	Router	Stores the router number of a particular router	String	3	R1-R99	Yes	No
5	ARP	Stores the ARP Protocol	Float	8	8 bit	Yes	No
6	Tables	Stores the mapping of MAC address to IP address	String	1000	-	Yes	No
7	MAC	Stores the MAC Address of the hardware	String	48	48 bit	Yes	No
8	IP	Stores the IP address of a Network	Float	32	32 bit	Yes	No
9	Interface	Stores the Ethernet Number for the hardware	String	-	-	Yes	No

USE CASE TEMPLATE 1



Version 1.0

Use Case ID	UC-01
Use Case	Identify and Display the Topology
Use case Description	The goal of this use case is to identify the topology the user has been working on and Display that topology on the UI/CLI/Terminal
Trigger	To display the identified topology:- The end user will write a command on the CLI/Terminal: “show topo”
Primary Actor	The primary actor here is the End User.
Secondary Actor	The secondary actors are: Application, Terminal, network, nodes, interfaces, etc.
Pre-Conditions	The user must’ve built a network topology using our TCP/IP stack implementation application. .
Normal Scenario	1. The User builds a network topology 2. The application assigns the IP and MAC addresses 3. The API’s create the desired topology, with all the info 4. The user types: “show topo” 5. The application dumps the currently made Topology by the user, with all the information

Extension points **Alternate Flow**	5a. In step 5, if there has been no topology built by the user: 1. Then the system will prompt an error of "No Topology exists" 2. The user returns to the basic step and builds a topology from scratch 3. Use case resumes on step 5 again 4a. At step 4, if the user wants to edit the topology he/she built: 1. They're sent to step 1 and let them make changes 2. Then all the steps are followed from the starting 3. All the changes are incorporated 4. New edited Topology is displayed 1a. The user can exit the application.
Post Conditions	<u>Success end condition</u> The build Topology is identified and displayed. <u>Failure end condition:</u> End-user can't build the desired topology, so no topology is displayed And the further steps couldn't follow.

Special Requirements

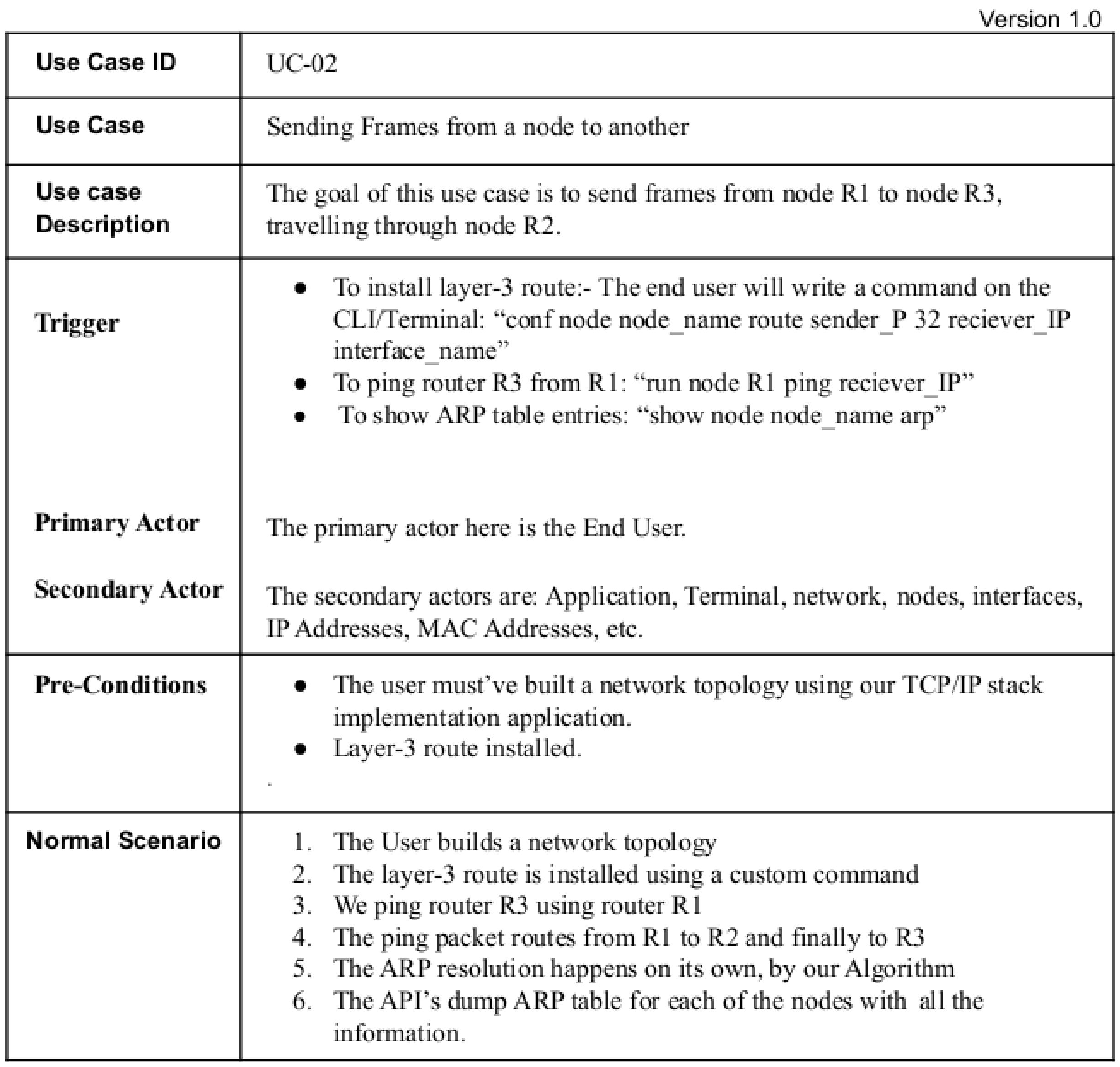
1. The supported Operating system to run the application is UNIX/LINUX-based systems.

2. The System should have GTK-3 installed

3. To understand the making of topology, the user should know Graph Modelling

4. The user must know the explicit commands of our exclusive CLI to operate the application.

USE CASE TEMPLATE 2



Version 1.0

Use Case ID	UC-02
Use Case	Sending Frames from a node to another
Use case Description	The goal of this use case is to send frames from node R1 to node R3, travelling through node R2.
Trigger	● To install layer-3 route:- The end user will write a command on the CLI/Terminal: "conf node node_name route sender_P 32 reciever_IP interface_name" ● To ping router R3 from R1: "run node R1 ping reciever_IP" ● To show ARP table entries: "show node node_name arp"
Primary Actor	The primary actor here is the End User.
Secondary Actor	The secondary actors are: Application, Terminal, network, nodes, interfaces, IP Addresses, MAC Addresses, etc.
Pre-Conditions	● The user must've built a network topology using our TCP/IP stack implementation application. ● Layer-3 route installed. .
Normal Scenario	1. The User builds a network topology 2. The layer-3 route is installed using a custom command 3. We ping router R3 using router R1 4. The ping packet routes from R1 to R2 and finally to R3 5. The ARP resolution happens on its own, by our Algorithm 6. The API's dump ARP table for each of the nodes with all the information.

Extension points	5a. In step 5, if there has been no topology built by the user: 1. Then the system will prompt an error of “No such node exists” 2. The user returns to the basic step and builds a topology from scratch 3. Use case resumes on step 5 again
Alternate Flow	4a. At step 4, if the user wants to edit the topology he/she built: 1. They’re sent to step 1 and let them make changes 2. Then all the steps are followed from the starting 3. All the changes are incorporated 4. New edited Topology is displayed 1a. The user can exit the application.
Post Conditions	<u>Success end condition</u> The build Topology is identified and displayed, the ping request passes And ARP table for each node is dumped. <u>Failure end condition:</u> End-user can’t build the desired topology, so no topology or ARP table is displayed And the further steps couldn’t follow.

Special Requirements

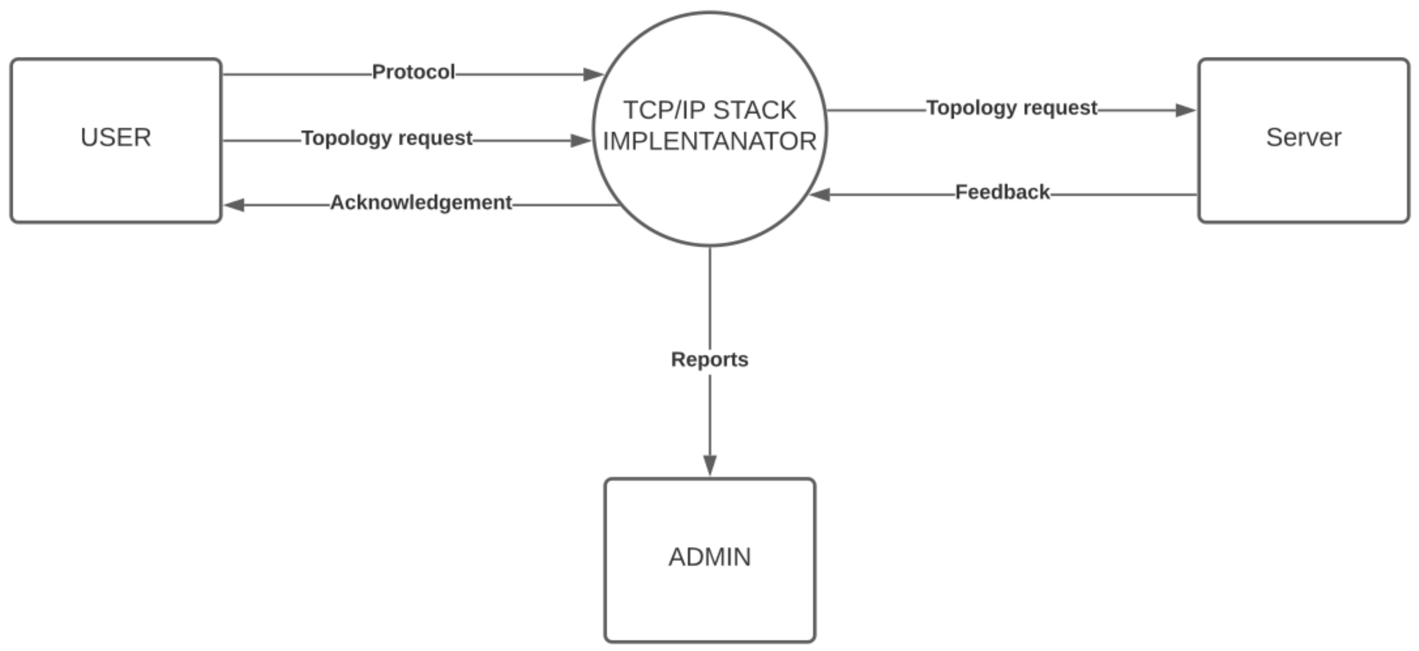
1. The supported Operating system to run the application is UNIX/LINUX-based systems.

2. The System should have GTK-3 installed

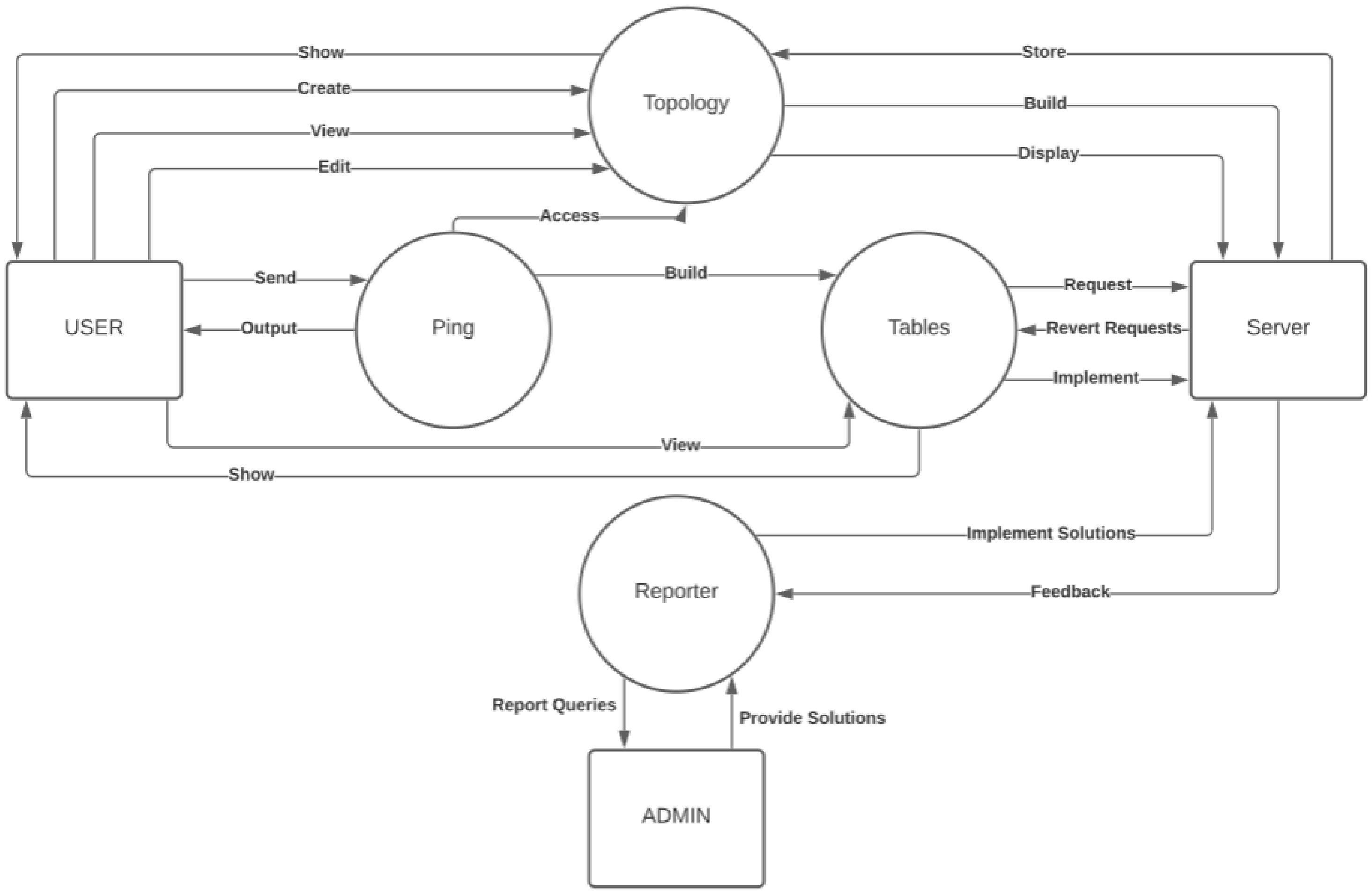
3. To understand the making of topology, the user should know Graph Modelling

4. The user must know the explicit commands of our exclusive CLI to operate the application.

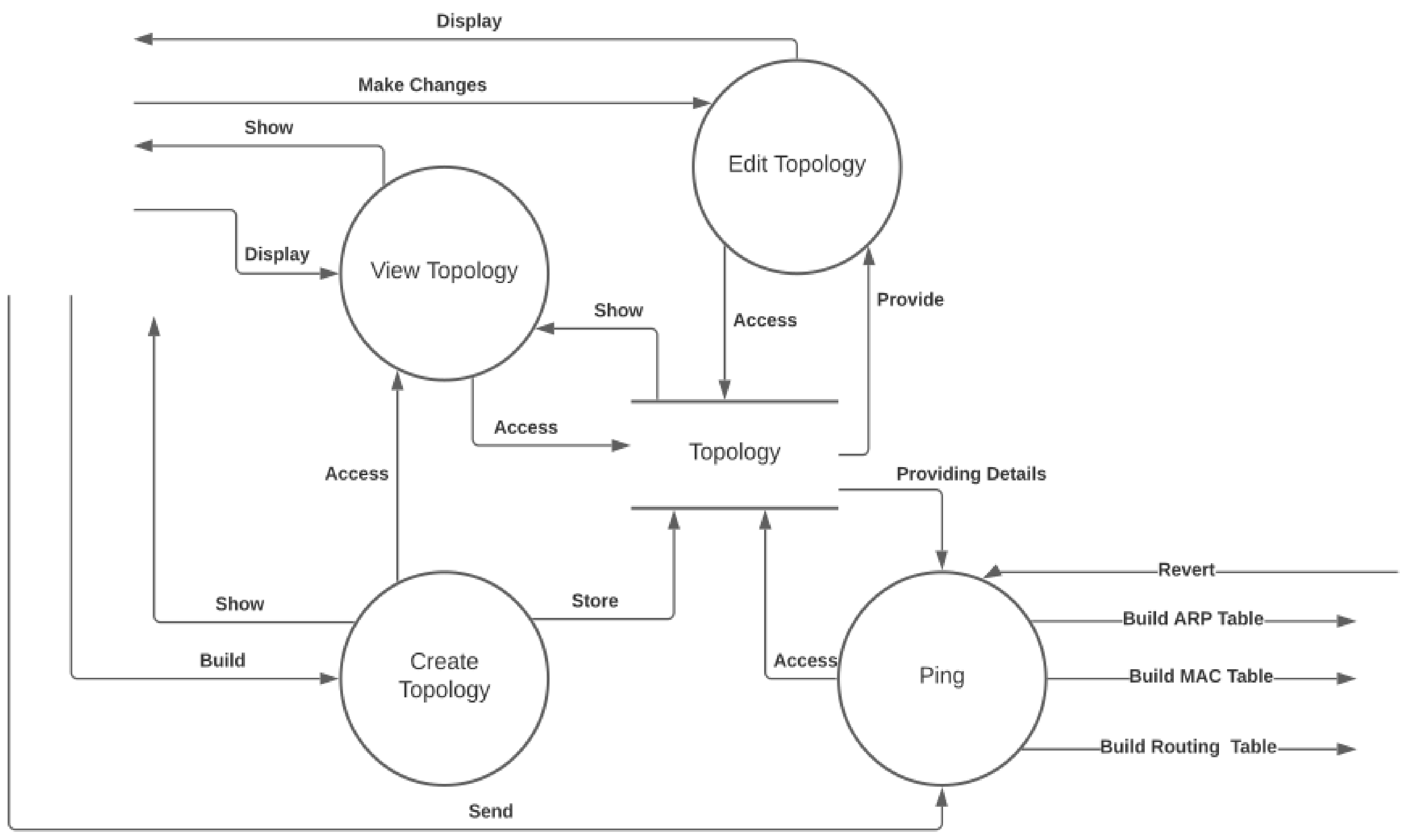
DFD Level-0


USER
Protocol
Topology request
Acknowledgement
TCP/IP STACK IMPLENTANATOR
Topology request
Feedback
Server
Reports
ADMIN

DFD Level-1


Show
Create
View
Edit
Topology
Store
Build
Display
Access
USER
Send
Output
Ping
Build
Tables
Request
Revert Requests
Implement
Server
View
Show
Implement Solutions
Reporter
Feedback
Report Queries
Provide Solutions
ADMIN

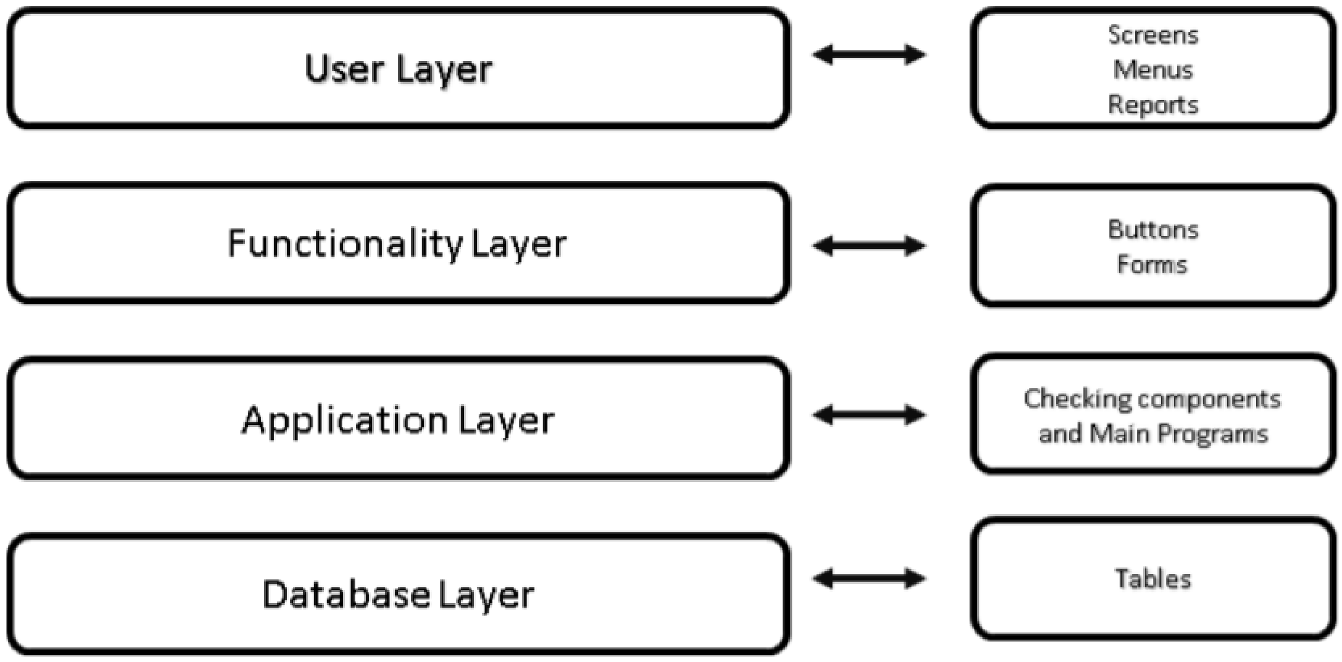
DFD Level-2


Display
Make Changes
Show
Edit Topology
Display
View Topology
Show
Access
Provide
Access
Topology
Access
Providing Details
Revert
Show
Store
Build ARP Table
Build
Create Topology
Access
Ping
Build MAC Table
Build Routing Table
Send

ARCHITECTURE STYLE

TCP/IP Implementation in C follows the Layered Architecture Style since there are a number of different layers that are defined with each layer performing a well-defined set of operations. Each layer will do some operations that become closer to the machine instruction set progressively.

The various layers in the program is as follows -



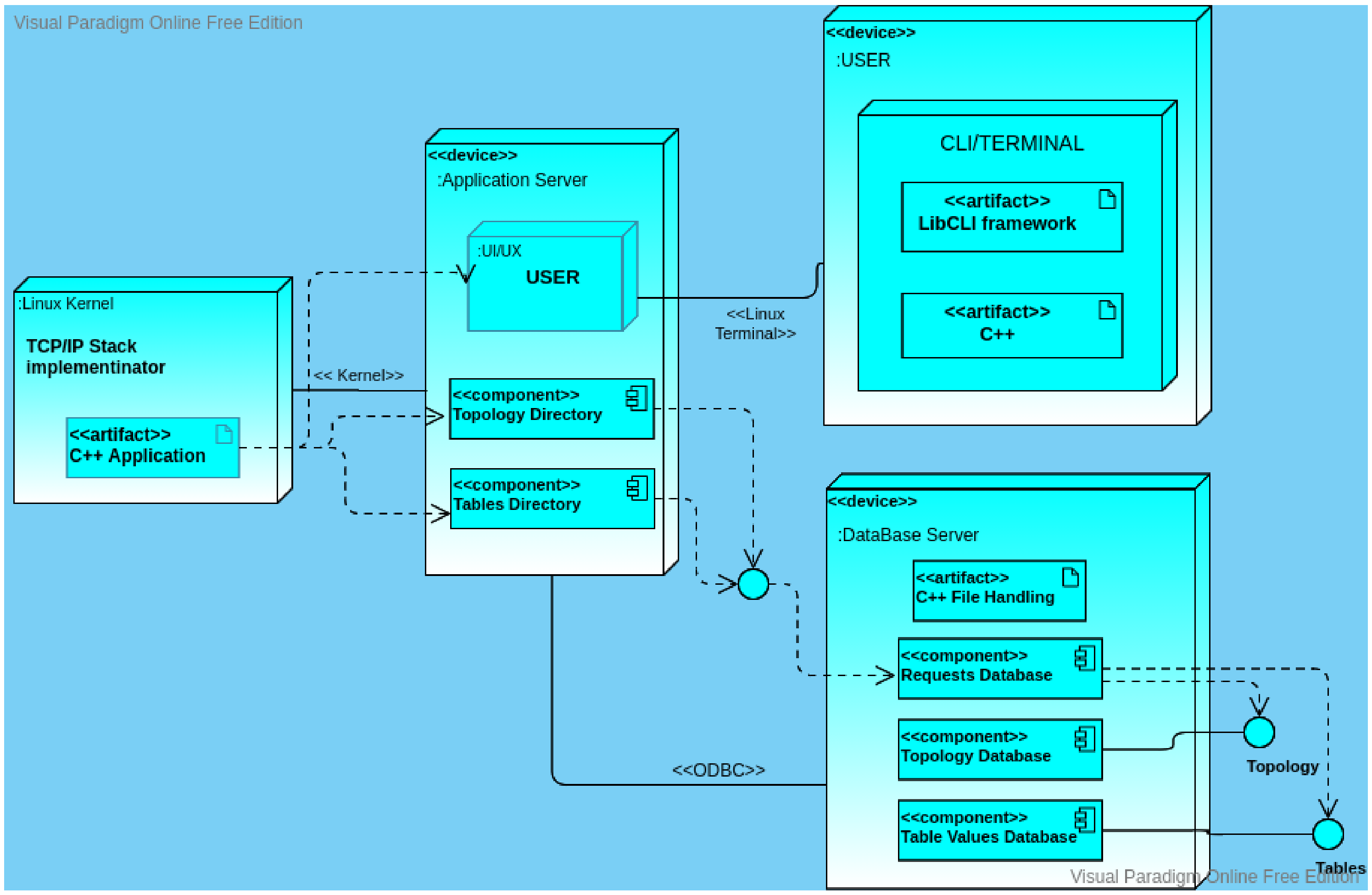
- **User interaction layer**: This is the layer that interacts with users through screens, forms, menus, reports, etc. It is the most visible layer of the application. It defines how the application looks

- **Functionality layer:** This is the layer that presents the functions, methods, and procedures of the system based on the business rules layer. It determines how the pull-down menus work, how the buttons work, and how the system navigates through screens.

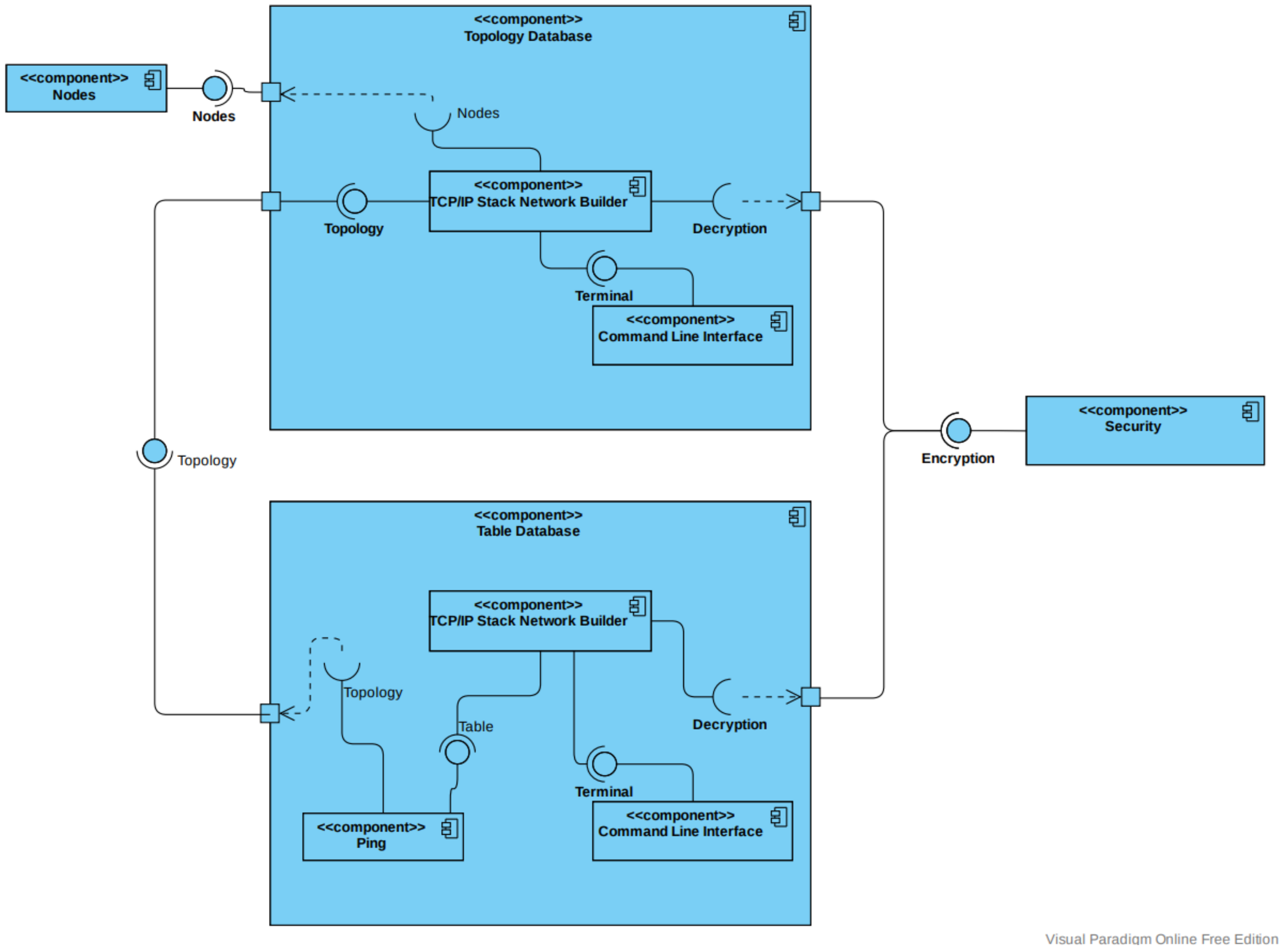
- **Application core layer:** This server contains the main programs, code definitions, and basic functions of the application. Programmers work in this layer most of the time

- **Database layer:** This server contains the main programs, code definitions, and basic functions of the application. Programmers work in this layer most of the time

DEPLOYMENT DIAGRAM

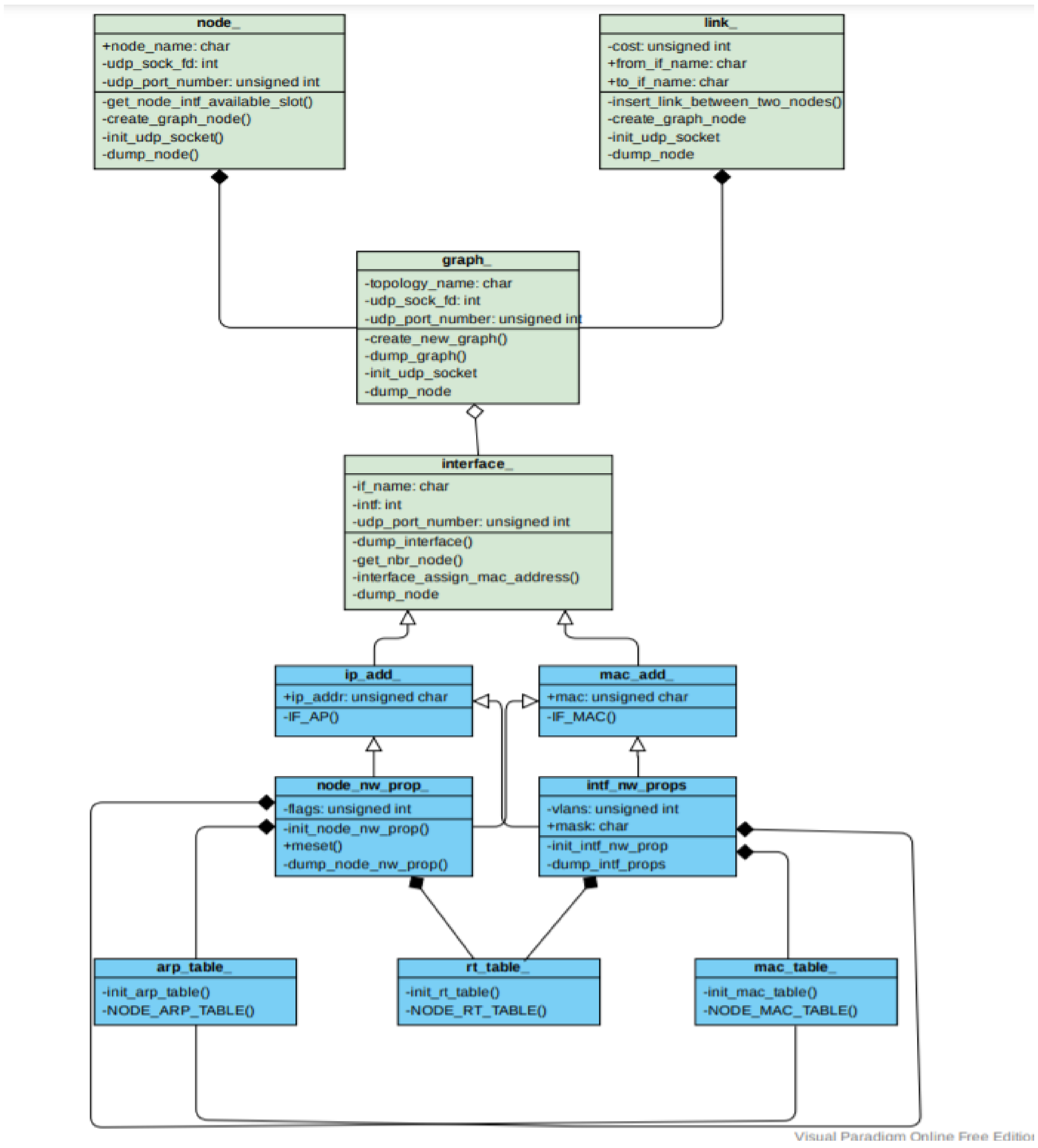


COMPONENT DIAGRAM



Full Size Version of Component Diagram

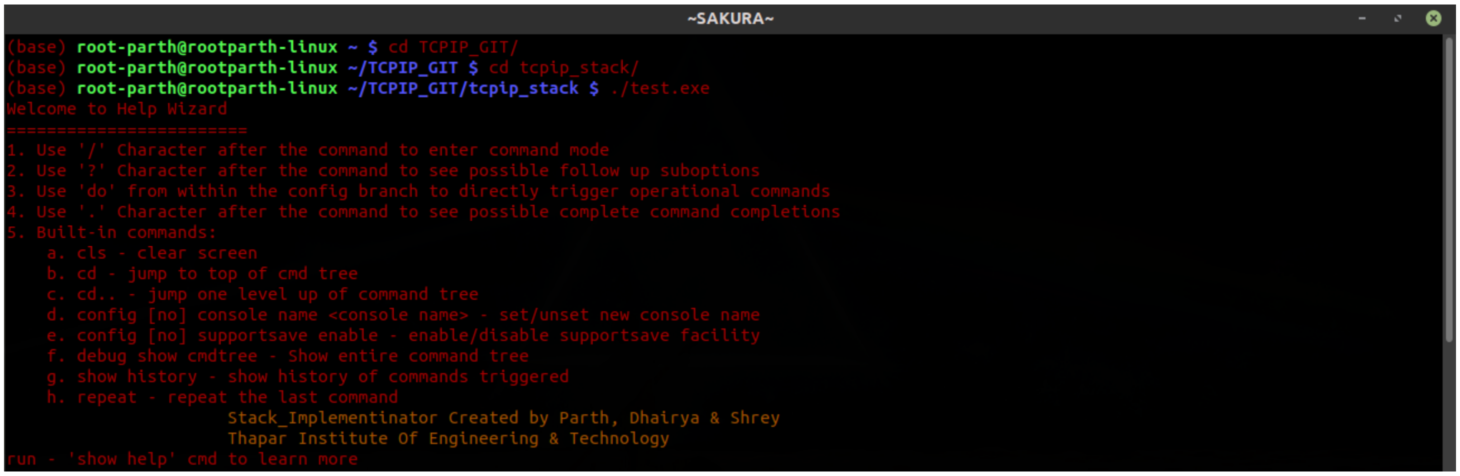
CLASS DIAGRAM



[Full Size Version of Class Diagram]

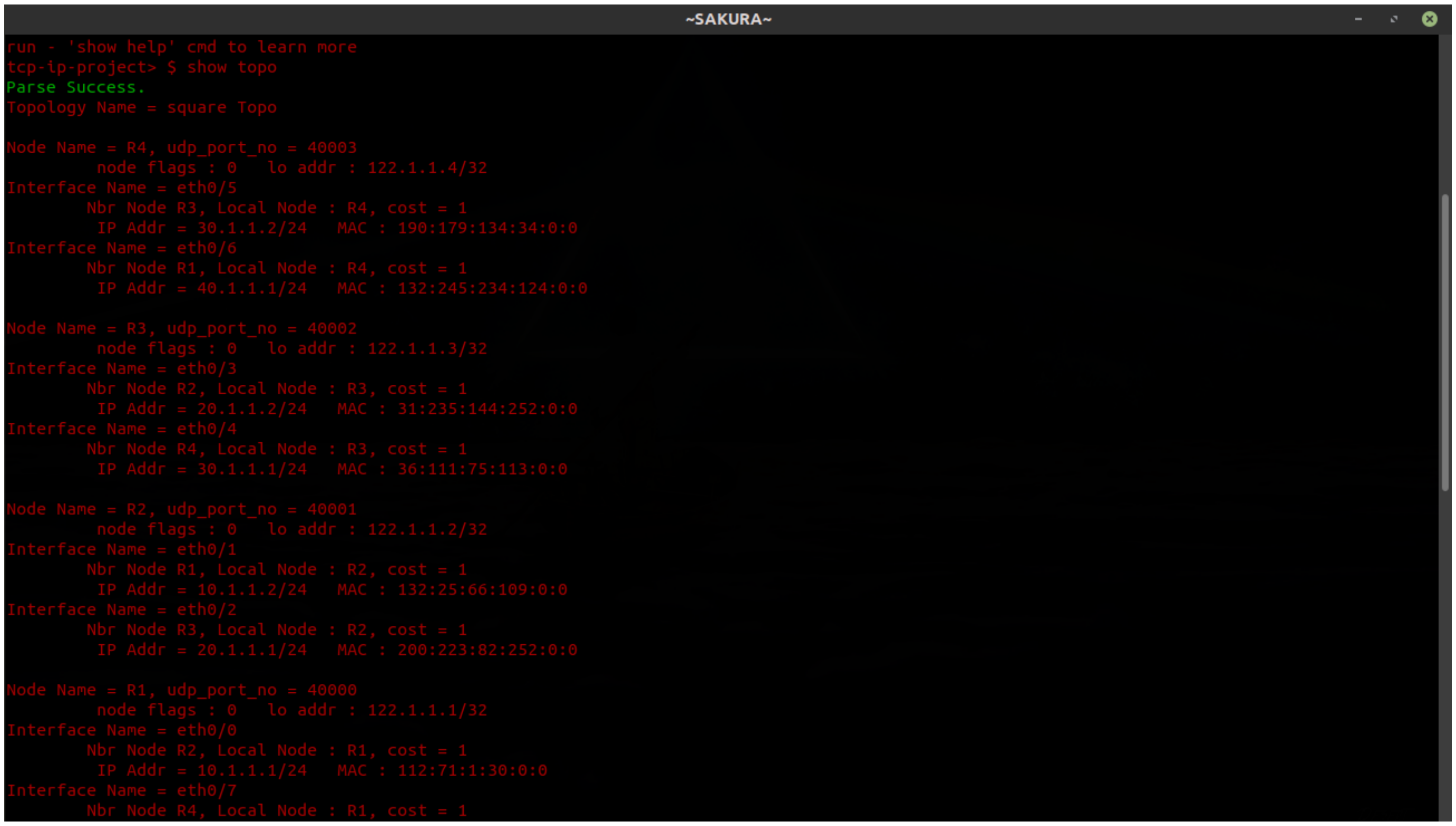
UID SNAPSHOTS

1. Welcome Pane/ Help Wizard



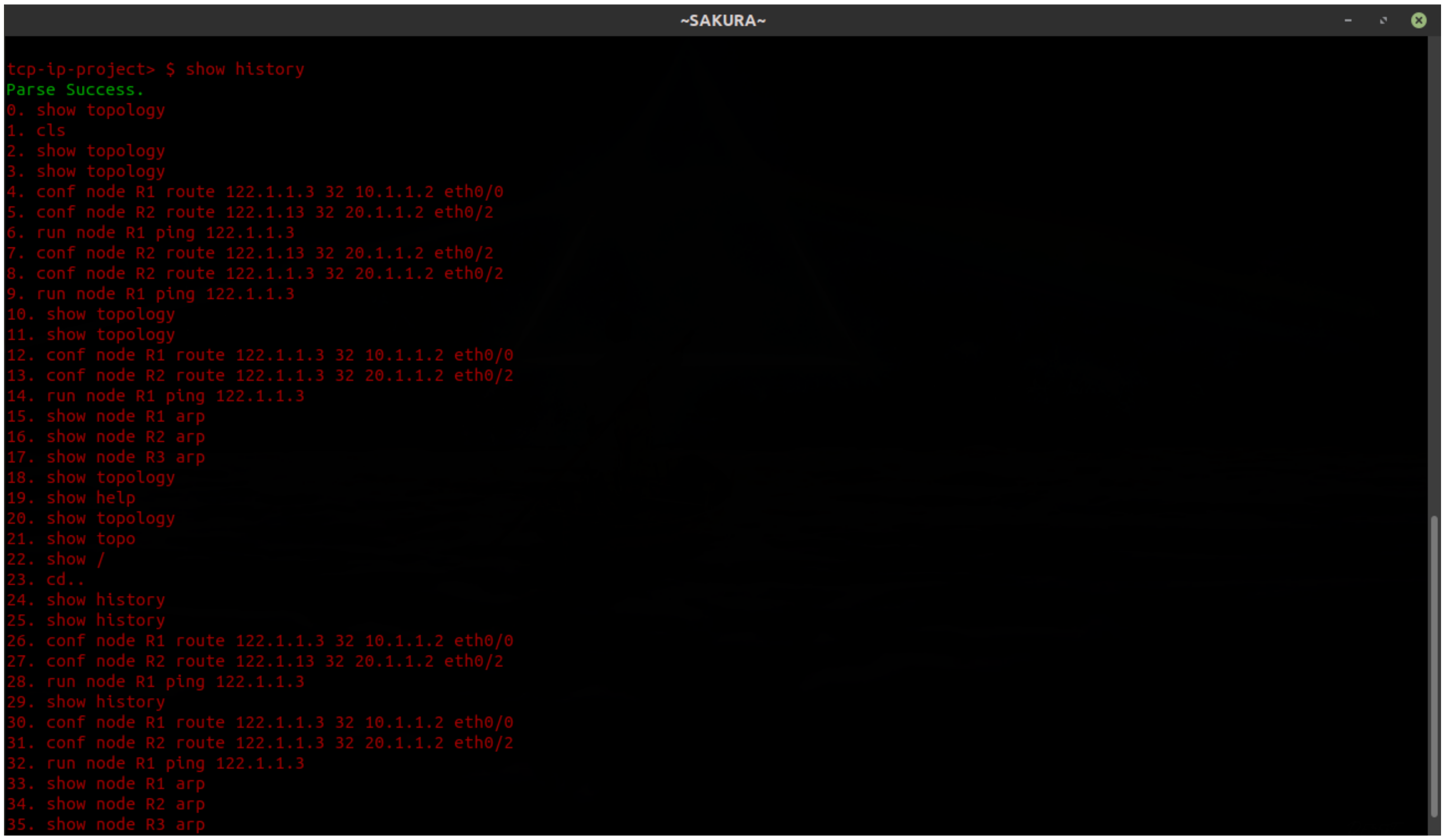
```
(base) root-parth@rootparth-linux ~ $ cd TCPIP_GIT/
(base) root-parth@rootparth-linux ~/TCPIP_GIT $ cd tcpip_stack/
(base) root-parth@rootparth-linux ~/TCPIP_GIT/tcpip_stack $ ./test.exe
Welcome to Help Wizard
========================
1. Use '/' Character after the command to enter command mode
2. Use '?' Character after the command to see possible follow up suboptions
3. Use 'do' from within the config branch to directly trigger operational commands
4. Use '.' Character after the command to see possible complete command completions
5. Built-in commands:
    a. cls - clear screen
    b. cd - jump to top of cmd tree
    c. cd.. - jump one level up of command tree
    d. config [no] console name <console name> - set/unset new console name
    e. config [no] supportsave enable - enable/disable supportsave facility
    f. debug show cmdtree - Show entire command tree
    g. show history - show history of commands triggered
    h. repeat - repeat the last command
                      Stack_Implementinator Created by Parth, Dhairya & Shrey
                      Thapar Institute Of Engineering & Technology
run - 'show help' cmd to learn more
```

2. Dumping Topology



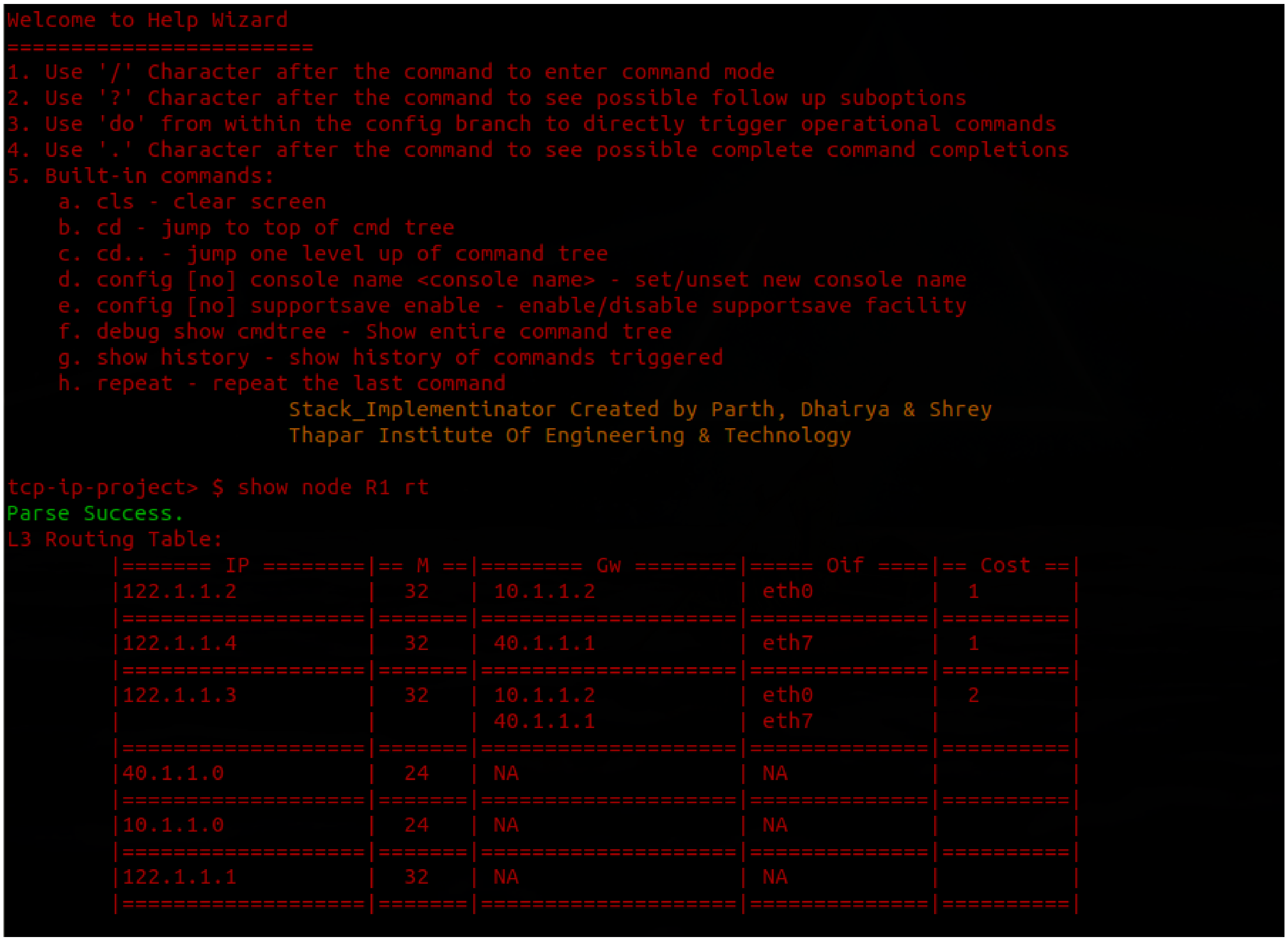
```
run - 'show help' cmd to learn more
tcp-ip-project> $ show topo
Parse Success.
Topology Name = square Topo

Node Name = R4, udp_port_no = 40003
         node flags : 0   lo addr : 122.1.1.4/32
Interface Name = eth0/5
        Nbr Node R3, Local Node : R4, cost = 1
         IP Addr = 30.1.1.2/24   MAC : 190:179:134:34:0:0
Interface Name = eth0/6
        Nbr Node R1, Local Node : R4, cost = 1
         IP Addr = 40.1.1.1/24   MAC : 132:245:234:124:0:0

Node Name = R3, udp_port_no = 40002
         node flags : 0   lo addr : 122.1.1.3/32
Interface Name = eth0/3
        Nbr Node R2, Local Node : R3, cost = 1
         IP Addr = 20.1.1.2/24   MAC : 31:235:144:252:0:0
Interface Name = eth0/4
        Nbr Node R4, Local Node : R3, cost = 1
         IP Addr = 30.1.1.1/24   MAC : 36:111:75:113:0:0

Node Name = R2, udp_port_no = 40001
         node flags : 0   lo addr : 122.1.1.2/32
Interface Name = eth0/1
        Nbr Node R1, Local Node : R2, cost = 1
         IP Addr = 10.1.1.2/24   MAC : 132:25:66:109:0:0
Interface Name = eth0/2
        Nbr Node R3, Local Node : R2, cost = 1
         IP Addr = 20.1.1.1/24   MAC : 200:223:82:252:0:0

Node Name = R1, udp_port_no = 40000
         node flags : 0   lo addr : 122.1.1.1/32
Interface Name = eth0/0
        Nbr Node R2, Local Node : R1, cost = 1
         IP Addr = 10.1.1.1/24   MAC : 112:71:1:30:0:0
Interface Name = eth0/7
        Nbr Node R4, Local Node : R1, cost = 1
```

3. History Of Commands



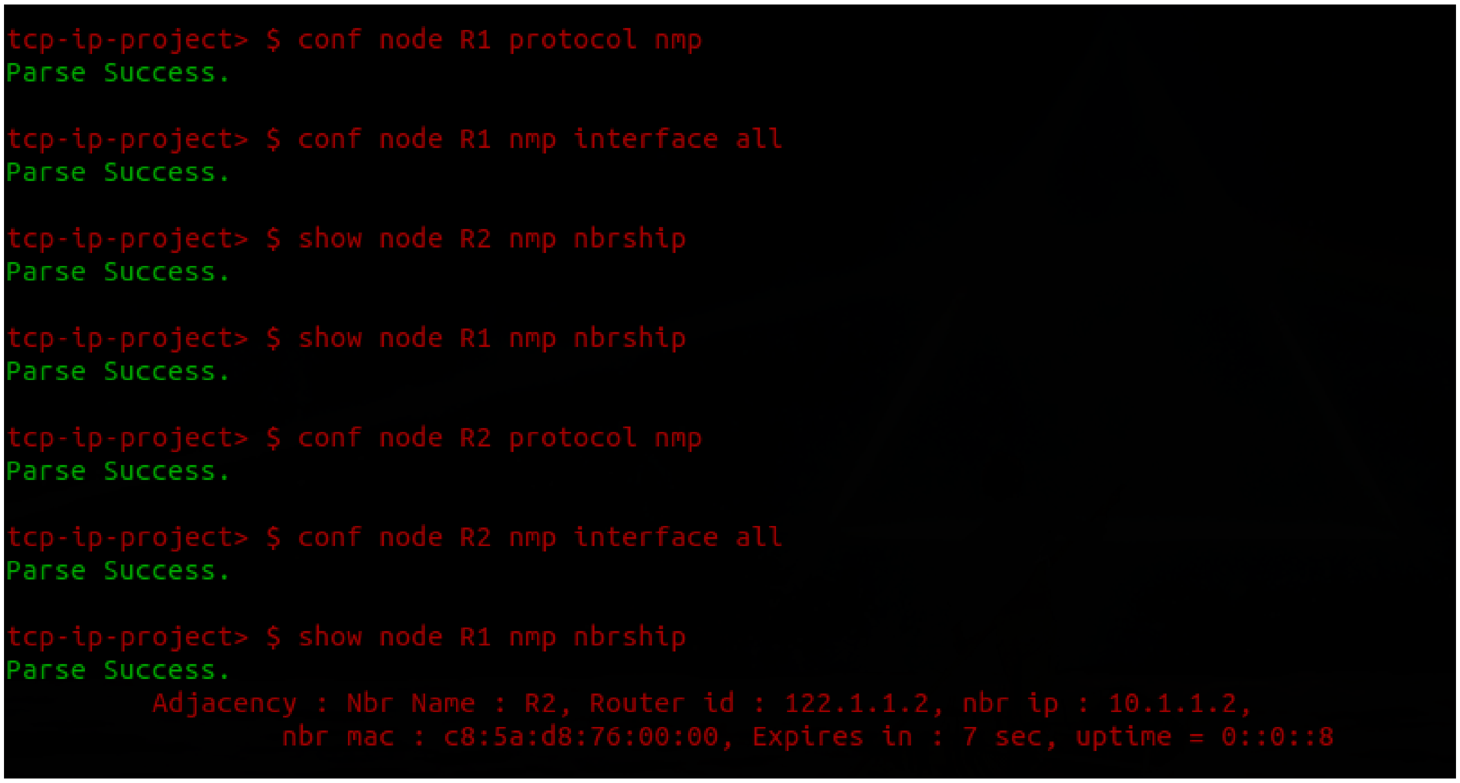
```
tcp-ip-project> $ show history
Parse Success.
0. show topology
1. cls
2. show topology
3. show topology
4. conf node R1 route 122.1.1.3 32 10.1.1.2 eth0/0
5. conf node R2 route 122.1.13 32 20.1.1.2 eth0/2
6. run node R1 ping 122.1.1.3
7. conf node R2 route 122.1.13 32 20.1.1.2 eth0/2
8. conf node R2 route 122.1.1.3 32 20.1.1.2 eth0/2
9. run node R1 ping 122.1.1.3
10. show topology
11. show topology
12. conf node R1 route 122.1.1.3 32 10.1.1.2 eth0/0
13. conf node R2 route 122.1.1.3 32 20.1.1.2 eth0/2
14. run node R1 ping 122.1.1.3
15. show node R1 arp
16. show node R2 arp
17. show node R3 arp
18. show topology
19. show help
20. show topology
21. show topo
22. show /
23. cd..
24. show history
25. show history
26. conf node R1 route 122.1.1.3 32 10.1.1.2 eth0/0
27. conf node R2 route 122.1.13 32 20.1.1.2 eth0/2
28. run node R1 ping 122.1.1.3
29. show history
30. conf node R1 route 122.1.1.3 32 10.1.1.2 eth0/0
31. conf node R2 route 122.1.1.3 32 20.1.1.2 eth0/2
32. run node R1 ping 122.1.1.3
33. show node R1 arp
34. show node R2 arp
35. show node R3 arp
```

4. Dumping Router Table



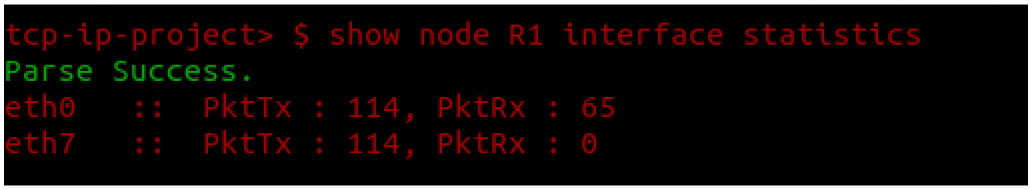
```
Welcome to Help Wizard
========================
1. Use '/' Character after the command to enter command mode
2. Use '?' Character after the command to see possible follow up suboptions
3. Use 'do' from within the config branch to directly trigger operational commands
4. Use '.' Character after the command to see possible complete command completions
5. Built-in commands:
    a. cls - clear screen
    b. cd - jump to top of cmd tree
    c. cd.. - jump one level up of command tree
    d. config [no] console name <console name> - set/unset new console name
    e. config [no] supportsave enable - enable/disable supportsave facility
    f. debug show cmdtree - Show entire command tree
    g. show history - show history of commands triggered
    h. repeat - repeat the last command
                      Stack_Implementinator Created by Parth, Dhairya & Shrey
                      Thapar Institute Of Engineering & Technology

tcp-ip-project> $ show node R1 rt
Parse Success.
L3 Routing Table:
        |======= IP ========|== M ==|======== GW ========|===== Oif ====|== Cost ==|
        |122.1.1.2          |  32   | 10.1.1.2           | eth0         |  1       |
        |===================|=======|====================|==============|==========|
        |122.1.1.4          |  32   | 40.1.1.1           | eth7         |  1       |
        |===================|=======|====================|==============|==========|
        |122.1.1.3          |  32   | 10.1.1.2           | eth0         |  2       |
        |                   |       | 40.1.1.1           | eth7         |          |
        |===================|=======|====================|==============|==========|
        |40.1.1.0           |  24   | NA                 | NA           |          |
        |===================|=======|====================|==============|==========|
        |10.1.1.0           |  24   | NA                 | NA           |          |
        |===================|=======|====================|==============|==========|
        |122.1.1.1          |  32   | NA                 | NA           |          |
        |===================|=======|====================|==============|==========|
```

5. Neighbourship Management Protocol



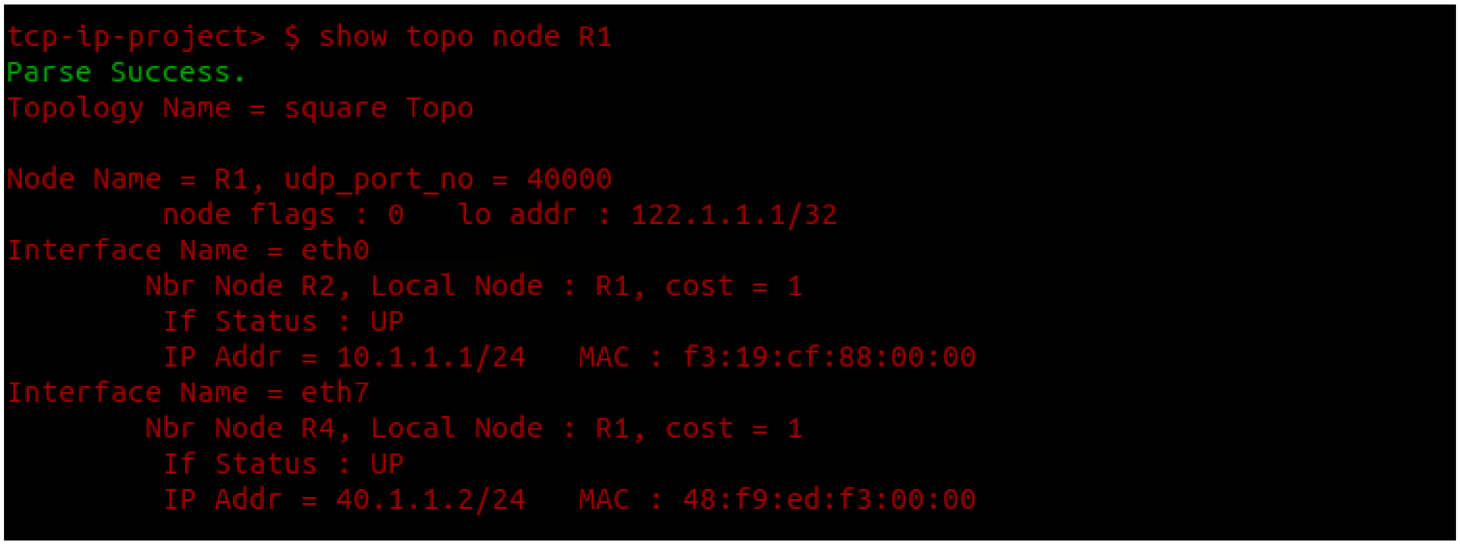
```
tcp-ip-project> $ conf node R1 protocol nmp
Parse Success.

tcp-ip-project> $ conf node R1 nmp interface all
Parse Success.

tcp-ip-project> $ show node R2 nmp nbrship
Parse Success.

tcp-ip-project> $ show node R1 nmp nbrship
Parse Success.

tcp-ip-project> $ conf node R2 protocol nmp
Parse Success.

tcp-ip-project> $ conf node R2 nmp interface all
Parse Success.

tcp-ip-project> $ show node R1 nmp nbrship
Parse Success.
         Adjacency : Nbr Name : R2, Router id : 122.1.1.2, nbr ip : 10.1.1.2,
                 nbr mac : c8:5a:d8:76:00:00, Expires in : 7 sec, uptime = 0::0::8
```

6. Interface Packet Stats



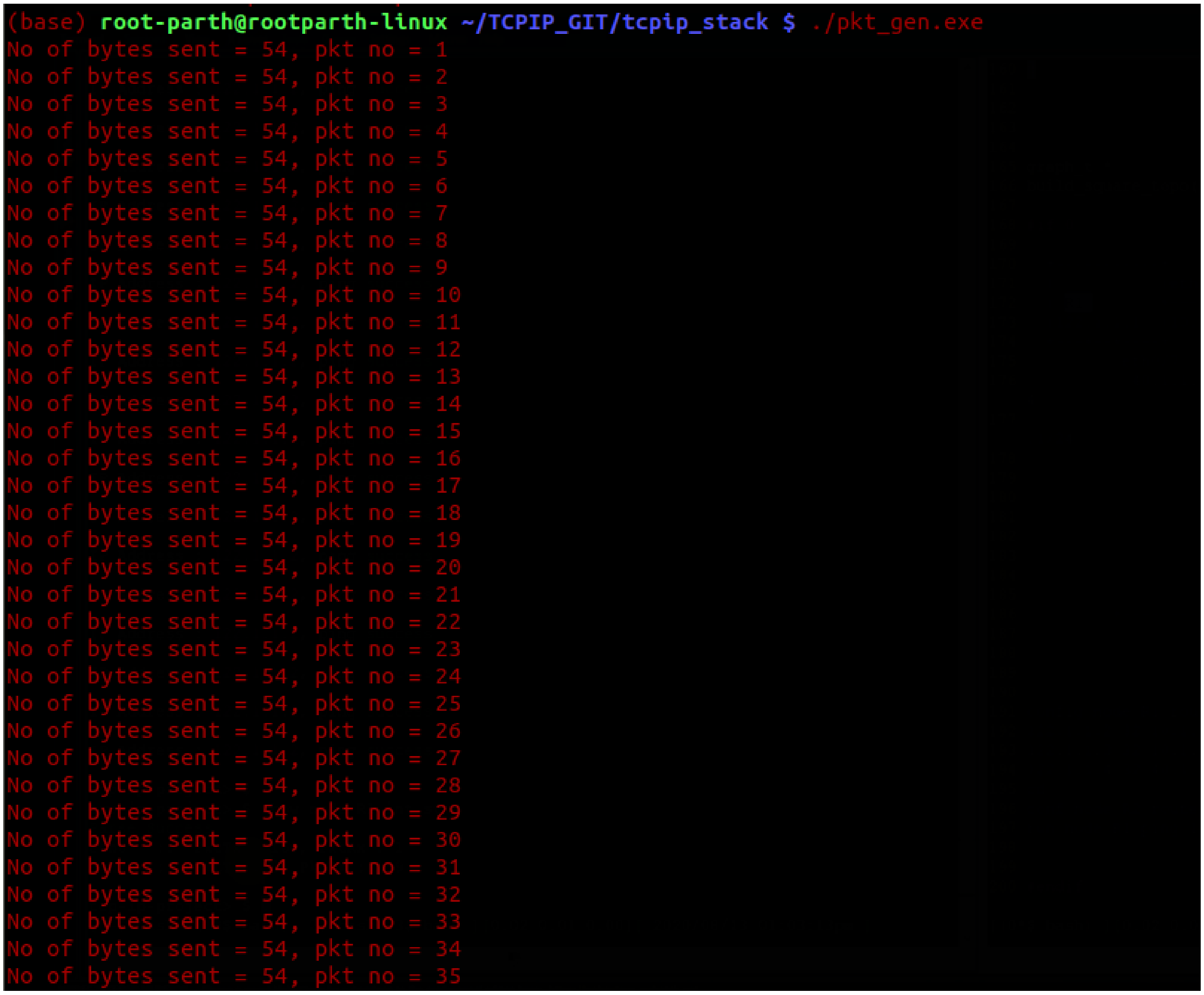
```
tcp-ip-project> $ show node R1 interface statistics
Parse Success.
eth0    ::  PktTx : 114, PktRx : 65
eth7    ::  PktTx : 114, PktRx : 0
```

7. Up/Down Status of nodes



```
tcp-ip-project> $ show topo node R1
Parse Success.
Topology Name = square Topo

Node Name = R1, udp_port_no = 40000
         node flags : 0   lo addr : 122.1.1.1/32
Interface Name = eth0
        Nbr Node R2, Local Node : R1, cost = 1
         If Status : UP
         IP Addr = 10.1.1.1/24   MAC : f3:19:cf:88:00:00
Interface Name = eth7
        Nbr Node R4, Local Node : R1, cost = 1
         If Status : UP
         IP Addr = 40.1.1.2/24   MAC : 48:f9:ed:f3:00:00
```

8. Packet Generator Application



```
(base) root-parth@rootparth-linux ~/TCPIP_GIT/tcpip_stack $ ./pkt_gen.exe
No of bytes sent = 54, pkt no = 1
No of bytes sent = 54, pkt no = 2
No of bytes sent = 54, pkt no = 3
No of bytes sent = 54, pkt no = 4
No of bytes sent = 54, pkt no = 5
No of bytes sent = 54, pkt no = 6
No of bytes sent = 54, pkt no = 7
No of bytes sent = 54, pkt no = 8
No of bytes sent = 54, pkt no = 9
No of bytes sent = 54, pkt no = 10
No of bytes sent = 54, pkt no = 11
No of bytes sent = 54, pkt no = 12
No of bytes sent = 54, pkt no = 13
No of bytes sent = 54, pkt no = 14
No of bytes sent = 54, pkt no = 15
No of bytes sent = 54, pkt no = 16
No of bytes sent = 54, pkt no = 17
No of bytes sent = 54, pkt no = 18
No of bytes sent = 54, pkt no = 19
No of bytes sent = 54, pkt no = 20
No of bytes sent = 54, pkt no = 21
No of bytes sent = 54, pkt no = 22
No of bytes sent = 54, pkt no = 23
No of bytes sent = 54, pkt no = 24
No of bytes sent = 54, pkt no = 25
No of bytes sent = 54, pkt no = 26
No of bytes sent = 54, pkt no = 27
No of bytes sent = 54, pkt no = 28
No of bytes sent = 54, pkt no = 29
No of bytes sent = 54, pkt no = 30
No of bytes sent = 54, pkt no = 31
No of bytes sent = 54, pkt no = 32
No of bytes sent = 54, pkt no = 33
No of bytes sent = 54, pkt no = 34
No of bytes sent = 54, pkt no = 35
```

Appendix B: Glossary

TERM	DEFINITION
GCC	GNU Compiler Collection
RAM	Random Access Memory
UNIX	UNiplexed Information System
SRS	Software Requirements Specification
MAC	Media Access Control