

A Project Report On
Reverse Traceroute for IPv4 and IPv6 networks



**VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELGAUM – 590 014, KARNATAKA**

Submitted in partial fulfillment of the requirement for the 8th semester
Bachelor of Engineering
in
Computer Science & Engineering

Submitted by
Akshayakumar [1DS12CS006]
Amogha [1DS12CS009]
Darshan K [1DS12CS024]
Harish KM [1DS12CS032]

Under the guidance of
Mrs. Prathyusha J
Asst. Professor
Dept. of CSE



**Department of Computer Science & Engineering
Dayananda Sagar College of Engineering**

Shavige Malleshwara Hills, Kumaraswamy Layout
Bangalore - 560078, Karnataka

2015-2016

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
DAYANANDA SAGAR COLLEGE OF ENGINEERING

Shavige Malleshwara Hills, Kumaraswamy Layout, Bangalore - 560078

Department of Computer Science & Engineering



CERTIFICATE

This is to certify that the project work entitled “**Reverse Traceroute for IPv4 and IPv6 networks**” is a bonafide work carried out by **Akshayakumar [1DS12CS006]**, **Amogha [1DS12CS009]**, **Darshan K [1DS12CS024]**, **Harish KM [1DS12CS032]** in a partial fulfillment for the 8th semester of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2015-2016. The project report has been approved as it satisfies the academic requirements in respect of Project Work prescribed for Bachelor of Engineering Degree.

Mrs. Prathyusha J

Asst. Professor
Dept. of CSE, DSCE

Dr. D.R.Ramesh Babu

Vice Principal, HOD
Dept. of CSE,DSCE

Dr. C.P.S. Prakash

Principal
DSCE

Examiners 1

Name:

Examiners 2

Name:

Acknowledgement

A successful project is a fruitful culmination of efforts by many people, some directly involved and some others who have quietly encouraged and extended support from the background.

A project is not complete if one fails to acknowledge all these individuals who have been instrumental in the successful completion of the project.

We would like to begin by expressing our sincere gratitude to our guide, **Mrs. Prathyusha J**, Asst. Professor, Department of Computer Science & Engineering and for her excellent and timely guidance, constant encouragement and strive for perfection that she brought to our project.

We express our sincere gratitude to **Dr. Ramesh Babu D.R**, Vice Principal & HOD of Department of Computer Science & Engineering, DSCE for his relentless support and helped us to make critical decisions during the course of our project.

We would also like to extend our thanks to **Dr. C.P.S. Prakash**, Principal of our college, DSCE for his advice and guidance, that helped us to complete this project with success.

Finally, its a pleasure and happiness to the friendly co-operation showed by all the staff members of Computer Science Department, DSCE.

Akshaykumar [1DS12CS006]

Amogha [1DS12CS009]

Darshan K [1DS12CS024]

Harish KM [1DS12CS032]

Abstract

A Tool to diagnose internet is one of the major necessity. TraceRoute is the tool widely used for this purpose. It is used to identify the failures in the routing, poor performance and misconfigurations in routers. It is used to map the internet, predict the performance and geolocate the routers and the classification of the performance of internet service providers. The analysis of a network path between a source and a destination using traditional networking tools like ping and traceroute, does not provide informations about the return path. There is no general method to determine the reverse path from an arbitrary destination. In the absence of those informations, malfunctions and faults can not be managed with adequate effectiveness.

Therefore we need a tool that provides the reverse path information. In this project we addressed the long standing limitation by building reverse traceroute system. We cannot assume the routing in internet to be symmetric. The path taken by the packets to reach the destination may not be the same path taken by the packets to reach the source. We made use of Socket Programming to determine the Reverse Information. This method can be well used for Networks bearing both IPv4 and IPv6 Protocols. For IPv4 Networks we get the complete Reverse Information since there are networks that support this protocol. And for the networks that implement IPv6 protocol we show the result for the loopback address.

Contents

1	Introduction	1
1.1	Overview	1
1.2	Objective	2
2	Problem and the proposed Technique	3
2.1	Problem statement	3
2.2	Existing system	3
3	Literature survey	5
4	Methodology	10
4.1	Components Required	10
4.2	Client/Server Methodology	10
4.2.1	The goal of client/server architecture development	10
4.2.2	Issues with the Database Design	11
4.3	Client/Server Components	12
4.3.1	Client	12
4.3.2	Server	12
4.3.3	Communications middleware	13
4.4	Thin vs. Fat client	13
4.5	Tiers	13
4.5.1	2-Tier Client / Server System	13
4.5.2	3-Tier Client / Server System	13
4.5.3	Multi-Tier Client / Server System	13
4.6	Client / Server Main Principles	14
5	Implementation	15
5.1	Implementation requirement	15
5.2	File hierarchy	16
5.3	Functions	17
5.3.1	Server functions	17
5.3.2	Client Functions	19
6	Verification and validation	20
6.1	Purpose of Testing	20
6.2	LEVELS OF TESTING	21
6.2.1	Unit Testing	21

6.2.2	Integration Testing	21
6.2.3	System Testing	21
6.2.4	Validation Testing	21
6.3	TYPES OF TESTING	21
6.3.1	Functional Testing	21
6.3.2	Structural Testing	22
6.3.3	Performance Testing	22
6.3.4	Acceptance Testing	23
6.3.5	Regression Testing	23
6.3.6	Alpha Testing	23
6.3.7	Beta Testing	23
7	Results	24
8	Conclusion	27
	Bibliography	28

List of Figures

2.1	Working of traceroute	4
4.1	TCP IP socket diagram	11
5.1	File hierarchy	16
5.2	Project DIR	17
5.3	DFD for listen	18
5.4	DFD for accept_connection	18
5.5	DFD for read_query	18
5.6	DFD for create_query	18
5.7	DFD for establish_connection	19
5.8	DFD for recv_rtr_info	19
7.1	Foreword and reverse traceroute	24
7.2	Binding error	25
7.3	Connection failed error	26

List of Tables

6.1	Unit test results	22
-----	-----------------------------	----

Chapter 1

Introduction

1.1 Overview

Traceroute is a widely used Internet diagnostic tool. It measures the sequence of routers from the source to destination. Operators use it to investigate routing failures and performance problems. Researchers use it as the basis for Internet maps ,path prediction ,geolocation ,ISP performance analysis ,and anomaly detection.

However, traceroute has a fundamental limitation it doesn't provide information taken by the processed packets in reaching back to the source. Our goal is to address this basic restriction of traceroute by building a tool to provide the same basic information as traceroute but the path from destination back to the source. Unlike traceroute, our reverse traceroute tool does require control of the destination. Our tool does not require new functionality from routers or other network components. Our system builds a reverse path incrementally, Using the Client server socket programming. There were three main measurement techniques to build the path backwards.

First, we feel that Internet routing is generally destination-based, allowing us to group together the path as one hop at a time.

Second, we add the IP time-stamp and record route options to the packet to identify the path taken by the processed packets.

Third, we use limited source spoofing, i.e., shifting to another vantage points.

Discussion about the first two measurements. We need to configure the packets the way we want in order to reach our goal, there is a tool called scapy(a python framework) or there can also be some other tools to meet the requirement. But the intermediate routers between the source and the destination drop all the options that we set at the time of configuring the packets. Therefore using the record route option which is one of the most reliable method cannot be implemented We usually rely on routers to be friendly to our techniques, yet most of these intermediate routers are not as friendly as we think. As we ultimately want our tool widely used operationally, we have attempted to pursue our approach in a way that it doesn't require the intermediate routers support. Since the system that we build is not a distributed system, we need not make use of various vantage points to collectively gather the intermediate router information. The system we build is a standalone system.

1.2 Objective

The objective of this project is to build a system which provides us the information that is, the path taken by the packets to reach the source from the destination once they get processed. The major objective of this project is of course to build a reverse traceroute tool which is a destination based system and doesn't take the help of the intermediate routers. The routers in the present days are configured in such a way that they drop all options or flags that are set to the configured packets. This makes it difficult to use the Record Route option. Therefore our goal is to build a system that doesn't depend on the functioning of the intermediate routers.

Chapter 2

Problem and the proposed Technique

2.1 Problem statement

The existing tool has the following drawbacks:

1. From our literature survey we know that there exists routing asymmetry in internet so it is necessary to determine the reverse route of the packet.
2. Record route options can be used to determine reverse route but all intermediate routers do not support processing of these packets with options enabled.
3. All existing systems are distributed systems which require a lot of processing.
4. It doesn't provide the reverse path taken by the packets to reach the source from the destination.
5. The existing system depends on the functioning of the intermediate routers. All the intermediate routers are to be configured in order to get the result.

These drawbacks can be overcome by making use of a client server programming, a part of the Socket Programming, which provides more accurate results and need not depend on the working/functioning of the intermediate hops.

2.2 Existing system

Each time the source sends the information in the form of the packets, for the first packet sent, it sets the TTL value to 1. Once the packet reaches the first intermediate hop/router, it decrements the TTL value to 0. Once the TTL value is 0, the ICMP Destination unreachable message is sent to the source of the packet. The ICMP message received contains the information of the router at which the TTL value reaches 0. In the next iteration, the source sends the next set of packets by incrementing the TTL value. This packet is then transmitted to various hops/routers by decrementing the TTL value until it becomes 0. Once it becomes 0, the ICMP Destination unreachable message is received along with the information of the intermediate device which is one hop further than the previous hop. This is the method implemented for the Traceroute tool. This is the only approach through which we can determine the route from source to destination but there is no tool which can provide reverse traceroute information. **Figure 2.1** shows the working of traceroute.

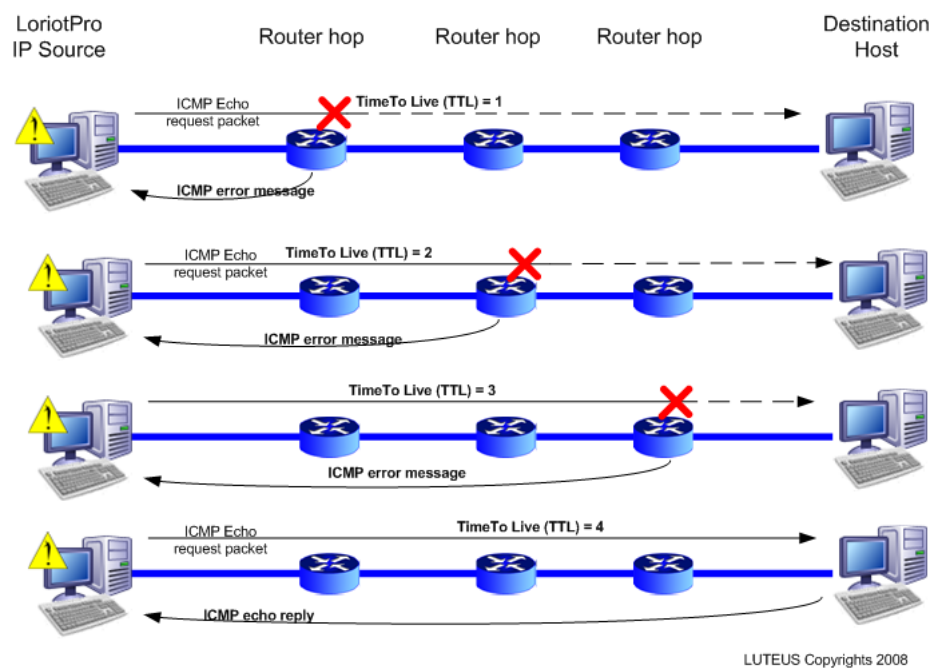


Figure 2.1: Working of traceroute

Chapter 3

Literature survey

Routing asymmetry in the Internet.

Authors: Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker.

Asymmetry can significantly affect the manner in the Internet in its behavior. This paper deals with the study of routing asymmetry in the Internet and present. Their quantitative evaluations and calculations have provided us a measure of the difference in the forward and reverse paths between two end points(systems). There is not been an extensive study on Routing asymmetry before but this is primarily because of the the lack of a organized approach for quantifying asymmetry in the routing if not for simply computing the difference between the lengths of forward and reverse paths. By auditing our framework for representing asymmetry, they considered routing asymmetry for both US higher education academic networks and general commercial networks at two different levels: the Autonomous System (AS) level and the router (or link) level. They not only considered the difference in the length of forward and reverse paths. They measured the AS level routing asymmetry, and provide upper lower bounds on link level routing asymmetry and their studies show that academic networks appear to be more symmetric than general commercially deployed networks. Even their studies demonstrate that routing asymmetry exhibits a skewed distribution, a kind off distribution where a few end-points seems to display a higher participation extent on asymmetric routes.

Avoiding traceroute anomalies with Paris traceroute.

Authors : Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, and Renata Teixeira.

Widely used tool for the diagnosis of network problems and to assemble the internet maps. However, there are a quite number of serious problems with this tool, due to the presence of load balancing routers in the network. This paper described a number of anomalies(something that deviates from what is standard, normal, or expected) that arise in almost all the measurements based on traceroute. We group them as loops, cycles, and diamonds. We furnish a new technique called Paris traceroute, which is available public. It controls the contents of packet header to obtain a more precise picture of the actual routes that packets follow. This new tool allows us to find conclusive explanations for some of the anomalies. [1]

Reverse traceroute.

Authors :Katz-Bassett et.

Traceroute is the widely used Internet diagnostic tool. It is used by Network operators to help identify poor performance, misconfigurations in the router and failures in routing. Researchers used to use it to develop Internet map, distinguish ISPs based on the performance, geolocate the routers, and predict the performance. However, the fundamental limitation of traceroute that affects all these applications is that it does not provide reverse path information. Although some of the various public traceroute servers across the Internet provide some visibility but there exists no general method for determining a reverse path from an arbitrary destination. In this paper, they have addressed that longstanding limitation of traceroute tool by building a reverse traceroute system. Their system provides the same information of reverse path and it works when the user may lack control of the destination. They use a variety of measurement techniques to incrementally piece together the path from the destination back to the source. They deploy their system on PlanetLab and compare reverse traceroute paths with traceroutes issued from the destinations. In the median case their tool and 87% of the hops seen in a directly measured traceroute along the same path, versus only 38% if one simply assumes the path is symmetric, a common fallback given the lack of available tools. They then illustrated how they can use their reverse traceroute system to study previously immeasurable aspects of the Internet: they present a case study of how a content provider could use their tool to troubleshoot poor path performance, we uncover more than a thousand peer-to-peer AS links invisible to current topology mapping efforts, and they measure the latency of individual backbone links with average error under a millisecond.[2]

Pros: Easy and fast calculation of Reverse Route.

Cons: only applicable for IPv4 networks and the proposed system is a distributed system.

A Comprehensive Framework for the Geolocalization of Internet Hosts.

Authors :Bernard Wong, Ivan Stoyanov, Emin Gun Sirer.

Determining the physical location of hosts in the internet is a critical task for many new location-aware services. In this paper, they have represented a Octant, a novel, comprehensive framework for determining the location of Internet hosts in the real world. The key insight behind this framework is to pose the geolocalization problem formally as one of error-minimizing constraint satisfaction, to create a system of constraints by deriving them aggressively from network measurements, and to solve the system geometrically to yield the estimated region in which the target resides. This approach gains its accuracy and precision by taking advantage of both positive and negative constraints, that is, constraints on where the node can and cannot be, respectively. The constraints are represented using regions bounded by Bezier curves, allowing precise constraint representation and low-cost geometric operations. The framework can reason in the presence of uncertainty, enabling it to gracefully cope with aggressively derived constraints that may contain errors. An evaluation of Octant using PlanetLab nodes and public traceroute servers shows that Octant can localize the median node to within 22 mi., a factor of three better than other evaluated approaches.

Pros: It provides a comprehensive framework for determining the location of Internet hosts in the real world based solely on network measurements.

Cons: An evaluation of Octant Using PlanetLab nodes and public traceroute servers shows that

Octant can localize the median node to only within 22 mi. [3]

Efficient algorithms for large-scale topology discovery.

Authors :B. Donnet, P. Raoult, T. Friedman, and M. Crovella.

There is a growing interest at the interface level in discovery of topology of internet. Currently being deployed system is a new generation of highly distributed measurement systems. Unfortunately, the research community has not examined the problem of how to perform such measurements efficiently and in a network-friendly manner. In this paper they have made two contributions towards that end. First, they showed that standard methods topology discovery (e.g., skitter) are quite inefficient, constantly probing the same interfaces. This is a matter of interest, because when scaled up, such methods will generate so much traffic that they will begin to resemble DDoS attacks. They measured two kinds of redundancy in probing (intra- and inter-monitor) and showed that both kinds are important as same. They also showed that straightforward approaches to address these two kinds of redundancy should always take opposite tacks, and are thus fundamentally are in conflict. Their second contribution is to evaluate and propose Doubletree, an algorithm that reduces both types of redundancy simultaneously on routers and end systems. The key ideas are to exploit the treelike structure of routes to and from a single point in order to guide when to stop probing, and to probe each path by starting near its midpoint. Our results show that Doubletree can reduce both types of measurement load on the network dramatically, while permitting discovery of nearly the same set of nodes and links.[4]

Internet path failure monitoring and characterization in wide-area services.

Authors :M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer.

Detecting network path anomalies generally requires examining large volumes of traffic data to find misbehavior. We observe that wide-area services, such as peer-to-peer systems and content distribution networks, exhibit large traffic volumes, spread over large numbers of geographically-dispersed endpoints. This makes them ideal candidates for observing wide-area network behavior. Specifically, we can combine passive monitoring of wide-area traffic to detect anomalous network behavior, with active probes from multiple nodes to quantify and characterize the scope of these anomalies. This approach provides several advantages over other techniques: (1) we obtain more complete and finer-grained views of failures since the wide-area nodes already provide geographically diverse vantage points; (2) we incur limited additional measurement cost since most active probing is initiated when passive monitoring detects oddities; and (3) we detect failures at a much higher rate than other researchers have reported since the services provide large volumes of traffic to sample.[5]

Effective diagnosis of routing disruptions from end systems.

Authors :Y. Zhang, Z. M. Mao, and M. Zhang.

Internet routing events are known to introduce severe disruption to applications. So far effective diagnosis of routing events has relied on proprietary ISP data feeds, resulting in limited ISP-centric views not easily accessible by customers or other ISPs. In this work, we propose a novel approach to diagnosing significant routing events associated with any large networks from the perspective of end systems. Our approach is based on scalable, collaborative probing launched from end systems and does not require proprietary data from ISPs. Using a greedy scheme for event correlation and cause inference, we can diagnose both inter-domain and intra-domain routing events. Unlike existing methods based on passive route monitoring, our approach can also measure the impact of routing events on end-to-end network performance. We demonstrate the effectiveness of our approach by studying five large ISPs over four months. We validate its accuracy by comparing with the existing ISP-centric method and also with events reported on NANOG mailing lists. Our work is the first to scalably and accurately diagnose routing events associated with large networks entirely from end systems.[6]

Scalable and Accurate identification of AS-level Forwarding Paths.

Authors :Z. Mao, D. Johnson, J. Rexford, J. Wang, R. Katz.

Traceroute is used heavily by network operators and researchers to identify the IP forwarding path from a source to a destination. In practice, knowing the autonomous system (AS) associated with each hop in the path is also quite valuable. In previous work we showed that the IP-to-AS mapping extracted from BGP routing tables is not sufficient for determining the AS-level forwarding paths. By comparing BGP and traceroute AS paths from multiple vantage points, Z. Morley Mao et al. (2003) proposed heuristics that identify the root causes of the mismatches and fix the inaccurate IP-to-AS mappings. These heuristics, though effective, are labor-intensive and mostly ad hoc. This paper proposes a systematic way to construct accurate IP-to-AS mappings using dynamic programming and iterative improvement. Our algorithm reduces the initial mismatch ratio of 15% between BGP and traceroute AS paths to 5% while changing only 2.9% of the assignments in the initial IP-to-AS mappings. This is in contrast to the results of Z. Morley Mao et al. (2003), where 10% of the assignments were modified and the mismatch ratio was only reduced to 9%. We show that our algorithm is robust and can yield near-optimal results even when the initial mapping is corrupted or when the number of probing sources or destinations is reduced.[7]

Touring the Internet in a TCP sidecar.

Authors :R. Sherwood and N. Spring.

An accurate router-level topology of the Internet would benefit many research areas, including network diagnosis, inter-domain traffic engineering, and overlay construction. We present TCP Sidecar and Passenger, two elements of a system for router-level Internet topology discovery. Sidecar transparently injects measurement probes into non-measurement TCP streams, while Passenger combines TTL-limited probes with the often-ignored IP record route option. The combined approach mitigates problems associated with traceroute-based topology discovery, including abuse reports, spurious edge inference from multi-path routing, unresolved IP aliases, long network timeouts, and link discovery behind NATs and firewalls. We believe that we are the first mapping project to measure MPLS use with ICMP extensions and record route behavior when the TTL is not decremented. We are able to discover NATs when monitoring TCP connections that tunnel through them. In this paper, we present preliminary results for TCP Sidecar and Passenger on PlanetLab. Our experiments inject measurement probes into traffic generated both from the CoDeeN Web proxy project and from a custom web crawler to 166,745 web sites.[8]

Internet control message protocol/RFC 791.

Author :J. Postel.

This document specifies a transition mechanism algorithm in addition to the mechanisms already specified in [TRANS-MECH]. The algorithm translates between IPv4 and IPv6 packet headers (including ICMP headers) in separate translator "boxes" in the network without requiring any per-connection state in those "boxes". This new algorithm can be used as part of a solution that allows IPv6 hosts, which do not have a permanently assigned IPv4 addresses, to communicate with IPv4-only hosts. The document neither specifies address assignment nor routing to and from the IPv6 hosts when they communicate with the IPv4-only hosts.

Chapter 4

Methodology

To provide the Reverse TraceRoute information for the networks bearing the IPv4 networks, we make use of a standard Network model called the client server Model. We make use of Socket Programming in c to implement the Client server model. The client/server architecture splits the business world into 2 parts: one that sends the request to a particular device for a service (a Client) and another one part/device that provides the service (a Server). This model is based on the distribution of system methods between two independent and autonomous processes: that of client and server. These Client and Server processes must be able to communicate with each other. The client/server architecture can be implemented both on the Distributed and the stand alone systems. If these two components are located on separate but connected computers, client/server system becomes at the same time distributed system. However it is not required and client/server architecture may be implemented on a stand alone computer.

4.1 Components Required

The three components required in the implementation of the Client /Server architecture are:

- Client it is any system that requests for a specific services from the Server processes.
- Server is a process that provides services for the requested services from Clients.
- Communication Middleware These are the supporting components, may it be a software or a hardware, that allows Clients and Servers to communicate with each other.

4.2 Client/Server Methodology

4.2.1 The goal of client/server architecture development

The major goal of the Client/server development is the usage of all processing and hardware capabilities of all computers connected at a system to be maximized. On doing this, The Client/Server will perform in a most effective manner in entering and getting the information. The benefit of this performance is to provide timely information retrieval to end users and also to enhance corporate competitiveness. the above mentioned are Accomplished by

- client/server targets are attained by breaking down the entire set of methods within a system (as much as possible) and developing each function as separate applications. Some applications are the Client Applications that request for a particular service functions; and other applications are those, The, Server Applications that provide service functions to those device requesting for a service.
- Until the processing demands are balanced between all computers in the system, one or more applications are placed on a separate computer in a system. on doing so, the system performance is maximized.

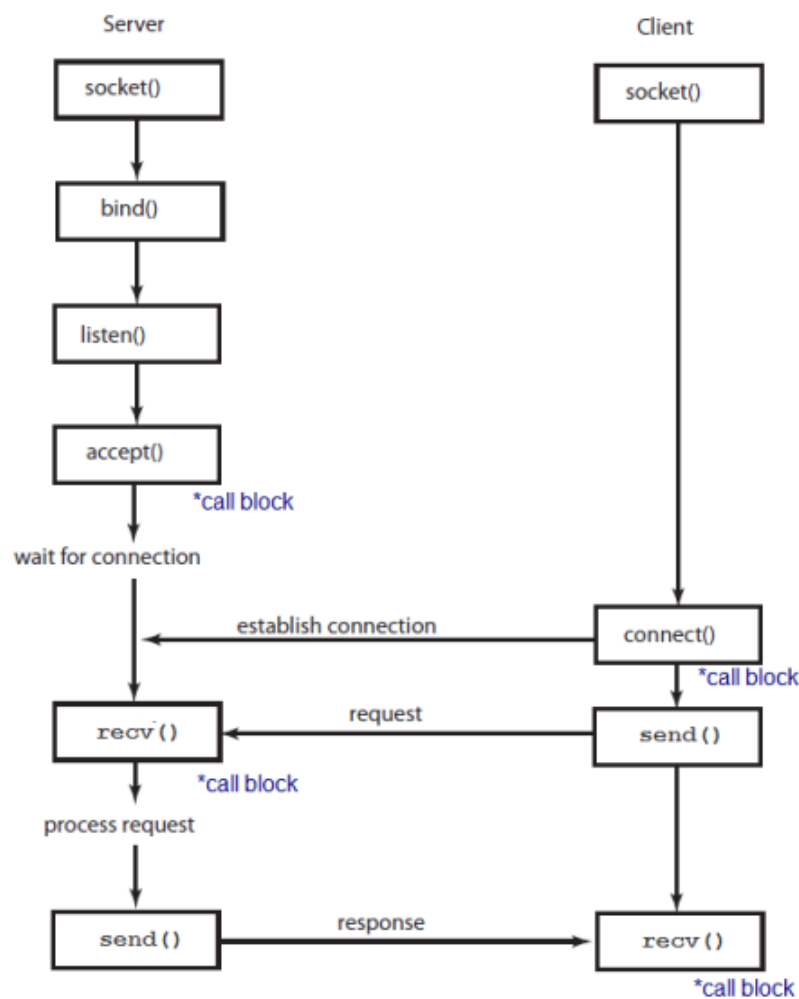


Figure 4.1: TCP IP socket diagram

4.2.2 Issues with the Database Design

The changes to the Conceptual Database Design is not required by the Client/Server architecture, But to support the Physical Implementation of the database and development of the overall application, we need to enhance the Database Design. The two processes of this architecture, the Client and the Server processes may reside on the same system or on different

systems which are connected together by a network. In order to implement the Client Server architecture the operating system must be capable to support multitask processing, it is the basic requirement for any OS.

4.3 Client/Server Components

4.3.1 Client

It is a system or a process that requests specific services from the Server System/Processes. It is the front-end application that is interacted by end-user. The conversation with the server is always initiated by the client. Since the client is the Front-end application only visible to the End-users, it always comprises UI and optionally comprises other parts of the application software.

4.3.2 Server

Is a that system or a process that provides services for the requested Clients. It is the back-end application that interacts with the background services for the client server. These servers continuously listen for the requests from the clients. few of the services that the servers serve are:

- **File Services** LAN environment's disk storage with a fast hard drive system.
- **Print Services** printing usually involves a shared printer accessible on the LAN. with this application, there is usually local data processing done on the print server to offload work from the database computer.
- **Fax Services** a computer is equipped with a Fax modem and dedicated modem line to be shared with other Clients on the system.
- **Communication Services** a computer should be dedicated to external transmissions if a client PC must occasionally connect to other host computers or services to which the client is not directly connected
- **Database Services** this is central to the client/server information system. The client sends SQL statements to the database server. The database server (DBMS) receives the SQL code, validates it, executes it, and sends only the results to the client. The data and database engine are located in the database server.
- **Transaction Services** a transaction server is connected to the database server. A transaction server contains the database transaction code or procedures that manipulate the data in the database. The front-end application (client) sends a request to the transaction server to execute a specific procedure stored on the database server. No SQL code travels through the network. As a result, transaction servers reduce network traffic and provide better performance than database servers.
- **Miscellaneous Services** anything not already specified, such as shared CD-ROM, Tape drives, and backups.

4.3.3 Communications middleware

It is a computer process used to maintain communication between the client and servers, which requires a network. All client requests and server responses travel through the network as messages that contain control information and data.

4.4 Thin vs. Fat client

Depending on how much of data processing provided by a client machine, we consider the following definitions:

- Thin client a system that carries out a minimum of processing on the client side when compared to the server side.
- Fat client a system that carries out a larger part of the processing on the client side compared to the server side.
- Thin server a system that carries out a minimum of processing on the server side compared to the client side.
- Fat server a system that carries out a larger part of the processing on the server side compared to the client side.

4.5 Tiers

4.5.1 2-Tier Client / Server System

The client requests services directly from the server. 2-Tier systems involve the requests only from the client to server. A 1-tier system is a bundled application with no separation of processors. Those kind of the systems are NOT client/server systems.

4.5.2 3-Tier Client / Server System

The client requests are managed by the intermediate servers, which co-ordinate the client requests with subordinate servers. 3-Tier systems involve client to server to server requests. 3-Tier system example: Client Database Middle-ware Database Server

4.5.3 Multi-Tier Client / Server System

The client requests are managed by intermediate servers, which co-ordinate the client requests with subordinate servers. The example: Client Application Server (such as WebLogic, WebSphere) Database Middle-ware Database Server

4.6 Client / Server Main Principles

This Client/Server architecture is based on Software and Hardware components that communicates between them to form a complete system. The following principles are the basics on which client/server systems are constructed:

- Hardware Independence

All the components of this architecture should work in the same way irrespective of the hardware in use.

- Software independence

All the components of this design should be able to be executed on choices of software, and should not be depended on one software choice.

- Open Access to Services

Any kind of Services must be provided by the server on demand of the client.

- Procedure distribution

The information to be processed is divided equally among the components of this architecture. The distribution of these procedures should accord to these laws:

- Client and Server Processes should run its procedures self-independent of other clients and servers, I.e. They must be Autonomous.
- Resource Optimization - The Server must be able to provide services to multiple requests from multiple clients to increase the usage of resources.
- Scalability and flexibility - To run more powerful Hardware and Software platforms it is required that the client and server processes should be easily upgrade able.
- Interoperability and integration - require that client and server processes to be indefectible, where bartering a server process is invisible to the client process. Also, integration between independent databases is becoming more and more common, especially due to the Internet presence and requirements.

Chapter 5

Implementation

5.1 Implementation requirement

We have implemented our project in Linux Environment using c programming language. We used GNU GCC compiler. Some support scripts have been written in bash scripting language. C header files required are

- `stdio.h` for standard I/O
- `stdlib.h` for exit status macros
- `string.h` for string operations
- `sys/types.h` for inbuilt data structures
- `sys/socket.h` for socket operations
- `netinet/in.h` for internet address families
- `errno.h` for error handling
- `fcntl.h` for function control
- `unistd.h` for POSIX complaint APIs

5.2 File hierarchy

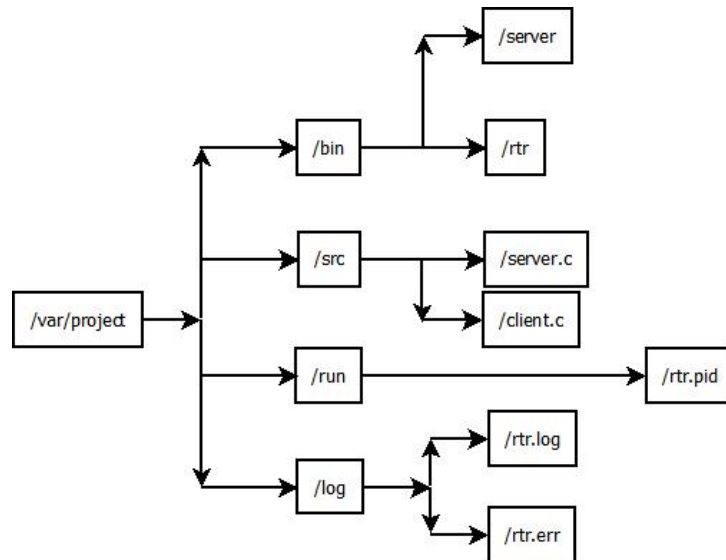


Figure 5.1: File hierarchy

- /bin - contains the binaries. server and rtr(reverse traceroute) files.
- /src - contains the source code of the components.
- /run - contains the file which has pid of the server program.
- /log - contains rtr.log and rtr.err files which has logs and standers error info respectively.

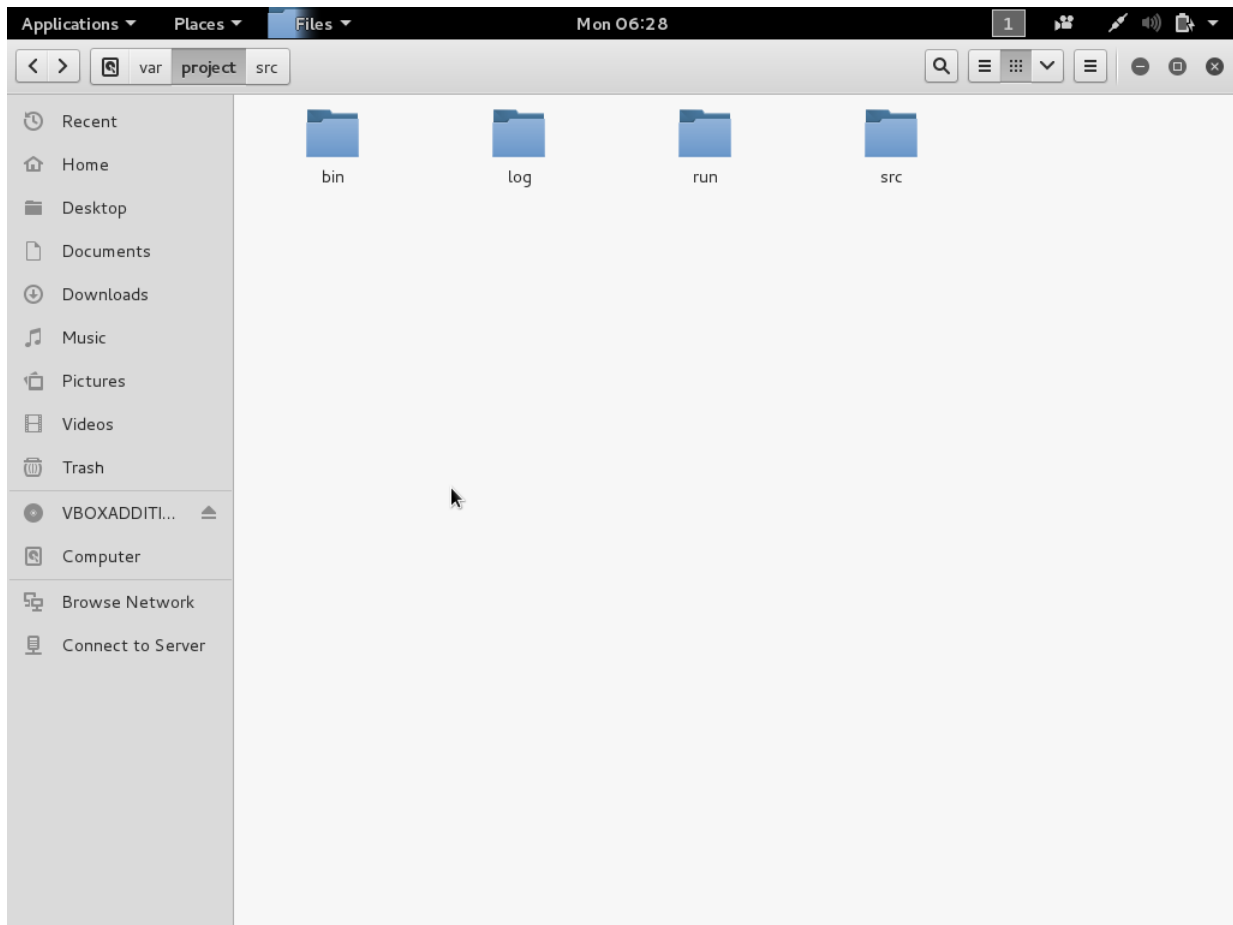


Figure 5.2: Project DIR

5.3 Functions

5.3.1 Server functions

On the server side we use the following user defined functions

- `int init_socket();`

Creates socket using `socket()` system function, clears memory of `sockaddr` structure member using `bzero()` function and Initializes the `sockaddr` member variables like `sin_family`, `sin_addr.s_addr`, `sin_port`, and binds IP and port using `bind()` system function. `errno` is set to appropriate errors in system functions.

- `int accept_connection(int);`

Used to accept TCP connections from client. This function takes the local socket as parameter of type `int` and after connection is established returns the socket descriptor of type `int`. It also initializes the structure type `sockaddr` with the client information. `errno` is set to appropriate errors in system functions.



Figure 5.3: DFD for listen



Figure 5.4: DFD for accept_connection

- `void read_query(int socket);`

Takes the connected socket of type int as a parameter and reads the input query from client into a global string variable. `errno` is set to appropriate errors in functions.



Figure 5.5: DFD for read_query

- `void create_query();`

After reading row query from the client through socket it has to be converted into executable command.



Figure 5.6: DFD for create_query

5.3.2 Client Functions

- `int establish_connection(int sockfd);`

This function reads command line argument which is an ip address and tries to establish TCP connection on port 7007. On error it sets `errno` to appropriate value.



Figure 5.7: DFD for establish_connection

- `void recv_rtr_info(int sockfd);`

This function uses `recv()` system function to read the processed information from the destination. It takes socket descriptor as its parameter. In case of error `errno` is set to appropriate value.



Figure 5.8: DFD for recv_rtr_info

Chapter 6

Verification and validation

Verification and validation is an important phase in the development life cycle of the product. This is the part of the Software development life cycle in which the faults are detected. Hence testing/ verification and validation play a very carping role for the assurance of the quality and ensuring the dependability of the software. One description for testing would be ” Software testing is a process of executing a program or a code or an application with the intention of finding the bugs in them., where the tester attempts to verify the software or the process by giving the set of test cases or inputs and verifies the result with the expected one.

6.1 Purpose of Testing

- The most and basic purpose behind Software Testing is to ensure that the program under examination is bug free and error free and in if it is not, it should be identified and sent for further amendments.
- Software testing plays a important role in improving the quality of the software/code by intimating bugs at different phases of development depending on the methodology followed for the project.
- Software testing makes sure if the application is ready to get delivered to the customer.
- The goal of software testing should primarily be an extension to have traceable tests with coverage of the priority requirements.
- The goal aims to ensure the flaws in application; methodologies used and are put forward in front of management in order to get it resolved.

Software testing on the whole is largely aimed at testing the application with the intent to keep the end user in mind. The goal is narrowed to the frame set where software delivery (free of bugs) within the scheduled time-line turns out to be the major element of the testing. The customer can access the application without any issue and support organizational effort should be reduced to a large extent.

6.2 LEVELS OF TESTING

6.2.1 Unit Testing

Individual components are tested to ensure that they operate correctly. Each component is tested independently. This system was tested with the set of noisy images for each module and the results were checked with the expected output. Unit testing focuses on verification effort on the smallest unit of the software design module. This is also known as MODULE TESTING. This testing is carried out during phases, each module is found to be working satisfactorily as regards to the expected output from the module.

6.2.2 Integration Testing

Integration testing is another aspect of testing that is generally done in order to uncover errors associated with the flow of data across interfaces. The unit-tested modules are grouped together and tested in small segment, which makes it easier to isolate and correct errors. This approach is continued until we have integrated all modules to form the system as a whole.

6.2.3 System Testing

System testing is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

6.2.4 Validation Testing

The validation testing can be defined in many ways, but a simple definition is that, validation succeeds when the software functions in a manner that can be reasonably expected by the end user.

6.3 TYPES OF TESTING

6.3.1 Functional Testing

Functional testing is a quality assurance (QA) process and a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output. Functional Testing usually describes what the system does. Functional testing typically involves five steps:

- The identification of functions that the software is expected to perform.
- The creation of input data based on the specifications.
- The determination of output based on the specifications.
- The execution of the test case.
- The comparison of actual and expected outputs.
- To check whether the application works as per the customer need.

Table 6.1: Functional test results

Sl. No	Functionality	Module	Result
1	Creating a Socket / Initializing the socket Variables	init_socket() initialize_socket()	Tested OK
2	Binding Address and Socket	init_socket() initialize_socket()	Tested OK
3	Maximum Number of Connections	listen()	Tested OK
4	Accept the Request	accept_connection()	Tested OK
5	Convert IP addr from network bites to string	inet_ntop()	Tested OK
6	Form the in a Proper way	create_query()	Tested OK
7	Read Query from an established Connection	read_query()	Tested OK
8	Try till we keep connecting to a Server	establish_connection()	Tested OK
9	Receive the Reverse Traceroute Information	recv_rtr_info()	Tested OK

6.3.2 Structural Testing

Structural testing, also known as white-box testing is a method of testing the software that tests internal structures or workings of an application. In white-box testing, an internal perspective of the system, as well as programming skills are used to design test cases. White-box testing is one of the two biggest testing methodologies used today. It primarily has three advantages:

- Side effects of having the knowledge of the source code is beneficial to thorough testing.
- Optimization of code by revealing hidden errors and being able to remove these possible defects.
- Gives the programmer introspection because developers carefully describe any new implementation.

6.3.3 Performance Testing

Performance testing is performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

6.3.4 Acceptance Testing

Acceptance testing is a test conducted to determine if the requirements of a specification or contract are met. The acceptance test suite is run against the supplied input data or using an acceptance test script to direct the testers. Then the results obtained are compared with the expected results. If there is a correct match for every case, the test suite is said to pass. The objective is to provide confidence that the delivered system meets the requirements. The acceptance phase may also act as the final quality gateway, where any quality defects not previously detected may be uncovered.

6.3.5 Regression Testing

Regression testing is a type of software testing that seeks to uncover new software bugs, or regressions, in existing functional and non-functional areas of a system after changes such as enhancements, patches or configuration changes, have been made to them. The intent of regression testing is to ensure that a change such as those mentioned above has not introduced new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software. Common methods of regression testing include rerunning previously completed tests and checking whether program behavior has changed and whether previously fixed faults have re-emerged.

6.3.6 Alpha Testing

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.

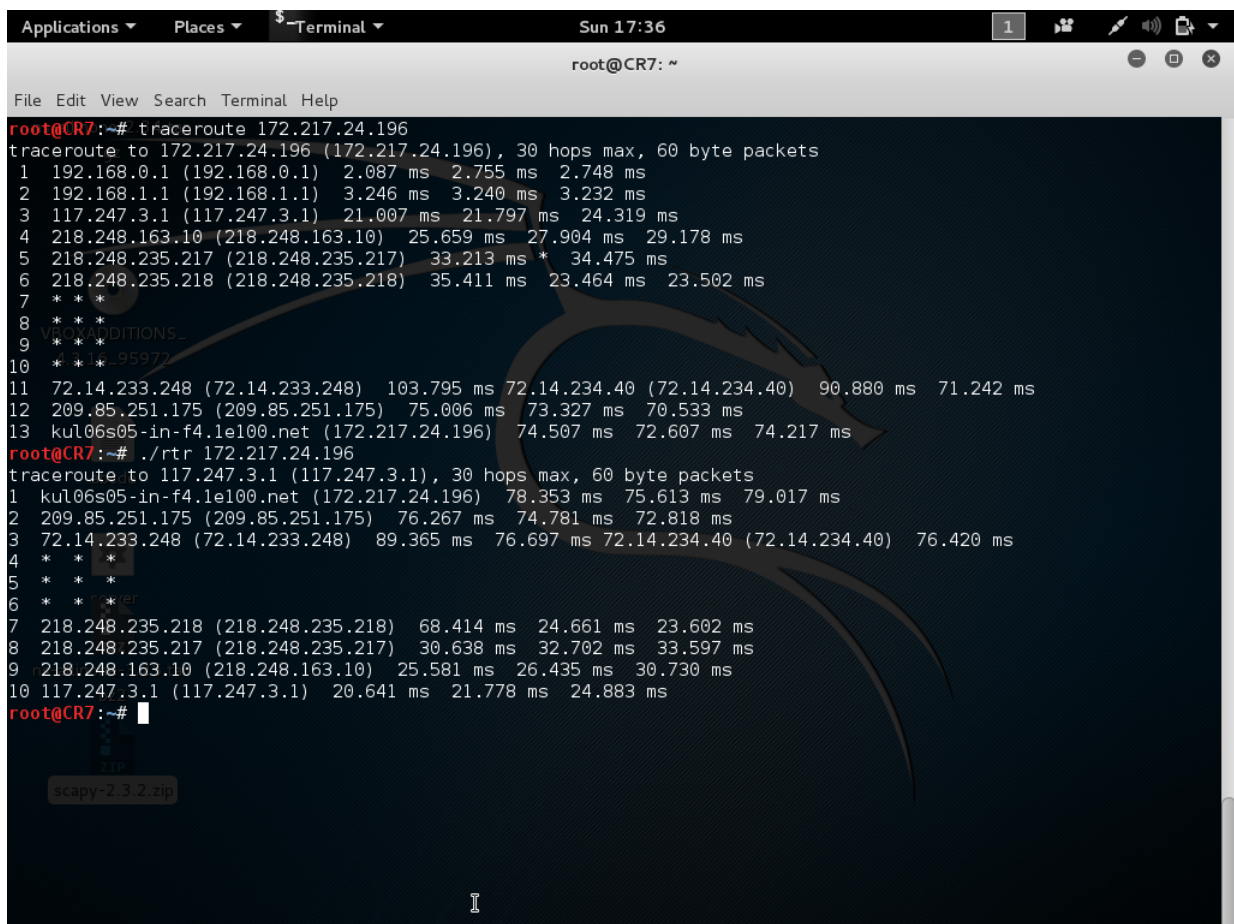
6.3.7 Beta Testing

Beta testing comes after alpha testing and can be considered as a form of external user acceptance testing. Versions of the software, known as beta versions, are released to groups of people so that further testing can ensure the product has few faults or bugs.

Chapter 7

Results

After successful testing of both server components and client components we placed server and client in two different networks with internet access. Then we executed the standard "traceroute" command for foreword path from client system to server system and then we executed our rtr(reverse traceroute) under similar scenario. We got identical paths for both foreword and reverse path. Screen-short of the result is shown below.



```
root@CR7:~# traceroute 172.217.24.196
traceroute to 172.217.24.196 (172.217.24.196), 30 hops max, 60 byte packets
 1  192.168.0.1 (192.168.0.1)  2.087 ms  2.755 ms  2.748 ms
 2  192.168.1.1 (192.168.1.1)  3.246 ms  3.240 ms  3.232 ms
 3  117.247.3.1 (117.247.3.1)  21.007 ms  21.797 ms  24.319 ms
 4  218.248.163.10 (218.248.163.10)  25.659 ms  27.904 ms  29.178 ms
 5  218.248.235.217 (218.248.235.217)  33.213 ms * 34.475 ms
 6  218.248.235.218 (218.248.235.218)  35.411 ms  23.464 ms  23.502 ms
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  72.14.233.248 (72.14.233.248)  103.795 ms  72.14.234.40 (72.14.234.40)  90.880 ms  71.242 ms
12  209.85.251.175 (209.85.251.175)  75.006 ms  73.327 ms  70.533 ms
13  ku106s05-in-f4.1e100.net (172.217.24.196)  74.507 ms  72.607 ms  74.217 ms
root@CR7:~# ./rtr 172.217.24.196
traceroute to 117.247.3.1 (117.247.3.1), 30 hops max, 60 byte packets
 1  ku106s05-in-f4.1e100.net (172.217.24.196)  78.353 ms  75.613 ms  79.017 ms
 2  209.85.251.175 (209.85.251.175)  76.267 ms  74.781 ms  72.818 ms
 3  72.14.233.248 (72.14.233.248)  89.365 ms  76.697 ms  72.14.234.40 (72.14.234.40)  76.420 ms
 4  * * *
 5  * * *
 6  * * *
 7  218.248.235.218 (218.248.235.218)  68.414 ms  24.661 ms  23.602 ms
 8  218.248.235.217 (218.248.235.217)  30.638 ms  32.702 ms  33.597 ms
 9  218.248.163.10 (218.248.163.10)  25.581 ms  26.435 ms  30.730 ms
10  117.247.3.1 (117.247.3.1)  20.641 ms  21.778 ms  24.883 ms
root@CR7:~#
```

Figure 7.1: Foreword and reverse traceroute

Some times when port is already in use bind() system call cannot bind IP and port then it give port already in use message

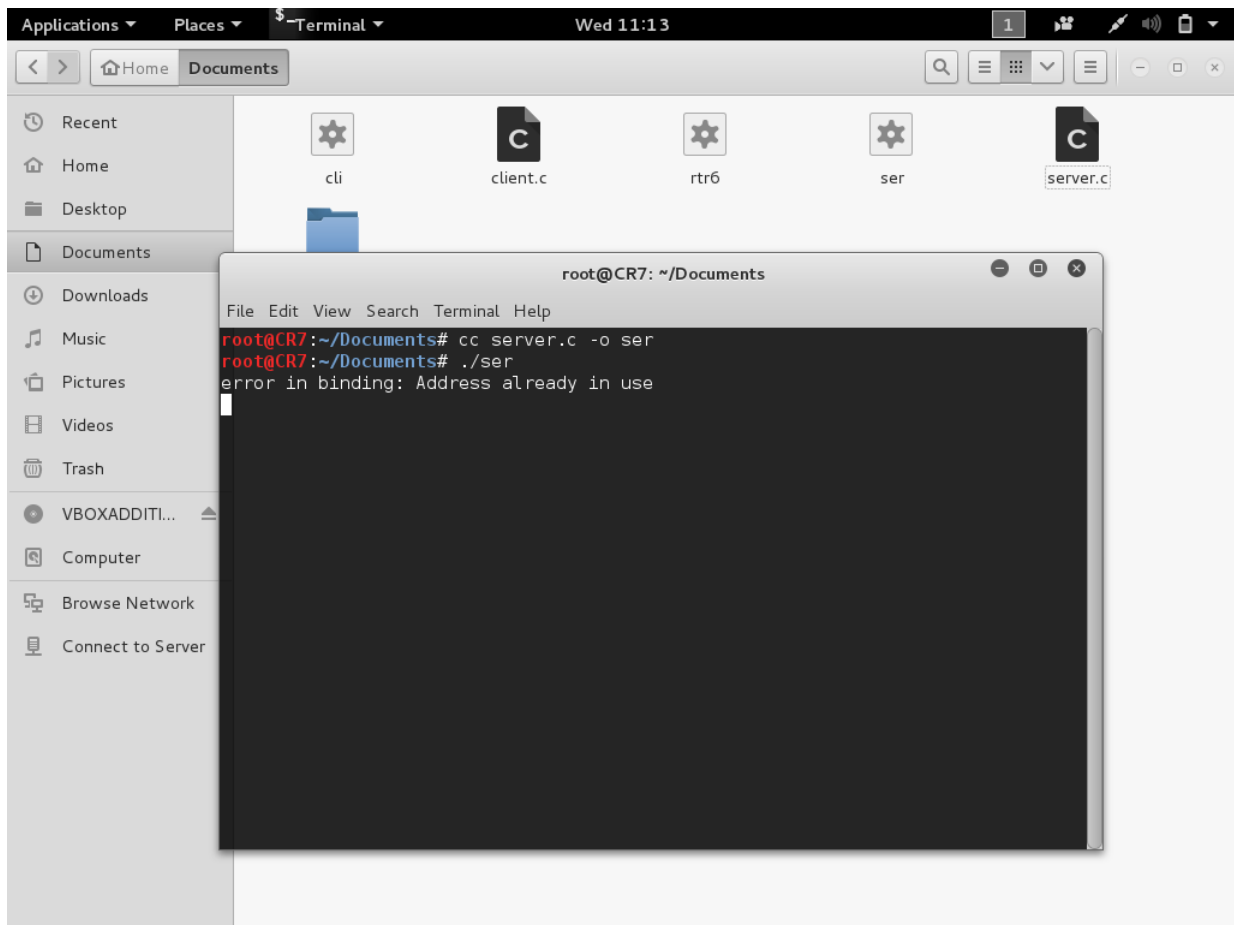
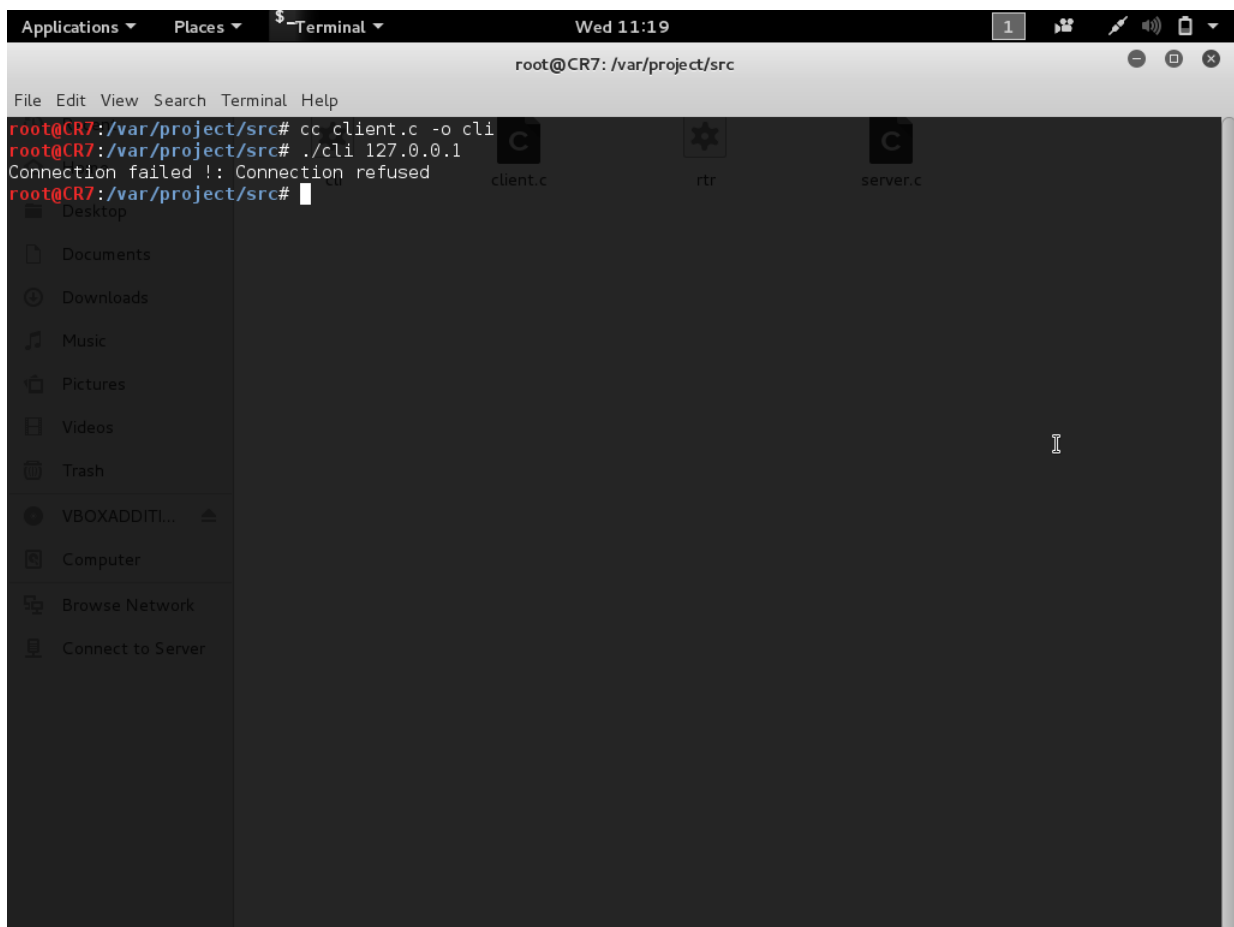


Figure 7.2: Binding error

When connections exceed more then the parameter to listen() function connection is simply terminated

A screenshot of a Linux terminal window. The window title is "Terminal" and it shows the current time as "Wed 11:19". The prompt is "root@CR7: /var/project/src". The user has entered the command "cc client.c -o cli" and then "cli 127.0.0.1". The output shows "Connection failed !: Connection refused". The terminal window has a sidebar on the left with a file manager view showing "Desktop", "Documents", "Downloads", "Music", "Pictures", "Videos", "Trash", "VBOXADDITI...", "Computer", "Browse Network", and "Connect to Server". The main area of the terminal is dark with a light cursor. There are also some icons in the top right corner of the terminal window, including a star, a magnifying glass, and a power button.

```
root@CR7: /var/project/src# cc client.c -o cli
root@CR7: /var/project/src# ./cli 127.0.0.1
Connection failed !: Connection refused
root@CR7: /var/project/src#
```

Figure 7.3: Connection failed error

Chapter 8

Conclusion

Although widely-used and popular, traceroute is fundamentally limited in that it cannot measure reverse paths . This limitation leaves network operators and researchers unable to answer important questions about Internet topology and performance. To solve this problem, we developed a reverse traceroute system to measure reverse paths from arbitrary destinations back to the user. To reach the requirement We initially set the Record Route option to the packets that we send, but most of the intermediate routers/hops dropped these packets for some security reasons. So, implementing this method just next to impossible. Then, according to the base paper, in one of the approached method, we need to configure all the intermediate routers/hops in such a way that a prior information has to be given to the routers as to which packets are to be allowed to pass through them. These information is placed in access lists. There will be a Hardware dependency on implementing this method.

To overcome all these, We Proposed a client-server system which provides the reverse information of the processed packets. This approach towards building the Reverse path from destination back to the source, works accurately for the networks which implement both IPv4 and IPv6 protocols. And moreover there is no hardware dependency. Since, not all the networks supports the IPv6 protocol, Our tool provides the accurate output for IPv4 implemented networks, and for IPv6 networks, we show the result for the loop back address. From the result obtained by performing the Reverse Traceroute to the IP address and cross verifying the same result on performing the standard Traceroute to the same IP, it is clear that the routing in internet is not symmetric.

Bibliography

- [1] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with paris traceroute. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 153–158. ACM, 2006. 3
- [2] Ethan K Bassett, Harsha V Madhyastha, Vijay K Adhikari, Colin Scott, Justine Sherry, Peter Van Wesep, Thomas Anderson, and Arvind Krishnamurthy. Reverse traceroute. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation, NSDI*, volume 10, page 15, 2010. 3
- [3] Bernard Wong, Ivan Stoyanov, and Emin Gün Sirer. Octant: A comprehensive framework for the geolocalization of internet hosts. In *NSDI*, volume 7, pages 23–23, 2007. 3
- [4] Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Efficient algorithms for large-scale topology discovery. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 327–338. ACM, 2005. 3
- [5] Ming Zhang, Chi Zhang, Vivek S Pai, Larry L Peterson, and Randolph Y Wang. Planetseer: Internet path failure monitoring and characterization in wide-area services. In *OSDI*, volume 4, pages 12–12, 2004. 3
- [6] Ying Zhang, Zhuoqing Morley Mao, and Ming Zhang. Effective diagnosis of routing disruptions from end systems. In *NSDI*, volume 8, pages 219–232, 2008. 3
- [7] Z Morley Mao, David Johnson, Jennifer Rexford, Jia Wang, and Randy Katz. Scalable and accurate identification of as-level forwarding paths. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1605–1615. IEEE, 2004. 3
- [8] Rob Sherwood and Neil Spring. Touring the internet in a tcp sidecar. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 339–344. ACM, 2006. 3

Appendix

Source code of Client.c

```
/* ++++++ REVERSE TRACEROUTE CLIENT ++++++ */

/* ===== Headers ===== */

#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<errno.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<fcntl.h>

/* ===== Macros ===== */

#define PORT_NO 7007
#define MAX_FILE_LENGTH 4000
#define MAX_CMD_LENGTH 256
#define MAX_ADDR_LENGTH 56

/* ===== Global variables ===== */

char rtr_info[MAX_FILE_LENGTH],*ip_addr;
struct sockaddr_in serv_addr;
int len;

/* ===== Functions ===== */

int initialize_socket() {
int sockfd ;
sockfd = socket(AF_INET,SOCK_STREAM,0);
// Create a socket
bzero((char*)&serv_addr,sizeof(serv_addr));
```

```
// Clear memory
serv_addr.sin_family=AF_INET;
serv_addr.sin_port=htons(PORT_NO);
serv_addr.sin_addr.s_addr=inet_addr(ip_addr);
// Take IP from CLA
return sockfd;
}

int establish_connection(int sockfd) {
int err;
err = connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
// Try connecting to
// server
if(err == -1) {
perror("Connection failed !");
exit(1);
}
return sockfd;
}

void send_query(int sockfd) {
int err;
err = write(sockfd,"traceroute",MAX_CMD_LENGTH);
// Send Reverse
// Traceroute query
if(err == -1) {
perror("Write function failed");
exit(1);
}
}

void recv_rtr_info(int sockfd) {
int err;
bzero(rtr_info,MAX_FILE_LENGTH);
// Clear memory
err = recv(sockfd,rtr_info,MAX_FILE_LENGTH-1,0);
// Receive the
// processed request
if(err == -1) {
perror("error in receiving");
exit(1);
}
printf("%s\n",rtr_info);
// Display RTR
// information
}

int main(int argc, char **argv)
{
```

```
int e,sockfd;
ip_addr = argv[1];
sockfd = initialize_socket();
sockfd = establish_connection(sockfd);
send_query(sockfd);
recv_rtr_info(sockfd);
close(sockfd);
return 0;
}
```

Source code for server.c

```
/* ++++++REVERSE TRACEROUTE SERVER+++++ */

/* ===== Headers ===== */

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<errno.h>
#include<fcntl.h>
#include<unistd.h>

/* ===== Macros ===== */

#define PORT_NO 7007
#define MAX_FILE_LENGTH 4000
#define MAX_CMD_LENGTH 256

/* =====Global variables===== */

struct sockaddr_in serv_addr,clin_addr;
char xcmd[MAX_CMD_LENGTH],cmd[MAX_CMD_LENGTH],c_addr[INET_ADDRSTRLEN];
/* ===== Functions ===== */

int accept_connection(int sockfd) {
int newsockfd,clienlen;
clienlen = sizeof(clin_addr);
newsockfd = accept(sockfd,(struct sockaddr*)&clin_addr, \
&clienlen);
    // Accept the Request
if(newsockfd == -1) {
```

```
perror("Accept function failed");
exit(1);
}
return newsockfd;
}

void read_query(int socket) {
int err;
bzero(cmd,MAX_CMD_LENGTH-1);
    // Clear memory for
    // incoming data
err = read(socket,cmd,MAX_CMD_LENGTH-1);
    // Read query from
    // established connection
printf("%s",cmd);
if(err == -1) {
perror("Read function failed");
exit(0);
}

}

int init_socket() {
int sockfd,err ;
sockfd = socket(AF_INET,SOCK_STREAM,0);
    // Creating socket
bzero((char*)&serv_addr,sizeof(serv_addr));
    // Clear memory of
    // the structure variable
serv_addr.sin_family=AF_INET;
    // Initialize
    // structure variable
serv_addr.sin_addr.s_addr=INADDR_ANY;
serv_addr.sin_port=htons(PORT_NO);
err = bind(sockfd,(struct sockaddr *)&serv_addr, \
sizeof(serv_addr));
    // Binding ADDR and PORT
if(err == -1) {
perror("error in binding");
exit(1);
}
return sockfd;
}
```



```
void create_query() {
    int i,j;
    for(i=0;cmd[i]!='\0';i++) {
        xcmd[i]=cmd[i];
    }
    xcmd[i++]=' ';
    for(j=0;c_addr[j]!='\0';j++) {
        xcmd[i++]=c_addr[j];
    }
    strcat(xcmd," >");
    strcat(xcmd,c_addr);
}
```

```
int main(int argc, char **argv) {
    int err,e,fd,sockfd,newsockfd,i,j;
    char c[MAX_FILE_LENGTH];
    sockfd = init_socket();
    while(1) {
        e = listen(sockfd,5);
        // Maximum number
        // of connections
        if(e == -1) {
            perror("error in listening");
            exit(1);
        }
        newsockfd = accept_connection(sockfd);
        inet_ntop(AF_INET, &(clin_addr.sin_addr),c_addr, INET_ADDRSTRLEN);
        // Convert IP addr from
        // network bites to string
        read_query(newsockfd);
        create_query();
        system(xcmd);
        // Process Request...
        fd = open(c_addr,O_RDONLY,0);
        if(fd == -1) {
            perror("Open function failed");
            exit(1);
        }
        err = read(fd,c,MAX_FILE_LENGTH-1);
        if(err == -1) {
            perror("Read function failed");
            exit(1);
        }
    }
}
```

```
}
e = send(newsockfd,c,MAX_FILE_LENGTH-1,0);
    // Send the Output...
if(e == -1) {
perror("error in sending");
exit(1);
}
close(newsockfd);
}
return 0;
}
```

Source code for serverIPv6.c

```
/* ++++++REVERSE TRACEROUTE SERVER+++++ */

/* ===== Headers ===== */

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<errno.h>
#include<fcntl.h>
#include<unistd.h>

/* ===== Macros ===== */

#define PORT_NO 7007
#define MAX_FILE_LENGTH 4000
#define MAX_CMD_LENGTH 256

/* ===== Global variables ===== */

struct sockaddr_in serv_addr,clin_addr;
char xcmd[MAX_CMD_LENGTH],cmd[MAX_CMD_LENGTH],c_addr[INET_ADDRSTRLEN];
/* ===== Functions ===== */

int accept_connection(int sockfd) {
int newsockfd,clienlen;
clienlen = sizeof(clin_addr);
newsockfd = accept(sockfd,(struct sockaddr*)&clin_addr, \
&clienlen);
```

```
// Accept the Request
if(newsockfd == -1) {
perror("Accept function failed");
exit(1);
}
return newsockfd;
}

void read_query(int socket) {
int err;
bzero(cmd,MAX_CMD_LENGTH-1);
// Clear memory for
// incoming data
err = read(socket,cmd,MAX_CMD_LENGTH-1);
// Read query from
// established connection
printf("%s",cmd);
if(err == -1) {
perror("Read function failed");
exit(0);
}

}

int init_socket() {
int sockfd,err ;
sockfd = socket(AF_INET,SOCK_STREAM,0);
// Creating socket
bzero((char*)&serv_addr,sizeof(serv_addr));
// Clear memory of
// the structure variable
serv_addr.sin_family=AF_INET;
// Initialize
// structure variable
serv_addr.sin_addr.s_addr=INADDR_ANY;
serv_addr.sin_port=htons(PORT_NO);
err = bind(sockfd,(struct sockaddr *)&serv_addr, \
sizeof(serv_addr));
// Binding ADDR and PORT
if(err == -1) {
perror("error in binding");
exit(1);
}
```

```
return sockfd;  
}
```

```
void create_query() {  
    int i,j;  
    for(i=0;cmd[i]!='\0';i++) {  
        xcmd[i]=cmd[i];  
    }  
    xcmd[i++]=' ';  
    for(j=0;c_addr[j]!='\0';j++) {  
        xcmd[i++]=c_addr[j];  
    }  
    strcat(xcmd," >");  
    strcat(xcmd,c_addr);  
}
```

```
int main(int argc, char **argv) {  
    int err,e,fd,sockfd,newsockfd,i,j;  
    char c[MAX_FILE_LENGTH];  
    sockfd = init_socket();  
    while(1) {  
        e = listen(sockfd,5);  
        // Maximum number  
        // of connections  
        if(e == -1) {  
            perror("error in listening");  
            exit(1);  
        }  
        newsockfd = accept_connection(sockfd);  
        inet_ntop(AF_INET, &(clin_addr.sin_addr),c_addr, INET_ADDRSTRLEN);  
        // Convert IP addr from  
        // network bites to string  
        read_query(newsockfd);  
        create_query();  
        system("/sbin/ip -6 addr add 2001:db8:0:f101::1 dev eth0");  
        //to set the IPv6 address  
        system("traceroute -6 2001:db8:0:f101::1 ");  
        // TraceRoute to this IPv6  
        // address  
        // Process Request...  
        fd = open(c_addr,O_RDONLY,0);  
        if(fd == -1) {  
            perror("Open function failed");  
        }  
    }  
}
```

```
exit(1);
}
err = read(fd,c,MAX_FILE_LENGTH-1);
if(err == -1) {
perror("Read function failed");
exit(1);
}
e = send(newsockfd,c,MAX_FILE_LENGTH-1,0);
    // Send the Output...
if(e == -1) {
perror("error in sending");
exit(1);
}
close(newsockfd);
}
return 0;
}
```

Start up script rtr.sh

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start daemon at boot time
# Description:        Enable service provided by daemon.
### END INIT INFO

dir="/var/project/source/"
cmd="/var/project/source/server"
user=""

name='basename $0'
pid_file="/var/project/run/$name.pid"
stdout_log="/var/project/log/$name.log"
stderr_log="/var/project/log/$name.err"

get_pid() {
    cat "$pid_file"
}

is_running() {
```

```
[ -f "$pid_file" ] && ps 'get_pid' > /dev/null 2>&1
}

case "$1" in
    start)
        if is_running; then
            echo "Already started"
        else
            echo "Starting $name"
            cd "$dir"
            if [ -z "$user" ]; then
                sudo $cmd >> "$stdout_log" 2>> "$stderr_log" &
            else
                sudo -u "$user" $cmd >> "$stdout_log" 2>> "$stderr_log" &
            fi
            echo $! > "$pid_file"
            if ! is_running; then
                echo "Unable to start, see $stdout_log and $stderr_log"
                exit 1
            fi
        fi
    ;;
    stop)
        if is_running; then
            echo -n "Stopping $name.."
            kill 'get_pid'
            for i in {1..10}
            do
                if ! is_running; then
                    break
                fi

                echo -n "."
                sleep 1
            done
            echo

            if is_running; then
                echo "Not stopped; may still be shutting down or \
shutdown may have failed"
                exit 1
            else
                echo "Stopped"
                if [ -f "$pid_file" ]; then
                    rm "$pid_file"
                fi
            fi
        fi
    ;;
esac
```

```
        fi
    else
        echo "Not running"
    fi
    ;;
restart)
$0 stop
if is_running; then
    echo "Unable to stop, will not attempt to start"
    exit 1
fi
$0 start
;;
status)
if is_running; then
    echo "Running"
else
    echo "Stopped"
    exit 1
fi
;;
*)
echo "Usage: $0 {start|stop|restart|status}"
exit 1
;;
esac

exit 0
```