MINOR – II

Systems And Network Programming

Title of the Project:

Server Load Testing And Network Analysis

विद्या तत्त्व ज्योतिरसम:
JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY UNIVERSITY

Members:

Saksham Bhardwaj - 13103402

Siddharth Agarwal - 13103410

Akanksha Singh – 13103414

BATCH – B1

ABSTRACT

What is Load Testing?

Load testing is the process of putting demand on a software system or computing device and measuring its response. Load testing is performed to determine a system's behavior under both normal and anticipated peak load conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation. When the load placed on the system is raised beyond normal usage patterns, in order to test the system's response at unusually high or peak loads, it is known as stress testing. The load is usually so great that error conditions are the expected result, although no clear boundary exists when an activity ceases to be a load test and becomes a stress test.

The term load testing is used in different ways in the professional software testing community. Load testing generally refers to the practice of modelling the expected usage of a software program by simulating multiple users accessing the program concurrently. As such, this testing is most relevant for multi-user systems; often one built using a client/server model, such as web servers. However, other types of software systems can also be load tested.

Load and performance testing analyses software intended for a multi-user audience by subjecting the software to different numbers of virtual and live users while monitoring performance measurements under these different loads. Load and performance testing is usually conducted in a test environment identical to the production environment before the software system is permitted to go live.

SECTION OF THE PROJECT WORKED UPON

Siddharth Agarwal:Server and Client Script for OS X Platform, Establishing Connection between Client and Server, Data Exchange and Explicit Handshaking between Client and Server, Multiple Client Handling via Threading on Server - Side, Connection verified on different systems, Packet Size Modification, Analyses of Performance parameters, Graph Implementation, Packet Sniffer on OS X Platform, Simulation of SYN attack on OS X Platform, Client Management.

Akanksha Singh:Server and Client Script for Windows Platform, Establishing Connection between Client and Server, Data Exchange, Multiple Client Handling via Threading on Server- Side, Connection verified on different systems, Analyses of Performance parameters, Packet Sniffer on Windows Platform, Simulation of SYN attack on Windows Platform.

Saksham Bhardwaj:Establishing Connection between Client and Server, GUI, Calculating Hops via Trace – Route, Connection verified on different systems.

OBJECTIVE

The objective of this project is to create a client-server connection and implement server load testing using major protocol (TCP). Along with Server Load testing, this project aims to analyze performance of the network connection through packet access and modification and report it to it to the user.

The application uses the concept of a file (any format- .mp3, .mp4, .txt etc) as a load, which shall be rendered to multiple connecting clients repeatedly as per their requests. The server being tested can have 'N Number of clients' connected to it, each of them making 'M Number of Requests' to the server. Upon client server interaction, the file is transferred to the client and performance parameters like Response Time and Throughput are computed for multiple packet sizes. Congestion can be visualized during the transfer.

Since the server can handle only a limited amount of load, the number of clients connecting is also limited and additional clients are rejected based upon a predefined limit. This limit shall be determined on the memory of the server which shall be allocated to each of the clients upon request. If a client makes a request that needs more memory than the server can allocate, this client will be rejected and not allowed to connect altogether. In the other scenario, once a client request has been processed completely, the previously allocated memory will be freed and made available for new connections.

The application would also have packet level access to allow the user to choose the size of the packet, in making requests, so that the number of bytes exchanged per packet is as defined by the user. Our Network Analysis tool will allow the user to view and save the actual TCP/IP Header details of the packets received from the server such as – sequence number, acknowledgement number and number of lost packets and retransmissions.

Our load tester will also allow testing the server performance under extreme conditions when the server has high load and is servicing an extraordinarily large number of clients. This can be verified by simulating SYN Attacks on the server by uploading a file multiple timeson the server, through multiple connections, to keep it busy and not facilitating proper transmission to the user.

Under such attacks, the server, will not have enough free memory as it will be kept occupied by false connections. So it will deny service to valid clients - DOS Attack.

BACKGROUND STUDY AND FINDINGS

WHAT IS LOAD TESTING?

Load testing is the process of putting demand on a software system or computing device and measuring its response. Load testing is performed to determine a system's behavior under both normal and anticipated peak load conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation. When the load placed on the system is raised beyond normal usage patterns to test the system's response at unusually high or peak loads, it is known as stress testing. The load is usually so great that error conditions are the expected result, but there is no clear boundary when an activity ceases to be a load test and becomes a stress test.

EXISTING SERVER TESTING TOOLS:

There are many existing server testing tools serving the same purpose differently on the basis of different parameters. Below are few examples:

Pylot - Pylot is a free open source tool for testing performance and scalability of web services. It runs HTTP load tests, which are useful for capacity planning, benchmarking, analysis, and system tuning.

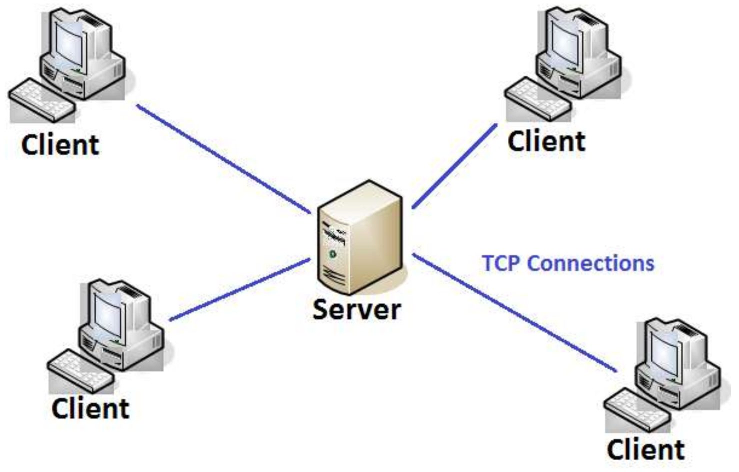
JCrawler - JCrawler is an open-source (under the CPL) Stress-Testing Tool for web-applications. It comes with the crawling/exploratory feature. You can give JCrawler a set of starting URLs and it will begin crawling from that point onwards, going through any URLs it can find on its way and generating load on the web application

Apache JMeter – JMeteris open source software, a 100% pure Java desktop application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

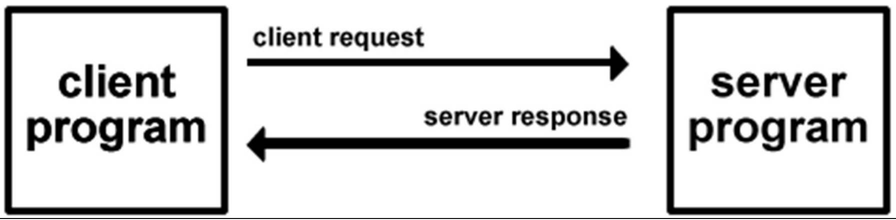
REFERENCE TOOL:

We have designed our application taking reference of APACHE JMETER and APACHE BENCHMARK which tests n simultaneous connections each with n requests and gives a feedback on stats once the process finishes. The application will expect an average number of requests per connection. This varies with different clients. If the average user will be hitting 5 requests per connection and the average response time that you find valid is 2 seconds that means that 10 seconds out of a minute 1 user will be on requests, meaning only 1/6 of the time it will be hitting the server. This also means that if you have 6 users hitting the server with simultaneously, you are likely to have 36 users in simulation, even though your concurrency level is only 6.

CLIENT – SERVER ARCHITECTURE:



The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.



In general, a service is an abstraction of computer resources. Clients and servers exchange messages in a request–response messaging pattern: The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. The language and rules of communication are defined in a communications protocol. All client-server protocols operate in the application layer. The application-layer protocol defines the basic patterns of the dialogue. To formalize the data exchange even further, the server may implement an API (such as a web service). The API is an abstraction layer for such resources as databases and custom software. By restricting communication to a specific content format, it facilitates parsing. By abstracting access, it facilitates cross-platform data exchange.

A server may receive requests from many different clients in a very short period of time. Because the computer can perform a limited number of tasks at any moment, it relies on a scheduling system to prioritize incoming requests from clients in order to accommodate them all in turn. To prevent abuse and maximize uptime, the server's software limits how a client can use the server's resources. Even so, a server is not immune from abuse. A denial of service attack exploits a server's obligation to process requests by bombarding it with requests incessantly. This inhibits the server's ability to respond to legitimate requests.

THREAD MANAGEMENT AND LOCKS:

Threads:

In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler. In most cases a thread is a component of a process. Multiple threads can exist within one process, executing concurrently and share resources such as memory, while different processes do not share these resources. Multithreading is a widespread programming and execution model that allows multiple threads to exist within the context of one process. These threads share the process's resources, but are able to execute independently.Multithreaded applications have the following advantages:

Responsiveness: multithreading can allow an application to remain responsive to input. By moving such long-running tasks to a worker thread that runs concurrently with the main execution thread, it is possible for the application to remain responsive to user input while executing tasks in the background.

Parallelization: applications looking to use multicore or multi-CPU systems can use multithreading to split data and tasks into parallel subtasks and let the underlying architecture manage how the threads run, either concurrently on one core or in parallel on multiple cores.

Multithreading has the following drawbacks:

Synchronization: since threads share the same address space, the programmer must be careful to avoid race conditions and other non-intuitive behaviours. Threads may also require mutually exclusive operations (often implemented using semaphores) in order to prevent common data from being simultaneously modified or read while in the process of being modified. Careless use of such primitives can lead to deadlocks.

Locks:

In computer science, a lock or mutex (from mutual exclusion) is a synchronization mechanism for enforcing limits on access to a resource in an environment where there are many threads of execution. A lock is designed to enforce a mutual exclusion concurrency control policy. The simplest type of lock is a binary semaphore. It provides exclusive access to the locked data.While a binary semaphore may be used as a lock, in that only the process that locked the lock is supposed to unlock it. Since only the process that locked the lock is supposed to unlock it, a lock may store the id of process that locked it and verify the same process unlocks it.

Locks may also provide deletion safety, where the process holding the lockcannot be accidentally deleted. Alternately, if the process holding the lock is deleted (perhaps due to an unrecoverable error), the lock can be automatically released.A lock may be recursive: a process is allowed to lock it multiple times without causing a deadlock.

In Python we have the Threading Module to implement Threads and Locks –
The threading module exposes all the methods of the thread module and provides some additional methods:

- threading.activeCount (): Returns the number of thread objects that are active.
- threading.currentThread (): Returns the number of thread objects in the caller's thread control.
- threading.enumerate (): Returns a list of all thread objects that are currently active.

The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads.

- A new lock is created by calling the Lock () method, which returns the new lock.

- The acquire (blocking) method of the new lock object is used to force threads to run synchronously. The optional blocking parameter enables you to control whether the thread waits to acquire the lock.
- If blocking is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired. If blocking is set to 1, the thread blocks and wait for the lock to be released.
- The release () method of the new lock object is used to release the lock when it is no longer required.

PACKET SNIFFING:

A packet analyser (also known as a network analyser, protocol analyser or packet sniffer—or, for particular types of networks, an Ethernet sniffer or wireless sniffer) is a computer program or piece of computer hardware that can intercept and log traffic that passes over a digital network or part of a network. As data streams flow across the network, the sniffer captures each packet and, if needed, decodes the packet's raw data, showing the values of various fields in the packet, and analyses its content according to the appropriate RFC or other specifications. Packet capture is the process of intercepting and logging traffic.

When traffic is captured, either the entire contents of packets are recorded, or the headers are recorded without recording the total content of the packet. This can reduce storage requirements, and avoid legal problems, yet provide sufficient information to diagnose problems. Captured information is decoded from raw digital form into a human-readable format that lets users easily review exchanged information. Protocol analysers vary in their abilities to display data in multiple views, automatically detect errors, determine root causes of errors, generate timing diagrams, reconstruct TCP and UDP data streams, etc.

Library Used: PCAPY

Pcapy is a Python extension module that enables software written in Python to access the routines from the pcap packet capture library. From libpcap's documentation: "Libpcap is a system-independent interface for user-level packet capture. Libpcap provides a portable framework for low-level network monitoring. Applications include network statistics collection, security monitoring, network debugging, etc."

What makes pcapy different from the others?

- Works with Python threads.
- Works both in UNIX with libpcap and Windows with WinPcap.
- Provides a simpler Object Oriented API.

Getting Pcapy
Current and past releases are available from https://github.com/CoreSecurity/pcapy

ATTACKS:

I. Denial Of Service (DDos) Attack:

In computing, a denial-of-service (DoS) attack is an attempt to make a machine or network resource unavailable to its intended users, such as to temporarily or indefinitely interrupt or suspend services of a host connected to the Internet. A distributed denial-of-service (DDoS) is where the attack source is more than one, often thousands of, unique IP addresses. It is analogous to a group of people crowding the entry door or gate to a shop or business, and not letting legitimate parties enter into the shop or business, disrupting normal operations. Criminal perpetrators of DoS attacks often target sites or services hosted on high-profile web servers such as banks, credit card payment gateways; but motives of revenge, blackmail or activism can be behind other attacks.

A denial-of-service attack is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service. There are two general forms of DoS attacks: those that crash services and those that flood services.The most serious attacks are distributed and in many or most cases involve forging of IP sender addresses (IP address spoofing) so that the location of the attacking machines cannot easily be identified, nor can filtering be done based on the source address.

A distributed denial-of-service (DDoS) attack occurs when multiple systems flood the bandwidth or resources of a targeted system, usually one or more web servers. Such an attack is often the result of multiple compromised systems flooding the targeted system with traffic. When a server is overloaded with connections, new connections can no longer be accepted. The major advantages to an attacker of using a distributed denial-of-service attack are that multiple machines can generate more attack traffic than one machine, multiple attack machines are harder to turn off than one attack machine, and that the behaviour of each attack machine can be stealthier, making it harder to track and shut down. This after all will end up completely crashing a website for periods of time.

II. <u>SYN Flood:</u>

A SYN flood is a form of denial-of-service attack in which an attacker sends a succession of SYN requests to a target's system in an attempt to consume enough server resources to make the system unresponsive to legitimate traffic.

Normally when a client attempts to start a TCP connection to a server, the client and server exchange a series of messages which normally runs like this: The client requests a connection by sending a SYN (synchronize) message to the server. The server acknowledges this request by sending SYN-ACK back to the client. The client responds with an ACK, and the connection is established. This is called the TCP three-way handshake, and is the foundation for every connection established using the TCP protocol.

A SYN flood attack works by not responding to the server with the expected ACK code. The malicious client can either simply not send the expected ACK, or by spoofing the source IP address in the SYN, causing the server to send the SYN-ACK to a falsified IP address - which will not send an ACK because it "knows" that it never sent a SYN.

The server will wait for the acknowledgement for some time, as simple network congestion could also be the cause of the missing ACK. However, in an attack, the half-open connections created by the malicious client bind resources on the server and may eventually exceed the resources available on the server. At that point, the server cannot connect to any clients, whether legitimate or otherwise. This effectively denies service to legitimate clients. Some systems may also malfunction or crash when other operating system functions are starved of resources in this way.

REQUIREMENT ANALYSIS

I. *SOFTWARE (including Libraries and Packages):*

- MAC OS X – El Capitan
- WINDOWS 10 Home Single Language
- WINDOWS 8.1 Home
- Python 2.7
- Python WinPcap – 4.1.3
- PCAPY Library – Pcapy -0.10.10
- SCAPY Library – Scapy – 0.0.11
- NUMPY Library – Numpy – 1.11.10
- MATPLOTLIB Library – MatplotLib – 1.2.0
- Command Line Interface
- OS X Terminal
- Text Editors – Sublime Text, Notepad++

II. *HARDWARE:*

- DELL Inspiron 15R –2.0 GHz Intel Core i7
- MacBook Air – 1.6 GHz Intel Core i5
- Ports - 9002
- Sniffing Device for Windows - \Device\NPF_{D095B278-15B0-44C3-81A4-07B227B4BF2E} for WIFI
- Sniffing Device for MAC – “en0” – for WIFI
- TCP/IP Protocol

III. *FUNCTIONAL REQUIREMENTS:*

- ***INPUTS:***
  - Data Packet Size.
  - Number of Requests by the Client.
  - Yes/No choice to start the test.
  - Choice for Re – Testing.
  - Defined Load File

- ***OUTPUTS:***
  - GUI Displaying Successful Connection.
  - GUI Displaying Performance Parameters – Response Time, Throughput.
  - Network Congestion Graph
  - Packet Sniffer Window sniffing packets transmitting over WIFI
  - Trace – Route Window showing number of HOPS between Server and Client.
  - NewFile.txt created with TCP Header details.
  - Server and Client Terminal showing packets in Transmission.
  - Retransmissions of packets dropped also displayed.

- ***DATA MANIPULATION AND PROCESSING:***
  - Once the server gets the packet – size and the number of requests from the user, it reads the load file as bytes and transmits the read bytes according to the packet size as specified by the user.
  - The client receives this file, also according to the defined packet size.
  - TCP/IP protocol ensures that any packet in transmission that is dropped is retransmitted, for reliable data delivery.
  - This occurs as many times as the client has requested.
  - After transmission, the performance parameters are calculated.

$$Response\ Time = End\ Time - Start\ Time$$

$$Throughput = \frac{Number\ Of\ Bytes\ Exchanged}{Response\ Time}$$

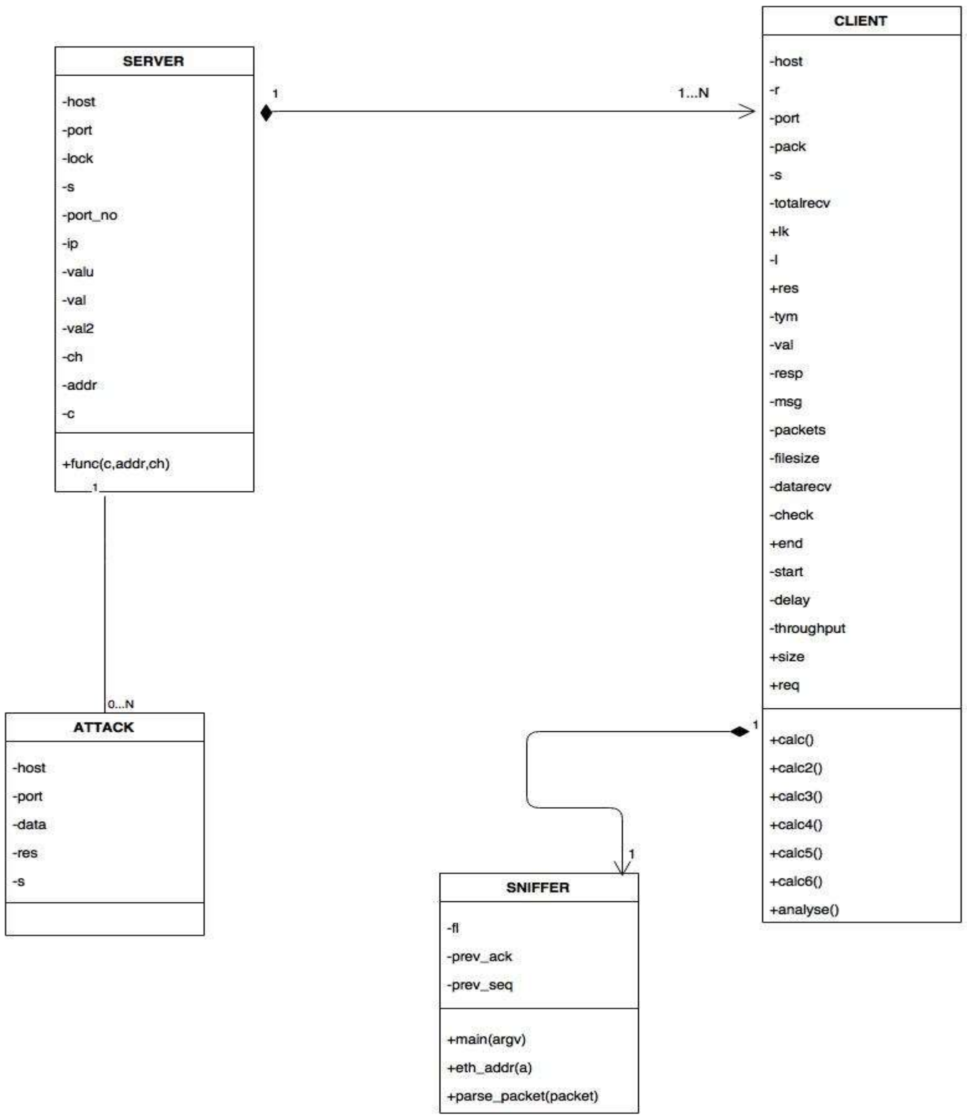
- Network Congestion Graph is plotted –
  Number of Bytes Received v/s Time Instant
- Client Management is implemented by managing Server Memory.
- The Server has a pre – defined memory under its process.
- Each time a client connects to it, it requests a certain portion of the server's memory.
- The server checks its memory resources and if it has the requested memory available, it allocates that memory to the client. If not, it rejects the client connection.
- In the other scenario, once a client request has been processed completely, the previously allocated memory will be freed and made available for new connections.
- The server performance is also tested under extreme conditions when the server has high load and is servicing an extraordinarily large number of clients.
- This can be verified by simulating SYN Attacks on the server by uploading a file multiple times on the server, through multiple connections, to keep it busy and not facilitating proper transmission to the user.
- Under such attacks, the server, will not have enough free memory as it will be kept occupied by false connections. So it will deny service to valid clients – DOS Attack.

IV. NON FUNCTIONAL REQUIREMENTS:

- In real time systems, the computer RAM is the memory constraint. But in our system, since it is just a simulation of the actual Load Testing Tool, hence we have limited the server memory according to the file being transmitted.
- In our tool, the load is a fixed file, which is defined by us. In real time systems, the load may be kept variable.

- The Packet Sniffer also has its own constraint. Because we have implemented our sniffer on WIFI Device, there must be no other incoming transmissions on that device other than our own file transfer.
- The system is required to be efficient in memory allocation and freeing, to allow proper client management. It considers the fact that a client cannot be served if there is insufficient memory. It should also take into account that if there is enough memory no client should be rejected or refused the connection.

V. *UML DIAGRAM:*



DETAILED DESIGN

- Server and Client are set up initially.
- Using Socket Programming in Python, client-server connection was established on same system, using TCP protocol.
- Connection was verified by simple messaging on both sides.
- Once the server gets the packet – size and the number of requests from the user, it reads the load file as bytes and transmits the read bytes according to the packet size as specified by the user.
- Data Exchange was implemented by reading and writing of different file formats in binary mode.
- Data Received on client side was verified.
- Explicit Handshaking was implemented wherein every data sent was acknowledged by both sides.
- The client receives this file, also according to the defined packet size.
- TCP/IP protocol ensures that any packet in transmission that is dropped is retransmitted, for reliable data delivery.
- This occurs as many times as the client has requested.
- After transmission, the performance parameters are calculated.
- Analysis of Performance parameters was done by computation of response time, throughput, total data exchanged (in terms of bytes).
- Calculations:

$$Response\ Time = End\ Time - Start\ Time$$

$$Throughput = \frac{Number\ Of\ Bytes\ Exchanged}{Response\ Time}$$

- Network Congestion Graph is plotted –

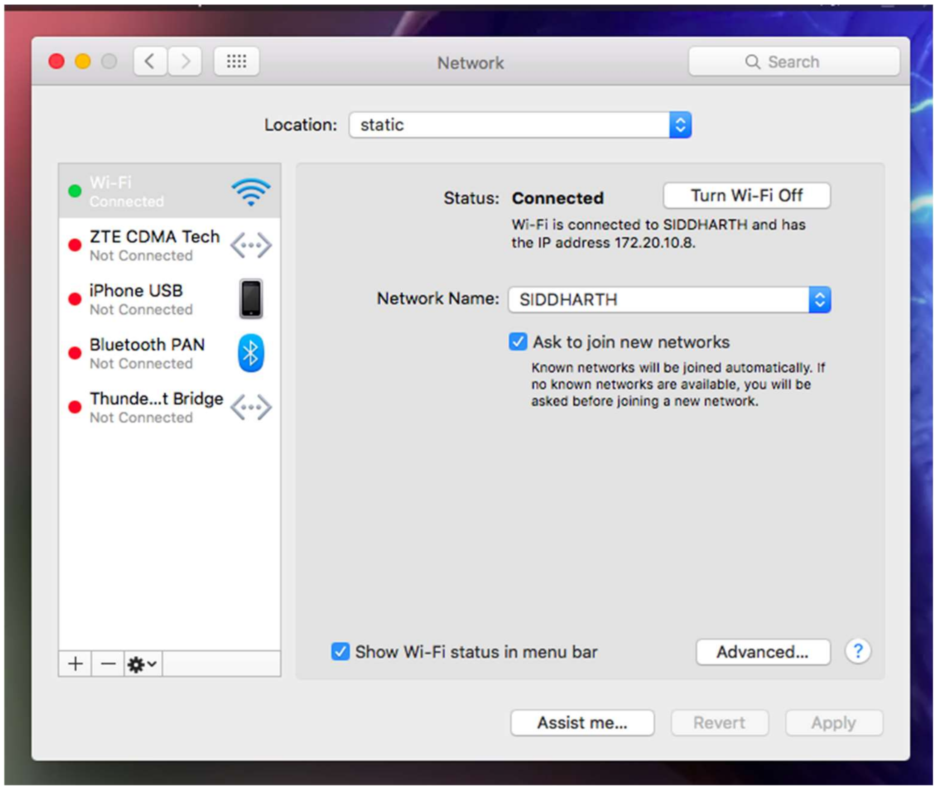
  Number of Bytes Received v/s Time Instant
- Side by Side the TCP Header of all packets is accessed and saved in memory – including sequence number, acknowledgement number etc.
- Client Management is implemented by managing Server Memory.
- The Server has a pre – defined memory under its process.

- Each time a client connects to it, it requests a certain portion of the server's memory.
- The server checks its memory resources and if it has the requested memory available, it allocates that memory to the client. If not, it rejects the client connection.
- In the other scenario, once a client request has been processed completely, the previously allocated memory will be freed and made available for new connections.
- The server performance is also tested under extreme conditions when the server has high load and is servicing an extraordinarily large number of clients.
- This can be verified by simulating SYN Attacks on the server by uploading a file multiple times on the server, through multiple connections, to keep it busy and not facilitating proper transmission to the user.
- Under such attacks, the server, will not have enough free memory as it will be kept occupied by false connections. So it will deny service to valid clients – DOS Attack.

IMPLEMENTATION

I. *SCREENSHOTS:*

- IP Address is detected.


Location: static

Status: Connected

Wi-Fi is connected to SIDDHARTH and has the IP address 172.20.10.8.

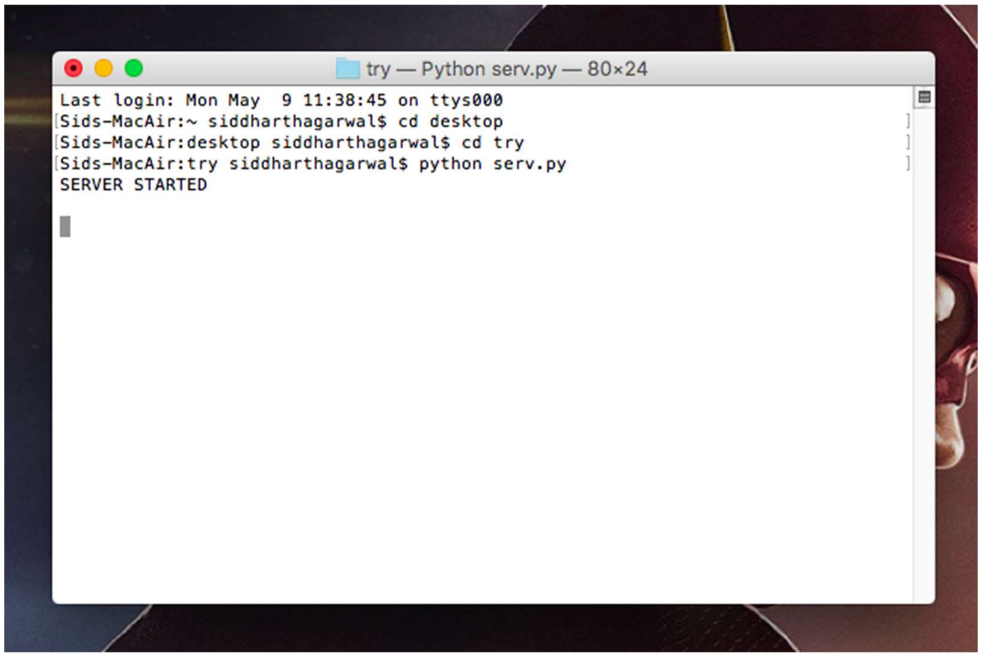
Network Name: SIDDHARTH

✓ Ask to join new networks

Known networks will be joined automatically. If no known networks are available, you will be asked before joining a new network.

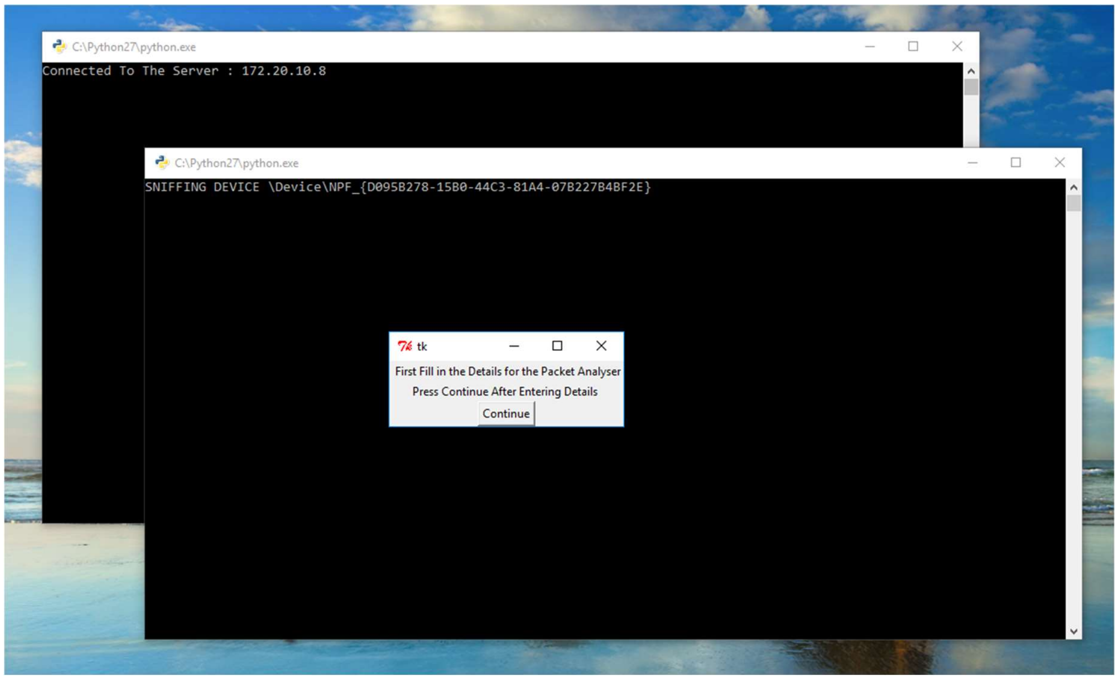
✓ Show Wi-Fi status in menu bar

- Server is started.



```
try — Python serv.py — 80×24
Last login: Mon May  9 11:38:45 on ttys000
Sids-MacAir:~ siddharthagarwal$ cd desktop
Sids-MacAir:desktop siddharthagarwal$ cd try
Sids-MacAir:try siddharthagarwal$ python serv.py
SERVER STARTED
```

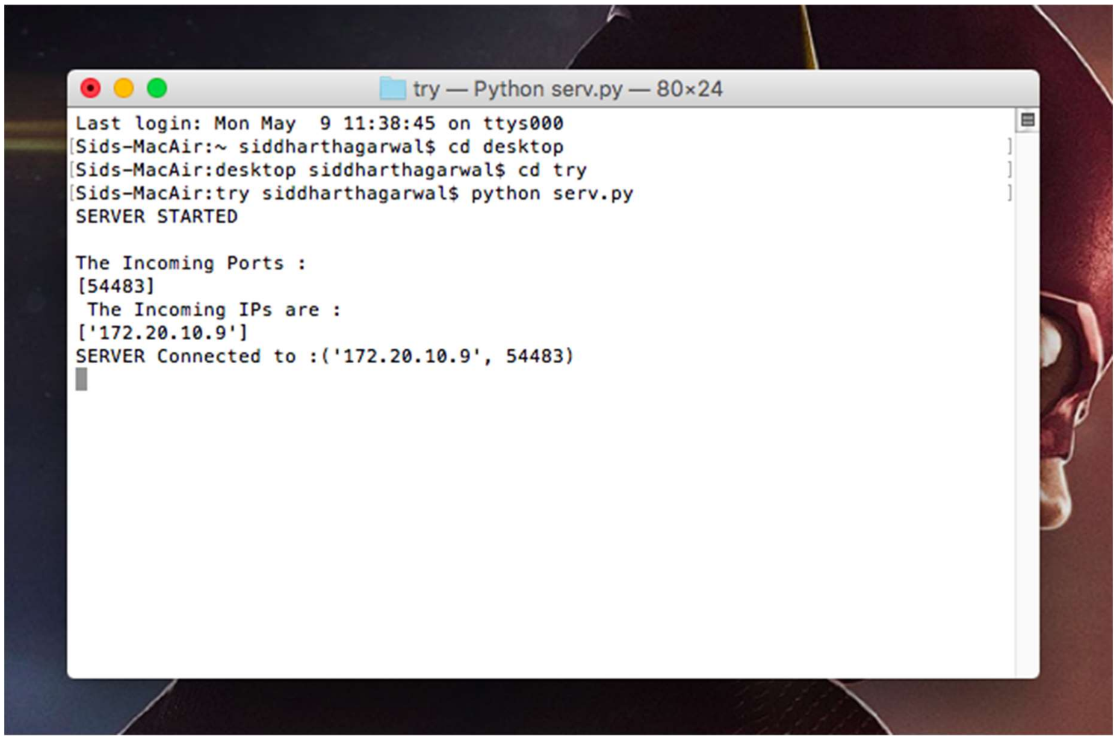
- Client sends request for connection. Server accepts if it has enough memory.



```
C:\Python27\python.exe
Connected To The Server : 172.20.10.8
```

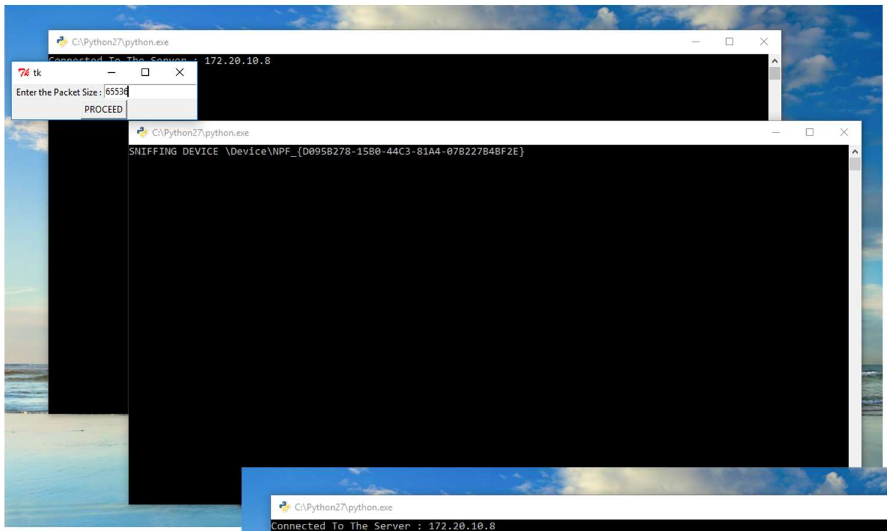
```
C:\Python27\python.exe
SNIFFING DEVICE \Device\NPF_{D095B278-15B0-44C3-81A4-07B227B4BF2E}
```

```
tk
First Fill in the Details for the Packet Analyser
Press Continue After Entering Details
```



```
Last login: Mon May  9 11:38:45 on ttys000
Sids-MacAir:~ siddharthagarwal$ cd desktop
Sids-MacAir:desktop siddharthagarwal$ cd try
Sids-MacAir:try siddharthagarwal$ python serv.py
SERVER STARTED

The Incoming Ports :
[54483]
 The Incoming IPs are :
['172.20.10.9']
SERVER Connected to :('172.20.10.9', 54483)
```

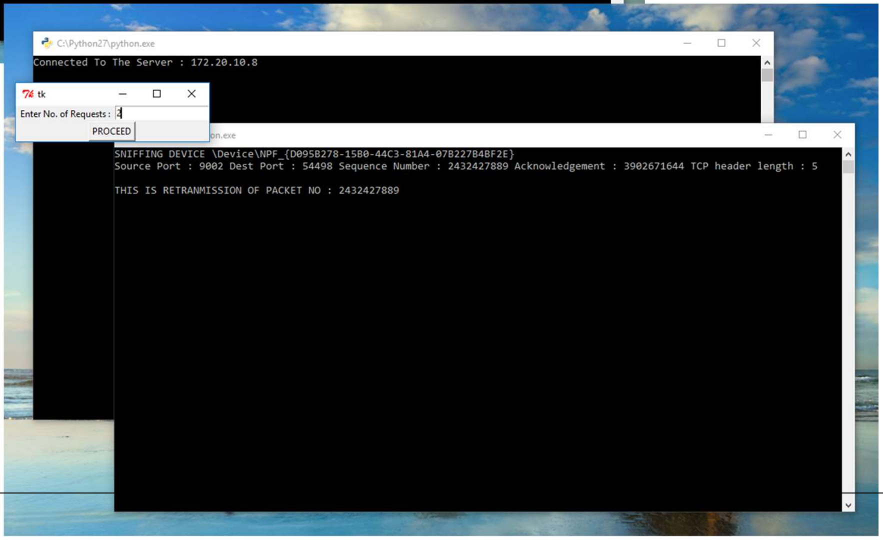
- Client enters the required details – size of packet, number of requests and choice for testing. Server acknowledged by handshaking that details have been received.



```
Connected To The Server : 172.20.10.8
```

Enter the Packet Size : 65536

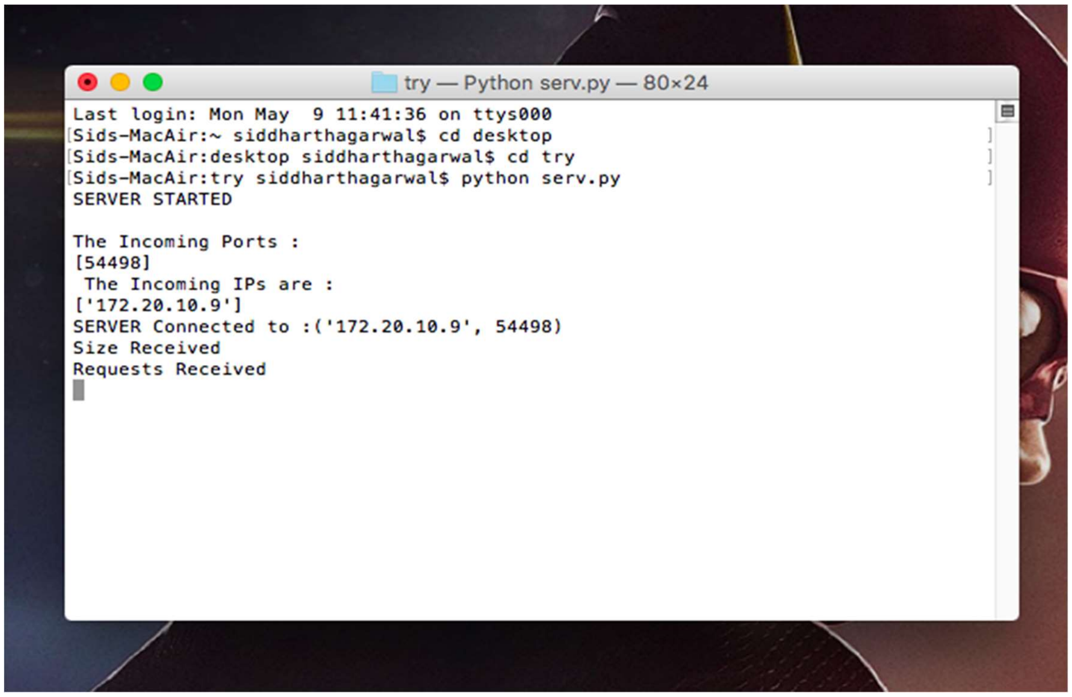
```
SNIFFING DEVICE \Device\NPF_{D095B278-15B0-44C3-81A4-07B227B4BF2E}
```



```
Connected To The Server : 172.20.10.8
```

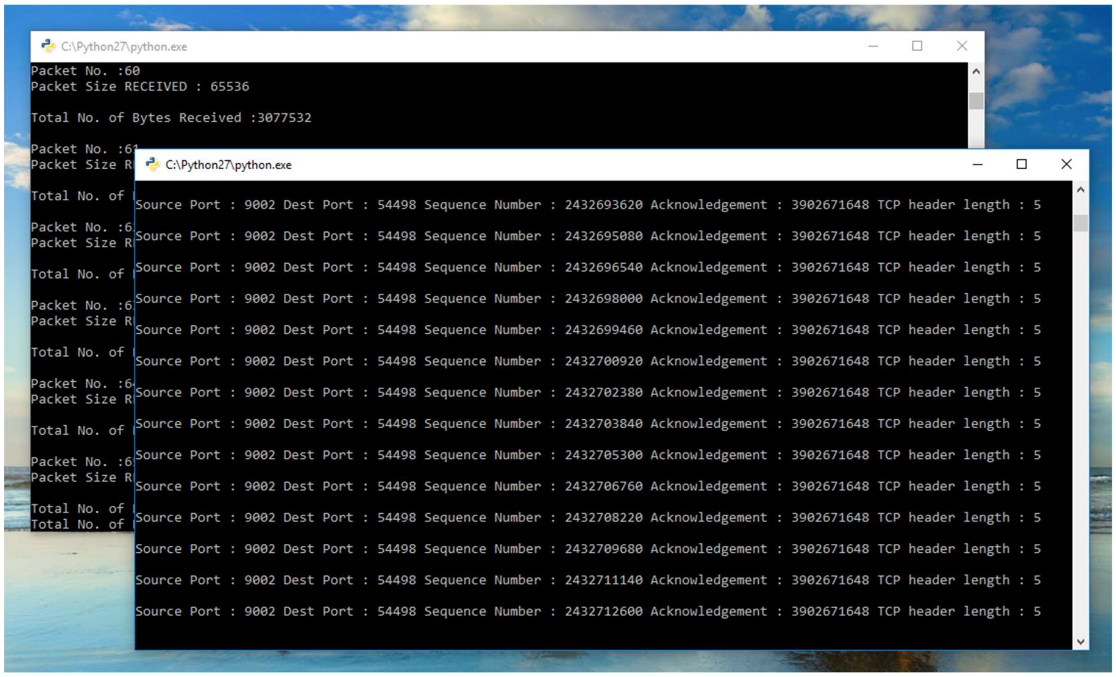
Enter No. of Requests : 2

```
SNIFFING DEVICE \Device\NPF_{D095B278-15B0-44C3-81A4-07B227B4BF2E}
Source Port : 9002 Dest Port : 54498 Sequence Number : 2432427889 Acknowledgement : 3902671644 TCP header length : 5

THIS IS RETRANMISSION OF PACKET NO : 2432427889
```



```
Last login: Mon May  9 11:41:36 on ttys000
Sids-MacAir:~ siddharthagarwal$ cd desktop
Sids-MacAir:desktop siddharthagarwal$ cd try
Sids-MacAir:try siddharthagarwal$ python serv.py
SERVER STARTED

The Incoming Ports :
[54498]
 The Incoming IPs are :
['172.20.10.9']
SERVER Connected to :('172.20.10.9', 54498)
Size Received
Requests Received
```

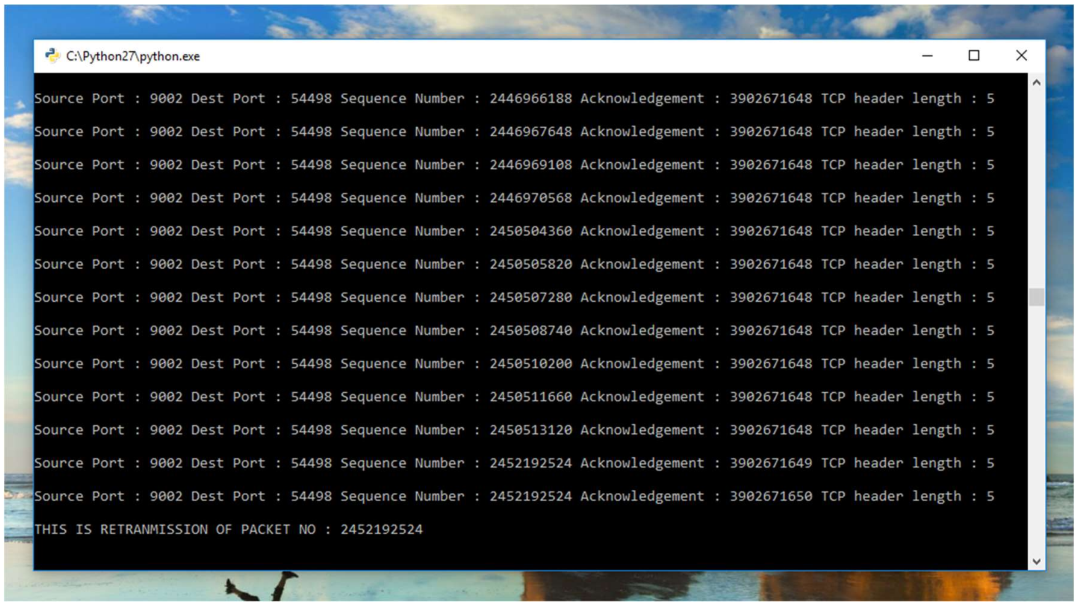
- Transmission of load begins. It is displayed on both sides.



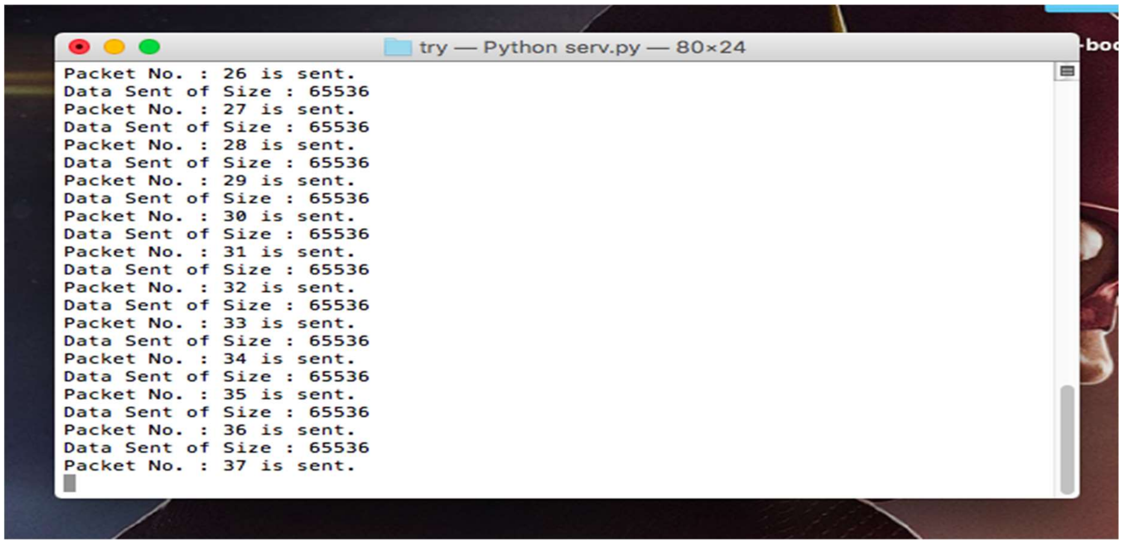
```
Packet No. :60
Packet Size RECEIVED : 65536

Total No. of Bytes Received :3077532

Packet No. :61
Packet Size R

Total No. of

Packet No. :6
Packet Size R

Total No. of

Packet No. :6
Packet Size R

Total No. of

Packet No. :6
Packet Size R

Total No. of

Packet No. :6
Packet Size R

Total No. of
Total No. of
```

```
Source Port : 9002 Dest Port : 54498 Sequence Number : 2432693620 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432695080 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432696540 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432698000 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432699460 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432700920 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432702380 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432703840 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432705300 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432706760 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432708220 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432709680 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432711140 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2432712600 Acknowledgement : 3902671648 TCP header length : 5
```

- Packet sniffer accesses TCP Header of all packets being transmitted on WIFI Device.

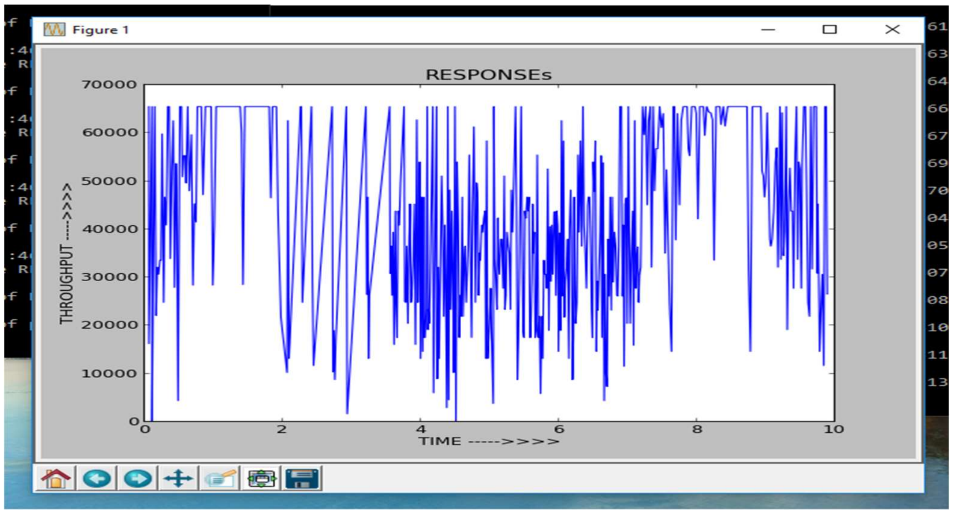


```
Source Port : 9002 Dest Port : 54498 Sequence Number : 2446966188 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2446967648 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2446969108 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2446970568 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2450504360 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2450505820 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2450507280 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2450508740 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2450510200 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2450511660 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2450513120 Acknowledgement : 3902671648 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2452192524 Acknowledgement : 3902671649 TCP header length : 5

Source Port : 9002 Dest Port : 54498 Sequence Number : 2452192524 Acknowledgement : 3902671650 TCP header length : 5

THIS IS RETRANMISSION OF PACKET NO : 2452192524
```

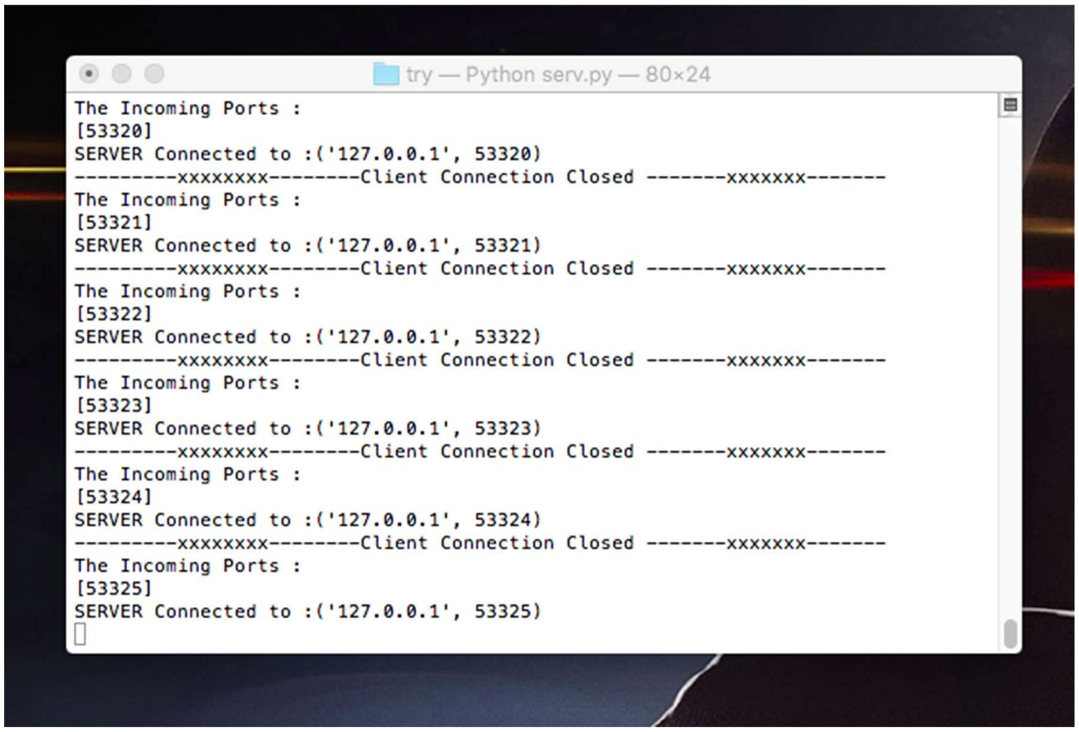


```
Packet No. : 26 is sent.
Data Sent of Size : 65536
Packet No. : 27 is sent.
Data Sent of Size : 65536
Packet No. : 28 is sent.
Data Sent of Size : 65536
Packet No. : 29 is sent.
Data Sent of Size : 65536
Packet No. : 30 is sent.
Data Sent of Size : 65536
Packet No. : 31 is sent.
Data Sent of Size : 65536
Packet No. : 32 is sent.
Data Sent of Size : 65536
Packet No. : 33 is sent.
Data Sent of Size : 65536
Packet No. : 34 is sent.
Data Sent of Size : 65536
Packet No. : 35 is sent.
Data Sent of Size : 65536
Packet No. : 36 is sent.
Data Sent of Size : 65536
Packet No. : 37 is sent.
```

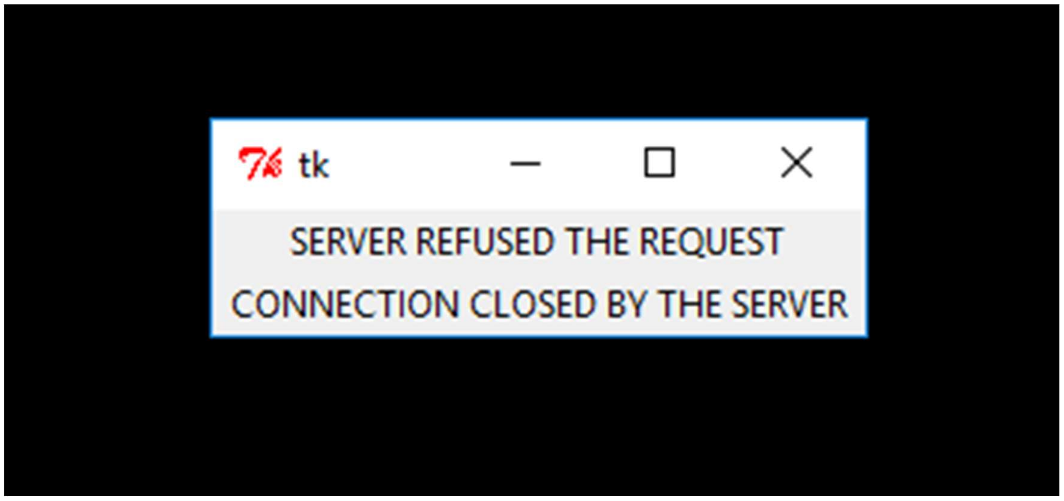
- Performance parameters are analysed and graph is plotted.



- Under attacks, server receives high number of connections, which causes denial of service to actual clients.



```
The Incoming Ports :
[53320]
SERVER Connected to :('127.0.0.1', 53320)
---------xxxxxxxx--------Client Connection Closed -------xxxxxxx-------
The Incoming Ports :
[53321]
SERVER Connected to :('127.0.0.1', 53321)
---------xxxxxxxx--------Client Connection Closed -------xxxxxxx-------
The Incoming Ports :
[53322]
SERVER Connected to :('127.0.0.1', 53322)
---------xxxxxxxx--------Client Connection Closed -------xxxxxxx-------
The Incoming Ports :
[53323]
SERVER Connected to :('127.0.0.1', 53323)
---------xxxxxxxx--------Client Connection Closed -------xxxxxxx-------
The Incoming Ports :
[53324]
SERVER Connected to :('127.0.0.1', 53324)
---------xxxxxxxx--------Client Connection Closed -------xxxxxxx-------
The Incoming Ports :
[53325]
SERVER Connected to :('127.0.0.1', 53325)
```



```
SERVER REFUSED THE REQUEST
CONNECTION CLOSED BY THE SERVER
```

TESTING REPORTS

TEST CASE	INPUT #1	INPUT #2	EXPECTED OUTPUT	ACTUAL OUTCOME
1.CONNECTING TO THE SERVER	IP ADDRESS (XXX.XXX.XXX.XXX)	PORT NUMBER (9002)	CONNECTION SUCCESSFUL	CONNECTION SUCCESSFUL
2.ENTER THE PACKET SIZE	65536	-	SIZE RECEIVED	SIZE RECEIVED
	XXXXX	-	SIZE RECEIVED	ENTER VALID SIZE
	0	-	IN-CORRECT VALUE	SIZE RECEIVED
4.ENTER THE NUMBER OF REQUESTS	2	-	REQUEST RECEIVED	REQUEST RECEIVED
	XXXX	-	REQUEST RECEIVED	ENTER VALID NUMBER OF REQUESTS
	0	-	REQUESTS CAN'T BE ZERO	REQUEST RECEIVED
5.ENTER YES TO START THE TEST	"YES"	-	TEST STARTED	TEST STARTED
	"NO"	-	REFUSED TESTING	REFUSED TESTING

	XXXX	-	INVALID INPUT	INVALID INPUT
6.ENTER THE PASSWORD FOR SNIFFER	PASSWORD	-	SNIFFING DEVICE	SNIFFING DEVICE
	ELSE	-	WRONG PASSWORD	WRONG PASSWORD
7.ENTER CHOICE TO RE-TEST	0	-	CONNECTION CLOSED	CONNECTION CLOSED
	1	-	ENTER THE PACKET SIZE	ENTER THE PACKET SIZE
	2	-	INVALID: RE-ENTER VALUE	INVALID: RE-ENTER VALUE
8.IF MEMORY FULL : ENTER SIZE AND REQ	65536	3	CLIENT REFUSED	CLIENT REFUSED
9.UNDER ATTACKS (DOS)	65536	2	SUCCESSFUL	CLIENT REFUSED
10.DOWNLOADING A FILE NOT PRESENT	"YES"	-	SUCCESSFUL	NO SUCH FILE EXISTS
11.WRITING SNIFFER OUTPUT IN A FILE	SNIFFER PASSWORD	-	FILE CREATED	FILE CREATED
	IN-CORRECT PASSWORD	-	FILE NOT CREATED	FILE NOT CREATED

GANTT CHART

LIST OF ACTIVITIES:

A – Searching the prospective topic of the project and Researching on the topic.

B – Consulting the faculty in charge.

C – Topic Finalised.

D – Objective and Scope of the project defined.

E – Language and IDE to be used decided.

F – Synopsis Created and Evaluated

G – Project Implementation Started.

H – Basic Client- Server Program Implemented.

I – Thread Management.

J – File Sharing.

K – GUI and Graph designed and implemented.

L – Connection Established between 2 Systems.

M – Computation of Testing Parameters.

N – Mid Phase Report Submission and Evaluation.

O – Final Phase Implementation Started

P – TCP/IP Header Access studied and Sniffer Implemented.

Q – SYN Flood Attack studied and implemented.

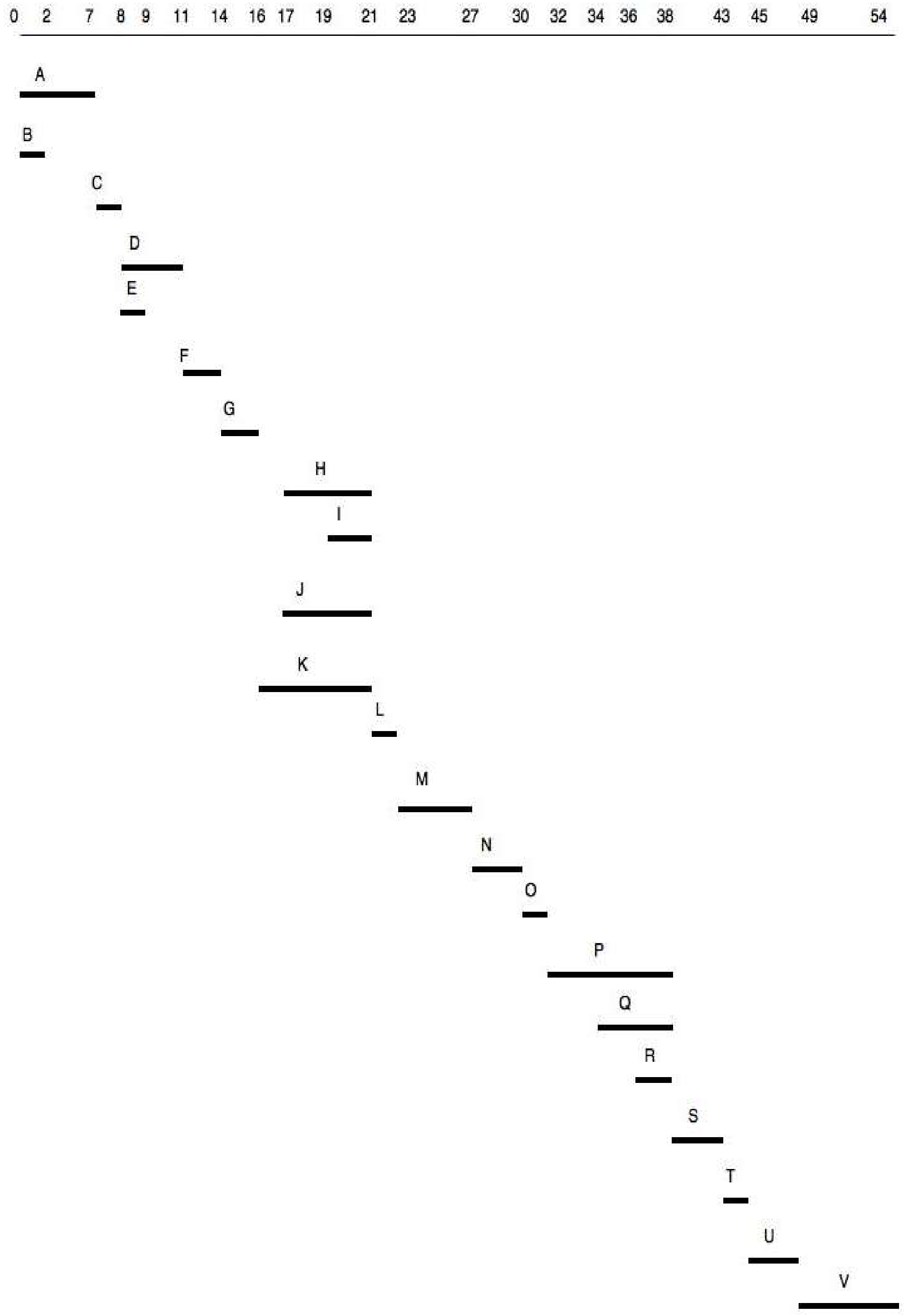
R – Connection Management.

S – Platform Independent Server Testing System created.

T – Testing on multiple systems and platforms.

U – Integration with Mid-Phase and Complete System Implementation.

V – Final Phase Report Submission and Evaluation.


0
2
7
8
9
11
14
16
17
19
21
23
27
30
32
34
36
38
43
45
49
54
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V

FUTURE SCOPE

Our project is a simulation of the actual real time analysis going on in multinational companies like Flipkart and Amazon, all of whom have to test their servers constantly to prevent them from crashing and providing their using the best experience they can. In offer days when they have to serve extra – ordinarily high number of clients, they constantly check their server through such tools. At present, each of these companies have their own server testing tools and none of them have outed their tools for fear of competition.

Considering this, we have implemented such a tool that can be expanded on large scale which has the actual memory as the limiting factor and serving clients with any kind of load. On a large scale, this would be able to test servers, with the number of clients that can be connected, as the load keep on increasing - which would in turn increase the effective load on the server (via downloading or uploading content). This tool would test the capacity of the server as to How Many clients it can connect, before slowing down and finally crashing. Also it can be used to predict unforeseen traffic, and the behaviour of the server in such scenarios by simulating DOS and SYN Attacks to be able to prep – up for such cases.

It can test any system by putting demand on thesoftware system or computing device and measuring its response. It can be used in the future to determine a system's behavior under both normal and anticipated peak load conditions, to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation. When load placed on the system is raised beyond normal usage patterns to test the system's response at unusually high or peak loads. The load is usually so great that error conditions are the expected result. These results can be analyzed to improve the overall performance of the company.

REFERENCES

[1]. http://jmeter.apache.org/

[2]. matplotlib-1.5.0-cp35-none-win_amd64.whl

[3]. https://en.wikipedia.org/wiki/Load_testing

[4]. http://www.pylot.org/

[5]. http://jcrawler.sourceforge.net/

[6]. http://istqbexamcertification.com/what-is-load-testing-in-software/

[7]. http://www.tutorialspoint.com/python/python_networking.htm

[8]. https://www.python.org/getit/windows/

[9]. http://askldjd.com/2014/01/15/a-reasonably-fast-python-ip-sniffer/

[10]. http://bt3gl.github.io/black-hat-python-building-a-udp-scanner.html

[11]. https://in.pycon.org/cfp/pycon-india-2015/proposals/make-your-own-packetsniffer~enZle/

[12]. http://www.secdev.org/projects/scapy/doc/usage.html

[13]. https://developer.apple.com/legacy/library/documentation/Darwin/Reference/ManPages/man8/traceroute.8.html

[14]. http://www.catonmat.net/blog/tcp-traceroute/

[15]. http://security.stackexchange.com/questions/39178/how-does-traceroute-over-tcp-work-what-are-the-risks-and-how-can-it-be-mitig

[16]. https://support.apple.com/en-in/HT202013

[17]. http://www.binarytides.com/tcpdump-tutorial-sniffing-analysing-packets/

[18]. http://www.ast.uct.ac.za/~sarblyth/pythonGuide/PythonPlottingBeginnersGuide.pdf

[19]. http://www.slashroot.in/how-does-traceroute-work-and-examples-using-traceroute-command

[20]. http://stackoverflow.com/questions/8196886/how-to-generate-tcp-ip-and-udp-packets-in-python

[21]. https://isc.sans.edu/forums/diary/TCP+Fuzzing+with+Scapy/14080/

[22]. http://juhalaaksonen.com/blog/2013/12/11/installing-scapy-for-mac-os-x/

[23]. http://docs.scipy.org/doc/numpy-1.10.1/user/install.html#id4

[24]. http://penandpants.com/2012/03/01/install-python-2/

[25]. https://dnaeon.github.io/traceroute-in-python/

[26]. http://www.mediacollege.com/internet/troubleshooter/traceroute.html

[27]. http://pylibpcap.sourceforge.net/

[28]. http://www.tcpdump.org/manpages/pcap.3pcap.html

[29]. http://computingforbeginners.blogspot.in/2014/07/client-server-program-in-c-to-simulate.html

[30]. https://www.net.princeton.edu/enetAddress.howto.html

[31]. https://github.com/CoreSecurity/pcapy/wiki/Compiling-Pcapy-on-Windows-Guide

[32]. https://technet.microsoft.com/en-us/library/cc725935(v=ws.10)
[33]. https://technet.microsoft.com/en-in/library/bb490921.aspx
[34]. https://technet.microsoft.com/en-us/library/bb490994.aspx
[35]. https://technet.microsoft.com/en-in/library/bb490913.aspx
[36]. http://www.pygame.org/docs/tut/MoveIt.html
[37]. http://www.pygame.org/docs/
[38]. https://github.com/zlorb/scapy/blob/master/README.md
[39]. http://www.apoorvakumar.tk/2015/06/30/installing-lxml-in-windows-for-python-3-4/
[40]. http://www.secdev.org/projects/scapy/doc/installation.html#windows
[41]. http://www.lfd.uci.edu/~gohlke/pythonlibs/
[42]. https://github.com/dugsong/pypcap/blob/master/INSTALL