TRACEROUTE

The main aim of this project is to develop a network tool which can be used to show the route taken by packets across an IP network. This tool would inject an IP packet into the available network interface and wait for the reply from the intermediate routers and the destination IP, giving information about the network setup and the firewall configuration on each intermediate node.

UDP datagram are used as probe packets and waiting for an ICMP-ECHO-REPLY. This UDP datagram passes through every intermediate router which in turn replies with an ICMP message. From the received message the address of the intermediate routers is retrieved. Hostnames are mainly displayed however in certain exceptions only IP addresses is displayed. This algorithm does not suffer from a changing path and allows the response to the Return Packet to be traced.

Traceroute is a program that can be used to send packets of information to remote computers for the purpose of retrieving information useful for testing Internet connection. In general, a traceroute tracks the path that a packet takes from your computer to a destination address. A traceroute also shows how many times your packets are being rebroadcast by other servers until it gets to the final destination.

PURPOSE

The main aim of this project is to develop a network tool which can be used to show the route taken by packets across an IP network. This tool is available on practically all Unix-like operating systems.

This tool would inject an IP packet into the available network interface and wait for the reply from the intermediate routers and the destination IP, giving information about the network setup and the firewall configuration on each intermediate node.

The main challenge will be to make the intermediate router forward probe packets or reply with an error message. If an intermediate router is configured for not forwarding a particular type of probe packets the destination system may never be traced on that interface using the same type of probe packets.

SCOPE

Any connection over the Internet actually depends on two routes: the route from the local system to the server and the route from that server back to the local system. These routes are completely asymmetrical. So a traceroute from local system to a server gives information only about one direction even though the reply packets may be taking a different route. So a problem reflected in a traceroute output may actually not lie with the obvious system in the route, it may rather be with some other system on the reverse route back from the system that looks, from the trace, to be the cause of the problem.

MOTIVATION

Many times during transmission of network packets, few packets are either dropped or lost during transmission. It is important to know where exactly the packets are being dropped. Only then it is possible to solve the problems. The traceroute is a diagnostic tool by which path and transit delays from packets can be measured. Traceroute tracks the path that a packet takes from one's computer to a destination address. A traceroute also shows how many times the packets are being rebroadcast by other servers until it gets to the final destination. It can show where there is a break in the network connection. This allows one to determine where exactly are the packets being dropped or lost. The traceroute actually is to identify any sort of problematic server which could cause an error. This is a very useful diagnostic tool and thus we were motivated to develop one.

LITERATURE SURVEY

In early days of Internet an IP option called the *source routing* could be used with the traceroute to see the reverse route. The idea is to specify what is called a *loose source route,* which specifies a system your packets should pass through before proceeding on to their destination. The ability to use loose source routing to see the reverse route could be pretty handy. Unfortunately, source routing has a great potential for abuse, and therefore most network administrators block all source-routed packets at their border routers. So, in practice, loose source routes aren't going to work.

So the complete information about the route will be available only if we a traceroute to the local system can be done from the server. The only hope likely of running a reverse traceroute is if the system to be traced from has a traceroute facility on their web site. Many systems and Usenet providers in particular, have a web page where a traceroute can be run from that system back to source. In combination with trace to their system, this can give the other half of the picture.

But, traceroute is a resource consuming operation and can usually slow ones server. Thus traceroute must be employed only if some sort of problem is being encountered. After the diagnose it is possible to track where exactly are the packets being lost and can design the solution to it accordingly.

DESIGN

The traceroute runs on a single system which is the source. The user provides the destination host name as command line parameter. The maximum hop value and the port value are by default taken as 30 and 33434 respectively. This is followed by the creation of separate sockets for UDP packets and ICMP messages. Also the socket options are set to obtain the same. The UDP socket is bound to the specified port number and the packet sending begins. The intermediate routers respond with ICMP messages. Thus the IP addresses are retrieved from the received messages. In order to obtain the host names from the IP address a reverse DNS lookup is performed. The IP addresses so retrieved are displayed until the packet reaches the destination. Finally, the socket connections are closed.

PROTOCOLS USED

First implementation of traceroute was done using ping packets i.e. ICMP echo request and ICMP echo reply packets. But modern implementations of traceroute is done using UDP datagram's by sending to some random high numbered port where nothing is likely listening. When that final system is reached, since nothing is answering on the port, an ICMP port unreachable error message is returned.

From the discussions till now it can be said that the entire traceroute implementation depends on the reply ICMP messages. But nowadays the trend is to block unnecessary ICMP traffic by some of the Internet routers, this is because an ICMP message reveals lot of information about the Network setup, for example if a probe-reply cycle is done on every port between 1 and 1024 it can easily know which all standard services are running on the remote machine. And thus the network administrator may configure the firewall such that it may forward the traceroute probe packets but not reply with an ICMP error message on time to live exceed, thus not revealing its IP to the traceroute tool.

The trend today is to use TCP probe packets instead of ping packets or UDP datagram's. Traceroute implementation using TCP is flexible and yields better results than UDP. Tcptraceroute can be implemented using SYN, FIN, SYN/ACK probe packets which may yield ACK, RST, FIN, SYN or ICMP as reply. If for example we use SYN packets for probing it will yield us ACK as reply if a service is running on destination port, hence port is open. Else it may result in an ICMP reply thus port closed. This is the concept of half open connection which is used by the legendary application "nmap".

This concept used by tcptraceroute is also called Firewalking. This is because even though the firewall may be configured for reducing ICMP traffic, the dependency on ICMP probe is reduced instead on default ports on which services may be actually listening. This kind of probing may also result in crashing the service process on the default port as it doesn't understand the data payload in the probe packet as it is just for debugging.

IMPLEMENTING DIFFERENT MODULES

In this section the implementation of different modules in the trace router is explained.

Creation of socket

Two types of sockets are created i.e. one for the UDP packets sent and the other for the ICMP messages received. Creation of sockets can be achieved by the following code snippet.

recv_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, icmp)
send_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, udp)
Using the UDP socket packets are sent to the specified destination and using the ICMP socket the ICMP messages are received as shown below.
recv_socket.bind(("", port))
send_socket.sendto("", (dest_name, port))

Obtaining IP addresses of intermediate routers
From the received ICMP messages the sender IP address is received which is the required address. Following the retrieval, a reverse DNS lookup is employed i.e. the host name is obtained as given below.
_, curr_addr = recv_socket.recvfrom(512)
curr_name = socket.gethostbyaddr(curr_addr)[0]

SUMMARY

Packets are injected into the network by overriding the Operating system code for the same purpose. A code written in python has been implemented to trace the route taken by the packets as it moves from its source to destination. The code has proven to be efficient due to its appreciable size resulting in good speed of execution. It has the functionality of not only providing the IP addresses of the intermediate routers but also their corresponding host names wherever reverse DNS is possible.

The advantage here is mainly being that the user needs to only input the destination host name which is easily available not the IP address. Since the host name is easily available it rules out the necessity to call the *dig* or the *host* utility to specify the destination IP address. Private IP addresses are displayed with a * symbol thus no intermediate IP addresses are omitted. Thus an accurate count of intermediate routers can be maintained. This application also supports other application by detecting the faulty routers i.e. loss of packets from the exact router can be traced.

LIMITATIONS

- This implementation is only compatible with IP version 4.
- The maximum number of hops by default is set to 30. Thus destinations that require more hops cannot be traced completely.
- As implementation is done using UDP packets in case of slow networks there may be a failure to gain reply.
- The ICMP protocol used may have been disabled by the firewall of routers.

FURTHER ENHANCEMENTS

- Implementing the IPv6 implementation.
- Implementing using TCP packets as probe packets and to provide the user with the option to choose which protocol to use for probe.
- Implementing a Graphical user interface i.e. Visual Traceroute

SCREENSHOTS

```
[root@localhost Documents]# ./traceroute.py www.rvce.edu.in
1       D-Link.Home (192.168.1.1)
2       117.192.192.1 (117.192.192.1)
3       218.248.160.198 (218.248.160.198)
4       218.248.255.74 (218.248.255.74)
5       59.163.206.177.static.chennai.vsnl.net.in (59.163.206.177)
6       ix-4-2.tcore2.CXR-Chennai.as6453.net (180.87.37.1)
7       if-5-2.tcore2.SVW-Singapore.as6453.net (180.87.15.69)
8       if-7-2.tcore2.LVW-LosAngeles.as6453.net (180.87.15.26)
9       Vlan28.icore1.EQL-LosAngeles.as6453.net (216.6.84.54)
10      be3003.ccr22.lax05.atlas.cogentco.com (154.54.11.193)
11      be2024.ccr22.lax01.atlas.cogentco.com (154.54.30.193)
12      te0-1-0-6.ccr22.iah01.atlas.cogentco.com (66.28.4.237)
13      te0-4-0-1.ccr22.dfw01.atlas.cogentco.com (154.54.24.2)
14      te0-4-0-7.mpd22.mci01.atlas.cogentco.com (154.54.80.106)
15      te0-2-0-3.ccr22.ord01.atlas.cogentco.com (154.54.6.202)
16      te3-2.ccr01.sbn01.atlas.cogentco.com (154.54.25.61)
17      te4-3.ccr01.dtw04.atlas.cogentco.com (154.54.5.153)
18      38.122.60.66 (38.122.60.66)
19      www.rvce.edu.in (173.230.242.176)


[root@localhost Documents]# _
```

Screenshot 1

A traceroute to www.rvce.edu.in shows that the 19th hop results in reaching the destination. It can also be observed that whenever host names are not available the ip addresses are displayed.

```
[root@localhost Documents]# host www.timesofindia.com
www.timesofindia.com is an alias for timesofindia.indiatimes.com.edgesuite.net.
timesofindia.indiatimes.com.edgesuite.net is an alias for a1521.g.akamai.net.
a1521.g.akamai.net has address 117.239.240.8
a1521.g.akamai.net has address 117.239.240.33
[root@localhost Documents]# ./traceroute.py www.timesofindia.com
1       D-Link.Home (192.168.1.1)
2       117.192.192.1 (117.192.192.1)
3       218.248.160.198 (218.248.160.198)
4       218.248.255.42 (218.248.255.42)
5       218.248.250.174 (218.248.250.174)
6       210.212.70.186 (210.212.70.186)
7       www.timesofindia.com (117.239.240.33)


[root@localhost Documents]# traceroute www.timesofindia.com
traceroute to www.timesofindia.com (117.239.240.33), 30 hops max, 60 byte packets
 1  D-Link.Home (192.168.1.1)  0.378 ms  0.443 ms  0.487 ms
 2  117.192.192.1 (117.192.192.1)  23.487 ms  27.167 ms  28.016 ms
 3  218.248.160.198 (218.248.160.198)  31.585 ms  32.720 ms  36.542 ms
 4  218.248.255.2 (218.248.255.2)  40.198 ms  42.779 ms  44.903 ms
 5  218.248.250.174 (218.248.250.174)  58.901 ms  61.203 ms  63.561 ms
 6  210.212.70.182 (210.212.70.182)  65.903 ms  66.372 ms  68.948 ms
 7  117.239.240.33 (117.239.240.33)  71.519 ms  35.475 ms  35.609 ms
[root@localhost Documents]# _
```

Screenshot 2

The output of the default UNIX traceroute command shows that the tracing achieved by this application is successful.