Ex.No.: 1 DATE: 21/08/2021	IMPLEMENTING THE BASIC COMMANDS

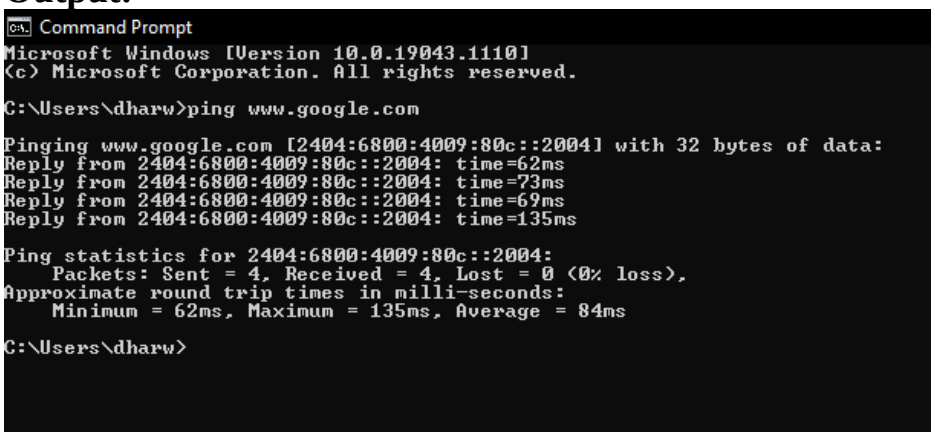
Aim:

To implement the basic commands.

1) ping command:

- The ping command is a Command Prompt command used to test the ability of the source computer to reach a specified destination computer.
- The ping command sends one datagram per second and prints one line of output for every response received.
- The ping command calculates round-trip times and packet loss statistics, and displays a brief summary on completion.
- The ping command completes when the program times out or on receipt of a SIGINT signal.

Output:



2) ipconfig command:

- ipconfig (standing for "Internet Protocol configuration") is a console application program of some computer operating systems that displays all current TCP/IP network configuration values and refreshes Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) settings.
- Another indispensable and frequently used utility that is used for finding network information about your local machine like IP addresses, DNS addresses etc
- Basic Use: Finding Your IP Address and Default Gateway

Output:

(a) ipconfig /all command:

- ipconfig /all displays all configuration information for each adapter bound to TCP/IP.
- It displays more information about the network setup on your systems including the MAC address.

Output:

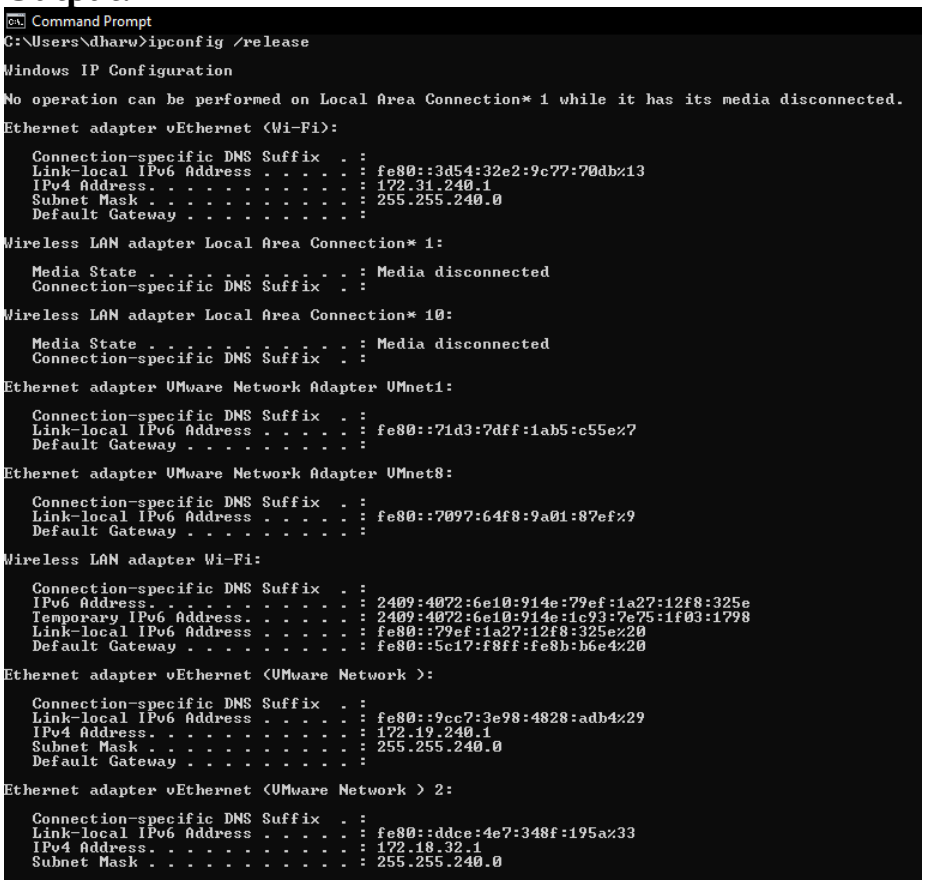
```
Command Prompt

C:\Users\dharw>ipconfig /all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : Dharwin
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter vEthernet (Wi-Fi):

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Hyper-V Virtual Ethernet Adapter
   Physical Address. . . . . . . . . : 00-15-5D-C2-02-27
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::3d54:32e2:9c77:70db%13(Preferred)
   IPv4 Address. . . . . . . . . . . : 172.31.240.1(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 218109277
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-27-A1-26-2F-00-45-E2-3B-5A-89
   DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
                                       fec0:0:0:ffff::2%1
                                       fec0:0:0:ffff::3%1
   NetBIOS over Tcpip. . . . . . . . : Enabled

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter
   Physical Address. . . . . . . . . : 02-45-E2-3B-5A-89
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Local Area Connection* 10:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter #2
   Physical Address. . . . . . . . . : 82-45-E2-3B-5A-89
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes

Ethernet adapter VMware Network Adapter VMnet1:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : VMware Virtual Ethernet Adapter for VMnet1
   Physical Address. . . . . . . . . : 00-50-56-C0-00-01
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::71d3:7dff:1ab5:c55e%7(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.161.1(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : 05 September 2021 14:58:12
```

```
Command Prompt

   Description . . . . . . . . . . . : Realtek 8822CE-US Wireless LAN 802.11ac PCI-E NIC
   Physical Address. . . . . . . . . : 00-45-E2-3B-5A-89
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . : 2409:4072:6e10:914e:79ef:1a27:12f8:325e(Preferred)
   Temporary IPv6 Address. . . . . . : 2409:4072:6e10:914e:1c93:7e75:1f03:1798(Preferred)
   Link-local IPv6 Address . . . . . : fe80::79ef:1a27:12f8:325e%20(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.43.69(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : 05 September 2021 14:49:39
   Lease Expires . . . . . . . . . . : 05 September 2021 15:50:13
   Default Gateway . . . . . . . . . : fe80::5c17:f8ff:fe8b:b6e4%20
                                       192.168.43.1
   DHCP Server . . . . . . . . . . . : 192.168.43.1
   DHCPv6 IAID . . . . . . . . . . . : 117458402
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-27-A1-26-2F-00-45-E2-3B-5A-89
   DNS Servers . . . . . . . . . . . : 192.168.43.1
   NetBIOS over Tcpip. . . . . . . . : Enabled

Ethernet adapter vEthernet (VMware Network ):

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Hyper-V Virtual Ethernet Adapter #2
   Physical Address. . . . . . . . . : 00-15-5D-EF-B0-F2
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::9cc7:3e98:4828:adb4%29(Preferred)
   IPv4 Address. . . . . . . . . . . : 172.19.240.1(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 486544733
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-27-A1-26-2F-00-45-E2-3B-5A-89
   DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
                                       fec0:0:0:ffff::2%1
                                       fec0:0:0:ffff::3%1
   NetBIOS over Tcpip. . . . . . . . : Enabled

Ethernet adapter vEthernet (VMware Network ) 2:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Hyper-V Virtual Ethernet Adapter #3
   Physical Address. . . . . . . . . : 00-15-5D-12-48-68
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::ddce:4e7:348f:195a%33(Preferred)
   IPv4 Address. . . . . . . . . . . : 172.18.32.1(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 553653597
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-27-A1-26-2F-00-45-E2-3B-5A-89
   DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
                                       fec0:0:0:ffff::2%1
                                       fec0:0:0:ffff::3%1
   NetBIOS over Tcpip. . . . . . . . : Enabled

C:\Users\dharw>
```

(b) ipconfig /release command:

- It releases the current IP address.
- First, ipconfig /release is executed to force the client to immediately give up its lease by sending the server a DHCP release notification which updates the server's status information and marks the old client's IP address as "available".
- Then, the command ipconfig /renew is executed to request a new IP address.

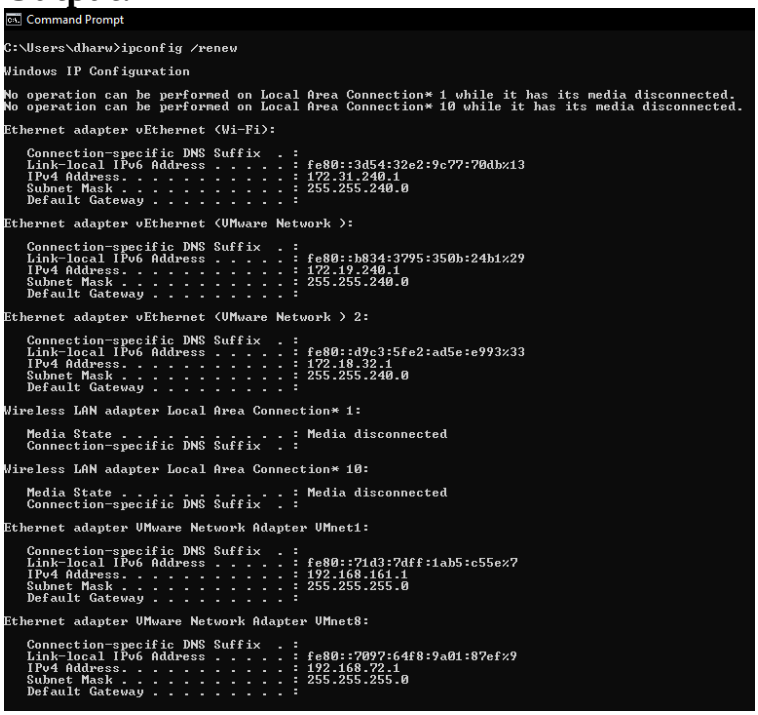
Output:



(c) ipconfig /renew command:

- It renews IP address.
- Ipconfig /renew is the command used to tell the DHCP server that your computer wishes to join the network and needs to be configured with an IP address to communicate with the other devices on the network.

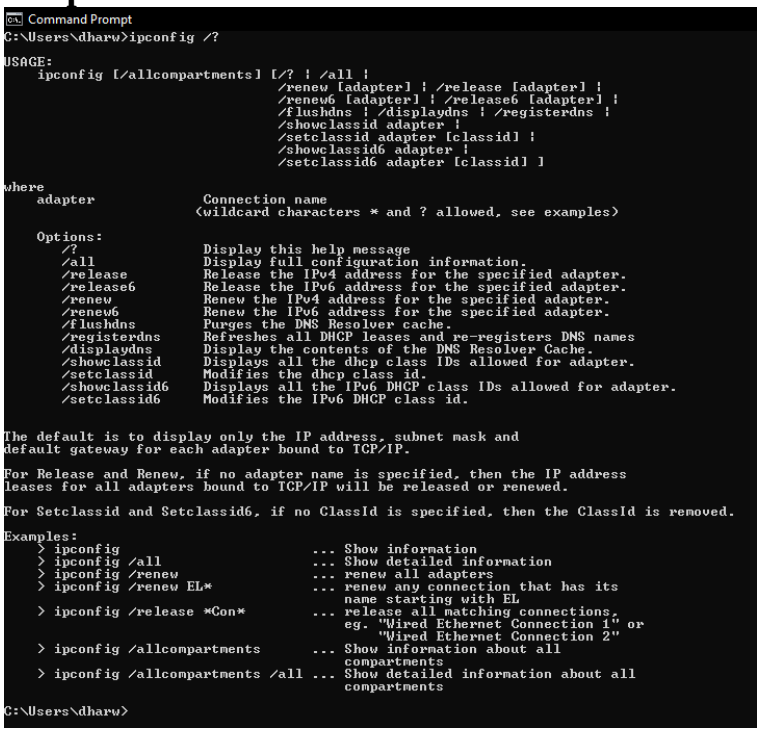
Output:



(d) ipconfig /? command:

- It shows help.

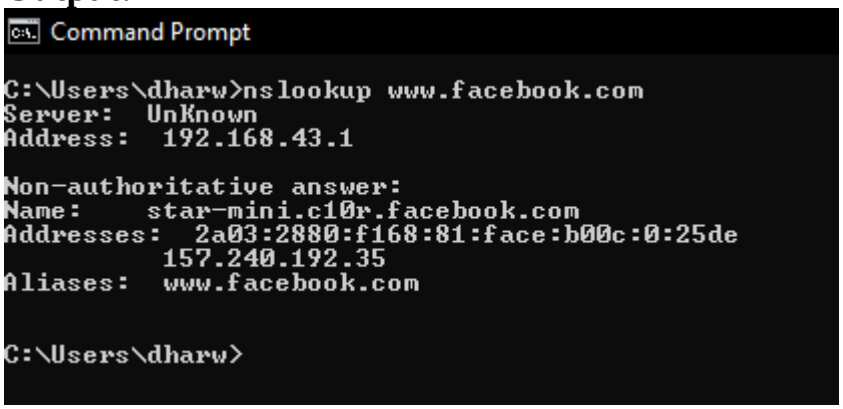
Output:



3) nslookup command:

- Used for checking DNS record entries.
- nslookup is a network administration command-line tool available for many computer operating systems.
- It is used for querying the Domain Name System (DNS) to obtain domain name or IP address mapping information.
- The main use of nslookup is for troubleshooting DNS related problems.

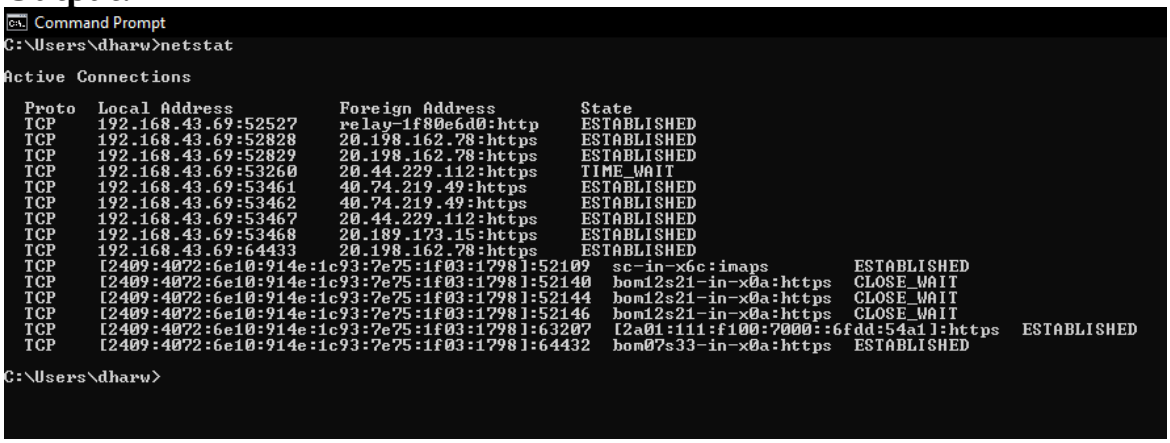
Output:



4) netstat command:

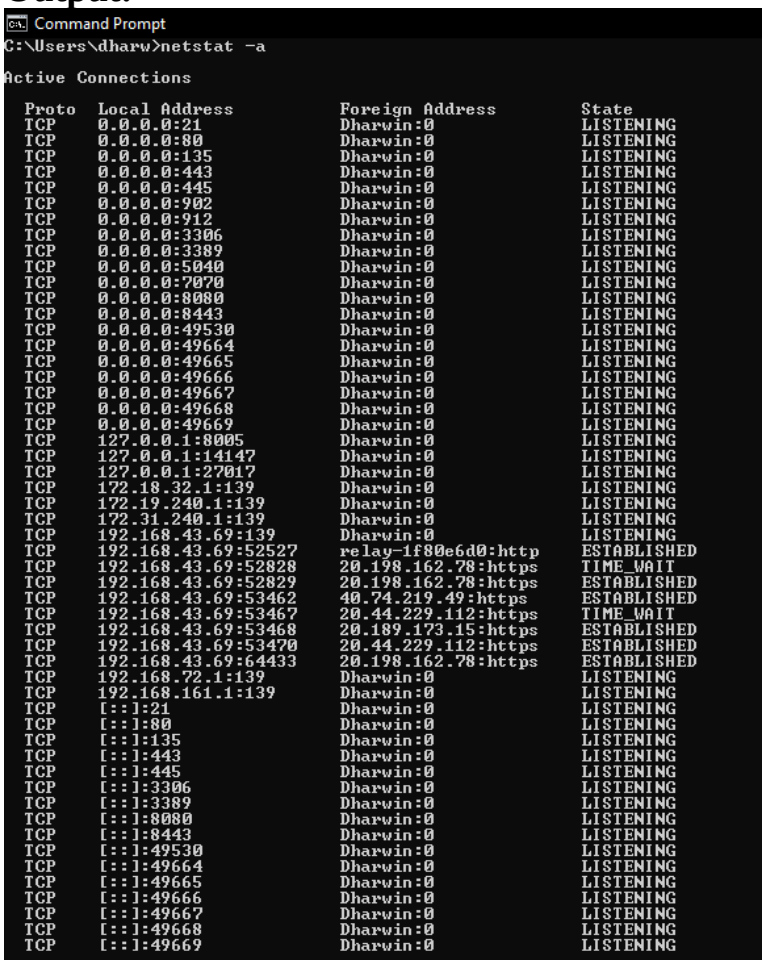
- The netstat command generates displays that show network status and protocol statistics.
- You can display the status of TCP and UDP endpoints in table format, routing table information, and interface information.
- The most frequently used options for determining network status are: s, r, and i.

Output:

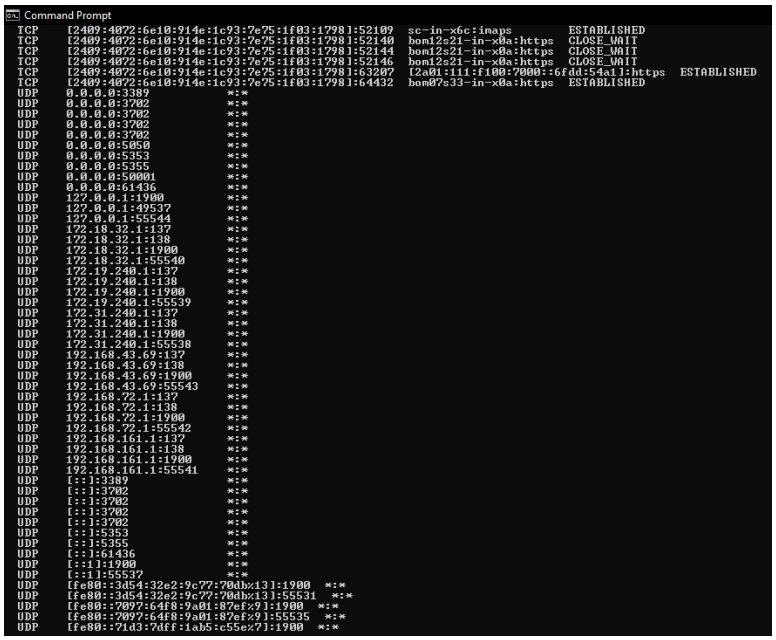


(a) netstat -a command: This switch displays active TCP connections, TCP connections with the listening state, as well as UDP ports that are being listened to.

Output:



```
Command Prompt
C:\Users\dharw>netstat -a

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:21             Dharwin:0              LISTENING
  TCP    0.0.0.0:80             Dharwin:0              LISTENING
  TCP    0.0.0.0:135            Dharwin:0              LISTENING
  TCP    0.0.0.0:443            Dharwin:0              LISTENING
  TCP    0.0.0.0:445            Dharwin:0              LISTENING
  TCP    0.0.0.0:902            Dharwin:0              LISTENING
  TCP    0.0.0.0:912            Dharwin:0              LISTENING
  TCP    0.0.0.0:3306           Dharwin:0              LISTENING
  TCP    0.0.0.0:3389           Dharwin:0              LISTENING
  TCP    0.0.0.0:5040           Dharwin:0              LISTENING
  TCP    0.0.0.0:7070           Dharwin:0              LISTENING
  TCP    0.0.0.0:8080           Dharwin:0              LISTENING
  TCP    0.0.0.0:8443           Dharwin:0              LISTENING
  TCP    0.0.0.0:49530          Dharwin:0              LISTENING
  TCP    0.0.0.0:49664          Dharwin:0              LISTENING
  TCP    0.0.0.0:49665          Dharwin:0              LISTENING
  TCP    0.0.0.0:49666          Dharwin:0              LISTENING
  TCP    0.0.0.0:49667          Dharwin:0              LISTENING
  TCP    0.0.0.0:49668          Dharwin:0              LISTENING
  TCP    0.0.0.0:49669          Dharwin:0              LISTENING
  TCP    127.0.0.1:8005         Dharwin:0              LISTENING
  TCP    127.0.0.1:14147        Dharwin:0              LISTENING
  TCP    127.0.0.1:27017        Dharwin:0              LISTENING
  TCP    172.18.32.1:139        Dharwin:0              LISTENING
  TCP    172.19.240.1:139       Dharwin:0              LISTENING
  TCP    172.31.240.1:139       Dharwin:0              LISTENING
  TCP    192.168.43.69:139      Dharwin:0              LISTENING
  TCP    192.168.43.69:52527    relay-1f80e6d0:http    ESTABLISHED
  TCP    192.168.43.69:52828    20.198.162.78:https    TIME_WAIT
  TCP    192.168.43.69:52829    20.198.162.78:https    ESTABLISHED
  TCP    192.168.43.69:53462    40.74.219.49:https     ESTABLISHED
  TCP    192.168.43.69:53467    20.44.229.112:https    TIME_WAIT
  TCP    192.168.43.69:53468    20.189.173.15:https    ESTABLISHED
  TCP    192.168.43.69:53470    20.44.229.112:https    ESTABLISHED
  TCP    192.168.43.69:64433    20.198.162.78:https    ESTABLISHED
  TCP    192.168.72.1:139       Dharwin:0              LISTENING
  TCP    192.168.161.1:139      Dharwin:0              LISTENING
  TCP    [::]:21                Dharwin:0              LISTENING
  TCP    [::]:80                Dharwin:0              LISTENING
  TCP    [::]:135               Dharwin:0              LISTENING
  TCP    [::]:443               Dharwin:0              LISTENING
  TCP    [::]:445               Dharwin:0              LISTENING
  TCP    [::]:3306              Dharwin:0              LISTENING
  TCP    [::]:3389              Dharwin:0              LISTENING
  TCP    [::]:8080              Dharwin:0              LISTENING
  TCP    [::]:8443              Dharwin:0              LISTENING
  TCP    [::]:49530             Dharwin:0              LISTENING
  TCP    [::]:49664             Dharwin:0              LISTENING
  TCP    [::]:49665             Dharwin:0              LISTENING
  TCP    [::]:49666             Dharwin:0              LISTENING
  TCP    [::]:49667             Dharwin:0              LISTENING
  TCP    [::]:49668             Dharwin:0              LISTENING
  TCP    [::]:49669             Dharwin:0              LISTENING
```



```
Command Prompt
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52109  sc-in-x6c:imaps        ESTABLISHED
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52140  bom12s21-in-x0a:https  CLOSE_WAIT
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52144  bom12s21-in-x0a:https  CLOSE_WAIT
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52146  bom12s21-in-x0a:https  CLOSE_WAIT
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:63207  [2a01:111:f100:7000::6fdd:54a1]:https  ESTABLISHED
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:64432  bom07s33-in-x0a:https  ESTABLISHED
  UDP    0.0.0.0:3389           *:*
  UDP    0.0.0.0:3702           *:*
  UDP    0.0.0.0:3702           *:*
  UDP    0.0.0.0:3702           *:*
  UDP    0.0.0.0:3702           *:*
  UDP    0.0.0.0:5050           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5355           *:*
  UDP    0.0.0.0:50001          *:*
  UDP    0.0.0.0:61436          *:*
  UDP    127.0.0.1:1900         *:*
  UDP    127.0.0.1:49537        *:*
  UDP    127.0.0.1:55544        *:*
  UDP    172.18.32.1:137        *:*
  UDP    172.18.32.1:138        *:*
  UDP    172.18.32.1:1900       *:*
  UDP    172.18.32.1:55540      *:*
  UDP    172.19.240.1:137       *:*
  UDP    172.19.240.1:138       *:*
  UDP    172.19.240.1:1900      *:*
  UDP    172.19.240.1:55539     *:*
  UDP    172.31.240.1:137       *:*
  UDP    172.31.240.1:138       *:*
  UDP    172.31.240.1:1900      *:*
  UDP    172.31.240.1:55538     *:*
  UDP    192.168.43.69:137      *:*
  UDP    192.168.43.69:138      *:*
  UDP    192.168.43.69:1900     *:*
  UDP    192.168.43.69:55543    *:*
  UDP    192.168.72.1:137       *:*
  UDP    192.168.72.1:138       *:*
  UDP    192.168.72.1:1900      *:*
  UDP    192.168.72.1:55542     *:*
  UDP    192.168.161.1:137      *:*
  UDP    192.168.161.1:138      *:*
  UDP    192.168.161.1:1900     *:*
  UDP    192.168.161.1:55541    *:*
  UDP    [::1]:3389             *:*
  UDP    [::]:3702              *:*
  UDP    [::]:3702              *:*
  UDP    [::]:3702              *:*
  UDP    [::]:3702              *:*
  UDP    [::]:5353              *:*
  UDP    [::]:5355              *:*
  UDP    [::]:61436             *:*
  UDP    [::1]:1900             *:*
  UDP    [::1]:55537            *:*
  UDP    [fe80::3d54:32e2:9c77:70db%13]:1900  *:*
  UDP    [fe80::3d54:32e2:9c77:70db%13]:55531  *:*
  UDP    [fe80::7097:64f8:9a01:87ef%91]:1900  *:*
  UDP    [fe80::7097:64f8:9a01:87ef%91]:55535  *:*
  UDP    [fe80::71d3:7dff:1ab5:c55e%7]:1900  *:*
```

(b) netstat -b: This netstat switch is very similar to the **-o** switch listed below, but instead of displaying the PID, will display the process's actual file name. Using **-b** over **-o** might seem like it's saving you a step or two but using it can sometimes greatly extend the time it takes netstat to fully execute.

Output:

```
C:\Users\dharw>netstat -b
The requested operation requires elevation.

C:\Users\dharw>
```

(c) netstat -e command: Use this switch with the netstat command to show statistics about your network connection. This data includes bytes, unicast packets, non-unicast packets, discards, errors, and unknown protocols received and sent since the connection was established.

Output:

```
C:\Users\dharw>netstat -e
Interface Statistics

                           Received            Sent

Bytes                     2356539          7816405
Unicast packets              5811             6974
Non-unicast packets           300            26821
Discards                        0                0
Errors                          0                0
Unknown protocols               0

C:\Users\dharw>
```

(d) netstat -f command: The **-f** switch will force the netstat command to display the Fully Qualified Domain Name (FQDN) for each foreign IP addresses when possible.

Output:

```
Command Prompt
C:\Users\dharw>netstat -f

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    192.168.43.69:52527    relay-1f80e6d0.net.anydesk.com:http  ESTABLISHED
  TCP    192.168.43.69:52829    20.198.162.78:https    ESTABLISHED
  TCP    192.168.43.69:53462    40.74.219.49:https     ESTABLISHED
  TCP    192.168.43.69:53471    20.44.229.112:https    ESTABLISHED
  TCP    192.168.43.69:64433    20.198.162.78:https    ESTABLISHED
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52109  sc-in-x6c.1e100.net:imaps  ESTABLISHED
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52140  bom12s21-in-x0a.1e100.net:https  CLOSE_WAIT
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52144  bom12s21-in-x0a.1e100.net:https  CLOSE_WAIT
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52146  bom12s21-in-x0a.1e100.net:https  CLOSE_WAIT
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:63207  [2a01:111:f100:7000::6fdd:54a1]:https  ESTABLISHED
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:64432  bom07s33-in-x0a.1e100.net:https  ESTABLISHED

C:\Users\dharw>
```

(e) netstat -n command: Use the **-n** switch to prevent netstat from attempting to determine host names for foreign IP addresses. Depending on your current network connections, using this switch could considerably reduce the time it takes for netstat to fully execute.

Output:

```
C:\Users\dharw>netstat -n

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    192.168.43.69:52527    168.119.209.188:80     ESTABLISHED
  TCP    192.168.43.69:52829    20.198.162.78:443      ESTABLISHED
  TCP    192.168.43.69:53462    40.74.219.49:443       ESTABLISHED
  TCP    192.168.43.69:53471    20.44.229.112:443      ESTABLISHED
  TCP    192.168.43.69:53473    40.125.122.176:443     TIME_WAIT
  TCP    192.168.43.69:53475    40.125.122.176:443     TIME_WAIT
  TCP    192.168.43.69:53476    52.249.36.207:443      ESTABLISHED
  TCP    192.168.43.69:53477    104.46.162.226:443     TIME_WAIT
  TCP    192.168.43.69:64433    20.198.162.78:443      ESTABLISHED
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52109  [2404:6800:4003:c02::6c]:993  ESTABLISHED
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52140  [2404:6800:4009:831::200a]:443  CLOSE_WAIT
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52144  [2404:6800:4009:831::200a]:443  CLOSE_WAIT
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52146  [2404:6800:4009:831::200a]:443  CLOSE_WAIT
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:53478  [2001:1900:2381:d08::1fe]:80  ESTABLISHED
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:63207  [2a01:111:f100:7000::6fdd:54a1]:443  ESTABLISHED
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:64432  [2404:6800:4009:826::200a]:443  ESTABLISHED

C:\Users\dharw>
```

(f) netstat -o command: A handy option for many troubleshooting tasks, the **-o** switch displays the process identifier (PID) associated with each displayed connection. See the example below for more about using **netstat -o**.

Output:

```
Command Prompt
C:\Users\dharw>netstat -o

Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    192.168.43.69:52527    relay-1f80e6d0:http    ESTABLISHED     4604
  TCP    192.168.43.69:52829    20.198.162.78:https    ESTABLISHED     4932
  TCP    192.168.43.69:53462    40.74.219.49:https     ESTABLISHED     13432
  TCP    192.168.43.69:53471    20.44.229.112:https    ESTABLISHED     9168
  TCP    192.168.43.69:53473    40.125.122.176:https   TIME_WAIT       0
  TCP    192.168.43.69:53475    40.125.122.176:https   TIME_WAIT       0
  TCP    192.168.43.69:53476    52.249.36.207:https    TIME_WAIT       0
  TCP    192.168.43.69:53477    104.46.162.226:https   TIME_WAIT       0
  TCP    192.168.43.69:53479    20.190.145.141:https   ESTABLISHED     11084
  TCP    192.168.43.69:53480    104.208.16.89:https    TIME_WAIT       0
  TCP    192.168.43.69:53481    20.54.89.15:https      ESTABLISHED     6152
  TCP    192.168.43.69:64433    20.198.162.78:https    ESTABLISHED     11064
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52109  sc-in-x6c:imaps        ESTABLISHED     12532
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52140  bom12s21-in-x0a:https  CLOSE_WAIT      1688
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52144  bom12s21-in-x0a:https  CLOSE_WAIT      1688
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52146  bom12s21-in-x0a:https  CLOSE_WAIT      1688
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:53478  [2001:1900:2381:d08::1fe]:http  TIME_WAIT       0
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:53488  [2001:1900:2362:7::1fe]:http  TIME_WAIT       0
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:53489  [2001:1900:2381:2011::1fe]:http  TIME_WAIT       0
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:53492  [2001:1900:2362:11::1fe]:http  TIME_WAIT       0
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:53493  [2001:1900:2362:7::1fe]:http  TIME_WAIT       0
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:53494  [2001:1900:2381:a0c::1fe]:http  TIME_WAIT       0
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:53495  [2001:1900:2381:a::1fe]:http  TIME_WAIT       0
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:63207  [2a01:111:f100:7000::6fdd:54a1]:https  ESTABLISHED     13848
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:64432  bom07s33-in-x0a:https  ESTABLISHED     1688

C:\Users\dharw>
```

(g) netstat -p protocol:

- Use the **-p** switch to show connections or statistics only for a particular *protocol*. You can not define more than one *protocol* at once, nor can you execute netstat with **-p** without defining a *protocol*.
- When specifying a *protocol* with the **-p** option, you can use **tcp**, **udp**, **tcpv6**, or **udpv6**. If you use **-s** with **-p** to view statistics by protocol, you can use **icmp**, **ip**, **icmpv6**, or **ipv6** in addition to the first four I mentioned

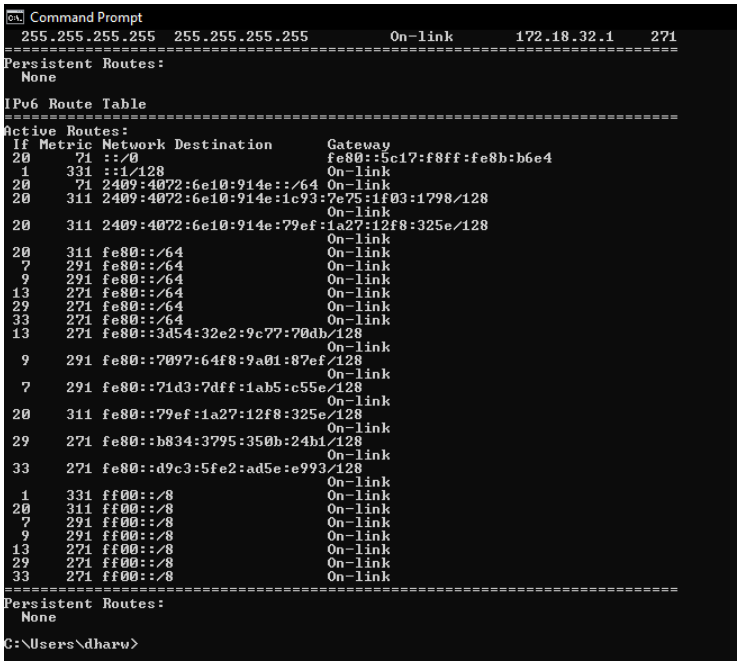
Output:

```
Command Prompt
Microsoft Windows [Version 10.0.19043.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dharw>netstat -p tcp

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    192.168.43.69:53323    40.74.219.49:https     ESTABLISHED
  TCP    192.168.43.69:58907    relay-1f80e6d0:http    ESTABLISHED
  TCP    192.168.43.69:58909    13.76.153.29:https     ESTABLISHED
  TCP    192.168.43.69:59012    20.44.229.112:https    ESTABLISHED
  TCP    192.168.43.69:59013    20.44.229.112:https    ESTABLISHED
  TCP    192.168.43.69:59974    20.198.162.78:https    ESTABLISHED
  TCP    192.168.43.69:62043    20.198.162.78:https    ESTABLISHED
  TCP    192.168.43.69:62048    20.198.162.78:https    ESTABLISHED

C:\Users\dharw>
```

(h) netstat -r command: Execute netstat with **-r** to show the IP routing table. This is the same as using the route command to execute **route print**.

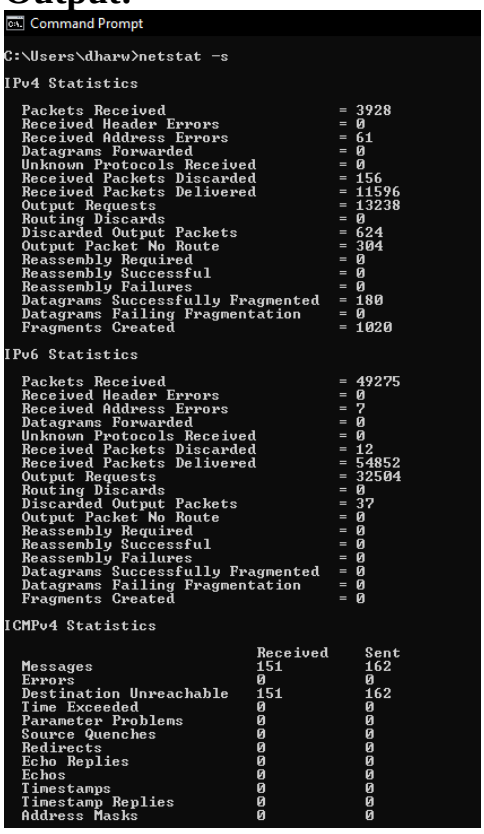
Output:

```
Command Prompt

C:\Users\dharw>netstat -r
===========================================================================
Interface List
 13...00 15 5d c2 02 27 ......Hyper-V Virtual Ethernet Adapter
 29...00 15 5d 64 06 16 ......Hyper-V Virtual Ethernet Adapter #2
 33...00 15 5d 99 fe 8c ......Hyper-V Virtual Ethernet Adapter #3
  8...02 45 e2 3b 5a 89 ......Microsoft Wi-Fi Direct Virtual Adapter
  4...82 45 e2 3b 5a 89 ......Microsoft Wi-Fi Direct Virtual Adapter #2
  7...00 50 56 c0 00 01 ......VMware Virtual Ethernet Adapter for VMnet1
  9...00 50 56 c0 00 08 ......VMware Virtual Ethernet Adapter for VMnet8
 20...00 45 e2 3b 5a 89 ......Realtek 8822CE-VS Wireless LAN 802.11ac PCI-E NIC
  1...........................Software Loopback Interface 1
===========================================================================

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.43.1    192.168.43.69     55
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
      172.18.32.0    255.255.240.0         On-link      172.18.32.1    271
      172.18.32.1  255.255.255.255         On-link      172.18.32.1    271
   172.18.47.255  255.255.255.255         On-link      172.18.32.1    271
     172.19.240.0    255.255.240.0         On-link     172.19.240.1    271
     172.19.240.1  255.255.255.255         On-link     172.19.240.1    271
   172.19.255.255  255.255.255.255         On-link     172.19.240.1    271
     172.31.240.0    255.255.240.0         On-link     172.31.240.1    271
     172.31.240.1  255.255.255.255         On-link     172.31.240.1    271
   172.31.255.255  255.255.255.255         On-link     172.31.240.1    271
     192.168.43.0    255.255.255.0         On-link    192.168.43.69    311
    192.168.43.69  255.255.255.255         On-link    192.168.43.69    311
   192.168.43.255  255.255.255.255         On-link    192.168.43.69    311
     192.168.72.0    255.255.255.0         On-link     192.168.72.1    291
     192.168.72.1  255.255.255.255         On-link     192.168.72.1    291
   192.168.72.255  255.255.255.255         On-link     192.168.72.1    291
    192.168.161.0    255.255.255.0         On-link    192.168.161.1    291
    192.168.161.1  255.255.255.255         On-link    192.168.161.1    291
  192.168.161.255  255.255.255.255         On-link    192.168.161.1    291
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link    192.168.43.69    311
        224.0.0.0        240.0.0.0         On-link    192.168.161.1    291
        224.0.0.0        240.0.0.0         On-link     192.168.72.1    291
        224.0.0.0        240.0.0.0         On-link     172.31.240.1    271
        224.0.0.0        240.0.0.0         On-link     172.19.240.1    271
        224.0.0.0        240.0.0.0         On-link      172.18.32.1    271
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link    192.168.43.69    311
  255.255.255.255  255.255.255.255         On-link    192.168.161.1    291
  255.255.255.255  255.255.255.255         On-link     192.168.72.1    291
  255.255.255.255  255.255.255.255         On-link     172.31.240.1    271
  255.255.255.255  255.255.255.255         On-link     172.19.240.1    271
  255.255.255.255  255.255.255.255         On-link      172.18.32.1    271
===========================================================================
Persistent Routes:
  None
```



(i) netstat -s command: The **-s** option can be used with the netstat command to show detailed statistics by protocol. You can limit the statistics shown to a particular protocol by using the **-s**option and specifying that *protocol*, but be sure to use **-s** before **-p** *protocol* when using the switches together.

Output:



```
Command Prompt
  Router Advertisements     0           0

ICMPv6 Statistics

                            Received    Sent
  Messages                  105         292
  Errors                    0           0
  Destination Unreachable   7           7
  Packet Too Big            0           0
  Time Exceeded             0           0
  Parameter Problems        0           0
  Echos                     0           4
  Echo Replies              4           0
  MLD Queries               0           0
  MLD Reports               0           0
  MLD Dones                 0           0
  Router Solicitations      0           83
  Router Advertisements     6           0
  Neighbor Solicitations    51          105
  Neighbor Advertisements   37          93
  Redirects                 0           0
  Router Renumberings       0           0

TCP Statistics for IPv4

  Active Opens                            = 194
  Passive Opens                           = 0
  Failed Connection Attempts              = 126
  Reset Connections                       = 56
  Current Connections                     = 4
  Segments Received                       = 3534
  Segments Sent                           = 1396
  Segments Retransmitted                  = 0

TCP Statistics for IPv6

  Active Opens                            = 78
  Passive Opens                           = 2
  Failed Connection Attempts              = 807
  Reset Connections                       = 6
  Current Connections                     = 6
  Segments Received                       = 49208
  Segments Sent                           = 23222
  Segments Retransmitted                  = 0

UDP Statistics for IPv4

  Datagrams Received    = 6471
  No Ports              = 185
  Receive Errors        = 0
  Datagrams Sent        = 8256

UDP Statistics for IPv6

  Datagrams Received    = 4571
  No Ports              = 8
  Receive Errors        = 0
  Datagrams Sent        = 6228
```

(j) netstat -t command: Use the **-t** switch to show the current TCP chimney offload state in place of the typically displayed TCP state.

Output:

```
Command Prompt

C:\Users\dharw>netstat -t

Active Connections

  Proto  Local Address          Foreign Address        State           Offload State

  TCP    192.168.43.69:52527    relay-1f80e6d0:http    ESTABLISHED     InHost
  TCP    192.168.43.69:52829    20.198.162.78:https    ESTABLISHED     InHost
  TCP    192.168.43.69:53462    40.74.219.49:https     ESTABLISHED     InHost
  TCP    192.168.43.69:53503    20.44.229.112:https    TIME_WAIT       InHost
  TCP    192.168.43.69:53504    52.109.12.20:https     ESTABLISHED     InHost
  TCP    192.168.43.69:53505    a-0003:https           ESTABLISHED     InHost
  TCP    192.168.43.69:64433    20.198.162.78:https    ESTABLISHED     InHost
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52109  sc-in-x6c:imaps         CLOSE_WAIT      InHost
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52140  bom12s21-in-x0a:https   CLOSE_WAIT      InHost
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52144  bom12s21-in-x0a:https   CLOSE_WAIT      InHost
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:52146  bom12s21-in-x0a:https   CLOSE_WAIT      InHost
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:63207  [2a01:111:f100:7000::6fdd:54a1]:https  ESTABLISHED     InHost
  TCP    [2409:4072:6e10:914e:1c93:7e75:1f03:1798]:64432  bom07s33-in-x0a:https   ESTABLISHED     InHost

C:\Users\dharw>
```

(k) netstat -x command: Use the **-x** option to show all NetworkDirect listeners, connections, and shared endpoints.

Output:

```
C:\Users\dharw>netstat -x

Active NetworkDirect Connections, Listeners, SharedEndpoints
  Mode   IfIndex Type            Local Address           Foreign Address         PID

C:\Users\dharw>
```

(l) netstat -y command: The **-y** switch can be used to show the TCP connection template for all connection. You cannot use **-y** with any other netstat option.

Output:

```
C:\Users\dharw>netstat -y

Active Connections
  Proto  Local Address          Foreign Address        State           Template
TCP    192.168.43.69:52527    relay-1f80e6d0:http    ESTABLISHED     Internet
TCP    192.168.43.69:52829    20.198.162.78:https    ESTABLISHED     Internet
TCP    192.168.43.69:53462    40.74.219.49:https     ESTABLISHED     Internet
TCP    192.168.43.69:53504    52.109.12.20:https     TIME_WAIT       Not Applicable
TCP    192.168.43.69:53506    20.44.229.112:https    ESTABLISHED     Internet
TCP    192.168.43.69:64433    20.198.162.78:https    ESTABLISHED     Internet

C:\Users\dharw>
```

(m) netstat [time_interval] command: This is the time, in seconds, that you'd like the netstat command to re-execute automatically, stopping only when you use Ctrl-C to end the loop.

Output:

```
Command Prompt
Microsoft Windows [Version 10.0.19043.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dharw>netstat 2

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    192.168.43.69:49885    20.44.229.112:https    TIME_WAIT
  TCP    192.168.43.69:49903    52.109.6.0:https       TIME_WAIT
  TCP    192.168.43.69:53323    40.74.219.49:https     ESTABLISHED
  TCP    192.168.43.69:58605    20.50.73.10:https      ESTABLISHED
  TCP    192.168.43.69:58907    relay-1f80e6d0:http    ESTABLISHED
  TCP    192.168.43.69:58908    52.242.97.97:https     TIME_WAIT
  TCP    192.168.43.69:58909    13.76.153.29:https     ESTABLISHED
  TCP    192.168.43.69:58910    20.189.173.4:https     TIME_WAIT
  TCP    192.168.43.69:58911    20.190.146.33:https    ESTABLISHED
  TCP    192.168.43.69:58912    52.152.90.172:https    TIME_WAIT
  TCP    192.168.43.69:58913    20.189.173.5:https     TIME_WAIT
  TCP    192.168.43.69:58914    20.189.173.5:https     TIME_WAIT
  TCP    192.168.43.69:58915    40.119.249.228:https   TIME_WAIT
  TCP    192.168.43.69:58916    40.119.249.228:https   TIME_WAIT
  TCP    192.168.43.69:58920    52.109.56.20:https     TIME_WAIT
  TCP    192.168.43.69:58923    204.79.197.222:https   ESTABLISHED
  TCP    192.168.43.69:58925    20.140.48.71:https     ESTABLISHED
  TCP    192.168.43.69:58926    117.18.237.29:http     ESTABLISHED
  TCP    192.168.43.69:59974    20.198.162.78:https    ESTABLISHED
  TCP    192.168.43.69:62043    20.198.162.78:https    ESTABLISHED
  TCP    192.168.43.69:62048    20.198.162.78:https    ESTABLISHED
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:52125  maa05s05-in-x0a:https  CLOSE_WAIT
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:52126  bom05s05-in-x0a:https  ESTABLISHED
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:52856  sa-in-f188:5228        ESTABLISHED
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:54905  maa05s05-in-x0a:https  CLOSE_WAIT
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:58604  bom07s26-in-x03:https  CLOSE_WAIT
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:58918  [2620:1ec:c11::200]:https  ESTABLISHED
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:58919  [2620:1ec:c11::200]:https  ESTABLISHED
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:58921  [2620:1ec:c11::200]:https  ESTABLISHED
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:58922  [2603:1046:700:5c::2]:https  ESTABLISHED
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:59010  whatsapp-cdn6-shv-02-maa2:https  ESTABLISHED
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:59188  sa-in-f108:imaps       ESTABLISHED
  TCP    [2409:4072:631f:eab6:51f4:d6ce:216:438c]:64115  maa05s05-in-x0a:https  CLOSE_WAIT

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    192.168.43.69:53323    40.74.219.49:https     ESTABLISHED
  TCP    192.168.43.69:58907    relay-1f80e6d0:http    ESTABLISHED
^C
C:\Users\dharw>
```

(n) netstat /? command: Use the help switch to show details about the netstat command's several options.

Output:

```
C:\Users\dharw>netstat /?

Displays protocol statistics and current TCP/IP network connections.

NETSTAT [-a] [-b] [-e] [-f] [-n] [-o] [-p proto] [-r] [-s] [-t] [-x] [-y] [interval]

  -a            Displays all connections and listening ports.
  -b            Displays the executable involved in creating each connection or
                listening port. In some cases well-known executables host
                multiple independent components, and in these cases the
                sequence of components involved in creating the connection
                or listening port is displayed. In this case the executable
                name is in [] at the bottom, on top is the component it called,
                and so forth until TCP/IP was reached. Note that this option
                can be time-consuming and will fail unless you have sufficient
                permissions.
  -e            Displays Ethernet statistics. This may be combined with the -s
                option.
  -f            Displays Fully Qualified Domain Names (FQDN) for foreign
                addresses.
  -n            Displays addresses and port numbers in numerical form.
  -o            Displays the owning process ID associated with each connection.
  -p proto      Shows connections for the protocol specified by proto; proto
                may be any of: TCP, UDP, TCPv6, or UDPv6.  If used with the -s
                option to display per-protocol statistics, proto may be any of:
                IP, IPv6, ICMP, ICMPv6, TCP, TCPv6, UDP, or UDPv6.
  -q            Displays all connections, listening ports, and bound
                nonlistening TCP ports. Bound nonlistening ports may or may not
                be associated with an active connection.
  -r            Displays the routing table.
  -s            Displays per-protocol statistics.  By default, statistics are
                shown for IP, IPv6, ICMP, ICMPv6, TCP, TCPv6, UDP, and UDPv6;
                the -p option may be used to specify a subset of the default.
  -t            Displays the current connection offload state.
  -x            Displays NetworkDirect connections, listeners, and shared
                endpoints.
  -y            Displays the TCP connection template for all connections.
                Cannot be combined with the other options.
  interval      Redisplays selected statistics, pausing interval seconds
                between each display.  Press CTRL+C to stop redisplaying
                statistics.  If omitted, netstat will print the current
                configuration information once.


C:\Users\dharw>
```

5) tracert command:

- TRACERT (Trace Route), is a command-line utility that you can use to trace the path that an Internet Protocol (IP) packet takes to its destination.
- Traceroute is a network diagnostic tool used to track in real-time the pathway taken by a packet on an IP network from source to destination, reporting the IP addresses of all the routers it pinged in between.
- Traceroute also records the time taken for each hop the packet makes during its route to the destination.

Output:

```
C:\Users\dharw>tracert www.facebook.com

Tracing route to star-mini.c10r.facebook.com [2a03:2880:f168:81:face:b00c:0:25de]
over a maximum of 30 hops:

  1     2 ms     2 ms     1 ms  2409:4072:6e10:914e::7c
  2     *        *        *     Request timed out.
  3   201 ms    98 ms    98 ms  2405:200:369:eeee:20::262
  4   204 ms    98 ms    98 ms  2405:200:801:2300::51e
  5   205 ms    98 ms    98 ms  2405:200:801:2300::51f
  6   204 ms    98 ms    99 ms  2405:200:801:2300::49b
  7   306 ms    98 ms    99 ms  ae14.pr01.tir1.tfbnw.net [2620:0:1cff:dead:beee::148]
  8   415 ms    98 ms    98 ms  ae14.pr01.tir1.tfbnw.net [2620:0:1cff:dead:beee::148]
  9   192 ms    98 ms    98 ms  po101.psw02.tir2.tfbnw.net [2620:0:1cff:dead:bef0::795]
 10   202 ms    98 ms    98 ms  po2.msw1aw.01.tir2.tfbnw.net [2a03:2880:f06d:ffff::61]
 11   193 ms    98 ms    98 ms  edge-star-mini6-shv-01-tir2.facebook.com [2a03:2880:f168:81:face:b00c:0:25de]

Trace complete.

C:\Users\dharw>_
```

TITLE	MARKS
OBSERVATION	
RECORD	
TOTAL	
SIGN	

Result:

Thus the basic commands are written and executed.

Ex.No.: 2A DATE: 09/09/2021	ECHO CLIENT AND ECHO SERVER USING TCP SOCKETS

AIM:

To write a socket program for implementation of echo client and echo server.

ALGORITHM:

CLIENT SIDE:

1. Start the program.
2. Create a socket which binds the Ip address of server and the port address to acquire service.
3. after establishing connection send a data to server.
4. Receive and print the same data from server.
5. Close the socket.
6. End the program.

SERVER SIDE:

1. Start the program.
2. Create a server socket to activate the port address.
3. Create a socket for the server socket which accepts the connection.
4. after establishing connection receive the data from client.
5. Print and send the same data to client.
6. Close the socket.
7. End the program.

PROGRAM:

SERVER SIDE:

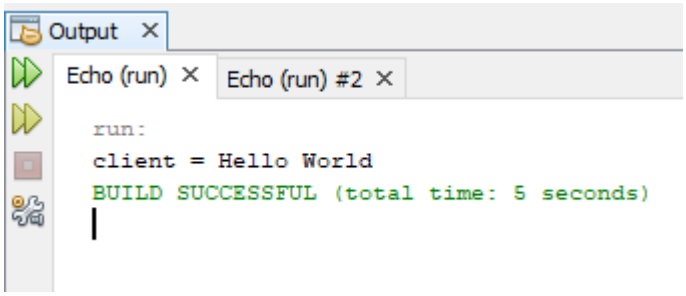
```
import java.io.*;
import java.net.*;

public class Server {

   public static void main(String[] args) {
      try {
         ServerSocket ss = new ServerSocket(6666);
         Socket s = ss.accept();//establishes connection
         DataInputStream dis = new DataInputStream(s.getInputStream());
         DataOutputStream dout = new DataOutputStream(s.getOutputStream());
         String str = (String) dis.readUTF();
         System.out.println("client = " + str);
         dout.writeUTF("Hello World");
         dout.flush();
         ss.close();
      } catch (Exception e) {
         System.out.println(e);
      }
   }
}
```

CLIENT SIDE:

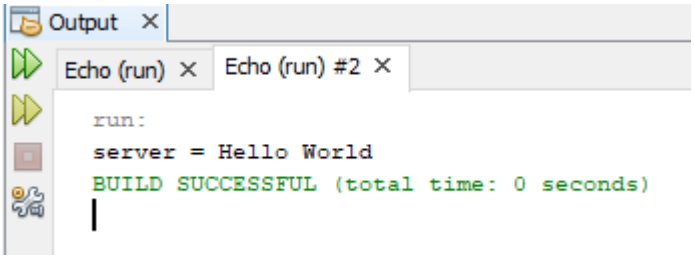
```
import java.io.*;
import java.net.*;

public class Client {

   public static void main(String[] args) {
      try {
         Socket s = new Socket("localhost", 6666);
         DataOutputStream dout = new DataOutputStream(s.getOutputStream());
         DataInputStream dis = new DataInputStream(s.getInputStream());
         dout.writeUTF("Hello World");
         dout.flush();
         String str = (String) dis.readUTF();
         System.out.println("server = " + str);
         dout.close();
         s.close();
      } catch (Exception e) {
         System.out.println(e);
      }
   }
}
```

OUTPUT:

SERVER SIDE:



CLIENT SIDE:



RESULT:

Thus java program for echo client and echo server using TCP sockets are written and executed.

Ex.No: 2B DATE: 09/09/2021	CHAT APPLICATION USING TCP SOCKETS

AIM:

To write a client-server application for chat using TCP sockets.

ALGORITHM:

CLIENT SIDE:

1. Start the program
2. Include necessary package in java
3. To create a socket in client to server.
4. The client establishes a connection to the server.
5. The client accept the connection and to send the data from client to server.
6. The client communicates the server to send the end of the message
7. Stop the program .

SERVER SIDE:

1. Start the program
2. Include necessary package in java
3. To create a socket in server to client
4. The server establishes a connection to the client.
5. The server accept the connection and to send the data from server to client and vice versa
6. The server communicates the client to send the end of the message.

Stop the program

PROGRAM:

SERVER SIDE:

```
import java.net.*;
import java.io.*;

class Server {

   public static void main(String args[]) throws Exception {
      ServerSocket ss = new ServerSocket(3333);
      Socket s = ss.accept();
      DataInputStream din = new DataInputStream(s.getInputStream());
      DataOutputStream dout = new DataOutputStream(s.getOutputStream());
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

      String str = "", str2 = "";
      while (!str.equals("stop")) {
         str = din.readUTF();
         System.out.println("client says: " + str);
         str2 = br.readLine();
         dout.writeUTF(str2);
         dout.flush();
      }
      din.close();
      s.close();
```

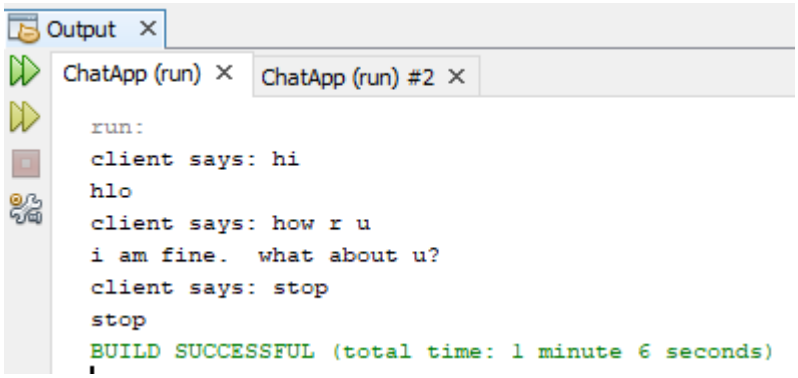
```
        ss.close();
    }
}
```

CLIENT SIDE:

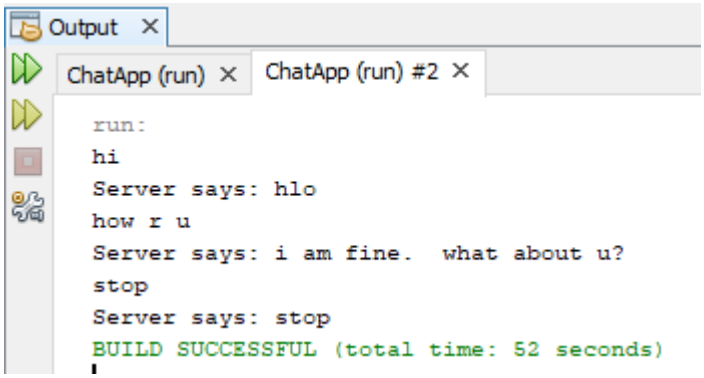
```
import java.net.*;
import java.io.*;

class Client {

    public static void main(String args[]) throws Exception {
        Socket s = new Socket("localhost", 3333);
        DataInputStream din = new DataInputStream(s.getInputStream());
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String str = "", str2 = "";
        while (!str.equals("stop")) {
            str = br.readLine();
            dout.writeUTF(str);
            dout.flush();
            str2 = din.readUTF();
            System.out.println("Server says: " + str2);
        }

        dout.close();
        s.close();
    }
}
```

OUTPUT:

SERVER SIDE:



CLIENT SIDE:



RESULT:

Thus java program for chat application using TCP sockets are written and executed.

Ex.No: 2C DATE: 09/09/2021	FILE TRANSFER IN CLIENT AND SERVER USING TCP SOCKETS.

AIM:

To Perform File Transfer in Client & Server Using TCP SOCKETS

ALGORITHM:

CLIENT SIDE :

1. Start.
2. Establish a connection between the Client and Server.
3. Socketss=new Socket(InetAddress.getLocalHost(),1100);
4. Implement a client that can send two requests.
   i) To get a file from the server.
   ii) To put or send a file to the server.
5. After getting approval from the server, the clients either get file from the server or send file to the server.

SERVER SIDE :

1. Start.
2. Implement a server socket that listens to a particular port number.
3. Server reads the filename and sends the data stored in the file for the‘get’ request.
4. It reads the data from the input stream and writes it to a file in the server for the ‘put’ instruction.
5. Exit upon client’s request.
6. Stop.

PROGRAM:

SERVER SIDE:

```
import java.net.*;
import java.io.*;

public class Server {

   public static void main(String[] args) throws Exception {
      ServerSocket ssock = new ServerSocket(5000);
      Socket socket = ssock.accept();
      InetAddress ia = InetAddress.getByName("localhost");
      File f = new
File("C:\\Users\\dharw\\Documents\\NetBeansProjects\\FileTransfer\\src\\filetransfer\\server.t
xt");
      FileInputStream fin = new FileInputStream(f);
      BufferedInputStream bis = new BufferedInputStream(fin);
      OutputStream os = socket.getOutputStream();
      byte[] contents;
      long fileLength = f.length();
      long current = 0;
      long start = System.nanoTime();
      while (current != fileLength) {
         int size = 10000;
```

```
            if (fileLength - current >= size) {
               current += size;
            } else {
               size = (int) (fileLength - current);
               current = fileLength;
            }
            contents = new byte[size];
            bis.read(contents, 0, size);
            os.write(contents);
            System.out.println("Sending file....." + (current * 100) / fileLength + "% complete");
         }
         os.flush();
         socket.close();
         ssock.close();
         System.out.println("File sent successfully");
      }
}
```

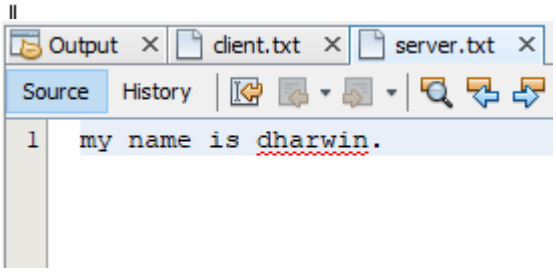
CLIENT SIDE:

```
import java.net.*;
import java.io.*;

public class Client {

   public static void main(String[] args) throws Exception {
      Socket socket = new Socket(InetAddress.getByName("localhost"), 5000);
      byte[] contents = new byte[10000];
      FileOutputStream fout = new
FileOutputStream("C:\\Users\\dharw\\Documents\\NetBeansProjects\\FileTransfer\\src\\filetr
ansfer\\client.txt");
      BufferedOutputStream bos = new BufferedOutputStream(fout);
      InputStream is = socket.getInputStream();
      int bytesRead = 0;
      while ((bytesRead = is.read(contents)) != -1) {
         bos.write(contents, 0, bytesRead);
      }
      bos.flush();
      socket.close();
      System.out.println("file saved successfully");
   }
}
```

OUTPUT:

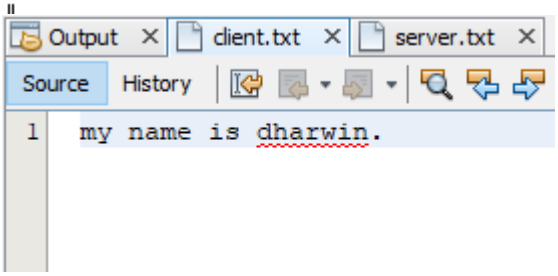
SERVER SIDE:

```
Sending file.....100% complete
File sent successfully
BUILD SUCCESSFUL (total time: 8 seconds)
```



CLIENT SIDE:

```
compile-single:
run-single:
file saved successfully
BUILD SUCCESSFUL (total time: 1 second)
```



TITLE	MARKS
OBSERVATION	
RECORD	
TOTAL	
SIGN	

RESULT:

Thus java program for file transfer in client and server using TCP sockets are written and executed.

Ex.No: 3A DATE: 22/09/21	ADDRESS RESOLUTION PROTOCOL(ARP)

AIM:

To simulate Address Resolution Protocol.

ALGORITHM:

CLIENT SIDE:

1. Establish a connection between the Client and Server.
Socket ss=new Socket(InetAddress.getLocalHost(),1100);
2. Create instance output stream writer
PrintWriter ps=new PrintWriter(s.getOutputStream(),true);
3. Get the IP Address to resolve its physical address.
4. Send the IPAddress to its output Stream.ps.println(ip);
5. Print the Physical Address received from the server.

SERVER SIDE:

1. Accept the connection request by the client.
ServerSocket ss=new ServerSocket(2000);
Socket s=ss.accept();
2. Get the IPaddress from its inputstream.
BufferedReader br1=new BufferedReader(newInputStreamReader(s.getInputStream()));
ip=br1.readLine();
3. During runtime execute the processRuntime r=Runtime.getRuntime();
Process p=r.exec("arp -a "+ip);
4. Send the Physical Address to the client.

PROGRAM:

CLIENT SIDE:

```
import java.io.*;

import java.net.*;

public class Client {

   public static void main(String[] args) {

      try {

         Socket s = new Socket("localhost", 6666);

         DataInputStream din = new DataInputStream(s.getInputStream());

         DataOutputStream dout = new DataOutputStream(s.getOutputStream());

         String ip="239.255.255.250";

         System.out.println("sending internet address - "+ip+" to server");
```

```
            System.out.println("physical address of "+ip+":");
            dout.writeUTF(ip);
            dout.flush();
            System.out.println(din.readUTF());
            dout.close();
            s.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

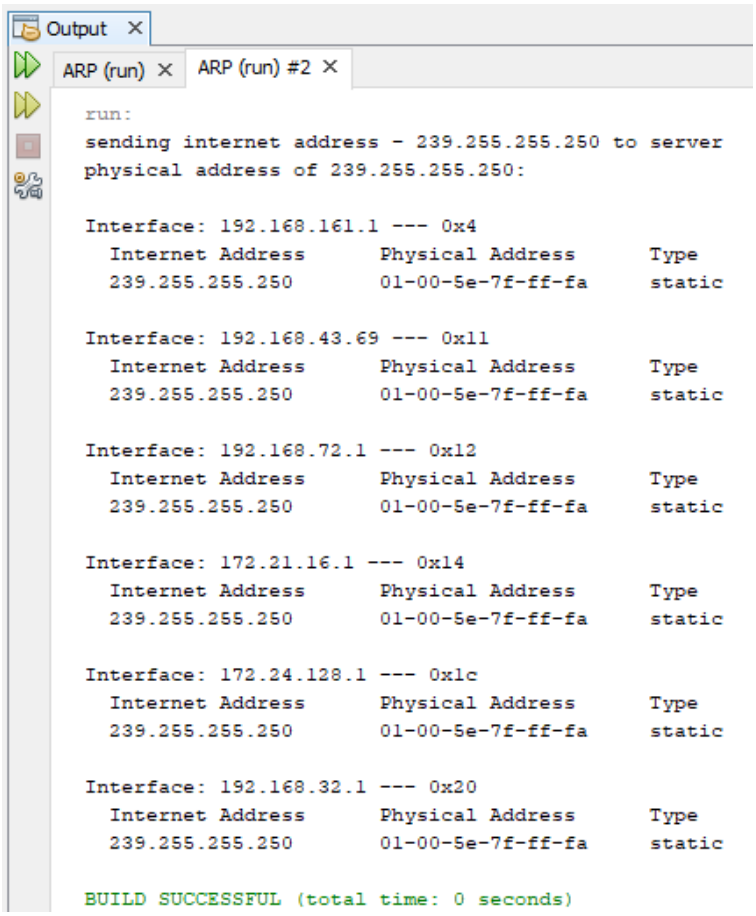
SERVER SIDE:

```
import java.io.*;
import java.net.*;
public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(6666);
            Socket s = ss.accept();//establishes connection
            DataInputStream din = new DataInputStream(s.getInputStream());
            DataOutputStream dout = new DataOutputStream(s.getOutputStream());
            String ip = (String) din.readUTF();
            Runtime r = Runtime.getRuntime();
            System.out.println("C:>arp -a "+ip);
            Process p = r.exec("arp -a " + ip);
            BufferedInputStream bin = new BufferedInputStream(p.getInputStream());
            String output = "";
            int temp = bin.read();
            while (temp != -1) {
                output += (char) temp;
```

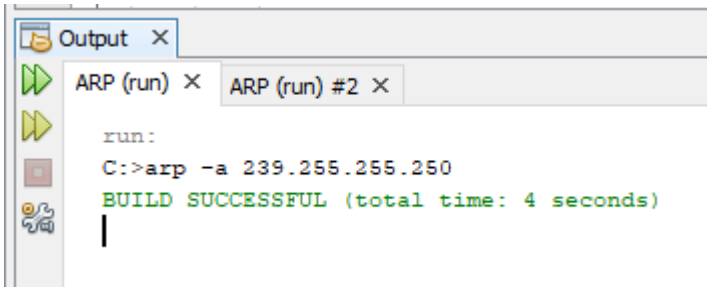
```
            temp = bin.read();
        }
        dout.writeUTF(output);
        dout.flush();
        dout.close();
        ss.close();
    } catch (Exception e) {
        System.out.println(e);
    }
  }
}
```

OUTPUT:

CLIENT SIDE:



SERVER SIDE:



RESULT:

Thus the simulation of Address Resolution Protocol(ARP) was successfully executed.

Ex.No.: 3B DATE: 22/09/21	REVERSE ADDRESS RESOLUTION PROTOCOL(RARP)

AIM:

To write a java program for simulating RARP protocols using UDP

ALGORITHM:

CLIENT SIDE:

1. Start the program
2. Using datagram sockets UDP function is established.
3. Get the MAC address to be converted into IP address.
4. Send this MAC address to server.
5. Server returns the IP address to client.
6. Stop the program

SERVER SIDE:

1. Start the program.
2. Server maintains the table in which IP and corresponding MAC addresses are stored.
3. Read the MAC address which is send by the client.
4. Map the IP address with its MAC address and return the IP address to client.
5. Stop the program

PROGRAM:

CLIENT SIDE:

```
import java.io.*;
import java.net.*;
public class Client {
   public static void main(String[] args) {
      try {
         DatagramSocket client = new DatagramSocket(1310);
         InetAddress addr = InetAddress.getByName("127.0.0.1");
```

```
            byte[] sendbyte = new byte[1024];
            byte[] receivebyte = new byte[1024];
            String mac = "01-00-5e-7f-ff-fa";
            sendbyte = mac.getBytes();
            System.out.println("sending mac address - "+mac+"to server");
            DatagramPacket sender = new DatagramPacket(sendbyte, sendbyte.length, addr, 1309);
            client.send(sender);
            DatagramPacket receiver = new DatagramPacket(receivebyte, receivebyte.length);
            client.receive(receiver);
            String str = new String(receiver.getData(), 0, receiver.getLength());
            String ip = str.trim();
            System.out.println("internet address of "+mac);
            System.out.println(str);
            client.close();

        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

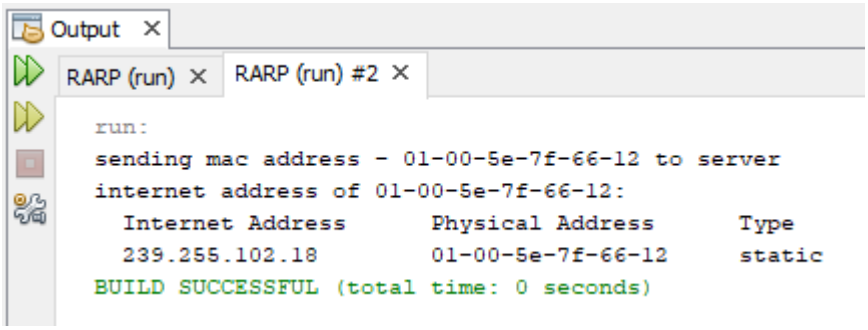
SERVER SIDE:

```
import java.io.*;
import java.net.*;
public class Server {
    public static void main(String[] args) {
        try {
            DatagramSocket server = new DatagramSocket(1309);
            InetAddress addr = InetAddress.getByName("127.0.0.1");
            byte[] sendbyte = new byte[1024];
            byte[] receivebyte = new byte[1024];
            DatagramPacket receiver = new DatagramPacket(receivebyte, receivebyte.length);
```

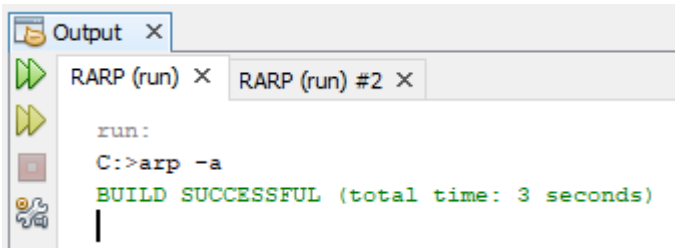
```
            server.receive(receiver);
            String str = new String(receiver.getData(), 0, receiver.getLength());
            String mac = str.trim();
            Runtime r = Runtime.getRuntime();
            System.out.println("C:>arp -a");
            Process p = r.exec("arp -a");
            BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream()));
            String line = "";
            String ip = "";
            while ((line = br.readLine()) != null) {
               if (line.contains(mac)) {
                  ip = "  Internet Address      Physical Address      Type\n" + line;
                  sendbyte = ip.getBytes();
                  break;
               }
            }
            if (line == null) {
               line = "Not Found";
            }
            DatagramPacket sender = new DatagramPacket(sendbyte, sendbyte.length, addr, 1310);
            server.send(sender);
            server.close();
         } catch (Exception e) {
            System.out.println(e);
         }
      }
   }
```

OUTPUT:

CLIENT SIDE:



SERVER SIDE:



TITLE	MARKS
OBSERVATION	
RECORD	
TOTAL	
SIGN	

RESULT:

Thus the simulation of Reverse Address Resolution Protocol(RARP) was successfully executed.

Ex.No.: 4 DATE: 30/09/2021	IMPLEMENTATION OF ERROR DETECTION AND CORRECTION TECHNIQUE

AIM:

To write a Java program to implement Error Detection and Correction Techniques.

ALGORITHM:

1.Start.

2.Get generator and data.

3.Encode it.

4.Transmit the encoded data.

5.At receiver side, decode it and check the data.

6.Stop.

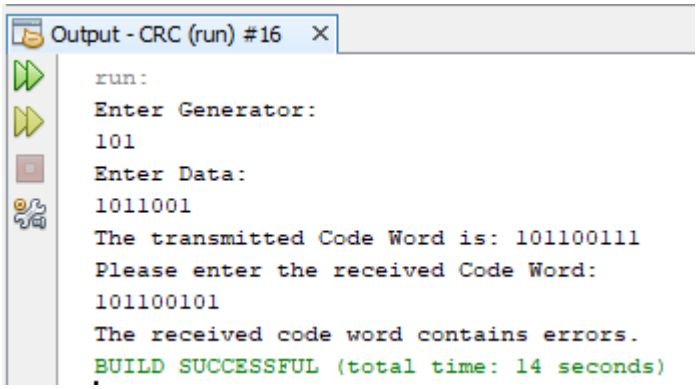
PROGRAM:

```
import java.io.*;

public class CRC {
   public static void main(String[] args) throws IOException {
      // TODO code application logic here
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter Generator:");
      String gen = br.readLine();
      System.out.println("Enter Data:");
      String data = br.readLine();
      String code = data;
      while (code.length() < (data.length() + gen.length() - 1)) {
         code = code + "0";
      }
      code = data + div(code, gen);
      System.out.println("The transmitted Code Word is: " + code);
      System.out.println("Please enter the received Code Word: ");
      String rec = br.readLine();
      if (Integer.parseInt(div(rec, gen)) == 0) {
         System.out.println("The received code word contains no errors.");
      } else {
         System.out.println("The received code word contains errors.");
      }
   }

   static String div(String code, String gen) {
      int pointer = gen.length();
      String result = code.substring(0, pointer);
```

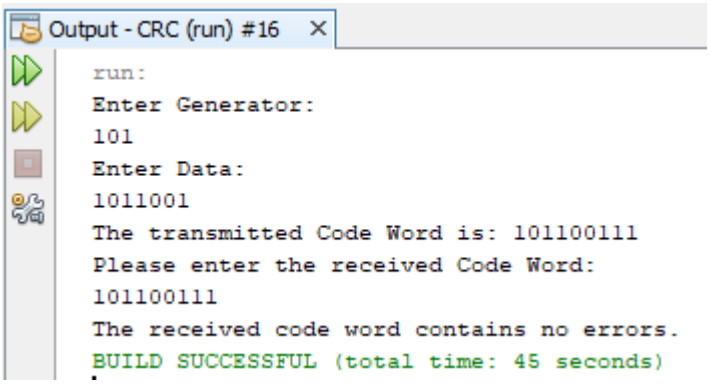
```
        String rem = "";
        for (int i = 0; i < gen.length(); i++) {
            if (result.charAt(i) == gen.charAt(i)) {
                rem += "0";
            } else {
                rem += "1";
            }
        }
        while (pointer < code.length()) {
            if (rem.charAt(0) == '0') {
                rem = rem.substring(1, rem.length());
                rem = rem + String.valueOf(code.charAt(pointer));
                pointer++;
            }
            result = rem;
            rem = "";
            if (result.charAt(0) == '0') {
                for (int i = 0; i < gen.length(); i++) {
                    if (result.charAt(i) == '1') {
                        rem += 1;
                    } else {
                        rem += 0;
                    }
                }
            } else {
                for (int i = 0; i < gen.length(); i++) {
                    if (result.charAt(i) == gen.charAt(i)) {
                        rem += "0";
                    } else {
                        rem += "1";
                    }
                }
            }
        }
        return rem.substring(1, rem.length());
    }
}
```

OUTPUT:

WITH ERROR:



WITHOUT ERROR:



TITLE	MARKS
OBSERVATION	
RECORD	
TOTAL	
SIGN	

RESULT:

Thus the Java program to implement Error Detection and Correction Techniques was successfully executed.

Ex. No: 5 DATE: 8/10/2021	DNS USING UDP SOCKET

AIM:

To simulate DNS using UDP transport protocol using java.

ALGORITHM:

CLIENT SIDE:

1. Start.
2. Using Datagram Sockets UDP function is established.
3. Get the host name to map IP address.
4. Send the host name to server.
5. Server return the IP address to client.
6. Print the IP address.
7. Stop.

SERVER SIDE:

1. Start.
2. Server maintains the table in which host name and corresponding IP address are stored.
3. Read the host name which is sent by the client.
4. Map the IP address with host name and return the IP address to the client.
5. Stop.

PROGRAM:

CLIENT SIDE:

```
import java.net.*;

import java.io.*;

public class Client {

   public static void main(String[] args) {

      try {

         DatagramSocket client = new DatagramSocket(1310);

         InetAddress addr = InetAddress.getByName("127.0.0.1");

         byte[] sendbyte = new byte[1024];

         byte[] receivebyte = new byte[1024];

         String host = "google.com";

         sendbyte = host.getBytes();

         System.out.println("sending host name - " + host + " to dns");
```

```
            System.out.println("host name: " + host);
            DatagramPacket sender = new DatagramPacket(sendbyte, sendbyte.length, addr,
1309);
            client.send(sender);
            DatagramPacket receiver = new DatagramPacket(receivebyte, receivebyte.length);
            client.receive(receiver);
            String str = new String(receiver.getData(), 0, receiver.getLength());
            String ip = str.trim();
            System.out.println("internet address: " + ip);
            client.close();

        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

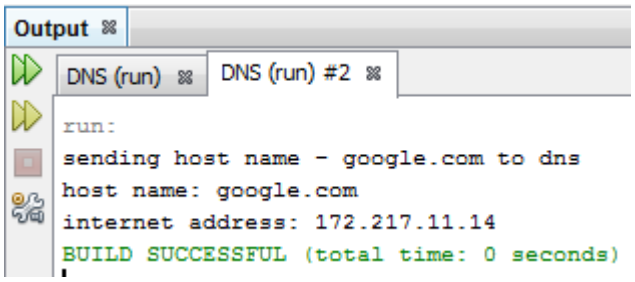
SERVERSIDE:

```
import java.net.*;
import java.io.*;
public class Server {
        public static void main(String[] args) {
        try {
            DatagramSocket server = new DatagramSocket(1309);
            InetAddress addr = InetAddress.getByName("127.0.0.1");
            byte[] sendbyte = new byte[1024];
            byte[] receivebyte = new byte[1024];
            String[] hosts = {"zoho.com", "gmail.com", "google.com", "facebook.com"};
            String[] ips = {"172.28.251.59", "172.217.11.5", "172.217.11.14", "31.13.71.36"};
            DatagramPacket receiver = new DatagramPacket(receivebyte, receivebyte.length);
            server.receive(receiver);
```

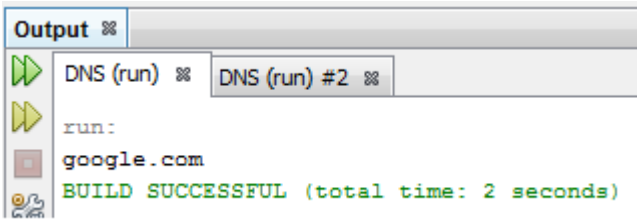
```
            String str = new String(receiver.getData(), 0, receiver.getLength());
            String host = str.trim();
            System.out.println(host);
            String ip = "not found";
            for (int i = 0; i < 4; i++) {
               if (hosts[i].equals(host)) {
                  ip = ips[i];
                  break;
               }
            }
            sendbyte = ip.getBytes();
            DatagramPacket sender = new DatagramPacket(sendbyte, sendbyte.length, addr,
1310);
            server.send(sender);
            server.close();
         } catch (Exception e) {
            System.out.println(e);
         }
      }
}
```

OUTPUT:

CLIENT SIDE:



SERVER SIDE:



TITLE	MARKS
OBSERVATION	
RECORD	
TOTAL	
SIGN	

RESULT:

Thus the DNS using UDP transport protocol using java was successfully executed.

Ex. No: 6A DATE: 22/10/2021	IMPLEMENTATION OF DISTANCE VECTOR ROUTING ALGORITHM

AIM:

To simulate and observe traffic route of a network using distance vector routing protocol.

ALGORITHM:

Input: Graph and a source vertex *src*

Output: Shortest distance to all vertices from *src*. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

1) This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.

2) This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.

Do following for each edge u-v

If dist[v] > dist[u] + weight of edge uv, then update dist[v] = dist[u] + weight of edge uv

3) This step reports if there is a negative weight cycle in graph.

Do following for each edge u-v

If dist[v] > dist[u] + weight of edge uv, then "Graph contains negative weight cycle"

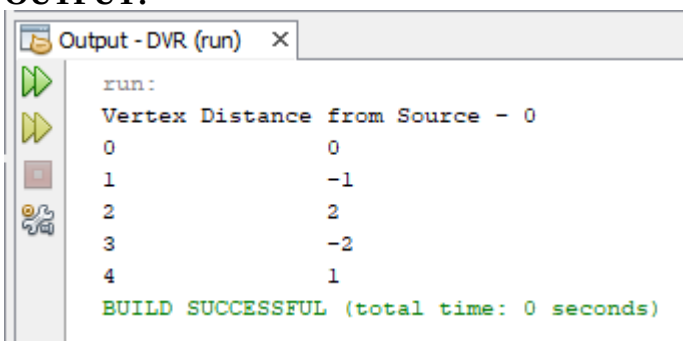
The idea of step 3 is, step 2 guarantees shortest distances if graph doesn't contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle.

PROGRAM:

```
public class DVR {
   static void BellmanFord(int graph[][], int V, int E, int src) {
      int[] dis = new int[V];
      for (int i = 0; i < V; i++) {
         dis[i] = Integer.MAX_VALUE;
      }
      dis[src] = 0;
      for (int i = 0; i < V - 1; i++) {
         for (int j = 0; j < E; j++) {
            if (dis[graph[j][0]] != Integer.MAX_VALUE && dis[graph[j][0]] + graph[j][2] <
dis[graph[j][1]]) {
               dis[graph[j][1]] = dis[graph[j][0]] + graph[j][2];
            }
         }
      }
      for (int i = 0; i < E; i++) {
         int x = graph[i][0];
         int y = graph[i][1];
         int weight = graph[i][2];
         if (dis[x] != Integer.MAX_VALUE && dis[x] + weight < dis[y]) {
            System.out.println("Graph contains negative" + " weight cycle");
         }
      }
      System.out.println("Vertex Distance from Source - " + src);
      for (int i = 0; i < V; i++) {
         System.out.println(i + "\t\t" + dis[i]);
```

```
        }
    }

    public static void main(String[] args) {
        // TODO code application logic here
        // TODO code application logic here
        int V = 5;
        int E = 8;
        int graph[][] = {{0, 1, -1}, {0, 2, 4},
        {1, 2, 3}, {1, 3, 2},
        {1, 4, 2}, {3, 2, 5},
        {3, 1, 1}, {4, 3, -3}};
        BellmanFord(graph, V, E, 0);
    }

}
```

OUTPUT:



RESULT:

Thus the simulation and observing traffic route of a network using distance vector routing protocol was successfully implemented.

Ex. No: 6B DATE: 22/10/2021	IMPLEMENTATION OF LINK STATE ROUTING ALGORITHM

AIM:

To simulate and observe traffic route of a network using distance vector routing protocol.

ALGORITHM:

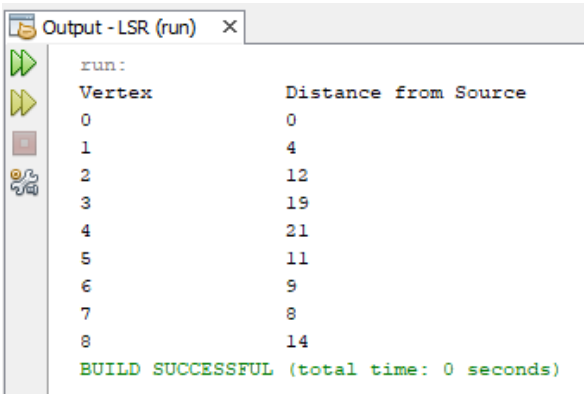
1) Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in the shortest-path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
3) While *sptSet* doesn't include all vertices
a) Pick a vertex u which is not there in *sptSet* and has a minimum distance value.
b) Include u to *sptSet*.
c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

PROGRAM:

```
public class LSR {
   static final int V = 9;
   int minDistance(int dist[], Boolean sptSet[]) {
      // Initialize min value
      int min = Integer.MAX_VALUE, min_index = -1;
      for (int v = 0; v < V; v++) {
         if (sptSet[v] == false && dist[v] <= min) {
            min = dist[v];
            min_index = v;
         }
      }
      return min_index;
   }
   void printSolution(int dist[]) {
      System.out.println("Vertex \t\t Distance from Source");
      for (int i = 0; i < V; i++) {
         System.out.println(i + " \t\t " + dist[i]);
      }
   }
   void dijkstra(int graph[][], int src) {
      int dist[] = new int[V];
      Boolean sptSet[] = new Boolean[V];
      for (int i = 0; i < V; i++) {
         dist[i] = Integer.MAX_VALUE;
         sptSet[i] = false;
      }
      dist[src] = 0;
      for (int count = 0; count < V - 1; count++) {
         int u = minDistance(dist, sptSet);
         sptSet[u] = true;
```

```
            for (int v = 0; v < V; v++) {
                if (!sptSet[v] && graph[u][v] != 0 && dist[u] != Integer.MAX_VALUE &&
dist[u] + graph[u][v] < dist[v]) {
                    dist[v] = dist[u] + graph[u][v];
                }
            }
        }
        printSolution(dist);
    }
    public static void main(String[] args) {
        int graph[][] = new int[][]{{0, 4, 0, 0, 0, 0, 0, 8, 0},
        {4, 0, 8, 0, 0, 0, 0, 11, 0},
        {0, 8, 0, 7, 0, 4, 0, 0, 2},
        {0, 0, 7, 0, 9, 14, 0, 0, 0},
        {0, 0, 0, 9, 0, 10, 0, 0, 0},
        {0, 0, 4, 14, 10, 0, 2, 0, 0},
        {0, 0, 0, 0, 0, 2, 0, 1, 6},
        {8, 11, 0, 0, 0, 0, 1, 0, 7},
        {0, 0, 2, 0, 0, 0, 6, 7, 0}};
        LSR t = new LSR();
        t.dijkstra(graph, 0);
    }
}
```

OUTPUT:



TITLE	MARKS
OBSERVATION	
RECORD	
TOTAL	
SIGN	

RESULT:

Thus the simulation and observing traffic route of a network using distance vector routing protocol was successfully implemented.

Ex. No: 7 DATE: 01/11/2021	HTTP WEB CLIENT PROGRAM TO DOWNLOAD A WEB PAGE USING TCP SOCKETS

AIM:

To download a webpage using TCP sockets.

ALGORITHM:

CLIENT SIDE:

1) Start the program.
2) Create a socket which binds the Ip address of server & the port address to acquire service.
3) After establishing connection send the url to server.
4) Open a file and store the received data into the file.
5) Close the socket.
6) End the program.

SERVER SIDE:

1) Start the program.
2) Create a server socket to activate the port address.
3) Create a socket for the server socket which accepts the connection.
4) After establishing connection receive url from client.
5) Download the content of the url received and send the data to client.
6) Close the socket.
7) End the program.

PROGRAM:

SERVER SIDE:

```
public class Server {
   public static void main(String[] args) throws SocketException {
      try {
         String line;
         ServerSocket ss = new ServerSocket(6666);
         Socket s = ss.accept();//establishes connection
         DataInputStream din = new DataInputStream(s.getInputStream());
         DataOutputStream dout = new DataOutputStream(s.getOutputStream());
         String str = (String) din.readUTF();
         URL url = new URL(str);
         System.out.println("url received");
         InputStream is = url.openStream();  // throws an IOException
         BufferedReader br = new BufferedReader(new InputStreamReader(is));
         System.out.println("sending web page contents to client");
         while ((line = br.readLine()) != null) {
            dout.writeUTF(line);
         }
         dout.close();
         din.close();
         is.close();
         br.close();
         s.close();
         ss.close();
      } catch (MalformedURLException mue) {
         mue.printStackTrace();
```

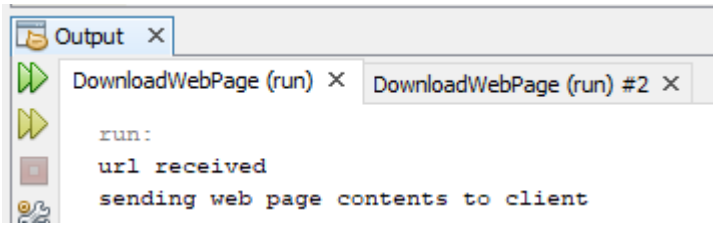
```
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

CLIENT SIDE:

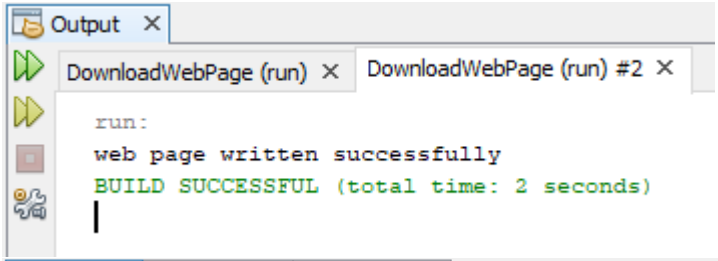
```
public class Client {
    public static void main(String[] args) throws SocketException {
        try {
            Socket s = new Socket("localhost", 6666);
            DataOutputStream dout = new DataOutputStream(s.getOutputStream());
            DataInputStream din = new DataInputStream(s.getInputStream());
            FileWriter fw = new
FileWriter("C:\\Users\\dharw\\Documents\\NetBeansProjects\\DownloadWebPage\\src\\down
loadwebpage\\web.html");
            dout.writeUTF("https://madurai.nic.in/");
            dout.flush();
            String str;
            do {
                str = (String) din.readUTF();
                fw.write(str);
                fw.write("\n");
            } while (!str.equals("</html>"));

            System.out.println("web page written successfully");
            fw.close();
            dout.close();
            din.close();
            s.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

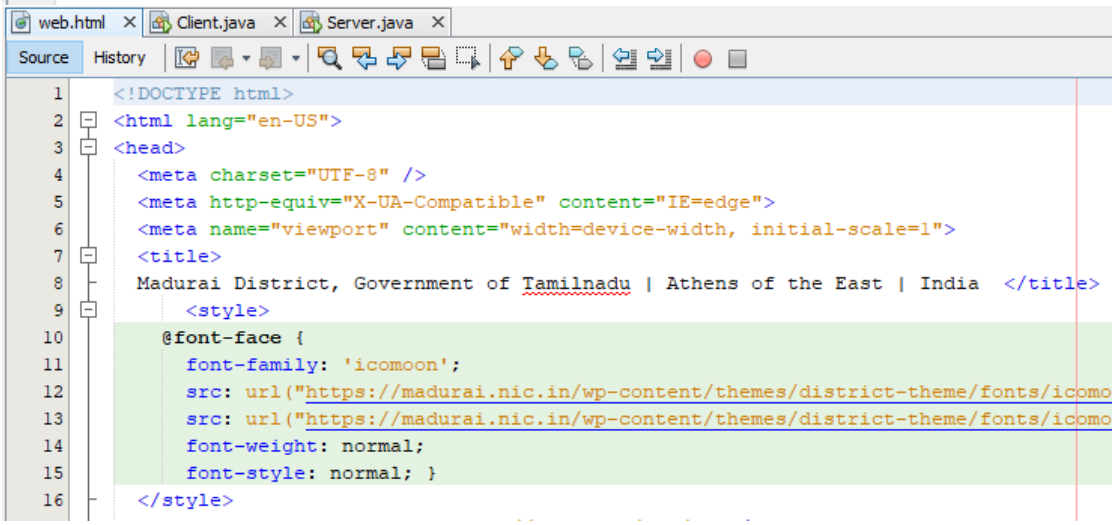
OUTPUT:

SERVER SIDE:



CLIENT SIDE:





TITLE	MARKS
OBSERVATION	
RECORD	
TOTAL	
SIGN	

RESULT:

Thus the HTTP web client program to download a webpage using TCP sockets was successfully implemented.

Ex.No: 8 DATE: 12/11/2021	SWITCH CONFIGURATION USING PACKET TRACER

AIM:

To establish a basic switch configuration between the end devices.

ALGORITHM:

1. Configure the end devices.
2. Establish connection from source to destination through switches.
3. Enable the switch.
4. Configure console password and vty password.

COMMANDS:

Switch>enable

Switch#configure terminal

Enter configuration commands, one per line. End with CNTL/Z.

Switch(config)#hostname cse

cse(config)#

cse (config)#line console 0

cse(config-line)#password lab1

cse (config-line)#login

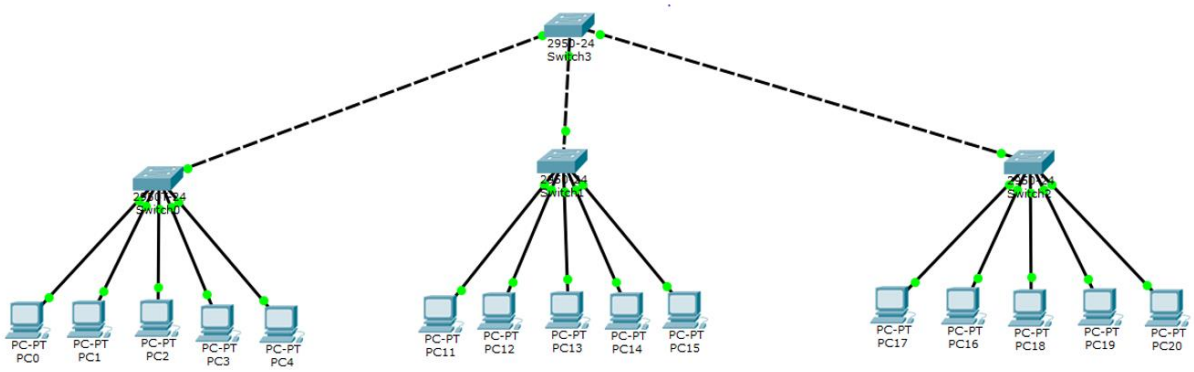
cse (config-line)#line vty 0 15

cse (config-line)#password lab1

cse (config-line)#login

cse (config-line)#exit

OUTPUT:



Observation	
Record	
Total	
Staff Signature	

RESULT:

Thus the switch configuration using packet tracer is performed and verified.

Ex.No: 9 DATE: 12/11/2021	ROUTER CONFIGURATION USING PACKET TRACER

AIM:

To establish a basic router configuration between the end devices.

ALGORITHM:

1. Configure the end devices.
2. Establish connection from source to destination through switches.
3. Click the router and enter the code configuration in cli tab.

COMMANDS:

ROUTER 1:

Router>enable

Router#configure terminal

Enter configuration commands, one per line. End with CNTL/Z.

Router(config)#interface serial 2/0

Router(config-if)#ip address 172.16.1.1 255.255.0.0

Router(config-if)#no shutdown

Router(config-if)#

%LINK-5-CHANGED: Interface Serial 2/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface serial0/0, changed state to up

Router(config-if)#interface serial 2/0

Router(config-if)#ip address 172.16.1.3 255.255.0.0

Router(config-if)#no shutdown

Router(config-if)#

%LINK-5-CHANGED: Interface serial 2/0, changed state to up

Router#

%SYS-5-CONFIG_I: Configured from console by console

ROUTER 2:

Router>enable

Router#configure terminal

Enter configuration commands, one per line. End with CNTL/Z.

Router(config)#interface serial 2/0

Router(config-if)#ip address 172.16.3.4 255.255.0.0

Router(config-if)#no shutdown

Router(config-if)#

%LINK-5-CHANGED: Interface serial 2/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface serial 2/0, changed state to up

Router(config)#interface serial 2/0

Router(config-if)#ip address 172.16.3.4 255.255.255.0

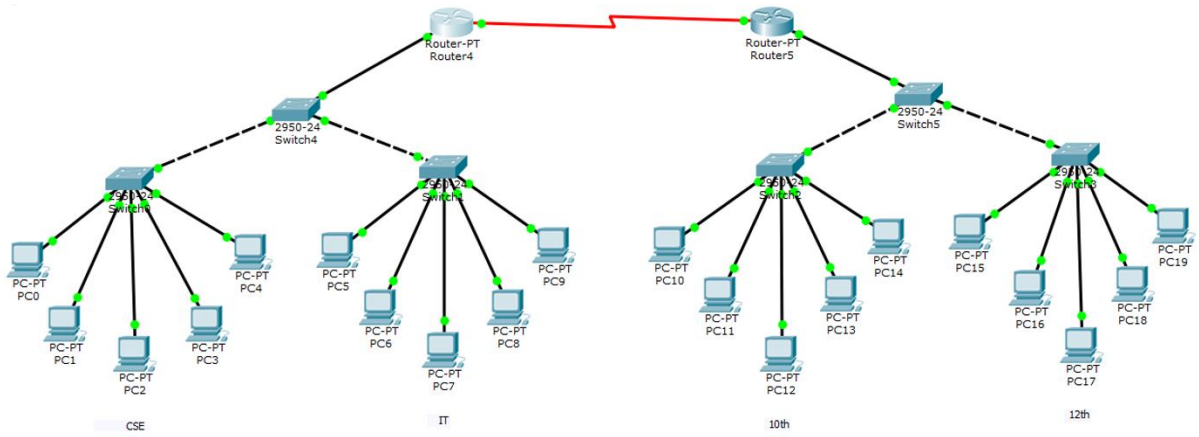
Router(config-if)#no shutdown

Router(config-if)#

%LINK-5-CHANGED: Interface serial 2/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface serial 2/0, changed state to up

OUTPUT:



Observation	
Record	
Total	
Staff Signature	

RESULT:

Thus the router configuration using packet tracer is performed and verified.

Ex.No: 10A DATE: 12/11/2021	OSPF CONFIGURATION USING PACKET TRACER

AIM:

To establish the OSPF (OPEN SHORTEST PATH FIRST) configuration between the end devices.

ALGORITHM:

1. Configure the end devices.
2. Configure connection from source to destination through routers
3. Enable the router configuration
4. Find the best path from source to destination.

COMMANDS:

ROUTER 1:

R1(config)#int fa 0/0

R1(config-if)#ip add 10.0.0.1 255.0.0.0

R1(config-if)#no shut

R1(config-if)#

R1(config-if)#int serial 0/0/0

R1(config-if)#ip add 20.0.0.1 255.0.0.0

R1(config-if)#no shut

ROUTER 2:

R2(config-if)#int fa0/0

R2(config-if)#ip add 30.0.0.1 255.0.0.0

R2(config-if)#no shut

R2(config-if)#

R2(config-if)#int serial0/0/0

R2(config-if)#ip address 20.0.0.2 255.0.0.0

R2(config-if)#no shut

PC1 IP add 10.0.0.2 Subnet mask 255.0.0.0 Default gateway 10.0.0.1

PC2 IP add 30.0.0.2 Subnet mask 255.0.0.0 Default gateway 30.0.0.1

Configure OSPF on the routers.

The configuration is pretty simple and requires only two major steps:

R1(config)#

R1(config)#router ospf 1

R1(config-router)#network 10.0.0.0 0.255.255.255 area 0

R1(config-router)#network 20.0.0.0 0.255.255.255 area 0

R2(config)#

R2(config)#router ospf 2

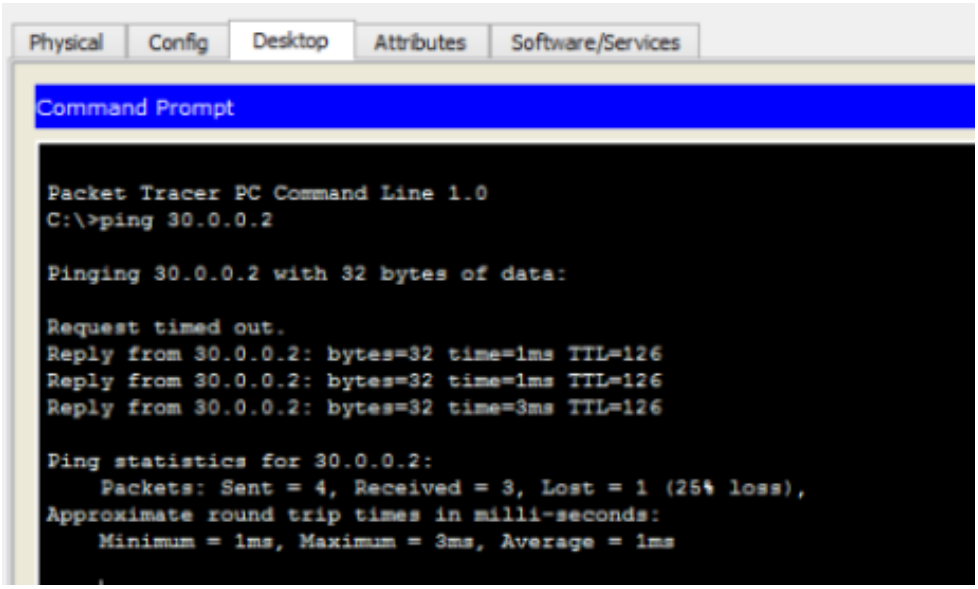
R2(config-router)#network 20.0.0.0 0.255.255.255 area 0

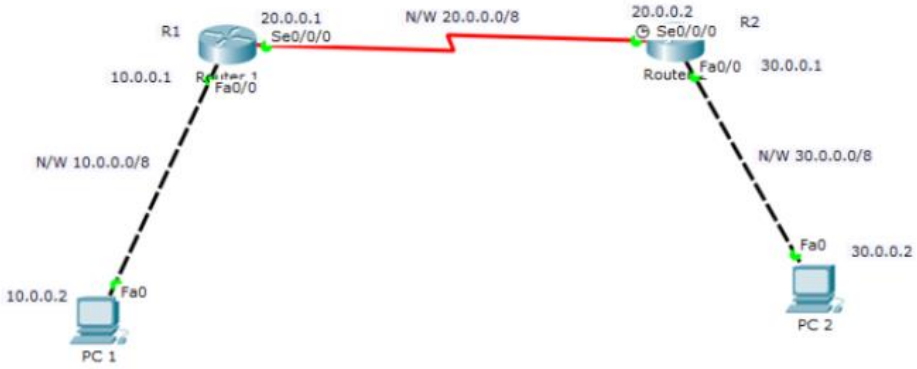
R2(config-router)#network 30.0.0.0 0.255.255.255 area 0

Verify OSPF configuration

```
R1#
R1#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address
Interface
30.0.0.1          0   FULL/  -        00:00:30    20.0.0.2
Serial0/0/0

R1#
R1#show ip route ospf
O    30.0.0.0 [110/65] via 20.0.0.2, 00:20:50, Serial0/0/0
```

OUTPUT:





RESULT:

Thus the OSPF configuration using packet tracer is established and verified.

Ex.No: 10B DATE: 12/11/2021	RIP CONFIGURATION USING PACKET TRACER

AIM:

To establish RIP (Routing Information Protocol) configuration between the end devices.

ALGORITHM:

1. Configure IP addresses on the PCs and the routers.
2. Configure the end devices.
3. Establish connection from source to destination through routers.
4. Enable the router configuration.
5. Find the best path from source to destination.

COMMANDS:

ROUTER 1:

R1(config)#int fa 0/0

R1(config-if)#ip add 10.0.0.1 255.0.0.0

R1(config-if)#no shut

R1(config-if)#

R1(config-if)#int serial 0/0/0

R1(config-if)#ip add 20.0.0.1 255.0.0.0

R1(config-if)#no shut

ROUTER 2:

R2(config-if)#int fa0/0

R2(config-if)#ip add 30.0.0.1 255.0.0.0

R2(config-if)#no shut

R2(config-if)#

R2(config-if)#int serial0/0/0

R2(config-if)#ip address 20.0.0.2 255.0.0.0

R2(config-if)#no shut

IP configuration on PCs:

PC1 IP add 10.0.0.2 Subnet mask 255.0.0.0 Default gateway 10.0.0.1

PC2 IP add 30.0.0.2 Subnet mask 255.0.0.0 Default gateway 30.0.0.1

Configure RIPV2 on the routers.

The configuration is pretty simple and requires only two major steps:

ROUTER 1

R1(config)#

R1(config-router)#router rip

R1(config-router)#version 2

R1(config-router)#network 10.0.0.0

R1(config-router)#network 20.0.0.0

ROUTER 2

R2(config)#

R1(config-router)#router rip

R1(config-router)#version 2

R2(config-router)#network 20.0.0.0

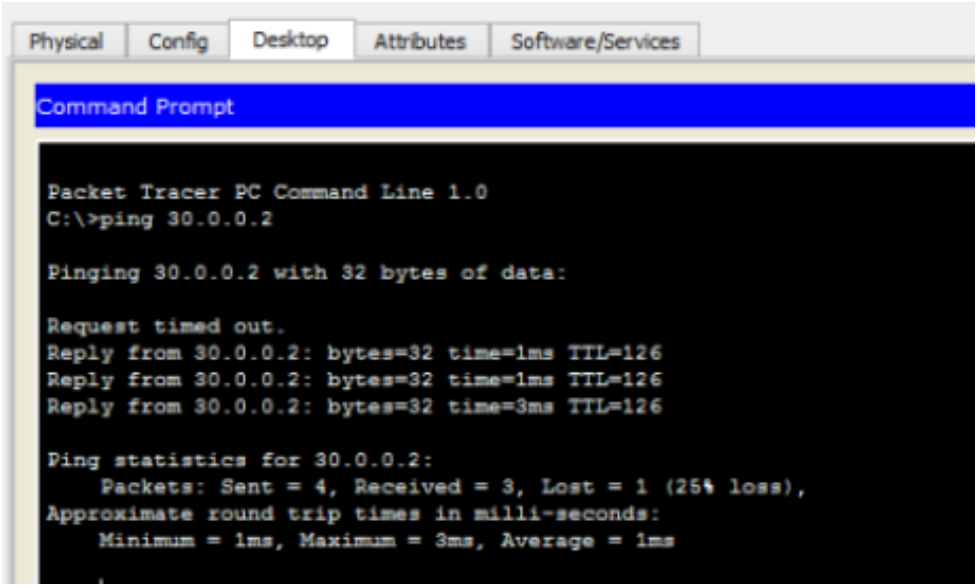
R2(config-router)#network 30.0.0.0

Verify RIP configuration

To verify that RIP is indeed advertising routes, we can use the show ip-route commands on R1.

```
R1#
R1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile,
B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter
area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external
type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E -
EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia -
IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

C    10.0.0.0/8 is directly connected, FastEthernet0/0
C    20.0.0.0/8 is directly connected, Serial0/0/0
R    30.0.0.0/8 [120/1] via 20.0.0.2, 00:00:17, Serial0/0/0
```

Output:



Observation	
Record	
Total	
Staff Signature	

Result:

Thus the RIP configuration using packet tracer is performed and verified.