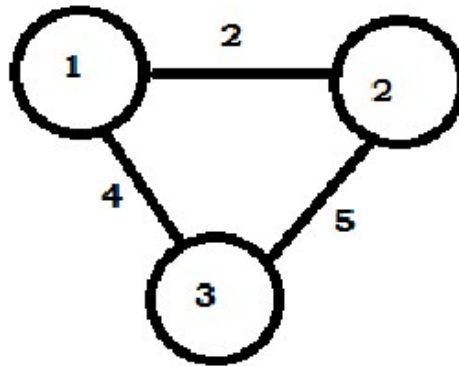


PROGRAM:

Node	Cost
1	0
2	2
3	4

Routing Table



Node	Cost
1	2
2	0
3	5

Routing Table

Node	Cost
1	4
2	5
3	0

Routing Table

HARDWARE REQUIREMENTS: Intel based Desktop PC:- RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C / Borland C.

THEORY:

Distance Vector Routing Algorithms calculate a best route to reach a destination based solely on distance. E.g. RIP. RIP calculates the reach ability based on hop count. It's different from link state algorithms which consider some other factors like bandwidth and other metrics to reach a destination. Distance vector routing algorithms are not preferable for complex networks and take longer to converge.

ALGORITHM/FLOWCHART:

Begin

Step1: Create struct node unsigned dist[20], unsigned from[20], rt[10]

Step2: initialize int dmat[20][20], n, i, j, k, count=0,

```

Step3: write "the number of nodes "
Step4: read the number of nodes "n"
Step5: write" the cost matrix : "
Step6: initialize i=0
Step7: repeat until i<n
Step8: increment i
Step9: initialize j=0
Step10: repeat Step(10-16)until j<n
Step11: increment j
Step12:read dmat[i][j]
Step13:intialize dmat[i][j]=0
Step14:intialize rt[i].dist[j]=dmat[i][j]
Step15:intialize rt[i].from[j]=j
Step16:end
Step17:start do loop Step (17-33)until
Step18:intilialize count =0
Step19:initialize i=0
Step20:repeat until i<n
Step21:increment i
Step22:initialize j=0
Step23:repeat until j<n
Step24:increment j
Step25:initialize k=0
Step26:repeat until k<n
Step27:increment k
Step28:if repeat Step(28-32) until rt[i].dist[j]>dmat[i][k]+rt[k].dist[j]
Step29:intialize rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j]
Step30:intialize rt[i].from[j]=k;
Step31:increment count
Step32:end if
Step33:end do stmt
Step34:while (count!=0)
Step35:initialize i=0
Step36:repeat Steps(36-44)until i<n
Step37:increment i
Step38:write ' state values for router',i+1
Step39:initialize j=0
Step40:repeat Steps ( 40-43)until j<n
Step41:increment j
Step42:write 'node %d via %d distance % ',j+1,rt[i].from[j]+1,rt[i].dist[j]
Step43:end
Step44:end
end

```

SOURCE CODE:

```
#include<stdio.h>
```

```

#include<conio.h>
struct node
{
unsigned dist[20];
unsigned from[20];
}rt[10];
int main()
{
int dmat[20][20];
int n,i,j,k,count=0;
clrscr();
printf("\n Enter the number of nodes : ");
scanf("%d",&n);
printf("Enter the cost matrix :\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
scanf("%d",&dmat[i][j]);
dmat[i][i]=0;
rt[i].dist[j]=dmat[i][j];
rt[i].from[j]=j;
}
do
{
count=0;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
for(k=0;k<n;k++)
if(rt[i].dist[j]>dmat[i][k]+rt[k].dist[j])
{
rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
rt[i].from[j]=k;
count++;
}
}while(count!=0);
for(i=0;i<n;i++)
{
printf("\nState value for router %d is \n",i+1);
for(j=0;j<n;j++)
{
printf("\nnode %d via %d Distance%d",j+1,rt[i].from[j]+1,rt[i].dist[j]);
}
}
printf("\n");
}

```

OUTPUT:

```
Turbo C++ IDE

Enter the number of nodes : 3
Enter the cost matrix :
0 2 4
2 0 5
4 5 0

State value for router 1 is
node 1 via 1 Distance0
node 2 via 2 Distance2
node 3 via 3 Distance4
State value for router 2 is
node 1 via 1 Distance2
node 2 via 2 Distance0
node 3 via 3 Distance5
State value for router 3 is
node 1 via 1 Distance4
node 2 via 2 Distance5
node 3 via 3 Distance0
```

Experiment:- 7

Aim:- Implement data encryption and data decryption

```

#include <stdio.h>
int main()
{
    int i, x;
    char str[100];
    printf("\nPlease enter a string:\t");
    gets(str);
    printf("\nPlease choose following options:\n");
    printf("1 = Encrypt the string.\n");
    printf("2 = Decrypt the string.\n");
    scanf("%d", &x);
    //using switch case statements
    switch(x)
    {
    case 1:
        for(i = 0; (i < 100 && str[i] != '\0'); i++)
            str[i] = str[i] + 3; //the key for encryption is 3 that is added to ASCII value
        printf("\nEncrypted string: %s\n", str);
        break;
    case 2:
        for(i = 0; (i < 100 && str[i] != '\0'); i++)
            str[i] = str[i] - 3; //the key for encryption is 3 that is subtracted to ASCII value

        printf("\nDecrypted string: %s\n", str);
        break;
    default:
        printf("\nError\n");
    }
    return 0;
}

```

OUTPUT:-

Encryption

```
"C:\Users\ICT\Google Drive\c project\code\encrypt-decrypt.exe"

Please enter a string:  hello

Please choose following options:
1 = Encrypt the string.
2 = Decrypt the string.
1

Encrypted string: khood

Process returned 0 (0x0)   execution time : 8.564 s
Press any key to continue.
```

#Decryption

```
"C:\Users\ICT\Google Drive\c project\code\encrypt-decrypt.exe"

Please enter a string:  khood

Please choose following options:
1 = Encrypt the string.
2 = Decrypt the string.
2

Decrypted string: hello

Process returned 0 (0x0)   execution time : 4.288 s
Press any key to continue.
```

Explanation

In the above program, we have used simple logic for encrypting and decrypting a given string by simply adding and subtracting the particular key from ASCII value.

Experiment:- 8

Aim:- Write a program for congestion control using Leaky bucket algorithm.

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int no_of_queries, storage, output_pkt_size;
    int input_pkt_size, bucket_size, size_left;
    // initial packets in the bucket
    storage = 0;
    // total no. of times bucket content is checked
    no_of_queries = 4;
    // total no. of packets that can
    // be accommodated in the bucket
    bucket_size = 10;
    // no. of packets that enters the bucket at a time
    input_pkt_size = 4;
    // no. of packets that exits the bucket at a time
    output_pkt_size = 1;
    for(int i = 0; i < no_of_queries; i++) //space left
    {
        size_left = bucket_size - storage;
        if(input_pkt_size <= size_left)
        {
            // update storage
            storage += input_pkt_size;
            printf("Buffer size= %d out of bucket size= %d\n", storage, bucket_size);
        }
        else
        {
            printf("Packet loss = %d\n", (input_pkt_size-(size_left)));
            // full size
            storage=bucket_size;
            printf("Buffer size= %d out of bucket size= %d\n", storage, bucket_size);
        }
        storage -= output_pkt_size;
    }
    return 0;
}
// This code is contributed by bunny09262002
```

Output :

Buffer size= 4 out of bucket size= 10

Buffer size= 7 out of bucket size= 10

Buffer size= 10 out of bucket size= 10

Packet loss = 3

Buffer size= 10 out of bucket size= 10

Difference between Leaky and Token buckets –

Experiment:- 9

Aim:- How to run Nmap scan

Nmap (<http://nmap.org>)

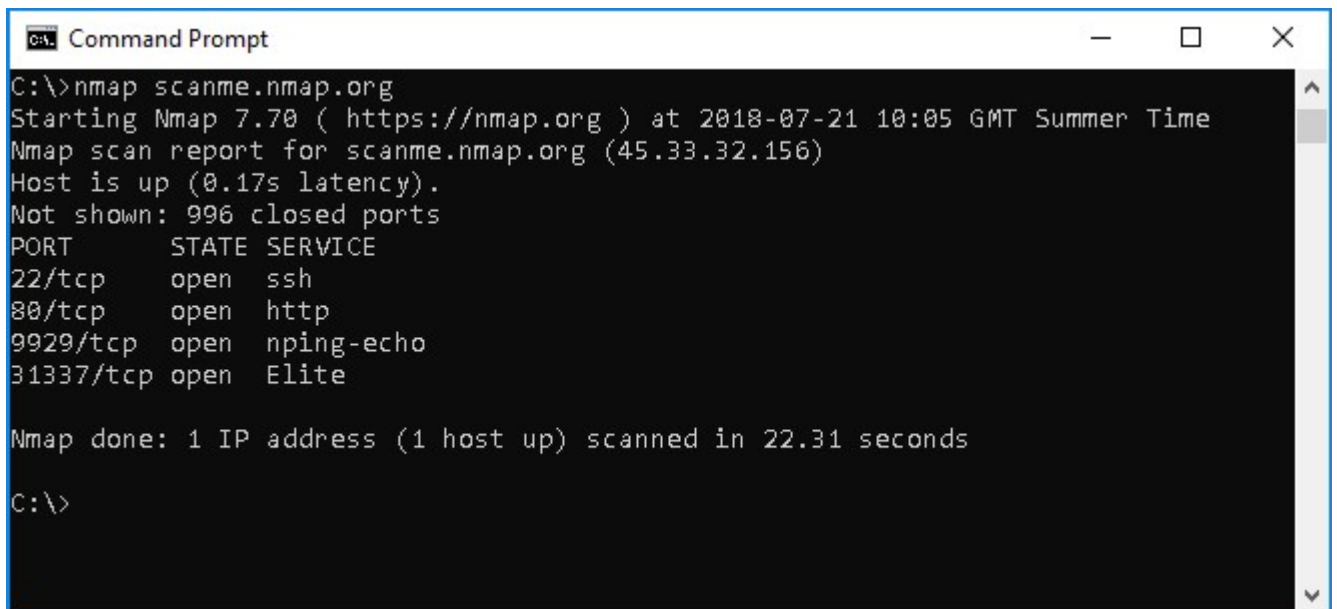
Nmap is a free, open source and multi-platform [network security](#) scanner used for network discovery and security auditing. Amongst other things, it allows you to create a network inventory, managing service upgrade schedules, monitor host or service uptime and scan for open ports and services on a host.

This post will focus on how to use Nmap to scan for open ports. Nmap can be extremely useful for helping you get to the root of the problem you are investigating, verify firewall rules or validate your routing tables are configured correctly.

To get started, download and install Nmap from the nmap.org website and then launch a command prompt.

Typing **nmap [hostname]** or **nmap [ip_address]** will initiate a default scan. A default scan uses 1000 common TCP ports and has Host Discovery enabled.

Host Discovery performs a check to see if the host is online. In a large IP range, this is useful for identifying only active or interesting hosts, rather than scanning every single port on every single IP in the range (a lot of which may not even be there).

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window has a black background with white text. The command entered is "C:\>nmap scanme.nmap.org". The output shows the Nmap version (7.70), the target IP (45.33.32.156), and a list of open ports (22/tcp, 80/tcp, 9929/tcp, 31337/tcp) with their corresponding services (ssh, http, nping-echo, Elite). The scan took 22.31 seconds to complete.

```
C:\>nmap scanme.nmap.org
Starting Nmap 7.70 ( https://nmap.org ) at 2018-07-21 10:05 GMT Summer Time
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.17s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
9929/tcp   open  nping-echo
31337/tcp  open  Elite

Nmap done: 1 IP address (1 host up) scanned in 22.31 seconds
C:\>
```

Note: nmap.scanme.org is a server the NMAP team spun up to allow you to test tool functionality.

When the scan is complete, you should see an Nmap scan report similar to the one shown in the image above. This confirms Nmap is installed and operating correctly.

You will notice the information returned is PORT | STATE | SERVICE. Before we take a deeper dive into the commands, it would be valuable to know what the different ‘STATES’ mean. The [Nmap Reference Guide](#) provides a pretty comprehensive explanation, but I’ll give you a brief summary here.

STATE	Description
Open	The target port actively responds to TCP/UDP/SCTP requests.
Closed	The target port is active but not listening.
Filtered	A firewall or packet filtering device is preventing the port state being returned.
Unfiltered	The target port is reachable but Nmap cannot determine if it is open or closed.
Open/Filtered	Nmap cannot determine if the target port is open or filtered.
Closed/Filtered	Nmap cannot determine if the target port is closed or filtered.

Let us now look at some commands we can use for scanning open ports.

Nmap Port Scanning Commands

The “–open” parameter

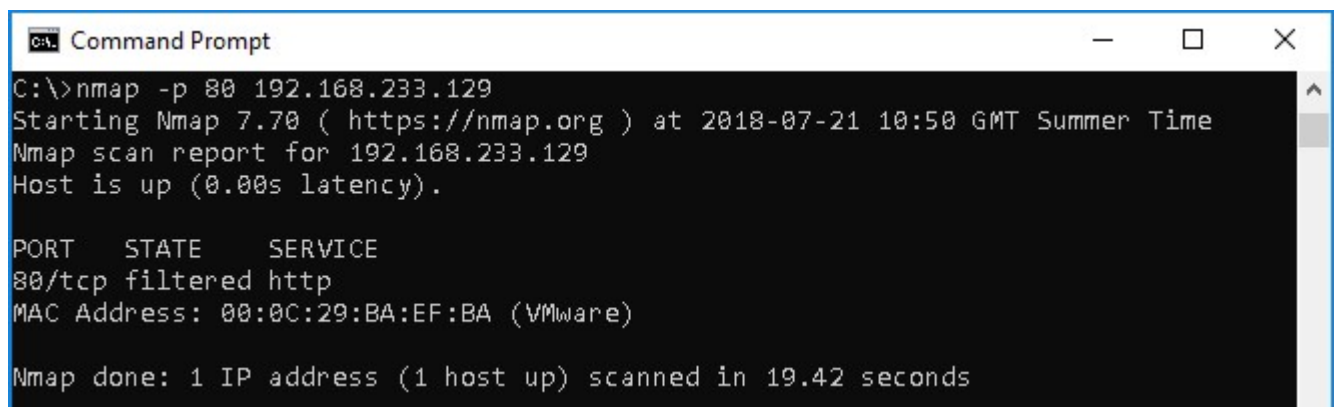
In any of the commands below, you can specify the “–open” parameter in your Nmap command to have Nmap only show you ports with an “Open” state.

`nmap –open [ip_address]`

Scanning a single port

`nmap -p 80 [ip_address]`

This command will initiate a default scan against the target host and look for port 80.



```
Command Prompt
C:\>nmap -p 80 192.168.233.129
Starting Nmap 7.70 ( https://nmap.org ) at 2018-07-21 10:50 GMT Summer Time
Nmap scan report for 192.168.233.129
Host is up (0.00s latency).

PORT      STATE      SERVICE
80/tcp    filtered  http
MAC Address: 00:0C:29:BA:EF:BA (VMware)

Nmap done: 1 IP address (1 host up) scanned in 19.42 seconds
```

Scanning a specific range of ports

`nmap -p 1-200 [ip_address]`

This command will initiate a default scan against the target host and look for ports between the range of 1-200.

```
Command Prompt
C:\>nmap -p 1-200 192.168.233.129
Starting Nmap 7.70 ( https://nmap.org ) at 2018-07-21 12:09 GMT Summer Time
Nmap scan report for 192.168.233.129
Host is up (0.0011s latency).
Not shown: 198 filtered ports
PORT      STATE SERVICE
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
MAC Address: 00:0C:29:BA:EF:BA (VMware)

Nmap done: 1 IP address (1 host up) scanned in 22.30 seconds
```

Scanning the entire port range

`nmap -p- [ip_address]`

This command will initiate a scan against the target host looking for all ports (1-65535).

```
Command Prompt
C:\>nmap -p- 192.168.233.129
Starting Nmap 7.70 ( https://nmap.org ) at 2018-07-21 12:32 GMT Summer Time
Nmap scan report for 192.168.233.129
Host is up (0.00086s latency).
Not shown: 65530 filtered ports
PORT      STATE SERVICE
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds
49154/tcp  open  unknown
49157/tcp  open  unknown
MAC Address: 00:0C:29:BA:EF:BA (VMware)

Nmap done: 1 IP address (1 host up) scanned in 124.12 seconds
```

Scanning the top 100 ports (fast scan)

`nmap -F [ip_address]`

This command will initiate a fast scan against the target host looking only for the top 100 common TCP ports.

```
Command Prompt
C:\>nmap -F 192.168.233.128
Starting Nmap 7.70 ( https://nmap.org ) at 2018-07-21 12:44 GMT Summer Time
Nmap scan report for 192.168.233.128
Host is up (0.00093s latency).
Not shown: 96 filtered ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
5357/tcp  open  wsapi
MAC Address: 00:0C:29:A2:6A:81 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 20.81 seconds
```

Scanning multiple TCP/UDP ports

`nmap -p U:53,67-68,T:21-25,80,135 [ip_address]`

This command will initiate a scan against the target host looking only for specified UDP and TCP ports.

```
Command Prompt
C:\>nmap -p U:53,67-68,T:21-25,80,135 192.168.233.128
Starting Nmap 7.70 ( https://nmap.org ) at 2018-07-21 13:11 GMT Summer Time
Nmap scan report for 192.168.233.128
Host is up (0.0011s latency).

PORT      STATE SERVICE
21/tcp    filtered ftp
22/tcp    filtered ssh
23/tcp    filtered telnet
24/tcp    filtered priv-mail
25/tcp    filtered smtp
80/tcp    filtered http
135/tcp   open  msrpc
MAC Address: 00:0C:29:A2:6A:81 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 22.87 seconds
```

Scanning for specific service names

`nmap -p http,ssh,msrpc,microsoft-ds [ip_address]`

This command will initiate a scan against the target host looking for ports associated with specified service names.

```
Command Prompt
C:\>nmap -p U:53,67-68,T:21-25,80,135 192.168.233.128
Starting Nmap 7.70 ( https://nmap.org ) at 2018-07-21 13:11 GMT Summer Time
Nmap scan report for 192.168.233.128
Host is up (0.0011s latency).

PORT      STATE      SERVICE
21/tcp    filtered  ftp
22/tcp    filtered  ssh
23/tcp    filtered  telnet
24/tcp    filtered  priv-mail
25/tcp    filtered  smtp
80/tcp    filtered  http
135/tcp   open      msrpc
MAC Address: 00:0C:29:A2:6A:81 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 22.87 seconds
```

Experiment:10

10. Operating system Detection using Nmap.

Sometimes on a network it is beneficial to know the Operating System (OS) of a machine. Accessing a system is easier when you know the OS because you can specifically search the Internet for known security holes in the OS. Granted, security holes are usually patched quickly, but you need to know when a security hole exists.

Scanning your own network to detect the OS types can help you to see what a hacker will be able to see about your network.

OS Detection Database

NMAP has a database which is installed when you install NMAP. The database is used when doing OS detection, but it is not automatically updated.

The database is located at '/usr/share/nmap/nmap-os-db'. The easiest way to manage an update is first to look at the database version number. Open the file in a text editor and the version number is usually listed on the second line. The second line of my database is '# \$Id: nmap-os-db 35407 2015-11-10 04:26:26Z dmiller \$'. The database version for this file is 35407.

To look on the Internet for an updated version go to '<https://svn.nmap.org/nmap>' as shown in Figure 1.

- Revision 36736: /nmap

- ..
- [.gitignore](#)
- [.travis.yml](#)
- [BSDmakefile](#)
- [CHANGELOG](#)
- [CONTRIBUTING.md](#)
- [COPYING](#)
- [FPENGINE](#)

FIGURE 1

Here you can see that the version number is 36736. This seems like quite an update compared to what is currently on my system. The database definitely needs to be updated for proper OS Detection of newer systems.

It may be a good idea to keep the older database version. Since my current version is 35407 I will perform the following command in a Terminal:

```
sudo mv /usr/share/nmap/nmap-os-db /usr/share/nmap/nmap-os-db-35407
```

The database is ‘moved’ or ‘renamed’ to include the version number. The next step is to download the new version from the website. Perform the following commands in a Terminal:

```
cd /usr/share/nmap  
sudo su  
wget https://svn.nmap.org/nmap/nmap-os-db
```

The new database should be downloaded and ready to go, but you should add the version number. As you saw in Figure 1 the version number is 36736. Use a text editor to open the database and on the second line add the version number. By adding the version number it will be easier to check later if the version number has changed. When the version number has changed you can update the database and add the version number so you are prepared when checking for updates again.

OS Detection Process

Before we get into the actual command and performing an OS Detection we should cover some details about what is happening during this scan.

There are five separate probes being performed. Each probe may consist of one or more packets. The response to each packet by the target system helps to determine the OS type.

The five different probes are:

1. Sequence Generation
2. ICMP Echo
3. TCP Explicit Congestion Notification
4. TCP
5. UDP

Now we can look at these individually to see what they are doing.

Sequence Generation

The Sequence Generation Probe consists of six packets. The six packets are sent 100 ms apart and are all TCP SYN packets.

The result of each TCP SYN packet will help NMAP determine the OS type.

ICMP Echo

Two ICMP Request packets are sent to the target system with varying settings in the packet.

The resulting responses will help verify the OS type by NMAP.

TCP Explicit Congestion Notification

When a lot of packets are being generated and passing through a router causing it to be burdened is known as congestion. The result is that systems slow down to reduce congestion so the router is not dropping packets.

The packet being sent is only to get a response from the target system. Specific values returned are used to determine the specific OS since each OS handles the packets in different ways.

TCP

Six packets are sent during this probe.

Some packets are sent to open or closed ports with specific packet settings. Again, the results will vary depending on the target OS.

The TCP Packets are all sent with varying flags as follows:

1. no flags
2. SYN, FIN, URG and PSH
3. ACK
4. SYN
5. ACK
6. FIN, PSH, and URG

UDP

This probe consists of a single packet sent to a closed port.

If the port used on the target system is closed and an ICMP Port Unreachable message is returned then there is no Firewall.

NMAP OS Detection Command

Now we need to run the actual command to perform an OS Detection. If you have read any of the other of my NMAP articles then it is best not to perform a PING. To skip the PING we use the parameter '-Pn'. To see the extra information we may require you should use the '-v' parameter for adding verbosity. Specifically to get the OS Detection the parameter '-O' is needed.

For the command to complete properly and perform the TCP SYN Scan you need to perform the command as ROOT. In my case, I will perform the scan on one system only and not the whole

network so the command would be:

```
sudo nmap -v -Pn -O 192.168.0.63
```

The results of the scan are shown in Figure 2. The scan shows that there are seven open ports using a SYN Stealth Scan.

```
jarret@jarret-PC ~
File Edit View Search Terminal Help
jarret@Jarret-PC ~ $ sudo nmap -v -Pn -O 192.168.0.63

Starting Nmap 7.01 ( https://nmap.org ) at 2017-04-23 16:29 EDT
Warning: File ./nmap-services exists, but Nmap is using /usr/bin/../share/nmap/nmap-services for security and consistency
reasons.  set NMAPDIR=. to give priority to files in your local directory (may affect the other data files too).
Initiating ARP Ping Scan at 16:29
Scanning 192.168.0.63 [1 port]
Completed ARP Ping Scan at 16:29, 0.23s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 16:29
Completed Parallel DNS resolution of 1 host. at 16:29, 0.03s elapsed
Initiating SYN Stealth Scan at 16:29
Scanning 192.168.0.63 [1000 ports]
Discovered open port 139/tcp on 192.168.0.63
Discovered open port 111/tcp on 192.168.0.63
Discovered open port 22/tcp on 192.168.0.63
Discovered open port 445/tcp on 192.168.0.63
Discovered open port 21/tcp on 192.168.0.63
Discovered open port 54045/tcp on 192.168.0.63
Discovered open port 2049/tcp on 192.168.0.63
Completed SYN Stealth Scan at 16:30, 1.23s elapsed (1000 total ports)
Initiating OS detection (try #1) against 192.168.0.63
Nmap scan report for 192.168.0.63
Host is up (0.00027s latency).
Not shown: 993 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
2049/tcp  open  nfs
54045/tcp open  unknown
MAC Address: 00:1E:4F:9F:DF:7F (Dell)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.6
Uptime guess: 0.324 days (since Sun Apr 23 08:43:32 2017)
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=255 (Good luck!)
IP ID Sequence Generation: All zeros
```

FIGURE 2

The open ports are:

1. 21/tcp ftp
2. 22/tcp ssh
3. 111/tcp rpcbind
4. 139/tcp netbios-ssn
5. 445/tcp microsoft-ds
6. 2049/tcp nfs
7. 54045/tcp unknown

The MAC Address of the system is 00:1E:4F:9FF:7F.

The final portion shows the OS type:

Device type: general purpose

Running: Linux 3.X\4.X

OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4

OS details: Linux 3.2 - 4.6

Uptime guess: 0.324 days (since Sun Apr 23 08:43:32 2017)

The system is running Ubuntu Server 16.10 and the Linux Kernel is 4.8. The uptime is guessed correctly.

I performed a second test against another system. The results are shown in Figure 3.

```
jarret@Jarret-PC ~
File Edit View Search Terminal Help
jarret@Jarret-PC ~ $ sudo nmap -v -Pn -O 192.168.0.201

Starting Nmap 7.01 ( https://nmap.org ) at 2017-04-23 16:42 EDT
Warning: File ./nmap-services exists, but Nmap is using /usr/bin/./share/nmap/nmap-services for security and consistency
reasons. set NMAPDIR=. to give priority to files in your local directory (may affect the other data files too).
Initiating ARP Ping Scan at 16:42
Scanning 192.168.0.201 [1 port]
Completed ARP Ping Scan at 16:42, 0.21s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 16:42
Completed Parallel DNS resolution of 1 host. at 16:42, 0.03s elapsed
Initiating SYN Stealth Scan at 16:42
Scanning 192.168.0.201 [1000 ports]
Discovered open port 445/tcp on 192.168.0.201
Discovered open port 139/tcp on 192.168.0.201
Completed SYN Stealth Scan at 16:42, 4.73s elapsed (1000 total ports)
Initiating OS detection (try #1) against 192.168.0.201
Nmap scan report for 192.168.0.201
Host is up (0.0038s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
MAC Address: 00:0B:DB:40:55:14 (Dell)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running: Microsoft Windows 2000|XP
OS CPE: cpe:/o:microsoft:windows_2000::sp4 cpe:/o:microsoft:windows_xp::sp2 cpe:/o:microsoft:windows_xp::sp3
OS details: Microsoft Windows 2000 SP4, Microsoft Windows XP SP2 or SP3
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=258 (Good luck!)
IP ID Sequence Generation: Incremental

Read data files from: /usr/bin/./share/nmap
OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.15 seconds
Raw packets sent: 2045 (92.508KB) | Rcvd: 25 (2.176KB)
jarret@Jarret-PC ~ $
```

FIGURE 3

In this scan different ports are open. The system is guessed to be ‘Microsoft Windows 2000|XP’. It is Windows XP running SP3.

Port Sniffer Results.

Let’s look at what is going on in the background of the first scan shown in Figure 2.

Initially NMAP is performing a TCP Stealth Scan. In this instance the OS Detection Probes start at Packet 2032 in Figure 4.

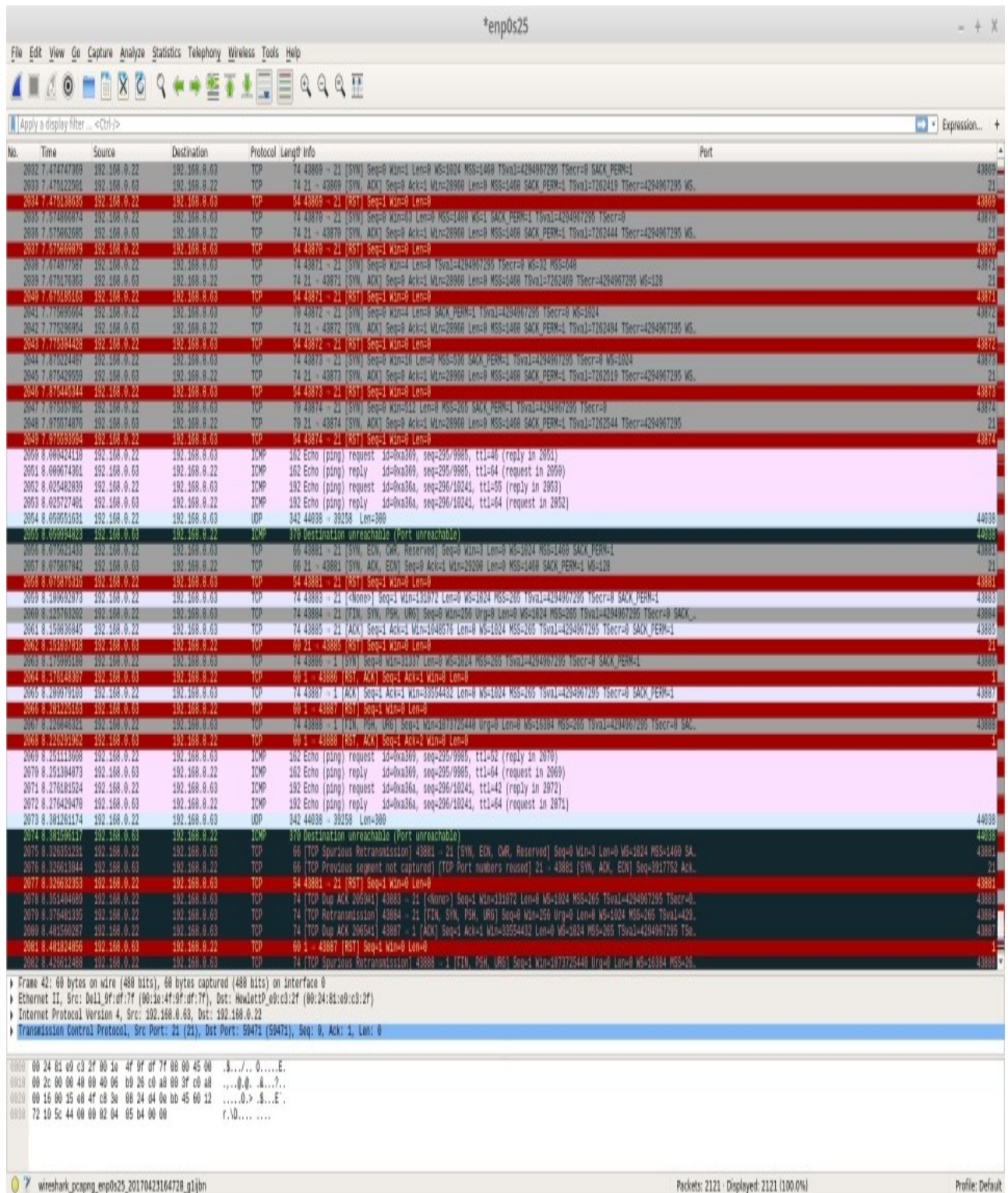


FIGURE 4

The Sequence Generation starts at Packet 2032/2033 and the sixth one is at Packet 2047/2048. Notice that each one is sent twice and the next packet in the probe is sent 100 ms later.

The ICMP Packets are sent at 2050-2053. The two packets are duplicated making a total of four.

Packets 2056-2057 are the TCP Explicit Congestion Notification packets.

The six probes for the TCP are on lines 2059, 2060, 2061, 2063, 2065 and 2067.

The last probe for UDP is line 2073.

These are the probes that are used to determine the OS of the target system.

I hope this helps you to understand how to update the NMAP OS Detection Database and perform a scan on systems. Be aware that you can run the scan on systems which are not on your network, so the scan can be performed on systems on the Internet.

I work at home and I have four children at home: Eilly, Alyse, Morgan and Grant. Three kids old enough to live on their own: Devyn, Logan and Caleb.

I had a book come out called **EPUB: From the Ground Up**.

You can go to Amazon or Barnes and Noble to order a copy.

You can contact me through **Linux.org** or at jarretbuse@hotmail.com.