Scalable, Low Fidelty, Network-Stack Emulation

Craig Ian Koorn

***Abstract*—Software-based routing simulators which are capable of operating at scale are in short supply. NKM, a Network Kernel Module, is an existing system which meets these requirements through mitigating the overhead introduced from user-space to kernel context switching; as well as by utilising the resources of external systems. However, on its own it, the system offers little interaction to its users. This research enhances the fidelity of NKM through demonstrating a level of interaction previously unavailable by providing the capability to allocate a minimalistic network-stack to each of its nodes. NKM, having been extended, could capably instantiate a network in which its nodes exhibited network-stack behaviour realistic enough to replicate common responses to ping, traceroute, and nmap.**

I. Introduction

With the prominence of the Internet in modern times, the world has come to rely on the proliferation of the many networks and protocols which together form what is known as the the Internet today. The burgeoning demand for both new and existing Internet and network services has led to many organisations investing in these technologies; be it for the development of new technologies; or technologies in support of existing organisational functions [1]. For this reason, understanding these technologies forms a critical role within any such organisation, and often requires a substantial investment of capital and/or time.

Simulating these networks and their protocols allows for mitigating some of the costs associated with the development of new services, or software that makes use of these services. It allows for a better understanding of what is currently happening as well as what is likely to happen going forward. This is both applicable in Educational institutions where hands-on, practical experience is tantamount to an effective education; as well as in business contexts where the ramifications of going live without sufficient prior testing can be costly.

The need to purchase expensive hardware can, in some cases, often be omitted through instead utilising a software component which provides a practical means for testing and training. However, readily available software which meets these requirements at scale is in short supply. In lieu of this necessity, [2] presents NKM (a Network Kernel Module) — a large-scale network routing simulator. Whilst NKM has proven effective in its ability to simulate vast and intricate network topologies, it requires the presence of physical systems to provide meaningful interaction.

In this regard, NKM can be enhanced, so as to not only facilitate the process of simulating large networks, but to also cater to the simulation of the hosts which reside therein. This addition, coupled with its ability to provide a level of interaction previously unavailable within the system itself, may further assist in alleviating financial costs. Furthermore, through representing these hosts virtually, the need to setup and assign dedicated hardware to this purpose diminishes. Not only does this assist in freeing up workspace, it too allows for representing these hosts in a manner which is easily modifiable and reproducible.

This extension aims to further enhance NKM, so as to not only add value to the applications for which it is already well-suited, but to advocate its use for functions it previously may not have been well positioned to accommodate. This research aims investigate the viability of this extension so as to create a well-rounded, realistic, feasible, large scale network routing simulator in support of both academic and business functions.

II. Related Work

The work in [2] presents a network routing simulator which has been implemented as a Linux kernel

module in an attempt to reduce the overhead incurred through kernel/user-space context switching. NKM has been designed with a high degree of scalability in mind, a goal it accomplishes utilising only commodity hardware. Moreover, NKM implements features which allow it to behave as a real network would. These features include delay, packet loss, packet mangling, and network jitter. These features, and the simulators ability to scale, make for an excellent platform to which host behaviour can be added. However, as this research focuses primarily on the network-stack behaviour pertaining to real hosts, NKM is discussed only in brevity. The interested reader is referred to [2] in which NKM is presented in detail. The sections which follow provide insight into how network host behaviour might be characterised so as to position define the properties a simulated node need exhibit in order to adequately beguile the networking tools used by professionals today.

A. Node Personalities

Information gathering and analysis is the initial stage in any information security audit; one which is essential in facilitating an effective vulnerability assessment. Put simply, the process consists of collecting as much information about the target system or network as possible. Typically, this would first involve the identification of all target systems which are publicly reachable; and proceeded by creating a network map of these systems [3, 4].

With this information in hand, one is positioned to begin extracting some of the more pertinent details belonging to each of the systems on the target network; a process referred to as fingerprinting. There exist subtleties in the manner with which each operating system uniquely implements its network stack. This, in conjunction with the services running on a host — and the information disclosed by these services — makes it possible for automated tools to infer, amongst other things, the underlying operating system of a target host [5].

1) Fingerprinting Techniques: There are several methods utilised by tools which together allow them to make accurate deductions. This topic will be covered in brevity and only some of the more popular methods will be mentioned, the first of which is banner grabbing.

Banner grabbing consists of either: inspecting the banner which is displayed when connecting to a service, such HTTP, FTP, SMTP, and others; or, where possible, by downloading a binary and determining the architecture for which it has been built [6]. Banner grabbing is not as reliable a method as some of the newer techniques, as it relies on the presence of services which disclose this information; however, it is also not so complex as to exclude human assessors from performing the task manually.

More advanced techniques rely on a method referred to as stack querying; a process which involves transmitting packets to the network stack of a remote host and then analysing the responses [6]. Initially this was done with only TCP segments, but later encompassed both ICMP and UDP response analysis mechanisms. These methods are particularly useful where response behaviour is not clearly outlined by the RFCs [7]. In such cases, the assumptions which were to be made in the implementation of an operating system's network stack, were done so at the discretion of the vendor. This allows for host architecture inferences to be made, based on the manner with which a host responds to such requests. In addition, many vendors have incorporated proprietary extensions into their distributions which further facilitates this process.

Another popular method is Initial Sequence Number (ISN) analysis. This method exploits the differences in the implementation of the random number generators used by each operating system's network stack. If, over a sufficient number of tests, a tool can identify the method being used in generating the ISNs, it can provide the user with insight into what the underlying host architecture might be [7].

A more recent method still, is temporal response analysis. As with many other methods, this technique relies on the TCP protocol; more specifically the way it behaves in meetings its requirement of providing a guaranteed delivery of service. Temporal response analysis works by monitoring the time it takes a host to again begin the retransmission of a packet once a retransmission timeout (RTO) has occurred [6].

2) Nmap: The aforementioned methods, amongst others, are used in conjunction with one another in order try and gain enough information about the behavioural characteristics of a target system, so that it is possible to deduce its operating system. One such tool which automates this process, and the *de facto* tool in network reconnaissance, is nmap[1].

In addition to providing operating system fingerprinting, nmap also provides: host discovery and detection services; service identification — and versioning thereof; packet filter/firewall policy information; and more.

Nmap fashions a total of 16 network probes in soliciting host operating system information. The information received back from these probes then undergoes a series of tests, the results of which are used to form a signature or fingerprint. This fingerprint is then referenced against known signatures in a signature database in an attempt to make a classification [5].

Of the 16 network probes send by nmap, 13 are TCP packets. The first 6 are TCP SYN packets, the next 6, TCP packets with various flags set; 3 of which are sent to an open port; and the remaining 3, a closed port. The last TCP packet which is sent, is one in which an explicit congestion notification (ECN) flag has been set in the IP header. Of the remaining 3 packets, 2 are ICMP echo packets, and 1, a UDP packet sent to a closed port [8].

III. System Design and Implementation

A virtual host, in the context of this paper, is a host which appears to be real, but is in actuality not. As NKM purely routes packets, all packet generation occurs externally; this of course is with the exception of the control messages generated in the routing process. As a result, should a virtual node be reached, which is not bound to a physical host, the traffic will be dropped. This section investigates the requirements necessary in providing the ability to allow physical hosts to converse with contextually real hosts, referred to hereafter as virtual hosts.

[1]https://nmap.org/

A. ICMP

Perhaps the most rudimentary check in determining whether a host is alive, is through issuing the **ping** command. The **ping** tool, in its simplest form, makes use of the Internet Control Message Protocol (ICMP) through generating an ICMP *ECHO_REQUEST* datagram in an attempt to elicit an ICMP *ECHO_RESPONSE*. Should an ICMP *ECHO_RESPONSE be seen*, the host is taken to be alive.

The *code* and *type* fields present in the ICMP header are used in conjunction with one another is identifying the control message. Both ICMP *ECHO_REQUEST* and *ECHO_RESPONSE* are of the type 0; however the former has a code of 8, whereas the latter has a code of 0 [9].

B. TCP SYN Scan

A more advanced technique in ascertaining that a host is alive, is through trying to establish a TCP connection with the target. A TCP connection is established once the three-way handshake has occurred. The method referred to as a SYN scan, attempts to elicit a response from a target, through transmitting a TCP packet in which the *SYN* flag has been set [5]. There are various ways in which a host may, or may not, respond; these are outlined below.

1) Open Ports: Should the host respond with its TCP *SYN*, and *ACK*, fields set, the port on which the TCP *SYN* was received is open; this is due to the target host attempting to continue in establishing the three-way handshake. In this case, it can easily be inferred that the target is alive.

2) Closed Ports: Should no application be running atop the TCP port in question, the target host will respond with a TCP packet in which its *RST* field has been set; signalling that the connection be torn-down immediately [10]. This intent is enough to infer the target is alive, and that the port in question is indeed a closed port.

3) Filtered Ports: A result in which it is inferred the port is filtered, is one in which no response is received [5]. This information on its own results in no information-gain; the target may or may not be alive. Where the

target is in fact alive, this behaviour is usually indicative of the presence of an intermediary such as a firewall.

C. TCP FIN Scan

Like the SYN scan discussed in the previous section, the TCP FIN scan is another method which exploits the manner in which TCP handles connection state. However, where the SYN scan was concerned with connection establishment, the TCP FIN scan is concerned with the procedure for tearing down a connection. Once two parties have established a connection, either party may request the session be destroyed by sending the other a TCP segment in which the FIN bit has been set. At this point, the recipient would acknowledge this intention by responding with an acknowledgement, before themselves transmitting a segment in which the FIN bit has been set [10]. However, this behaviour is only the case where a session exists to be tore down. Therefore, as a consequence of a lack of state, the TCP FIN scan can make the following inferences upon transmitting a TCP FIN.

1) Open or Filtered Ports: In the case of an open port, the target would typically not respond to a FIN. This is because there exists no session between the scanner and target to be destroyed. However as a filtered port would behave in much the same fashion, one cannot differentiate between these two statuses. This lack of a response is information enough where trying infer the status of a port but says nothing where the status of the target host itself is concerned [5].

2) Closed Ports: As there does not exist the potential for a connection to be established, the target will immediately respond with a TCP reset. As previously stated, this information is enough to infer that the host is up.

D. UDP Port Unreachable Scanning

A similar method to that of the TCP SYN scan is UDP port scanning; which too tries to elicit a response from a target host. As a result of UDP being a far simpler protocol than TCP, response behaviour is less well defined. Consequently, open ports need not respond, and closed ports are not required to do so either; however, more often than not, hosts tend to respond with an *ICMP DESTINATION_PORT_UNREACHABLE* message if a closed UDP port is encountered [9].

As a result, UDP port scanning is not so reliable a method in determining whether a host is alive, and is found to be more applicable when trying to determine the status of various UDP ports [5].

E. Managing State

Whilst ICMP and UDP do not present much difficulty in their implementation due to their stateless nature, TCP requires a great deal of consideration where state is concerned. Figure 1 illustrates the many possible state transitions which can occur within, or during the course of, a TCP connection. Within the context of this implementation, this means that each virtual host needs to store this information, so as to act accordingly, for each and every other host (virtual or otherwise) with which it is communicating. Whilst not detailed in Figure 1, state information pertaining to TCPs reliable delivery, flow control, and error correction services, would also need to be maintained.

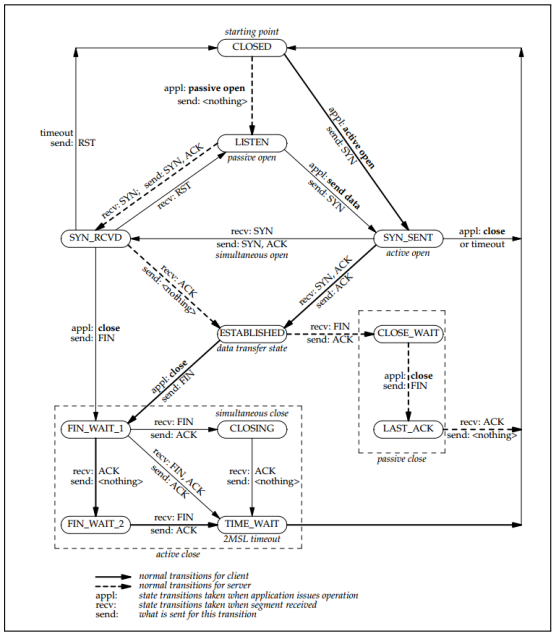

Fig. 1: TCP State Transition Diagram [11]

When considering the scale for which NKM was initially designed, incorporating an entire TCP stack on a per-virtual-host basis, introduces serious overhead. From

a processing perspective, this overhead may manifest in the form of delay or, at worst, a complete lack of responsivity. In terms of memory requirements, storing this information places huge restrictions on the number of virtual hosts which could potentially exist. In either case, the simulator becomes no longer suitable for the commodity hardware for which it was designed.

In an effort to implement a TCP stack which appears to be real, whilst at the same time minimise the overhead introduced as a result, a far simpler design to that illustrated in Figure 1 has been conceived. This approach aims to mitigate the identified overhead, as far as possible, by eliminating the possibility of state where a virtual host is concerned. This is made possible due to each virtual host only ever being directly involved in one side of any conversation that takes place over TCP. As a result of this, each virtual host can provide the illusion of maintaining state without actually doing so. This process is illustrated in Figure 2, and explained hereafter.

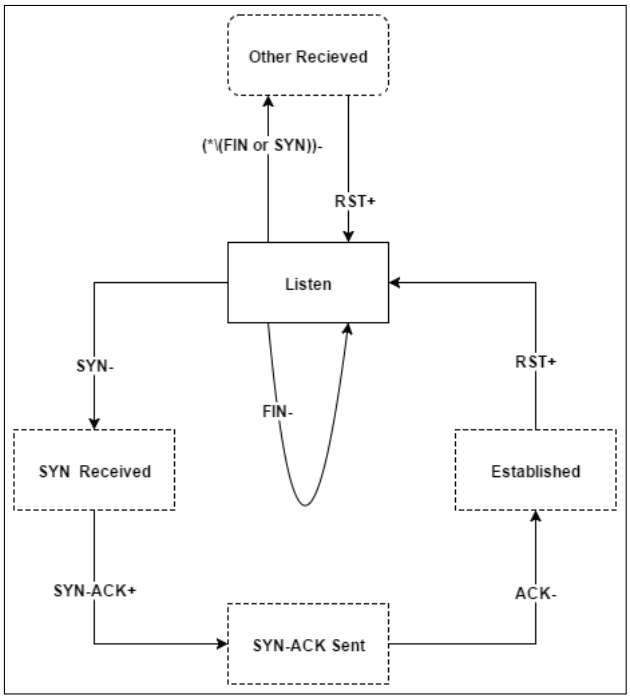

Fig. 2: TCP Pseudo-States for an Open Port

Figure 2 depicts a series of possible pseudo-states for a port which has been configured to be open. The signs depicted in Figure 2 which precede each transition are representative of direction; a minus sign is indicative of traffic having been received, and a plus sign, the associated response. Whilst the Figure depicts five states, only the *Listen* state is ever actually assumed; the other states exist only in context.

For clarification, consider again the process for establishing a TCP connection. With reference to Figure 2, this process is initiated when a virtual host receives a SYN. At this point the virtual host exists in the *SYN Received* pseudo-state. The virtual host will then respond with a SYN-ACK, thereby transitioning to the *SYN-ACK Sent* pseudo-state. Upon receiving a final ACK, the connection is *Established*.

This means the entirety of the three-way handshake can be completed, with only a virtual host responding to a SYN with a SYN-ACK. As the the last ACK that occurs is immediately followed by a TCP reset, the only state which is persistent is the *Listen* state. This behaviour, which is inherently reactionary, permits statefulness, without the implications associated with maintaining state. There are then only three cases which need to be handled for an open TCP port. These cases are listed below

1) If a SYN is received, the virtual host is to respond with a SYN-ACK (As was discussed in Section III-B1).
2) If a FIN is received, the virtual host is to not respond at all (as was discussed in Section III-C1).
3) If the conditions in Case 1 and 2 are not met, the virtual host is to respond with an RST.

Whilst Figure 2 differentiates between the pseudo-states labelled *Other Received* and *Established*, they are in actuality no different; the *Established* state exists only as a consequence of the secondary hosts intention to establish a connection. Furthermore, as a result of the connection immediately being reset, there does not exist a session to be torn down. As a result of this, a virtual host need not respond to any FIN requests, as detailed in Section III-C.

Whilst this section has up until now focused primarily on state where an open TCP port is concerned, it has alluded to address the response behaviour a virtual host is to exhibit in the case of closed, or filtered, TCP ports; ICMP; and UDP response behaviour. So as to be explicit, this information is provided below.

1) IF a TCP segment is received on a port which been

configured to be filtered, the virtual host will not respond at all (as was discussed in Section III-B3).
2) IF a TCP segment is received on a port which been configured to be closed, the virtual host will respond with a TCP RST (as was discussed in Section III-B2).
3) IF an ICMP ECHO_REQUEST is received, and the virtual host has been configured to respond, it will do so with an ICMP ECHO_RESPONSE (as was discussed in Section III-A).
4) IF a UDP datagram is received on any port, the virtual host will respond with an ICMP DESTINATION_PORT_UNREACHABLE message (as was discussed in Section III-D).

IV. Virtual Host Quality Testing

The degree to which a virtual host serves its purpose can be characterised by how closely it bares resemblance to that of a real host. Networking tools which leverage the behavioural characteristics exhibited by hosts, serve as an apt method for determining the extent to which this likeness exists; thereby providing the ability to test this feature qualitatively.

Three tools have been selected for this purpose, namely **ping**, **traceroute**, and **nmap**. Each of these tools serve to test the virtual host feature from a different perspective, and were chosen as a result of their widespread usage and success they hold in a variety of networking applications. Each of these tools will be explained where not done so before.

A. ICMP Testing with ping

The **ping** tool was first introduced in Section III-A. It is discussed further here with reference to the networking applications it facilitates. Within the field of Network Security, the tool can provide a simple test for determining the status of a target host; that being up or down. When applied to entire networks, it can serve as a method for host discovery, as discussed in Section II-A. In the fields of System, and Network, Administration the tool is predominantly purposed towards troubleshooting. This is due to its ease of use, as well as its ability to provide insight into the state of a network despite the simplicity of its application.

As an example, consider a scenario in which it is suspected that connectivity between two end-points is being affected adversely. There could be a number of possible reasons for this. As a starting point, one could **ping** the target. If a reply is seen, one is guaranteed there exists connectivity — though the extent to which this is true may be unknown. If not, one can begin an investigation into where in the route the problem is located. This can be achieved through *pinging* each node on the path, until such a time as a response is no longer seen. This information is illustrated graphically in Figure 3.

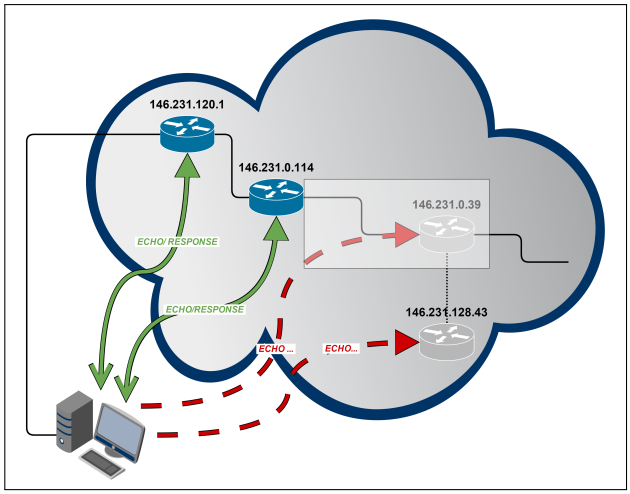

Fig. 3: Troubleshooting with ping

If the problem is not as well defined as a total lack of connectivity between end-points, the tool may still prove useful. Though **ping** comes in a variety of flavours, it, in most cases, provides information relating to *packet loss* and *round trip time* (RTT) measurements; this information is typically provided for individual probes, as well as on average. Should a deviation in the expected RTT, or a failure to acquire a response, occur for one or more probes, it may indicate that the network is congested, or that the target is under load; at which point further investigation is required.

With the addition of virtual hosts, and their ability to respond, or not respond, to ICMP ECHO_REQUESTs, the simulator is positioned to accommodate the recreation of these scenarios; be it for training, or otherwise. Increasing the amount of jitter, chance to drop a packet, and the chance a packet has to be mangled, would facilitate the simulation of the scenario in which packet loss and congestion were discussed. As both tests and examples

have been provided for these features in [2], none of these will be replicated here. Instead, information relating to the recreation of the scenario depicted in Figure 3 will follow.

As can be seen, there exist a total of 4 virtual nodes. Details relating to the configuration thereof have been omitted from this paper. The interested reader is referred to [12], in which this information in presented in detail. Each virtual host which to respond to **ping**s, must be configured to respond to ICMP ECHO_REQUESTs. For the purpose of a concrete example, the scenario can be formalised into a set of steps that might be taken based on the topology presented in Figure 3. These steps are provided in the use case below:

1) The user pings 146.231.128.43, for which a there is no response.
2) The user pings 146.231.120.1 in an attempt to determine where in the path the problem resides. The tool reports on having received a response. 146.231.120.1 is reachable
3) The user pings 146.231.0.114, again a receiving a response. 146.231.0.114 too is reachable.
4) The user pings 146.231.0.39, for which there is no response. At this point the user can infer the problem is localised to reaching 146.231.0.39.

In accordance with the use case above, Listings 1 and 2 provide the results having issued the **ping** command in steps 3 and 4 respectively. As can be seen in Listing 1, **ping** reports that all 4 transmitted packets returned a response. Whereas in Listing 2, the tool reports there was 100% packet loss. The *'-c '* option is used to specify the number of packets the tool is to transmit before stopping.

Listing 1: Successful ping Example:

```
1  root@demo6:~# ping -c 4 146.231.0.114
2  PING 146.231.0.114 (146.231.0.114) 56(84) bytes of data.
3  64 bytes from 146.231.0.114: icmp_req=1 ttl=62 time=0.248 ms
4  64 bytes from 146.231.0.114: icmp_req=2 ttl=62 time=0.200 ms
5  64 bytes from 146.231.0.114: icmp_req=3 ttl=62 time=0.199 ms
6  64 bytes from 146.231.0.114: icmp_req=4 ttl=62 time=0.204 ms
7
8  --- 146.231.0.114 ping statistics ---
9  4 packets transmitted, 4 received, 0% packet loss, time 2997ms
10 rtt min/avg/max/mdev = 0.199/0.212/0.248/0.027 ms
```

B. UDP Testing with traceroute

Yet another tool which proves useful within networking applications is **traceroute**. Simply put, the tool prints the route its packets take in reaching a specified destination. This it achieves, through successively transmitting a UDP packet, in which a destination port likely to be closed has been specified. The first of these packets has its TTL set to 1, each subsequent packet will have this value incremented. Each gateway these packets traverse will decrement the TTL and, should it reach 0, notify the sender of the TTL having expired. Each ICMP TTL_EXCEEDED message received by the tool in turn, is then used to identify each gateway on the route. The tool will persist in its behaviour until it encounters an ICMP DESTINATION_PORT_UNREACHABLE message; this message is triggered as a result of the selected destination port being closed.

Listing 2: Unsuccessful ping Example:

```
1  root@demo6:~# ping -c 1 146.231.0.39
2  PING 146.231.0.39 (146.231.0.39) 56(84) bytes of data.
3
4  --- 146.231.0.39 ping statistics ---
5  1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

Though this explanation overlooks some of the intricacies present in the process, it is sufficient enough to demonstrate the merits of its application. For one, consider again the example presented in Section IV-A. As an alternative to *pinging* each node in the routes path, one could instead have simply performed a traceroute. This would then indicate at which point in the route the problem in connectivity in located. As another example, consider a scenario in which it is suspected there is a routing misconfiguration; i.e. the wrong default gateway has been specified. If this is the case, communication outside of ones own subnet becomes impossible. In performing a **traceroute**, one is able to ascertain that traffic is not being routed as it should, thereby permitting its user to attribute the problem to a misconfiguration in the default gateway.

Whilst the original work of [2] catered to the tool, its nodes possessed only the ability to act in the capacity of the routers they might represent. As the tool, by default, terminates upon soliciting an ICMP DESTINATION_PORT_UNREACHABLE message, it requires the target possess a UDP stack; a feature which was absent in the simulators nodes, but present in the real hosts to which they might have been bound. With the addition of the virtual hosts feature, each virtual

host needs to possess this capability in order for the tool to be supported. This is achieved through all UDP ports behaving as though they are closed.

The basis of the topology represented in Figure 3 was obtained as a result of having performed a traceroute on real website www.ru.ac.za — which has the real address 146.231.128.43. It was then reproduced using the simulator. Verification that this information was not misrepresented was performed by again using traceroute. Listing 3 reveals the results having performed traceroute within the simulation. The option *'- n'* omits the resolution of IP addresses, and *'- q 1'* specifies that only one probe is to be transmitted per query — by default traceroute will transmit three probes in order to provide statistical information as well as alternative routes, should they be taken.

Listing 3: Traceroute Example

```
1 [root@demo6]# traceroute -n -q 1 146.231.128.43
2 traceroute to 146.231.128.43, 30 hops max, 60 byte packets
3  1  146.231.120.1      0.210 ms
4  2  146.231.0.114      0.169 ms
5  3  146.231.0.39       0.151 ms
6  4  146.231.128.43     0.119 ms
```

C. TCP Testing with Nmap

Nmap was first introduced in Section II-A2 and some of the methods employed by the tool discussed thereafter in Section III-B. The tool is discussed here for the purpose of determining the extent to which a virtual host is found to appear real. These tests are carried out through using a variety of different options. Provided in Table I is a description of the virtual host configuration file used for the tests which follow hereafter.

Virtual IP	Open Ports	Filtered Ports	ECHO_RESPONSE
146.231.120.1	22/tcp	-	Yes
146.231.0.114	22/tcp	-	Yes
146.231.0.39	22/tcp	-	Yes
146.231.128.43	80/tcp, 443/tcp	22/tcp	Yes

TABLE I: Virtual Host Test Configuration

The virtual host 146.231.128.43, might be representative of a web server. Port 22/tcp is typically associated with SSH, whilst 80/tcp and 443/tcp typically HTTP and HTTPS respectively [13]. In an effort to create a scenario that is somewhat realistic, port 22/tcp is filtered; this might, in reality, be the case were access control mechanisms enforced. Listing 4 reveals the results of having had nmap scan on the target with no additional arguments. *Note, that by default nmap performs a TCP SYN scan, as detailed in Section III-B.*

Listing 4: Nmap SYN Scan Example

```
1  root@demo6:~# nmap 146.231.128.43
2
3  Starting Nmap 6.00 ( http://nmap.org ) at 2015-10-20 02:06 PDT
4  Nmap scan report for 146.231.128.43
5  Host is up (0.00014s latency).
6  Not shown: 997 closed ports
7  PORT    STATE    SERVICE
8  22/tcp  filtered ssh
9  80/tcp  open     http
10 443/tcp open     https
11
12 Nmap done: 1 IP address (1 host up) scanned in 14.63 seconds
```

As can seen in Listing 4, the tool correctly reports on the on the non-closed ports which correspond to the description of the virtual host configuration presented in Table I. Additionally the tool reports on it having found 997 closed ports. This is due to the tool only scanning the first 1000 common ports by default. Whilst supporting information is provided for the service affiliated with each non-closed port, it is at this point pure guesswork; nmap has not yet performed any tests which will corroborate this information. In order to acquire this information, one must make use of one of its more overt methods. One method for doing this is through specifying the tool make use of its service versioning features which can be invoked through supplying the *'- sV'* option. However, as a result of it being a far more vigorous test, the *'-A'* option will be used. This option, in addition to performing service versioning, incorporates OS detection, script scanning, and traceroute into the scan [5]. Listing 5 details the results of having performed nmap on 146.231.128.43 with all of these features enabled.

The result of having incorporated version detection into the scan is that nmap now reports the services other than SSH to be *tcpwrapped*. This is due to each of its TCP probes being permitted only to complete the three-way handshake and having 146.231.128.43 reset the connection thereafter. As a result of this behaviour, no further communication can take place after the probe has traversed the transport layer of the destinations network stack. A situation which would typically arise as result of the source address not being to permitted to continue in its activities; E.g. whitelisting. As a consequence of port

Listing 5: Nmap OS Fingerprinting and Service Versioning Example

```
1  root@demo6:~# nmap -A 146.231.128.43
2
3  Starting Nmap 6.00 ( http://nmap.org ) at 2015-10-20 02:32 PDT
4  Nmap scan report for 146.231.128.43
5  Host is up (0.00018s latency).
6  Not shown: 997 closed ports
7  PORT    STATE    SERVICE    VERSION
8  22/tcp  filtered ssh
9  80/tcp  open     tcpwrapped
10 443/tcp open     tcpwrapped
11 No OS matches for host (If you know what OS is running on it, see
12 http://nmap.org/submit/ ).
13 TCP/IP fingerprint:
14 OS:SCAN(V=6.00%E=4%D=10/19%OT=80%CT=1%CU=37750%PV=N%DS=4%DC=T%G=Y%TM=562504
15 OS:6C%P=x86_64-unknown-linux-gnu)SEQ(SP=0%GCD=0%ISR=0%TI=RD%CI=RD)OPS(O1=WA
16 OS:NM5B4T10S%O2=M578W0ST10L%O3=T10NNW5NM280%O4=ST10WAL%O5=M218ST10WAL%O6=M1
17 OS:09ST10)WIN(W1=1%W2=3F%W3=4%W4=4%W5=10%W6=200)ECN(R=Y%DF=N%T=3D%W=3%O=WAN
18 OS:M5B4SNN%CC=S%Q=RU)T1(R=Y%DF=N%T=3D%S=A%A=S+%F=AS%RD=0%Q=)T2(R=Y%DF=Y%T=3
19 OS:D%W=80%S=A%A=Z%F=R%O=WANM109T10S%RD=0%Q=)T3(R=Y%DF=N%T=3D%W=100%S=A%A=S+
20 OS:%F=UAPSF%O=WANM109T10S%RD=0%Q=)T4(R=Y%DF=Y%T=3D%W=400%S=A%A=Z%F=R%O=WANM
21 OS:109T10S%RD=0%Q=)T5(R=Y%DF=N%T=3D%W=7A69%S=A%A=Z%F=R%O=WANM109T10S%RD=0%Q
22 OS:=)T6(R=Y%DF=Y%T=3D%W=8000%S=A%A=Z%F=R%O=WANM109T10S%RD=0%Q=)T7(R=Y%DF=N%
23 OS:T=3D%W=FFFF%S=A%A=Z%F=RF%O=WFNM109T10S%RD=0%Q=)U1(R=Y%DF=N%T=FC%IPL=38%U
24 OS:N=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=N)
25
26 Network Distance: 4 hops
27
28 TRACEROUTE (using port 554/tcp)
29 HOP RTT      ADDRESS
30 1   0.23 ms 146.231.120.1
31 2   0.23 ms 146.231.0.114
32 3   0.22 ms 146.231.0.39
33 4   0.22 ms 146.231.128.43
34
35 OS and Service detection performed. Please report any incorrect results at
36 http://nmap.org/submit/ .
37
38 Nmap done: 1 IP address (1 host up) scanned in 92.89 seconds
```

22/tcp being filtered, each probe is simply discarded, and a TCP reset sent back; at which point no other inferences can be made.

The TCP/IP fingerprint provided in Listing 5 is a result of nmaps inability to adequately make a classification. Were the target identified, one would typically be presented with a textual description of the OS, vendor name, underlying OS and its generation, and in some cases a Common Platform Enumeration[2] (CPE). Instead the tool has produced what is referred to as a subject fingerprint — a representation of the information obtained from having probed the target. Each entry of the subject fingerprint refers to the outcome of a specific set of tests [3] or contains information specific to the conditions surrounding the tests themselves [5].

This information is what would typically be used to form a reference fingerprint; this however depends on favourable test conditions, anything less and little can be said about the target. Usually each reference fingerprint is the result of having processed several subject fingerprints. Results owing to tests which are known to be less reliable, as well as conflicting subject fingerprint information, may be modified or discarded. Each reference fingerprint is then stored in a database for later use where it can be used in making a classification. Whilst in this instance, nmap has failed to make a classification, a result which is not all that uncommon, it has reported the test conditions to be good; this is indicated by the presence of *'G=Y%'* present in line 14 of Listing 5.

Whilst traceroute was discussed in the Section IV-B it was predominantly concerned with its application where UDP was concerned. In truth, traceroute needn't make use of this protocol; however it does do so by default. Arguments such as *'-I'* or *'-T'* specify that the tool instead utilises ICMP or TCP. In the case of Listing 5, nmap makes use of 554/tcp, a port which is known to be closed as a result of the virtual hosts configuration. In reaching this port, a TCP reset is triggered, signalling the destination has been reached.

The TCP FIN scan was discussed in Section III-C. In an effort to target the virtual hosts ability to accommodate this type of scan, whilst at the same time demonstrate how nmap can be purposed towards host discovery, a FIN scan of the 146.231.0.0/24 subnet is performed. This scan first pings each and every address on the subnet, and then performs a FIN scan on responsive hosts thereafter. The *'-n'* option skips the DNS-bases address resolution process which is enabled by default. As can be seen in Figure 6, both hosts have successfully been identified, as has the status of their ports; the ambiguity that exists between a port being open or filtered is a limitation of the scan itself, as was discussed in Section III-C.

Listing 6: Nmap Host Discovery using a FIN Scan Example

```
1  root@demo6:~# nmap -sF -n 146.231.0.0/24
2
3  Starting Nmap 6.00 ( http://nmap.org ) at 2015-10-20 02:54 PDT
4  Nmap scan report for 146.231.0.39
5  Host is up (0.00012s latency).
6  Not shown: 999 closed ports
7  PORT    STATE         SERVICE
8  22/tcp open|filtered ssh
9
10 Nmap scan report for 146.231.0.114
11 Host is up (0.00012s latency).
12 Not shown: 998 closed ports
13 PORT    STATE         SERVICE
14 22/tcp  open|filtered ssh
15 179/tcp open|filtered bgp
16
17 Nmap done: 256 IP addresses (2 hosts up) scanned in 1.87 seconds
```

[2]CPE is a structured naming scheme for information technology systems, software, and packages.

[3]The tests nmap performs in creating a subject fingerprint was discussed in Section II-A2.

V. Conclusion

This research sought out to investigate the viability of extending NKM so as to provide a level interaction it, on its own, previously did not possess. The extent to which the virtual hosts feature was found to appear realistic was tested in IV. Sections IV-A and IV-B revealed the features ability to correctly handle, and respond to, ICMP and UDP payloads. The tests performed in obtaining this information served to demonstrate how **ping** and **traceroute** might be used in facilitating future scenarios in light of their support.

In Section IV-C, the virtual hosts ability to correctly handle various TCP payloads was assessed. With reference to Section III-E, its capacity to provide the illusion of state was tested through using several different options provided by **nmap**. The first of these tests was provided in the form of a TCP SYN Scan, to which the virtual host demonstrated the appropriate behaviour for open, closed, and filtered ports. In performing a test more substantial, the virtual hosts ability to conceal the absence of any underlying services was tested, to which it exhibited behaviour not unlike a TCP wrapper in the case of ports which were open. Furthermore, though the host itself could not be classified, it did not reveal itself to be unreal. The last test that was performed targeted the virtual hosts ability to handle teardown requests. This test was carried out with a TCP FIN Scan, to which the virtual host responded appropriately.

A. Future Work

The below lists several ideas for possible extensions to the system:

1) The virtual hosts feature could be extended to include varying levels of fidelity; thus permitting a higher degree of realism for applications less demanding of scale. As an alternative to the existing feature, a virtual host could exhibit the behavioural characteristics of specific network hosts. This could be achieved through reverse engineering nmaps fingerprint database. Each reference fingerprint contained therein, could then be used to determine the response behaviour for a selected profile. This would allow for allocating operating system personalities to virtual hosts in a manner that is dynamic.
2) The virtual hosts feature could yet further be enhanced through providing the option to disclose service information, as was discussed in Section II-A1. This however will require state information to be maintained for services that make use of TCP. This feature could be offered as another level of fidelity where scale is a factor.

References

[1] Internet World Stats. (2014) Usage and Population Statistics. Miniwatts Marketing Group. http://www.internetworldstats.com/stats.htm. Accessed 11 May 2015. [Online]. Available: http://www.internetworldstats.com/stats.htm

[2] A. Herbert, "Towards large scale software based network routing simulation," Rhodes University, South Africa, pp. 16–144, 2014.

[3] C. T. Wai, "Conducting a penetration test on an organization," Tech. Rep., 2002.

[4] A. Stoica, "A study on the information gathering method for penetration testing," *Journal of Security Engineering*, vol. 5, no. 5, p. 10, 2008.

[5] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009, ISBN: 0979958717.

[6] R. Spangler, "Analysis of remote active operating system fingerprinting tools," *University of Wisconsin*, 2003.

[7] F. Veysset, O. Courtay, and O. Heen, "New tool and technique for remote operating system fingerprinting," *Intranode Software Technologies*, 2002.

[8] L. G. Greenwald and T. J. Thomas, "Toward undetected operating system fingerprinting." *WOOT*, vol. 7, pp. 1–10, 2007.

[9] J. Postel, "Internet Control Message Protocol," RFC 792 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 950, 4884, 6633, 6918. [Online]. Available: http://www.ietf.org/rfc/rfc792.txt

[10] ——, "Transmission Control Protocol," RFC 793 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168, 6093, 6528. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[11] I. S. Yoo, "Method for preventing denial of service at-

tacks using transmission control protocol state transition," Dec. 2014, uS Patent 8,925,068.

[12] C. Koorn, "Network simulation," Rhodes University, South Africa, pp. 1–71, 2015.

[13] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry," RFC 6335 (Best Current Practice), Internet Engineering Task Force, Aug. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6335.txt