CS 638 Lab 2: End Host Tools and DNS

Joe Chabarek, Kevin Springborn, Mike Blodgett and Paul Barford
University of Wisconsin –Madison
jpchaba,springbo,mblodget,pb@cs.wisc.edu

Managing and troubleshooting end hosts - especially in large enterprise environments - is essential for efficient operations in most businesses today. However, end host management is extremely complicated and time consuming due to the scale and diversity of systems, applications and users in most environments. Furthermore, services that are routinely used by end hosts such as email servers, web servers, file systems/backups and the domain name system must be as close to 100% available as possible. This second lab is intended to give you experience with common tools that are used to evaluate and assess end host connectivity, and to give you experience with the domain name system.

1 Overview and Objectives

Lab 2 is divided into two parts. The first part will introduce you to tools that provide important insights into end host network configuration and on their network behavior. These tools are commonly used for debugging and troubleshooting. The second part of the lab will focus on the domain name system. You will setup a simple DNS hierarchy and execute requests that demonstrate how DNS works in the wide area.

Upon completion of this lab exercises below, students will be able to:

1. *Use standard tools to specify and inspect end host network configuration*
2. *Passively monitor packet streams to/from an end host*
3. *Use Simple Network Management Protocol (SNMP) to gather data from end hosts*
4. *Inspect and trouble shoot DNS in a local area environment*

2 Part 1: Standard Networking Utilities

Several networking utilities/tools are frequently used by network administrators and researchers to verify connectivity, observe packets, map physical addresses

to IP addresses and manage network devices. Gaining experience with these tools is the focus of part one of this lab. The tools are listed below along with a description of what they do. Students should use `man` pages along with web resources for further information on the tools, how they work and how to use them for specific tasks. Following the tool description is a set of tasks that are the specific focus of part one and should be commented on in your notebooks.

2.1 Ping

Ping is one of the most commonly used tools for debugging problems an IP network. The purpose of ping is to test whether a particular host is *accessible*. To do this the utility generates an ICMP (Internet Control Message Protocol http://www.faqs.org/rfcs/rfc792.html) Echo Request datagram sent to a specified destination and waits for a ICMP Echo Reply. Unless otherwise specified, the ping command will continue to send packets until a command line interrupt is given (Control-C). An example of how to specify a ping command is as follows: **ping -c 5 192.13.4.2** where -c 5 indicates that only five ICMP Echo Request datagrams are to be sent and 192.13.4.2 is the IP address of the computer that is being queried. It is good practice to ensure connectivity of the PC's and routers in your experimental networks before you alter the configurations. Ping is typically available on systems without requiring superuser privileges, though certain options to the ping command may change that fact.

2.2 Tcpdump

Passive network monitoring typically refers to capturing packet traffic that is transmitted on a link. It is typically done by the host that is itself sending or receiving packets, and is important for understanding the details of the traffic - especially at the application, transport and network layers. Tcpdump is the most commonly used utility for passive network monitoring. A risk in passive monitoring is that there can be a great deal of data passed on a network that the listener is not interested in observing. To address this risk, filtering is commonly used to avoid capturing unwanted packets. The filters can be based on packet or payload information. An example of how to invoke the tcpdump utility is as follows: **tcpdump -n -l > filename & tail -f filename**. The tail command views the end of the file as it is being written. Alternatively, the user can omit the tail command and view the file after capturing has been completed (hit Control-c) to end packet capture. Tcpdump requires superuser privileges.

2.3 Wireshark, formerly known as Ethereal

Wireshark is a packet monitor that is similar to tcpdump, but includes a sophisticated user interface. The interface can be very helpful while interactively examining network traffic. In addition, Wireshark provides detailed information for each packet via drop down menus. To begin type 'wireshark' at the command prompt. The main window will be displayed if you have X Widows

functionality. If you are running Wireshark from a remote system via ssh make sure to use the -Y flag to forward X11 packets. Next, select and set the capture options from the Capture:Start box in the main window. Some helpful options include automatic scrolling in live capture, update list of packets in real time, enable MAC name resolution, enable network name resolution, and enable transport name resolution. Be sure to select the correct interface. Hit **OK** to begin passive monitoring. To save captured traffic print via the File tab to a file.

2.4 Ifconfig

Ifconfig is a utility used to manipulate network interfaces on an end host. Using ifconifig, an administrator can activate/deactivate a network interface card and can be used while the system is running to dynamically modify the network configuration parameters, or during initialization of the operating system via a configuration file. To view the configuration parameters of the interfaces on a host, type **ifconfig** with the -a flag, which will display information for all devices (those active and inactive). You can also display the configuration for a specific interface via **ifconfig [device]**. To disable a device issue **ifconfig [interface] down**. Be sure NOT to disable or corrupt the control interface (eth0) on any of the workstations in your experiment. To enable an interface issue the **ifconfig [interface] up** command. To set the address of an interface issue the following command: **ifconfig [interface] [IPAddress] netmask [Netmask] broadcast [Broadcast]** (*e.g.,* ifconfig eth1 192.168.15.2 netmask 255.255.255.0 broadcast 192.168.15.255). Typically, to modify an address the interface is taken down, the address change is issued, and then the interface is brought back up. Ifconfig requires superuser privileges.

2.5 Netstat

Netstat is a utility that reports end host network statistics including the kernel routing table, network interface information, and other useful information on network behavior. Commonly used instances of the netstat utility include:

1. **netstat -i** Displays network interface information.
2. **netstat -r** Displays routing table information.
3. **netstat -rn** -n Displays IP Addresses instead of domain names.
4. **netstat -an** Displays information on TCP and UDP ports that are in use.
5. **netstat -tan** Displays only the TCP ports that are in use.
6. **netstat -uan** Displays only the UDP ports that are in use.
7. **netstat -s** Displays usage statistics for various networking protocols.

2.6 Address Resolution Protocol

During a filter free packet monitoring session an observer will most likely see a series of ARP packets being exchanged from time to time. ARP (Address Resolution Protocol) is used to establish a translation or mapping between MAC (physical) addresses and IP addresses. End hosts typically maintain an ARP cache in order to speed up the process of resolving and IP to MAC translation. The ARP cache can be viewed with the **arp -a** command. Each entry in the ARP cache will timeout periodically triggering a lookup on the network. It is possible to prematurely delete an ARP cache entry with the **arp -d Address** command. Additionally, a user can statically add an ARP entry into the cache with the **arp -s IPaddress MACaddress** command. This entry does NOT have a timeout, and the MAC address in entered as 6 hexadecimal bytes separated by colons (*i.e.,* 00:00:00:00:00:00).

2.7 SNMP

The Simple Network Management Protocol (SNMP) provides a broad framework for managing devices in a network. There are a number of different components in SNMP: SNMP agents, SNMP managers, Management Information Bases (MIBs), and the SNMP protocol itself. The Management Information Bases (MIBs - see below) load and organize records of interest (stored in /usr/share/snmp/mibs) which are distributed via the agents, and managers using the SNMP protocol. The SNMP agent runs on a particular network device and updates the MIB records so that they represent the near real-time state of the host system. A SNMP manager queries network devices with SNMP agents for access to particular elements of the MIB database.

2.7.1 Management Information Base

A Management Information Base (MIB) consists of text files that form a database of information about a particular network system. This database uses a hierarchical tree to define a name space of object identifiers. These identifiers are implement via Abstract Syntax One, a standard notation to describe data structures. Each node in the tree is identified by an integer or a string, for example. the string CISCO-CONFIG-MAN-MIB or the sequence of integers 1.3.6.1.4.1.9.9.43.2

2.7.2 SNMP Agents and Managers

For this lab we will demonstrate how the different components of the SNMP protocol interact and what types of information are found in the MIBs. There is a freely available software package that implements the SNMP components and protocol called Net-SNMP. It will be necessary to begin the SNMP agent daemon on some of the workstations in your experiment. Then you will use the snmpget,snmpgetnext, or snmpwalk tools from the Net-SNMP package to request information from the agent. You can consider these tools the SNMP

manager components. There are five different messages defined in the SNMP protocol which are used by the commands in the Net-SNMP package:

1. **GET REQUEST** is used to retrieve a piece of management information.
2. **GET NEXT REQUEST** is used iteratively to retrieve sequences of management information.
3. **GET RESPONSE** is issued from an SNMP agent to a request message.
4. **SET** is used to make a change to a managed subsystem.
5. **TRAP** is used to report an alert or other asynchronous event about a managed subsystem.

We are interested primarily in the Get-request and Get-next-request. These are implemented in Net-SNMP as *snmpget* and *snmpgetnext* respectively.

3 Part 1 Tasks and Questions

Use the Schooner "GUI-Editor" you learned in Lab #1 to construct the topology shown on the pre-lab page. Use the FC6-STD OSID for each node.

3.0.3 Adding RPMs to Nodes in Schooner

The testbed images are smaller installations of an OS and don't contain a number of software packages that you might want to use in the lab by default. In Schooner, you can load an image on a node, install a piece of software manually, and then save the new image of your node with the software installed. You can then use this image later or on a large number of nodes. But making new images can be a slow process. Another way you can have your custom software installed is to have the testbed do it for you.

RPM's (RPM Package Manager) are used across many Linux distributions for installing software. By giving directives to the testbed, it will install RPM's for you. For this experiment the correct RPM's have already been downloaded, you just need to add them to your experiment. If you select node0 in the GUI and then select the Edit button for 'RPM Files' in the properties window, a box will appear. In the box, click the + sign which will add an entry. You can write out the paths manually, but it's easier to just pull down the path entry field. It will show the rpms we have downloaded into /proj/cs638/rpms. Make an entry for each of the RPMs in the below list, and then repeat for node1 and node2.

1. desktop-file-utils-0.10-7.i386.rpm
2. htmlview-4.0.0-3.fc6.noarch.rpm
3. redhat-menus-6.7.8-2.fc6.noarch.rpm
4. wireshark-0.99.6-1.fc6.i386.rpm

5. wireshark-gnome-0.99.6-1.fc6.i386.rpm
6. net-snmp-5.3.1-15.fc6.i386.rpm
7. net-snmp-libs-5.3.1-15.fc6.i386.rpm
8. net-snmp-utils-5.3.1-15.fc6.i386.rpm
9. xorg-x11-fonts-Type1-7.1-2.noarch.rpm

This method can be used to install many different software packages. HTTP, NTP, and LDAP are a few common network services which you can easily install with RPM's. Now that your configuration is complete create and swapin your experiment.

3.0.4 Configuring and starting the SNMP daemon

Once your experiment has been swapped in, you will have to configure and manually start the daemon or SNMP agent that will run on the host. After logging into one of your nodes run the command 'snmpconf'. This program will allow you to read in the current default SNMPD configuration file in /etc/snmpd/snmpd.conf, and then merge in changes. See if you can get through the configuration. You need to make changes to the snmpd.conf file, Access Control, SNMPv1/SNMPv2c read-only, add a community 'public' accessable from localhost, with no restrictions on the OIDs it can read. Then you type, finished, finised,quit, and save the file off. Take a look at the bottom of the file it should have something like

```
com2sec notConfigUser  default       public
group   notConfigGroup v1           notConfigUser
group   notConfigGroup v2c           notConfigUser
view    systemview    included   .1.3.6.1.2.1.1
view    systemview    included   .1.3.6.1.2.1.25.1.1
access  notConfigGroup ""      any       noauth    exact  systemview none none
```

If you can't get the config to work, there is a good config file at /proj/cs638/snmpd.wail. Copy the config file to /etc/snmpd/snmpd.conf overwriting the system default configuration, Then start the daemon with '/etc/rc.d/init.d/snmpd start'.

3.1 Standard Networking Utilities

1. ping: check to see if connectivity between all of the workstations. Make sure to use the correct addresses 198.133.225.0/24 are the control interfaces and are not what you want to be using.
2. tcpdump: set up tcpdump on PC1 and run ping between PC1 and PC3, describe the traffic generated by the ping. Specify a filter to view only ICMP packets, record this filter in your lab notebook. Make sure that you understand what tcpdump is reporting.

3. wireshark: run wireshark on PC1. Establish an SSH session between PC1 and PC3. Observe and record the traffic generated by SSH. Try to describe what triggers packets that are transmitted in SSH.

4. arp: keep wireshark running and clear the arp table on PC1 and PC3. Issue a ping from PC1 to PC3 and describe what you see in terms of traffic.

5. ifconfig: change the last 8 bits of the IP addresses for eth1 on PC1,PC2,PC3. When you have finished, use ping to ensure that all computers can still communicate.

6. netstat: run some of the netstat commands on any one of the PCs to familiarize yourself with the utility.

3.2 SNMP

Start the snmpd daemon on some of your PCs. Below are a few values available via SNMP.

1. *ipForwarding*
2. *sysName*
3. *sysLocation*
4. *ipRouteTable*
5. *tcpRtoAlgorithm*
6. *tcpConnState*

Use the snmpget,snmpgetnext, and or the snmpwalk command to display the values on one of the PCs (use localhost for PCs address). Use the man pages for each of the commands to find more information about usage. Also, report the numerical OID and full textual descriptor for the fields that are listed. Try finding the same values on a different host, do they match?

4 Part 2: The Domain Name System

4.1 Overview

A DNS server is simply a machine that handles key-value pairs. The pairs are a many to many relation. A user can query the server to find values, or users can insert new key-value pairs. DNS servers typically are used to map the human readable Internet addresses to corresponding IP addresses. For instance I can issues a query to the DNS sever to lookup "www.yahoo.com" and it will return the IP address associated with that site.

By using a hierarchical structure and efficient caching, the number of DNS servers needed in the Internet is kept reasonably low. The DNS specification requires *zones* to be served by a minimum of two DNS servers for redundancy, but this is by no means a maximum. The Computer Science Department at UW-Madison, for example, uses five servers, load balancing routers, and off site services rendered by another University. Large ISP's and content providers maintain large numbers of servers that are geographically distributed. The DNS servers for a zone are responsible for keeping key-value pairs for all computers inside of the zone.

A DNS server is also needed to handle address lookups for other computers on a network. The operating systems on most computers do not know about the global structure of DNS, these machines implement what as known as a *stub resolver*. Applications that need the network ask the operating system for the IP address of *e.g.,* www.google.com. The local machine only knows how to pass this request on to a small list of DNS servers it has configured by a system administrator. These servers provide 'recursive' DNS services to the hosts. For example, your request for www.google.com would generate a request from the local machine to the local DNS server, the server would then query one of many *root servers*. The root server will respond by informing the local DNS server who it should ask next. This process repeats until the local DNS server contacts a DNS server that contains the record for the address in question. The local DNS server would then respond to your machine with the IP address for www.google.com. Servers that offer recursive services are frequently separated from servers that contain real zone data. Servers that contain authoritative information about a zone, are aptly called *authoritative DNS servers*, and while you would think recursive DNS servers would be referred to as recursive servers, which they sometimes are, because of their caching abilities they are most commonly referred to as *caching nameservers*.

The domain name itself specifies the path of DNS servers to be queried, and it is always read from the right to the left. A trailing '.' is added to the address, which corresponds to the root servers. Looking up www.cs.wisc.edu would begin with a lookup to the root server to find the DNS server for the 'edu.' zone. The root server would respond with the address of a DNS server for the edu. zone. The local DNS server would then ask the edu. DNS server for the IP address of the wisc.edu. DNS server. The local DNS server would then ask the wisc.edu. DNS server for the IP address of the cs.wisc.edu. DNS server, and so on.

In reality things are a bit more complicated, but this general picture should help you get started.

4.2 Setup

The experimental setup for this part of the lab requires several machines. You can easily construct a larger network topology, five nodes on a simple LAN are recommended. Using the Schooner "GUI Editor" from Lab #1, create your five node topology shown in Figure 1. You'll also need to do some configuration of the nodes in the GUI. For each node, select FC6-STD as your OSID as shown

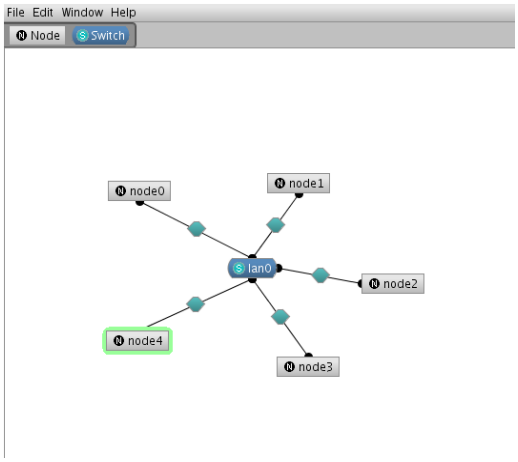

Figure 1: Network configuration for the DNS experiment.

in Figure 2. If you want some experience with another OS, feel free to use FBSD62-STD for node3 and/or node4, the others have to be FC6-STD.

4.2.1 Setting IP Addresses Manually

The testbed software will automatically figure out which IP addresses to use in your experiment, but their are times where you need to set an IP address to a specific value. Later on we have some configuration files that require IP addresses, and while dynamically generating these configurations from certain testbed data files is possible, we're not doing that here. In the GUI, click the hexagon shape in the middle of the network link between node0 and the lan0 icons. In the properties window, expand the 'IP Addresses' menu, fill in the appropriate IP for node0, repeat this for each of the nodes.

```
node0 192.168.1.10
node1 192.168.1.11
node2 192.168.1.12
node3 192.168.1.13
node4 192.168.1.14
```

4.2.2 Adding RPMs

Use the same procedure as part one, but this time use the bind, bind-libs, and bind-util rpms. Make an entry for each of these three RPMs, and then repeat for node2 and node3.

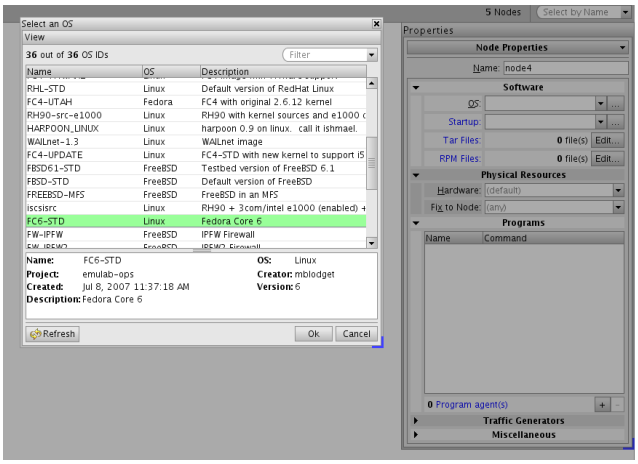

Figure 2: Snapshot of OSID configuration window.

4.2.3 Adding TAR files to Nodes in Schooner

In addition to the RPM files, you can also direct the testbed to add a TAR file to a node. A TAR archive is commonly used to store software, data, or configuration files. For this part of the lab, we have already created default configurations TAR files, you just need to tell the testbed which archive to use. For each node, in the GUI select the node, and the edit button for 'TAR Files', add the appropriate file from /proj/cs638/DNS. When the experiment is swapped in, it will untar the archive into the root directory, placing all our default configuration in place.

4.2.4 Auto Start Scripts

After your experiment swaps in, you could log into each node and start the name service, but the testbed software also provides a method for you to have a script run at boot time. In the GUI, for node0, node1, node2 as the 'Startup" property, type in "sudo /etc/rc.d/init.d/named start". This will start the name service on boot. Think about the possibilities with this. You could script a whole experiment, starting, running, and transferring data. Schooner actually provides a capability to batch multiple runs of a scripted experiment. though you won't be using these capabilities anytime soon.

Your experiment should now be ready, so create it and swap it in.

4.3 DNS Experiment Configuration

In the real world the root zone and TLDs are served by different servers, but for our experiment we will collapse our fake root zone, and TLD of org and com into a single server. Node1 will be used to serve our zone book.org and node2 will server desk.com. The diagram in Figure 3 shows our delegation tree.

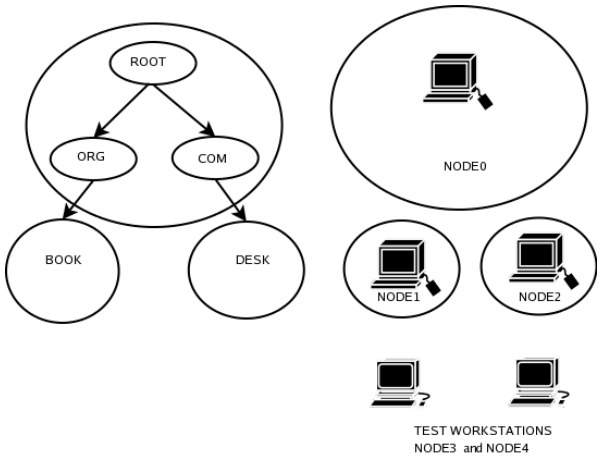

Figure 3: DNS hierarchy used in part 2.

Node3 and Node4 will be used as workstations to test resolving names later in the exercise.

4.3.1 Editing Configuration Files

The most common DNS server software today is BIND or Berkeley Internet Name Daemon, which has a program binary called *named*. Configuration of BIND is done through a series of files. The first file is named.conf, which is commonly in the /etc directory. Take a look at our example named.conf from node0, at the top of the file you will find the global options block.

```
options {
        directory "/var/named/";
        pid-file "/var/run/named/named.pid";
        recursion no;
};
```

These options affect all zones served. The rest of the file is a series of zone declarations. Here is our root zone declaration.

```
zone "." {
        type master;
        file "root";
};
```

This block declares the root zone, of type “master”, with a data or zone file “root”. From the previous global options block, the directory statement declares where the zone files are kept, so the root zone file is /var/named/root. Take a look at the root zone file, at the top we find a block

```
.       IN      SOA     dns. mail.dns. (
                        200710505       ; serial#
                        28800            ; refresh, seconds
                        7200            ; retry, seconds
                        604800            ; expire, seconds
                        86400 )          ; minimum, seconds
```

This is known as the SOA, you will find a number of different timing values which are used for various time-to-live fields used by DNS caches. The first number in the field though is the zone serial, which is used for exchanging the zone files between master and backup slave servers. Get in the habit, every time you make a change to the zone file increment the serial number. While the serial number is just an integer, many people commonly use a date such as ¡4DigitYear¿¡2DigitMonth¿¡2DigitDay¿¡2DigitRevision¿.

Below the SOA you will start to see the zone records, the first of which will be

```
.       IN      NS      dns.root-server.org
```

NS records are used to delegate parts of the DNS name-space. You might ask why does the root zone have to have a delegation record for itself? This is another part of the master/slave configuration, so don't worry about it for now. Further down you will see the delegation for the book.org zone.

```
book.org        NS      dns.book.org.
dns.book.org    A       192.168.2.2
```

The first record delegates responsibility for the book.org domain to the server dns.book.org. This brings us to our first problem - how can you delegate the responsibility to a server in book.org domain? You need the IP address of the DNS server to communicate with it, but you can only get that IP from that server. This is where the second line comes in. To solve this circular dependency, A records for a delegated zones DNS servers are added to the higher zone, these are called *glue records*.

Now take a look at some of the files on node1. The config file /etc/-named.conf is very similar to the one on node0 but has zone declarations for our additional zones. Look at the book.org file, you'll find some A records, like those you saw earlier.

```
hardcover.book.org   IN      A       192.168.2.2
                     IN      MX      0       mail.book.org

scifi.book.org       IN      A       192.168.2.3
                     IN      MX      0       virus-scanner-1.mail-security.com
```

Additionally, you see the MX records, an MX or mail exchanger records specifies what machine will handle electronic mail for the machine. In this case,

hardcover has it's mail handled by a machine mail.book.org, and scifi has it's mail directed to a totally separate domain. It is increasingly common place to outsource network services to outside companies. If I have a small number of machines, it's not cost effective to run say my own mail servers. It is much easier for me to pay a provider who can run a large service handling thousands of small domains.

The zone files we have been looking at map names to IP addresses, and are referred to as forward files. However, we can also do a reverse lookup to map an IP address to a name. For diagnostic purposes this is very important, and comes into play in a number of security scenarios. Instead of developing a second system, or even a separate root for the reverse entries, we can use our existing DNS setup. Reverse entries are mapped into the domain in-addr.arpa, and use a different record type (PTR). For example:

```
10:53am reed ~ (1)% host www.google.com
www.google.com is an alias for www.l.google.com.
www.l.google.com has address 64.233.167.99
...
10:53am reed ~ (2)% host 64.233.167.99
99.167.233.64.in-addr.arpa domain name pointer py-in-f99.google.com.
10:53am reed ~ (5)% host -t PTR 99.167.233.64.in-addr.arpa
99.167.233.64.in-addr.arpa domain name pointer py-in-f99.google.com.
```

First, we get an address for www.google.com. Using the host command we can ask host to do the reverse map, but the host program understands it's getting an IP address and knows how to change the name to the in-addr.arpa zone. We can the host program directly to request a PTR record for 99.167.233.64.in-addr.arpa, as is shown.

4.4 Tools, Utilities, Commands and Files

/etc/rc.d/init.d/named is the system script used to start and stop the DNS server program. For example:

```
mblodget@node0 ~]$ sudo /etc/rc.d/init.d/named start
Starting named:                                            [  OK  ]
[mblodget@node0 ~]$ sudo /etc/rc.d/init.d/named stop
Stopping named:                                            [  OK  ]
[mblodget@node0 ~]$ sudo /etc/rc.d/init.d/named restart
Stopping named:                                            [  OK  ]
Starting named:                                            [  OK  ]
```

rndc reload is the name server control utility. Read the man page for other commands beside reload. For example:

```
[mblodget@node0 ~]$ rndc reload
server reload successful
```

sudo named -u named -g -p 53 allows you to see any errors experienced on named startup.

sudo tail -f /var/log/messages /var/log/messages is the file named logs to - use it to watch for errors and troubleshoot named.

dig is the DNS lookup utility that shows details on a local host's DNS configuration. See the man page for dig.

host enables host name lookup using a built-in OS resolver.

5 Part 2 Tasks and Questions

- Test the name resolution of the current system (including reverse lookups) *e.g.,* using the dig command for hardcover.book.org, scifi.book.org and chair.desk.com (should return 10.57.0.4, 10.57.0.8 and 172.16.0.12 respectively). Using the dig command allows you to query a nameserver directly and bypass the stub resolver that is builtin to the operating systems of the localhost.

```
[blodge@node3 ~]$ dig @192.168.0.12 drawer.desk.com
....
;; QUESTION SECTION:
;drawer.desk.com.               IN      A

;; ANSWER SECTION:
drawer.desk.com.        259200  IN      A       172.16.0.14
.....
```

- Change the IP address of one of the nodes and make the necessary changes to DNS. Remember to update the serial and change both the forward and reverse zone files
- Refer to http://www.linuxhomenetworking.com/linux-hn/dns-static.htm to set up some canonical names.
- What would happen if the link running to node1 broke (take the interface down)? How could you prevent this problem?
- What happens if you lookup 'hardcover' (command: **host hardcover**) on node3? You will find it broken, what changes would you need to make it work? What changes would you have to make to have the command **host chair** work correctly on node3? What happens with the output of **host hardcover** after those changes?
- Which takes precedence DNS or the hosts file?
- Do you get errors if a name has two different A records? How about two names pointing to the same IP? Why might someone want to do this?