SUNWAY UNIVERSITY SCHOOL OF ENGINEERING AND TECHNOLOGY

Subject Code and Title: NET2103 NETWORK & SYSTEM ADMINISTRATION

Semester: August 2022

FINAL REPORT DEADLINE: 16/7/2023

Lecturer Name:	Prof. Ts. Yap Kian Meng
Tutor Name:	Ms. Teoh Jiehan
Group Number:	5

No.	Student Full Name	Student ID	Program Code	Signature
1	Ashwin Vigneswaran	21009527	BCNS	Ashwin
2	Chen Ern Khai	21006978	BCNS	Ern Khai
3	Elena Khoo Sze Kay	21012570	BCNS	Elena Khoo
4	Esther Leslie Bala	23039597	BCNS	Esther
5	Nidal Bencheikh Lehocine	19097617	BCNS	Nidal

We hereby swear that the work done on this assignment is our own and we have not given nor received aid that is inappropriate for this assignment. We understand that by the school code, violation of these principles will lead to a zero mark on this assignment.

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 AIMS & OBJECTIVES

The purpose of this report is to chew over the procedures involved in the configuration of a web server, a DNS server, a DHCP server, and a wireless access point, as well as additional features that are integrated to enhance the existing and newly added components in a Linux-based environment. Furthermore, the various implementation detail sections in the report will provide detailed documentation of the configurations and related discussions. This paper will demonstrate how these server components are integrated and implemented in Arch Linux, showcasing an extensive manual on the step-by-step configuration process for meeting the project requirements which includes details on hosting websites, managing DNS resolves, dynamically allocating IP addresses within a specific range, and setting up a wireless access point. In view of aims and objectives, this report hopes to provide a robust and effective infrastructure that will assist and guide people.

1.2 GENERAL REQUIREMENTS

In order to meet the general requirements, a Local Area Network (LAN) had to be constructed to be able to support web server hosting, DNS resolution, DHCP IP leasing, wireless client authentication, and web page retrieval. From an overview, users should be able to access the web pages after being redirected by the wireless access point created. A local web domain should be set up in the Apache web server to allow web browsing. By doing so, the clients should be able to access websites, like "www.website1.com". Next, the DNS replies must

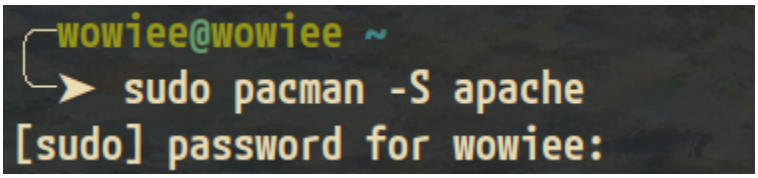
provide the reserved IP addresses by DHCP for hosting the desired web pages. The clients would be redirected to a local web domain once they pass the authentication through the wireless access point. The DNS response provided to the client would allow them to navigate to a web page from the local web domain that was established by the web server. Lastly, two additional features had to be integrated into the network scenario for tackling additional issues and improving the servers. Fulfilling these requirements aid in achieving the project goals stated. In this project, we use a laptop to host our services. This laptop is running Arch Linux bare-metal, kernel version 6.4.1. Below are the hardware specifications for our machine:

Since we are using an unconventional distribution of Linux to run our services, some commands and file locations will vary from those using a more common distro, such as Ubuntu. The most notable differences will be the locations and names of configuration files.

2.0 WEB SERVER

2.1 OVERVIEW (WEB SERVER)

A web server is a type of server that provides web content. These servers accept requests through HTTP(s) to provide content to users, who most commonly access these servers through web browsers. We chose to use the Apache HTTP Server as our web server software as it is free, open source, and well-maintained. Once installed, we were required to configure virtual hosting, maximum simultaneous users, and directory authentication. For monitoring the server's performance, we had to develop a shell script that periodically gathers information using tools and server-status modules. Lastly, the web server had to be configured to start automatically after the system reboot. On Arch Linux, Apache is installed by running the command "sudo pacman -S apache" in the terminal:



The command "sudo pacman -S apache" is executed with superuser privileges to instruct the package manager, "Pacman", to search for the Apache package in the repositories and install it into the system. It ensures that Apache is installed and implemented into the system properly to allow for setting up and hosting websites.

2.2 IMPLEMENTATION DETAILS (WEB SERVER)

2.2.1 REQUIREMENT 1 - VIRTUAL HOSTING

Virtual hosting is a way to host multiple websites on the same web server, each with different domain names. This allows us to use the same machine to host multiple websites, without requiring a new computer. In this case, we are using name-based virtual hosting to host two websites: site1 and site2. In name-based hosting, the two domain names are unique but share the same IP address. The domain names are used to 'point' to the site's source files stored on the web server. Firstly, a directory has to be created to contain the files used for the website. To demonstrate virtual hosting, two websites will be hosted on this server. Thus, two directories will be created in the "/var/www/htdocs" directory:

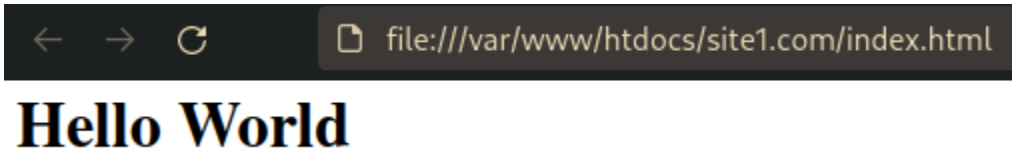
```
[root@wowiee htdocs]# pwd
/var/www/htdocs
[root@wowiee htdocs]# mkdir site1.com
[root@wowiee htdocs]# mkdir site2.com
[root@wowiee htdocs]# ls -l
total 8
drwxr-xr-x 2 root root 4096 Jul  4 12:40 site1.com
drwxr-xr-x 2 root root 4096 Jul  4 12:40 site2.com
```

We also had to specify these directories as the ones Apache needs to use. We can do that by modifying "/etc/httpd/httpd.conf":

```
256 DocumentRoot "/var/www/htdocs"
```

Next, we added HTML index files in respective directories and we used a simple “Hello World” output for demonstration purposes:

```
[root@wowiee site1.com]# rm index.html
[root@wowiee site1.com]# touch index.html
[root@wowiee site1.com]# echo "<h1>Hello World</h1>" >> index.html
```



The same index file can be copied into the “site2” directory using “cp” for ease and reduced redundancy of commands:

```
[root@wowiee site1.com]# cp index.html ../site2.com/
```

(Of course, we made some slight differences to the second site in order to tell them apart).

To enable virtual hosting, we have to configure the “httpd” configuration file that is located at “/etc/httpd/conf/”. We needed to uncomment line 515, which showed “Include conf/extra/httpd-vhosts.conf” to include and activate the additional virtual host configuration file, “httpd-vhosts.conf”. This separate configuration file defines the virtual hosts in Apache. By uncommenting that, we are instructing Apache to include and process the virtual host configurations defined in the “httpd-vhosts.conf” file. This enables us to set up and manage multiple websites or domains on the server:

```
[root@wowiee ~]# cd /etc/httpd/conf/
[root@wowiee conf]# vim httpd.conf
```

```
514 # Virtual hosts
515 Include conf/extra/httpd-vhosts.conf
```

After enabling virtual hosting, we had to set up and configure the virtual hosts by modifying the “httpd-vhosts.conf” file:

```
[root@wowiee conf]# vim extra/httpd-vhosts.conf
```

```
<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host.example.com
    DocumentRoot "/var/www/htdocs/site1.com/"
    ServerName group5.xyz
    ServerAlias www.group5.xyz
    ErrorLog "/var/log/httpd/www.group5.xyz-error_log"
    CustomLog "/var/log/httpd/www.group5.xyz-access_log" common
</VirtualHost>

<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host2.example.com
    DocumentRoot "/var/www/htdocs/site2.com/"
    ServerName group5.lol
    ServerAlias www.group5.lol
    ErrorLog "/var/log/httpd/blog.group5.xyz-error_log"
    CustomLog "/var/log/httpd/blog.group5.xyz-access_log" common
</VirtualHost>
```

The most notable configuration is the “DocumentRoot”, which contains the path to the directories we created earlier with the HTML index files. Apache will look into these directories for the index files. We also added a “ServerAlias", which basically redirects the user to the site if they enter the alias (www.group5.xyz). The “ServerName" directive specifies that the virtual host is associated with the domain name. When a request comes in from the respective domain name, Apache will know how to serve requests for website content from the appropriate directory.

It is advisable to check for any errors after and during the configuration process. The “apachectl configtest” command is used to check the syntax of the Apache configuration files for any errors or issues.

```
[root@wowiee conf]# apachectl configtest
Syntax OK
```

When you run this command, Apache reads the configuration files and performs a syntax check to ensure that the configuration is valid. It provides feedback by displaying a message. In our case, everything has been configured properly so the syntax displayed is “OK”. However, in the case of errors, the feedback will provide a message specifying the problem and the location in the configuration files where the error was found.

Once this is done, we had to put our websites’ domain names and addresses into the hosts' files. This file is located in “/etc/hosts”:

```
[root@wowiee ~]# vim /etc/hosts
```

```
1 # Static table lookup for hostnames.
2 # See hosts(5) for details.
3 #
4 127.0.0.1 localhost
5 127.0.1.1 wowiee@wowiee
6
7 192.168.5.10 group5.xyz
8 192.168.5.10 group5.lol
```

Next, we assigned a network interface to host the webserver, whose IP address is 192.168.5.10. We still wanted to use our primary network interface to use the internet for testing, so we hosted our web server on a virtual network interface:

Enabling the dummy kernel module:

```
[root@wowiee wowiee]# modprobe dummy
```

Adding an interface named ‘web’:

```
[root@wowiee wowiee]# ip link add web type dummy
```

Assigning our web server IP address to the web interface, and bringing the interface up:

```
[root@wowiee wowiee]# ip addr add 192.168.5.10/24 brd + dev web
[root@wowiee wowiee]# ip link set dev web up
```

We can run “ifconfig” to check if the interface has been successfully created:

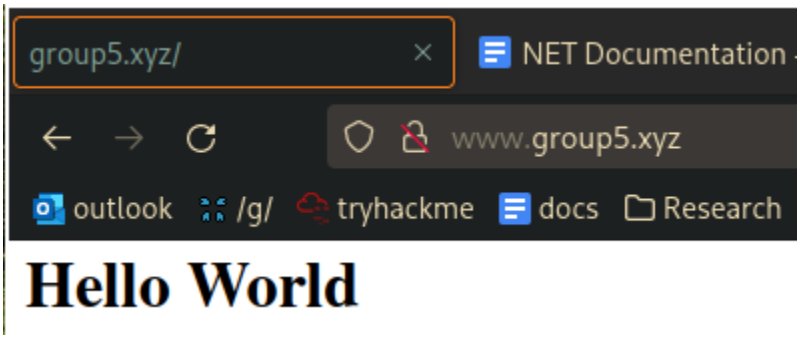
```
web: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 1500
        inet 192.168.5.10  netmask 255.255.255.0  broadcast 192.168.5.255
        inet6 fe80::1066:ebff:fe8c:4a0d  prefixlen 64  scopeid 0x20<link>
        ether 12:66:eb:8c:4a:0d  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3  bytes 210 (210.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Then, we restarted Apache so the changes will take effect:

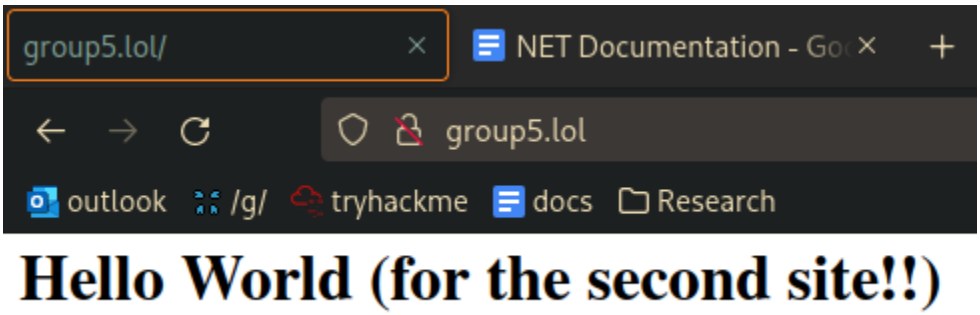
```
[root@wowiee wowiee]# systemctl restart httpd
```

To check if everything worked as intended, we simply used our browser to check our websites:

For our first website:



As shown here, we used “www.group5.xyz” instead of just “group5.xyz”. This is to prove that the server alias is working. Now, for checking the second website:



With this, it was confirmed that we had gotten our virtual hosts to work.

2.2.2 REQUIREMENT 2 - LIMITING SIMULTANEOUS USERS

We limited the maximum amount of simultaneous users by modifying the “httpd-mpm.conf” file:

```
[root@wowiee ~]# vim /etc/httpd/conf/extra/httpd-mpm.conf
```

Under the “mpm_prefork_module” section, we specified the number of concurrent connections to the web server with the “MaxClients” parameter:

```
28 <IfModule mpm_prefork_module>
29     StartServers          5
30     MinSpareServers       5
31     MaxSpareServers      10
32     MaxClients            2
33     MaxRequestsPerChild   0
34 </IfModule>
```

2.2.3 REQUIREMENT 3 - DIRECTORY AUTHENTICATION

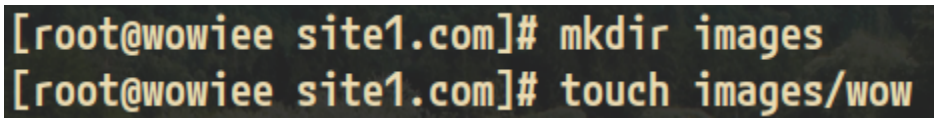
To set up password authentication, we needed to use “htpasswd” (which came with the Apache package) to generate a “.htpasswd” file. We set this password for the “admin” user:

```
[root@wowiee httpd]# htpasswd -c /etc/httpd/.htpasswd admin
New password:
```

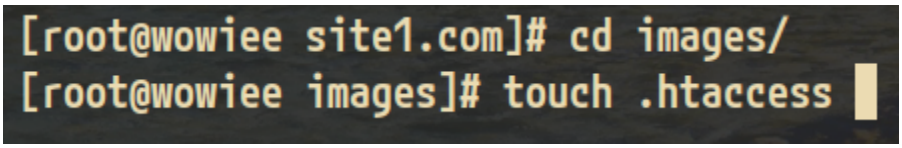
We were prompted to enter a password for the admin user. We chose to use “funnypants” as our password. We need to retype the password before it is added to the file:

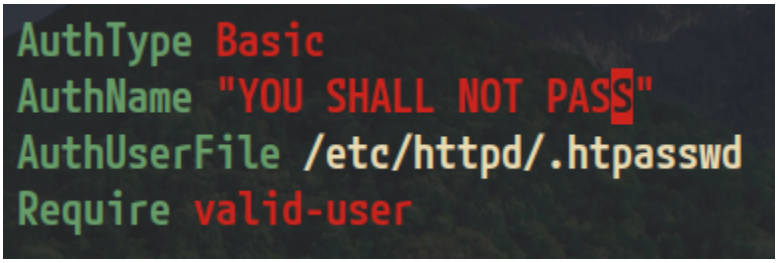
```
Re-type new password:
Adding password for user admin
```

Next, we needed to create the password-protected directory. We created a directory at "/var/www/htdocs/site1.com/images". Then, for testing, we created a file in this directory:



To set our password authentication to apply only for this directory, we had to create a ".htaccess" file at our file's document root to be "/var/www/htdocs/site1.com/images/.htaccess". We also had to modify this file:





"AuthType" specified the type of authentication used. It could be basic (username and password), none, or any of the other types that come with the Apache modules. "AuthUserFile" points to the ".htpasswd" file we just made.

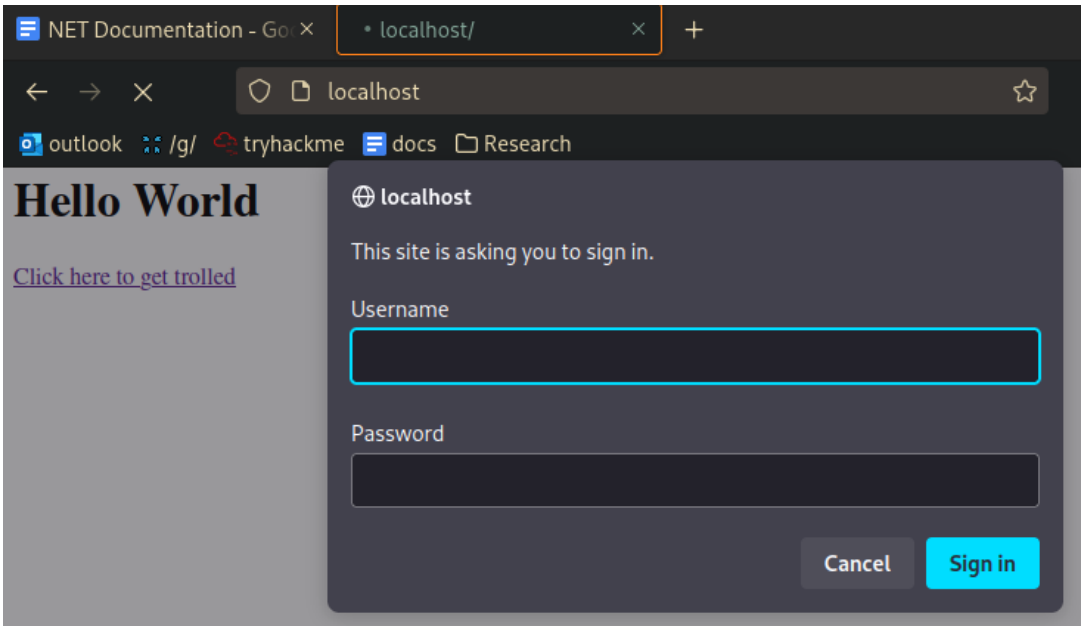
We also had to set “AllowOverride” to “All” in “/etc/httpd/conf/httpd.conf”:

```
273     # AllowOverride controls what directives may be placed in .htaccess files.
274     # It can be "All", "None", or any combination of the keywords:
275     #   AllowOverride FileInfo AuthConfig Limit
276     #
277     AllowOverride All
```

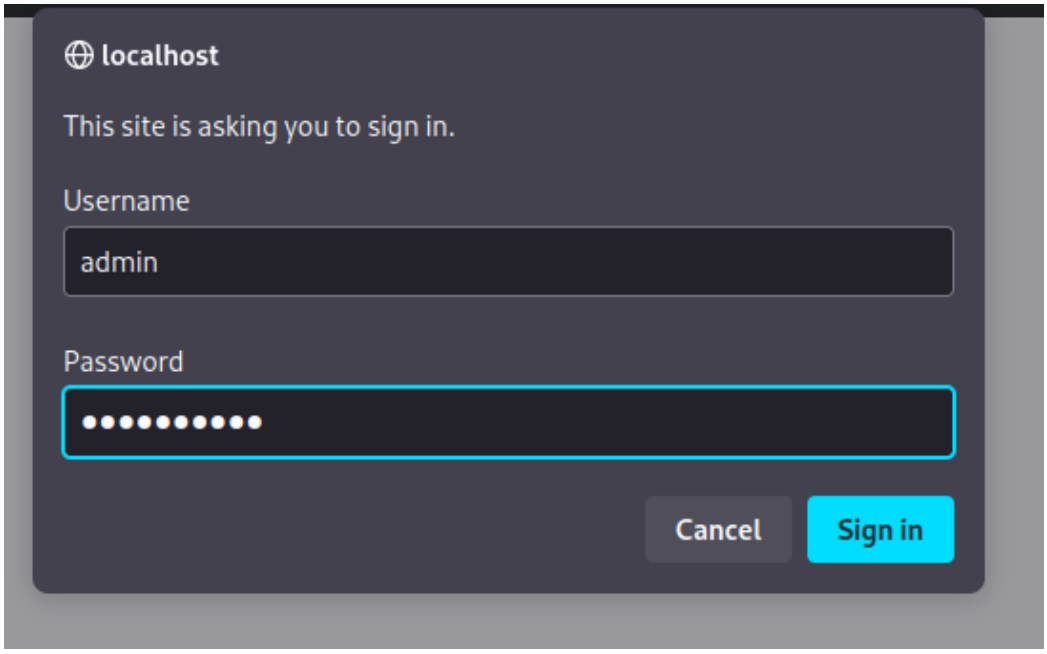
This allowed our “.htaccess” file to take effect. To apply our changes, we restarted Apache again:

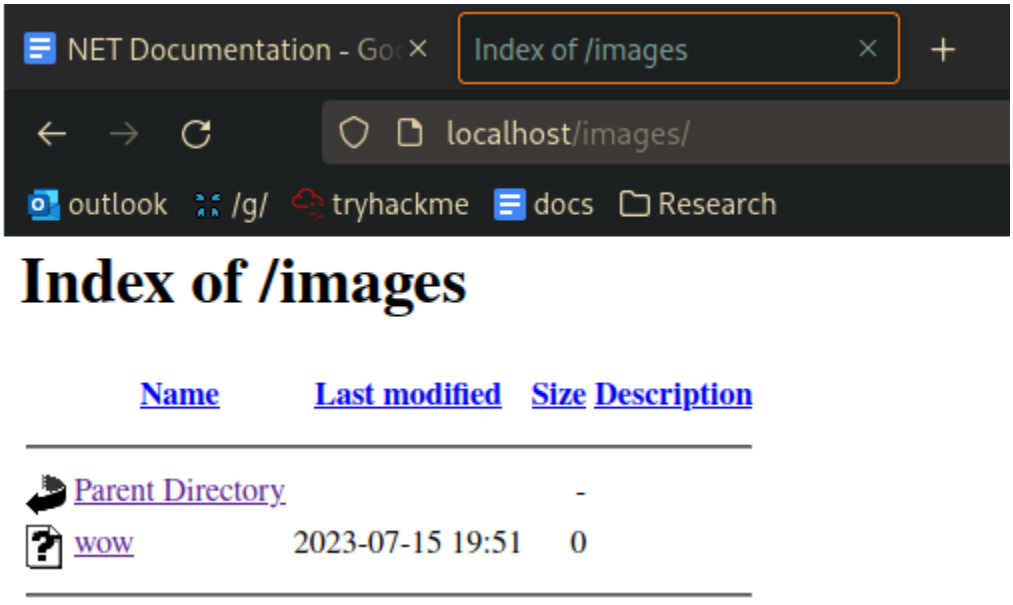
```
[root@wowiee htdocs]# systemctl restart httpd
```

To test if the authentication configuration worked, we navigated to the directory in our browser and tried to access it by clicking on the hyperlink:



Upon seeing this prompt, we entered the correct credentials:





This confirmed that our directory authentication works successfully.

2.2.4 REQUIREMENT 4 - SERVER PERFORMANCE SHELL SCRIPT

We were required to create a shell script to help monitor our web server. Our shell script code will be focused on monitoring and keeping track of the connected client(s), CPU usage, bandwidth utilization, and uptime. In our home directory, we used the command “sudo vim apache_monitor.sh” to create the shell script to monitor the performance criteria. We specified the appropriate interfaces and ports that Apache was running on in order to obtain the specific and correct response. The following is the code and output for bandwidth utilization:

```
21 echo BANDWIDTH
22 INTERFACE=enp0s3
23 echo "$INTERFACE down (Kib/s) up (Kib/s)"
24 while :
25 do
26 awk '{
27 if (rx) {
28         printf (" %.0f   %.0f\n", ($2-rx)/1024, ($10-tx)/1024)
29 }       else {
30         rx=$2; tx=$10;
31 }
32 }' \
33         <(grep $INTERFACE /proc/net/dev) \
34         <(sleep 1; grep $INTERFACE /proc/net/dev)
35         sleep 2;
36 done
37
```

```
BANDWIDTH
enp0s3 down (Kib/s) up (Kib/s)
 362   20
 390   14
 1   3
```

To calculate our bandwidth, we created a while loop that keeps running and updating periodically. Firstly, we defined our interface name and set columns for the data to be displayed. Inside the while loop, we used the awk command which will process the network statistics: “rx” received data and “tx” the transmitted data. It would calculate the differences between the previously stored values and the currently stored values. After that, it calculates the rate in kb/s. If no value is stored currently, it will use the previous value. Then, “grep” is used to read the contents of the “proc/net/dev” file from the specified interface and it will extract the data. The loop will stop for 1 second before redoing the process again after each execution. The loop pauses for 2 seconds and runs again to provide continuous feedback and response on the bandwidth usage.

On the other hand, this is the code and the output for the amount of currently connected client(s):

```
14 echo CONNECTED CLIENTS
15 connected_clients=$(netstat -ant | grep ":80" | wc -l)
16 echo $connected_clients
17 echo " "
```

```
CONNECTED CLIENTS
1
```

To calculate the number of connected clients, we used “netstat -ant” to display all the TCP connections as well as their addresses. It is redirected to the grep command which reads all the connections displayed that are directed through port 80. Lastly, the “wc” command counts the number of lines and that will display the number of connected clients for us. We used port 80 as it is commonly associated with Apache and other web servers because it is the default port for serving HTTP traffic. Apache listens on port 80 by default to handle incoming HTTP requests.

The following is the code and output for monitoring the CPU usage of the Apache web server:

```
18 echo CPU USAGE
19 grep 'cpu ' /proc/stat | awk '{usage=($2+$4)*100/($2+$4+$5)} END {print usage
   "%"}'
20 echo " "
```

```
CPU USAGE
2.06067%
```

The grep command will read the CPU statistics from the /proc/stat file and pass it on to the awk command. The second, fourth, and fifth fields are used to calculate the CPU usage. The second (user CPU time) and fourth (system CPU time) fields are summed up and multiplied by 100. They are then divided by the sum of the user’s system and idle CPU time. The end result is

stored in a user variable and its percentage is printed out. The process ID (PID) is taken from the "apache2" process.

The following is the code and output for monitoring the uptime of the Apache web server upon startup of the system:

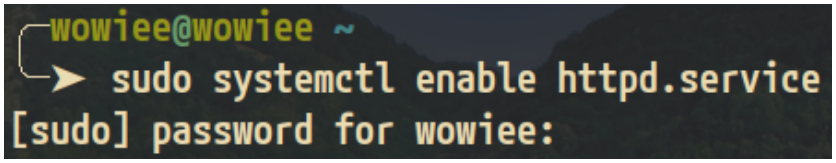
```
echo " "
echo "UPTIME"

if systemctl is-active --quiet apache2; then
        echo " "
        echo "Apache2 is running."
        echo " "

        start_time=$(systemctl show -p ExecMainStartTimestamp --value apache2)

        current_time=$(date +%s)

        uptime=$((current_time - $(date -d "$start_time" +%s)))

        days=$((uptime / 86400))
        seconds=$((uptime % 60))
        minutes=$((uptime % 3600 / 60))
        hours=$((uptime % 86400 / 3600))


        echo "Uptime: $days days, $hours hours, $minutes minutes, $seconds sec>
        echo " "
else
        echo " "
        echo "Apache2 is not running"
```

```
UPTIME

Apache2 is running.

Uptime: 0 days, 0 hours, 42 minutes, 48 seconds.
```

The created shell script allows us to check the uptime of the “apache2” web server. It begins by checking the status of “apache2” using the “systemctl” command, which determines if the web server is currently running. If “apache2” is active, the script proceeds to retrieve the start time of the service using “systemctl” show. To calculate the uptime, the script captures the current time using the date command and converts it to the Unix timestamp format. The difference between the current time and the start time is then computed, representing the uptime of the “apache2” service in seconds. The script further breaks down the uptime into days, hours, minutes, and seconds for a more human-readable format. This information offers valuable insights into how long the “apache2” web server has been continuously operational. By utilizing this script, we can efficiently track the uptime of the web server to ensure service availability.

2.2.5 REQUIREMENT 5 - AUTOMATIC START ON REBOOT

Lastly, enabling the “httpd.service” ensures that Apache starts automatically upon reboot, allowing websites and applications hosted on the server to be accessible without manual intervention. The command creates symbolic links that tell “systemctl” to start the service during system startup. Below is the command to ensure that the Apache service starts on reboot:



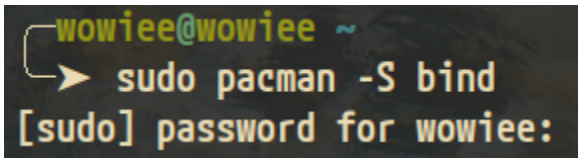
2.3 CONFIGURATION CHALLENGES (WEB SERVER)

During the configuration of the web server, several challenges were encountered and addressed. One significant challenge was implementing virtual hosting to serve multiple websites using the same IP address. This required careful and precise configuration of Apache's configuration files like “httpd.conf” and “httpd-vhosts.conf” in order to make virtual hosting work effectively. Not to mention, developing the shell script to monitor the web server’s performance posed quite a challenge as in-depth knowledge of command-line tools and scripting was needed. Extracting and presenting real-time information about connected clients, uptime, bandwidth usage, and cpu usage demanded creative approaches and interpretation of data with help from various sources and articles across the web.

3.0 DNS SERVER

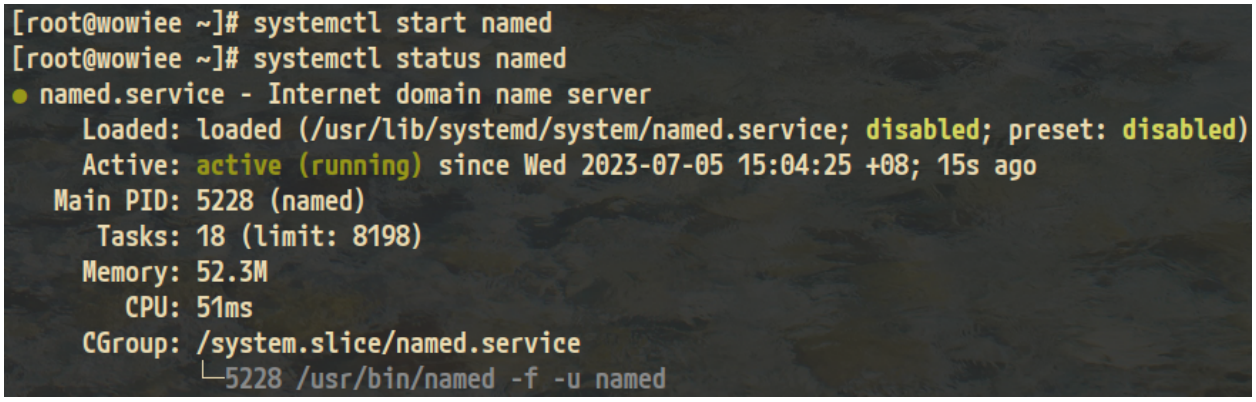
3.1 OVERVIEW (DNS SERVER)

A DNS (Domain Name System) server is another type of server that controls the hosting of web services. DNS servers are responsible for 'redirecting' users to the IP address corresponding to the domain name (e.g., www.google.com) entered in the web browser (known as DNS queries). For our server, we used the BIND or Berkeley Internet Name Daemon. We chose BIND because it is widely used, and is the standard for Linux and Unix servers. According to the requirements, we were required to set up forward and reverse zones, and different aliases to the previously created two web domains, as well as ensure the DNS server starts automatically after rebooting the server. On Arch Linux, BIND is installed by running the command "sudo pacman -S bind" in the terminal:



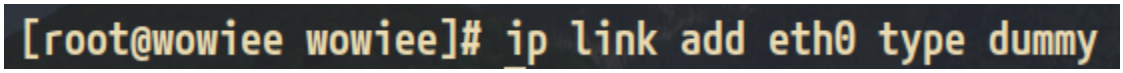
In this context, the command is used to install the BIND (Berkeley Internet Name Domain) package. By executing sudo pacman -S bind, we are instructing the package manager to download and install the BIND package on our system.

After installing BIND, we use the command “systemctl start named” to start the service and “systemctl status named” to check the status of the service. “Named” is the daemon service used to run BIND. This ensures the service is properly up and running to proceed with the configurations:



We decided to use a virtual interface “eth0”, with the specific IP address of “192.168.5.20” for the DNS server. Using a virtual interface means that we created a separate interface apart from the primary network interface. This approach ensures that the reserved IP address is dedicated exclusively to the DNS server, allowing for clear identification. We made use of the commands below to create the virtual interface:

Adding a dummy (virtual) interface named “eth0”:



Assigning an IP address to the virtual interface:

```
[root@wowiee wowiee]# ip addr add 192.168.5.20/24 brd + dev eth0
```

Bringing the interface up and checking its details with “ifconfig”:

```
[root@wowiee wowiee]# ip link set dev eth0 up
```

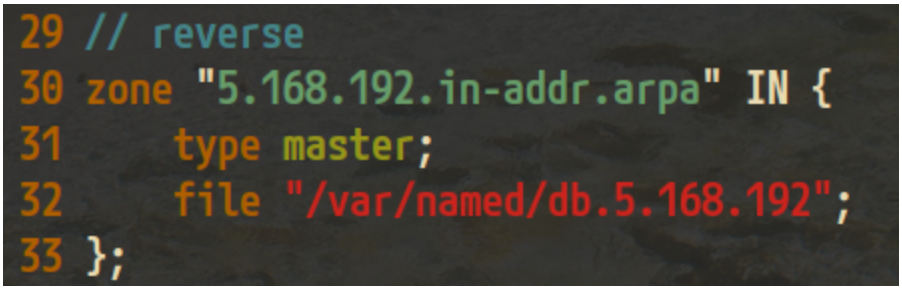
```
[root@wowiee wowiee]# ifconfig
eth0: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 1500
        inet 192.168.5.20  netmask 255.255.255.0  broadcast 192.168.5.255
        ether 96:b0:0f:48:37:48  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 420 (420.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

By utilizing this virtual interface, we configured the DNS server to bind specifically to that IP address, ensuring all DNS-related requests are handled by that interface. This can also lead to better network management and troubleshooting in the future.

3.2 IMPLEMENTATION DETAILS (DNS SERVER)

3.2.1 REQUIREMENT 1 - FORWARD AND REVERSE ZONES

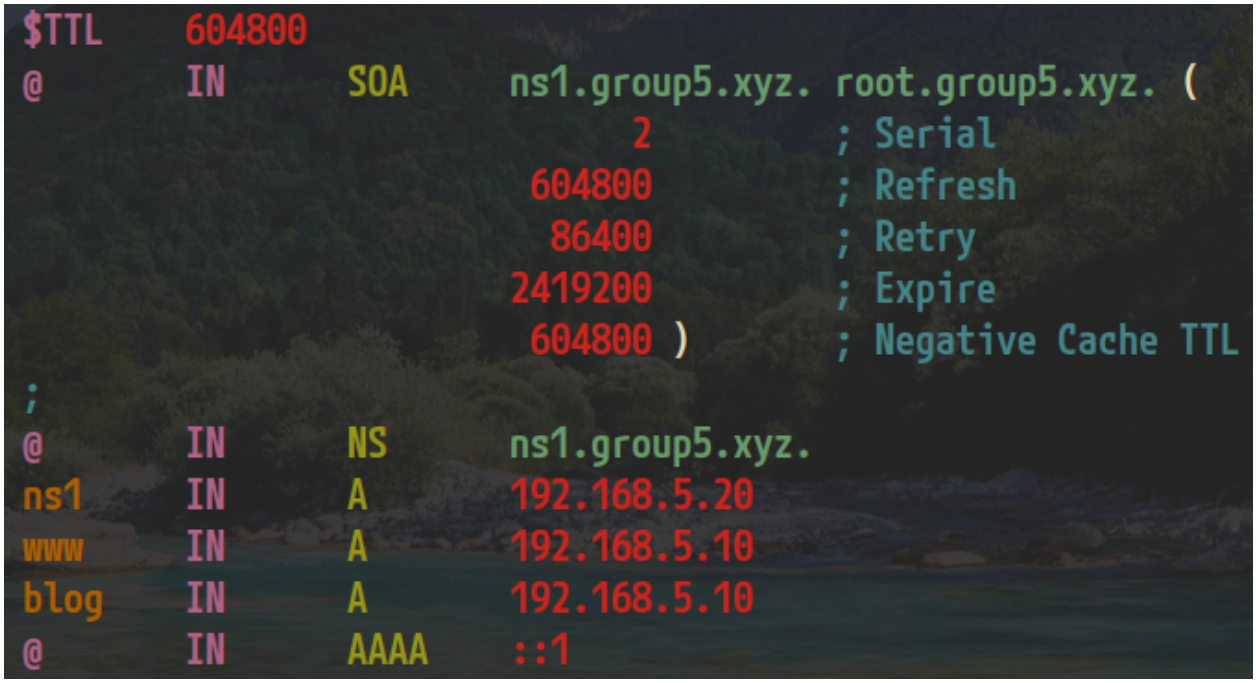
We had to create forward and reverse zones and define them properly in the respective configuration files. Firstly, we defined the forward and reverse zones in the "/etc/named.conf" configuration file:



We defined forward and reverse zones in the “named.conf” configuration file to specify configuration details, such as the zone type which is master. It also clearly specifies the location of the zone files for forward and reverse. This allows the DNS to properly handle requests and provide accurate DNS resolution. For example, the reverse zone is specified as "zone "5.168.192.in-addr.arpa" IN" is used to handle reverse DNS lookups. In reverse DNS, the IP address is reversed and appended with the "in-addr.arpa" domain. So, for the IP address 192.168.5.10, the reverse DNS zone would be "10.5.168.192.in-addr.arpa". The type “master" indicates that this DNS server is the primary server for the reverse zone. It holds the authoritative data and responds to queries for reverse lookups within this zone. The file directive specifies the location of the zone file that contains the mapping of IP addresses to domain names for the reverse zone. In this case, the file is "db.5.168.192". For the forward zone, we are using “db.group5.xyz” as the file. The “db” prefix refers to the database. DNS databases are used to store and manage DNS zone information.

3.2.2 REQUIREMENT 2 - DIFFERENT ALIASES

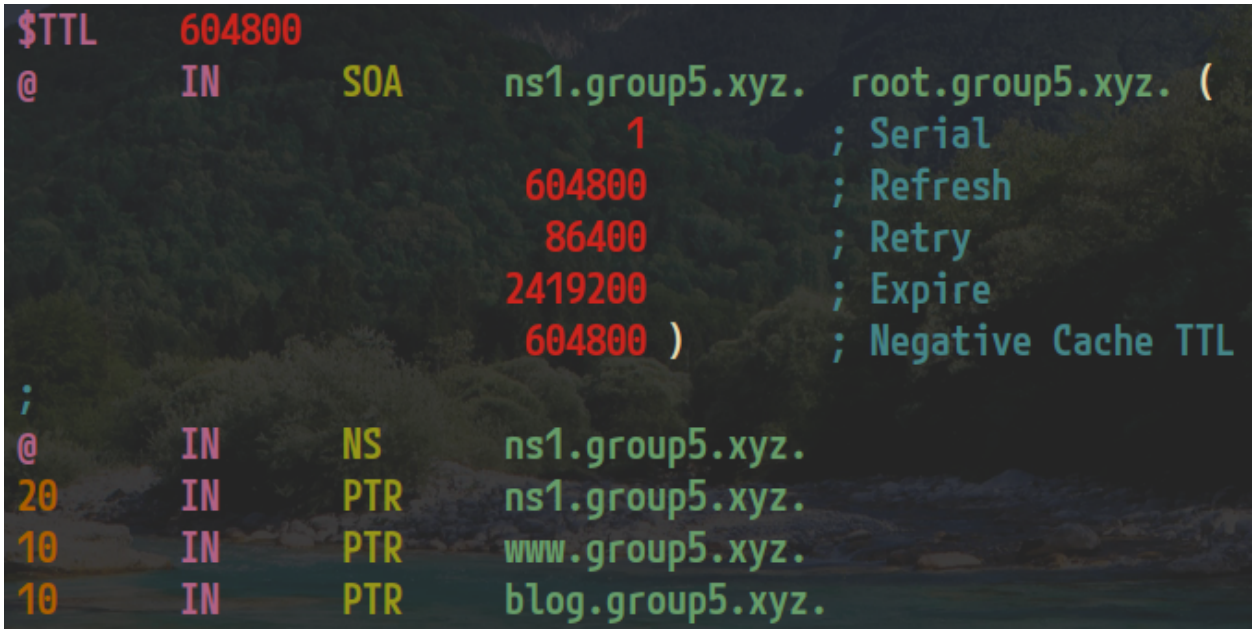
After that, we create the forward zone file "db.group5.xyz" in the "/var/named/" directory. Below is the file for the forward zone file storing DNS data:



The first "@" line specifies the name server (NS) record for the domain, indicating the primary DNS server responsible for resolving queries for this zone which is "ns1.group5.xyz". The next line maps the domain name "ns1" to the IP address specified in the address record. In this case, the IP address is "192.168.5.20", which is the reserved IP address for the DNS server. Following that, additional mappings can be added for specific subdomains or hosts within the domain. For example, we mapped "www" to the IP address "192.168.5.10", which is the reserved IP address of the web server. By configuring the forward zone file in this manner, the DNS server will

respond with the appropriate IP address when queried for the domain name “ns1.group5.xyz” or its subdomains.

We also created the reverse zone file “db.5.168.192” in the “/var/named/” directory. Below is the file for the reverse zone storing the DNS data:



The first “@” line specifies the name server (NS) record for the domain, indicating the primary DNS server responsible for resolving queries for this zone which is “ns1.group5.xyz”. The IP address ending with 20 is mapped to the domain name “ns1.group5.xyz” which is our DNS, whereas the IP address ending with 10 is mapped to the web server. Taking one example would be “www.group5.xyz”. When queried with an IP address, the DNS server will provide the corresponding domain name, allowing reverse resolution for clients and other DNS servers.

It is advisable to check the configuration files for any errors. In order to do this, we can use the command “named-checkconf” to check for any errors. Since there is no output, this indicates that there is no error:

```
[root@wowiee etc]# named-checkconf
```

In order to check if the zones are functional or not, we ran the command “named-checkzone www.group5.xyz db.group5.xyz” and “named-checkzone www.group5.xyz db.5.168.192”. We proceeded after there was no error and the syntax displayed “OK”:

```
[root@wowiee named]# named-checkzone www.group5.xyz db.group5.xyz
zone www.group5.xyz/IN: loaded serial 2
OK
```

```
[root@wowiee named]# named-checkzone www.group5.xyz db.5.168.192
zone www.group5.xyz/IN: loaded serial 1
OK
```

Of course, we repeated the steps to check for the subdomain as well:

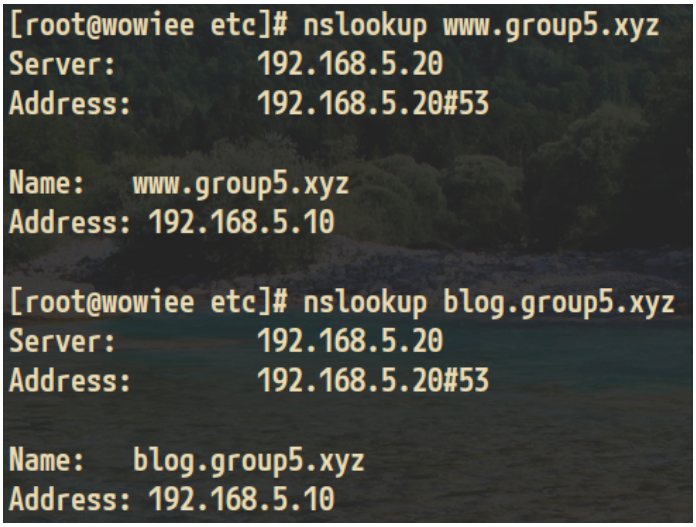
```
[root@wowiee named]# named-checkzone blog.group5.xyz db.group5.xyz
zone blog.group5.xyz/IN: loaded serial 2
OK
[root@wowiee named]# named-checkzone blog.group5.xyz db.5.168.192
zone blog.group5.xyz/IN: loaded serial 1
OK
```

Next, we used “nslookup” to check if our DNS server is working. The command “nslookup” stands for ‘Name Server Lookup’, and is used to query internet name servers interactively. However, before we did this, we needed to add the IP address of our DNS to “/etc/resolv.conf”:

```
# Generated by dhcpcd from wlp2s0.dhcp, wlp2s0.ra
# /etc/resolv.conf.head can replace this line
nameserver 192.168.5.20
nameserver 8.8.8.8
search localdomain
# /etc/resolv.conf.tail can replace this line
```

The file “resolv.conf” is what the system uses to determine which name servers to use. As shown above, the first address listed belongs to our DNS at 192.168.5.20. The next name server, 8.8.8.8, which is Google’s DNS IP address, will be used if the first one fails.

After completing this, we finally ran “nslookup” on both the domain and subdomain:



As shown by the output, our DNS configurations are functional. “192.168.5.10” IP address is correctly being used for our web server, and “192.168.5.20” IP address is being used for our DNS server.

We also repeated these steps for another zone file, which was necessary in order to have our Apache virtual host for the second domain (not subdomain) work:

```
// forward 2
zone "group5.lol" IN {
    type master;
    file "/var/named/db.group5.lol";
};

// reverse2
zone "5.168.192.in-addr.arpa" IN {
    type master;
    file "/var/named/db.5.168.192a";
};
```

```
[root@wowiee named]# cp db.group5.xyz db.group5.lol
[root@wowiee named]# vim db.group5.lol
```

```
$TTL    604800
@       IN      SOA     ns1.group5.lol. root.group5.lol. (
                              0716        ; Serial
                         604800           ; Refresh
                          86400           ; Retry
                        2419200           ; Expire
                         604800 )         ; Negative Cache TTL
;
@       IN      NS      ns1.group5.lol.
ns1     IN      A       192.168.5.20
www     IN      A       192.168.5.10
@       IN      AAAA    ::1
```

```
[root@wowiee named]# cp db.5.168.192 db.5.168.192a
```

```
$TTL    604800
@       IN      SOA     ns1.group5.xyz.  root.group5.xyz. (
                              0715        ; Serial
                         604800           ; Refresh
                          86400           ; Retry
                        2419200           ; Expire
                         604800 )         ; Negative Cache TTL
;
@       IN      NS      ns1.group5.xyz.
20      IN      PTR     ns1.group5.xyz.
10      IN      PTR     www.group5.lol.
10      IN      PTR     blog.group5.lol.
```

3.2.3 REQUIREMENT 3 - AUTOMATIC START ON REBOOT

Lastly, enabling the “named” service ensures the DNS server starts automatically upon reboot, allowing the DNS server to be available and operational whenever the system starts up. The command makes the “named” service become active during system startup unless manually stopped or disabled. Below is the command to ensure that the “named” service starts on reboot:

```
[root@wowiee named]# systemctl enable named
```

3.3 CONFIGURATION CHALLENGES (DNS SERVER)

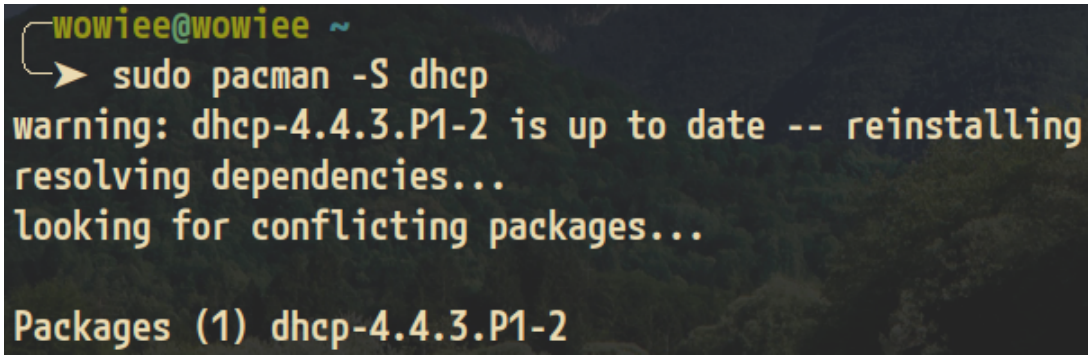
During the DNS server configuration for the assignment, there were some configuration challenges that we encountered. We had problems with the zone file content accuracy which lead to “nslookup” errors. It was important to constantly ensure that the zone file had up-to-date information and we had troubles with the mapping between domain names and IP addresses in the forward and reverse zones. This resulted in DNS resolution failures. Furthermore, it was crucial to keep track of the correct syntax as we faced syntax errors that prevented the DNS server from functioning properly. Verifying the configuration for accuracy and proper syntax is important. To overcome these challenges, it is essential to carefully review the configuration files, verify the accuracy of the zone file content, and ensure proper syntax. Commands like “named-checkconf” helped us to identify issues early on during the testing and monitoring phase.

4.0 DHCP SERVER

4.1 OVERVIEW (DHCP SERVER)

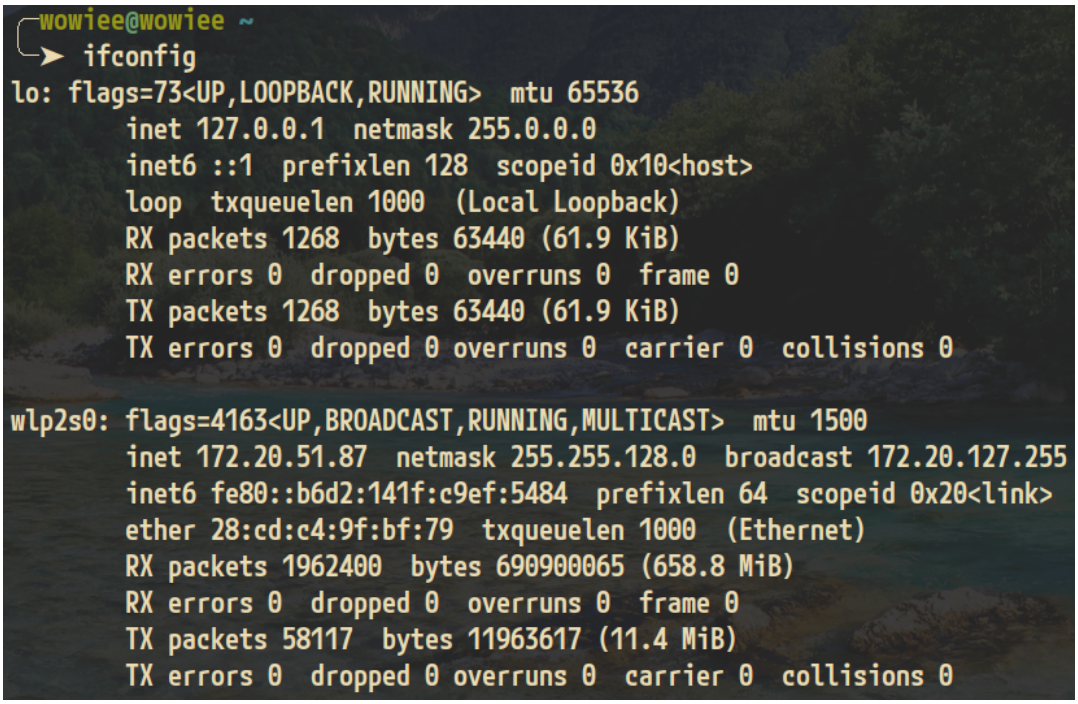
The DHCP (Dynamic Host Configuration Protocol) configuration involves setting up a DHCP server to provide IP addresses and network configuration parameters to clients on a network. The step-by-step process establishes a functional DHCP server that assigns IP addresses dynamically, reserves addresses for specific servers and provides necessary network settings to clients, ensuring proper network connectivity within the assigned IP address range. Also, we had to specify and configure the appropriate lease times by DHCP as part of the requirement.

Firstly, install a DHCP client. We chose the “dhcp” client. Although it is older, it was still sufficient for us:



We used the command “sudo pacman -S dhcp” in order to install the DHCP client onto our system. The DHCP package is fetched from the official repositories and it is installed in our

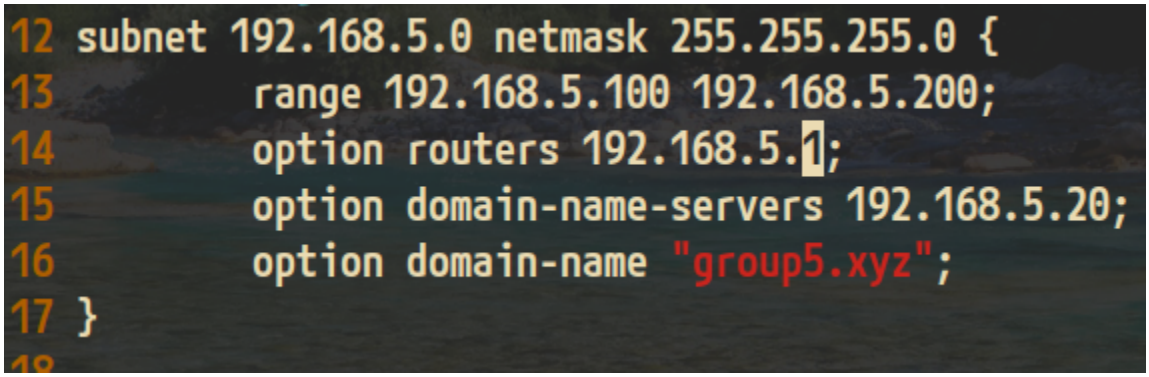
system. Moving on, we listed down the network interfaces using the “ifconfig” command. This showed us the loopback interface, “lo”, and our primary interface, “wlp2s0”:



4.2 IMPLEMENTATION DETAILS (DHCP SERVER)

4.2.1 REQUIREMENT 1 - IP ADDRESSING RANGE

According to the requirements, we had to configure the DHCP server to offer IP addresses in the given range of “192.168.5.100/24 - 192.168.5.200”. As we are group 5, we changed the network subnet to our group number to avoid conflicts with other groups. We added this configuration to the “dhcpd.conf” configuration file:



The “subnet 192.168.5.0 netmask 255.255.255.0” block defines the subnet and its corresponding netmask. We specified the range by using “range 192.168.5.100 192.168.5.200” for the specified subnet and netmask. This specifies the range of IP addresses that the DHCP can assign to its clients. In order to set the default gateway IP address that will be assigned to clients as their router, we used “option routers 192.168.5.30” to point to the address of our DHCP server. We

also had to create another dummy interface for the DHCP server. We repeated the steps taken to do this:

```
[root@wowiee wowiee]# ip link add eth2 type dummy
```

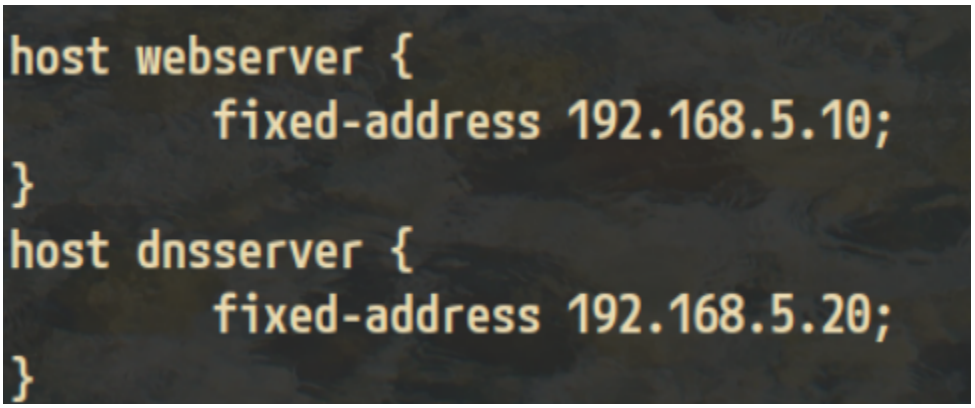
```
[root@wowiee hostapd]# ip addr add 192.168.5.1/24 brd + dev eth2
```

```
[root@wowiee hostapd]# ip link set dev eth2 up
```

We assigned the address “192.168.5.1” for our DHCP server.

4.2.2 REQUIREMENT 2 - RESERVED IP ADDRESSES

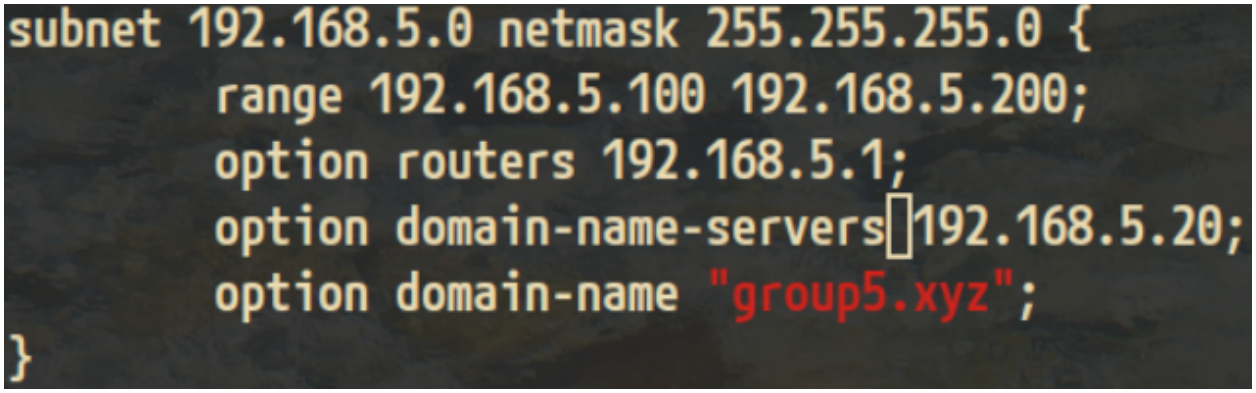
We also had to reserve the following IP addresses “192.168.5.10” and “192.168.5.20” for the web and DNS servers, respectively. In the “dhcpd.conf” file, we defined the specific host entries using the “host” directive. In this case, we have two host entries: “webserver” and “dnsserver”. The “webserver” and “dnsserver” entries represent specific devices on the network that require reserved IP addresses. These reserved IP addresses ensure that the web server and DNS server always receive the same IP addresses each time they request an address from the DHCP server:



The “fixed-address” directive assigns a fixed IP address of “192.168.5.10” to the web server and a fixed IP address of “192.168.5.20” to the DNS server. By defining these host entries, we ensured that the web server and DNS server receive the reserved IP addresses whenever they communicate with the DHCP server. This configuration ensures that these servers have consistent and predictable IP addresses, which can be useful for services that rely on fixed IP addresses for proper functioning.

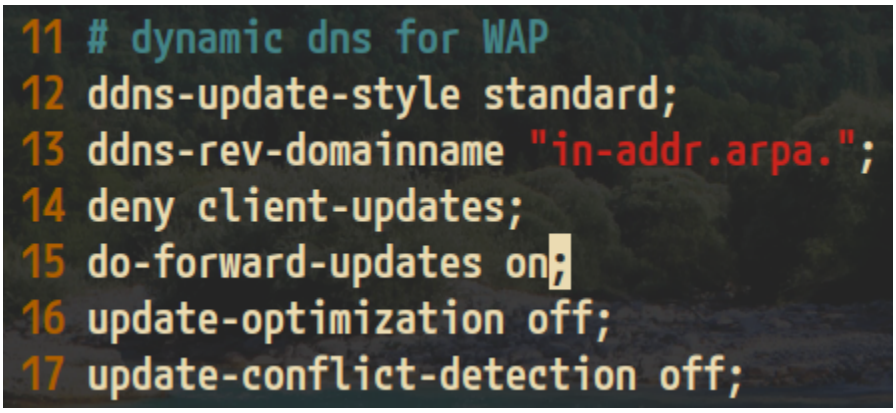
4.2.3 REQUIREMENT 3 - DNS IP LEASING

We made sure that the DHCP server includes the DNS server IP address in its leases to clients by adding “option domain-name-servers 192.168.5.20” into the “dhcpd.conf” configuration file:



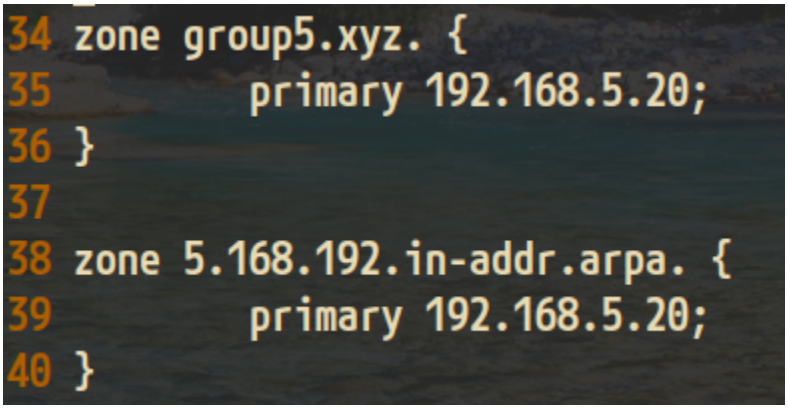
The “option domain-name-servers” directive will specify the IP address of the DNS server that will be provided to the clients. By providing the “option domain-name-servers in the DHCP configuration, we ensured that DHCP clients are configured with the correct DNS server IP address. This allows them to perform DNS lookups and resolve domain names to IP addresses on the network. The “option domain-name” directive is used to specify the domain name that will be assigned to DHCP clients. This option provides the DNS domain name that clients should be using when resolving hostnames on the network. The domain name “group5.xyz” is specified as the option value. DHCP clients that receive an IP address from this DHCP server will be configured with the domain name “group5.xyz”.

To allow our DNS and DHCP server to work together, we had to create a “dynamic DNS server” or DDNS. To do this, we first modified our “/etc/dhcpd.conf” file:



The line “ddns-update-style standard;” in the “dhcpd.conf” file specifies the DDNS update style that the DHCP server will use to update the DNS records. In this context, the DHCP server will use the standard and default methods for the DNS records. We used this because the particular method is usually used when the DNS and DHCP servers are on the same machine (virtual machine). The “ddns-rev-domainname “in-addr.arpa.”;” directive indicates that the DHCP server will use the standard "in-addr.arpa." reverse domain name for DDNS updates. The "in-addr.arpa." domain is a special top-level domain used for reverse DNS lookups based on IP addresses. When a DHCP client obtains or renews an IP address lease, the DHCP server will automatically update the reverse DNS record in the "in-addr.arpa." domain to map the client's IP address to its hostname. By including "deny client-updates;", the DHCP server will not allow DHCP clients to perform DDNS updates. This means that clients won't be able to modify their DNS records through the DHCP server, and any updates to DNS records must be managed manually by administrators.

Furthermore, the line "do-forward-updates on;" in the "dhcpd.conf" file for the DHCP server enables the forwarding of DDNS updates to the DNS server. By setting "do-forward-updates on;", the DHCP server is configured to forward the DDNS updates to the DNS server for processing. This is typically used when the DHCP server and DNS server are separate entities, and the DHCP server needs to update DNS records hosted on the DNS server. By default, DHCP servers may optimize DDNS updates to reduce the number of updates sent to the DNS server. This optimization may consolidate multiple updates into a single update, which can be more efficient in certain situations. However, setting "update-optimization off;" disables this optimization, meaning that the DHCP server will send individual updates for each DHCP client. This can be useful in scenarios where you want to ensure that each update is processed independently by the DNS server. Then, we added the forward and reverse zones in the file:



Next, we restarted both the “named” and “dhcpd” services to apply the changes:

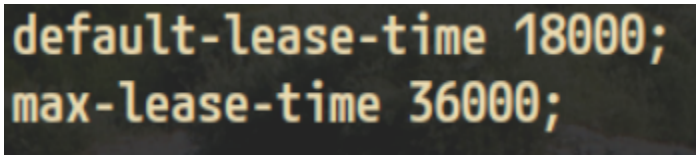
```
[root@wowiee system]# systemctl restart named
[root@wowiee system]# systemctl restart dhcpd4
```

After these configurations, we proved that the clients have connected within the specified range of addresses by checking the leases file, at “/var/lib/dhcp/dhcpd.leases”:

```
[root@wowiee etc]# cat /var/lib/dhcp/dhcpd.leases
# The format of this file is documented in the dhcpd.leases(5) manual page.
# This lease file was written by isc-dhcp-4.4.3-P1

# authoring-byte-order entry is generated, DO NOT DELETE
authoring-byte-order little-endian;

lease 192.168.5.100 {
  starts 6 2023/07/15 15:15:49;
  ends 6 2023/07/15 20:15:49;
  tstp 6 2023/07/15 20:15:49;
  cltt 6 2023/07/15 15:15:49;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet 1c:b7:96:70:23:79;
  uid "\001\034\267\226p#y";
  set vendor-class-identifier = "HUAWEI:android:INE";
  client-hostname "HUAWEI_nova_3i-344c2cd2ab";
}
server-duid "\000\001\000\001,Em\310ZV\205H\034u";
```

4.2.4 REQUIREMENT 4 - LEASE TIME

It was required to ensure that the DHCP server specifies a lower limit of 5 hours (18000 seconds), and an upper limit of 10 hours (36000 seconds) in the “dhcpd.conf” configuration file. This means defining the “default-lease-time” and the “max-lease-time”:



The “default-lease-time” directive describes a lower limit of 5 hours, and the “max-lease-time” directive describes an upper limit of 10 hours. Moreover, the lower limit defines the amount of time that a DHCP client will hold an IP address before it must renew the lease. The upper limit defines the maximum amount of time that a DHCP client can hold an IP address. If a client tries to request a lease longer than the specified maximum, it will be capped at the maximum lease time.

Below is the complete “dhcpd.conf” file to show the formatting and placement of directives:

```
default-lease-time 18000;
max-lease-time 36000;

# dynamic dns for WAP
ddns-update-style standard;
ddns-rev-domainname "in-addr.arpa.";
deny client-updates;
do-forward-updates on;
update-optimization off;
update-conflict-detection off;

# NSA assignment
subnet 192.168.5.0 netmask 255.255.255.0 {
        range 192.168.5.100 192.168.5.200;
        option routers 192.168.5.1;
        option domain-name-servers 192.168.5.20;
        option domain-name "group5.xyz";
}

host webserver {
        fixed-address 192.168.5.10;
}
host dnsserver {
        fixed-address 192.168.5.20;
}

zone group5.xyz. {
        primary 192.168.5.20;
}

zone 5.168.192.in-addr.arpa. {
        primary 192.168.5.20;
}
```

4.2.5 REQUIREMENT 5 - AUTOMATIC START ON REBOOT

Lastly, enabling the “dhcpd4” service ensures that the DHCP server starts automatically upon reboot, allowing it to be available and operational whenever the system starts up. The command makes the “dhcpd4” service become active during system startup unless manually stopped or disabled. Below are the commands to enable the “dhcpd4” service to start automatically upon reboot and to check its status:

```
[root@wowiee wowiee]# systemctl enable dhcpd4
```

```
[root@wowiee wowiee]# systemctl status dhcpd4
● dhcpd4.service - IPv4 DHCP server
     Loaded: loaded (/usr/lib/systemd/system/dhcpd4.service; enabled; preset: disabled)
     Active: activating (start) since Mon 2023-07-10 15:27:50 +08; 2ms ago
Cntrl PID: 31522 ((dhcpd))
      Tasks: 1 (limit: 8198)
     Memory: 0B
        CPU: 0
     CGroup: /system.slice/dhcpd4.service
             └─31522 "(dhcpd)"
```

In order to apply the configurations we made to the DHCP server, we ran “systemctl restart dhcpd4” to restart the service. By running this command, we effectively applied the changes made to its configuration files, like “dhcpd.conf”:

```
[root@wowiee wowiee]# systemctl restart dhcpd4
```

4.3 CONFIGURATION CHALLENGES (DHCP SERVER)

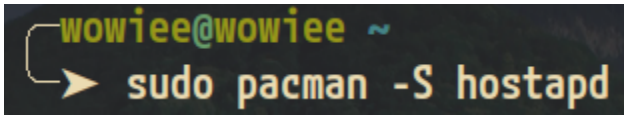
During the DHCP server configuration for the assignment, there were some configuration challenges that we encountered. One of the main challenges was ensuring the “dhcpd.conf” file was correctly formatted with no syntax errors. To address this, we carefully reviewed the configuration file, double-checked the syntax, and used syntax validation tools or commands like “dhcpd -t” to check for errors. Another challenge was to properly configure the DNS server IP address in the DHCP configuration file. It was crucial to ensure that the correct IP address is specified in the “option domain-name-servers” directive to enable DHCP clients to communicate with the DNS server. These challenges can be addressed by careful configuration review, thorough testing, and following the assignment requirements and guidelines.

5.0 WIRELESS ACCESS POINT

5.1 OVERVIEW (WIRELESS ACCESS POINT)

A wireless access point (WAP) is a device that enables wireless devices, such as smartphones, laptops, and tablets, to connect to a wired network. It serves as a bridge between the wireless and wired networks, allowing wireless clients to access resources and services available on the local area network (LAN). The wireless access point (WAP) configuration is a critical component of the assignment. The goal was to set up a Linux-based wireless access point (WAP). Key requirements included defining the Service Set Identifier (SSID) as "NetworkAdmin_5", setting the operating wireless channel to 11, and enabling “WPA2” authentication with the key "sunway123". Additionally, a shell script was created to monitor the real-time in or out throughput of the WAP. To ensure seamless operation, the wireless access point should start automatically after the system reboot. By accomplishing these objectives, a secure and efficient wireless network was established, facilitating connectivity for wireless clients and promoting effective network management.

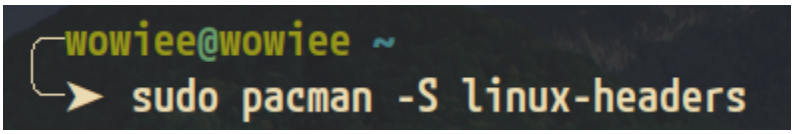
For our wireless access point, we had to install “hostapd”. “Hostapd” is a user space daemon software program that allows a computer to act as a wireless access point (AP) or hotspot. Below is the command to install the “hostapd” software onto our system:



The command “sudo pacman -S hostapd” requests “pacman” to download and install the “hostapd” package on our Arch Linux system. This makes the “hostapd” software available for configuring and managing a wireless access point (AP) on our computer. We chose to use an external network adapter for our Wireless Access Point. The model of our network adapter is the “TP-Link TL WN722N” adapter. Firstly, to make sure the adapter works properly, we need to install the drivers for this adapter. This adapter uses the RTL8188 chipset. The driver was freely available on the “aircrack-ng” GitHub repository (https://github.com/aircrack-ng/rtl8188eus), however as of 14/7/2023 it has been recently deprecated for the most recent kernel. We used a fork of this repository, at https://github.com/gglluukk/rtl8188eus. Below is the complete process of installing the relevant drivers:

Installing Drivers:

Installing kernel headers (important):



Installing make dependencies:

```
wowiee@wowiee ~
➤ sudo pacman -S base-devel libelf bc
warning: base-devel-1-1 is up to date -- reinstalling
warning: libelf-0.189-1 is up to date -- reinstalling
warning: bc-1.07.1-4 is up to date -- reinstalling
```

```
wowiee@wowiee ~
➤ sudo pacman -S dkms
warning: dkms-3.0.11-1 is up to date -- reinstalling
```

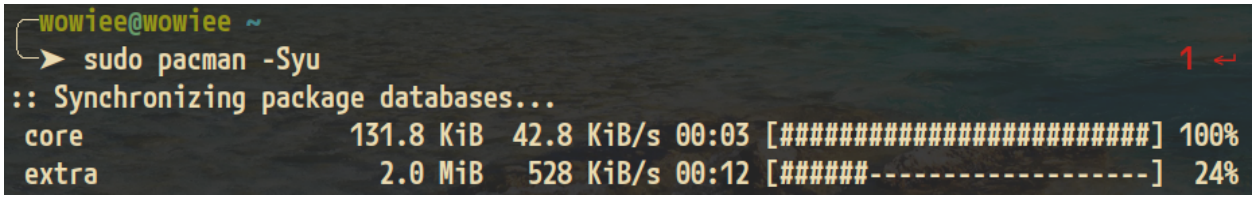
Cloning the repository:

```
wowiee@wowiee ~
➤ git clone https://github.com/gglluukk/rtl8188eus
```

Specifying which drivers to install by modifying the configuration file:

```
wowiee@wowiee ~
➤ cd rtl8188eus
wowiee@wowiee ~/rtl8188eus ‹v5.3.9*›
➤ echo "blacklist r8188eu" > "/etc/modprobe.d/realtek.conf"
```

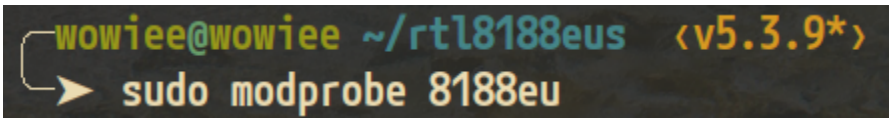
After these steps were completed, we had to reboot the system and perform a system upgrade:



After the system update was done, we had to build the driver with ‘sudo make’:

Since no errors are encountered, we proceeded to use ‘sudo make install’ to install the drivers:

Once this is done, we had to replace the old kernel driver (module) with the new driver we just installed:



5.2 IMPLEMENTATION DETAILS (WIRELESS ACCESS POINT)

5.2.1 REQUIREMENT 1 - SERVICE SET IDENTIFIER

5.2.2 REQUIREMENT 2 - OPERATING WIRELESS CHANNEL

5.2.3 REQUIREMENT 3 - WPA2 AUTHENTICATION

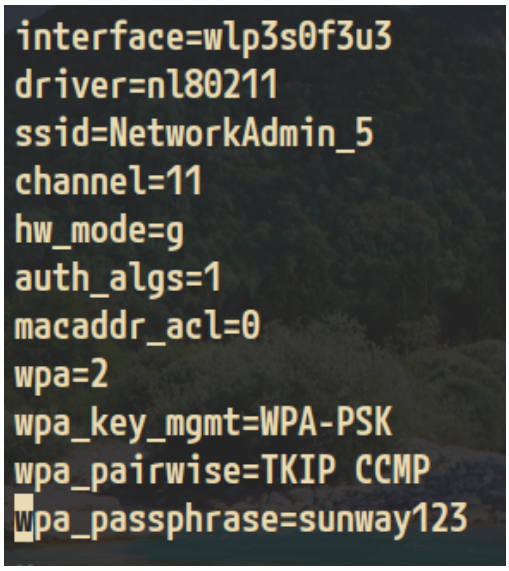
After confirming that the wireless adapter works, we started configuring “hostapd”. The “hostapd” configuration file is located in “/etc/hostapd/hostapd.conf” directory:

```
[root@wowiee hostapd]# vim hostapd.conf
```

Firstly, we had to identify the name of our wireless adapter to be able to use it. We can do this by executing “ifconfig”:

```
wlp3s0f3u3: flags=4099<UP,BROADCAST,MULTICAST>  mtu 2312
        ether fe:cb:ea:b7:3c:39  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

After finding out that the adapter is named “wlp3s0f3u3”, we started configuring “hostapd” in the “hostapd.conf” configuration file:

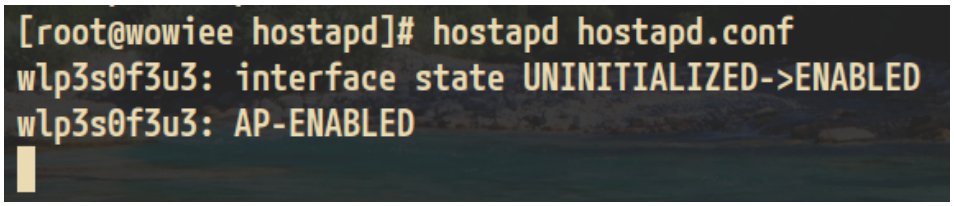


“Interface” refers to the name of the network interface, which we previously found. The “nl80211” driver supports most network interface cards (NIC), so we chose that. The SSID, passphrase, and channel has been set according to the requirements; though the passphrase had to be extended from “sunway” to “sunway123” in order to fulfill WPA2 standards. The “hw_mode” parameter defines the operating mode of the interface; “g” defines that both 2.4 and 5 GHz are allowed even though this adapter only supports 2.4 GHz. There is flexibility, so you are able to change it according to your respective adapter.

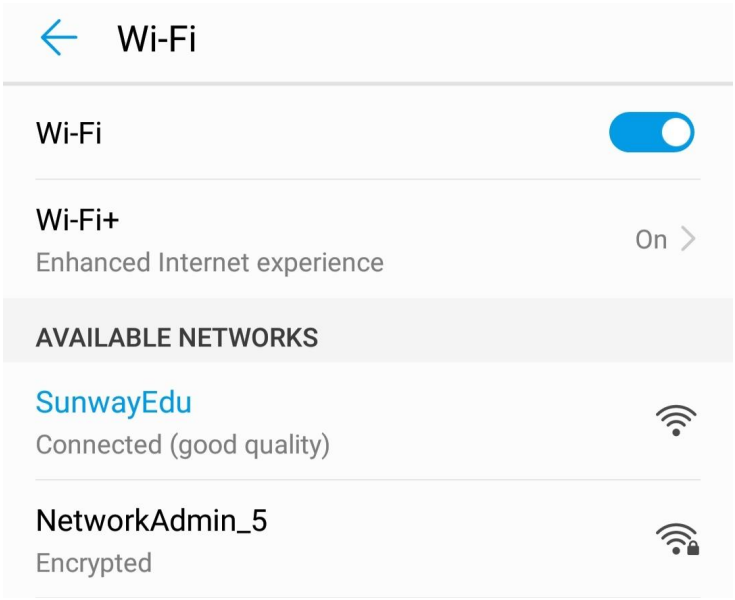
The “auth_algs” parameter defines WAP authentication; “1” signifies “open auth” authentication. “Macaddr_acl” is a filter used to prevent MAC addresses from spoofing. We switched it to “0” (off) since it is not required. “Wpa_key_mgmt” defines the key management algorithms the client is allowed to use. “Wpa_pairwise” controls the type of data encryption. Once the configuration was done, we had to start the “hostapd” service:

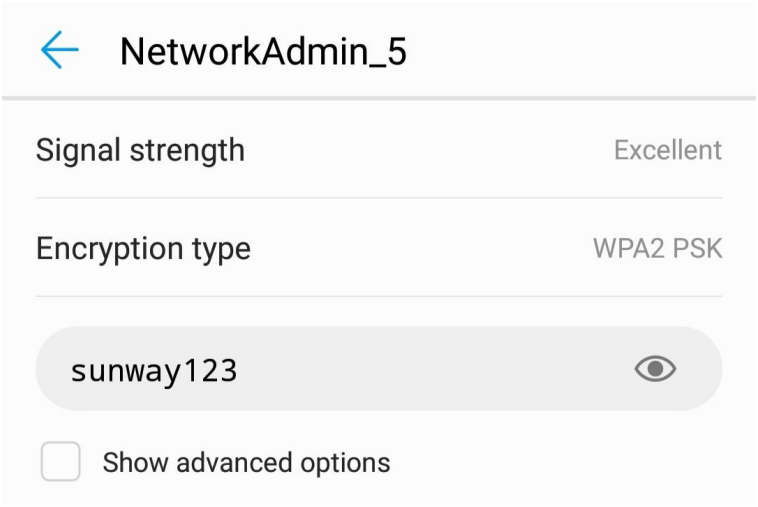
First, we killed all existing “hostapd” processes that could be running in the background:

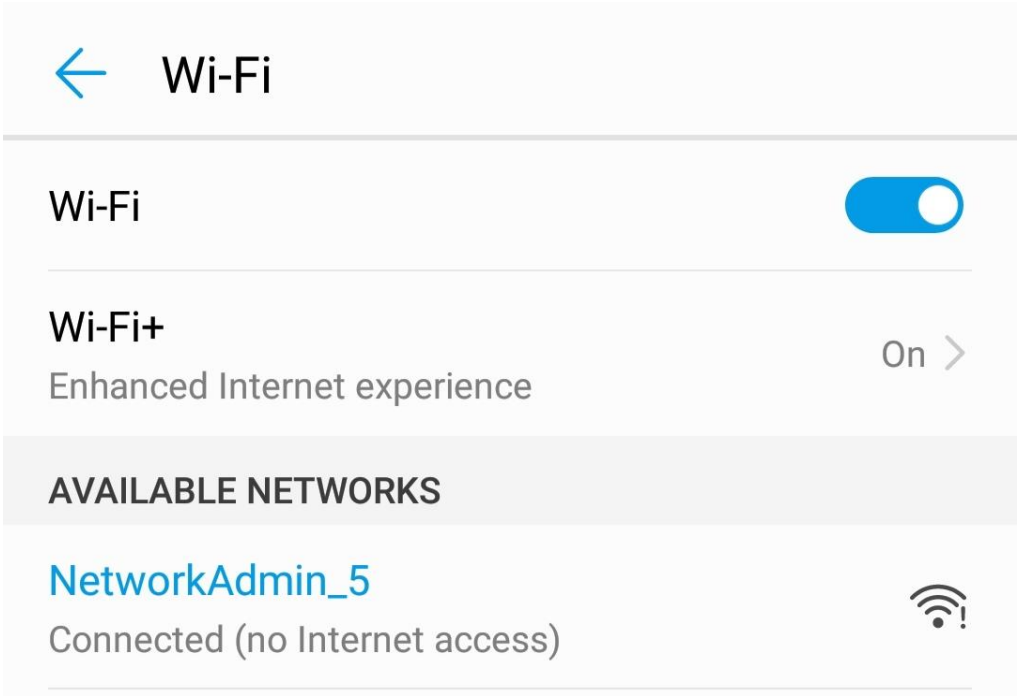
Then, we started “hostapd” and specified the configuration file we modified:



To check if the access point was up, we tried connecting to it with another device:

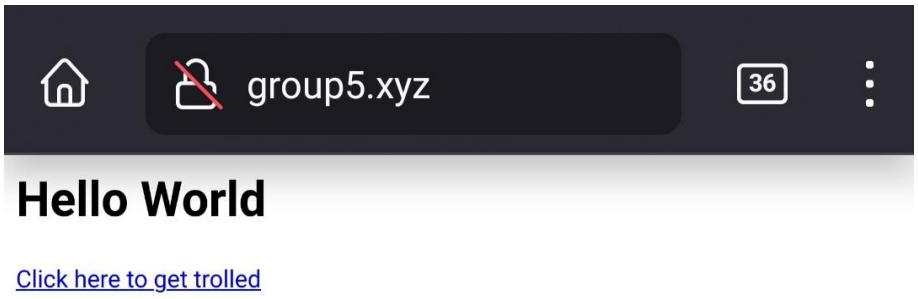






As shown above, the device was able to connect successfully.

Next, we checked if it was able to access our web server by domain name:



Once we tried connecting with our phone, we saw that “hostapd” was able to detect the connection attempt:

```
wlp3s0f3u3: AP-STA-CONNECTED 1c:b7:96:70:23:79
wlp3s0f3u3: STA 1c:b7:96:70:23:79 RADIUS: starting accounting session 69D517E2873F5
C54
wlp3s0f3u3: STA 1c:b7:96:70:23:79 WPA: pairwise key handshake completed (RSN)
wlp3s0f3u3: EAPOL-4WAY-HS-COMPLETED 1c:b7:96:70:23:79
```

5.2.4 REQUIREMENT 4 - AP REAL-TIME THROUGHPUT SHELL SCRIPT

We were required to create a shell script to monitor in real-time the in or out throughput (kB/s) of the AP. Below is the shell script that we created to meet this requirement:

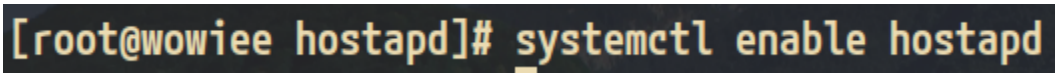
```
if ! command -v ifstat &> /dev/null; then
    echo "ifstat is not installed. Please install it first."
    exit 1
fi

# Our virtual interface that we created
interface="wlp3s0f3u3"

monitor_throughput() {
    while true; do
        clear
        ifstat -i "$interface" 1 1
    done
}

monitor_throughput
```

The interface we monitored is the virtual interface that we created for the virtual adapter that we used, not the main interface of our system. This script uses the “ifstat” command to monitor the

virtual network interface (“wlp3s0f3u3” in this case) and displays the real-time incoming and outgoing throughput in kilobits per second (kb/s). The script continuously updates the output on the terminal to show the current throughput values. It helped us to monitor the wireless AP's network activity and identify any fluctuations or anomalies in the data transfer.

5.2.5 REQUIREMENT 5 - AUTOMATIC START ON REBOOT

We ensured our wireless access point starts up at boot by enabling the “hostapd” service. We executed the command “systemctl enable hostapd” to do this. Enabling "hostapd" with “systemctl” ensures that the service remains enabled across system reboots. This ensured that our wireless access point (WAP) functionality is always available without the need to manually start the service after each reboot:



5.3 CONFIGURATION CHALLENGES (WIRELESS ACCESS POINT)

During the configuration of the wireless access point (WAP) for the assignment, we encountered several challenges. One of the main challenges was setting up and connecting the external wireless adapter to our system. This required ensuring compatibility, installing appropriate drivers, and configuring the adapter to function as a wireless access point. However, we tried to use our internal adapter at first, and from this, more problems arose. Timeless hours were spent on this, only to figure out that we had to use an external adapter since there was a

problem with the host's hardware. Additionally, setting the operating wireless channel to 11 posed a challenge as we needed to verify channel availability and ensure it did not interfere with other nearby networks. This is because we initially tried experimenting with channel 12.

6.0 ADDITIONAL FEATURES

6.1 OVERVIEW (ADDITIONAL FEATURES)

In addition to the core requirements of the assignment, we implemented some additional features to enhance the functionality and management of the system. Firstly, we configured the “ufw” firewall to provide an added layer of security for the Apache web server and DNS server. This firewall helps protect against unauthorized access and potential threats, ensuring the integrity and confidentiality of the network services. We also developed several shell scripts to monitor various aspects of the Apache web server. These scripts allow us to track the other server’s uptimes, disk usage, and memory usage of the servers, providing valuable insights into its performance and resource utilization. Additionally, we implemented a script to monitor and analyze the error logs generated by the Apache server, facilitating effective fault management and troubleshooting.

6.2 IMPLEMENTATION DETAILS (ADDITIONAL FEATURES)

6.2.1 REQUIREMENT 1 - ADDITIONAL MONITORING

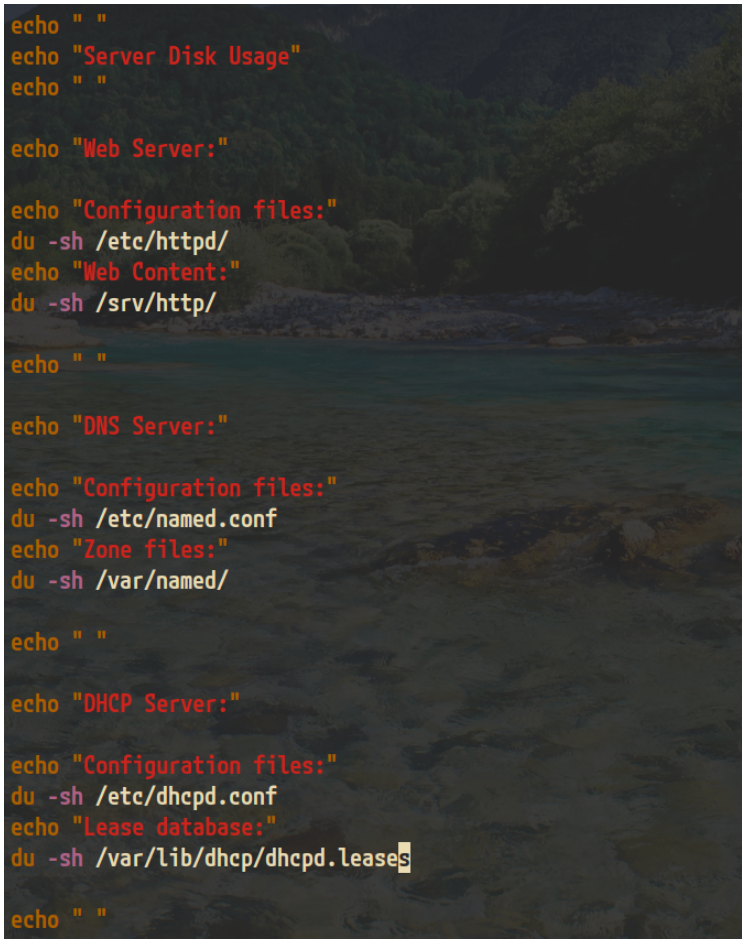
We decided to implement additional monitoring shell scripts in order to assist with fault management, security management, and performance management. Firstly, we created a memory usage monitoring shell script to keep track of the amount of memory (kB) used by the three servers:

```
echo " "
echo "Server Memory Monitoring"
echo " "

monitor_memory() {
        process_name=$1
        pid=$(pgrep $process_name)

        if [[ -n $pid ]]; then
                mem_usage=$(pmap -x $pid | tail -n 1 | awk '{print $3}')
                echo "$process_name service memory usage: $mem_usage kB"
        else
                echo "$process_name service is not running."
        fi
}


echo "Web Server:"
monitor_memory "apache2"
echo " "
echo "DNS Server"
monitor_memory "named"
echo " "
echo "DHCP server"
monitor_memory "dhcpd"
```

The provided shell script allows you to monitor the memory usage of the three servers. The script uses a “monitor_memory()” function that generalizes the commands to check the memory of a server. If the server is active, it retrieves the process ID (PID) of the respective process using “pgrep”. Next, the script utilizes the “pmap” command with the “-x” flag to obtain detailed information about the memory usage of the service. It extracts the memory usage value from the last line of the output and stores it in the “mem_usage” variable. Finally, the script displays the memory usage of the three servers in kilobytes (kB). This information provides insights into the amount of memory consumed by each server, aiding in resource monitoring and capacity planning. Monitoring memory usage is a valuable additional feature for performance management, fault management, and optimization. It allows us to identify any potential memory leaks, inefficient resource utilization, or excessive memory consumption by the three servers. This shell script would help to optimize performance and ensure that memory resources are

efficiently allocated. Of course, you could check the memory used by running a command like “htop”, but it is split and spread out; posing a challenge on cumulative calculation.

Tracking disk usage is just as important for performance management and resource management. Below is the shell script that we created in order to extract relevant data for disk usage from the default directory of the three servers:



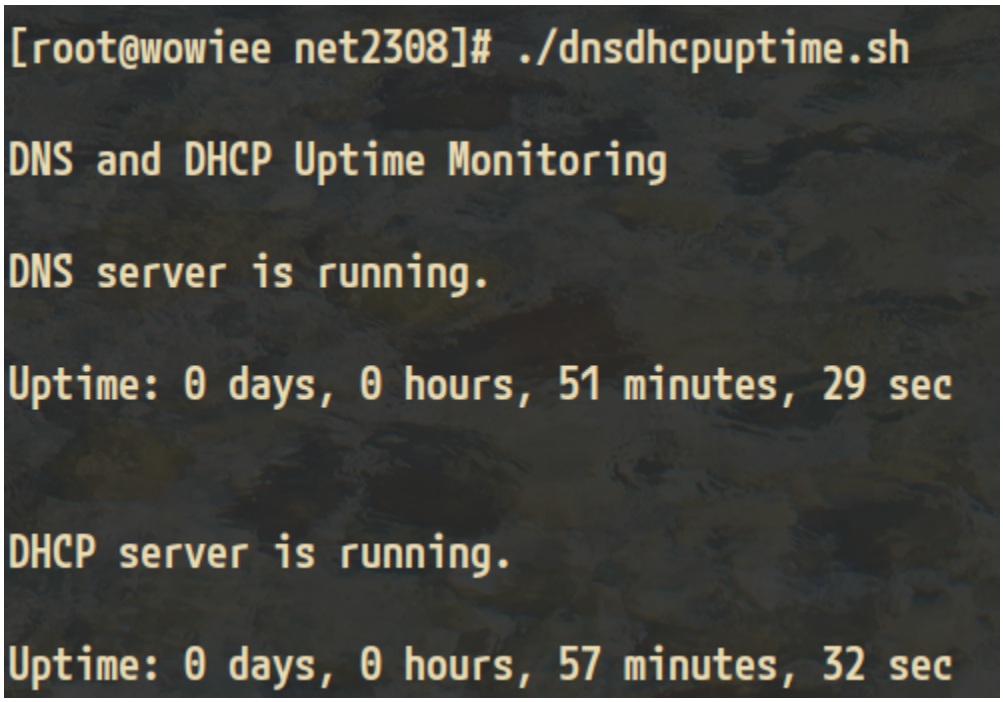
```
echo " "
echo "Server Disk Usage"
echo " "

echo "Web Server:"

echo "Configuration files:"
du -sh /etc/httpd/
echo "Web Content:"
du -sh /srv/http/

echo " "

echo "DNS Server:"

echo "Configuration files:"
du -sh /etc/named.conf
echo "Zone files:"
du -sh /var/named/

echo " "

echo "DHCP Server:"

echo "Configuration files:"
du -sh /etc/dhcpd.conf
echo "Lease database:"
du -sh /var/lib/dhcp/dhcpd.leases

echo " "
```

We used the command “du -sh" to determine the total disk space used. “du” stands for disk usage and “-s” is to summarize and “h” stands for human readable which makes the results more readable formats like KB, MB or GB. So, the command will get disk usage data from the web server, DNS server, and DHCP server. For specification purposes, we separated the configuration files from the web content, zone files, and lease database so that disk usage can be monitored more effectively and appropriately. Monitoring disk usage allows us to keep track of available disk space on our servers. We are able to proactively identify and address situations where disk space is running low. This prevents circumstances where issues are caused by insufficient disk spaces, such as service interruptions or data corruption. Monitoring disk space usage can also help with fault management, as we are able to identify potential bottlenecks, or disk space hogs that can affect the performance of a server.

Since creating a shell script for monitoring the uptime of the web server was a requirement, we figured that creating a shell script for monitoring the uptime of the DNS and DHCP server would also serve as a benefit in performance management:

```
if systemctl is-active --quiet named; then
        echo " "
        echo "DNS server is running."
        echo " "

        start_time=$(systemctl show -p ExecMainStartTimestamp --value named)

        current_time=$(date +%s)

        uptime=$((current_time - $(date -d "$start_time" +%s)))

        days=$((uptime / 86400))
        seconds=$((uptime % 60))
        minutes=$((uptime % 3600 / 60))
        hours=$((uptime % 86400 / 3600))


        echo "Uptime: $days days, $hours hours, $minutes minutes, $seconds sec"
        echo " "
else
        echo " "
        echo "DNS server is not running"
        echo " "
fi
```

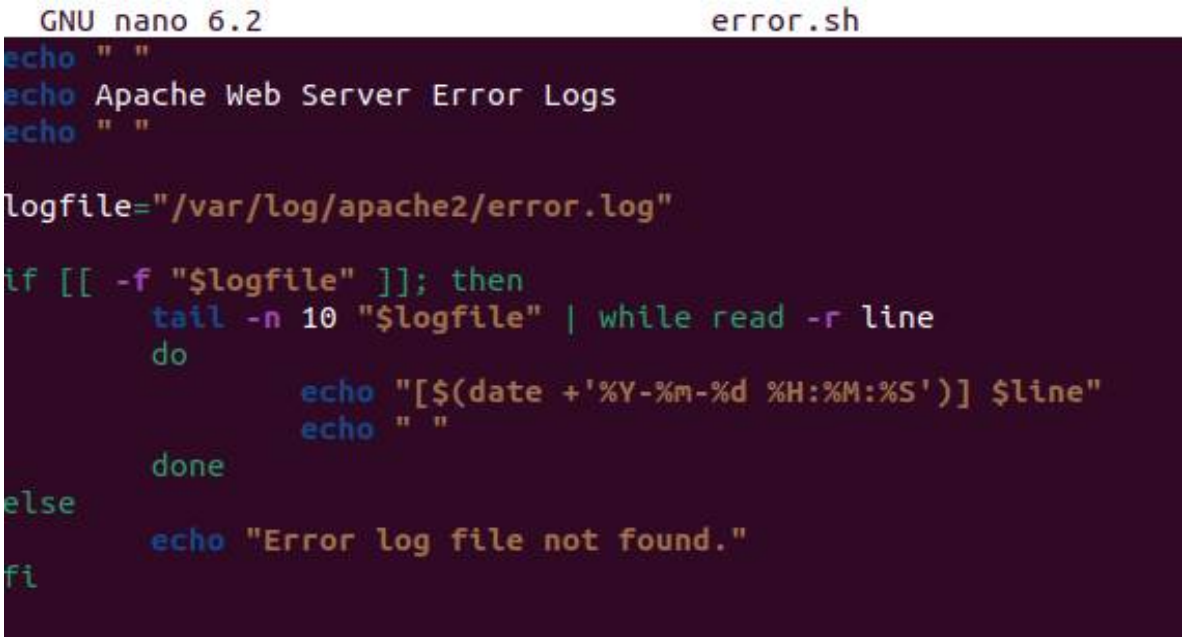
```
if systemctl is-active --quiet dhcpd; then
        echo " "
        echo "DHCP server is running."
        echo " "

        start_time=$(systemctl show -p ExecMainStartTimestamp --value dhcpd)

        current_time=$(date +%s)

        uptime=$((current_time - $(date -d "$start_time" +%s)))

        days=$((uptime / 86400))
        seconds=$((uptime % 60))
        minutes=$((uptime % 3600 / 60))
        hours=$((uptime % 86400 / 3600))


        echo "Uptime: $days days, $hours hours, $minutes minutes, $seconds sec"
        echo " "
else
        echo " "
        echo "DHCP server is not running"
        echo " "
fi
```



The created shell script allows us to check the uptime of the servers. It begins by checking the status of the services by using the "systemctl" command, which determines if the servers are currently running or not. If the service is active, the script proceeds to retrieve the start time of the service using "systemctl show". To calculate the uptime, the script captures the current time using the date command and converts it to the Unix timestamp format. The difference between the current time and the start time is then computed, representing the uptime of the service in seconds. The script further breaks down the uptime into days, hours, minutes, and seconds for a more human-readable format. This information offers valuable insights into how long the servers have been continuously operational. By utilizing this script, we can efficiently track the uptime of the servers to ensure service availability. Monitoring server uptime is also a valuable additional feature for server management as it allows us to assess the stability and reliability of

the servers through intervals and extended periods of time. Not to mention, we are also able to analyze the duration and frequency of server downtime by noticing how long servers usually last. By identifying these patterns, we can investigate causes and implement measures to minimize downtime.

Extracting data from and monitoring error logs are crucial components of fault management and security management. We created a shell script that formats and extracts data from the web server's error logs. The default error logs are not as user-friendly as they are presented in a format that is hard to read. Below is the shell script for the web server's error logs:



```
ashwin@ashwin-VirtualBox:~$ sudo bash error.sh

Apache Web Server Error Logs

[2023-07-08 13:50:47] [Fri Jun 16 11:44:04.021713 2023] [mpm_event:notice] [pid
 765:tid 139749472270208] AH00489: Apache/2.4.52 (Ubuntu) configured -- resumin
g normal operations

[2023-07-08 13:50:47] [Fri Jun 16 11:44:04.022678 2023] [core:notice] [pid 765:
tid 139749472270208] AH00094: Command line: '/usr/sbin/apache2'

[2023-07-08 13:50:47] [Wed Jun 21 11:55:29.102825 2023] [mpm_event:notice] [pid
 744:tid 139873965340544] AH00489: Apache/2.4.52 (Ubuntu) configured -- resumin
g normal operations

[2023-07-08 13:50:47] [Wed Jun 21 11:55:29.107264 2023] [core:notice] [pid 744:
tid 139873965340544] AH00094: Command line: '/usr/sbin/apache2'

[2023-07-08 13:50:47] [Sat Jul 01 11:54:35.603651 2023] [mpm_event:notice] [pid
 770:tid 139763756300160] AH00489: Apache/2.4.52 (Ubuntu) configured -- resumin
g normal operations

[2023-07-08 13:50:47] [Sat Jul 01 11:54:35.651306 2023] [core:notice] [pid 770:
tid 139763756300160] AH00094: Command line: '/usr/sbin/apache2'

[2023-07-08 13:50:47] [Sat Jul 01 15:03:20.128002 2023] [mpm_event:notice] [pid
 761:tid 139889795356544] AH00489: Apache/2.4.52 (Ubuntu) configured -- resumin
g normal operations
```

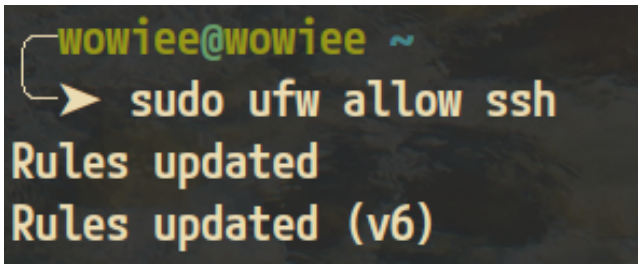
The command ‘logfile="/var/log/apache2/error.log"’ assigns the file path ‘/var/log/apache2/error.log’ to the variable ‘logfile’. By storing the file path, it is easier to reuse and preserve the path without having to repeat again in the code. The command ‘-f “$logfile”’ is to check if the file is a regular file (‘-f”) and if the variable ‘$logfile’ exists. If the file exists, it will proceed with the next command which is the “tail -n 10 “$logfile | while read -r line”. The command “tail -n 10 “$logfile" obtains the last 10 lines from the log file and the command “while read -r line” starts a while loop that reads each line from the tail command output and writes it to the variable line. Next, the command "[$(date + '%Y-%m-%d %H:%M:%S')] $line” where it is a string that consists of two parts. The command generates a timestamp in the

“YYYY-MM-DD” format, which is year, month, and date. “HH:MM:SS” is in hours, minutes, and seconds format. The command “Error log file not found” will be executed when the condition of the if statement is false and there is no default error log file. Monitoring error logs is an effective additional feature for web server management and troubleshooting. By monitoring the error logs, we can quickly identify any errors or issues occurring on the web server. This allows for timely investigation and resolution of problems, minimizing potential downtime and improving the overall user experience. Error logs can contain information about potential security incidents, such as unauthorized access attempts or suspicious activities. Therefore, monitoring error logs will help us detect and respond to security threats promptly, enhancing the security posture of the web server.

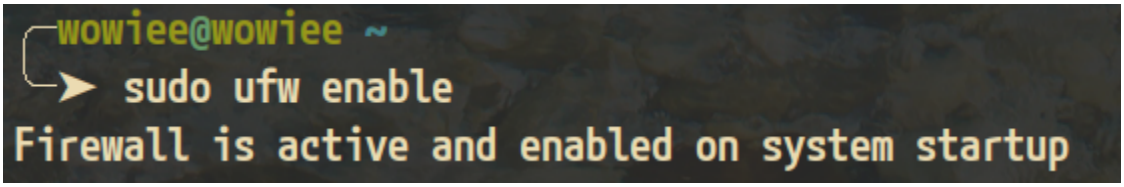
6.2.2 REQUIREMENT 2 - FIREWALL

For part of the additional features, we have installed a firewall (specifically UFW). A firewall is a security component of a network that acts as a barrier between internal and external networks. Any form of data transfer is monitored and controlled through the firewall, deciding whether or not the data is safe or a potential threat to the network. The reason we used UFW (Uncomplicated Firewall) has a few reasons. UFW is the simplest and easiest to configure firewall in any server and its straightforward configuration is beneficial for novice users. Another reason is that UFW is integrated with Arch Linux, therefore it is possible to configure a firewall without needing to source from external services. The firewall that we implemented is for the web server and DNS server since it correlates with connections from port 80 and port 53. This is needed because Apache listens in using port 80 and DNS listens in using port 53.

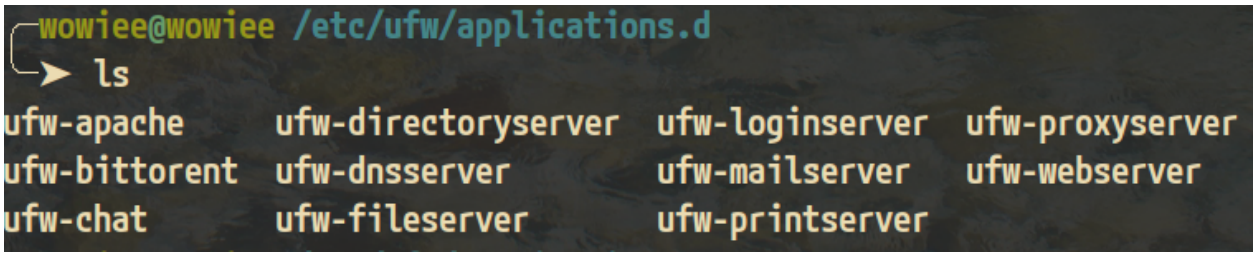
To set up the firewall, first, we have to allow our server to accept requests from port 22, because port 22 is where UFW listens. Port 22 can also be translated to “ssh”:

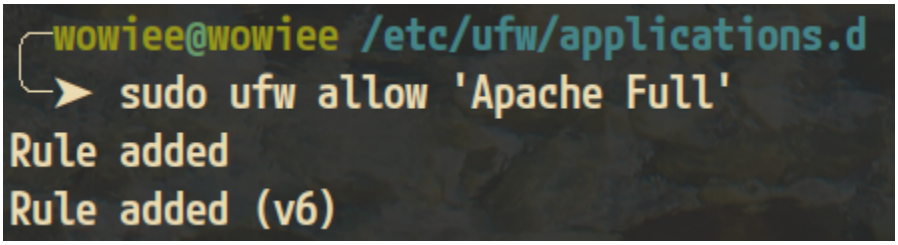


By running this command, we configured the firewall to allow incoming SSH connections, which enables us to remotely access the server using SSH. It is important to allow SSH connections only from trusted sources and ensure proper security measures are in place, such as using strong passwords or key-based authentication, to protect the server from unauthorized access. Therefore, we did prior research on UFW before going on with implementing it. After that, we used the command “sudo ufw enable” to allow the firewall to automatically start on system startup. It will also start its course of action of enforcing the defined firewall rules on the system based on the configured rules that we are about to add:

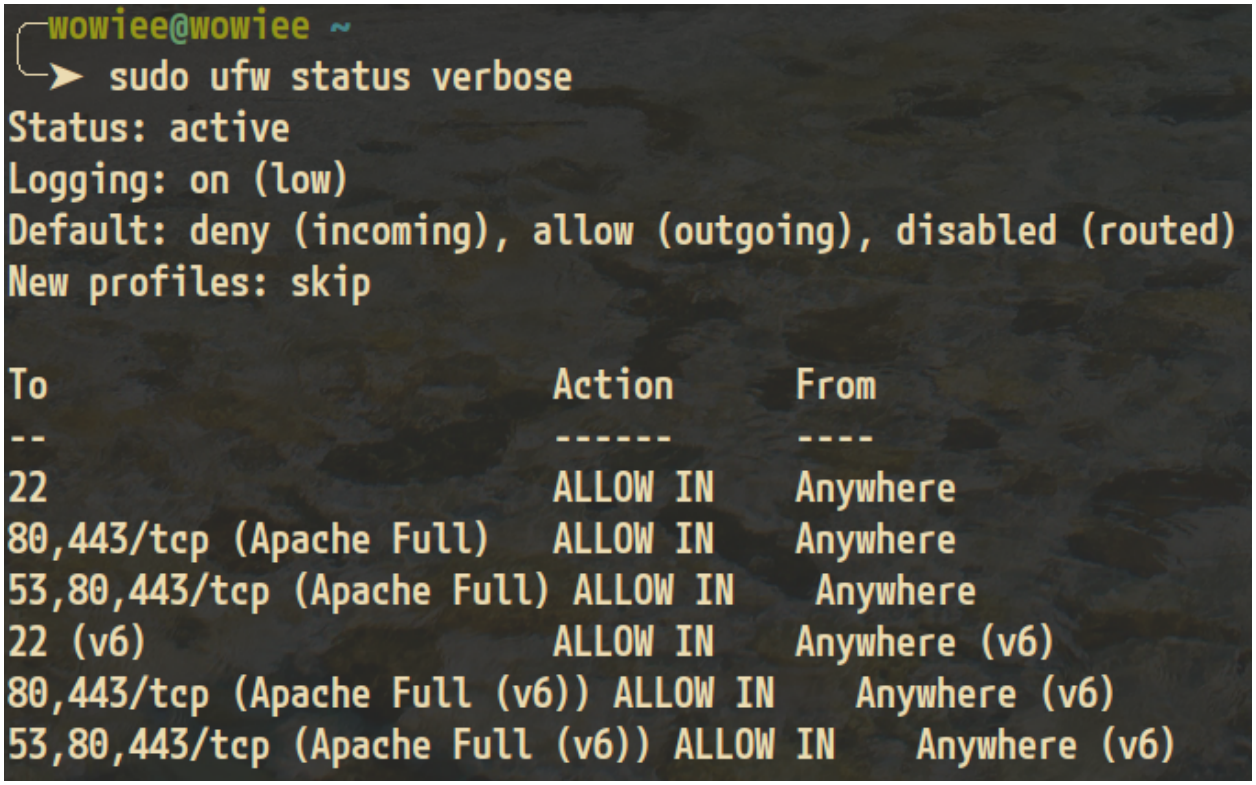


Then, we had to check the application list for the available application profiles in the Uncomplicated Firewall (UFW) by doing “sudo ufw app list”. We also ran the command “sudo ufw allow ‘Apache Full’” to specifically allow incoming traffic on port 80 and port 443:



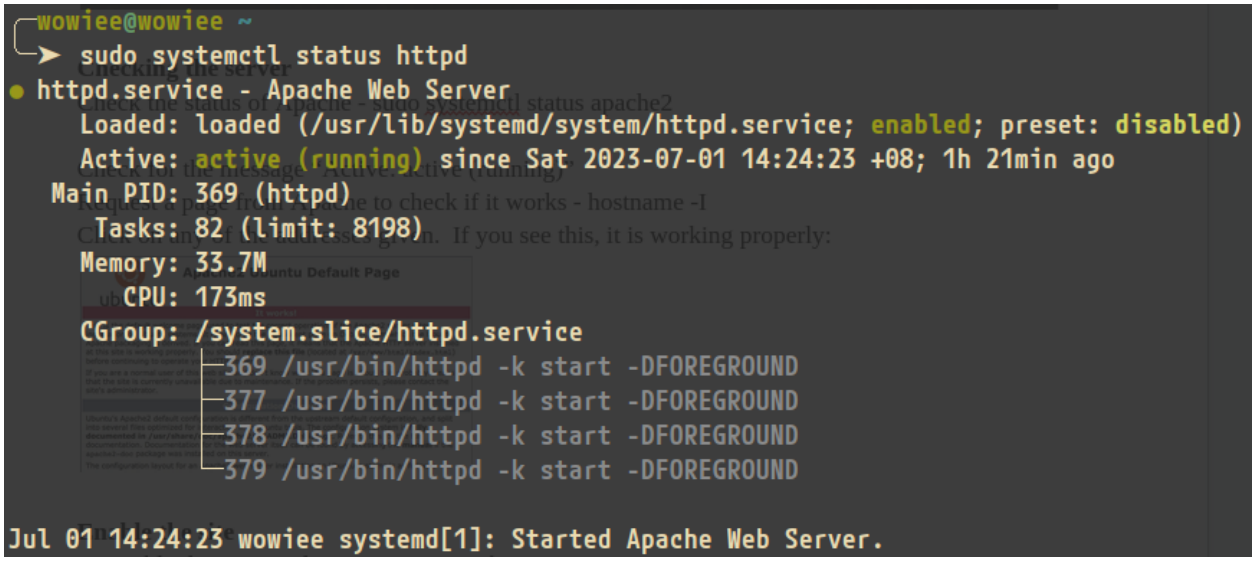


The message "Rule added (v6)" and “Rule added” indicates that a new firewall rule has been added to the IPv6 (Internet Protocol version 6) and IPv4 (Internet Protocol version 4) firewall configuration. We did the same for the DNS server by doing “sudo ufw allow dns”. In order to properly verify the configuration rules we added, we have to execute the command “sudo ufw status verbose”:



The command "sudo ufw status verbose" is used to display the current status and detailed information about the firewall rules managed by the UFW. In the context of firewall configurations, "deny (incoming)" refers to the rule that blocks or denies incoming network connections or traffic. This means that any attempts to establish connections or send data to the specified port or IP address will be rejected by the firewall. On the other hand, "allow

(outgoing)" refers to the rule that permits outgoing network connections or traffic. This allows the system to initiate connections or send data to other devices or servers on the network or the internet. Lastly, "disabled (routed)" indicates that the firewall rules for a specific interface or network route are disabled. In this state, the firewall does not actively filter or control the network traffic for that particular interface or route. From the screenshot, we can clearly see that the configuration rules for the firewall have been added to ports 53 and 80. We have allowed the action to “ALLOW IN” since we do need clients to connect to the web server and access the web pages according to the requirements. "ALLOW IN" refers to the rule or policy that allows incoming network connections or traffic through the firewall. It means that the firewall is configured to permit incoming connections from external sources to reach the specified port or service. In order to ensure that the configurations made did not cause the server to go down, we checked the status of the servers. For example, we ran “sudo systemctl status httpd” to check the status of the web server after implementing the firewall:



6.3 CONFIGURATION CHALLENGES (ADDITIONAL FEATURES)

During the implementation of the additional features, we encountered several configuration challenges. One of the main challenges was creating shell scripts to monitor the uptime, memory usage, disk usage, and error logs of the servers. It required understanding the appropriate commands and parsing the output to extract the relevant information. We also had to ensure that the scripts ran efficiently and provided accurate data. Another challenge was implementing the firewall using UFW for the DNS and Apache web servers. We had to carefully configure the firewall rules to allow necessary traffic while blocking unauthorized access. It involved understanding the required ports and services for each server and applying the appropriate rules.

7.0 LESSONS LEARNED

7.1 GROUP REFLECTION

In this section, each student from our group reflects on the valuable knowledge and skills gained throughout the project, as well as how they plan to apply these skills in the future. By doing this group reflection, we can share our thoughts, observations, and reflections on the project as a whole. It entails what we learned, both in terms of technical knowledge and personal skills development. It allows for a comprehensive understanding of the strengths, weaknesses, successes, and areas for improvement.

7.1.1 ASHWIN VIGNESWARAN (REFLECTION)

Throughout this assignment, I have gained valuable knowledge and skills in configuring and managing various components of a network infrastructure. I learned how to set up a web server, configure DNS and DHCP servers, and create a wireless access point. This experience has deepened my understanding of configuration and server administration. In terms of skills, I have honed my ability to work with command-line interfaces, troubleshoot server configurations, and create monitoring shell scripts. Moving forward, I plan to apply these skills in my future endeavors. While I have made significant progress, there are areas where I can improve. Enhancing my knowledge of advanced DNS and DHCP configurations will be a notable factor for future growth. Additionally, I aim to further develop my problem-solving and analytical skills to effectively address challenges in network administration.

7.1.2 NIDAL BENCHEIKH LEHOCINE (REFLECTION)

While working on this assignment I was able to further my understanding of how servers work and I gained a small vision of how the websites I use every day function. By working on the shell script monitoring, I furthered my understanding of commands and I learned new commands that I have not learned before in class. Working on the WAP also allowed me to understand how they are set up and I realized that it is way more in-depth than I thought it was. By working in a group, I was able to further better my communication skills as well as collaboration skills. In the future, I hope to apply these skills in the working field as well as during my Capstone project as I believe they will come in handy for the project I have in mind. As for the time being, I will need to work more on expanding my knowledge of DNS and DHCP. I believe improving these skills will be a great advantage to my network administration skills.

7.1.3 ELENA KHOO SZE KAY (REFLECTION)

I have learned a lot about maintaining and optimizing network structure and also gained useful insights throughout this assignment. I have learned how the error logs for the Apache server work, to work with web servers, and to be able to develop scripts that provide real-time insights into the server performance, like bandwidth, CPU usage, uptime, and many more. My team also included additional features such as adding a firewall and showing real-time insights into the other server uptimes, disk usage, etc. I now have a better understanding of how network architectures work and interact with various components because of this experience. I will implement these skills in the future. On the other hand, since it is crucial, I will get better at

configuring the DHCP and DNS servers. To continue advancing network quality, I look forward to strengthening my knowledge in these areas and discovering new avenues.

7.1.4 ESTHER LESLIE BALA (REFLECTION)

Many things were learned in this assignment. Firstly: if your system is made up of many parts which have to work together, documentation is important. Secondly: there is no second point. Only the first point applies; documentation is important. I cannot stress how lost we would be if we didn't self-document our steps throughout this assignment. And even ignoring our own documentation, it was only through other people's documentation that we were able to complete this assignment in the first place. But bad documentation has the opposite effect. Bad (or a lack of) documentation leads to following forum posts, scraping through archived emails, and watching Youtube videos in a language I previously didn't know existed. So, I tried my best to improve our documentation and include every step and struggle. Another challenge was the wireless access point; I had never touched my kernel or drivers before out of fear of bricking my laptop, but I had to in order to install the drivers for my adapter. So that was terrifying. In terms of improvement, I want to learn more about server hardware. A laptop and a USB network adapter are obviously not ideal, so I would like to get my hands on something better if possible.

7.1.5 CHEN ERN KHAI (REFLECTION)

This assignment has given me valuable insights into creating a web server and the many functionalities of using Linux as an OS. There were many trials and errors that we had to research. Although it was hard to figure out in the beginning, things slowly started to piece

together on their own. Working on the firewall has given me a detailed understanding of how it works and the commands used to configure it. While there was a portion of the assignment where I had to be away on medical leave, I really thank my group members for covering my back. This team effort has taught me that it is important to treasure your partners and also the time shared to complete this assignment together. Overall, this positive experience of creating the servers has made me more excited and passionate about this industry and I cannot wait for more to come in the future.