CONVENTIONAL-HARDWARE GNU-LICENSED (CHGL) INFRASTRUCTURE
A WISE ARCHITECTURE

For a Small Business to an Enterprise

Table of Contents

INTRODUCTION

This project aimed to achieve the goal of providing a cost-effective and easy to deploy architecture design, fulfilling business infrastructure requirements for multi-site and branch headquarter needs. The project considered all positive network performance factors such as redundancy, security, and availability in its design and deployment.

This experiment and implementation reached the goal of facilitating companies to interconnect their branches by using conventional devices. This solution fits all ranges of need from an enterprise-class of a network, to a small office with one or two employees.

Additionally, using free software would not prevent businesses from a necessary upgrade to gain higher throughput. Using conventional hardware under the GNU licensed software made an outstanding opportunity to have a budget network in any size.

Universal design method and implementation facilitated all sorts of teleworking needs, which could be a home, workplace, or a remote foreign location connection.

Different hardware has been studied and tested for this practice, the greatest of which was chosen to implement. In this design and implementation practice, considering network security was the main factor. In all steps, the architect tried to follow the best practices and reliable network design principals.

METHODOLOGY

Finding the best architecture

Different methodologies define the way various network components communicate with each other. Choosing different topology can change the availability, security, cost, and complexity of the system. Before I choose a method for interconnection, I must have a brief overview of communication methods.

Mesh, one of the primary methods of communication, aims to deliver the message in one hop from one end to the other. Therefore, the full mesh needs all sites in the network fully interconnected.

As fig.1 shows, every site has a link to the other peers.

In the partial mesh, only some of the sites have redundant paths to the other peer.

As fig.2 shows, R2 does not have a direct link to the R3 and R4.

Alternatively, Star topology uses hub and spoke approach. In essence, the hub acts as an access point or gateway for all sites, which act as spokes. This point-to-point (single-hop site to center) communication in star topology makes the implementation and the management more straightforward.

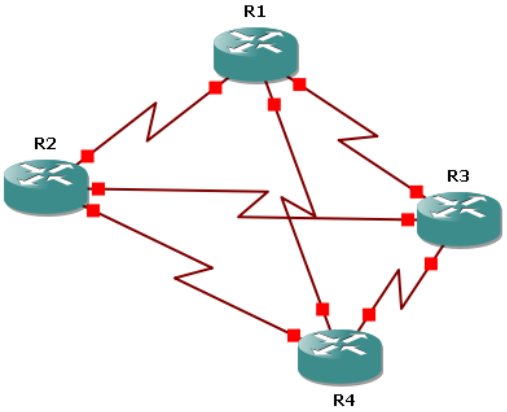


Figure 1

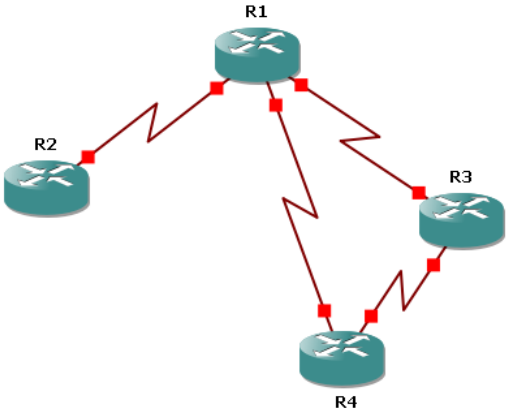


Figure 2

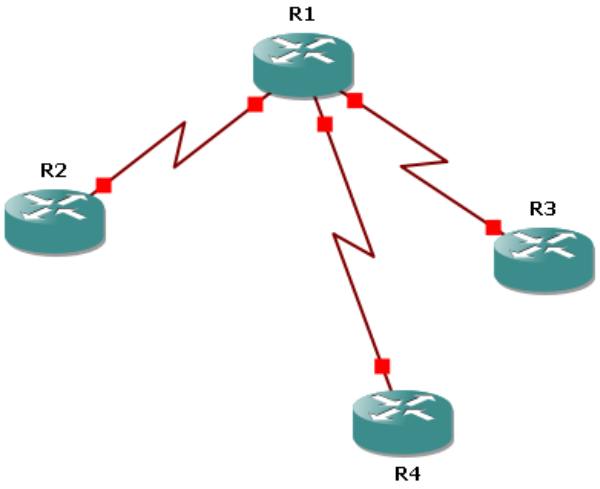


Figure 3

Simplicity in management and implementation could be the main reason for choosing the hub and spoke approach for this project. Therefore, I picked hub and spoke as the target architecture and started searching for a server to act as a hub and experimented with using conventional Wi-Fi Routers as spokes.

Finding Hardware

As mentioned in the proposal, a big portion of the project is selecting the hardware needed to implement the architecture. The central portion of that is the hub, which can run on an old server running hypervisor, making it capable of handling other tasks at the same time for the business. Also, a cheap, available and reliable conventional Wi-Fi Router as a spoke was required, which would be capable of replacing the firmware and converting it to a generic router.

For the hub hardware, HP ProLiant DL380 G6[1] was chosen, which cost less than $300.Vmware vSphere ESXi 5.5[2] hypervisor was then installed (using free license offers by VMware for hypervisor only).

Using hypervisor will make the implementation more comfortable, and ensure OS maintenance tasks will perform smoothly. It also adds a form of redundancy and reliability in the future if I add more hosts into the cluster.

Also, to have better reliability, the decision was made to use old server hardware with redundant power supply and ECC memory, and not a regular PC.

It is necessary to mention that spoke hardware is selected between many different products. Firmware availability, software compatibility, and hardware stability were the main factors in the choice. Wireless Router TP-Link Archer C7 ver: 2.0[3] was the choice here, which cost less than $40.

[1] *https://h20195.www2.hpe.com/v2/getdocument.aspx?docname=c04282582*

[2] *https://www.vmware.com/ca/products/esxi-and-esx.html*

[3] *https://www.tp-link.com/ca/home-networking/wifi-router/archer-c7*

Finding software

For the OS on the Hub, I used CentOS[4] (Community Enterprise Operating System), which is one of the most reliable GNU licensed[5] OS.

For the VPN server, many open-source options were available. Openswan[6] is an IPsec with lots of overhead, and Tcpcrypt[7] has its own unique features, but was not a good fit for this project. SoftEther[8] as a functional clone of the OpenVPN[9] could be a choice, but after some tests and study, OpenConnect[10] (OC) selected as the best option to provide VPN functionality on the hub.

An OpenConnect client was initially created as a VPN client to cover numerous security vulnerabilities of Cisco's AnyConnect VPN[11] client. Then the OC client was developed to be a complete replacement of CiscoAnyConnect and addressed all of its deficiencies, and is now acting as Cisco alternative for any Linux user.

OpenConnectServer[12] (OCserv), as the best pick for the server-side VPN software, was designed to provide SSL VPN to make spoke to hub interconnection; it also has complete compatibility with the OC client software.

For the replacement OS (firmware) on the spoke's hardware, research was conducted on all available open-source firmware software, including DD-WRT[13], AdvancedTomato[14] , and OpenWRT[15]. OpenWRT was picked as the most stable, full-featured and highly customizable embedded firmware for the TP-Link gateway. This firmware was also cross-checked with hardware compatibility, using hardware compatibility matrix with the route's architecture and the processor capabilities.

---

[4] *https://www.centos.org/about/*

[5] *http://mirror.centos.org/centos/7/os/x86_64/EULA*

[6] *https://www.openswan.org*

[7] *http://www.tcpcrypt.org*

[8] *https://www.softether.org*

[9] *https://openvpn.net*

[10] *https://www.infradead.org/openconnect/*

[11] *https://www.cisco.com/c/en/us/support/security/anyconnect-secure-mobility-client/tsd-products-support-series-home.html*

[12] *https://ocserv.gitlab.io/www/index.html*

[13] *https://dd-wrt.com*

[14] *https://advancedtomato.com*

[15] *https://openwrt.org/about*

Stability testing

Testing the stability for the hub was done using CentOS installed as a VM machine, while running multiple VPNs against it to see how the OCserv reacts.

Testing the stability of the client device was a more challenging process. Factory firmware of the wireless router was removed and replaced with the OpenWRT firmware, which was then tested by using the gateway as a home internet router for multiple days. By completing this phase, it was made sure that the router OS matches to the hardware, and is stable enough to use as the client spoke router.

The next phase of testing was about evaluating the system service and throughput using iPerf[16] software to generate traffic from each end on TCP and UDP and observing the desired output, and running this test from one end of the network to the other end, to mimic the site to site communication.

Moreover, the final step for testing included providing this service to multiple individuals and collecting their experience and feedback with using this system.

DISCUSSION

Execution

ESXi and CentOS

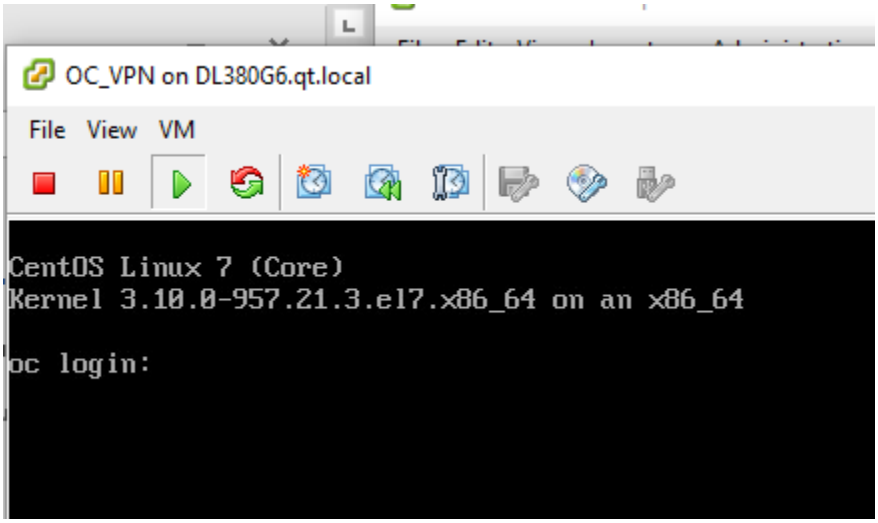
The VPN hub execution and testing were done by installing and deploying ESXi and CentOS and using the package manager to install the OCserv.

ESXi was set up using VMware guides (VMware, n.d.) on the HP server after downloading a copy of the hypervisor ISO file from the VMware website and a free copy of hypervisor license key. ESXi was then installed from CD-Rom after setting the Boot priority settings on the server BIOS setup settings, and booting from CD-Rom. This Internal SD-Card was selected in the setup steps to be the target of the ESXi installation storage. The SD-Card was then formatted and ESXi was installed completely. Then in the next reboot, the management IP address was changed to make the host reachable from the lab network. vSphere Client software was used, and installed on Windows to connect to the host, and add the server hard disk drive as Datastore. The next step was to upload the CentOS Minimal image ISO file downloaded from the CentOS website or the mirrors to the datastore. A new virtual machine was added to the host and CentOS v7 64-bit was selected as OS type. 20 Gigabyte of thin provisioned disk was then dedicated to the VM. All

[16] *https://iperf.fr*

default VM settings were accepted and the virtual network card assigned to the “VM Network” vSwitch. CentOS ISO image was selected as CD/DVD image from the datastore and VM installation started by powering on the VM.

The CentOS was installed by following the guide (centos, n.d.), after which a local user account was added to the CentOS and the root password was set. The VM was then rebooted, and this time installed CentOS bootup and prompted for credentials.



After a successful login to the server by running these commands, I could make sure SSH service is available and running and read the IP address assigned to the OC server using DHCP and start using PuTTY[17] to SSH to the server.

```
[root@oc ~]# service sshd status
Redirecting to /bin/systemctl status sshd.service
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2019-11-19 14:01:25 MST; 2 months 25 days ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 1244 (sshd)
   CGroup: /system.slice/sshd.service
           └─1244 /usr/sbin/sshd -D
```

```
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1
000
    link/ether 00:0c:29:ac:b0:26 brd ff:ff:ff:ff:ff:ff
    inet 192.168.254.169/24 brd 192.168.254.255 scope global dynamic ens160
       valid_lft 259192sec preferred_lft 259192sec
```

[17] *https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html*

As is visible, the IP address showed in the output of the “ip a” command, and “ens160” is the name of the interface I have on the VM machine.

This VM will be my hub server, and servers should always use static IP addresses. The server IP address changed through the SSH session and network service restarted for the change to apply. Elevation to the root was required.

```
[ocadmin@oc ~]$ sudo su
[sudo] password for ocadmin:
[root@oc ocadmin]#
```

```
[root@oc ~]# nmcli -p dev
=====================
  Status of devices
=====================
DEVICE  TYPE      STATE      CONNECTION
------------------------------------------------------------------------
ens160  ethernet  connected  Wired connection 1
lo      loopback  unmanaged  --
[root@oc ~]#
```

```
[root@oc ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group
default qlen 1000
    link/ether 00:50:56:b9:6f:a9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.254.9/32 brd 192.168.254.9 scope global noprefixroute ens160
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:feb9:6fa9/64 scope link
       valid_lft forever preferred_lft forever
```

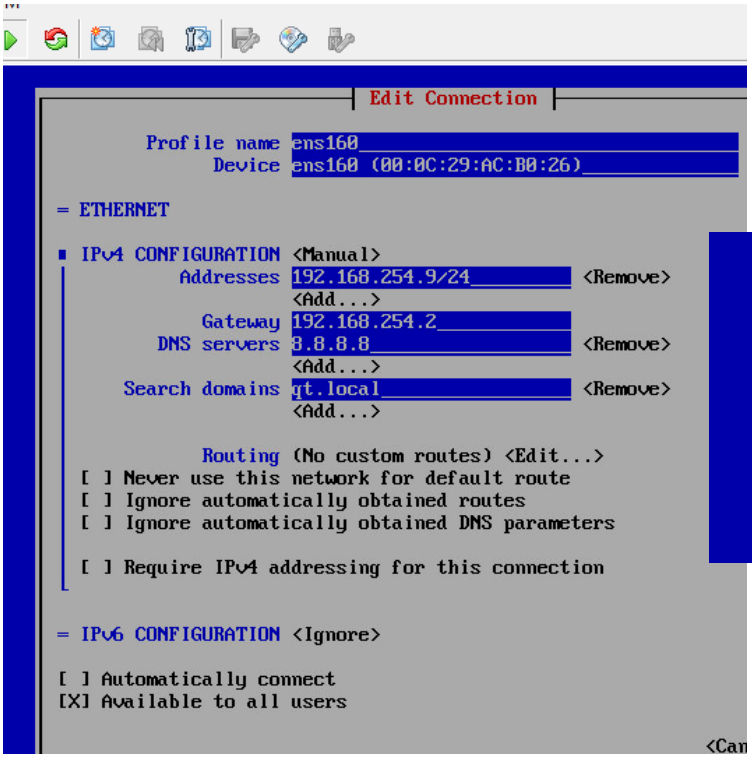
```
[root@oc ~]# vi /etc/sysconfig/network-scripts/ifcfg-Wired_connection_1
HWADDR=00:50:56:B9:6F:A9
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPADDR=192.168.254.9
PREFIX=32
GATEWAY=192.168.254.2
DNS1=8.8.8.8
DEFROUTE=yes
PEERROUTES=no
```

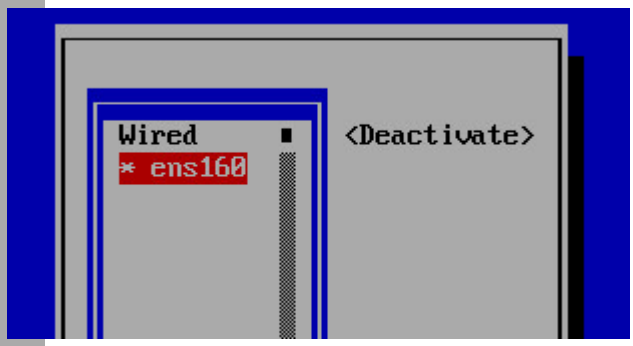
```
IPV4_FAILURE_FATAL=no
IPV6INIT=no
NAME="Wired connection 1"
UUID=5786644a-4c6e-3691-aa43-99d04d253944
ONBOOT=yes
AUTOCONNECT_PRIORITY=-999
ZONE=trusted
~
```

```
# systemctl restart network
```

The change was also verified by using CentOS NetworkManager “nmtui-edit” and “nmtui-connect” guide (Red_Hat_Enterprise_Linux, n.d.) to activate the settings.

```
# nmtui-edit ens160
# nmtui-connect
```





The next step was to make sure that the server has a route for its gateway and the DNS server IP address is correctly assigned.

```
[root@oc ~]# ip r
default via 192.168.254.2 dev ens160 proto static metric 100
```

```
192.168.254.2 dev ens160 proto static scope link metric 100
192.168.254.9 dev ens160 proto kernel scope link src 192.168.254.9 metric 100
[root@oc ~]#
[root@oc ~]# cat /etc/resolv.conf
# Generated by NetworkManager
search qt.local
nameserver 8.8.8.8
[root@oc ~]#
```

Now the CentOS was ready for OCserv package installation. However, before that, I needed to make sure that I had good Internet access. Also, to make sure the server received all necessary update packages, the following commands were used.

```
[root@oc ~]# ping www.centos.org
PING www.centos.org (81.171.33.202) 56(84) bytes of data.
64 bytes from ip-81.171.33.202.centos.org (81.171.33.202): icmp_seq=1 ttl=46
time=158 ms
64 bytes from ip-81.171.33.202.centos.org (81.171.33.202): icmp_seq=2 ttl=46
time=158 ms

[root@oc ~]# yum update
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos.les.net
 * epel: mirrors.sonic.net
 * extras: muug.ca
 * updates: centos.ca-west.mirror.fullhost.io
Resolving Dependencies
--> Running transaction check
```

OpenConnect server

After installing all updates and software patches, it was time to install OCServ, using the manual (gitlab, n.d.).

The EPEL repository was then added (fedoraproject, n.d.).

```
[root@oc ~]# yum install epel-release
[root@oc ~]# yum update
```

It is good to know that CentOS users, along with other Linux distributions like Scientific Linux, Oracle Linux, and Red Hat Enterprise Linux (RHEL), can easily access and install packages for commonly used software using Extra Packages for Enterprise Linux (EPEL) repository. This repository was created to provide greater ease of access to software on Enterprise Linux compatible distributions. (Singer, n.d.)

OCserv package and GNU TLS utilities were installed.

```
[root@oc ~]# yum install ocserv gnutls-utils
```

CA and Server Cert.

Because OCserv is an SSL server and server authentication is based on certificates, I needed to add a certificate directory and create a new certificate authority (CA) template. (Guoan, n.d.)

```
[root@oc ~]# mkdir /etc/ocserv/cert
[root@oc ~]# cd /etc/ocserv/cert/
[root@oc cert]# echo "cn = "OC VPN"
> organization = "QT"
> serial = 1
> expiration_days = -1
> ca
> signing_key
> cert_signing_key
> crl_signing_key" > certauth.tmplate
[root@oc cert]# ls
certauth.tmplate
```

Then the Certificate Authority's certificate needed to be created using a generated private key, and the template was made.

```
[root@oc cert]# certtool --generate-privkey --outfile certauth-key.pem
Generating a 2048 bit RSA private key...
[root@oc cert]# certtool --generate-self-signed --load-privkey certauth-
key.pem --template certauth.tmplate --outfile certauth-cert.pem
Generating a self signed certificate...
X.509 Certificate Information:
        Version: 3
        Serial Number (hex): 01
        Validity:
                Not Before: Sun Feb 16 04:25:30 UTC 2020
                Not After: Fri Dec 31 23:59:59 UTC 9999
        Subject: CN=OC VPN,O=QT
        Subject Public Key Algorithm: RSA
.
[Intentionally omitted]
.
Signing certificate...
[root@oc cert]#
```

Now I had the CA certificate, as well as a pem file which included my public key and private key for the CA.

```
[root@oc cert]# ls
certauth-cert.pem  certauth-key.pem  certauth.tmplate
[root@oc cert]#
```

The next step was to create the server certificate template and publish a server certificate using the server cert template, server's private key, and the CA private key and CA certificate.

```
[root@oc cert]# echo " cn = "qtocserv"
> dns_name = "www.qt.local"
> organization = "QT"
> expiration_days = -1
> signing_key
> encryption_key
> tls_www_server" > server.tmplate
[root@oc cert]#
[root@oc cert]# certtool --generate-privkey --outfile qtocserv-key.pem
Generating a 2048 bit RSA private key...
[root@oc cert]# ls
certauth-cert.pem  certauth-key.pem  certauth.tmplate  qtocserv-key.pem
server.tmplate
[root@oc cert]#
[root@oc cert]# certtool --generate-certificate --load-privkey qtocserv-
key.pem --load-ca-certificate certauth-cert.pem  --load-ca-privkey certauth-
key.pem --template server.tmplate --outfile qtocserv-cert.pem
Generating a signed certificate...
X.509 Certificate Information:
        Version: 3
        Serial Number (hex): 5e48c9800a632fa71c609c95
        Validity:
                Not Before: Sun Feb 16 04:48:00 UTC 2020
                Not After: Fri Dec 31 23:59:59 UTC 9999
        Subject: CN=qtocserv,O=QT
        Subject Public Key Algorithm: RSA
.
[Intentionally omitted]
.
Signing certificate...
[root@oc cert]#
```

The outputs were then moved to the "ssl" directory.

```
[root@oc cert]# ls
certauth-cert.pem  certauth.tmplate   qtocserv-key.pem
certauth-key.pem   qtocserv-cert.pem  server.tmplate
[root@oc cert]# mv certauth-cert.pem qtocserv-key.pem qtocserv-cert.pem
/etc/ocserv/ssl/
[root@oc cert]#
```

Moreover, like all Linux services, OCserv also has a configuration file that I adjusted, in order to force it to use the OCserv password file for the spoke's gateway authentication.

The config file /etc/ocserv/ocserv.conf changed to comment(#) the default method and added the password file for authentication method.

```
[root@oc cert]# cd /etc/ocserv/
[root@oc ocserv]# vi ocserv.conf
```

```
#auth = "pam"
#auth = "pam[gid-min=1000]"
auth = "plain[passwd=./passwd]"
#auth = "certificate"
#auth = "radius[config=/etc/radiusclient/r
```

To use the generated certificate for server authentication, I needed to assign the config file to use those cert files.

```
# a SIGHUP signal to main server will force reload).

server-cert = /etc/ocserv/ssl/qtocserv-cert.pem
server-key = /etc/ocserv/ssl/qtocserv-key.pem

# Diffie-Hellman parameters. Only needed if you requi
# for the DHE ciphersuites (by default this server su
```

```
# is set.
ca-cert = /etc/ocserv/ssl/certauth-cert.pem

# The object identifier that will be used to
```

Furthermore, this hub needed to assign an IP address per spoke after they connected, and to do that, I needed to uncomment a line and assign a correct network into it. With this change, my clients would get an IP address from the range 172.16.17.0/24

```
ipv4-network = 172.16.17.0
ipv4-netmask = 255.255.255.0
```

DNS servers needed to be added to the config too:

```
dns = 1.1.1.1
dns = 8.8.8.8
dns = 4.2.2.2
dns = 4.2.2.3
dns = 4.2.2.4
```

In my implementation, I also had other adjustments to the config file. For security reasons, I did not want to accept two connections using the same credentials; at the same time, it was not ideal to limit the system to accept a limited number of connections.

```
max-clients = 0
max-same-clients = 1
```

In order to have a better and more reliable communication, I needed to detect and eliminate the dead connections. Therefore, I added the keepalive and dead-peer-detection (DPD) to the config.

```
keepalive = 32400
dpd = 90
mobile-dpd = 1800
```

To have a better connection quality and use the most bandwidth, OCserv used UDP instead of TCP after connection establishment. Therefore I asked OCserv to use a timer to fall back in case of an issue.

```
switch-to-tcp-timeout = 25
```

For the MTU on the WAN or Internet connection, there are occasional limitations, and each VPN connection should detect its own maximum MTU for the tunnel interface.

```
try-mtu-discovery = true
```

For the SSL key refreshment, a timer was set.

```
rekey-time = 172800
rekey-method = ssl
```

After the client connects, I expect to see a new virtual network interface added to the server and “vpns” is the prefix for the VPN interfaces.

```
device = vpns
```

The following was added for the default domain search, and to keep all DNS queries and response secure.

```
default-domain = qt.local
tunnel-all-dns = true
```

Moreover, for security reasons, I changed the service port to a different TCP and UDP port. I wanted to have this service run on ports other than the default port of HTTPS, to make confusions on the firewalls on the path.

```
tcp-port = 543
udp-port = 543
```

Because I added a service to this server, I needed to make sure that those customized ports were allowed on the CentOS firewall. (Linode, n.d.) To make the job simple at this step, I just added the server main interface “ens160” to the trusted zone of the firewall, and activated that.

```
[root@oc ~]# firewall-cmd --zone=trusted --add-interface=ens160 -permanent
[root@oc ~]# firewall-cmd --set-default-zone=trusted
[root@oc ~]# firewall-cmd --reload
```

We added the following to check what zone is active and which rules are in effect.

```
[root@oc ~]# firewall-cmd --get-active-zone
trusted
  interfaces: ens160
  sources: 0.0.0.0/0
[root@oc ~]# sudo firewall-cmd --zone=trusted --list-all
trusted (active)
  target: ACCEPT
  icmp-block-inversion: no
  interfaces: ens160
  sources: 0.0.0.0/0
  services:
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
        rule family="ipv4" source address="0.0.0.0/0" accept
[root@oc ~]#
```

As the design displays, all site to hub authenticates itself by the username and password; therefore, adding usernames to the password file as indicated in the configuration file was required.

We then created the password file in the configuration directory and added some usernames and passwords.

```
[root@oc ~]# touch /etc/ocserv/passwd
[root@oc ~]# ocpasswd -c /etc/ocserv/passwd -g default site1
Enter password:
Re-enter password:
```

At this step, OCserv service started and checked to see if the connection can be established or not. A service start or restart was required to load the newly adjusted config file and after that,

Cisco Anyconnect client tried from a Windows machine. Moreover, to make this service available in all system startups, I needed to enable it using “systemctl.”

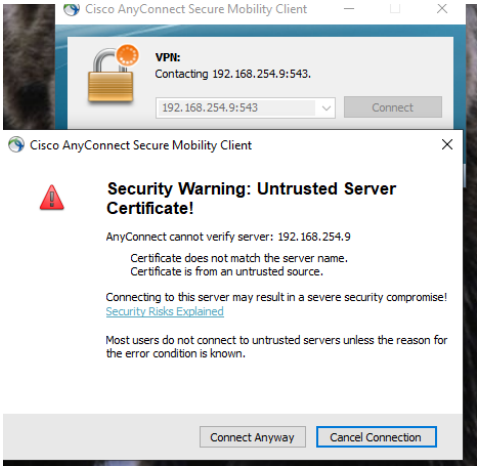
```
[root@oc ~]# systemctl start ocserv
```

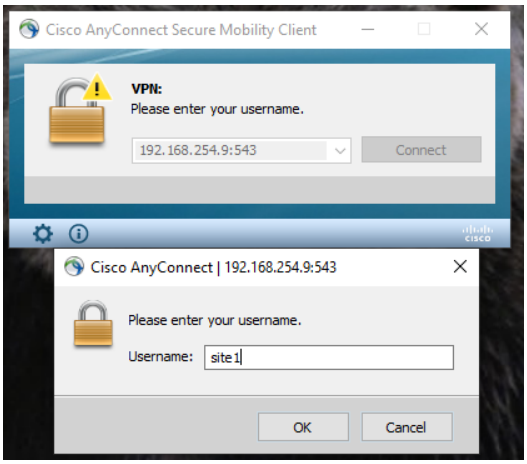
```
[root@oc ~]# systemctl enable ocserv
```

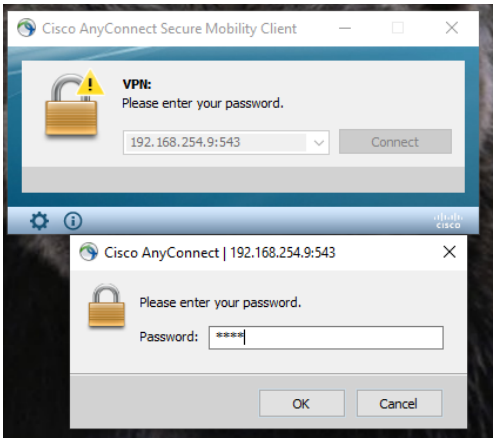
Symlink was then created from from /etc/systemd/system/multi-user.target.wants/ocserv.service to /usr/lib/systemd/system/ocserv.service.

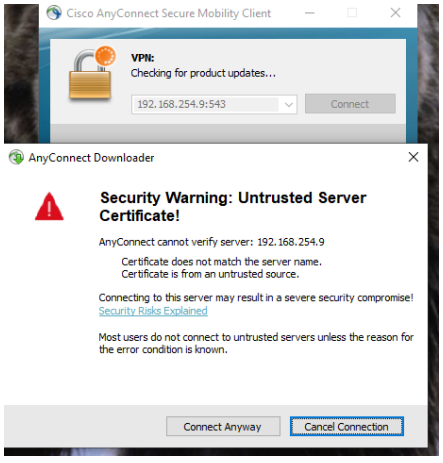
```
[root@oc ~]# systemctl status ocserv
● ocserv.service - OpenConnect SSL VPN server
   Loaded: loaded (/usr/lib/systemd/system/ocserv.service; enabled; vendor
preset: disabled)
   Active: active (running)
     Docs: man:ocserv(8)
 Main PID: 1245 (ocserv-main)
```

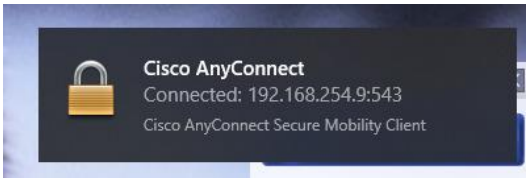
To test the service, I needed to run the Cisco AnyConnect to check against that. I received two warning messages through the process and that is all about the self-signed server certificates, but connection established.

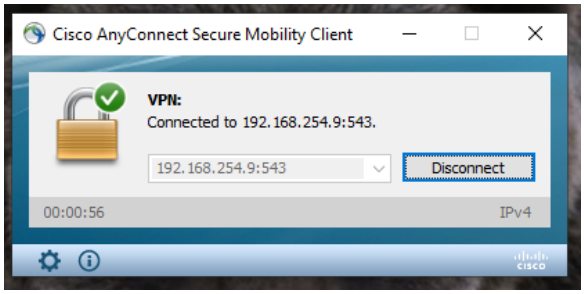




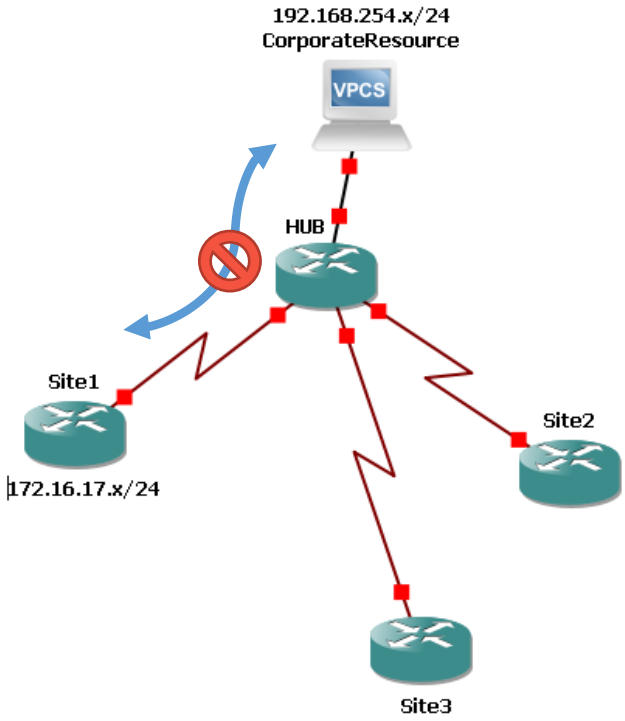








However, at this step, something surprising happened that did not allow Site Router (mimicked by a Windows machine using Cisco Anyconnect) to see anything from the corporate resource. As I know, all operating systems are acting as a router, and the expectation was, if this connection established, the client (172.16.17.x/24) should be able to see the corporate network (192.168.254.x/24). However, because of the CentOS secure configuration about the IPv4 forwarding, OS eliminates all forwarding from one network interface to the other, and this connection as a new network interface needed to be allowed by a forwarding capability of the OS.



For example, “vpns0” as the new dynamic network interface which is created after the first established connection, needed to have forwarding access to the “ens160” which is the corporate resource.

```
3: vpns0: <POINTOPOINT,UP,LOWER_UP> mtu 1434 qdisc pfifo_fast state UNKNOWN
group default qlen 500
    link/none
    inet 172.16.17.1 peer 172.16.17.2/32 scope global vpns10
       valid_lft forever preferred_lft forever
    inet6 fe80::29d3:7294:d947:632f/64 scope link flags 800
       valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group
default qlen 1000
    link/ether 00:50:56:b9:6f:a9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.254.9/32 brd 192.168.254.9 scope global noprefixroute ens160
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:feb9:6fa9/64 scope link
       valid_lft forever preferred_lft forever
Furthermore, ip forwarding needs to be enabled. so open /etc/sysctl.conf:
```

We then tested to query and see if IPv4 forwarding is enabled already on the OS.

```
[root@oc ~]# sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0
```

To fix this issue, I enabled one of the CentOS network settings for IPv4 Forwarding, but to run a quick test, I executed this command to enable IPv4 forwarding. (kvasirsg, n.d.)

```
[root@oc ~]# sysctl -w net.ipv4.ip_forward=1
```

Moreover, to make this change permanent on the OCserv and have the same settings all the time when OS reboots, I needed to add a line to the “sysctl.conf”, and ask for applying.

```
[root@oc ~]# cat /etc/sysctl.conf
[root@oc ~]# echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.conf
[root@oc ~]# sysctl -p /etc/sysctl.conf
net.ipv4.ip_forward = 1
```

Why SSL?

Though earlier in the architecture part of the report, I could have discussed the SSL/TLS VPN functionality and benefits in comparison with IPsec VPN, the intention was to show how easily an SSL/TLS VPN server could be configured and deployed and then dive into the technical details. Details could be discussed, regarding how strong the encryption implemented on each method and how the encryption key will manage. It is also needed to discuss what protocols and ports were used and which network layer was involved in the process, as well as looking at ease of deployment, and higher performance. The protocol was implemented to provide data privacy

and integrity, and would accomplish this by making sure that the connection is secure. This is because each connection uses a key that generates based on the shared secret negotiated on TLS handshake separately for symmetric crypto to encrypt and decrypt data. Then the sender and receiver would negotiate the details of the encryption algorithm keys for data transmission. (Transport_Layer_Security, n.d.)

It is good to know that both parties can authenticate using public-key or at minimum, as noticed in OCserv config, the server-side authenticated by the client. Message integrity was provided, which leads to data loss and alternation prevention.

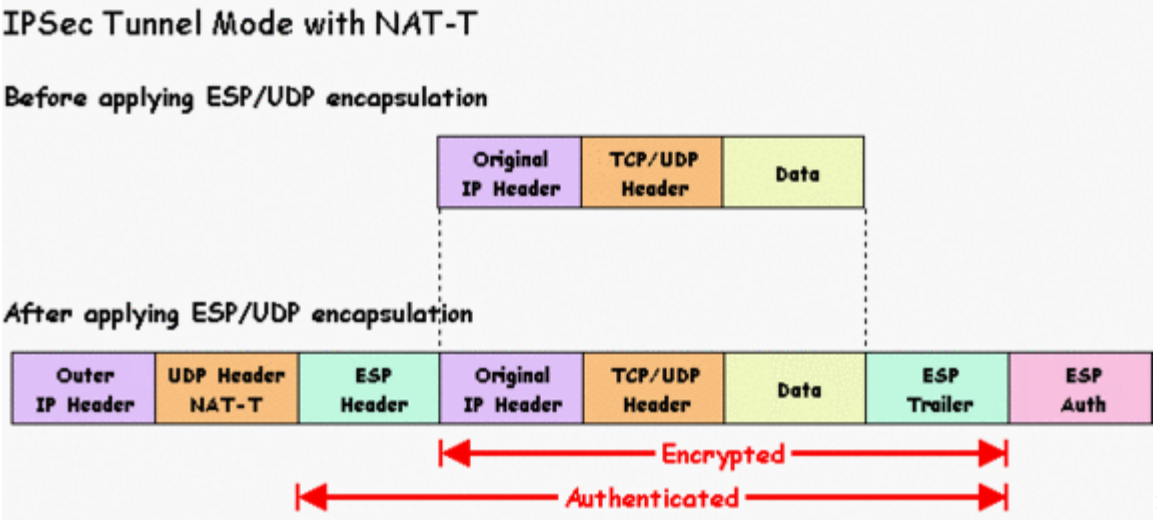
In other words, the TLS connection will use TCP/IP application-layer functionality to do the encryption. Unlike IPsec VPN, which is complicated and needs lots of client-side setup and maintenance, SSL can setup quickly, as noticed in the implementation phase of the report.

IPSec connections require a pre-shared key to exist on both the client and the server in order to encrypt and send traffic to each other. The exchange of this key presents an opportunity for an attacker to crack or capture the pre-shared key.

SSL VPNs do not have this problem because they use public-key architecture to negotiate a handshake and exchange encryption keys securely. This fact about the implementation for this project may be challenged, by presenting how SSL VPNs allow untrusted, self-signed certificates and do not verify clients which is vulnerable to man-in-the-middle attacks. This is correct, but all these flaws are manageable by using public certificates and adding client-side certificate authentication and one-time passwords (OTP) into the equation. However, TLS/SSL still has a long list of vulnerabilities.

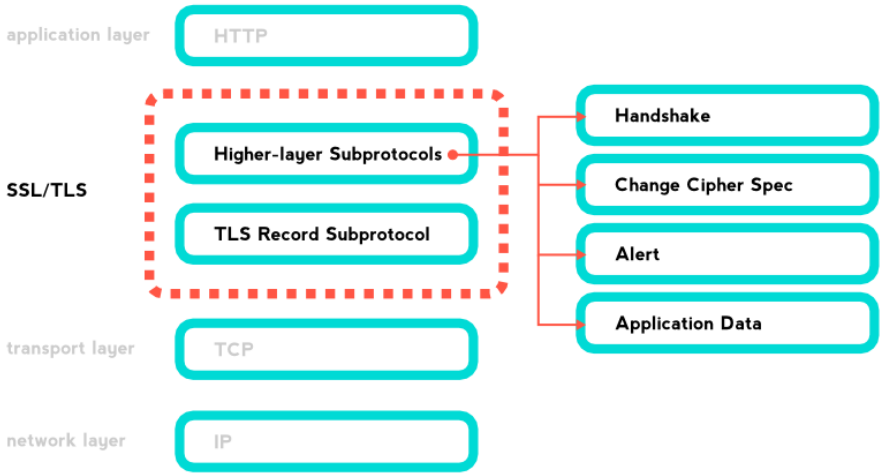
There is a significant issue about the IPSec VPN, which does not use the open-source protocol, and that creates some challenges in verifying the truthfulness of the code. Projects like “Bullrun” remind us not to apply any software or hardware using a proprietary protocol, code, firmware, or software. (BISCHOFF, n.d.)

The architecture I designed was built to work over the internet almost from everywhere, and therefore how this security protocol deals with firewalls is important. SSL-based VPNs are application layer VPNs, and they can efficiently bypass firewalls. In contrast, Encapsulating Security Payloads (ESP), a principal member of the IPSec protocol suite, have no port numbers, and that would be a problem traversing such firewalls doing NAT, which would block the whole connection. There is a workaround here to address this issue by using UDP port 4005 and encapsulating the whole packet, but using that trick would add some overhead and could cause other issues.



[18]

In comparison, SSL uses the standard HTTPs port TCP 443, and by default, this traffic is allowed and Firewall NAT friendly. I specifically used this part of the SSL functionality to bypass the limited Internet access, if needed by this architecture.



[19]

Spoke Hardware spec.

The most challenging part of the project was the TP-Link Archer C7 ver: 2.0 hardware specification, and making sure that this hardware is compatible with the OpenWRT.

[18] *http://techgenix.com/ipsec_passthrough/*

[19] *http://blog.fourthbit.com/2014/12/23/traffic-analysis-of-an-ssl-slash-tls-session/*

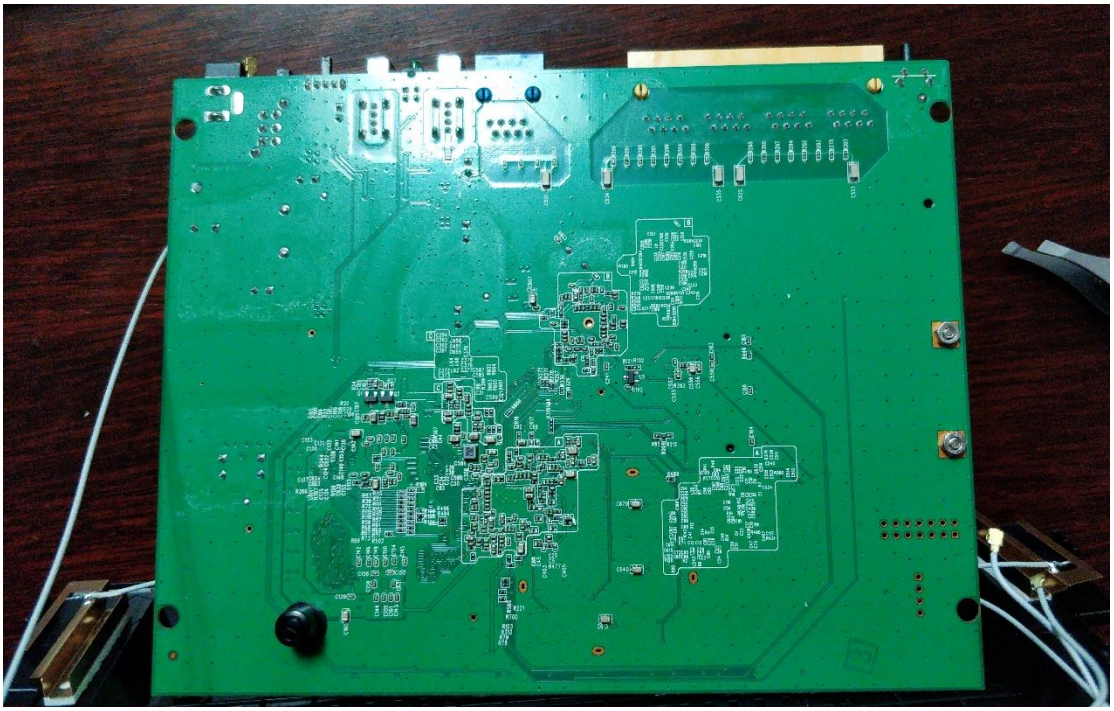
After briefly studying this router, I noticed that this hardware could be identified as one of the most reliable WiFi Routers in the market, and also has an excellent hardware design. This gateway has multiple hardware revisions, 1 to 5, that are each a bit different.

[20]

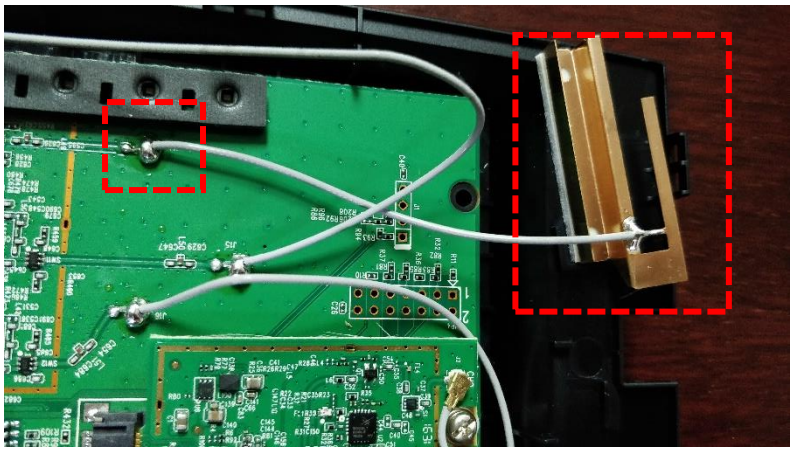
[21]In order to better view the hardware, let us have a look at the motherboard first.

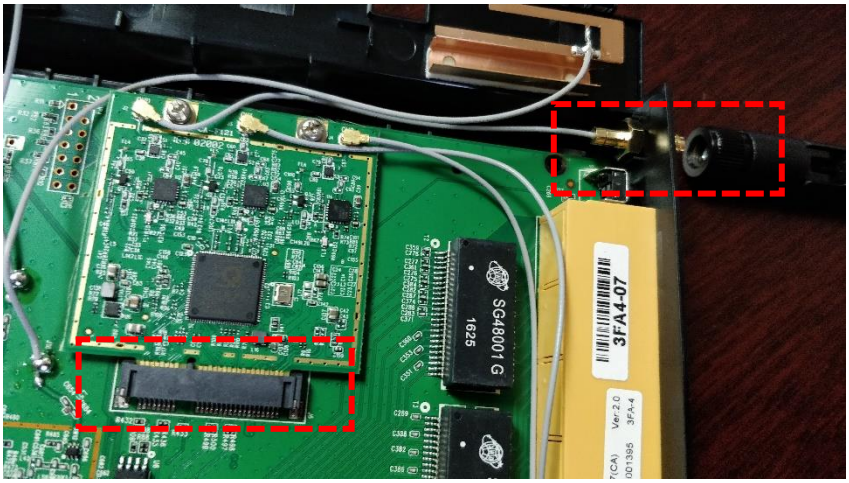
[20] *Lable shows the version of the hardware Ver2.0.*

[21]



Version 1 to 3 uses independent antennas for 2.4Ghz and 5Ghz. The 2.4Ghz uses a pigtail cable to connect to the internal copper antenna attached to the frame, while the 5Ghz connects to the external antenna at the back of the frame.





In contrast, version 4 to 5 are equipped with three dual 2.4 and 5GHz antennas.

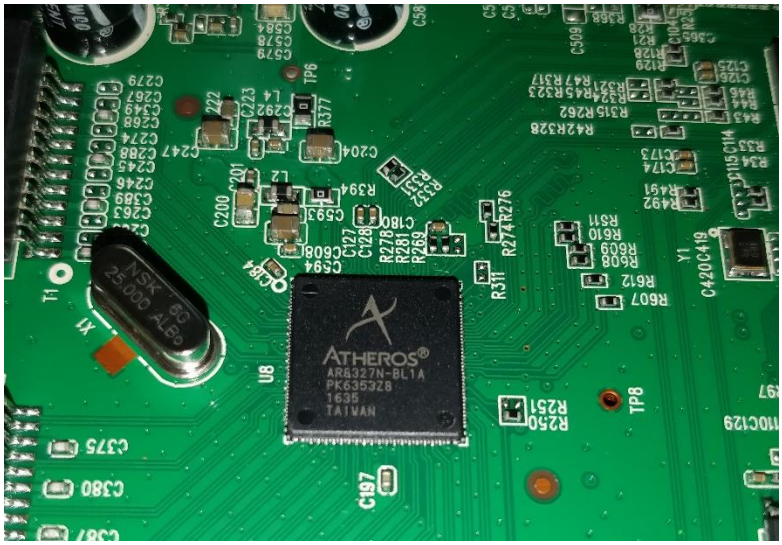
Other than that, Version 1-3 uses a Mini PCI-e slot on the motherboard to attach the external 5Ghz network card, but on newer revisions, a card is embedded on the motherboard.

The second step was hacking the hardware and creating a way to access the CLI (command-line interface) console of the WiFi Router in order to change the firmware.

It is good to know that boot loader on this Wif Router is U-Boot, which is an opensource universal boot loader.

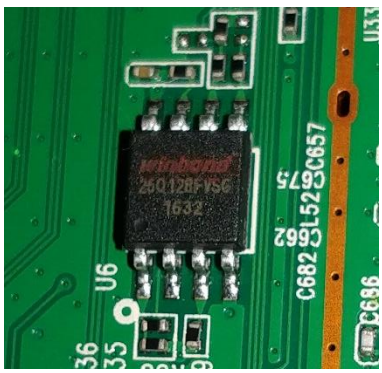
The processor is a Qualcomm Atheros QCA9558-AT4A 720MHz MIPS32® 74Kc™ core from MIPS Technologies equipped with a 3×3 MIMO for 2.4GHz 802.11b/g/n. This microcontroller is a high-performance, low-power, 32-bit MIPS RISC core. (mips, n.d.) (wikipedia, n.d.)

The processor on the card is a QCA9880-BR4A equipped with 3×3 MIMO for 5GHz 802.11a/n/ac. (qca9880-br4a, n.d.)



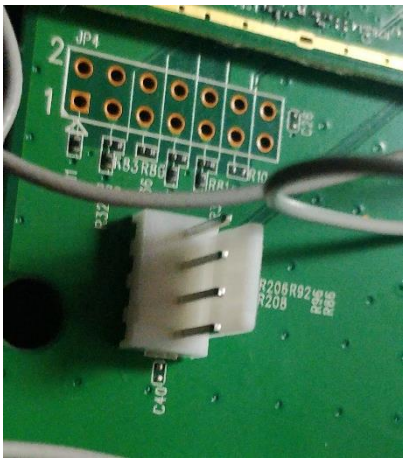
The other chip on the board is an AR8327N-BL1A, which is the 7-port Gigabit Ethernet switch chip with non-blocking switch fabric, a high-performance lookup unit with 2048 MAC address, and a four-traffic class Quality of Service (QoS) engine. This chip also supports hardware NAT, including the basic NAT and Network Address Port Translation in full, restricted, port restricted and symmetric NAT. (AR8327, n.d.)

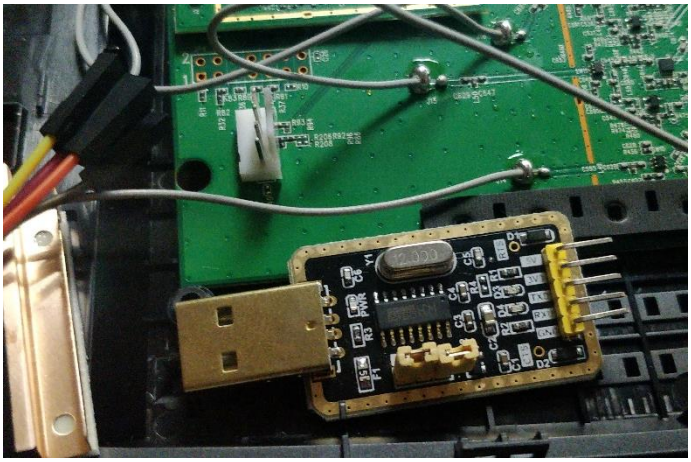
The other significant chip on the board that I can locate two of are Winboard W9751G6KB-25 with DRAM 512Mb DDR2-800, x16, which means the total memory of this board is one Gigabyte. (winbond, n.d.) Also, a Non-Volatile Flash NOR IC can be seen on the system W25Q128FVSIG. (digikey, n.d.)



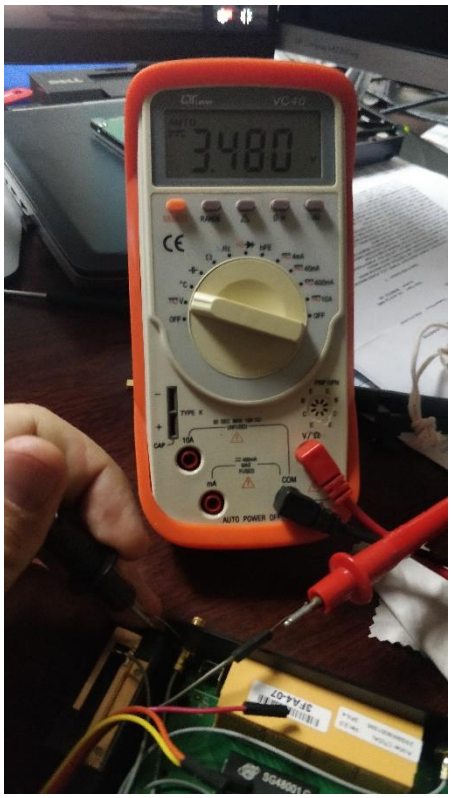
The other two essential parts of the board are a serial port Action use to access the CLI of the system, and the JTAG (Test Group) port, which is placed there for verifying designs and testing, and to write software and data into internal non-volatile Flash. (archer-c7-1750, n.d.)

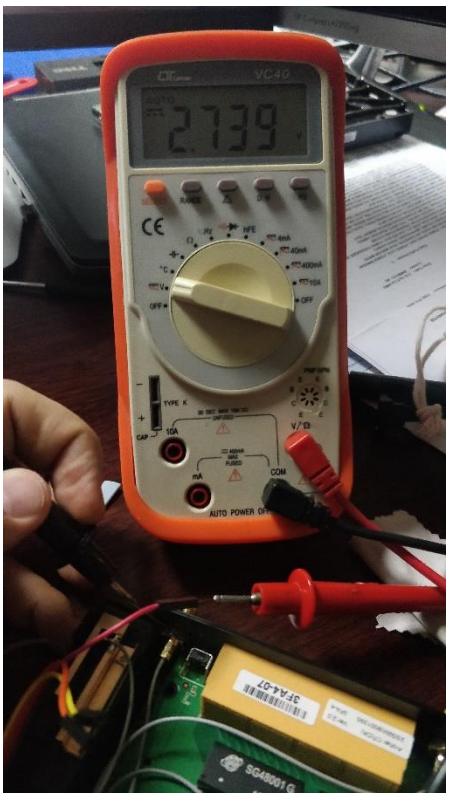
To get access to the console port, a JST connector was needed, and then a USB to Serial (RS232) adapter could be used.



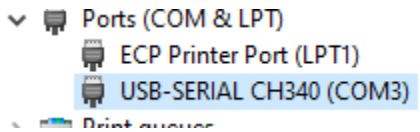


At this step, I needed to find the pinot to determine the VCC/GND/TX and RX. I can easily use a voltmeter to do this job by grounding the voltmeter by antenna connector and connecting the other end to the pins. The VCC was 3.4, th TX was at a voltage around 3 volts, and the RX and Ground were at 0 volts.





Moreover, USB adapter detected as COM3, and I started PuTTY using serial COM3, and Baud-rate 115200 DataBits 8 and Stopbits 1 and no Flowcontrol.



Move to OpenWRT

To perform this action, I first needed to make sure that I did not do any official firmware upgrades from the TP-Link website. Moreover, I needed to check the compatibility matrix on OpenWRT website to see which firmware would work with the platform.

The following is the compatibility matrix of OpenWRT TP-Link Archer C7 ver: 2.0 (TableOfHardware, n.d.)

Filtered by brand*~tp-link & model*~Archer C7

Show all (remove filter/sort)

#	↓ Brand	Model	Versions	Supported Current Release	Device Page	Device Techdata
	tp-link	Archer C7				
1	TP-Link	Archer C7 AC1750	v2, v2.1	19.07.1	archer-c5-c7-wdr7500	View/Edit data
2	TP-Link	Archer C7R WDR7500	v3.0 (CN)	19.07.1	archer-c5-c7-wdr7500	View/Edit data
3	TP-Link	Archer C7 AC1750	v1, v1.1	19.07.1	archer-c5-c7-wdr7500	View/Edit data
4	TP-Link	Archer C7 AC1750	v4	19.07.1	archer-c5-c7-wdr7500	View/Edit data
5	TP-Link	Archer C7 AC1750	v5	19.07.1	archer-c5-c7-wdr7500	View/Edit data
6	TP-Link	Archer C7 AC1750	v3	19.07.1	archer-c5-c7-wdr7500	View/Edit data

As shown, Version 2 is compatible even with version 19.07.1, which is the latest version. I went with 18.06.1 for this project. Firmware can be downloaded from the OpenWRT website and the TP-Link router web interface can be used to upgrade.

After the upgrade, the output of the CLI will change to OpenWRT.

```
Please press Enter to activate this console.


BusyBox v1.28.4 () built-in shell (ash)

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 OpenWrt 18.06.1, r7258-5eb055306f
 -----------------------------------------------------
root@OpenWrt:/#
```

This is when I can SSH into the box using a cable to the LAN ports.

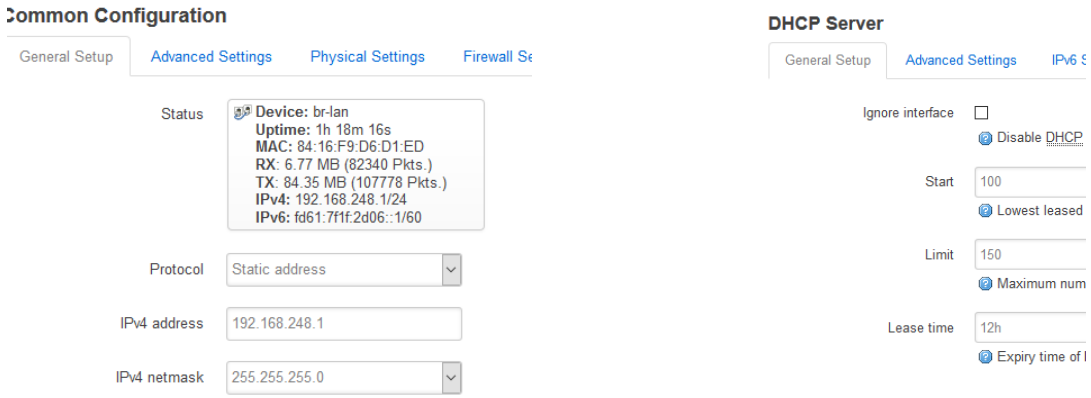
```
login as: root
root@192.168.248.1's password:


BusyBox v1.28.4 () built-in shell (ash)

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 OpenWrt 18.06.1, r7258-5eb055306f
```

```
 ----------------------------------------------------
root@OpenWrt:~#
```

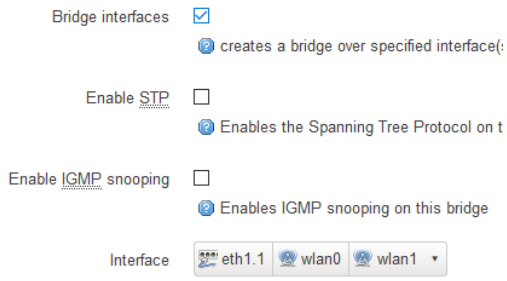
Also, I can add LuCI Web UI by installing the package using “opkg.” It is useful to mention that about 3500 different packages are available to use on the platform.

```
root@OpenWrt:/# opkg update
root@OpenWrt:/# opkg install luci
```

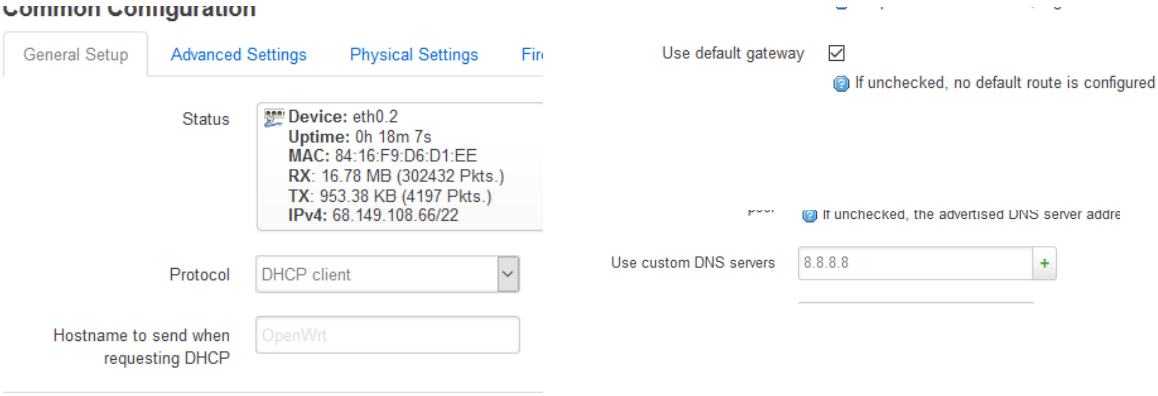
The next step is to change the LAN interface IP address to 192.168.248.1/24, and enable the DHCP to start from 100 for 150 numbers which are enough for a site.



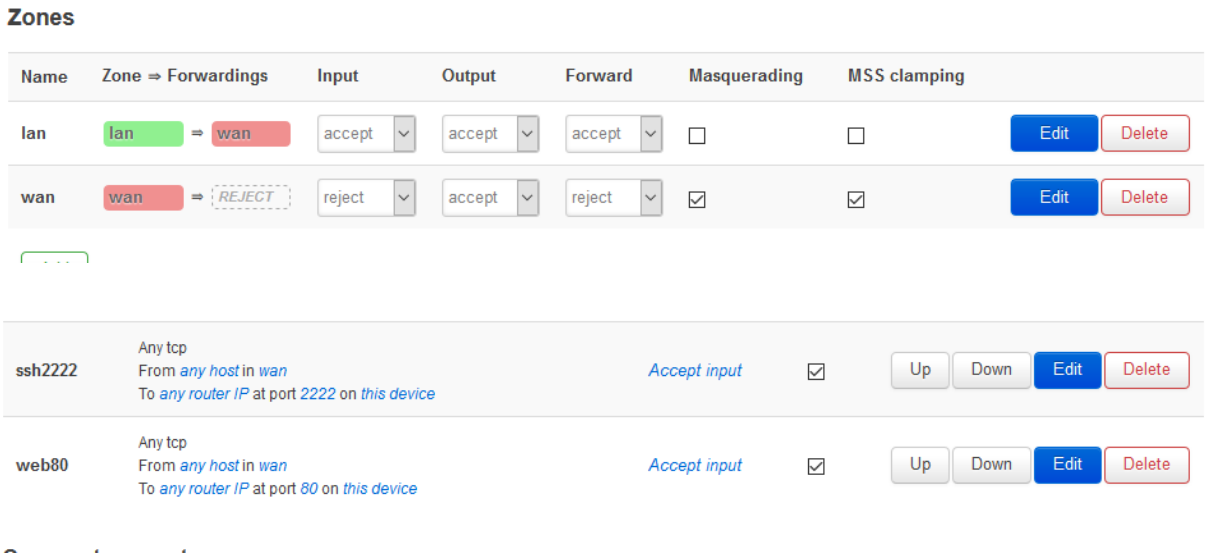
I then needed to check the bridge interface option and make sure this LAN interface (which is all four physical switch ports) is a member of a bridge. For Firewall settings, I need to include the LAN interface as a trusted interface.



The other interface I need to work on is the WAN interface. I need to be a DHCP client on that, and make sure that the default gateway checked on that, so that this interface will take the traffic out of the site to the Internet. DNS IP address is also needed to be set. WAN interface can then be set as an untrusted interface on Firewall settings.

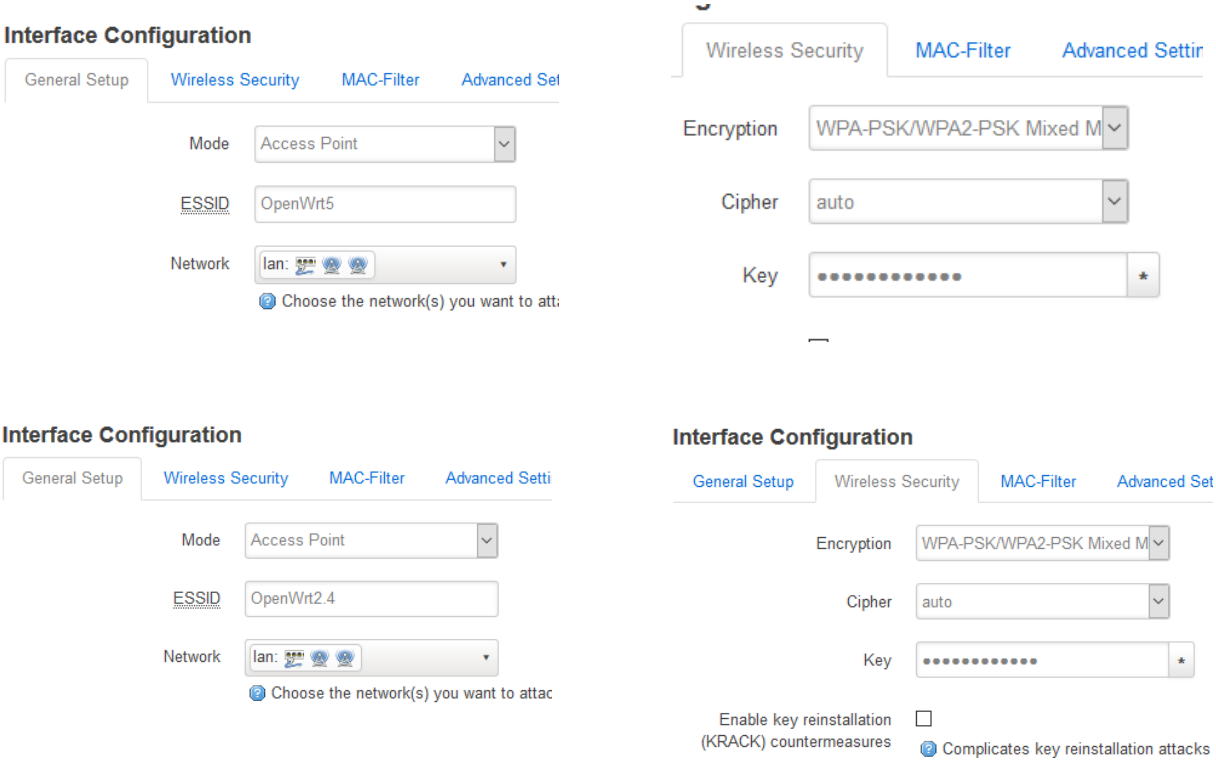


On the Firewall settings, I need to make sure that Masquerading checked to do PAT on the WAN interface. And if I want to have remote access to the site Router, I can allow the SSH and WEB on the firewall.



The other two essential interfaces are Wireless 2.4Ghz and 5Ghz that WPA-2 security settings, which need to be adjusted using the PSK (Preshared Key. A suitable channel then needs to be picked, one that has the least interference in the environment.

Both Wireless interfaces need to be assigned to the LAN bridge.



To verify the configuration, I can connect to the LAN port and get the IP address from the router, and SSH to the router, in order to verify that I get a public IP address on the WAN interface and am also able to ping the internet.

```
root@OpenWrt:~# ifconfig
br-lan    Link encap:Ethernet  HWaddr 84:16:F9:D6:D1:ED
          inet addr:192.168.248.1  Bcast:192.168.248.255  Mask:255.255.255.0
          inet6 addr: fd61:7f1f:2d06::1/60 Scope:Global
          inet6 addr: fe80::8616:f9ff:fed6:d1ed/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:85620 errors:0 dropped:0 overruns:0 frame:0
          TX packets:111445 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7072287 (6.7 MiB)  TX bytes:85915660 (81.9 MiB)

eth0      Link encap:Ethernet  HWaddr 84:16:F9:D6:D1:EE
          inet6 addr: fe80::8616:f9ff:fed6:d1ee/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:668617 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5984 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:47441303 (45.2 MiB)  TX bytes:1299279 (1.2 MiB)
          Interrupt:4
```

```
eth0.2    Link encap:Ethernet  HWaddr 84:16:F9:D6:D1:EE
          inet addr:68.149.108.66  Bcast:68.149.111.255  Mask:255.255.252.0
          inet6 addr: fe80::8616:f9ff:fed6:d1ee/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:668617 errors:0 dropped:40 overruns:0 frame:0
          TX packets:5976 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:35406197 (33.7 MiB)  TX bytes:1274487 (1.2 MiB)

eth1      Link encap:Ethernet  HWaddr 84:16:F9:D6:D1:ED
          inet6 addr: fe80::8616:f9ff:fed6:d1ed/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23059 errors:0 dropped:0 overruns:0 frame:0
          TX packets:20399 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13718616 (13.0 MiB)  TX bytes:1720724 (1.6 MiB)
          Interrupt:5

eth1.1    Link encap:Ethernet  HWaddr 84:16:F9:D6:D1:ED
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23036 errors:0 dropped:0 overruns:0 frame:0
          TX packets:20384 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13302385 (12.6 MiB)  TX bytes:1636729 (1.5 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:18238 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18238 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1676371 (1.5 MiB)  TX bytes:1676371 (1.5 MiB)

wlan0     Link encap:Ethernet  HWaddr 84:16:F9:D6:D1:EB
          inet6 addr: fe80::8616:f9ff:fed6:d1eb/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4685 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:692114 (675.8 KiB)

wlan1     Link encap:Ethernet  HWaddr 84:16:F9:D6:D1:EC
          inet6 addr: fe80::8616:f9ff:fed6:d1ec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:100089 errors:0 dropped:0 overruns:0 frame:0
          TX packets:135728 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9172246 (8.7 MiB)  TX bytes:103511093 (98.7 MiB)

root@OpenWrt:~#
```

I can then check the routing table.

```
root@OpenWrt:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         68.149.108.1    0.0.0.0         UG    0      0        0
eth0.2
68.149.108.0    0.0.0.0         255.255.252.0   U     0      0        0
eth0.2
192.168.248.0   0.0.0.0         255.255.255.0   U     0      0        0 br-
lan
root@OpenWrt:~#
root@OpenWrt:~# ping www.google.com
PING www.google.com (172.217.3.196): 56 data bytes
64 bytes from 172.217.3.196: seq=0 ttl=57 time=30.504 ms
64 bytes from 172.217.3.196: seq=1 ttl=57 time=29.394 ms
```

I then needed to add the OpenConnect Client to this Router using “opkg” installer. (opkg, n.d.)

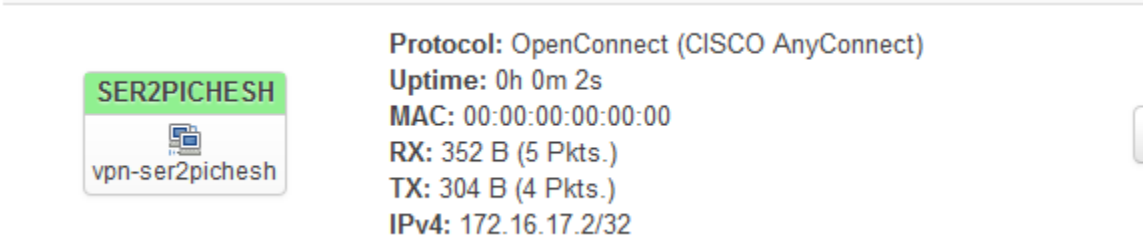
```
root@OpenWrt:~# opkg install openconnect
Package openconnect (7.08-8) installed in root is up to date.
```

All interface configuration like I did above will record on /etc/config/network. Therefore, I can add the openconnect client settings in that file. The connection was named “'ser2pichesh”, and it attempted to make a connection to the public server name “ser2.pichesh.com” on port “543.” (openconnect, n.d.)

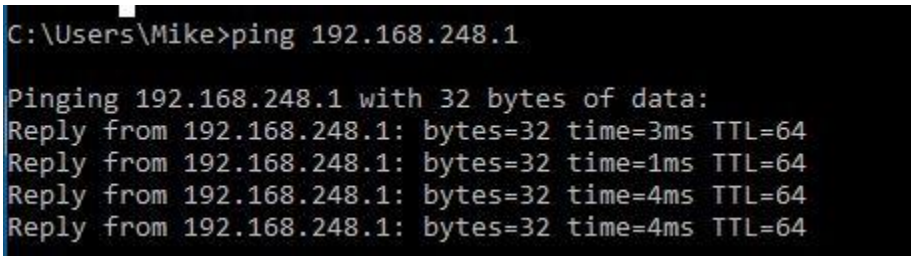
```
root@OpenWrt:/# cat /etc/config/network

config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config globals 'globals'
        option ula_prefix 'fd61:7f1f:2d06::/48'

config interface 'lan'
        option type 'bridge'
        option ifname 'eth1.1'
        option proto 'static'
        option netmask '255.255.255.0'
        option ip6assign '60'
        option ipaddr '192.168.248.1'

config interface 'wan'
        option ifname 'eth0.2'
        option proto 'dhcp'
        option peerdns '0'
        option dns '8.8.8.8'
        option metric '0'
```

```
config interface 'wan6'
        option ifname 'eth0.2'
        option proto 'dhcpv6'
        option reqaddress 'try'
        option reqprefix 'auto'
        option peerdns '0'
        option auto '0'

config switch
        option name 'switch0'
        option reset '1'
        option enable_vlan '1'

config switch_vlan
        option device 'switch0'
        option vlan '1'
        option ports '2 3 4 5 0t'

config switch_vlan
        option device 'switch0'
        option vlan '2'
        option ports '1 6t'

config interface 'ser2pichesh'
        option proto 'openconnect'
        option server 'ser2.pichesh.com'
        option port '543'
        option authgroup 'DEFAULT'
        option serverhash 'ff7be8c2f88edbfb29074950354659e5f5924903'
        option username 'openwrt'
        option password 'wrtopen'
        option auto '0'

root@OpenWrt:/#
```

When I start the OCclient connection on the LuCI, it connects and gets a correct IP address from the OCserver DHCP pool 172.16.17.x/24.



To test the project targets, I checked and made sure that in the first step I can ping my default gateway, while I am connected wirelessly from a laptop to the router. I needed to also check to see if the VPN server is pingable, considering that I do not have any PAT setup over the VPN tunnel, and IP addresses shows the same on the other end of the tunnel.



As is shown, ping to the router was successful but ping to the OCserver was not, and by checking the routing table, I noticed that the VPN connection did not take the default route of the system. There is an option on the OpenConnect connection to set as default gateway, but that did not solve the issue. If I enable that function, and establish the tunnel, the tunnel destination would take the gateway and internet service for the tunnel itself would get lost. This means it would redirect all the traffic to the tunnel, but the correct route is to keep the tunnel traffic the same way towards the internet.

Technical challenges

The first challenge I faced was to fix the connectivity issue. Network script was needed to be adjusted in such a way that OCclient takes over the role of the default gateway.

```
root@OpenWrt:/lib/netifd# pwd
/lib/netifd
root@OpenWrt:/lib/netifd# ls
dhcp.script          hostapd.sh           netifd-wireless.sh   ppp-up
proto                vpnc-script          wireless
dhcpv6.script        netifd-proto.sh      ppp-down             ppp6-up
utils.sh             vpnc-script.back
root@OpenWrt:/lib/netifd#
```

The file which controls the VPN connections is vpnc-script. In the following, I show the change I made on the code to make it work. (Pierre-Francois, n.d.)

```
DEFAULT_ROUTE_FILE=/var/etc/openconnect-defaultroute

get_default_gw() {
    netstat -r -n | awk '/:/ { next; } /^(default|0\.0\.0\.0)/ { print $2; }'
```

```
}

set_vpngateway_route() {
    route del -host "$VPNGATEWAY"
    route add -host "$VPNGATEWAY" gw "`get_default_gw`"
    echo "`get_default_gw`" > "$DEFAULT_ROUTE_FILE"
}

del_vpngateway_route() {
    route del -host "$VPNGATEWAY"
    if [ -s "$DEFAULT_ROUTE_FILE" ]; then
        route add default gw `cat "$DEFAULT_ROUTE_FILE"`
        rm -f -- "$DEFAULT_ROUTE_FILE"
    fi
}
.
.
.
set_vpngateway_route
.
.
.
del_vpngateway_route
```

The whole technicality behind this code is to save the system default gateway (WAN Internet) on a file and only use it for VPN Server packets, and then all other packets will forward to the tunnel end (OCserv). Furthermore, in reverse, all those added routes will be removed and the system default gateway will read from that file and replace.

That issue was fixed, but still the VPN server remains not pingable. That was because of the return route that I needed to add to the OCserv. OCserv needs to know where to look for 192.168.248.x, and the simple fix was adding a route to the OCserv. Nevertheless, the route gets added only if the spoke router makes a successful VPN connection. I automatized this job with Linux crontab and a bash script file.

Crontab will check the reachability of the site (172.16.17.2) and right after the site becomes reachable, it will add the return route to the table. If the spoke loses its connection to the OCserv, the route will delete automatically, because the VPN's interface will go away.

```
[root@oc ocserv]# cat /script/addroute.sh
#!/bin/bash
/bin/ping -c 1 172.16.17.2 >> /dev/null
if [ $? -eq 0 ] ; then
/usr/sbin/ip route add  192.168.248.0/24 via 172.16.17.2
fi
[root@oc ocserv]#
[root@oc ocserv]# crontab -l
* * * * * /script/addroute.sh
[root@oc ocserv]#
```

After that change the ping works fine, with no issues.

```
C:\Users\Mike>ping 172.16.17.1

Pinging 172.16.17.1 with 32 bytes of data:
Reply from 172.16.17.1: bytes=32 time=23ms TTL=63
Reply from 172.16.17.1: bytes=32 time=19ms TTL=63

Ping statistics for 172.16.17.1:
    Packets: Sent = 2, Received = 2, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 19ms, Maximum = 23ms, Average = 21ms
Control-C
^C
```

The other aspect that remains to be mentioned is the secret behind reserving an IP address for a site, because as I setup, the return route is based on the IP address of a site, and that should define and stay unchanged.

To accomplish that task and have better manageability over the credentials, a centralized Radius server was established to manage credentials, and also dictate the VPN peer IP address.

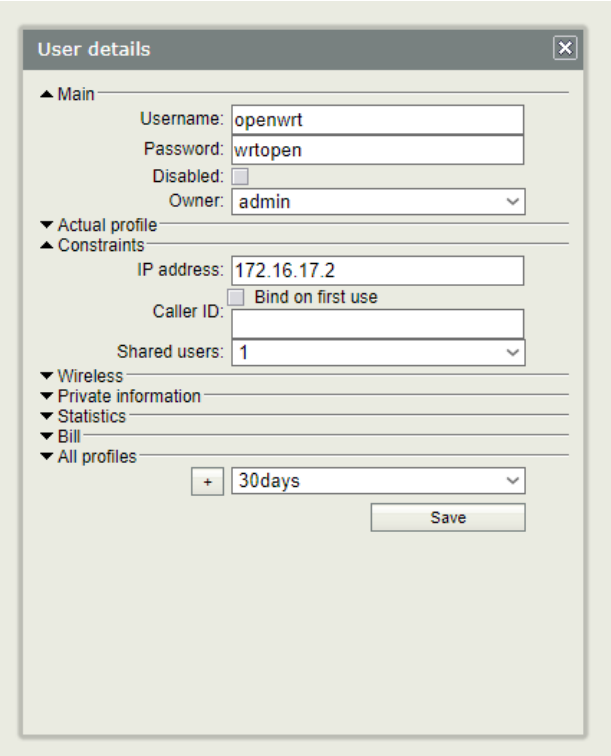
Also need to be mentioned, is that OpenConnect server setup needs to be adjusted to accept Radius, and changes need to be applied on the Radius server side. (Gaspari, n.d.)

```
[root@oc ~]# vi /etc/ocserv/ocserv.conf
auth = "radius [config=/etc/radcli/radiusclient.conf,groupconfig=true]"
```

```
[root@oc ~]# vi /etc/radcli/radiusclient.conf
nas-identifier MikroUser
authserver 192.168.254.2
acctserver 192.168.254.2
servers         /etc/radcli/servers
dictionary      /etc/radcli/dictionary
default_realm
radius_timeout  10
radius_retries  3
bindaddr *
```

```
[root@oc ~]# cat /etc/radcli/servers
## Server Name or Client/Server pair            Key
## ----------------                             ---------------
#
#portmaster.elemental.net                       hardlyasecret
#portmaster2.elemental.net                      donttellanyone
#
## uncomment the following line for simple testing of radlogin
```

```
## with freeradius-server
#
#localhost/localhost                                    testing123
192.168.254.2 asd2345
You have mail in /var/spool/mail/root
[root@oc ~]#
```



PERFORMANCE TESTS

A well-known tool to run performance tests in the network is Iperf. The Spoke gateway was moved to the other location and test results were collected as below.

Iperf results show a slow but stable connection over the internet, which is enough to satisfy many cases of need for corporate resource use in remote locations.

```
C:\Users\admin\Downloads\iperf-3.1.3-win64>iperf3.exe -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.248.153, port 51003
[  5] local 192.168.254.159 port 5201 connected to 192.168.248.153 port 51005
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec   264 KBytes  2.16 Mbits/sec
[  5]   1.00-2.00   sec  1009 KBytes  8.26 Mbits/sec
[  5]   2.00-3.00   sec   716 KBytes  5.87 Mbits/sec
[  5]   3.00-4.00   sec  1.19 MBytes  9.96 Mbits/sec
[  5]   4.00-5.00   sec  1.31 MBytes  11.0 Mbits/sec
[  5]   5.00-6.00   sec  1.58 MBytes  13.3 Mbits/sec
[  5]   6.00-7.00   sec  1.34 MBytes  11.3 Mbits/sec
[  5]   7.00-8.00   sec   313 KBytes  2.56 Mbits/sec
[  5]   8.00-9.00   sec  1.25 MBytes  10.5 Mbits/sec
[  5]   9.00-10.00  sec  1.20 MBytes  10.0 Mbits/sec
[  5]  10.00-10.04  sec  70.8 KBytes  15.0 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-10.04  sec  0.00 Bytes  0.00 bits/sec                  sender
[  5]   0.00-10.04  sec  10.2 MBytes  8.52 Mbits/sec                  receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
iperf3: interrupt - the server has terminated
```

CONCLUSION

This type of experiment creates an excellent opportunity for network engineers to consider all challenges of building a solution for a certain need, rather than purchasing a ready to run product from a provider, with instructions for implementation and troubleshooting. This project also breaks the preconceived notion which implies that the only way of running a stable system is to follow the market-leading solution providers. Additionally, the experiment helps to form a self-awareness and confidence that making an extensive system is always possible if one has a good understanding of the technologies. All factors considered, this research helped me understand the limitless power of GNU Open-source Hardware and Software and their applications.

Michael Wise
Graduate student in Computing Science
University of Alberta MINT program.

BIBLIOGRAPHY

AR8327. (n.d.). Retrieved from lafibre: https://lafibre.info/images/doc/201106_spec_AR8327.pdf

archer-c7-1750. (n.d.). Retrieved from openwrt: https://openwrt.org/toh/tp-link/archer-c7-1750

BISCHOFF, P. (n.d.). *ipsec-vs-ssl-vpn*. Retrieved from comparitech: https://www.comparitech.com/blog/vpn-privacy/ipsec-vs-ssl-vpn/

centos. (n.d.). *centos install-guide*. Retrieved from centos: https://docs.centos.org/en-US/centos/install-guide/

digikey. (n.d.). Retrieved from https://www.digikey.ca/product-detail/en/winbond-electronics/W25Q128FVSIG/W25Q128FVSIG-ND/3008697

fedoraproject. (n.d.). *EPEL*. Retrieved from wiki: https://fedoraproject.org/wiki/EPEL

Gaspari, M. (n.d.). *authentication-radius-radcli*. Retrieved from ocserv: https://ocserv.gitlab.io/www/recipes-ocserv-authentication-radius-radcli.html

gitlab. (n.d.). *ocserv-installation-generic*. Retrieved from ocserv: https://ocserv.gitlab.io/www/recipes-ocserv-installation-generic.html

Guoan, X. (n.d.). Retrieved from linuxbabe: https://www.linuxbabe.com/ubuntu/certificate-authentication-openconnect-vpn-server-ocserv

kvasirsg. (n.d.). *ip-forwarding*. Retrieved from kvasirsg: https://docs.kvasirsg.com/centos-7/prefilight-configuration/how-to-enable-ip-forwarding

Linode. (n.d.). *firewalld-on-centos*. Retrieved from linode: https://www.linode.com/docs/security/firewalls/introduction-to-firewalld-on-centos/

mips. (n.d.). Retrieved from https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00346-2B-24K-DTS-04.00.pdf

openconnect. (n.d.). Retrieved from openwrt: https://openwrt.org/docs/guide-user/services/vpn/openconnect

opkg. (n.d.). Retrieved from openwrt: https://openwrt.org/docs/guide-user/additional-software/opkg

Pierre-Francois. (n.d.). *issues*. Retrieved from github: https://github.com/openwrt/packages/issues/2548

qca9880-br4a. (n.d.). Retrieved from arrow: https://www.arrow.com/en/products/qca9880-br4a/qualcomm

Red_Hat_Enterprise_Linux. (n.d.). *Networking_Config_Using_nmtui*. Retrieved from Networking_Guide: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Networking_Guide/sec-Networking_Config_Using_nmtui.html

Singer, D. (n.d.). Retrieved from https://www.liquidweb.com/kb/enable-epel-repository/

TableOfHardware. (n.d.). Retrieved from openwrt: https://openwrt.org/toh/start

Transport_Layer_Security. (n.d.). Retrieved from wikipedia: https://en.wikipedia.org/wiki/Transport_Layer_Security

VMware. (n.d.). *installation-setup-guide*. Retrieved from vsphere-esxi-vcenter: https://docs.vmware.com/en/VMware-vSphere/5.5/vsphere-esxi-vcenter-server-552-installation-setup-guide.pdf

wikipedia. (n.d.). *List_of_MIPS_architecture_processors*. Retrieved from https://en.wikipedia.org/wiki/List_of_MIPS_architecture_processors

winbond. (n.d.). Retrieved from https://www.winbond.com/resource-files/da00-w9751g6kbg1.pdf