GRE- Generic Routing Encapsulation

What is a GRE Tunnel?

GRE stands for Generic Routing Encapsulation, which is a very simple form of tunneling.

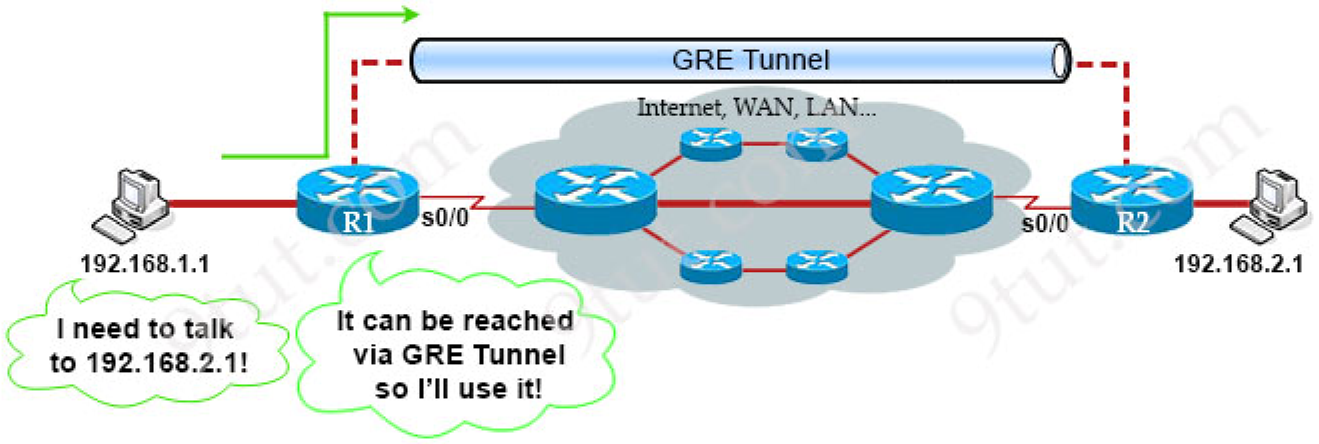
In general, GRE allows a tunnel to be created using a certain protocol, which then hides the contents of another protocol carried within the tunnel. Tunneling provides a mechanism to transport packets of one protocol within another protocol. The protocol that is carried is called as the passenger protocol, and the protocol that is used for carrying the passenger protocol is called as the transport protocol.

Generic Routing Encapsulation (GRE) is one of the available tunneling mechanisms which uses IP as the transport protocol and can be used for carrying many different passenger protocols.
The tunnels behave as virtual point-to-point links that have two endpoints identified by the tunnel source and tunnel destination addresses at each endpoint.

With GRE we can easily create a virtual link between routers and allow them to be directly connected, even if they physically aren’t.

How a GRE Tunnel works?

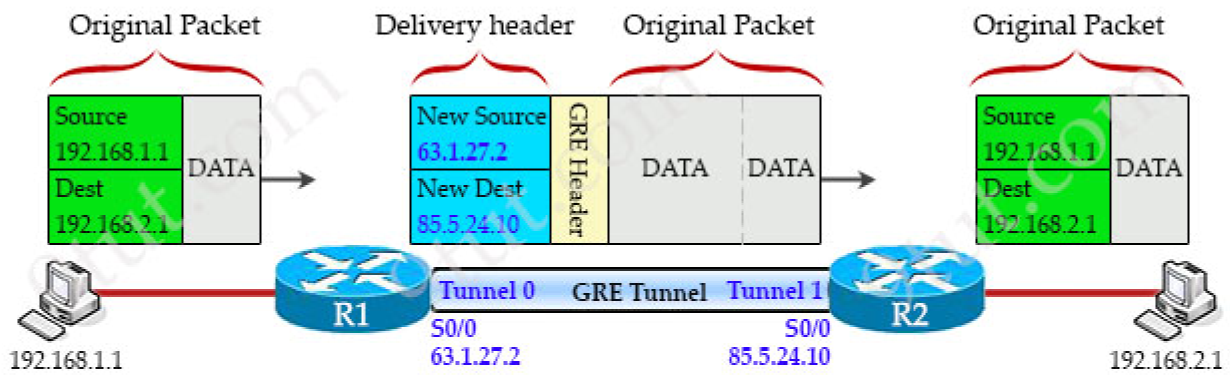
Let’s have a look at the topology below:


GRE Tunnel
Internet, WAN, LAN...
R1
s0/0
s0/0
R2
192.168.1.1
192.168.2.1
I need to talk to 192.168.2.1!
It can be reached via GRE Tunnel so I'll use it!

Suppose R1 and R2 are routers at two far ends of the same company. They are connected to two computers who want to communicate. Although R1 and R2 are not physically connected to each other but with GRE Tunnel, they appear to be! This is great when you have multiple end points and don't care the path between them. The routing tables of two routers show that they are directly connected via GRE Tunnel.

When the sending router decides to send a packet into the GRE Tunnel, it will "wrap" the whole packet into another IP packet with two headers: one is the GRE header (4 bytes) which uses to manage the tunnel itself. The other is called "Delivery header" (20 bytes) which includes the new source and destination IP addresses of two virtual interfaces of the tunnel (called tunnel interfaces).

This process is called encapsulation.



In the example above when R1 receives an IP packet, it wraps the whole packet with a GRE header and a delivery header. The delivery header includes new source IP address of 63.1.27.2 (the IP address of R1's physical interface which is used to create tunnel) and new destination IP address of 85.5.24.10 (the IP address of R2's physical interface which is used to create tunnel).

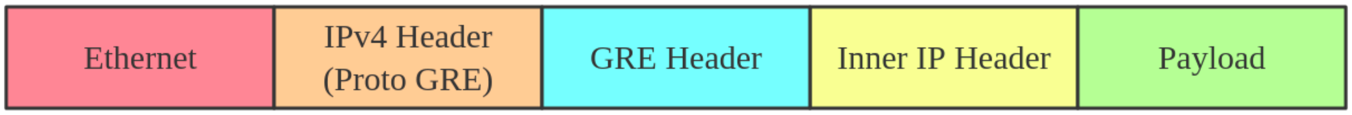
When the GRE packet arrives at the other end of the tunnel (R2 in this case), the receiving router R2 needs to remove the GRE header and delivery header to get the original packet.

Note: The IP addresses of the two ends of GRE Tunnel (63.1.27.2 & 85.5.24.10 in this case) can be in the same subnet or different subnet (like over the Internet with public IP addresses), provided that two routers know how to reach each other's tunnel IP address. For example in this case R1 must know how to reach 85.5.24.10 and R2 must reach 63.1.27.2.

GRE tunnels are stateless which means the tunnel endpoint does not keep any information about the state or availability of the remote tunnel endpoint. Therefore the local tunnel endpoint does not bring the line protocol of the GRE tunnel interface down if the remote tunnel endpoint is unreachable. For example, if R2 tunnel interface is brought down for some reason, R1 tunnel interface will remain in up state.

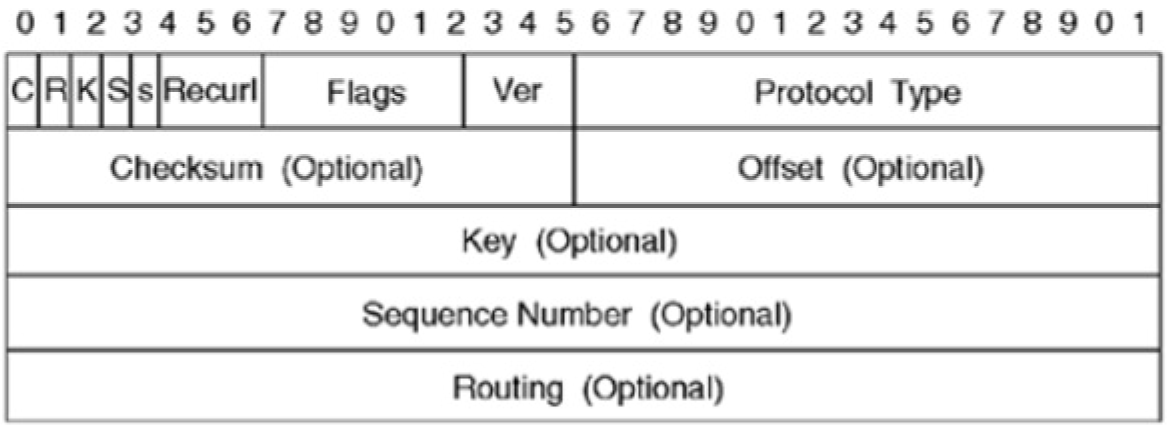
Ethernet Frame with GRE Encapsulation

GRE tunneling adds an additional GRE header between the inside and outside IP headers. In theory, GRE could encapsulate any Layer 3 protocol with a valid Ethernet type, unlike IPIP, which can only encapsulate IP. The GRE header looks like:



GRE Packet Header

A GRE packet header structure is represented in the diagram below.



Why Use a GRE Tunnel?

Connect two Remote sites ie connection of non-contiguous subnetworks

Two Routers can be made Layer 2 Adjacent even if they are connected across say the internet.

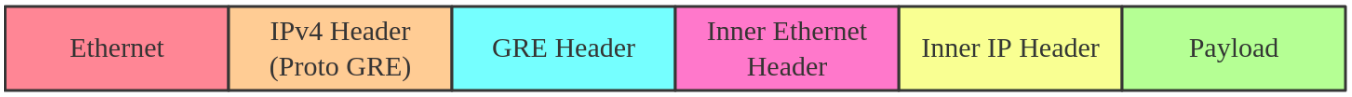
Inner IP Packets TTL will not be decremented untill it is recieved by the Router on the other end of the tunnel and decapsulated.

Therefor GRE tunnels provide workarounds for network configuration with limited hops before a packet is dropped.

Unlike VPN which does not support multicast, GRE tunnel does support multicast so many popular routing protocols (like OSPF, EIGRP) can operate along with.

Another use case is routing an IPV6 packet across IPV4 network in between.

While GRE tunnels operate at OSI Layer 3, GRETAP works at OSI Layer 2, which means there is an Ethernet header in the inner header.

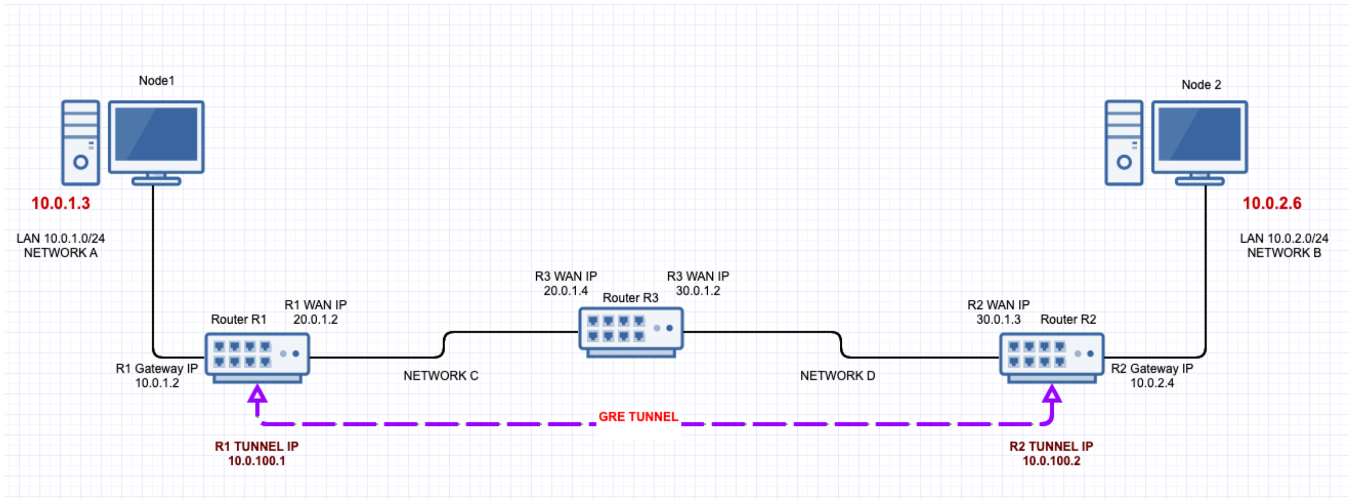


Eg:GRETAP can be used to encapsulate Ethernet Frames in a Wifi Frame at Layer 2.

GRE tunnels lack mechanisms related to flow-control and security by default

It is important to note that packets travelling inside a GRE tunnel are not encrypted as GRE does not encrypt the tunnel but encapsulates it with a GRE header. If data protection is required, IPSec must be configured to provide data confidentiality – this is when a GRE tunnel is transformed into a secure VPN GRE tunnel.

How to Create a GRE Tunnel?



Our goal is to inter-connect the two Nodes Node1 and Node2 which are on different local area networks separated by WAN.

We can simulate such a network environment using Virtualbox.

Simulation

1. Setup Four Internal Networks in Virtualbox.

   Each internal network will have a Virtualbox virtual switch and Dhcp server can be added.

```
VBoxManage dhcpserver add --netname intnetA --ip 10.0.1.1 --netmask 255.255.255.0 --lowerip 10.0.1.2
--upperip 10.0.1.254 --enable
VBoxManage dhcpserver add --netname intnetB --ip 10.0.2.1 --netmask 255.255.255.0 --lowerip 10.0.2.2
--upperip 10.0.2.254 --enable
VBoxManage dhcpserver add --netname intnetC --ip 20.0.1.1 --netmask 255.255.255.0 --lowerip 20.0.1.2
--upperip 20.0.1.254 --enable
VBoxManage dhcpserver add --netname intnetD --ip 30.0.1.1 --netmask 255.255.255.0 --lowerip 30.0.1.2
--upperip 30.0.1.254 --enable
```

2. We can now proceed and create the. Nodes and Routers with a lightweight Linux distro such as Alpine Linux.

   https://alpinelinux.org/downloads/

   We will use in total 3 Routers and 2 Nodes ,so 5 VM instances of the same ISO.

   Tip:Create one VM Instance and create Linked Clones for better space utilization on the host.

   Configure the Network Settings of the VM according to the scheme in the diagram above.

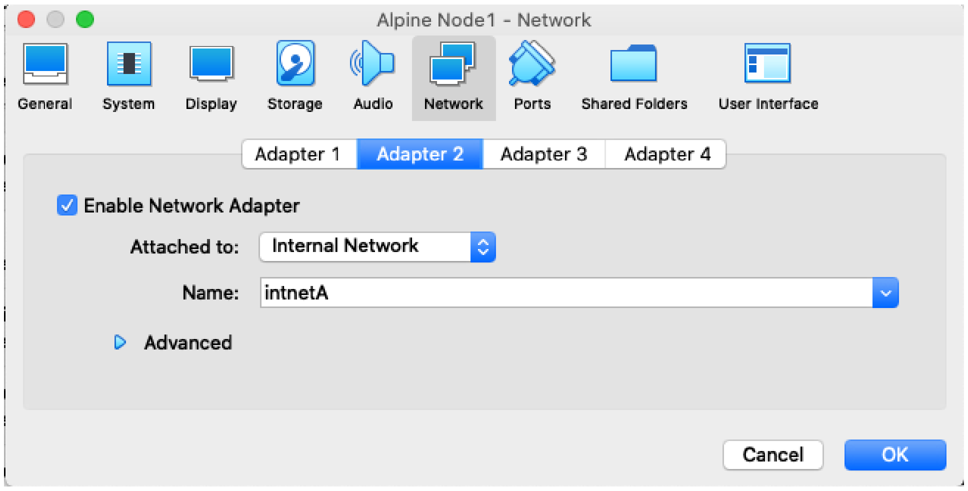
   Example:

   Node 1

   Only **one** network adapter should be enabled.

   This will be of type internal network

   Enter name of the network created using the VBoxManage command for network A. (intnetA)



Enable Network Adapter ✓

Attached to: Internal Network

Name: intnetA

Advanced

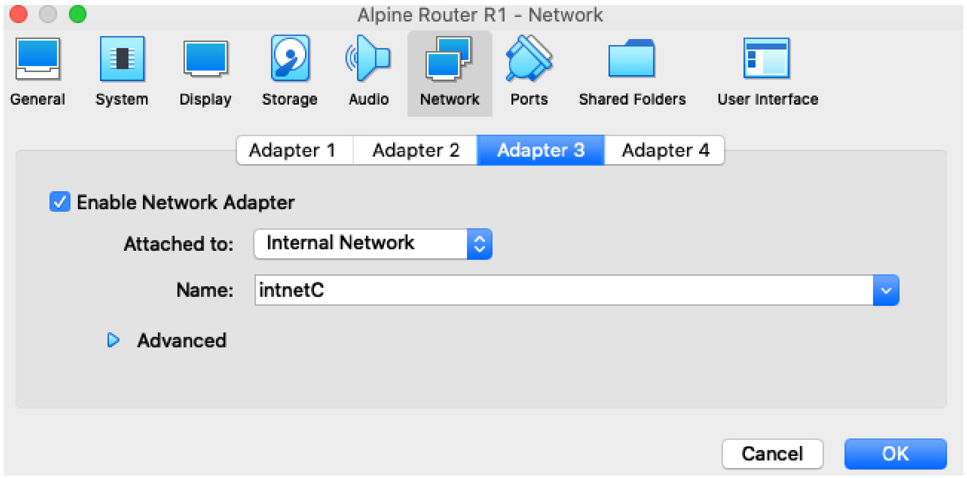
Router R1

For Routers **two** network adapters should be enabled.

Both will be of type internal network

Enter name of the network created using the VBoxManage command for network A for one Adapter. (intnetA)

Enter name of the network created using the VBoxManage command for network C for the other Adapter. (intnetC)

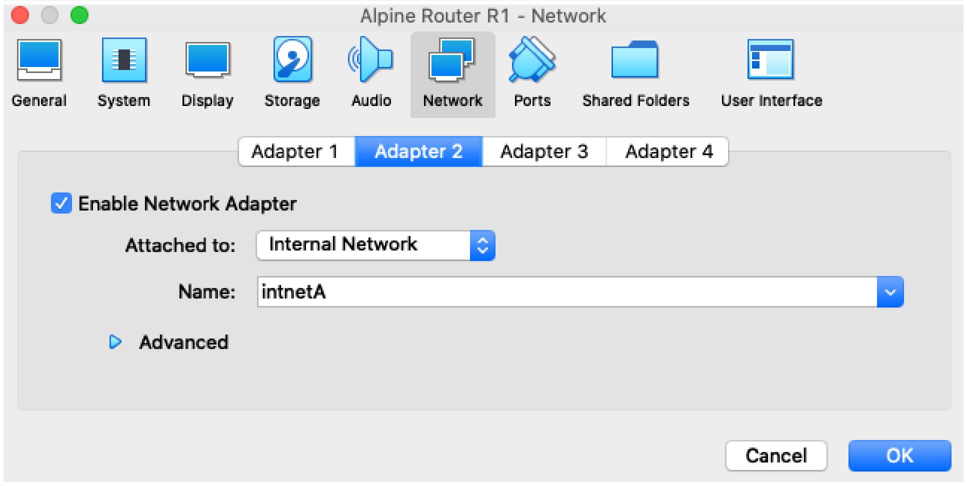


Enable Network Adapter ✓

Attached to: Internal Network

Name: intnetC

Advanced



Enable Network Adapter ✓

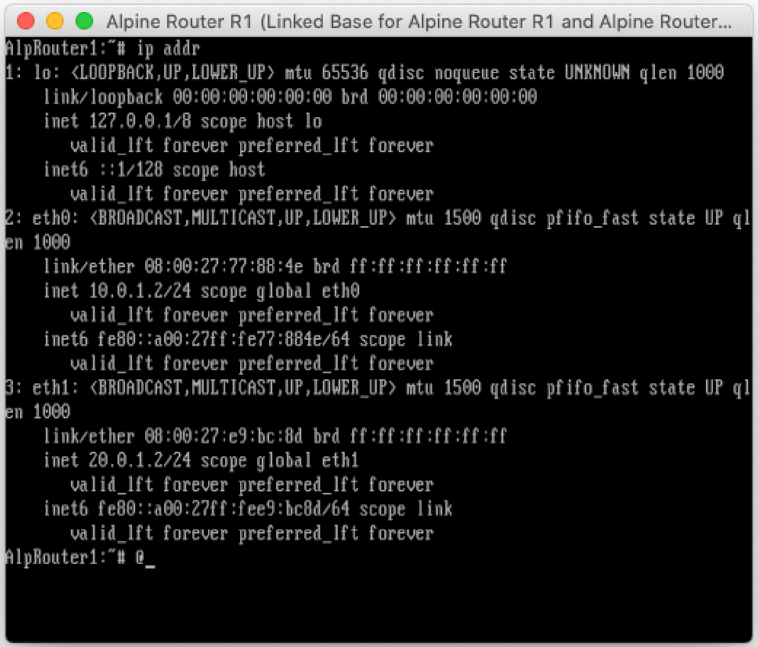
Attached to: Internal Network

Name: intnetA

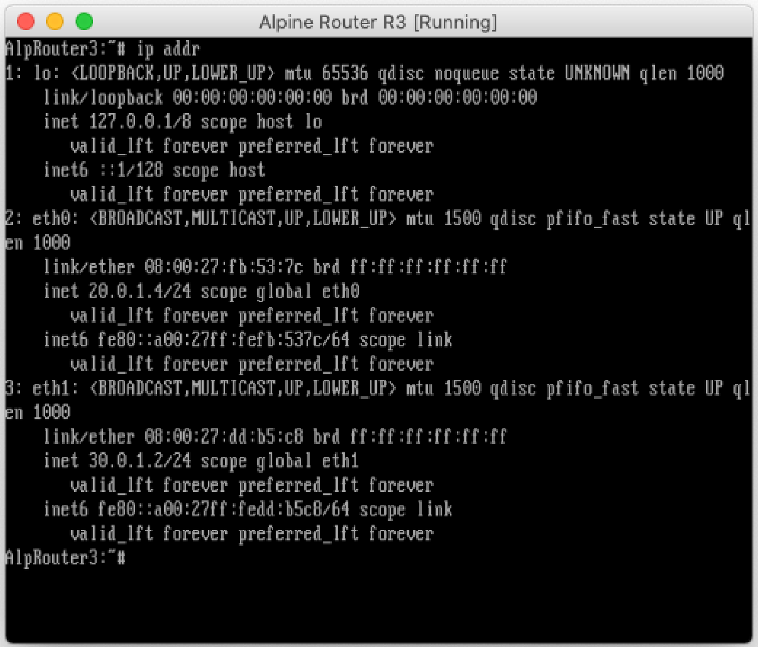
Advanced

Replicate this procedure for the other 3 Instances.

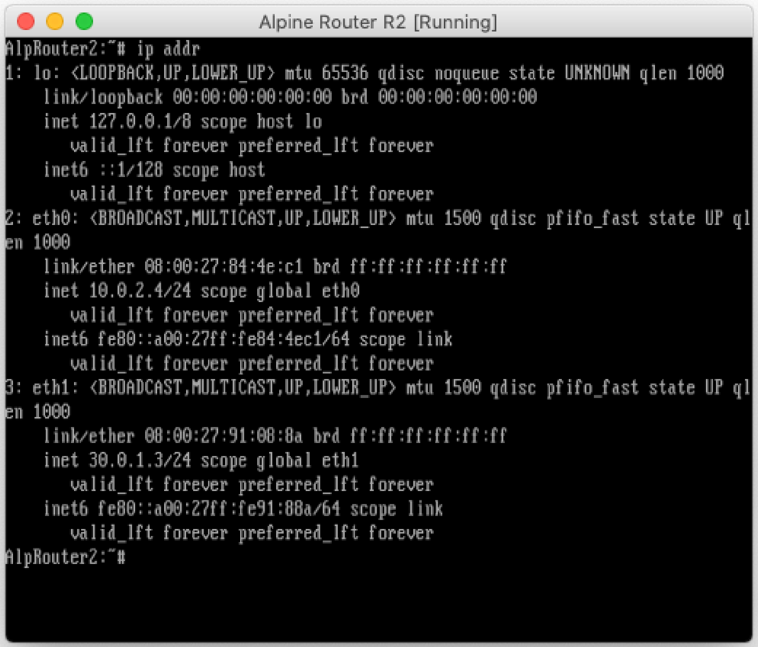
3. Boot the 3 Routers
   Check IP Addresses on the 3 Routers.


Alpine Router R1 (Linked Base for Alpine Router R1 and Alpine Router...

```
AlpRouter1:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN 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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:77:88:4e brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.2/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe77:884e/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:e9:bc:8d brd ff:ff:ff:ff:ff:ff
    inet 20.0.1.2/24 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fee9:bc8d/64 scope link
       valid_lft forever preferred_lft forever
AlpRouter1:~# @_
```



```
AlpRouter3:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN 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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:fb:53:7c brd ff:ff:ff:ff:ff:ff
    inet 20.0.1.4/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fefb:537c/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:dd:b5:c8 brd ff:ff:ff:ff:ff:ff
    inet 30.0.1.2/24 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fedd:b5c8/64 scope link
       valid_lft forever preferred_lft forever
AlpRouter3:~#
```



```
AlpRouter2:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN 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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:84:4e:c1 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.4/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe84:4ec1/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:91:08:8a brd ff:ff:ff:ff:ff:ff
    inet 30.0.1.3/24 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe91:88a/64 scope link
       valid_lft forever preferred_lft forever
AlpRouter2:~#
```

4. Establish Connectivity between R1 and R2 via R3
   In-order to reach R2 (WAN IP) from R1, on R1 we should establish routing rule such that all hosts on network C (intnetC : 30.0.1.0/24) is routed via R2

   on R1

   ```
   ip route add 30.0.1.0/24 via 20.0.1.4 dev eth1
   ```

   on R2

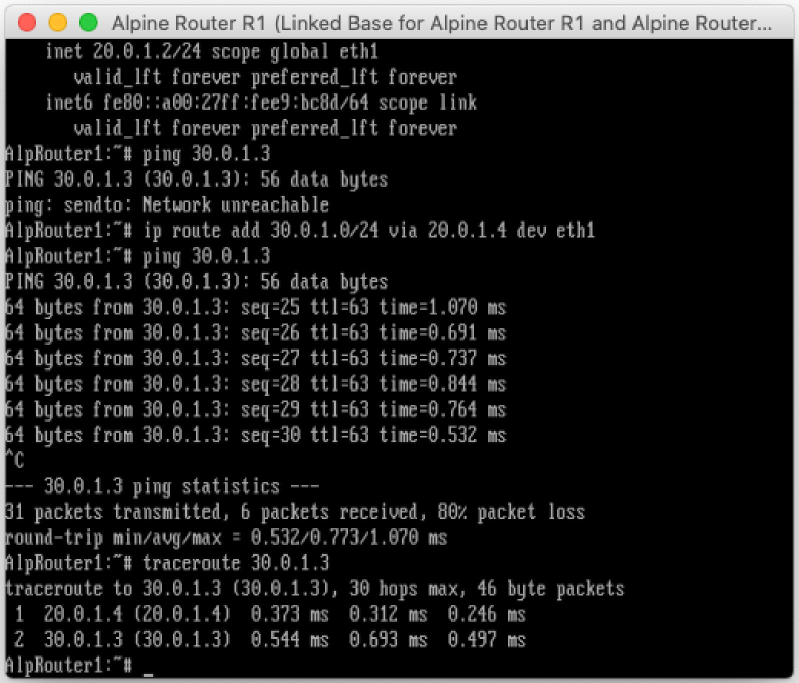
   ```
   ip route add 20.0.1.0/24 via 30.0.1.2 dev eth1
   ```

   on R3

   Enable ip packet forwarding from one Interface to the other if a route exists.

   ```
   echo 1 > /proc/sys/net/ipv4/ip_forward
   ```

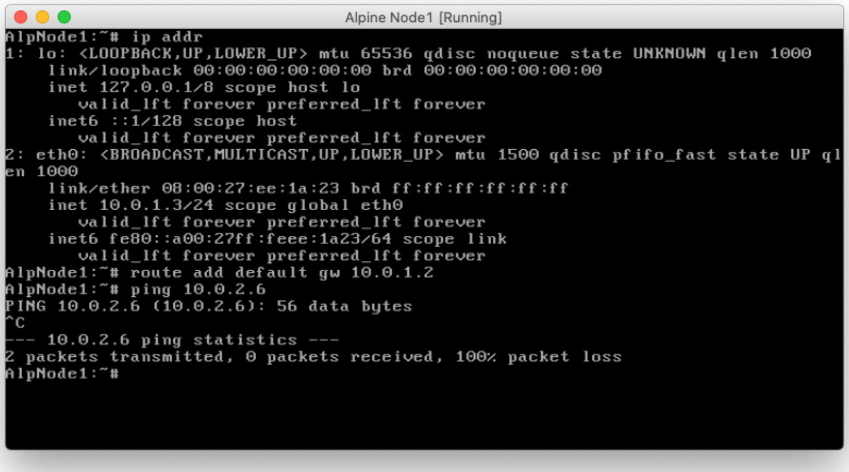
   Check Ping between R1 and R2


Alpine Router R1 (Linked Base for Alpine Router R1 and Alpine Router...

```
    inet 20.0.1.2/24 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fee9:bc8d/64 scope link
       valid_lft forever preferred_lft forever
AlpRouter1:~# ping 30.0.1.3
PING 30.0.1.3 (30.0.1.3): 56 data bytes
ping: sendto: Network unreachable
AlpRouter1:~# ip route add 30.0.1.0/24 via 20.0.1.4 dev eth1
AlpRouter1:~# ping 30.0.1.3
PING 30.0.1.3 (30.0.1.3): 56 data bytes
64 bytes from 30.0.1.3: seq=25 ttl=63 time=1.070 ms
64 bytes from 30.0.1.3: seq=26 ttl=63 time=0.691 ms
64 bytes from 30.0.1.3: seq=27 ttl=63 time=0.737 ms
64 bytes from 30.0.1.3: seq=28 ttl=63 time=0.844 ms
64 bytes from 30.0.1.3: seq=29 ttl=63 time=0.764 ms
64 bytes from 30.0.1.3: seq=30 ttl=63 time=0.532 ms
^C
--- 30.0.1.3 ping statistics ---
31 packets transmitted, 6 packets received, 80% packet loss
round-trip min/avg/max = 0.532/0.773/1.070 ms
AlpRouter1:~# traceroute 30.0.1.3
traceroute to 30.0.1.3 (30.0.1.3), 30 hops max, 46 byte packets
 1  20.0.1.4 (20.0.1.4)  0.373 ms  0.312 ms  0.246 ms
 2  30.0.1.3 (30.0.1.3)  0.544 ms  0.693 ms  0.497 ms
AlpRouter1:~# _
```

Ping should work and traceroute results will confirm the route taken via 20.0.1.4 which is the Router R3

5. Boot the Nodes on Network A and Network B
   Check ping from Node 1 to Node 2



```
AlpNode1:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN 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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:ee:1a:23 brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.3/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feee:1a23/64 scope link
       valid_lft forever preferred_lft forever
AlpNode1:~# route add default gw 10.0.1.2
AlpNode1:~# ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6): 56 data bytes
^C
--- 10.0.2.6 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
AlpNode1:~#
```

As expected we are Unable to ping Node2 from Node 1.

Configure Routing rule such that R1 is set as the default gateway on Node1.

```
route add default gw 10.0.1.2
```

Configure Routing rule such that R2 is set as the default gateway on Node2.

```
route add default gw 10.0.2.4
```

Also Enable Ip forwarding on R1 and R2

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

6. Now Lets Setup a GRE tunnel between R1 and R2

Commands For Establing GRE tunnel on Linux

on Router R1

```
ip tunnel add gre-R2 mode gre remote 30.0.1.3 local 20.0.1.2 ttl 255
ip addr add 10.0.100.1/24 dev gre-R2
ip link set gre-R2 up
ip route add 10.0.2.0/24 dev gre-R2
```

Lets explore the commands
In the First line we added a tunnel device, and called it gre-R2 (which is kind of obvious because that's where we want it to go).
Furthermore we told it to use the GRE protocol (mode gre), that the remote address is 30.0.1.3 (the router at the other end),
that our tunneling packets should originate from 20.0.1.2 and that the TTL field of the packet should be set to 255 (ttl 255).

In the second line we gave the newly born interface gre-R2 the address 10.0.100.1. This is the tunnel IP Address on this Router.We set the link status up.

In the fourth line we set the route for network B to be routed via the tunnel device gre-R2

on Router R2

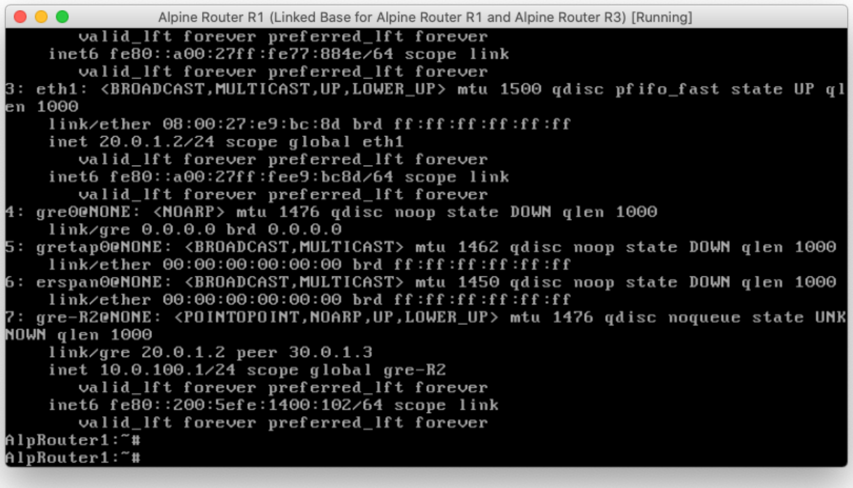
```
ip tunnel add gre-R1 mode gre remote 20.0.1.2 local 30.0.1.3 ttl 255
ip addr add 10.0.100.2/24 dev gre-R1
ip link set gre-R1 up
ip route add 10.0.1.0/24 dev gre-R1
```

For setting up a GRE tunnel on Linux you must have ip_gre module loaded in your kernel. To make sure it's loaded just do:

```
sudo modprobe ip_gre
lsmod | grep gre
```

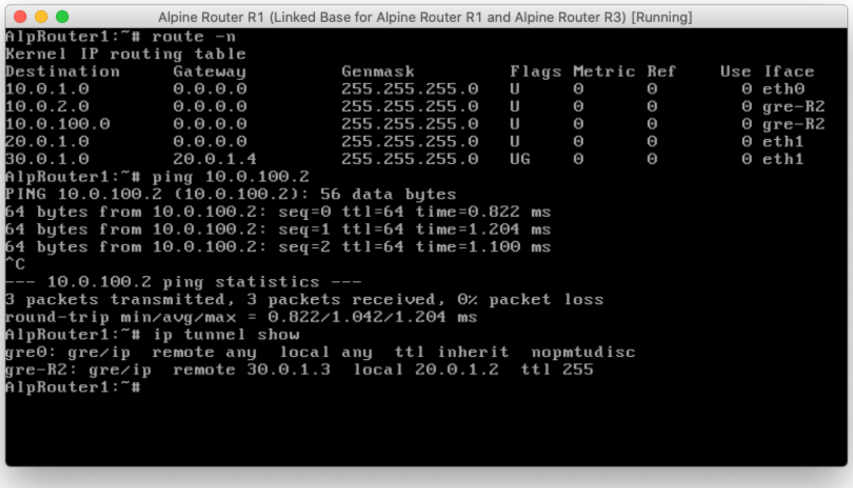
ip_gre ##### 0
gre ##### 1 ip_gre
If you see something else it's possible that your kernel does not support GRE.

When the gre module is loaded, the Linux kernel will create a default device, named gre0 and gretap0


Alpine Router R1 (Linked Base for Alpine Router R1 and Alpine Router R3) [Running]

```
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe77:884e/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:e9:bc:8d brd ff:ff:ff:ff:ff:ff
    inet 20.0.1.2/24 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fee9:bc8d/64 scope link
       valid_lft forever preferred_lft forever
4: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN qlen 1000
    link/gre 0.0.0.0 brd 0.0.0.0
5: gretap0@NONE: <BROADCAST,MULTICAST> mtu 1462 qdisc noop state DOWN qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
6: erspan0@NONE: <BROADCAST,MULTICAST> mtu 1450 qdisc noop state DOWN qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
7: gre-R2@NONE: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1476 qdisc noqueue state UNK
NOWN qlen 1000
    link/gre 20.0.1.2 peer 30.0.1.3
    inet 10.0.100.1/24 scope global gre-R2
       valid_lft forever preferred_lft forever
    inet6 fe80::200:5efe:1400:102/64 scope link
       valid_lft forever preferred_lft forever
AlpRouter1:~#
AlpRouter1:~#
```

Check Ping between the tunnel ip addresses


Alpine Router R1 (Linked Base for Alpine Router R1 and Alpine Router R3) [Running]

```
AlpRouter1:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 gre-R2
10.0.100.0      0.0.0.0         255.255.255.0   U     0      0        0 gre-R2
20.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 eth1
30.0.1.0        20.0.1.4        255.255.255.0   UG    0      0        0 eth1
AlpRouter1:~# ping 10.0.100.2
PING 10.0.100.2 (10.0.100.2): 56 data bytes
64 bytes from 10.0.100.2: seq=0 ttl=64 time=0.822 ms
64 bytes from 10.0.100.2: seq=1 ttl=64 time=1.204 ms
64 bytes from 10.0.100.2: seq=2 ttl=64 time=1.100 ms
^C
--- 10.0.100.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.822/1.042/1.204 ms
AlpRouter1:~# ip tunnel show
gre0: gre/ip  remote any  local any  ttl inherit  nopmtudisc
gre-R2: gre/ip  remote 30.0.1.3  local 20.0.1.2  ttl 255
AlpRouter1:~#
```

Tip:Command for GRETAP tunnel (not used in this setup)

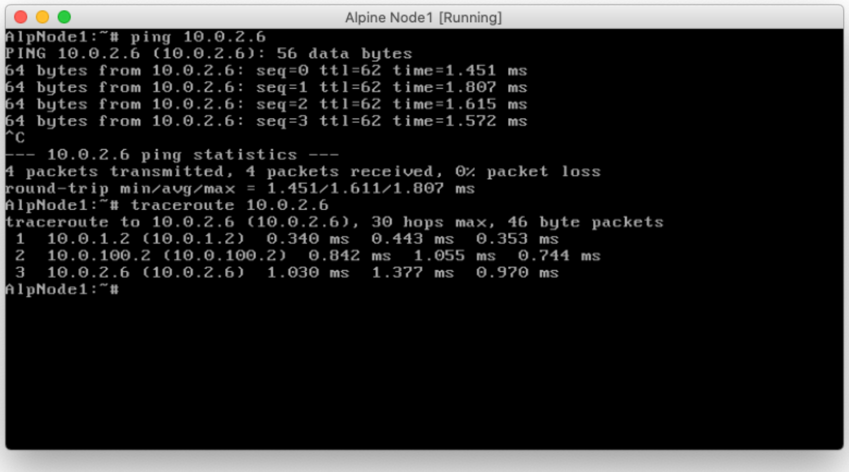
```
ip link add name gretap1 type gretap local LOCAL_IPv4_ADDR remote REMOTE_IPv4_ADDR
```

7. Check Connection between Node1 and Node2 over the GRE tunnel

With the Tunnel in place Lets send an ICMP echo request from Node1 to Node2 .

Observe the traceroute result.

Traceroute confirms that the route was established using the tunnel.



```
AlpNode1:~# ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6): 56 data bytes
64 bytes from 10.0.2.6: seq=0 ttl=62 time=1.451 ms
64 bytes from 10.0.2.6: seq=1 ttl=62 time=1.807 ms
64 bytes from 10.0.2.6: seq=2 ttl=62 time=1.615 ms
64 bytes from 10.0.2.6: seq=3 ttl=62 time=1.572 ms
^C
--- 10.0.2.6 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 1.451/1.611/1.807 ms
AlpNode1:~# traceroute 10.0.2.6
traceroute to 10.0.2.6 (10.0.2.6), 30 hops max, 46 byte packets
 1  10.0.1.2 (10.0.1.2)  0.340 ms  0.443 ms  0.353 ms
 2  10.0.100.2 (10.0.100.2)  0.842 ms  1.055 ms  0.744 ms
 3  10.0.2.6 (10.0.2.6)  1.030 ms  1.377 ms  0.970 ms
AlpNode1:~#
```

8. Commands to Disable the Tunnel

```
ip link set gre-R1 down
ip tunnel del gre-R1
```

9. **Yay, GRE Tunnel Rocks!!.**

GRE Tunnel in RDK-B Gateway Router

Usecase

- ### Wifi HotSpot

  1. **What is a Community Wi-Fi Service ?**

     Wi-Fi has become the access network of choice for both Communications Service Providers and consumers alike.
     Consumer prefer and demand 24/7 Internet access everywhere.

     Many CSPs across the world are deploying public Wi-Fi hotspots to meet the demands of their customers. Yet while deploying dedicated public Wi-Fi hotspots is a great way to provide Wi-Fi services to customers in areas such as malls, stadiums, and parks, they may not be the most economical tool to provide similar services in residential neighborhoods and sparsely populated areas. CSPs are complementing their public Wi-Fi hotspot deployments by using existing Wi-Fi assets, for instance residential and SMB Wi-Fi gateways, to provide public Wi-Fi services to customers in residential neighborhoods, as the so-called Community Wi-Fi.

     This allows CSPs to use an existing residential broadband connection as backhaul for both public and private Wi-Fi services.
     Private Wi-Fi is for exclusive use by the ‘broadband customer’ who is paying for broadband services to their home.

     The public Wi-Fi is a resource for the operator to provide services to other customers.
     The residential subscribers accessing the network from inside their homes have prioritized access to Wi-Fi and backhaul resources.
     Roaming and on-the-go subscribers are only allowed to use Wi-Fi network capacity that is not currently used by the subscriber at home.

     The residential Wi-Fi infrastructure is configured in a manner that allows both residential and visitor subscribers to access Wi-Fi resources simultaneously in a manner that affords security, privacy, and service quality for both In order to offer Community Wi-Fi services, an operator enables a residential Wi-Fi gateway with two Wi-Fi SSIDs, consisting of a private SSID (i.e., the residential Wi-Fi network) and public SSID (i.e., the Community Wi-Fi network). Where “user” is defined as the end service subscriber, and “operator” is defined as the CSP

     Members of the public who have an account with the same Internet service provider as the subscriber has at home can access the Internet and mobile network through the public part of the subscriber’s Wi-Fi connection when they are in close proximity to the subscriber’s home.

     Why use a Community Wi-Fi Service?

Allows connectivity to the users of a Service provider even if not in the range of their home device WiFi by using Wireless Network from a third persons device belonging to the same service provider.
Homeowner gets to keep their Wi-Fi password secret.
Traffic seperation between private and public wifi networks.
Subscriber services such as authentication, authorization, and accounting (AAA); address assignment; hierarchical quality of service (QoS); lawful intercept; and class of service (CoS) are supported

Example

Comcast Xfinity Wifi Hotspot is one such Community Wi-Fi solution
Xfinity-Wi-Fi is a feature through which customer can connect to internet (not Home Device) using their Comcast account from anywhere. For example, suppose we are in a restaurant which has Xfinity with hotspot enabled (by default all Xfinity Boxes have hotspot enabled), that Restaurant Xfinity Box will broadcast Xfinity Wi-Fi SSID. We can connect your Mobile/Device to that Xfinity SSID but once you connect it will ask your Comcast account details and verify the same. Once verified, we can browse the internet with your account.

2. Network Architecture Overview

In general, the Community Wi-Fi infrastructure model consists of split public/private wireless access, a tunnel between the public SSID and the CSP core, and the associated AAA related services.
Radio resource management (RRM) is also highly desirable, especially in dense residential or retail deployments where significant co-channel and adjacent-channel interference is prevalent.
Managing the co-existence of the public and private networks is one of the primary technical challenges in Community Wi-Fi networks.

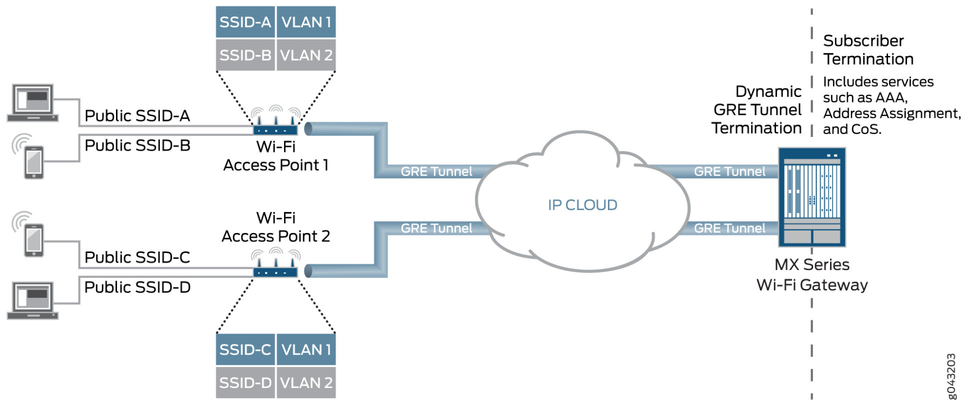
Traffic Separation and Forwarding

Traffic separation between private and public networks and traffic forwarding are crucial requirements to be considered when designing a Community Wi-Fi network.

Main common requirements regarding this topic are:
a. Traffic separation between private and public networks and among users within the public network
b. A user connected to their private SSID will not have any visibility or access to public SSID traffic
c. A user connected to the public SSID will not have any visibility or access to private SSID traffic or any connected devices (e.g., network printer, disk, etc.)
d. A user connected to the public SSID will not have visibility or access to the public SSIDs other users' traffic
e. The system should allow for user identification and traffic differentiation from one user to another in the public network (e.g., for lawful interception purposes).
f. Private and public traffic will be separated in the backhaul
g. Downstream traffic will not be flooded on both public and private networks

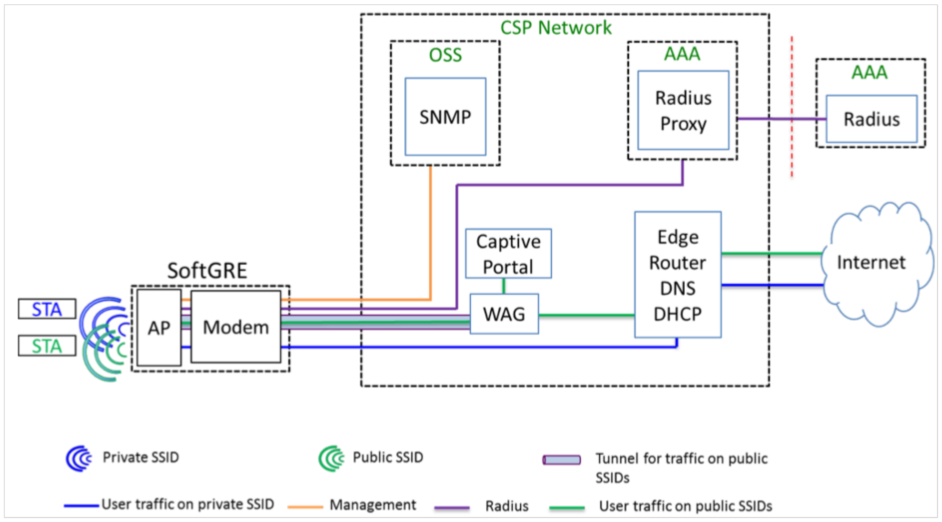
GRE is used for wired traffic segregation and management,which offers a secure tunnel back to the CSP core.

Usage of GRE Tunnel



Layer 2 Wi-Fi roaming traffic is encapsulated between the Wi-Fi GW and WAG within a GRE tunnel.
A single GRE tunnel is used per public SSID. Traffic separation of roaming traffic from the residential subscriber traffic is provided by the GRE encapsulation.
The GRE tunnel can be assigned service profile-based QoS on the backhaul network, using IP classifiers,
in order to establish a separate traffic priority for the public SSID. The controller provides a RADIUS client interface to the AAA and generates usage based accounting.
Stateless GRE is proposed for less overhead on the AP and better scaling.
Tunnel and subscriber context are auto-created such that the GRE control plane signaling is avoided.
This helps the proposed architecture to scale and reduces the amount of GRE provisioning needed.
The tunnel context only exists if one or more roaming Wi-Fi subscriber is admitted to the network, further helping the proposed architecture to scale.

When the user equipment device connects to the SSID and begins to send traffic, the access point initiates a Layer 2 soft GRE or Ethernet-over-GRE connection to the Wi-FI Access Gateway and dynamically builds the GRE tunnel. GRE tunnels are cleared after all of the subscribers within a GRE tunnel have logged out and a configurable timer has expired.
Using GRE tunnels for Wi-Fi provides the following benefits:
• Wi-Fi users who are not directly connected through Layer 2 to Wi-FI Access Gateway (WAG) are authenticated because GRE tunnels transmit Layer 2 information across any IP network.
• Services based on user equipment-specific information are applied using the media access control (MAC) address.
• Services are applied in the network, not just at the Wi-Fi access point.
• The soft GRE or Ethernet-over-GRE standard is supported on most Wi-Fi access points. For services using the Ethernet over GRE standard, only one side of the tunnel needs to be configured;
the other end learns the remote IP addresses of all remote tunnel endpoints by examining the incoming GRE packets.



3. **Comparison of Public-SSID vs Private SSID**

	Private SSID	**Public SSID**
Purpose	Private SSID on the residential gateway is for exclusive use by the “broadband customer” paying for broadband services for that residence	Public SSID on the residential gateway is for the operator to provide services to other customers
SSID Configuration	Preconfigured by operator and/or configured by customer	Operator managed/configured SSID
Admission Control	Residential customer controls access on private SSID	Operator controls access on public SSID
Traffic Separation	Customers on the public SSID on the same residential GW are not allowed to communicate directly with the devices on the private SSID	Two users on the public SSID are not allowed to communicate with each other directly. Additionally, users on the public SSID cannot directly communicate with the devices on the private SSID
Security	Customer may or may not enable authentication /encryption	All users of the Community Wi-Fi services shall use network authentications such as WPA2-PSK, WPA2-ENTERPRISE, or web authentication (assuming a secured Wi-Fi interface)
Services	Operator uses the private SSID to offer services (e.g., voice, video) to the broadband customer	Operators may offer different services to different customers on the public SSID
QoS on the Backhaul network	Since both private and public SSID are hosted on a single residential gateway, the same physical network is used to backhaul traffic on both private and public SSID. Different treatment (e.g., QoS, forwarding) of traffic from public and private SSID	Since both private and public SSID are hosted on a single residential gateway, the same physical network is used to backhaul traffic on both private and public SSID. Different treatment (e.g., QoS, forwarding) of traffic from public and private SSID

Accounting	No need for per user accounting on private SSID	Per user accounting is a requirement
Protection of traffic on private SSID	Different priority to traffic on the private SSID than the traffic on public SSID	Different Quality of Service (QoS) profiles to different customers or groups of customers on public SSID

4. Public-SSID/Wi-Fi Hotspot Requirements for Cable Modem Gateways (CableLabs)

The Wi-Fi GW MUST support the configuration of the operator-controlled SSIDs and the associated attributes via remote access by the operator.
The Wi-Fi GW MUST NOT allow the operator-controlled SSIDs and the associated attributes to be configured or changed by the user.
Operators can select to organize traffic from each SSID in separate Virtual Local Area Networks (VLANs) to assist in traffic priority settings and to help ensure traffic separation and traffic forwarding.
The Wi-Fi GW MUST prevent routing between private and public SSID VLANs.
When VLANs are supported, the Wi-Fi GW MUST put the clients connected to each SSID in a separate VLAN and route the traffic from each VLAN transparently to the upstream network.

GRE is a capability for the Wi-Fi GW that operators can use to segregate and forward user traffic per SSID.
A separate GRE tunnel can be established for each SSID, or traffic associated with multiple SSIDs can be combined in a single GRE tunnel. In this case, VLAN tags (802.1Q or 802.1ad) are used to separate traffic between SSIDs within the GRE tunnel. The use of GRE is transparent to the CMTS with this approach.
DSCP markings can be added by the Wi-Fi GW to provide differentiation of traffic from separate GRE tunnels as well as unique traffic flows (e.g., per SSID) within a given GRE tunnel.

GRE Requirements for the public SSID

The Wi-Fi GW MUST support the mapping of an IP GRE tunnel to a single SSID or to multiple SSIDs. The mapping supports bi-directional traffic.
The destination IP address in the transport IP header MUST be set to the IP address of the GRE tunnel endpoint in the network north of the Wi-Fi GW.
The source IP address in transport IP header MUST be set to the WAN IP address (Public IP) of Wi-Fi GW
The protocol type in transport IP header MUST be set to "GRE" (47)
The protocol type in GRE header MUST be set to "Transparent LAN Bridging" (0x6558).
The Wi-Fi GW MAY insert 802.1q or 802.1ad VLAN tag corresponding to the SSID in the encapsulated (i.e., inner) 802.3 frame.
The 802.1q or 802.1ad VLAN tag MAY be used to identify the SSID(s).
The Wi-Fi GW MUST prevent user-to-user switching of frames within same SSIDs or across SSID. All traffic (including broadcast and multicast packets but excluding EAP messages for 802.1x authentication) coming in on 802.11 WLAN interfaces must be tunneled to the GRE tunnel endpoint in the network north of the Wi-Fi GW.
The Wi-Fi GW must support applying DSCP value in the transport IP header for upstream packets. This is used to classify traffic from SSIDs into different DOCSIS service flows, based on the DSCP marking on the GRE tunnel packets. The set of DSCP markings used to map to a GRE tunnel is static.

The Wi-Fi GW MUST rewrite MSS option to or less than 1390B in both TCP SYN and TCP SYN-ACK packets.
The Wi-Fi GW MUST implement DHCP relay agent in order to insert sub-options in option-82 of DHCP messages received from the client device connected to the public SSID.
The Wi-Fi GW MUST support inserting information about the AP and the SSID (e.g., AP-MAC-Address, SSID- Name, and SSID-Type i.e., "open" or "secure") on which the DHCP message is received from the client device connected to the public SSID in the circuit-ID sub-option in DHCP relay agent option option-82. Insertion of SSID in the circuit-ID sub-option MUST be configurable.
The Wi-Fi GW MUST support inserting the MAC address of client device connected to a public SSID remote- id sub-option of DHCP option-82. Insertion of MAC address of client device in remote-id sub-option MUST be configurable.
Remote-id MUST be a string containing MAC address of client device connected to the public in the format "xx:xx:xx:xx:xx:xx"

5. Comcast Xfinity Case Study

Comcast enables its customers to get Internet access outside the home using a Community Wi-Fi network (xfinitywifi) deployed in Comcast markets.

Comcast Community Wi-Fi Devices

Comcast is able to take advantage of the existing deployed CPE that currently provides high speed Internet and other services (including private Wi-Fi, VoIP, Home Security) to its customers.
The CPE is integrated into units consisting of cable modem, eMTA, eRouter, and 802.11n radio. The CPE is delivered across multiple vendors, models, and firmware variants.
By applying a software upgrade to the CPE, Comcast is able to additionally provide a Community Wi-Fi public hotspot at that location.

Comcast Community Wi-Fi Network Core

Comcast's Community Wi-Fi CPE uses SoftGRE tunnels to bridge the xfinitywifi traffic over the Comcast Hybrid Fiber-Coaxial (HFC) plant to the WLAN gateway core.
In order to minimize traffic latency and core network load, the WLAN gateways are located on the natural egress path of the traffic to the Internet.
Utilizing the WLAN gateway core allows Comcast to keep the Community Wi-Fi CPE 'feature light' while providing functionality (such as captive portal, authentication, accounting, lawful intercept, parental control, watermarking, QoS,) through a powerful WLAN gateway core.

Tunnel Architecture

Maintaining stateful Layer 2 tunnels between the AP and the WLAN gateway is a common limiting factor of the WLAN gateway capacity.
Comcast chose to utilize SoftGRE tunnels as a tunneling protocol for the Community Wi-Fi solution for the following reasons:

a. GRE is a very lightweight encapsulation
b. Only the Community Wi-Fi CPE end of the tunnel needs to be configured
c. No WLAN gateway heartbeat is needed to maintain the tunnel
d. The tunnel is only established when traffic is being passed

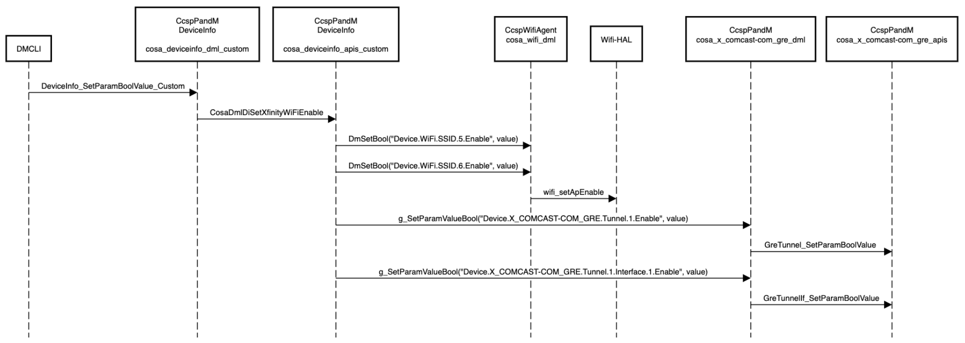
WLAN gateway's maximum tunnel capacity is aligned to the number of Community Wi-Fi CPE in use, rather than the number of Community Wi-Fi CPE configured
Comcast segregates public and private traffic by VLAN at the eRouter and by DOCSIS service flow at the cable modem.

- **Wifi Extender**

RDK-B Implementation: GRE tunnel

- **XFINITY WIFI Hotspot**

DeviceInfo Data model Object "X_COMCAST_COM_xfinitywifiEnable" can be used to enable the XFINITY wifi hotspot functionality


DMCLI
CcspPandM DeviceInfo cosa_deviceinfo_dml_custom
CcspPandM DeviceInfo cosa_deviceinfo_apis_custom
CcspWifiAgent cosa_wifi_dml
Wifi-HAL
CcspPandM cosa_x_comcast-com_gre_dml
CcspPandM cosa_x_comcast-com_gre_apis
DeviceInfo_SetParamBoolValue_Custom
CosaDmlDiSetXfinityWiFiEnable
DmSetBool("Device.WiFi.SSID.5.Enable", value)
DmSetBool("Device.WiFi.SSID.6.Enable", value)
wifi_setApEnable
g_SetParamValueBool("Device.X_COMCAST-COM_GRE.Tunnel.1.Enable", value)
GreTunnel_SetParamBoolValue
g_SetParamValueBool("Device.X_COMCAST-COM_GRE.Tunnel.1.Interface.1.Enable", value)
GreTunnelIf_SetParamBoolValue

X_COMCAST-COM_GRE Object needs to be committed with appropriate parameters such as PrimaryEndpoint(WAG) to which the tunnel should be connected prior to using X_COMCAST_COM_xfinitywifiEnable object.
CcspHotSpot Daemon runs a thread which pings and checks the connectivity and sends KeepAlive messages to the PrimaryEndPoint and can also switch over to SecondaryEndpoint ,if the primary is not reachable.
Daemon is listening to sysevent notifications for any changes to value of PrimaryEndPoint set via Datamodel object.

- **RDK Emulator**

On RDK-B Emulator the systemD service unit for CcspHotSpot Module starts the following.

CcspHotSpot.service

```
[Unit]
Description=CcspHotSpot
After=network-online.target network-setup.service hostapd.service


[Service]
ExecStart=/bin/sh -c '/lib/rdk/hotspot_emu.sh;'
Type=forking
Restart=always

[Install]
WantedBy=multi-user.target
```

This service starts the helper script (lib/rdk/hotspot_emu.sh) for creating GRE Tunnel endpoints for Xfinity WIFI Hotspot.

```
HOTSPOT_ENABLE=`dmcli simu getv Device.DeviceInfo.X_COMCAST_COM_xfinitywifiEnable | grep value | cut -
f3 -d : | cut -f2 -d" "`
/lib/rdk/handle_emu_gre.sh create
echo "CCSP-HOTSPOT IS SUCCESSFULLY RUNNING"
```

The script /lib/rdk/handle_emu_gre.sh does the following:
Reads current values of GRE tunnel data model from PSM using psmcli.
Launches the CcspHotSpot Daemon and creates the GRE tunnel using ip commands.

```
LOCAL_IP="`ifconfig -a eth0 | grep inet | grep -v inet6 | tr -s " " | cut -d ":" -f2 | cut -d " " -
f1`"
ip link add gretap0 type gretap remote `sysevent get hotspotfd-tunnelEP` local $LOCAL_IP  ttl 255 dev
$WAN_IF
ifconfig gretap0 up
```

- ## Datamodel

This Data model is implemented by CCSP P-and-M Layer.

X_COMCAST-COM_GRE ((Simplified))

```
        <object>
        <name>X_COMCAST-COM_GRE</name>
        <objectType>object</objectType>
        <objects>
            <object>
                <name>Tunnel</name>
                <objectType>staticTable</objectType>
                <maxInstance>4</maxInstance>
                <functions>
                    <func_GetEntryCount>GreTunnel_GetEntryCount</func_GetEntryCount>
                    <func_GetEntry>GreTunnel_GetEntry</func_GetEntry>
                    <func_GetParamBoolValue>GreTunnel_GetParamBoolValue</func_GetParamBoolValue>
                    <func_GetParamUlongValue>GreTunnel_GetParamUlongValue</func_GetParamUlongValue>
                    <func_GetParamStringValue>GreTunnel_GetParamStringValue</func_GetParamStringValue>
                    <func_GetParamIntValue>GreTunnel_GetParamIntValue</func_GetParamIntValue>
                    <func_SetParamBoolValue>GreTunnel_SetParamBoolValue</func_SetParamBoolValue>
                    <func_SetParamUlongValue>GreTunnel_SetParamUlongValue</func_SetParamUlongValue>
                    <func_SetParamStringValue>GreTunnel_SetParamStringValue</func_SetParamStringValue>
                    <func_SetParamIntValue>GreTunnel_SetParamIntValue</func_SetParamIntValue>
                    <func_Validate>GreTunnel_Validate</func_Validate>
                    <func_Commit>GreTunnel_Commit</func_Commit>
                    <func_Rollback>GreTunnel_Rollback</func_Rollback>
                </functions>
                <parameters>
                        <name>Enable</name>
                        <name>Status</name>
                        <name>LastChange</name>
                        <name>PrimaryRemoteEndpoint</name>
                        <name>SecondaryRemoteEndpoint</name>
                        <name>ConnectedRemoteEndpoint</name>
                        <name>KeyIdentifierGenerationPolicy</name>
                        <name>KeyIdentifier</name>
                        <name>UseSequenceNumber</name>
                        <name>UseChecksum</name>
                        <name>DSCPMarkPolicy</name>
                        <name>KeepAlivePolicy</name>
                        <name>RemoteEndpointHealthCheckPingCount</name>
                        <name>RemoteEndpointHealthCheckPingInterval</name>
                        <name>RemoteEndpointHealthCheckPingFailThreshold</name>
                        <name>RemoteEndpointHealthCheckPingIntervalInFailure</name>
                        <name>ReconnectToPrimaryRemoteEndpoint</name>
                        <name>EnableCircuitID</name>
                        <name>EnableRemoteID</name>
                        <name>GRENetworkTunnel</name>
                                                <name>InterfaceNumberOfEntries</name>
                                </parameters>
            </object>
            <object>
                <name>Interface</name>
                                <objectType>staticTable</objectType>
                                <maxInstance>4</maxInstance>
                <functions>
                    <func_GetEntryCount>GreTunnelIf_GetEntryCount</func_GetEntryCount>
                                        <func_GetEntry>GreTunnelIf_GetEntry</func_GetEntry>
                                        <func_GetParamBoolValue>GreTunnelIf_GetParamBoolValue<
/func_GetParamBoolValue>
                                        <func_GetParamUlongValue>GreTunnelIf_GetParamUlongValue<
/func_GetParamUlongValue>
                                        <func_GetParamStringValue>GreTunnelIf_GetParamStringValue<
/func_GetParamStringValue>
                                        <func_GetParamIntValue>GreTunnelIF_GetParamIntValue<
/func_GetParamIntValue>
                                        <func_SetParamBoolValue>GreTunnelIf_SetParamBoolValue<
/func_SetParamBoolValue>
                                        <func_SetParamUlongValue>GreTunnelIf_SetParamUlongValue<
/func_SetParamUlongValue>
                                        <func_SetParamStringValue>GreTunnelIf_SetParamStringValue<
/func_SetParamStringValue>
                                        <func_SetParamIntValue>GreTunnelIf_SetParamIntValue<
/func_SetParamIntValue>
                                        <func_Validate>GreTunnelIf_Validate</func_Validate>
                                        <func_Commit>GreTunnelIf_Commit</func_Commit>
                                        <func_Rollback>GreTunnelIf_Rollback</func_Rollback>
                </functions>
                <parameters>
                                        <name>Enable</name>
                                        <name>Status</name>
                                        <name>LastChange</name>
                                        <name>LocalInterfaces</name>
                                        <name>VLANID</name>
                                        <name>AssociatedBridges</name>
                                        <name>AssociatedBridgesWiFiPort</name>
                                </parameters>
                        </object>
                        <object>
                                <name>SSID</name>
                                <objectType>dynamicTable</objectType>
                                <maxInstance>128</maxInstance>
                                <functions>
                                        <func_GetEntryCount>HsSsid_GetEntryCount</func_GetEntryCount>
                                        <func_GetEntry>HsSsid_GetEntry</func_GetEntry>
                                        <func_GetParamStringValue>HsSsid_GetParamStringValue<
/func_GetParamStringValue>
                                </functions>
                                <parameters>
                                    <name>SSID</name>
                                </parameters>
                                <objects>
                                        <object>
```

```
                                                <name>AssociatedDevice</name>
                                                <objectType>dynamicTable</objectType>
                                                <maxInstance>128</maxInstance>
                                                <functions>
                                                        <func_GetEntryCount>HsAssoDev_GetEntryCount<
/func_GetEntryCount>
                                                        <func_GetEntry>HsAssoDev_GetEntry<
/func_GetEntry>
                                                        <func_IsUpdated>HsAssoDev_IsUpdated<
/func_IsUpdated>
                                                        <func_Synchronize>HsAssoDev_Synchronize<
/func_Synchronize>

<func_GetParamStringValue>HsAssoDev_GetParamStringValue</func_GetParamStringValue>

<func_GetParamIntValue>HsAssoDev_GetParamIntValue</func_GetParamIntValue>
                                                </functions>
                                                <parameters>
                                                        <name>MACAddress</name>
                                                        <name>Hostname</name>
                                                        <name>RSSILevel</name>
                                                        <name>IPv4Address</name>
                                                        <name>DHCPv4Status</name>
                                                        <name>IPv6Address</name>
                                                        <name>IPv6Prefix</name>
                                                        <name>DHCPv6Status</name>
                                                        <name>IPv6LinkLocalAddress</name>
                                                </parameters>
                                        </object>
                                </objects>
                        </object>
                </objects>
        </object>
```

How Tunnel Security is added?

While GRE provides a stateless, private connection, it is not considered a secure protocol because it does not use encryption .

Creating a point-to-point GRE tunnel without any encryption is extremely risky as sensitive data can easily be extracted from the tunnel and viewed by others.

If data protection is required, IPSec must be configured to provide data confidentiality – this is when a GRE tunnel is transformed into a secure VPN GRE tunnel.

While many might think a GRE IPSec tunnel between two routers is similar to a site to site IPSec VPN (crypto), it is not. A major difference is that GRE tunnels allow multicast packets to traverse the tunnel whereas IPSec VPN does not support multicast packets. In large networks where routing protocols such as OSPF, EIGRP are necessary,GRE tunnels are your best bet. For this reason, plus the fact that GRE tunnels are much easier to configure, engineers prefer to use GRE rather than IPSec VPN.

IPSEC

Internet Protocol is not secure .There are no mechanisms for performing data integrity checks or to enforce confidentiality of data.

IPSEC is a set of protocols and algorithms used to secure IP data at network layer. Its a Layer 3 protocol.

IPSec's objective is to provide the following security services for IP packets

- Confidentiality & Integrity by encrypting sensitive data,
- Authentication by using signatures and certificates
- Protection against replay attacks using Sequence number checks
- Key Management using IKE protocol

IPSec can be used in conjunction with GRE to provide top-notch security encryption for our data, thereby providing a complete secure and flexible VPN solution.

Operating Modes

IPSec can operate in two different modes, Tunnel mode and Transport mode.

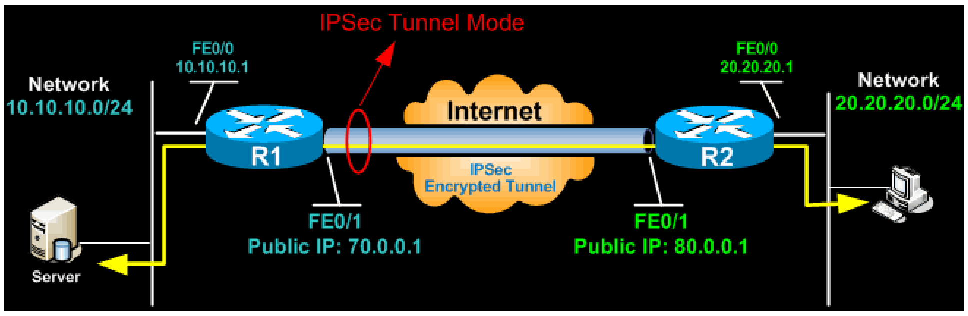
1. **Tunnel Mode**
   Tunnel Mode is used in Site to Site VPN.
   Original IP Packet is encrypted and becomes the data payload for a new IP Packet.

   IPSec tunnel mode is the default mode. With tunnel mode, the entire original IP packet is protected by IPSec.
   This means IPSec wraps the original packet, encrypts it, adds a new IP header and sends it to the other side of the VPN tunnel (IPSec peer).

   Tunnel mode is most commonly used between gateways or at an end-station to a gateway, the gateway acting as a proxy for the hosts behind it.

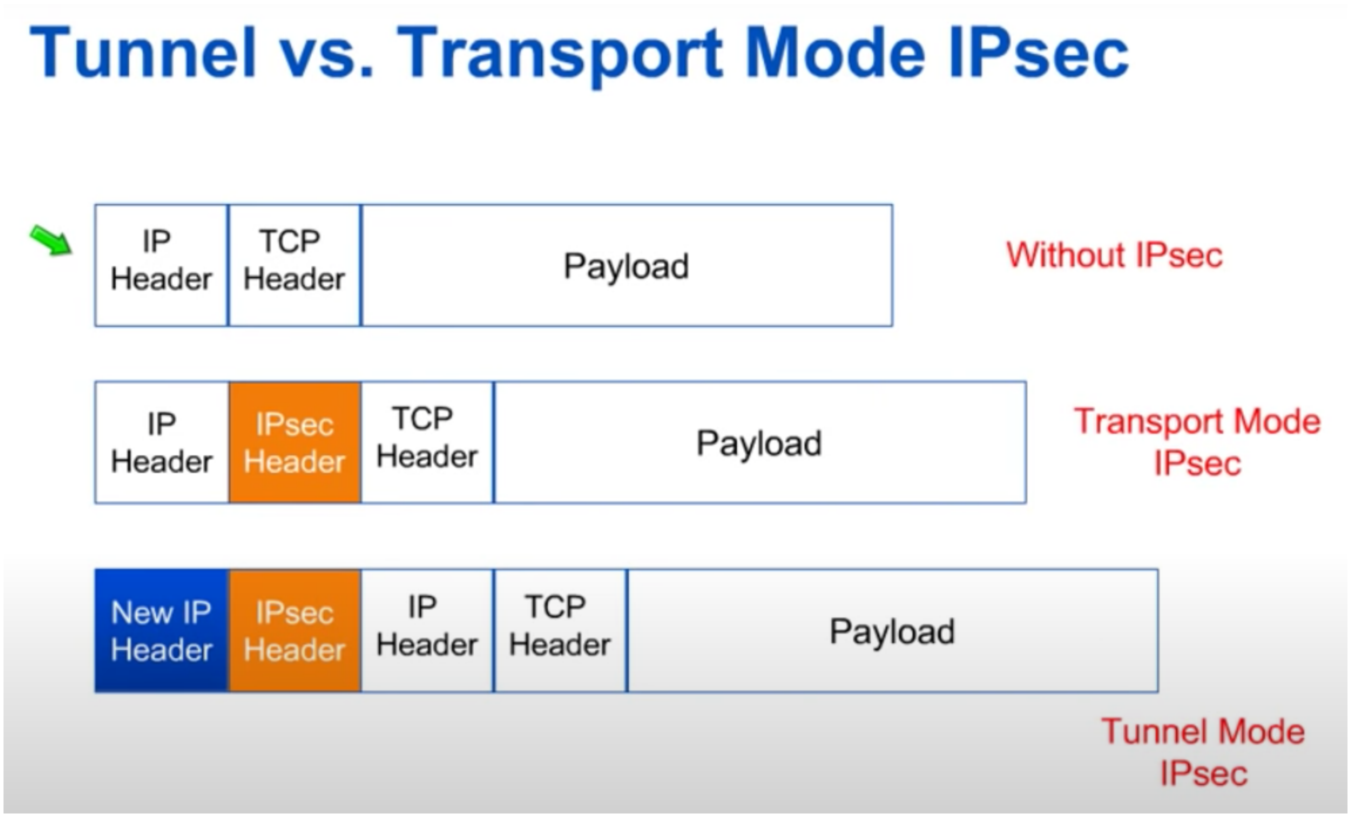
   The client connects to the IPSec Gateway. Traffic from the client is encrypted, encapsulated inside a new IP packet and sent to the other end.
   Once decrypted by the firewall appliance, the client's original IP packet is sent to the local network.

   In tunnel mode, an IPSec header (AH or ESP header) is inserted between the IP header and the upper layer protocol.
   Between AH and ESP, ESP is most commonly used in IPSec VPN Tunnel configuration.



2. **Transport Mode**
   Transport Mode is used in Remote Access VPNs.
   IPSEC header is inserted into the existing IP Packet. No new IP Packet is created.
   IPSec Transport mode is used for end-to-end communications, for example, for communication between a client and a server or between a workstation and a gateway (if the gateway is being treated as a host). A good example would be an encrypted Telnet or Remote Desktop session from a workstation to a server.

<u>**IP Packet Format with IPSec**</u>



IPSec Architecture

Security Protocols

1. **Authentication Header (AH)**

   AH Protocol provides authentication and data integrity for the packet.
   However no encryption is done.
   Spoofing and replay attacks are prevented by using AH.
   Only used when data is not confidential but the source needs to be verified.
   AH has been deprecated in favour of ESP

   The Authentication Header information is added into the packet which is generated by the sender, right between the Network (Layer 3) and Transport (Layer 4) Layer (see picture below).

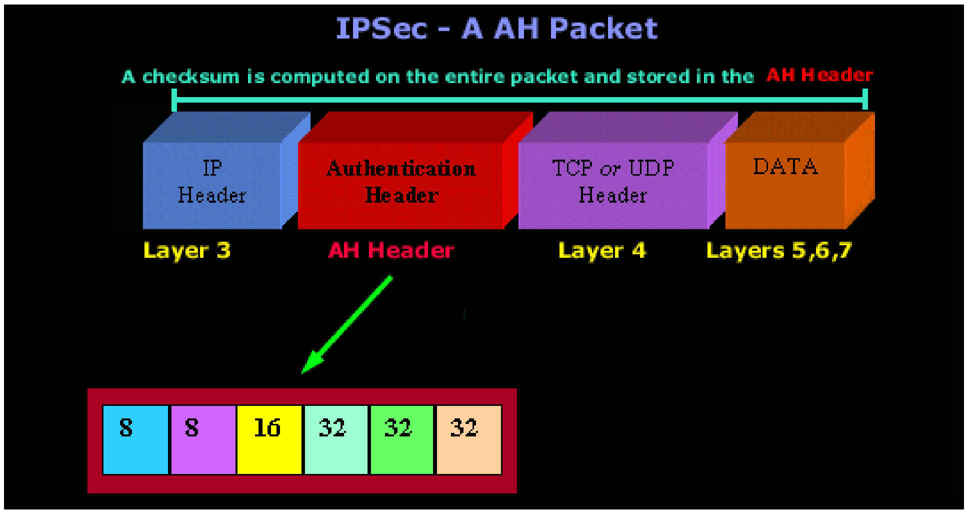
   Authentication protects your network, and the data it carries, from tampering. Tampering might be a hacker sitting between the client and server, altering the contents of the packets sent between the client and server, or someone trying to impersonate either the client or server, thus fooling the other side and gaining access to sensitive data.

   To overcome this problem, IPSec uses an *Authentication Header* (AH) to digitally sign the entire contents of each packet. This signature provides 3 benefits:

   1) Protects against replay attacks. If an attacker can capture packets, save them and modify them, and then send them to the destination, then they can impersonate a machine when that machine is not on the network. This is what we call a replay attack. IPSec will prevent this from happening by including the sender's signature on all packets.

   2) Protection against tampering. The signatures added to each packet by IPSec means that you can't alter any part of a packet undetected.

   3) Protection against spoofing. Each end of a connection (e.g client-server) verifies the other's identity with the authentication headers used by IPSec.



2. **Encapsulating Security Payload (ESP)**
   Provides Authentication ,Integrity and Confidentiality.
   Uses symmetric key Encryption

   ESP is used to encrypt the entire payload of an **IPSec** packet (Payload is the portion of the packet which contains the upper layer data).

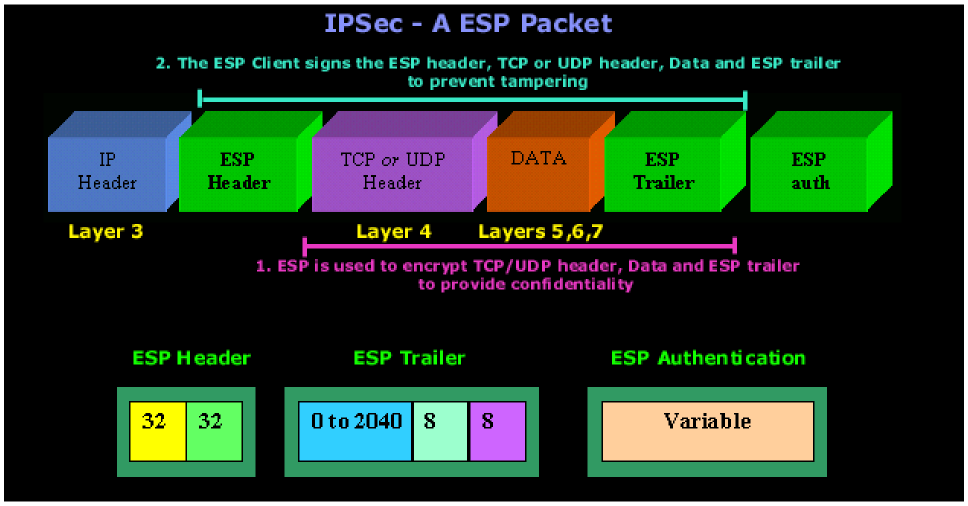
   ESP is a bit more complex than AH because *alone* it can provide authentication, replay-proofing and integrity checking. It acomplishes this by adding 3 separate components:

   1) An ESP header

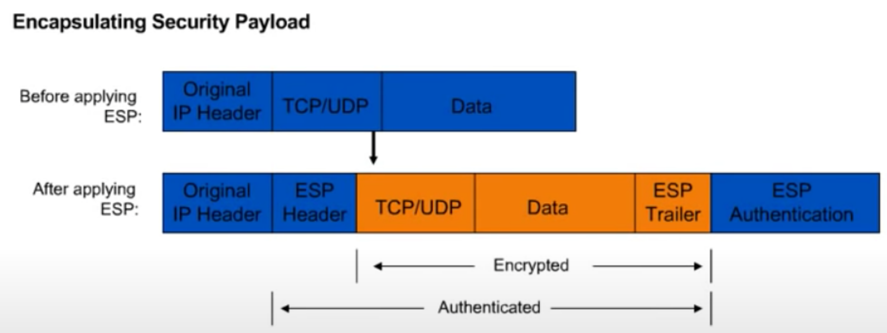
   2) An ESP trailer and

   3) An ESP authentication block.

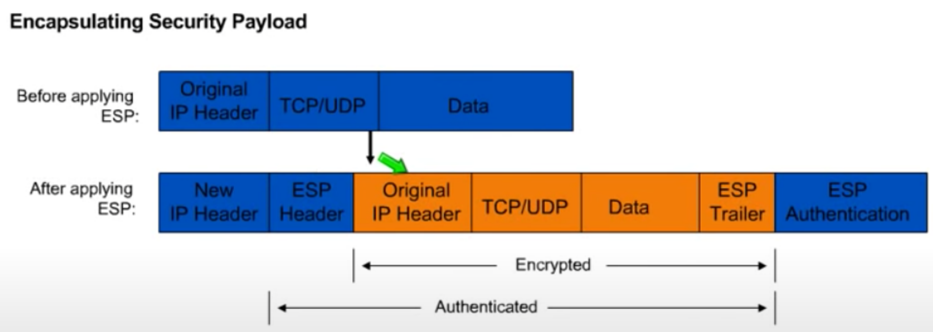
Each of these components contains some of the data needed to provide the necessary authentication and integrity checking. To prevent tampering, an ESP client has to sign the ESP header, application data, and ESP trailer into one unit, of course ESP is used to encrypt the application data and the ESP trailer to provide confidentiality. The combination of this overlapping signature and encryption operation provides good security.


IPSec - A ESP Packet
2. The ESP Client signs the ESP header, TCP or UDP header, Data and ESP trailer to prevent tampering
IP Header
ESP Header
TCP or UDP Header
DATA
ESP Trailer
ESP auth
Layer 3
Layer 4
Layers 5,6,7
1. ESP is used to encrypt TCP/UDP header, Data and ESP trailer to provide confidentiality
ESP Header
ESP Trailer
ESP Authentication
32
32
0 to 2040
8
8
Variable

Packet Format Alteration for ESP Transport Mode



Packet Format Alteration for ESP Tunnel Mode



Key Management Protocols

1. ### ISAKMP

   The Internet Security Agreement/Key Management Protocol and Oakley (ISAKMP)

   ISAKMP provides a way for two computers to agree on security settings and exchange a security key that they can use to communicate securely. A Security Association (SA) provides all the information needed for two computers to communicate securely. The SA contains a policy agreement that controls which algorithms and key lengths the two machines will use, plus the actual security keys used to securely exchange information.

   There are two steps in this process. First, the two computers must agree on the following three things:

   1) The encryption algorithm to be used (DES, triple DES)

   2) Which algorithm they'll use for verifying message integrity (MD5 or SHA-1)

   3) How connections will be authenticated: using public-key certificate, a shared secret key or Kerberos.

   Once all that has been sorted out, they start another round of negotiations which cover the following:

   1) Whether the Authentication Header (AH) protocol will be used

   2) Whether the Encapsulating Security Payload (ESP) protocol will be used

   3) Which encryption algorithm will be used for ESP

   4) Which authentication protocol will be used for AH

2. ### Internet Key Exchange (IKE)

   An IPSec component used for performing mutual authentication and establishing security associations.

   First step in IKE is to Establish the ISAKMP Session
   The two endpoints involved in IKE protocol exchange are called initiator and responder.

   #### Sequence of Events in IKE.

   1>Initiator sends one or more supported security proposals to the peer.Contains information on supported encryption protocols , authentication protocols , parameters such as key lifetime ,size etc.
   2>Responder replies with set of parameters which are supported by it.
   3>Initiator sends a Diffie-Helman Key exchange. With this process a Shared Symmetric key is arrived at by both initiator and responder.
   4>ISAKMP Session established and ISAKMP Security Associations based on the agreed parameters created.

When an IPSec datagram arrives at a router the it looks at the Security Associations database and determines the key, algorithm etc, which was created during the IPSec Tunnel Creation (IKE) Phase.

References

- https://david-waiting.medium.com/a-beginners-guide-to-generic-routing-encapsulation-fb2b4fb63abb
- http://www.firewall.cx/cisco-technical-knowledgebase/cisco-routers/872-cisco-router-gre-ipsec-tunnel-transport.html
- http://www.linux-admins.net/2010/09/tunneling-ipip-and-gre-encapsulation.html
- CableLabs_WR-SP-WiFi-GW-I03-140311-1.pdf
- https://developers.redhat.com/blog/2019/05/17/an-introduction-to-linux-virtual-interfaces-tunnels/