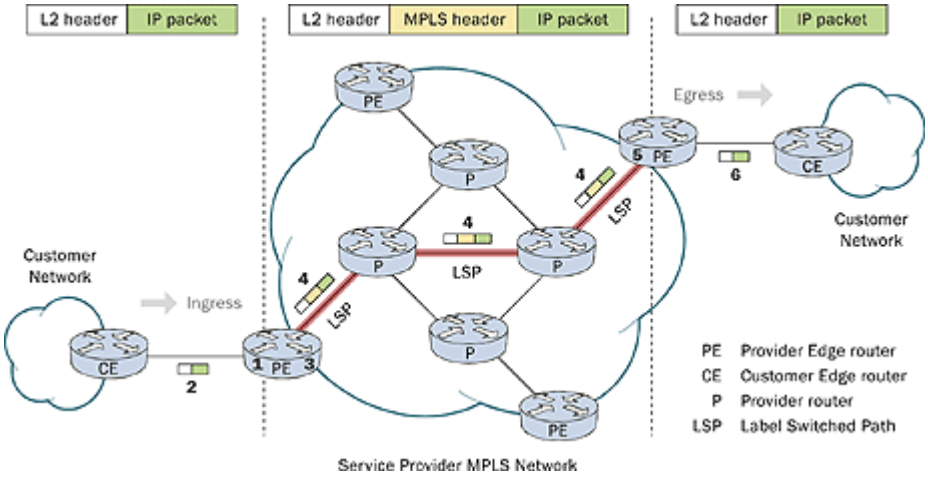
# What is MPLS really?

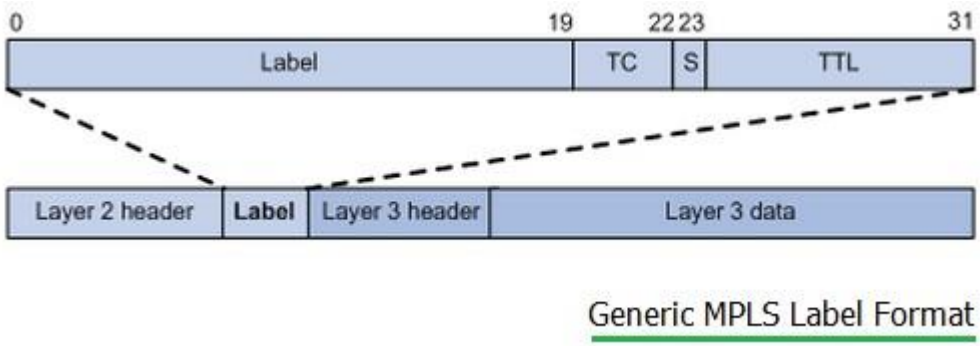
# PART 1: Quick introduction / basic routing.

Here is a reference for a quick overview if not familiar. Feel free to skip onto the next section

> ***Reference #1 -*** *https://en.wikipedia.org/wiki/Multiprotocol_Label_Switching*
>
> *Multiprotocol Label Switching (**MPLS**) is a type of data-carrying technique for high-performance telecommunications networks that directs data from one network node to the next based on short path labels rather than long network addresses, avoiding complex lookups in a routing table.*




**What does a MPLS header look like?**




Generic MPLS Label Format

Right so first of all MPLS is NOT an ISP exclusive technology, MPLS on its own is not a circuit or magical cloud made of fairy dust. Some people think of it this way because ISP are usually the only ones to use it… but really, everyone can use MPLS' encapsulation to get over hurdles.

**OSI MODEL**

MPLS slaps a label right between layer 2 and layer 3; it's sometimes referred to as a **LAYER 2.5**. This label range is **16 - 1048575** and used by routers to forward traffic rather than destination IP address.

Try to visualize it as a type of encapsulation, or a tunnel, where all the information from Layer 3 to layer 7 is "hidden" and ignored by the router

Some other tunnelling techniques are:

- GRE tunnels
- IPsec VPN tunnels
- Frame Relay
- ATM
- NHRP

They all follow the same concept. IP packets need to forward to a destination but either we intentionally DON'T want the core to be aware for security reasons or traffic engineering purposes **OR** we have no choice because the core **MUST NOT** be aware.

Three examples could be:

1) IP address conflict with core network
2) Too many routes already in the core
3) Certain ACLS or policies are in the core that will result in undesired filtering

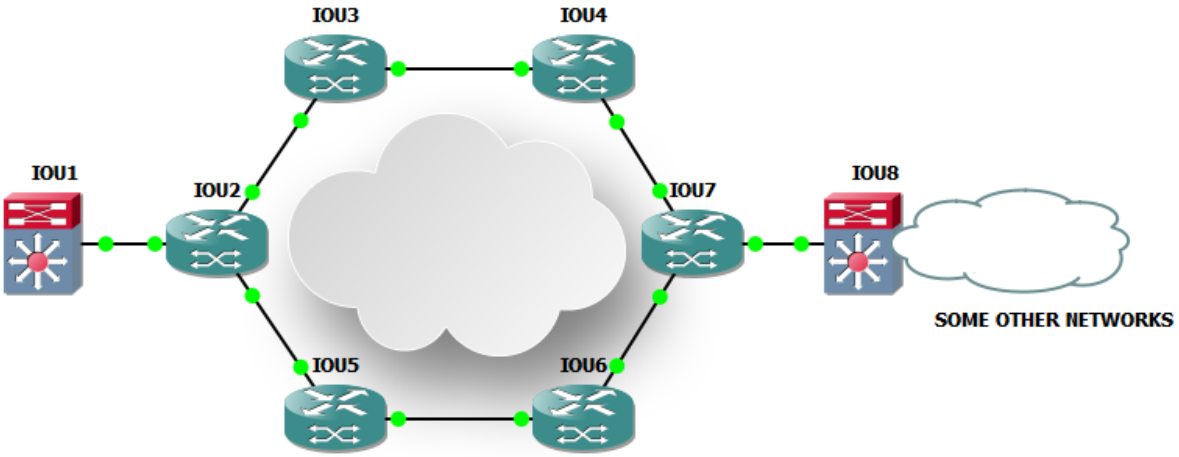
We'll have to tunnel/encapsulate to allow each side of the network to route the traffic.

So far this should be easily understood as a basic networking concept.

*Note: The rest of this document will refer to the devices in the exhibits as IOUX or ROUTERX*

**The Topology**

MPLS isn't exclusive to ISPs so the network topology will be referenced as our internal network rather than traditional PE / P / CE nodes, let's say we have two Cisco 6509s on each side and some ASRs in-between. **IOU8** connects to some other networks.




**On each device we add basic OSPF configuration to enable routing in the core**

```
IOU1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
IOU1(config)#router ospf 1
IOU1(config-router)#network 10.0.0.0 0.255.255.255 area 0
IOU1(config-router)#
```

**After OSPF convergence, we verify all routes learned for the 10.x.x.x networks**

- Networks 10.x.x.x are the networks connecting all visible devices
- IPs have been assigned to reflect the router number
- E.g. Link between router 3 and 4 is 10.3.4.0/24

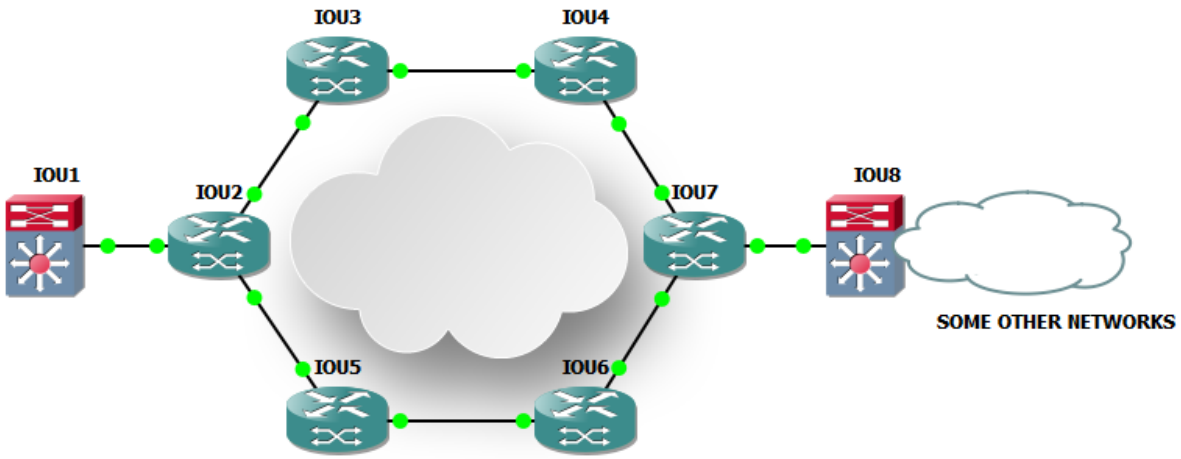
```
IOU1#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override

Gateway of last resort is not set

      10.0.0.0/8 is variably subnetted, 16 subnets, 2 masks
C        10.0.0.1/32 is directly connected, Loopback100
O        10.0.0.2/32 [110/11] via 10.1.2.2, 00:00:52, Ethernet0/0
O        10.0.0.4/32 [110/51] via 10.1.2.2, 00:00:36, Ethernet0/0
O        10.0.0.5/32 [110/21] via 10.1.2.2, 00:00:26, Ethernet0/0
O        10.0.0.6/32 [110/31] via 10.1.2.2, 00:00:26, Ethernet0/0
O        10.0.0.7/32 [110/41] via 10.1.2.2, 00:00:15, Ethernet0/0
O        10.0.0.8/32 [110/51] via 10.1.2.2, 00:00:05, Ethernet0/0
C        10.1.2.0/24 is directly connected, Ethernet0/0
L        10.1.2.1/32 is directly connected, Ethernet0/0
O        10.2.3.0/24 [110/20] via 10.1.2.2, 00:05:10, Ethernet0/0
O        10.2.4.0/24 [110/20] via 10.1.2.2, 00:04:28, Ethernet0/0
O        10.3.4.0/24 [110/60] via 10.1.2.2, 00:04:18, Ethernet0/0
O        10.4.6.0/24 [110/30] via 10.1.2.2, 00:04:18, Ethernet0/0
O        10.4.7.0/24 [110/50] via 10.1.2.2, 00:04:18, Ethernet0/0
O        10.6.7.0/24 [110/40] via 10.1.2.2, 00:04:18, Ethernet0/0
O        10.7.8.0/24 [110/50] via 10.1.2.2, 00:04:18, Ethernet0/0
IOU1#
IOU1#
IOU1#ping 10.0.0.8 source loopback 100
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.8, timeout is 2 seconds:
Packet sent with a source address of 10.0.0.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/5/5 ms
IOU1#
```

Okay so we have full reachability between the core devices

Remote networks **192.168.x.x** have not been advertised / redistributed… and are only reachable from IOU8 at the moment



**IOU1**

```
IOU1#ping 10.0.0.8
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.8, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/5/5 ms
IOU1#
IOU1#
IOU1#ping 192.168.1.33
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.33, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

**IOU8**

```
IOU8#ping 192.168.1.33
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.33, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/5 ms
IOU8#
```

Now we need **IOU1** to reach these networks **192.168.X.X**, how can we do this?

Redistribution and default routing is out the window as don't want these networks to exist in the core and a default route exists somewhere else. Feel free to dream up of some scenario where we wouldn't want this for whatever reason. Just go with me here…

An option and probably most popular way of doing this, is to create a **GRE** tunnel between IOU1 and IOU8, a good choice for simple things.

But what about a static route like this below on IOU1, should work right?

***IOU1 (config) Ip route 192.168.0.0 255.255.0.0 10.0.0.8***

**Let's try it**

1) **Route added**

```
IOU1(config)#
IOU1(config)#ip route 192.168.0.0 255.255.0.0 10.0.0.8
IOU1(config)#
```

2) **Ping to 192.168.1.33 results in UUUUU which means unrouteable. Traceroute fails also**

```
IOU1#ping 192.168.1.33
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.33, timeout is 2 seconds:
UUUUU
Success rate is 0 percent (0/5)
```

```
IOU1#trace 192.168.1.33
Type escape sequence to abort.
Tracing the route to 192.168.1.33
VRF info: (vrf in name/id, vrf out name/id)
  1 10.1.2.2 5 msec 5 msec 4 msec
  2 10.1.2.2 !H  !H  !H
IOU1#
IOU1#
IOU1#
IOU1#
IOU1#
IOU1#sh ip route | include 192\.168
S     192.168.0.0/16 [1/0] via 10.0.0.8
IOU1#
```

3) **CEF is fine though**

```
IOU1#show ip cef 192.168.1.33
192.168.0.0/16
  nexthop 10.1.2.2 Ethernet0/0
IOU1#
```

**The issue here is the packet is sent to router 2 but router 2 doesn't have a route to any 192.168.X.X networks. You might be thinking**

- But hang on, the static route was configured to route the traffic to **10.0.0.8**?
- Whats the point of recursive routing?
- I've definitely seen this work before, why isn't it working here?
- Router 2 shouldn't care and just pass the traffic to **10.0.0.8** which knows about the **192.168.X.X** routes.

### And here lies a common misconception with recursive routing

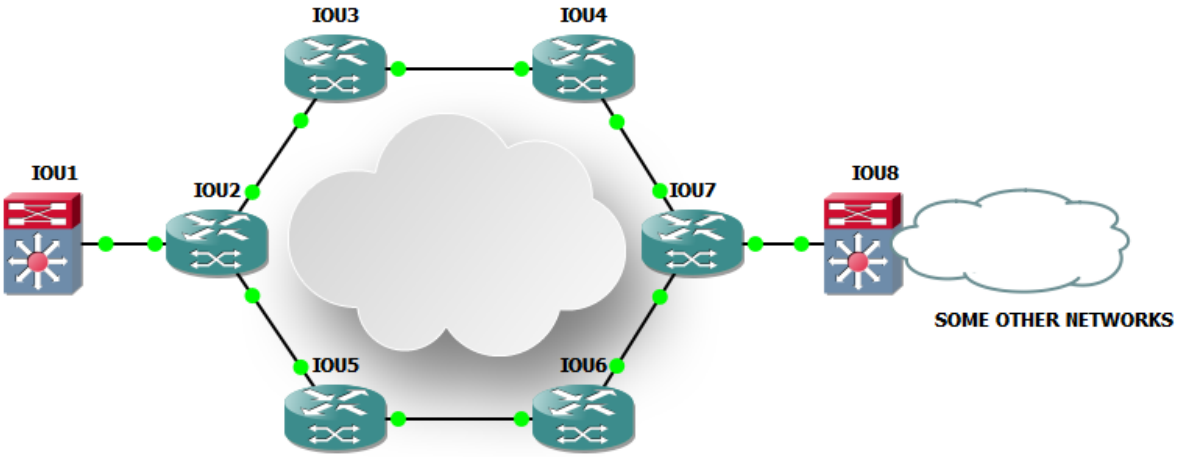
When we type ***Ip route 192.168.0.0 255.255.0.0 10.0.0.8*** the router isn't going to forward the traffic to **10.0.0.8**, instead, it will perform local recursion to **10.0.0.8**.

**10.0.0.8's** network doesn't exist on **IOU1** therefore **IOU1** is unable to choose a predetermined exit interface connected to it. It must therefore do a route lookup locally and pick whatever next hop and exit interface is used for **10.0.0.8** – This is what route recursion is!

*Route recursion is a local lookup NOT an encapsulation or tunnel technique.*

The only way **IOU2** would be able to direct the traffic to **IOU8** would be if IOU1 added a new layer to the IP packet to change the destination from **192.168.1.33** to **10.0.0.8.**

Whatever method IOU1 used to add this layer, IOU8 would need to speak the same language and know how to remove this outer layer to deliver the packet to its original destination.




So you're probably thinking one of 3 things.

1) Duh!!
2) That's interesting didn't know that.
3) That's interesting but wtf does that have to do with MPLS.

Just hang tight, we're getting there!

**Okay so the static route on its own is no good**

What if we add a GRE tunnel?

```
IOU1#sh run int tun 10
Building configuration...

Current configuration : 117 bytes
!
interface Tunnel10
 ip address 10.1.8.1 255.255.255.0
 tunnel source Loopback100
 tunnel destination 10.0.0.8
end
```

```
IOU8#sh run int tun 10
Building configuration...

Current configuration : 117 bytes
!
interface Tunnel10
 ip address 10.1.8.8 255.255.255.0
 tunnel source Loopback100
 tunnel destination 10.0.0.1
end
```

```
IOU1(config)#ip route 192.168.0.0 255.255.0.0 10.1.8.8
```

```
IOU1#ping 192.168.1.33
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.33, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/2 ms
IOU1#
IOU1#
IOU1#
IOU1#
IOU1#
IOU1#trace 192.168.1.33
Type escape sequence to abort.
Tracing the route to 192.168.1.33
VRF info: (vrf in name/id, vrf out name/id)
  1 10.1.8.8 2 msec 1 msec 1 msec
IOU1#
```

```
IOU1#sh ip cef 192.168.0.0
192.168.0.0/16
  nexthop 10.1.8.8 Tunnel10
IOU1#
```

**Okay so this works** - The traffic is now **encapsulated**. This is not route recursion!

The actual destination of the packet has been changed from **192.168.1.33** to **10.0.0.8** by the tunnel command "tunnel destination 10.0.0.8"

Now… technically speaking that isn't correct, the destination itself hasn't changed but rather the router adds a new Generic Routing & Encapsulation (GRE) header with a new source and destination as seen below.

Router 2, 3, 4, 5, 6 and 7 will only care about the outer header. When traffic reaches IOU8 this header is stripped off and the true destination is then delivered to the routing engine.

```
> Frame 64: 138 bytes on wire (1104 bits), 138 bytes captured (1104 bits) on interface 0
> Ethernet II, Src: aa:bb:cc:00:15:00 (aa:bb:cc:00:15:00), Dst: aa:bb:cc:00:0a:00 (aa:bb:cc:00:0a:00)
> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.8
> Generic Routing Encapsulation (IP)
> Internet Protocol Version 4, Src: 10.1.8.1, Dst: 192.168.1.33
> Internet Control Message Protocol
```

This is acceptable for simple things, small networks or patch-work but not a scalable solution. What if we had **10** routers fully meshed? That's **45** GRE tunnels

What if you had **50** routers fully meshed? That's **1225** GRE tunnels you've got to create, manage and find IP addresses for them all – Good luck with that! *(Full Mesh = "n x (n-1)/2")*

This is where MPLS comes in handy.

# PART 2: Enabling MPLS and basic operation

MPLS uses the same technique of adding a new layer of information for the routers in the core to use to forward traffic instead of looking at the true destinations.

Let's enable MPLS in the core and see what it looks like. This needs to be done on all core routers but the config couldn't be simpler…

**On each interface add command "mpls ip" – That's actually all there is to it!**

MPLS uses Label distribution protocol (LDP) they work hand in hand to create a Label Switch Path (LSP), this sort of works like a dynamic routing protocol.

```
IOU1(config-if)#interface e0/0
IOU1(config-if)#
IOU1(config-if)#
IOU1(config-if)#mpls ip
IOU1(config-if)#
IOU1(config-if)#
IOU1(config-if)#
*Feb 26 01:15:20.322: %LDP-5-NBRCHG: LDP Neighbor 10.0.0.2:0 (1) is UP
```

After LDP convergence, we inspect IOU1's mpls database.

The set of labels to the left, local and outgoing are the identifiers and bits of information used to swap/route traffic.

10.0.0.8/32 local label is **16** and if a packet is destined for label **16** the outgoing label **53** is attached or "swapped". It's important to note… the lookup isn't done for **10.0.0.8** but rather just the label **16**

```
IOU1#show mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing    Next Hop
Label      Label      or Tunnel Id     Switched      interface
16         53         10.0.0.8/32      0             Et0/0       10.1.2.2
17         54         10.0.0.7/32      0             Et0/0       10.1.2.2
18         55         10.0.0.6/32      0             Et0/0       10.1.2.2
19         50         10.0.0.5/32      0             Et0/0       10.1.2.2
20         56         10.0.0.4/32      0             Et0/0       10.1.2.2
21         Pop Label  10.0.0.2/32      0             Et0/0       10.1.2.2
22         57         10.3.4.0/24      0             Et0/0       10.1.2.2
23         Pop Label  10.2.3.0/24      0             Et0/0       10.1.2.2
24         58         10.7.8.0/24      0             Et0/0       10.1.2.2
25         59         10.4.7.0/24      0             Et0/0       10.1.2.2
26         60         10.6.7.0/24      0             Et0/0       10.1.2.2
29         No Label   192.168.0.0/16   0             Et0/0       10.1.2.2
30         Pop Label  10.2.5.0/24      0             Et0/0       10.1.2.2
31         52         10.5.6.0/24      0             Et0/0       10.1.2.2
IOU1#
```

Let's re-create that original static route, **DELETE** the tunnel interfaces and see what happens

***Ip route 192.168.0.0 255.255.0.0 10.0.0.8***

```
IOU1(config)#
IOU1(config)#ip route 192.168.0.0 255.255.0.0 10.0.0.8
IOU1(config)#
```

```
IOU1#ping 192.168.1.33
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.33, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/6 ms
IOU1#
```

**It works this time! But how?**

A quick peek at the Cisco Express Forwarding (CEF) entry reveals the trick!

```
IOU1#sh ip cef 192.168.1.33
192.168.0.0/16
  nexthop 10.1.2.2 Ethernet0/0 label 16
IOU1#
```

The same recursive process takes place, however just before the data is passed to IOU2 with the destination of 192.168.1.33, as with the first example, the router makes one more decision, it essentially reads the CEF entry and says "Oh I see we have a label 16 here, let's route with this label **16** thing instead and add a new MPLS header before forwarding the traffic"

**16** is then looked up in the FIB or LFIB (MPLS forwarding information base) and swapped for outgoing label **53** – it is at this point where the core routers **lose visibility** or… as I would prefer to say "Couldn't give 2 shits" about the IP header

**Packet with label 53 is sent to router 2**

```
IOU1#show mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
16         53         10.0.0.8/32      0             Et0/0      10.1.2.2
17         54         10.0.0.7/32      0             Et0/0      10.1.2.2
18         55         10.0.0.6/32      0             Et0/0      10.1.2.2
19         50         10.0.0.5/32      0             Et0/0      10.1.2.2
20         56         10.0.0.4/32      0             Et0/0      10.1.2.2
21         Pop Label  10.0.0.2/32      0             Et0/0      10.1.2.2
22         57         10.3.4.0/24      0             Et0/0      10.1.2.2
23         Pop Label  10.2.3.0/24      0             Et0/0      10.1.2.2
24         58         10.7.8.0/24      0             Et0/0      10.1.2.2
25         59         10.4.7.0/24      0             Et0/0      10.1.2.2
26         60         10.6.7.0/24      0             Et0/0      10.1.2.2
29         No Label   192.168.0.0/16   0             Et0/0      10.1.2.2
30         Pop Label  10.2.5.0/24      0             Et0/0      10.1.2.2
31         52         10.5.6.0/24      0             Et0/0      10.1.2.2
IOU1#
```

**And what's IOU2 up to?**

Label **53** is swapped for label **64** and so forth and so forth, this is how the forwarding is done by using this Label Switched Path (LSP) you can sort of think of these LSPs of it as a web of mini dynamic tunnels in the core.

```
IOU2#sh mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
50         Pop Label  10.0.0.5/32      0             Et0/2      10.2.5.5
51         Pop Label  10.0.0.1/32      0             Et0/0      10.1.2.1
52         Pop Label  10.5.6.0/24      0             Et0/2      10.2.5.5
53         64         10.0.0.8/32      0             Et0/2      10.2.5.5
54         65         10.0.0.7/32      0             Et0/2      10.2.5.5
55         66         10.0.0.6/32      0             Et0/2      10.2.5.5
56         67         10.0.0.4/32      0             Et0/2      10.2.5.5
57         68         10.3.4.0/24      0             Et0/2      10.2.5.5
58         69         10.7.8.0/24      0             Et0/2      10.2.5.5
59         70         10.4.7.0/24      0             Et0/2      10.2.5.5
60         71         10.6.7.0/24      0             Et0/2      10.2.5.5
IOU2#
```

IOU3, 4, 5 and 6 will do the same thing as above, swapping labels along the path so no screenshot is necessary. By the time it reaches **IOU8** the MPLS tag is "**popped**" / removed by **IOU7** and the true destination is once again visible by **IOU8** who will then route the traffic.

The reason it is removed by **IOU7** is because the embedded traffic belongs to **IOU8** so no tag is needed there. It's similar to a computer plugged into a switch with **"switch port access Vlan X"** where the Vlan tag is removed at the switch before passing the traffic to the computer, as the computer has no visibility nor does it care what Vlan it is actually in

```
IOU7#sh mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
700        Pop Label  10.0.0.8/32      0             Et0/0      10.7.8.8
701        Pop Label  10.0.0.6/32      0             Et0/2      10.6.7.6
702        18         10.0.0.5/32      0             Et0/2      10.6.7.6
703        Pop Label  10.0.0.4/32      0             Et0/1      10.4.7.4
704        20         10.0.0.2/32      0             Et0/2      10.6.7.6
705        21         10.0.0.1/32      0             Et0/2      10.6.7.6
706        Pop Label  10.3.4.0/24      0             Et0/1      10.4.7.4
707        23         10.2.3.0/24      0             Et0/2      10.6.7.6
708        24         10.1.2.0/24      0             Et0/2      10.6.7.6
709        27         10.2.5.0/24      0             Et0/2      10.6.7.6
710        Pop Label  10.5.6.0/24      0             Et0/2      10.6.7.6
IOU7#
```

**Traceroute shows the labels as the traverse the MPLS core**

```
IOU1#traceroute 192.168.1.33
Type escape sequence to abort.
Tracing the route to 192.168.1.33
VRF info: (vrf in name/id, vrf out name/id)
  1 10.1.2.2 [MPLS: Label 53 Exp 0] 6 msec 0 msec 1 msec
  2 10.2.5.5 [MPLS: Label 64 Exp 0] 1 msec 0 msec 0 msec
  3 10.5.6.6 [MPLS: Label 16 Exp 0] 1 msec 6 msec 1 msec
  4 10.6.7.7 [MPLS: Label 700 Exp 0] 1 msec 1 msec 0 msec
  5 10.7.8.8 1 msec 1 msec 1 msec
IOU1#
```

**Wireshark capture –** Just like the GRE tunnel, the MPLS layer is above the IPv4 header which is now ignored by the router.

```
> Frame 17: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface 0
> Ethernet II, Src: aa:bb:cc:00:15:00 (aa:bb:cc:00:15:00), Dst: aa:bb:cc:00:0a:00 (aa:bb:cc:00:0a:00)
v MultiProtocol Label Switching Header, Label: 53, Exp: 0, S: 1, TTL: 255
    0000 0000 0000 0011 0101 .... .... .... = MPLS Label: 53
    .... .... .... .... .... 000. .... .... = MPLS Experimental Bits: 0
    .... .... .... .... .... ...1 .... .... = MPLS Bottom Of Label Stack: 1
    .... .... .... .... .... .... 1111 1111 = MPLS TTL: 255
> Internet Protocol Version 4, Src: 10.1.2.1, Dst: 192.168.1.33
> Internet Control Message Protocol
```

And this is MPLS tunnelling working it its most basic and simplest form.

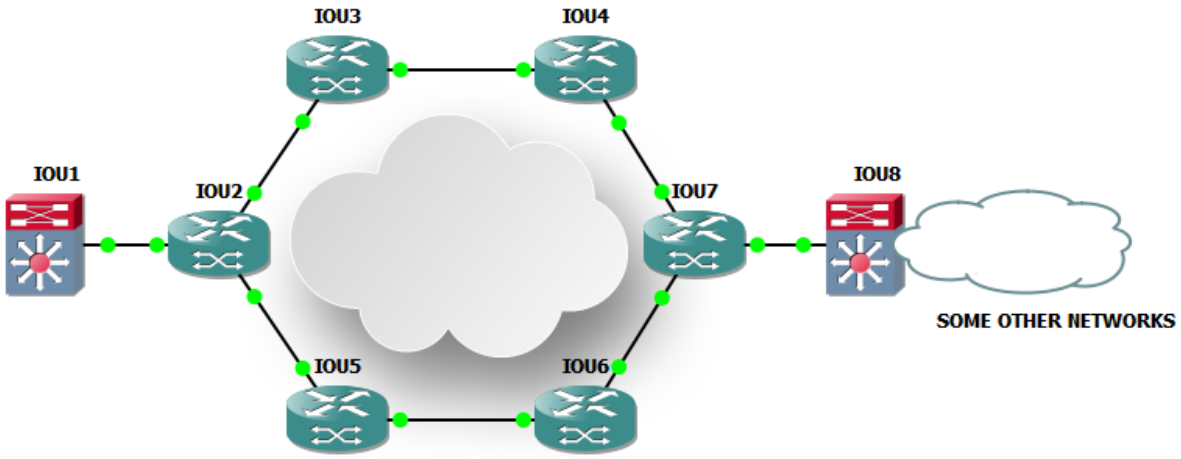
# PART 3: USING MPLS with BGP for NEXT-HOP reachability and encapsulation

Part 1 and 2 covered basic routing and MPLS as a tunnelling mechanism to reach networks not existing in the core, as demonstrated, if there is no encapsulation, the core routers will attempt to route the packet based on the true destination and fail miserably because these routes do not exist in the core. (And don't say – Well why not put them in the core you miserable bastard)

What this means is if we had BGP between IOU1 and IOU8, we wouldn't be able to route traffic between the BGP speakers without a GRE tunnel, IPsec tunnel, or MPLS (or some other tunnel technique) which would be a shame as BGP is an excellent choice to share network information (**NLRI**) between routers not directly connected as BGP can preserve the next hop as itself from miles away unlike an IGP like EIGRP, OSPF etc…

**Let's remove the static route and peer IOU1 with IOU8 using BGP**

We will then advertise the **192.168.X.X** prefixes from 8 to 1… these routes MUST not show up in the core




**IOU8**

```
router bgp 800
 bgp log-neighbor-changes
 network 192.168.1.0 mask 255.255.255.224
 network 192.168.1.32 mask 255.255.255.224
 network 192.168.1.64 mask 255.255.255.224
 network 192.168.1.96 mask 255.255.255.224
 neighbor 10.0.0.1 remote-as 100
 neighbor 10.0.0.1 ttl-security hops 5
 neighbor 10.0.0.1 update-source Loopback100
 neighbor 10.0.0.1 timers 10 20
IOU8#
```

**IOU1**

```
router bgp 100
 bgp log-neighbor-changes
 neighbor 10.0.0.8 remote-as 800
 neighbor 10.0.0.8 ttl-security hops 5
 neighbor 10.0.0.8 update-source Loopback100
 neighbor 10.0.0.8 timers 10 20
IOU1#
```

```
IOU1#show ip bgp
BGP table version is 5, local router ID is 10.0.0.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 *>  192.168.1.0/27   10.0.0.8                 0             0 800 i
 *>  192.168.1.32/27  10.0.0.8                 0             0 800 i
 *>  192.168.1.64/27  10.0.0.8                 0             0 800 i
 *>  192.168.1.96/27  10.0.0.8                 0             0 800 i
IOU1#
```

**Routing still works.** Recursion is done to 10.0.0.8 but the tag is also picked up on the CEF entry and MPLS takes over and tunnels the traffic. So ping works.

```
IOU1#ping 192.168.1.33
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.33, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/5/5 ms
IOU1#
```

Quick verification - Core router IOU2 has no routes to 192.168.X.X and is unable to ping as expected

```
IOU2#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override

Gateway of last resort is not set

      10.0.0.0/8 is variably subnetted, 18 subnets, 2 masks
O        10.0.0.1/32 [110/11] via 10.1.2.1, 01:32:30, Ethernet0/0
C        10.0.0.2/32 is directly connected, Loopback100
O        10.0.0.4/32 [110/41] via 10.2.5.5, 01:06:42, Ethernet0/2
O        10.0.0.5/32 [110/11] via 10.2.5.5, 01:32:30, Ethernet0/2
O        10.0.0.6/32 [110/21] via 10.2.5.5, 01:32:03, Ethernet0/2
O        10.0.0.7/32 [110/31] via 10.2.5.5, 01:06:42, Ethernet0/2
O        10.0.0.8/32 [110/41] via 10.2.5.5, 01:00:34, Ethernet0/2
C        10.1.2.0/24 is directly connected, Ethernet0/0
L        10.1.2.2/32 is directly connected, Ethernet0/0
C        10.2.3.0/24 is directly connected, Ethernet0/1
L        10.2.3.2/32 is directly connected, Ethernet0/1
C        10.2.5.0/24 is directly connected, Ethernet0/2
L        10.2.5.2/32 is directly connected, Ethernet0/2
O        10.3.4.0/24 [110/50] via 10.2.5.5, 01:06:42, Ethernet0/2
O        10.4.7.0/24 [110/40] via 10.2.5.5, 01:06:42, Ethernet0/2
O        10.5.6.0/24 [110/20] via 10.2.5.5, 01:32:30, Ethernet0/2
O        10.6.7.0/24 [110/30] via 10.2.5.5, 01:07:23, Ethernet0/2
O        10.7.8.0/24 [110/40] via 10.2.5.5, 01:03:44, Ethernet0/2
IOU2#
IOU2#
IOU2#ping 192.168.1.33
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.33, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
IOU2#
```

So we've demonstrated how to MPLS as a tunnelling mechanism instead of manual static GRE tunnels and routing using either **BGP** or **static** routes. It may or may not seem obvious but these are the only two methods to use for routing (excluding policy routing), we're unable to use **OSPF**, **EIGRP** or **RIP** to do this. These IGPs are designed to work closely with the routers, their link states, metrics, and depend heavily on information from its connected neighbour.

Static recursive routing and BGP routing is arbitrary. A static route is extremely simple with no interest in the network state and is localised.

BGP neighbours can be many hops apart and can also behave as if it's localised, doesn't take link states into consideration, metrics etc. – after all BGP isn't' a real routing protocol, it's more of an application gateway protocol

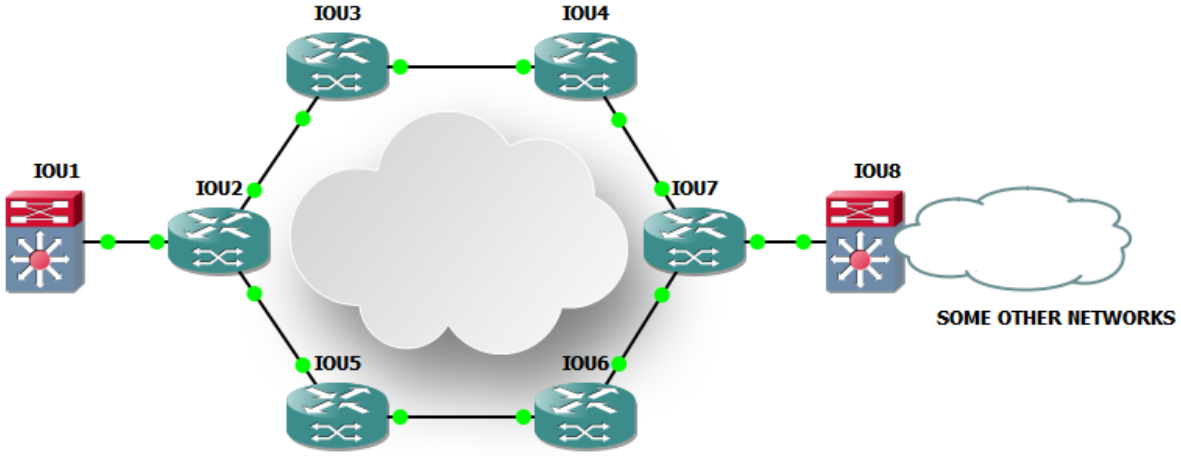
# PART 4: USE CASE #1 – SEGREGATION

**Introducing the NINJA NETWORKS – The real fun begins here ;-)**

SCENARIO: Let's say we acquired a 3rd party super-secret ninja network behind IOU8 in range **172.20.X.X** and they MUST to be separate to the Global Routing Table (GRT) on all routers. Because the company is so large they've used almost all private IPv4 addresses and there is a large part of this ninja network that just so happens to already have been configured with same IP addresses that exists in the GRT therefore re-addressing is not feasible. NAT can't be used due to too many static NAT requirements

**We need a Virtual Routing & Forwarding (VRF) instance**

And here is where it gets interesting



For the VRF to work we need to do one of the following:

1) Create the VRF on each core router route hop by hop

Or

2) Create the VRF on each end and tunnel the VRF over a tunnel

Creating the VRF on each router is technically ok, this method is called **VRF-lite**. The issue with this is its scalability limitations. Any time a new VRF is needed, we'll be forced to configure each core router with sub-interfaces and VRFs, it would be an administrative nightmare, and you really don't want to be messing around with the core network in this way

Tunnelling is a good idea; a GRE tunnel between IOU1 and IOU8 will work. But if we're going to be tunnelling, why not use the existing dynamic MPLS tunnel network?

**Step 1 - On IOU1 and IOU8, create your VRFs**

```
!
vrf definition NINJA-NETWORKS
 rd 100:800
 !
 address-family ipv4
  route-target export 100:800
  route-target import 100:800
 exit-address-family
!
```

**RD** (Route Distinguisher) separates and places routes in its own container on the local router

**RT** (Route Target) controls which routes/prefixes are members for the RD container / VRF

**Step 2 – find a way to transport this information between the two end devices**

This can be done with the **BGP VPNv4 Address Family Identifier** (Might want to do some research on BGP AFI/SAFI but the common ones are IPv4, VPNv4 and their IPv6 counterparts)

**IOU1**

```
!
router bgp 100
 bgp log-neighbor-changes
 neighbor 10.0.0.8 remote-as 800
 neighbor 10.0.0.8 ttl-security hops 5
 neighbor 10.0.0.8 update-source Loopback100
 neighbor 10.0.0.8 timers 10 20
 !
 address-family vpnv4
  neighbor 10.0.0.8 activate
  neighbor 10.0.0.8 send-community extended
 exit-address-family
!
```

**IOU8**

```
!
router bgp 800
 bgp log-neighbor-changes
 network 192.168.1.0 mask 255.255.255.224
 network 192.168.1.32 mask 255.255.255.224
 network 192.168.1.64 mask 255.255.255.224
 network 192.168.1.96 mask 255.255.255.224
 neighbor 10.0.0.1 remote-as 100
 neighbor 10.0.0.1 ttl-security hops 5
 neighbor 10.0.0.1 update-source Loopback100
 neighbor 10.0.0.1 timers 10 20
 !
 address-family vpnv4
  neighbor 10.0.0.1 activate
  neighbor 10.0.0.1 send-community extended
 exit-address-family
!
```

*The VPNv4 acts as a….what's the simplest way to explain this… method of exchanging VRF attributes like **RD** and **RT** and only needs to be configured on the end points (or PE routers)*

*RFC4760 - https://tools.ietf.org/html/rfc4760 - Explains how the AFI and SAFIs work with NLRI and relevant encoding*

*RFC 4364 - https://tools.ietf.org/html/rfc4364 - (Section 4) gives a breakdown of the structure of VPNv4 addresses*

*Detailed workings of these things are out of scope of his document though so let's move on.*

**View from IOU1.**

This on its own doesn't do much; all it does is create a peering… or backbone if you will, for exchanging VRF information between the BGP speakers. Now we need to add the VRF specific routes and AFI + SAFI to BGP

```
IOU1#sh bgp vpnv4 unicast all summary
BGP router identifier 10.0.0.1, local AS number 100
BGP table version is 1, main routing table version 1

Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.0.0.8        4          800      85      85        1    0    0 00:08:43        0
IOU1#
```

**Now add the VRF to BGP on IOU1**

```
!
router bgp 100
 bgp log-neighbor-changes
 neighbor 10.0.0.8 remote-as 800
 neighbor 10.0.0.8 ttl-security hops 5
 neighbor 10.0.0.8 update-source Loopback100
 neighbor 10.0.0.8 timers 10 20
 !
 address-family vpnv4
  neighbor 10.0.0.8 activate
  neighbor 10.0.0.8 send-community extended
 exit-address-family
 !
 address-family ipv4 vrf NINJA-NETWORKS
  network 172.20.100.1 mask 255.255.255.255
 exit-address-family
!
```

**Add the same on IOU8**

```
!
router bgp 800
 bgp log-neighbor-changes
 network 192.168.1.0 mask 255.255.255.224
 network 192.168.1.32 mask 255.255.255.224
 network 192.168.1.64 mask 255.255.255.224
 network 192.168.1.96 mask 255.255.255.224
 neighbor 10.0.0.1 remote-as 100
 neighbor 10.0.0.1 ttl-security hops 5
 neighbor 10.0.0.1 update-source Loopback100
 neighbor 10.0.0.1 timers 10 20
 !
 address-family vpnv4
  neighbor 10.0.0.1 activate
  neighbor 10.0.0.1 send-community extended
 exit-address-family
 !
 address-family ipv4 vrf NINJA-NETWORKS
  network 172.20.1.0 mask 255.255.255.224
  network 172.20.1.32 mask 255.255.255.224
  network 172.20.1.64 mask 255.255.255.224
  network 172.20.1.96 mask 255.255.255.224
 exit-address-family
!
```

NOW - These details will be transferred between the VPNv4 peers

Verification from R1

```
IOU1#sh ip bgp vpnv4 vrf NINJA-NETWORKS | begin Route
Route Distinguisher: 100:800 (default for vrf NINJA-NETWORKS)
 *>  172.20.1.0/27    10.0.0.8                 0             0 800 i
 *>  172.20.1.32/27   10.0.0.8                 0             0 800 i
 *>  172.20.1.64/27   10.0.0.8                 0             0 800 i
 *>  172.20.1.96/27   10.0.0.8                 0             0 800 i
 *>  172.20.100.1/32  0.0.0.0                  0         32768 i
IOU1#
IOU1#show ip route vrf NINJA-NETWORKS | begin Gateway
Gateway of last resort is not set

      172.20.0.0/16 is variably subnetted, 5 subnets, 2 masks
B        172.20.1.0/27 [20/0] via 10.0.0.8, 00:33:28
B        172.20.1.32/27 [20/0] via 10.0.0.8, 00:32:57
B        172.20.1.64/27 [20/0] via 10.0.0.8, 00:32:57
B        172.20.1.96/27 [20/0] via 10.0.0.8, 00:32:57
C        172.20.100.1/32 is directly connected, Loopback1
IOU1#
IOU1#ping vrf NINJA-NETWORKS 172.20.1.33
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.20.1.33, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/5 ms
```

Alright so we have a working VRF tunnel through the network

You might be thinking, “Okay this looks all good, VRF headers are placed in the IP packets and assigned an MPLS label for routing”. It would be very wrong to think so.

VRFs are locally significant, so much so when configuring two routers to communicate in a VRF without MPLS, you must place the interconnecting interfaces into the VRF **(ip vrf forwarding SALES)** however, this doesn’t add any special VRF tags, instead it isolates the interface on the router and removes any identifiers before passing out the interface, very much like a Vlan on a switch with **“switchport access Vlan xx”**

Also if you’re thinking

*“But hang on; BGP knows about the VRFs and exchanges routes between VRFs, but if there’s no VRF header or IP vrf forwarding set, how does it know where to put the data packets then”*

And this is done by – ANOTHER mpls label! (Otherwise known as VPNv4 label or BGP label)

**Let's look at IOU8**

*Got a bit bored with the "Solarized Dark" theme so switched to this grey pastel theme… anyway…*

Below, outlined in blue – new labels for BGP VPNv4, the "V" outlined in red indicates it's a VPNv4 label

There are a few ways of seeing these labels, just to list a couple:

1. show mpls forwarding-table
2. show ip bgp vpnv4 all labels

```
IOU8#show ip bgp vpnv4 vrf NINJA-NETWORKS labels
   Network          Next Hop      In label/Out label
Route Distinguisher: 100:800 (NINJA-NETWORKS)
   172.20.1.0/27    0.0.0.0         30/nolabel(NINJA-NETWORKS)
   172.20.1.32/27   0.0.0.0         31/nolabel(NINJA-NETWORKS)
   172.20.1.64/27   0.0.0.0         32/nolabel(NINJA-NETWORKS)
   172.20.1.96/27   0.0.0.0         33/nolabel(NINJA-NETWORKS)
   172.20.100.1/32  10.0.0.1        nolabel/30

IOU8#
IOU8#
IOU8#sh mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
17         Pop Label  10.0.0.7/32      0             Et0/0      10.7.8.7
18         703        10.0.0.4/32      0             Et0/0      10.7.8.7
19         704        10.3.4.0/24      0             Et0/0      10.7.8.7
20         Pop Label  10.4.7.0/24      0             Et0/0      10.7.8.7
21         Pop Label  10.6.7.0/24      0             Et0/0      10.7.8.7
22         706        10.0.0.6/32      0             Et0/0      10.7.8.7
23         707        10.0.0.5/32      0             Et0/0      10.7.8.7
24         708        10.0.0.2/32      0             Et0/0      10.7.8.7
25         709        10.0.0.1/32      0             Et0/0      10.7.8.7
26         710        10.2.3.0/24      0             Et0/0      10.7.8.7
27         711        10.1.2.0/24      0             Et0/0      10.7.8.7
28         712        10.2.5.0/24      0             Et0/0      10.7.8.7
29         713        10.5.6.0/24      0             Et0/0      10.7.8.7
30         No Label   172.20.1.0/27[V] 0             aggregate/NINJA-NETWORKS
31         No Label   172.20.1.32/27[V]   \
                                       0             aggregate/NINJA-NETWORKS
32         No Label   172.20.1.64/27[V]   \
                                       0             aggregate/NINJA-NETWORKS
33         No Label   172.20.1.96/27[V]   \
                                       0             aggregate/NINJA-NETWORKS
IOU8#
```

1. Packet capture shows the outer label **705** used by the core and **31** used for the VRF.
2. The outer label **705** will be stripped by IOU7 so label **31** is visible by IOU8
3. BGP on IOU8 examines it's label table and determines the data is destined for VRF NINJA-NETWORKS
4. BGP removes the label and the packet heads off to its destination **172.20.1.33**

```
> Frame 20: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface 0
> Ethernet II, Src: aa:bb:cc:00:03:20 (aa:bb:cc:00:03:20), Dst: aa:bb:cc:00:08:20 (aa:bb:cc:00:08:20)
> MultiProtocol Label Switching Header, Label: 705, Exp: 0, S: 0, TTL: 252
> MultiProtocol Label Switching Header, Label: 31, Exp: 0, S: 1, TTL: 255
> Internet Protocol Version 4, Src: 172.20.100.1, Dst: 172.20.1.33
> Internet Control Message Protocol
```

This technique using stacked MPLS labels is favoured by ISPs to offer MPLS based services using minimum hardware and permits their customers to share the same hardware and IP space but as we see above this can be an efficient way to tunnel traffic on local networks also.

# PART 5: USE CASE #2 – L3VPN over MGRE

Here we will examine a scenario where MPLS is not used in the core, maybe the core doesn't support MPLS, or we're traversing the internet or maybe the core is so large and enabling MPLS on every core device and interface will take forever.

Again we can solve this by creating GRE tunnels between the edge routers; however, creating static GRE tunnels is subject to:

A) Limited scalability
B) Complexity

Depending on the design, one or more edge routers may be:

1) Acting as a BGP hub or route-reflector-client, the edge routers may be connected to multiple hubs
2) Utilizing BGP multipath
3) Some routes learnt by a BGP peer might actually have a shorter path through another router and changing the NEXT-HOP might be desirable

A manual GRE tunnel for each possible scenario – This can quickly get out of hand. L3VPNs over mGRE will help us here but creating dynamic GRE tunnels when needed, similar to how DMVPN works.

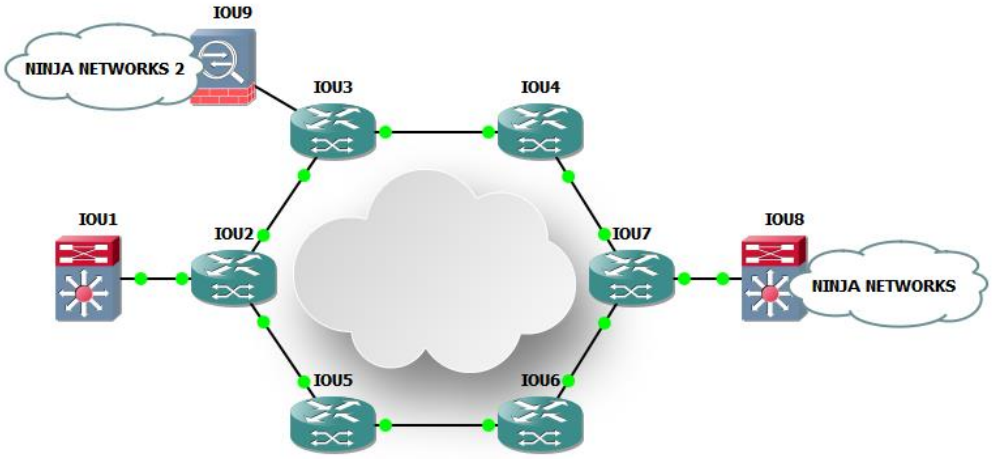
*This document shares benefits of BGP NEXT-HOP modification – Read only if interested in finer details*

*http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_bgp/configuration/xe-3s/irg-xe-3s-book/irg-NEXT-HOP.pdf*

**Let's add IOU9 to the mix.**

THE SCENARIO

- New NINJA NETWORKS **(169.254.X.X)** are located behind a firewall on device **IOU9**, labelled NINJA-NETWORKS 2 on the diagram but in the same VRF
- IOU9 has a BGP peering with IOU8
- IOU1 still has its BGP peering with IOU8
- For security and policy reasons, BGP peeing between IOU1 and IOU9 is **NOT** permitted but other network traffic is allowed
- MPLS has been **REMOVED** from the core




First let's get communication up between IOU1 and IOU8, IOU1 tries to reach networks via IOU8 but fails as expected.

```
IOU1#sh ip route vrf NINJA-NETWORKS | begin Gateway
Gateway of last resort is not set

      169.254.0.0/27 is subnetted, 4 subnets
B        169.254.1.0 [20/0] via 10.0.0.8, 00:01:14
B        169.254.1.32 [20/0] via 10.0.0.8, 00:01:14
B        169.254.1.64 [20/0] via 10.0.0.8, 00:01:14
B        169.254.1.96 [20/0] via 10.0.0.8, 00:01:14
      172.20.0.0/16 is variably subnetted, 5 subnets, 2 masks
B        172.20.1.0/27 [20/0] via 10.0.0.8, 00:00:43
B        172.20.1.32/27 [20/0] via 10.0.0.8, 00:00:43
B        172.20.1.64/27 [20/0] via 10.0.0.8, 00:00:43
B        172.20.1.96/27 [20/0] via 10.0.0.8, 00:00:43
C        172.20.100.1/32 is directly connected, Loopback1
IOU1#
IOU1#
IOU1#ping vrf NINJA-NETWORKS ip 169.254.1.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 169.254.1.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
IOU1#trace vrf NINJA-NETWORKS ip 169.254.1.1
Type escape sequence to abort.
Tracing the route to 169.254.1.1
VRF info: (vrf in name/id, vrf out name/id)
  1  *  *  *
  2  *  *  *
  3  *  *  *
  4
IOU1#
```

**We need a GRE Tunnel…**

Annoyingly, because we're creating a static tunnel, we're forced to

1) Find a new IP range for the tunnel
2) BGP was smart enough to insert itself as a NEXT-HOP even though 10.0.0.8 isn't in the VRF network but if we use a static tunnel, we now have to add the IP tunnels for peering – argh!!

```
interface Tunnel20
 ip address 10.1.8.1 255.255.255.0
 mpls bgp forwarding
 mpls ip
 tunnel source Loopback100
 tunnel destination 10.0.0.8
 passive-interface Tunnel20
IOU1#
IOU1#
IOU1#
IOU1#sh run | s router bgp
router bgp 100
 bgp log-neighbor-changes
 neighbor 10.0.0.8 remote-as 800
 neighbor 10.0.0.8 ebgp-multihop 5
 neighbor 10.0.0.8 update-source Loopback100
 neighbor 10.0.0.8 timers 10 20
 neighbor 10.1.8.8 remote-as 800
 !
 address-family vpnv4
  neighbor 10.0.0.8 activate
  neighbor 10.0.0.8 send-community extended
  neighbor 10.1.8.8 activate
  neighbor 10.1.8.8 send-community extended
 exit-address-family
```

**But at least we can send data though right?**

```
IOU1#ping vrf NINJA-NETWORKS ip 172.20.1.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.20.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/4/5 ms
IOU1#
```

Now we need to repeat the process between IOU8 and IOU9 – this requires creating another tunnel, with new IP ranges and BGP peer. Slightly annoying but… what are a few more lines of code right?

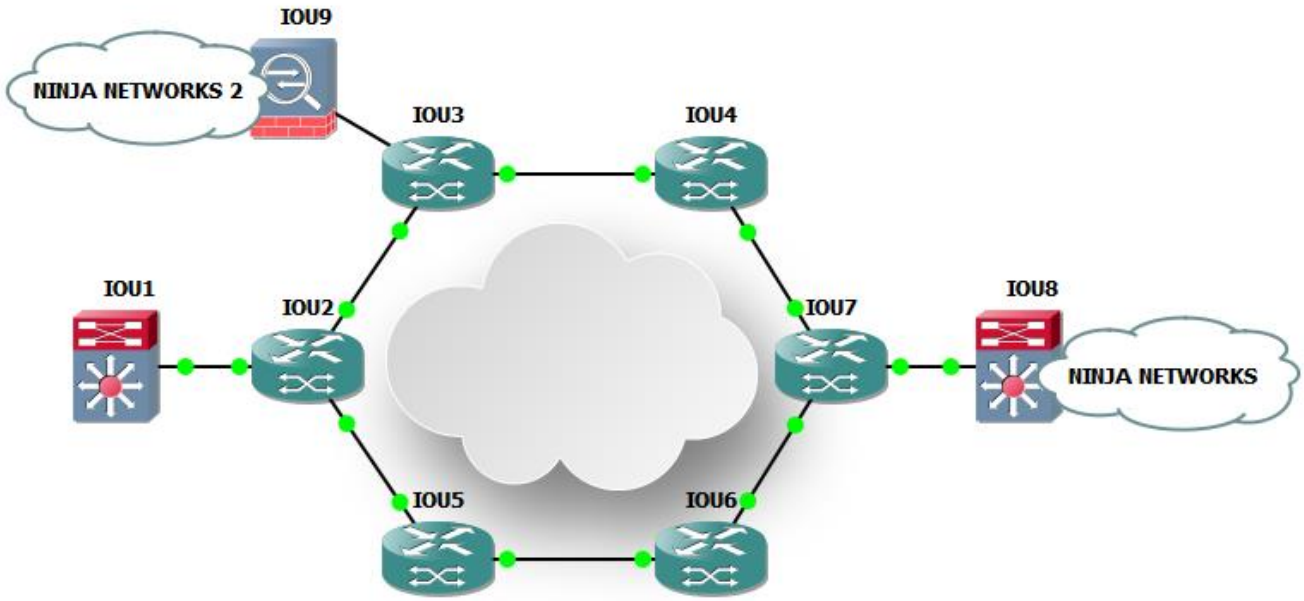
**IOU1** now has a requirement to reach **NINJA-NETWORKS 2** behind IOU9 **(169.254.X.X)**

IF we route to **NINJA NETWORKS 2** we will follow though the core to IOU8 and then IOU8 will follow the core back to IOU9 (presuming there is a GRE tunnel between IOU8 and IOU9 also)

- *Remember, IOU1 and IOU9 aren't peered and IOU8 is telling IOU1 it's the NEXT-HOP for the IOU9 routes*

But we don't want this; no one in their right mind would… unless it's a very specific case! It is clearly an inefficient path; IOU1 should just follow this path – **IOU1 →IOU2 → IOU3 → IOU9 → NINJA NETWORKS 2**

We can manipulate the NEXT-HOP value in the BGP update towards IOU1 to say use IOU9 as the next hop.




**On IOU8, we apply the following ROUTE-MAPS –** The following basically says, any routes going to IOU1 that were learnt from IOU9, change the NEXT-HOP to IOU9 and vice versa for 9 to 1

```
!
ip prefix-list NINE seq 5 permit 169.254.0.0/16 le 32
!
ip prefix-list ONE seq 5 permit 172.20.100.1/32
!
route-map IOU1-NEXT-HOP permit 10
 match ip address prefix-list NINE
 set ip next-hop 10.0.0.9
!
route-map IOU1-NEXT-HOP permit 20
!
route-map IOU9-NEXT-HOP permit 10
 match ip address prefix-list ONE
 set ip next-hop 10.0.0.1
!
route-map IOU9-NEXT-HOP permit 20
!
```

```
IOU8(config-router-af)#address-family vpnv4
IOU8(config-router-af)#
IOU8(config-router-af)#  neighbor 10.0.0.1 route-map IOU1-NEXT-HOP out
IOU8(config-router-af)#  neighbor 10.0.0.9 route-map IOU9-NEXT-HOP out
IOU8(config-router-af)#
```

**How does it look in IOU1?**

**NINJA NETWORKS 2** Routes are there with the new NEXT-HOP of **10.0.0.9** but ping fails? Of course it fails; we don't have a GRE tunnel between IOU1 and IOU9

See where this is going? This is no good, you'd have to build static tunnels for every possible conceivable scenario - Far from scalable

```
Gateway of last resort is not set

      169.254.0.0/27 is subnetted, 4 subnets
B        169.254.1.0 [20/0] via 10.0.0.9, 00:00:04
B        169.254.1.32 [20/0] via 10.0.0.9, 00:00:04
B        169.254.1.64 [20/0] via 10.0.0.9, 00:00:04
B        169.254.1.96 [20/0] via 10.0.0.9, 00:00:04
      172.20.0.0/16 is variably subnetted, 5 subnets, 2 masks
B        172.20.1.0/27 [20/0] via 10.1.8.8, 01:23:41
B        172.20.1.32/27 [20/0] via 10.1.8.8, 01:23:41
B        172.20.1.64/27 [20/0] via 10.1.8.8, 01:23:41
B        172.20.1.96/27 [20/0] via 10.1.8.8, 01:23:41
C        172.20.100.1/32 is directly connected, Loopback1
IOU1#
IOU1#
IOU1#
IOU1#ping vrf NINJA-NETWORKS ip 169.254.1.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 169.254.1.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
IOU1#
```

## Dynamic Multi-Point GRE tunnels to the rescue! (L3VPN over mGRE)

- L3VPN over mGRE (Multi-Point GRE) will allow the routers to build dynamic GRE tunnels to any router configured to use L3VPN tunnels
- Under the hood a GRE tunnel is still being created but unlike static GRE tunnels where both **SOURCE** and **DESTINATION** interfaces **MUST** be configured.
- MGRE configuration only specifies half the requirement - The tunnel **SOURCE** interface.
- It then picks whatever IP address is specified as the NEXT-HOP in the BGP RIB to dynamically create a **SOURCE + DESTINATION** binding and creates a tunnel

**On IOU1 and IOU9 configure both as below:**

```
!
l3vpn encapsulation ip L3VPN-ENCAP
 transport ipv4 source Loopback100
 !
```

```
!
route-map NEXT-HOP-MODIFICATION permit 10
 set ip next-hop encapsulate l3vpn L3VPN-ENCAP
!
```

```
router bgp 100
 bgp log-neighbor-changes
 neighbor 10.0.0.8 remote-as 800
 neighbor 10.0.0.8 ebgp-multihop 5
 neighbor 10.0.0.8 update-source Loopback100
 neighbor 10.0.0.8 timers 10 20
 !
 address-family vpnv4
  neighbor 10.0.0.8 activate
  neighbor 10.0.0.8 send-community extended
  neighbor 10.0.0.8 route-map NEXT-HOP-MODIFICATION in
 exit-address-family
 !
```

```
IOU1#ping vrf NINJA-NETWORKS ip 169.254.1.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 169.254.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/6 ms
IOU1#trace vrf NINJA-NETWORKS ip 169.254.1.1
Type escape sequence to abort.
Tracing the route to 169.254.1.1
VRF info: (vrf in name/id, vrf out name/id)
  1 169.254.1.1 5 msec 5 msec 5 msec
IOU1#
```

**And finally, the packet capture**

```
> Frame 12: 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits) on interface 0
> Ethernet II, Src: aa:bb:cc:00:02:10 (aa:bb:cc:00:02:10), Dst: aa:bb:cc:00:06:10 (aa:bb:cc:00:06:10)
> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.9
> Generic Routing Encapsulation (MPLS label switched packet)
> MultiProtocol Label Switching Header, Label: 110, Exp: 0, S: 1, TTL: 255
> Internet Protocol Version 4, Src: 172.20.100.1, Dst: 169.254.1.1
> Internet Control Message Protocol
```

If you're wondering why there is an MPLS header and label, it's because BGP still needs at least one label to identify the VPNv4 routes but luckily for us this is automatically generated and tunnelled in the GRE packet, we just can't escape MPLS after all ;-)

```
IOU1#show bgp vpnv4 unicast vrf NINJA-NETWORKS labels
   Network          Next Hop      In label/Out label
Route Distinguisher: 100:800 (NINJA-NETWORKS)
   172.20.1.0/27    10.0.0.8        nolabel/42
   172.20.1.32/27   10.0.0.8        nolabel/43
   172.20.1.64/27   10.0.0.8        nolabel/44
   172.20.1.96/27   10.0.0.8        nolabel/45
   172.20.100.1/32  0.0.0.0         33/nolabel(NINJA-NETWORKS)

IOU1#
```

# PART 6: USE CASE #3 – Ethernet over MPLS (EoMPLS )

So if you aren't bowing down to the MPLS gods by now, hopefully this will make you do so. MPLS can be leveraged further to support **Any Transport over MPLS (AToM)** this is the ability to tunnel traditional lower layer protocols, it can do this because MPLS sits just above the data-link layer.

AToM supports the following like-to-like transport types:

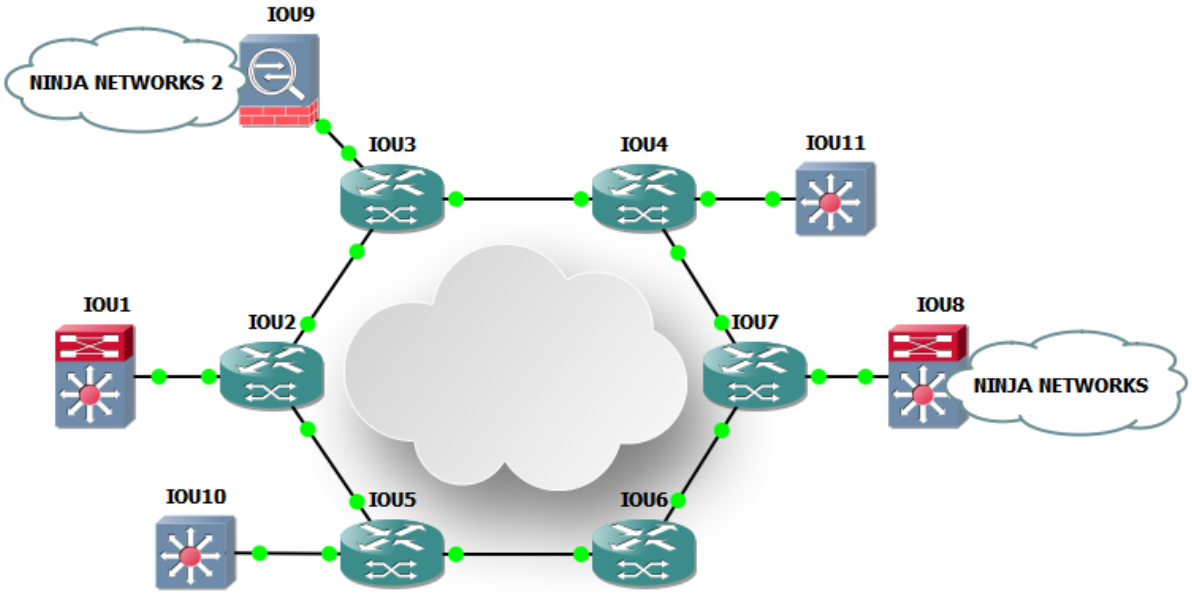
- ATM Adaptation Layer Type-5 (AAL5) over MPLS
- ATM Cell Relay over MPLS
- Ethernet over MPLS (VLAN and port modes)
- Frame Relay over MPLS
- PPP over MPLS
- High-Level Data Link Control (HDLC) over MPLS

Do some research if wanting to learn details of how this works, in this example we'll focus on the **EoMPLS** option of AToM

http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/any-transport-over-multiprotocol-label-switching-atom/prod_white_paper09186a00800a8442.html

**Back to our trusty topology -** We've added IOU10 and IOU11; these will be switches.

THE SCENARIO: There is a service requiring layer 2 connectivity across the datacentre, the software used must be on the same subnet, perhaps it's some legacy replication software.




**Let's add some basic config for IOU10**

On IOU10 add some vlans

```
VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Et0/0, Et0/1, Et0/2, Et0/3
                                                 Et1/0, Et1/1, Et1/2, Et1/3
                                                 Et2/0, Et2/1, Et2/2, Et2/3
                                                 Et3/0, Et3/1, Et3/2, Et3/3
10   BOBAZ                             active
20   ACIBASE                           active
30   JUNICOSYS                         active
```

A bit of VTP never hurt anyone

```
IOU10#sh vtp status
VTP Version capable             : 1 to 3
VTP version running             : 2
VTP Domain Name                 : JUNICO
VTP Pruning Mode                : Disabled
VTP Traps Generation            : Disabled
Device ID                       : aabb.cc00.0a00
Configuration last modified by 0.0.0.0 at 3-1-17 11:39:21
Local updater ID is 0.0.0.0 (no valid interface found)

Feature VLAN:
--------------
VTP Operating Mode                : Server
Maximum VLANs supported locally   : 1005
Number of existing VLANs          : 8
Configuration Revision            : 6
MD5 digest                        : 0x93 0xE1 0x49 0x29 0x93 0xB6 0xF8 0x46
                                    0x79 0x6C 0x12 0xE3 0x03 0x8D 0xA3 0x57
IOU10#
```

IP addresses on the SVIs

```
!
interface Vlan10
 description BOBBAZ
 ip address 10.10.11.10 255.255.255.0
!
interface Vlan20
 description ACIBASE
 ip address 20.10.11.10 255.255.255.0
!
interface Vlan30
 description JUNICOSYS
 ip address 30.10.11.10 255.255.255.0
!
```

IOU10 sees IOU5 as its neighbour via CDP

```
IOU10#show cdp neighbor
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater, P - Phone,
                  D - Remote, C - CVTA, M - Two-port Mac Relay

Device ID        Local Intrfce     Holdtme    Capability  Platform  Port ID
IOU5.junico.uk   Eth 0/3           166              R B   Linux Uni Eth 0/3
IOU10#
```

Now to work the magic on the core networks:

**IOU4**

```
!
interface Ethernet0/3
 description Link to IOU11
 no ip address
 xconnect 10.0.0.5 45 encapsulation mpls
!
```

**IOU5**

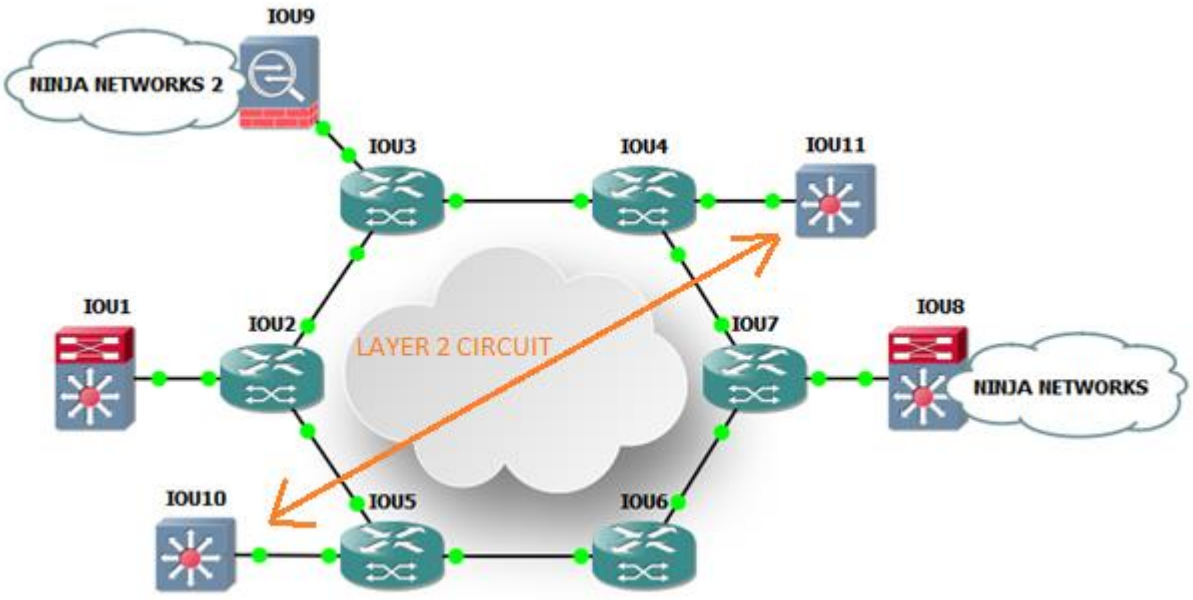
```
!
interface Ethernet0/3
 description Link to IOU10
 no ip address
 xconnect 10.0.0.4 45 encapsulation mpls
!
```

That's all there is to it (Connect to the peer loopback and Virtual Circuit identifier and encapsulate with MPLS)

**IOU10** can see **IOU11** via CDP instead of IOU5; this means LAYER 2 traffic is being tunnelled across the MPLS network to the other side… So yeah… we pay our service providers hundreds of pounds/dollars to add 2 lines of code what a rip off, 😐

```
IOU10#sh cdp neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater, P - Phone,
                  D - Remote, C - CVTA, M - Two-port Mac Relay

Device ID        Local Intrfce     Holdtme    Capability  Platform  Port ID
IOU11.junico.uk  Eth 0/3           136             R S I  Linux Uni Eth 0/3
IOU10#
```



Let’s verify other layer2 things works

- VTP
- Spanning-Tree
- HSRP

VTP is fine

```
IOU11#sh vtp status
VTP Version capable             : 1 to 3
VTP version running             : 2
VTP Domain Name                 : JUNICO
VTP Pruning Mode                : Disabled
VTP Traps Generation            : Disabled
Device ID                       : aabb.cc00.0b00
Configuration last modified by 0.0.0.0 at 3-1-17 11:39:21

Feature VLAN:
--------------
VTP Operating Mode                : Client
Maximum VLANs supported locally   : 1005
Number of existing VLANs          : 8
Configuration Revision            : 6
MD5 digest                        : 0x93 0xE1 0x49 0x29 0x93 0xB6 0xF8 0x46
                                    0x79 0x6C 0x12 0xE3 0x03 0x8D 0xA3 0x57
```

Vlans learnt via VTP

```
IOU11#show vlan br

VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Et0/0, Et0/1, Et0/2, Et1/0
                                                 Et1/1, Et1/2, Et1/3, Et2/0
                                                 Et2/1, Et2/2, Et2/3, Et3/0
                                                 Et3/1, Et3/2, Et3/3
10   BOBAZ                             active
20   ACIBASE                           active
30   JUNICOSYS                         active
```

ICMP works

```
IOU10#ping 10.10.11.11
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.10.11.11, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/5/5 ms
IOU10#
```

HSRP is happy

```
IOU10#show standby
Vlan10 - Group 10
  State is Standby
    4 state changes, last state change 00:12:26
  Virtual IP address is 10.10.11.1
  Active virtual MAC address is 0000.0c07.ac0a (MAC Not In Use)
    Local virtual MAC address is 0000.0c07.ac0a (v1 default)
  Hello time 3 sec, hold time 10 sec
    Next hello sent in 0.416 secs
  Preemption disabled
  Active router is 10.10.11.11, priority 100 (expires in 10.464 sec)
  Standby router is local
  Priority 100 (default 100)
  Group name is "hsrp-Vl10-10" (default)
IOU10#
```

**Finally, the packet capture – and what a beast of a capture it is!**

- Green header – Original Ethernet header
- Layered by a Pseudowire Purple header
- Layered by the L2VPN's MPLS header with label **60** (See Fig #2)
- Layered by the core MPLS header with label **302** that connects the loopbacks
- Red header – Outer Ethernet header

**Fig #1**

```
> Frame 136: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits) on interface 0
v Ethernet II, Src: aa:bb:cc:00:02:10 (aa:bb:cc:00:02:10), Dst: aa:bb:cc:00:06:10 (aa:bb:cc:00:06:10)
    > Destination: aa:bb:cc:00:06:10 (aa:bb:cc:00:06:10)
    > Source: aa:bb:cc:00:02:10 (aa:bb:cc:00:02:10)
      Type: MPLS label switched packet (0x8847)
> MultiProtocol Label Switching Header, Label: 302, Exp: 0, S: 0, TTL: 254
> MultiProtocol Label Switching Header, Label: 60, Exp: 0, S: 1, TTL: 255
> PW Ethernet Control Word
v Ethernet II, Src: aa:bb:cc:80:0a:00 (aa:bb:cc:80:0a:00), Dst: aa:bb:cc:80:0b:00 (aa:bb:cc:80:0b:00)
    > Destination: aa:bb:cc:80:0b:00 (aa:bb:cc:80:0b:00)
    > Source: aa:bb:cc:80:0a:00 (aa:bb:cc:80:0a:00)
      Type: 802.1Q Virtual LAN (0x8100)
> 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 10
> Internet Protocol Version 4, Src: 10.10.11.10, Dst: 10.10.11.11
> Internet Control Message Protocol
```

**Fig #2**

```
IOU5#sh mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing    Next Hop
Label      Label      or Tunnel Id     Switched      interface
60         No Label   l2ckt(45)        161134        Et0/3       point2point
61         57         10.0.0.9/32      0             Et0/2       10.2.5.2
62         56         10.0.0.8/32      0             Et0/2       10.2.5.2
63         55         10.0.0.7/32      0             Et0/2       10.2.5.2
```

There's way more to MPLS

- MPLS Traffic Engineering
- MPLS QoS
- MPLS-TP
- VPLS

The list goes on. But we've demonstrated some common uses in this article.

**I hope this has been clear and informative.**

GNS3 Lab files for this topology available at https://labs.junico.uk/labs/lab-7

Junico infrastructure – www.junico.uk