DMVPN Explained

56 Comments
Posted by Petr Lapukhov, 4xCCIE/CCDE in VPN

DMVPN stands for Dynamic Multipoint VPN and it is an effective solution for dynamic secure overlay networks. In short, DMVPN is combination of the following technologies:

1) Multipoint GRE (mGRE)
2) Next-Hop Resolution Protocol (NHRP)
4) Dynamic Routing Protocol (EIGRP, RIP, OSPF, BGP)
3) Dynamic IPsec encryption
5) Cisco Express Forwarding (CEF)

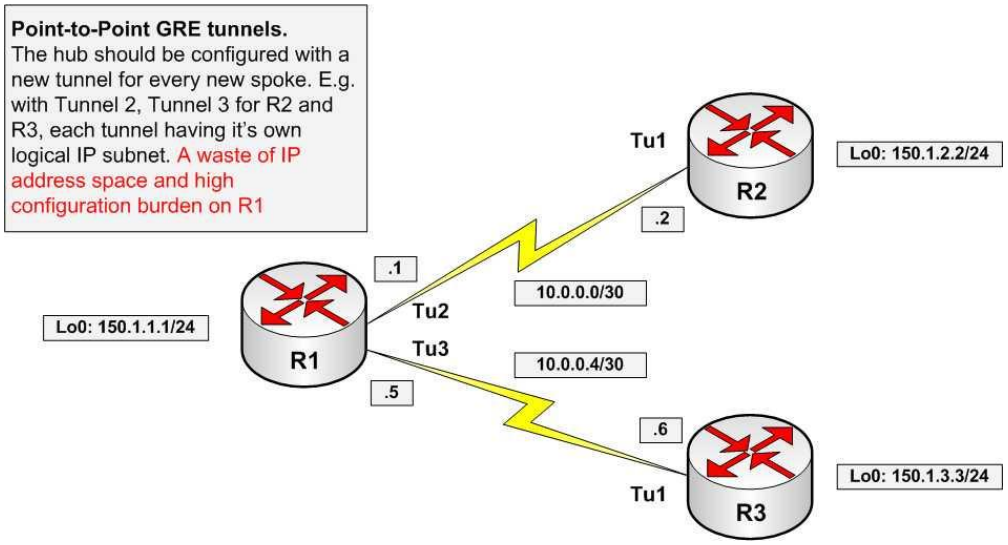
Assuming that reader has a general understanding of what DMVPN is and a solid understanding of IPsec/CEF, we are going to describe the role and function of each component in details. In this post we are going to illustrate two major phases of DMVPN evolution:

1) Phase 1 – Hub and Spoke (mGRE hub, p2p GRE spokes)
2) Phase 2 – Hub and Spoke with Spoke-to-Spoke tunnels (mGRE everywhere)

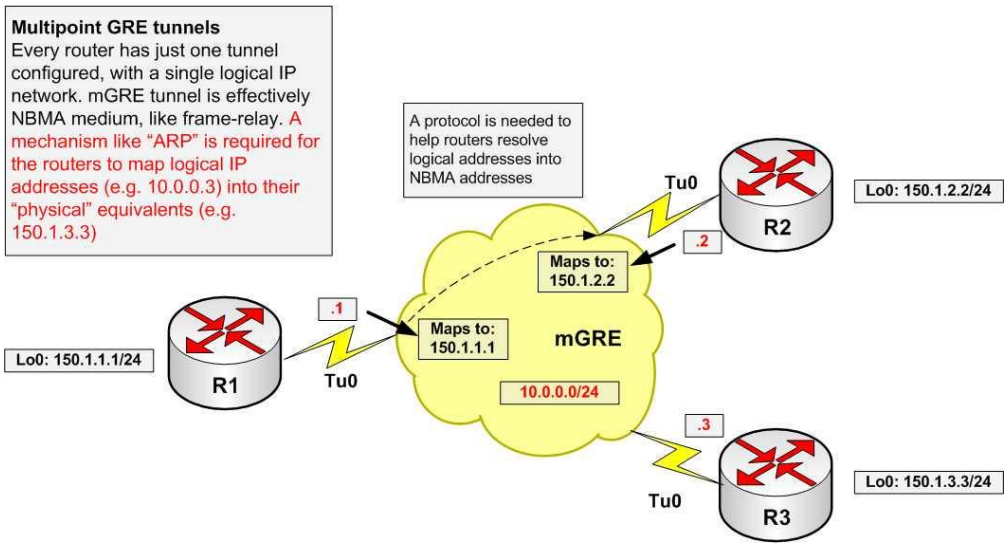
As for DMVPN Phase 3 – “Scalable Infrastructure”, a separate post is required to cover the subject. This is due to the significant changes made to NHRP resolution logic (NHRP redirects and shortcuts), which are better being illustrated when a reader has good understanding of first two phases. However, some hints about Phase 3 will be also provided in this post.
Note: Before we start, I would like to thank my friend Alexander Kitaev, for taking time to review the post and providing me with useful feedback.

Multipoint GRE

Let us start with the most basic building component of DMVPN – multipoint GRE tunnel. Classic GRE tunnel is point-to-point, but mGRE generalizes this idea by allowing a tunnel to have “multiple” destinations.



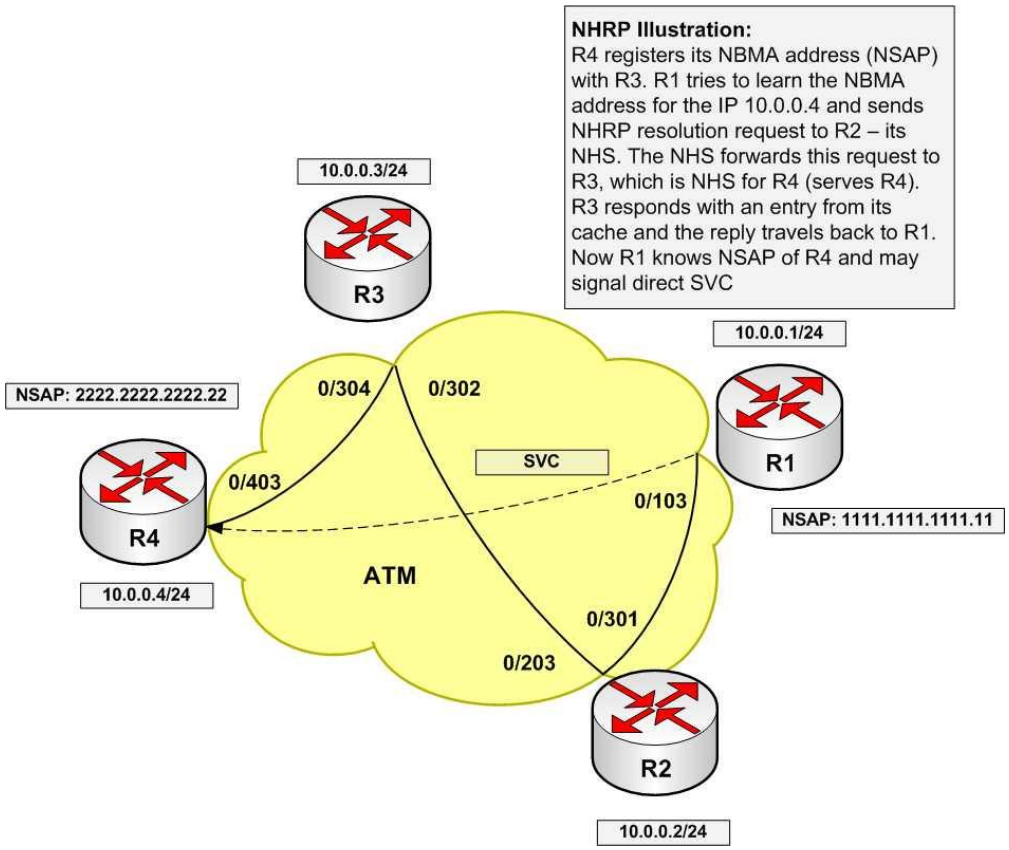
This may seem natural if the tunnel destination address is multicast (e.g. 239.1.1.1). The tunnel could be used to effectively distribute the same information (e.g. video stream) to multiple destinations on top of a multicast-enabled network. Actually, this is how mGRE is used for Multicast VPN implementation in Cisco IOS. However, if tunnel endpoints need to exchange unicast packets, special "glue" is needed to map tunnel IP addresses to "physical" or "real" IP addresses, used by endpoint routers. As we'll see later, this glue is called NHRP.



Note, that if you source multiple mGRE tunnels off the same interface (e.g. Loopback0) of a single router, then GRE can use special "multiplexor" field the tunnel header to differentiate them. This field is known as "tunnel key" and you can define it under tunnel configuration. As a matter of fact, up to IOS 12.3(14)T or 12.3(11)T3 the use of "tunnel key" was mandatory – mGRE tunnel would not come up, until the key is configured. Since the mentioned versions, you may configure a tunnel without the key. There were two reasons to remove the requirement. First, hardware ASICs of 6500 and 7600 platforms do not support mGRE tunnel-key processing, and thus the optimal switching performance on those platforms is penalized when you configure the tunnel key. Second, as we'll see later, DMVPN Phase 3 allows interoperation between different mGRE tunnels sharing the same NHRP network-id only when they have the same tunnel-key or have no tunnel-key at all (since this allows sending packets "between" tunnels).

Generic NHRP

Now let's move to the component that makes DMVPN truly dynamic – NHRP. The protocol has been defined quite some time ago in RFC 2332 (year 1998) to create a routing optimization scheme inside NBMA (non-broadcast multiple-access) networks, such as ATM, Frame-Relay and SMDS (anybody remembers this one nowadays? The general idea was to use SVC (switched virtual circuits) to create temporary shortcuts in non-fully meshed NBMA cloud. Consider the following schematic illustration, where IP subnet 10.0.0.0/24 overlays partial-meshed NBMA cloud. NHRP is similar in function to ARP, allowing resolving L3 to L2 addresses, but does that in more efficient manner, suitable for partially meshed NBMA clouds supporting dynamic layer 2 connections.



The following is simplified and schematic illustration of NHRP process. In the above topology, in order for R1 to reach R4, it must send packets over PVCs between R1-R2, R2-R3 and finally R3-R4. Suppose the NMBA cloud allows using SVC (Switched virtual circuits, dynamic paths) – then it would be more reasonable for R1 to establish SVC directly with R4 and send packets over the optimal way. However, this requires R1 to know NMBA address (e.g. ATM NSAP) associated with R4 to "place a call". Preferably, it would be better to make R1 learn R4 IP address to NSAP (NBMA address) mapping dynamically.

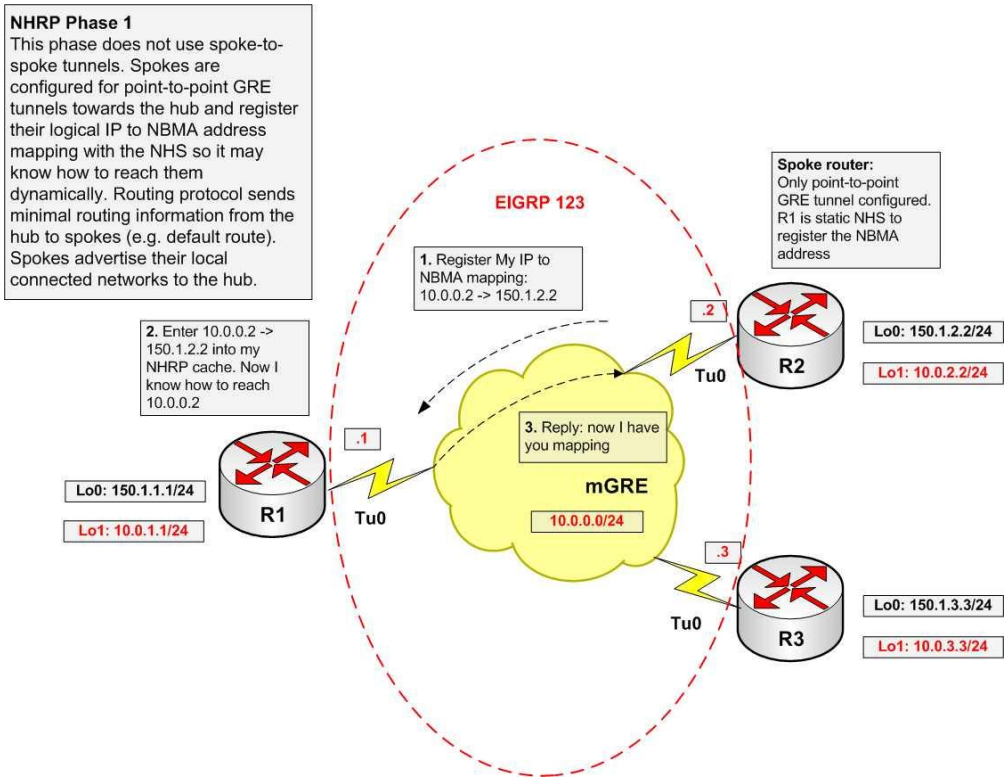
Now assume we enable NHRP on all NBMA interfaces in the network. Each router in topology acts as either NHC (Next-Hop Client) or NHS (Next-Hop Server). One of the functions of NHC is to register with NHS its IP address mapped to NBMA Layer 2 address (e.g. ATM NSAP address). To make registration possible, you configure each NHC with the IP address of at least one NHS.

In turn, NHS acts as a database agent, storing all registered mappings, and replying to NHC queries. If NHS does not have a requested entry in its database, it can forward packet to another NHS to see if it has the requested association. Note that a router may act as a Next-Hop server and client at the same time. Back to the diagram, assume that R2 and R3 are NHSes, R1 and R4 are NHCs. Further, assume R4 is NHC and registers its IP to NBMA address mapping with R4 and R1 thinks R2 is the NHS. Both R2 and R3 treat each other as NHS. When R1 wants to send traffic to R4 (next-hop 10.0.0.4), it tries to resolve 10.0.0.4 by sending NHRP resolution request to R2 – the configured NHS. In turn, R2 will forward request to R3, since it has no local information.

Obviously, modern networks tend not to use ATM/SMDS and Frame-Relay SVC too much, but one can adopt NHRP to work with “simulated NBMA” networks, such as mGRE tunnels. The NBMA layer maps to “physical” underlying network while mGRE VPN is the “logical” network (tunnel internal IP addressing). In this case, mGRE uses NHRP for mapping “logical” or “tunnel inside” IP addresses to “physical” or real IP addresses. Effectively, NHRP perform the “glue” function described above, allowing mGRE endpoints discovering each other’s real IP address. Since NHRP defines a server role, it’s natural to have mGRE topology lay out in Hub-and-Spoke manner (or combination of hubs and spokes, in more advanced cases). Let’s see some particular scenarios to illustrate NHRP functionality with mGRE.

NHRP Phase 1

With NHRP Phase 1 mGRE uses NHRP to inform the hub about dynamically appearing spokes. Initially, you configure every spoke with the IP address of the hub as the NHS server. However, the spoke’s tunnel mode is GRE (regular point-to-point) tunnel with a fixed destination IP that equals to the physical address of the hub. The spokes can only reach the hub and only get to other spoke networks across the hub. The benefit of Phase 1 is simplified hub router configuration, which does not require static NHRP mapping for every new spoke.



As all packets go across the hub, almost any dynamic routing protocol would help with attaining reachability. The hub just needs to advertise a default route to spokes, while spokes should advertise their subnets dynamically to the hub. Probably it makes sense to run EIGRP and summarize all subnets to 0.0.0.0/0 on the hub, effectively sending a default route to all spokes (if the spokes do not use any other default route, e.g. from their ISPs). Configure spokes as EIGRP stubs and advertise their respective connected networks. RIP could be set up in similar manner, by simply configuring GRE tunnels on spokes as passive interfaces. Both EIGRP and RIP require split-horizon disabled on the hub mGRE interface in order to exchange subnets spoke to spoke. As for OSPF, the optimal choice would be using point-to-multipoint network type on all GRE and mGRE interfaces. In addition to that, configure **ip ospf database filter-all out** on the hub and set up static default routes via tunnel interfaces on the spokes (or static specific routes for corporate networks).

Here is a sample configuration. The detailed explanation of NHRP commands and “show” commands output follows the example.

mGRE + NHRP Phase 1 + EIGRP

```
R1:
!
! Hub router
!
router eigrp 123
 no auto-summary
 network 10.0.0.0 0.255.255.255
!
```

```
! Tunnel source
!
interface Loopback0
 ip address 150.1.1.1 255.255.255.0
!
! VPN network
!
interface Loopback 1
 ip address 10.0.1.1 255.255.255.0
!
! mGRE tunnel
!
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 no ip redirects
 ip nhrp authentication cisco
 ip nhrp map multicast dynamic
 ip nhrp network-id 123
 no ip split-horizon eigrp 123
 ip summary-address eigrp 123 0.0.0.0 0.0.0.0 5
 tunnel source Loopback0
 tunnel mode gre multipoint
 tunnel key 123
```

R2:

```
!
! Spoke Router
!
router eigrp 123
 no auto-summary
 network 10.0.0.0 0.255.255.255
 eigrp stub connected
!
interface Loopback0
 ip address 150.1.2.2 255.255.255.0
!
interface Loopback 1
 ip address 10.0.2.2 255.255.255.0
!
! GRE tunnel
!
interface Tunnel0
 ip address 10.0.0.2 255.255.255.0
 ip nhrp authentication cisco
 ip nhrp map multicast 150.1.1.1
 ip nhrp map 10.0.0.1 150.1.1.1
```

```
 ip nhrp nhs 10.0.0.1
 ip nhrp network-id 123
 ip nhrp registration timeout 30
 ip nhrp holdtime 60
 tunnel source Loopback0
 tunnel destination 150.1.1.1
 tunnel key 123

R3:
!
! Spoke Router
!
router eigrp 123
 no auto-summary
 network 10.0.0.0 0.255.255.255
 eigrp stub connected
!
interface Loopback0
 ip address 150.1.3.3 255.255.255.0
!
interface Loopback 1
 ip address 10.0.3.3 255.255.255.0
!
interface Tunnel0
 ip address 10.0.0.3 255.255.255.0
 ip nhrp authentication cisco
 ip nhrp map multicast 150.1.1.1
 ip nhrp map 10.0.0.1 150.1.1.1
 ip nhrp nhs 10.0.0.1
 ip nhrp network-id 123
 ip nhrp registration timeout 30
 ip nhrp holdtime 60
 tunnel source Loopback0
 tunnel destination 150.1.1.1
 tunnel key 123
```

Note that only the hub tunnel uses mGRE encapsulation, and spokes use regular point-to-point GRE tunnels. Now, let's look at the NHRP commands used in the example above. The most basic command **ip nhrp map [Logical IP] [NBMA IP]** – creates a static binding between a logical IP address and NBMA IP address. Since mGRE is treated by NHRP as NMBA medium, logical IP corresponds to the IP address "inside" a tunnel ("inner") and the NBMA IP address corresponds to the "outer" IP address (the IP address used to source a tunnel). (From now on, we are going to call "inner" IP address and simply "IP address" or "logical IP address" and the "outer" IP address as "NBMA address" or "physical IP address"). The use of static NHRP mappings is to "bootstrap" information for the spokes to reach the logical IP address of the hub. The next command is **ip nhrp map multicast *dynamic*|[StaticIP]** and its purpose is the same

as “frame-relay map… **broadcast**”. The command specifies the list of destination that will receive the multicast/broadcast traffic originated from this router. Spokes map multicasts to the static NBMA IP address of the hub, but hub maps multicast packets to the “dynamic” mappings – that is, the hub replicates multicast packets to all spokes registered via NHRP. Mapping multicasts is important in order to make dynamic routing protocol establish adjacencies and exchange update packets. The **ip nhrp nhs [ServerIP]**command configures NHRP client with the IP address of its NHRP server. Note the “ServerIP” is the logical IP address of the hub (inside the tunnel) and therefore spokes need the static NHRP mappings in order to reach it. The spokes use the NHS to register their logical IP to NBMA IP associations and send NHRP resolution request. (However, in this particular scenarios, the spokes will **not** send any NHRP Resolutions Requests, since they use directed GRE tunnels – only registration requests will be sent). The commands **ip nhrp network-id** and **ip nhrp authentication [Key]** identify and authenticate the logical NHRP network. The [ID] and the [Key] must match on all routers sharing the same GRE tunnel. It is possible to split an NBMA medium into multiple NHRP networks, but this is for advanced scenarios. As for the authentication, it’s a simple plain-text key sent in all NHRP messages. While the “network-id” is mandatory in order for NHRP to work, you may omit the authentication. Next command is**ip nhrp holdtime** that specifies the hold-time value set in NHRP registration requests. The NHS will keep the registration request cached for the duration of the hold-time, and then, if no registration update is receive, will time it out. The NHS will also send the same hold-time in NHRP resolution responses, if queried for the respective NHRP association. Note that you configure the **ip nhrp holdtime** command on spokes, and spoke will send registration requests every 1/3 of the hold-time seconds. However, if you also configure the **ip nhrp registration timeout [Timeout]** on a spoke, the NHRP registration requests will be sent every [Timeout] sends, not 1/3 of the configured hold-time. The hold-time value sent in NHRP Registration Requests will remain the same, of course.

Now let’s move to the show commands. Since it’s only the hub that uses the NHRP dynamic mappings to resolve the spokes NBMA addresses, it is useful to observe R1 NHRP cache:

```
Rack1R1#show ip nhrp detail
10.0.0.2/32 via 10.0.0.2, Tunnel0 created 00:16:59, expire 00:00:30
  Type: dynamic, Flags: authoritative unique registered used
  NBMA address: 150.1.2.2
10.0.0.3/32 via 10.0.0.3, Tunnel0 created 00:11:34, expire 00:00:55
  Type: dynamic, Flags: authoritative unique registered used
  NBMA address: 150.1.3.3
```

As we can see, the logical IP “10.0.0.2” maps to NBMA address “150.1.2.2” and the logical IP 10.0.0.3 maps to NBMA address 150.1.3.3. The “authoritative” flag means that the NHS has learned about the NHRP mapping directly from a registration request (the NHS “serves” the particular NHC). The “unique” flag means that the NHRP registration request had the same “unique” flag set. The use of this flag is to prevent duplicate NHRP mappings in cache. If unique mapping for a particular logical IP is already in the NHRP cache and another NHC tries to register the same logical IP with the NHS, the server will reject the registration, until the unique entry expires. Note that by default IOS routers set this flag in registration request, and this can be

disabled by using **ip nhrp registration no-unique** command on a spoke. Sometimes this may be needed when spoke change its NBMA IP address often and needs to re-register a new mapping with the hub. The last flag, called “used” flag, means that the router uses the NHRP entry to switch IP packets. We will discuss the meaning of this flag in NRHP process switching section below. Also, note the “expires” field, which is a countdown timer, started from the “holdtime” specified in the Registration Request packet.

Let’s see the NHRP registration and reply process flows on the NHS.

```
Rack1R1#debug nhrp
NHRP protocol debugging is on

Rack1R1#debug nhrp packet
NHRP activity debugging is on
```

First, R3 tries to register its Logical IP to NBMA IP mapping with the hub. Note the specific NHRP packet format, split in three parts.

1) (F) – fixed part. Specifies the version, address family (afn) and protocol type (type) for resolution, as well as subnetwork layer (NBMA) type and length (shtl and sstl). Note that “shtl” equals 4, which is the length of IPv4 address in bytes, and “sstl” is for “subaddress” field which is not used with IPv4.

2) (M) – mandatory header part. Specifies some flags, like “unique” flag and the “Request ID”, which is used to track request/responses. Also includes are the source NBMA address (tunnel source in GRE/mGRE) and the source/destination protocol IP addresses. Destination IP address is the logical IP address of the hub and the source IP address is the logical IP address of the spoke. Using this information hub may populate the spoke logical IP address to NBMA IP address mapping.

3) (C-1) – CIE 1, which stands for “Client Information Element” field. While it’s not used in the packets below, in more advanced scenarios explored later, we’ll see this filed containing the information about networks connected to requesting/responding routers.

Also note the NAT-check output, which is Cisco’s extension used to make NHRP work for routers that tunnel from behind the NAT.

```
NHRP: Receive Registration Request via Tunnel0 vrf 0, packet size: 81
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "unique", reqid: 26
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.0.1
 (C-1) code: no error(0)
       prefix: 255, mtu: 1514, hd_time: 60
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
```

```
NHRP: netid_in = 123, to_us = 1
NHRP: NAT-check: matched destination address 150.1.3.3
NHRP: Tu0: Found and skipping dynamic multicast mapping  NBMA: 150.1.3.3
NHRP: Attempting to send packet via DEST 10.0.0.3
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.3.3
```

After processing the request, the router responds with NHRP Registration Reply. Note that the (M) header did not change, just the source and destination logical IP address of the packet are reversed. (R1->R3)

```
NHRP: Send Registration Reply via Tunnel0 vrf 0, packet size: 101
 src: 10.0.0.1, dst: 10.0.0.3
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "unique", reqid: 26
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.0.1
 (C-1) code: no error(0)
       prefix: 255, mtu: 1514, hd_time: 60
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: 101 bytes out Tunnel0
```

Now the NHS receives the Registration Request from R2, and adds the corresponding entry in its NHRP cache

```
NHRP: Receive Registration Request via Tunnel0 vrf 0, packet size: 81
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "unique", reqid: 38
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.0.1
 (C-1) code: no error(0)
       prefix: 255, mtu: 1514, hd_time: 60
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: netid_in = 123, to_us = 1
NHRP: NAT-check: matched destination address 150.1.2.2
NHRP: Tu0: Found and skipping dynamic multicast mapping  NBMA: 150.1.2.2
NHRP: Attempting to send packet via DEST 10.0.0.2
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.2.2

NHRP: Send Registration Reply via Tunnel0  vrf 0, packet size: 101
 src: 10.0.0.1, dst: 10.0.0.2
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "unique", reqid: 38
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.0.1
 (C-1) code: no error(0)
```

```
       prefix: 255, mtu: 1514, hd_time: 60
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: 101 bytes out Tunnel0
```

We see how NRHP Phase 1 works now. The spokes register their associations with the hub via NHRP and the hub learns their NBMA addresses dynamically. At the same time, spokes use point-to-point tunnels to speak to the hub and reach each other. Note that EIGRP is not the only protocol suitable for use with NHRP Phase 1. OSPF is also a viable solution, thank to **point-to-multipoint** network type and **database filter-all out** command. See the example below for OSPF configuration with NHRP Phase 1:

mGRE + NHRP Phase 1 + OSPF

```
R1:
!
! Hub router
!
router ospf 123
 router-id 10.0.0.1
 network 10.0.0.0 0.255.255.255 area 0
!
interface Loopback0
 ip address 150.1.1.1 255.255.255.0
!
interface Loopback 1
 ip address 10.0.1.1 255.255.255.0
!
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 no ip redirects
 ip nhrp authentication cisco
 ip nhrp map multicast dynamic
 ip nhrp network-id 123
 tunnel source Loopback0
 tunnel mode gre multipoint
 tunnel key 123
 ip ospf network point-to-multipoint
 ip ospf database-filter all out

R2:
!
! Spoke Router
!
router ospf 123
 network 10.0.0.0 0.255.255.255 area 0
 router-id 10.0.0.2
```

```
!
interface Loopback0
 ip address 150.1.2.2 255.255.255.0
!
interface Loopback 1
 ip address 10.0.2.2 255.255.255.0
!
interface Tunnel0
 ip address 10.0.0.2 255.255.255.0
 ip nhrp authentication cisco
 ip nhrp map multicast 150.1.1.1
 ip nhrp map 10.0.0.1 150.1.1.1
 ip nhrp nhs 10.0.0.1
 ip nhrp network-id 123
 ip nhrp registration timeout 30
 ip nhrp holdtime 60
 tunnel source Loopback0
 tunnel destination 150.1.1.1
 tunnel key 123
 ip ospf network point-to-multipoint
!
ip route 0.0.0.0 0.0.0.0 Tunnel0
```

R3:

```
!
! Spoke Router
!
router ospf 123
 network 10.0.0.0 0.255.255.255 area 0
 router-id 10.0.0.3
!
interface Loopback0
 ip address 150.1.3.3 255.255.255.0
!
interface Loopback 1
 ip address 10.0.3.3 255.255.255.0
!
interface Tunnel0
 ip address 10.0.0.3 255.255.255.0
 ip nhrp authentication cisco
 ip nhrp map multicast 150.1.1.1
 ip nhrp map 10.0.0.1 150.1.1.1
 ip nhrp nhs 10.0.0.1
 ip nhrp network-id 123
 ip nhrp registration timeout 30
 ip nhrp holdtime 60
```

```
 tunnel source Loopback0
 tunnel destination 150.1.1.1
 tunnel key 123
 ip ospf network point-to-multipoint
!
ip route 0.0.0.0 0.0.0.0 Tunnel0
```

As we said, the main benefit of using NHRP Phase 1 is simplified configuration on the hub router. Additionally, spoke routers receive minimal routing information (it's either summarized or filtered on the hub) and are configured in uniform manner. In most simple case, spoke routers could be configured without any NHRP, by simply using point-to-point GRE tunnels. This scenario requires the hub to create a static NHRP mapping for every spoke. For example:

mGRE + NHRP Phase 1 + OSPF + Static NHRP mappings

```
R1:
!
! Hub router
!
router ospf 123
 router-id 10.0.0.1
 network 10.0.0.0 0.255.255.255 area 0
!
interface Loopback0
 ip address 150.1.1.1 255.255.255.0
!
interface Loopback 1
 ip address 10.0.1.1 255.255.255.0
!
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 no ip redirects
 ip nhrp authentication cisco
 ip nhrp map 10.0.0.2 150.1.2.2
 ip nhrp map 10.0.0.3 150.1.3.3
 ip nhrp map multicast 150.1.2.2
 ip nhrp map multicast 150.1.3.3
 ip nhrp network-id 123
 tunnel source Loopback0
 tunnel mode gre multipoint
 tunnel key 123
 ip ospf network point-to-multipoint
 ip ospf database-filter all out

R2:
!
! Spoke Router
```

```
!
router ospf 123
 network 10.0.0.0 0.255.255.255 area 0
 router-id 10.0.0.2
!
interface Loopback0
 ip address 150.1.2.2 255.255.255.0
!
interface Loopback 1
 ip address 10.0.2.2 255.255.255.0
!
interface Tunnel0
 ip address 10.0.0.2 255.255.255.0
 tunnel source Loopback0
 tunnel destination 150.1.1.1
 tunnel key 123
 ip ospf network point-to-multipoint
!
ip route 0.0.0.0 0.0.0.0 Tunnel0
```

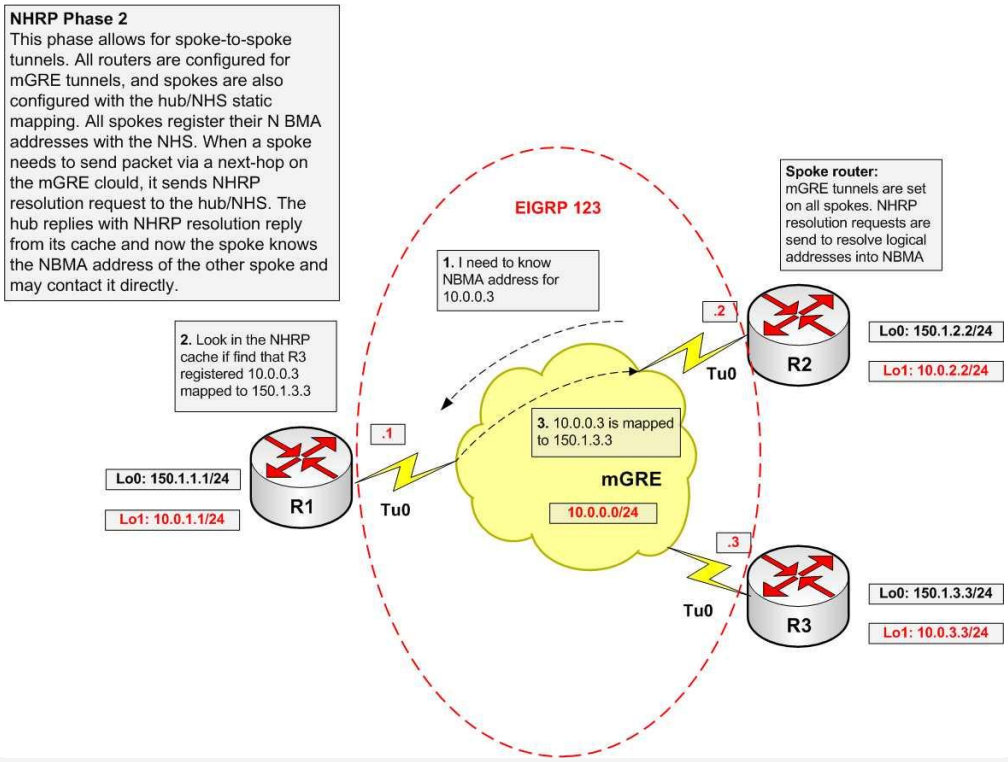
R3:

```
!
! Spoke Router
!
router ospf 123
 network 10.0.0.0 0.255.255.255 area 0
 router-id 10.0.0.3
!
interface Loopback0
 ip address 150.1.3.3 255.255.255.0
!
interface Loopback 1
 ip address 10.0.3.3 255.255.255.0
!
interface Tunnel0
 ip address 10.0.0.3 255.255.255.0
 tunnel source Loopback0
 tunnel destination 150.1.1.1
 tunnel key 123
 ip ospf network point-to-multipoint
!
ip route 0.0.0.0 0.0.0.0 Tunnel0
```

The disadvantage of NHRP Phase 1 is the inability to establish spoke-to-spoke shortcut tunnels. NHRP Phase 2 resolves this issue and allows for spoke-to-spoke tunnels. To better understand the second phase, we first need to find out how NHRP interacts with CEF – the now default IP

switching method on most Cisco routers. Consider the topology and example configuration that follows. See the detailed breakdown after the configuration.

mGRE + NHRP Phase 2 + EIGRP




```
R1:
!
! Hub router
!
router eigrp 123
 no auto-summary
 network 10.0.0.0 0.255.255.255
!
interface Loopback0
 ip address 150.1.1.1 255.255.255.0
!
interface Loopback 1
 ip address 10.0.1.1 255.255.255.0
!
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 no ip redirects
 ip nhrp authentication cisco
 ip nhrp map multicast dynamic
 ip nhrp network-id 123
 no ip split-horizon eigrp 123
 no ip next-hop-self eigrp 123
```

```
 tunnel source Loopback0
 tunnel mode gre multipoint
 tunnel key 123
```

R2:

```
!
! Spoke Router
!
router eigrp 123
 no auto-summary
 network 10.0.0.0 0.255.255.255
 eigrp stub connected
!
interface Loopback0
 ip address 150.1.2.2 255.255.255.0
!
interface Loopback 1
 ip address 10.0.2.2 255.255.255.0
!
interface Tunnel0
 ip address 10.0.0.2 255.255.255.0
 ip nhrp authentication cisco
 ip nhrp map multicast 150.1.1.1
 ip nhrp map 10.0.0.1 150.1.1.1
 ip nhrp nhs 10.0.0.1
 ip nhrp network-id 123
 ip nhrp registration timeout 30
 ip nhrp holdtime 60
 tunnel source Loopback0
 tunnel mode gre multipoint
 tunnel key 123
```

R3:

```
!
! Spoke Router
!
router eigrp 123
 no auto-summary
 network 10.0.0.0 0.255.255.255
 eigrp stub connected
!
interface Loopback0
 ip address 150.1.3.3 255.255.255.0
!
interface Loopback 1
 ip address 10.0.3.3 255.255.255.0
```

```
!
interface Tunnel0
 ip address 10.0.0.3 255.255.255.0
 ip nhrp authentication cisco
 ip nhrp map multicast 150.1.1.1
 ip nhrp map 10.0.0.1 150.1.1.1
 ip nhrp nhs 10.0.0.1
 ip nhrp network-id 123
 ip nhrp registration timeout 30
 ip nhrp holdtime 60
 tunnel source Loopback0
 tunnel mode gre multipoint
 tunnel key 123
```

Note that both spokes use mGRE tunnel encapsulation mode, and the hub sets the originating router next-hop IP address in “reflected” EIGRP updates (by default EIGRP sets the next-hop field to “0.0.0.0” – that is, to self). By the virtue of the EIGRP configuration, the subnet “10.0.2.0/24” (attached to R2) reaches to R3 with the next-hop IP address of “10.0.0.2” (R2). It is important that R3 learns “10.0.2.0/24” with the next hop of R2 logical IP address. As we see later, this is the key to trigger CEF next-hop resolution. The mGRE encapsulation used on spokes will trigger NHRP resolutions since now this is NBMA medium. Now, assuming that traffic to 10.0.2.0/24 does not flow yet, check the routing table entry for 10.0.2.2 and the CEF entries for the route and its next-hop:

```
Rack1R3#show ip route 10.0.2.2
Routing entry for 10.0.2.0/24
  Known via "eigrp 123", distance 90, metric 310172416, type internal
  Redistributing via eigrp 123
  Last update from 10.0.0.2 on Tunnel0, 00:09:55 ago
  Routing Descriptor Blocks:
  * 10.0.0.2, from 10.0.0.1, 00:09:55 ago, via Tunnel0
      Route metric is 310172416, traffic share count is 1
      Total delay is 1005000 microseconds, minimum bandwidth is 9 Kbit
      Reliability 255/255, minimum MTU 1472 bytes
      Loading 1/255, Hops 2

Rack1R3#show ip cef 10.0.2.2
10.0.2.0/24, version 48, epoch 0
0 packets, 0 bytes
  via 10.0.0.2, Tunnel0, 0 dependencies
    next hop 10.0.0.2, Tunnel0
    invalid adjacency

Rack1R3#show ip cef 10.0.0.2
10.0.0.0/24, version 50, epoch 0, attached, connected
0 packets, 0 bytes
  via Tunnel0, 0 dependencies
```

```
    valid glean adjacency
```

Note that CEF prefix for “10.0.2.0/24” is invalid (but not “glean”), since “10.0.0.2” has not yet been resolved. The CEF prefix for “10.0.0.2” has “glean” adjacency, which means the router needs to send an NHRP resolution request to map the logical IP to NBMA address. Therefore, with CEF switching, NHRP resolution requests are only sent for “next-hop” IP addresses, and never for the networks (e.g. 10.0.2.0/24) themselves (the process-switching does resolve any prefix as we’ll see later). Go ahead and ping from R3 to “10.0.3.3” and observe the process:

```
Rack1R3#ping 10.0.2.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 36/80/180 ms
```

Check the mappings on the hub router. The only two entries registered are the VPN IP addresses of R2 and R3, together with the respective NBMA IP addresses. Note the “expire” field, which, as mentioned above, counts the time for the entry to expire based on the “holdtime” settings of the registering router’s interface. Later we will see how CEF uses this countdown timer to refresh or delete CEF entries for the next-hop IP address

```
Rack1R1#show ip nhrp
10.0.0.2/32 via 10.0.0.2, Tunnel0 created 00:16:33, expire 00:00:43
  Type: dynamic, Flags: authoritative unique registered
  NBMA address: 150.1.2.2
  Requester: 10.0.0.3 Request ID: 798
10.0.0.3/32 via 10.0.0.3, Tunnel0 created 00:16:34, expire 00:00:51
  Type: dynamic, Flags: authoritative unique registered
  NBMA address: 150.1.3.3
  Requester: 10.0.0.2 Request ID: 813

Check the mappings on R2 (note that R2 now has mapping for R3’s
next-hop associated with its NBMA IP address)

Rack1R2#show ip nhrp
10.0.0.1/32 via 10.0.0.1, Tunnel0 created 00:14:52, never expire
  Type: static, Flags: authoritative used
  NBMA address: 150.1.1.1
10.0.0.2/32 via 10.0.0.2, Tunnel0 created 00:05:49, expire 00:00:10
  Type: dynamic, Flags: router authoritative unique local
  NBMA address: 150.1.2.2
    (no-socket)
10.0.0.3/32 via 10.0.0.3, Tunnel0 created 00:00:30, expire 00:00:29
  Type: dynamic, Flags: router used
  NBMA address: 150.1.3.3
```

```
The same command output on R3 is symmetric to the output on R2:

Rack1R3#show ip nhrp
10.0.0.1/32 via 10.0.0.1, Tunnel0 created 00:14:00, never expire
  Type: static, Flags: authoritative used
  NBMA address: 150.1.1.1
10.0.0.2/32 via 10.0.0.2, Tunnel0 created 00:00:05, expire 00:00:54
  Type: dynamic, Flags: router
  NBMA address: 150.1.2.2
10.0.0.3/32 via 10.0.0.3, Tunnel0 created 00:01:46, expire 00:00:13
  Type: dynamic, Flags: router authoritative unique local
  NBMA address: 150.1.3.3
    (no-socket)
```

Now check the CEF entry for R2's next-hop IP address on R3:

```
Rack1R3#sh ip cef 10.0.0.2
10.0.0.2/32, version 65, epoch 0, connected
0 packets, 0 bytes
  via 10.0.0.2, Tunnel0, 0 dependencies
    next hop 10.0.0.2, Tunnel0
    valid adjacency
```

The CEF entry for "10.0.0.2" is now valid, since NHRP mapping entry is present. If the next-hop for the prefix "10.0.2.0/24" was pointing toward the hub (R1) (e.g. if the hub was using the default **ip next-hop-self eigrp 123**) then the NHRP lookup will not be triggered, and cut-through NHRP entry will not be installed. Let's see the debugging command output on R1, R2 and R3 to observe how the routers collectively resolve the next-hop IP addresses when R3 pings R1:

```
Rack1R1#debug nhrp
NHRP protocol debugging is on
Rack1R1#debug nhrp packet
NHRP activity debugging is on

Rack1R2#debug nhrp
NHRP protocol debugging is on
Rack1R2#debug nhrp packet
NHRP activity debugging is on

Rack1R3#debug nhrp
NHRP protocol debugging is on
Rack1R3#debug nhrp packet
NHRP activity debugging is on
```

It all starts when R3 tries to route a packet to "10.0.2.2" and finds out it has "glean" adjacency for its next-hop of "10.0.0.2". Then, R3 attempt to send NHRP resolution request directly to R2, but fails since R2 NMBA address is unknown. At the same time, the original data packet (ICMP echo) follows to R2 across the hub (R1).

```
Rack1R3#
NHRP: MACADDR: if_in null netid-in 0 if_out Tunnel0 netid-out 123
NHRP: Checking for delayed event 0.0.0.0/10.0.0.2 on list (Tunnel0).
NHRP: No node found.
NHRP: Sending packet to NHS 10.0.0.1 on Tunnel0
NHRP: Checking for delayed event 0.0.0.0/10.0.0.2 on list (Tunnel0).
NHRP: No node found.
NHRP: Attempting to send packet via DEST 10.0.0.2
NHRP: Send Resolution Request via Tunnel0 vrf 0, packet size: 81
 src: 10.0.0.3, dst: 10.0.0.2
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 994
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.0.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: Encapsulation failed for destination 10.0.0.2 out Tunnel0
```

Next, R3 tries to send resolution request to the NHS, which is R1. The resolution request contains information about source NBMA address of R3, and source protocol (logical IP) addresses of R3 and R2.

```
Rack1R3#
NHRP: Attempting to send packet via NHS 10.0.0.1
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.1.1
NHRP: Send Resolution Request via Tunnel0 vrf 0, packet size: 81
 src: 10.0.0.3, dst: 10.0.0.1
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 994
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.0.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: 81 bytes out Tunnel0
```

The Resolution Request from R3 arrives to NHS. In essence, R3 tries to resolve the “glean” CEF adjacency using NHRP the same way it uses ARP on Ethernet. Note that request only mentions logical IP addresses of R3 (“10.0.0.3”) and R2 (“10.0.0.2”) and NBMA address of R3.

```
Rack1R1#
NHRP: Receive Resolution Request via Tunnel0 vrf 0, packet size: 81
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 994
```

```
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.0.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: netid_in = 123, to_us = 0
NHRP: NAT-check: matched destination address 150.1.3.3
NHRP: nhrp_rtlookup yielded Tunnel0
NHRP: Tu0: Found and skipping dynamic multicast mapping  NBMA: 150.1.3.3
NHRP: netid_out 123, netid_in 123
NHRP: nhrp_cache_lookup_comp returned 0x855C7B90
NHRP: Attempting to send packet via DEST 10.0.0.3
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.3.3
```

The NHS has the NHRP mapping for “10.0.0.2” in its NHRP cache – R2 registered this associating with R1. The NHS may immediately reply to the client. Note the “(C-1)” – CIE header in the NHRP reply packet. While the “(M)” (mandatory) header contains the same information received in request packet from R3, the CIE header contains the actual NHRP reply, with the mapping information for R2. This is because the NHS considers R2 to be the “client” of it, and therefore it sends the actual information in CIE header. Note the “prefix” length of 32 – this means the reply is just for one host logical IP address.

```
Rack1R1#
NHRP: Send Resolution Reply via Tunnel0 vrf 0, packet size: 109
 src: 10.0.0.1, dst: 10.0.0.3
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth dst-stable unique src-stable", reqid: 994
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.0.2
 (C-1) code: no error(0)
       prefix: 32, mtu: 1514, hd_time: 342
       addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client NBMA: 150.1.2.2
       client protocol: 10.0.0.2
NHRP: 109 bytes out Tunnel0
```

At this point, R2 receives the original data packet from R3 (ICMP echo) and tries to send a response back. The problem is that the destination IP address for the echo reply is “10.0.3.3” and the next-hop is “10.0.0.3”, which has “glean” CEF adjacency. Again, R2 replies back across the hub and send a Resolution Request packet: first, directly R3 – this attempt fails – then it sends the resolution request to the NHS.

```
Rack1R2#
NHRP: MACADDR: if_in null netid-in 0 if_out Tunnel0 netid-out 123
NHRP: Checking for delayed event 0.0.0.0/10.0.0.3 on list (Tunnel0).
NHRP: No node found.
```

```
NHRP: Sending packet to NHS 10.0.0.1 on Tunnel0
NHRP: Checking for delayed event 0.0.0.0/10.0.0.3 on list (Tunnel0).
NHRP: No node found.
NHRP: Attempting to send packet via DEST 10.0.0.3
NHRP: Send Resolution Request via Tunnel0 vrf 0, packet size: 81
 src: 10.0.0.2, dst: 10.0.0.3
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 1012
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.0.3
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: Encapsulation failed for destination 10.0.0.3 out Tunnel0
NHRP: Attempting to send packet via NHS 10.0.0.1
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.1.1

Rack1R2#
NHRP: Send Resolution Request via Tunnel0 vrf 0, packet size: 81
 src: 10.0.0.2, dst: 10.0.0.1
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 1012
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.0.3
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: 81 bytes out Tunnel0
NHRP: MACADDR: if_in null netid-in 0 if_out Tunnel0 netid-out 123
NHRP: Checking for delayed event 0.0.0.0/10.0.0.3 on list (Tunnel0).
NHRP: No node found.
NHRP: Sending packet to NHS 10.0.0.1 on Tunnel0
```

R3 finally receive the Resolution Reply from the NHS, and now it may complete the CEF adjacency for “10.0.0.2”. Since that moment, it switches all packets to “10.0.2.2” directly via R2, not across R1.

```
Rack1R3#
NHRP: Receive Resolution Reply via Tunnel0 vrf 0, packet size: 109
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth dst-stable unique src-stable", reqid: 994
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.0.2
 (C-1) code: no error(0)
```

```
       prefix: 32, mtu: 1514, hd_time: 342
       addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client NBMA: 150.1.2.2
       client protocol: 10.0.0.2
NHRP: netid_in = 0, to_us = 1
NHRP: Checking for delayed event 150.1.2.2/10.0.0.2 on list (Tunnel0).
NHRP: No node found.
NHRP: No need to delay processing of resolution event nbma src:150.1.3.3 nbma
dst:150.1.2.2
```

The resolution request that R2 sent before in attempted to resolve the NBMA address for “10.0.0.3” arrives to R1. Since the NHS has all the information in its local cache (R3 registered its IP to NBMA address mapping) it immediately replies to R2. Note the CIE header in the NHRP reply packet, which contains the actual mapping information.

```
Rack1R1#
NHRP: Receive Resolution Request via Tunnel0 vrf 0, packet size: 81
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 1012
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.0.3
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: netid_in = 123, to_us = 0
NHRP: NAT-check: matched destination address 150.1.2.2
NHRP: nhrp_rtlookup yielded Tunnel0
NHRP: Tu0: Found and skipping dynamic multicast mapping  NBMA: 150.1.2.2
NHRP: netid_out 123, netid_in 123
NHRP: nhrp_cache_lookup_comp returned 0x848EF9E8
NHRP: Attempting to send packet via DEST 10.0.0.2
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.2.2

Rack1R1#
NHRP: Send Resolution Reply via Tunnel0 vrf 0, packet size: 109
 src: 10.0.0.1, dst: 10.0.0.2
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth dst-stable unique src-stable", reqid: 1012
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.0.3
 (C-1) code: no error(0)
       prefix: 32, mtu: 1514, hd_time: 242
       addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client NBMA: 150.1.3.3
       bclient protocol: 10.0.0.3
```

```
NHRP: 109 bytes out Tunnel0
```

At last, R2 receive the reply to its original request, and now it has all the information to complete the CEF entry for “10.0.0.3” and switch packets across the optimal path to R3. At this moment both spokes have symmetric information to reach each other

```
Rack1R2#
NHRP: Receive Resolution Reply via Tunnel0 vrf 0, packet size: 109
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth dst-stable unique src-stable", reqid: 1012
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.0.3
 (C-1) code: no error(0)
       prefix: 32, mtu: 1514, hd_time: 242
       addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client NBMA: 150.1.3.3
       client protocol: 10.0.0.3
NHRP: netid_in = 0, to_us = 1
NHRP: Checking for delayed event 150.1.3.3/10.0.0.3 on list (Tunnel0).
NHRP: No node found.
NHRP: No need to delay processing of resolution event nbma src:150.1.2.2 nbma
dst:150.1.3.3
```

Timing out NHRP entries

Now that we know that CEF resolves the next-hop information via NHRP, how does it time-out the unused cut-through tunnel? As we remember, each NHRP entry has countdown expire timer, initialized from the registration hold-time. Every 60 seconds global NHRP process runs on a router and checks the expire timer on all NHRP entries. If the expire timer for an NHRP entry is greater than 120 seconds, nothing is done to the corresponding CEF entry. If the timer is less than 120 seconds, the NHRP process marks the corresponding CEF entry as “stale” but still usable. As soon as the router switches an IP packet using the “stale” entry, it triggers new NHRP resolution request, and eventually refreshes the corresponding NHRP entry as well as CEF entry itself. If no packet hits the “stale” CEF entry, the NHRP mapping will eventually time-out (since the router does not send any “refreshing” requests) and the corresponding CEF entry will become invalid. This will effectively tear down the spoke-to-spoke tunnel.

NHRP Phase 2 Conclusions

Let us quickly recap what we learned so far about NHRP Phase 2 and CEF. Firstly, this mode requires all the spokes to have complete routing information with the next-hop preserved. This may limit scalability in large networks, since not all spokes may accept full load of routing updates. Secondly, CEF only resolve the next-hop information via NHRP, not the full routing prefixes. Actually, the second feature directly implies the first limitation. As we noted, the **no ip next-hop-self eigrp 123** command is required to make spoke-to-spoke tunnels work with CEF. However, they added the command only in IOS version 12.3. Is there a way to make spoke-to-

spoke tunnels work when the next-hop is set to “self” (the default) in EIGRP updates? Actually, there are few ways. First and the best one – do not use old IOS images to implement DMVPN Actually, it is better to use the latest 12.4T train images with DMVPN Phase 3 for the deployment – but then again those images are from the “T”-train! OK, so the other option is get rid of EIGRP and use OSPF, with the network type “broadcast”. OSPF is a link-state protocol – it does not hide topology information and does not mask the next-hop in any way (well, at least when the network-type is “broadcast”). However, the limitation is that the corresponding OSPF topology may have just two redundant hubs – corresponding to OSPF DR and BDR for a segment. This is because every hub must form OSPF adjacencies with all spokes. Such limitation is not acceptable in large installations, but still works fine in smaller deployments. However, there is one final workaround, which is probably the one you may want to use in the current CCIE lab exam – disable CEF on spokes. This is a very interesting case per se, and we are going to see now NHRP works with process switching.

NHRP Phase 2 + EIGRP next-hop-self + no CEF

In this scenario, EIGRP next-hop self is enabled on R1 (the hub). Now R3 sees 10.0.2.0/24 with the next hop of R1. Disable CEF on R2 and R3, and try pinging 10.0.2.2 off R3 loopback1 interface.

```
R3 sees the route behind R2 as reachable via R1

Rack1R3#show ip route 10.0.2.2
Routing entry for 10.0.2.0/24
  Known via "eigrp 123", distance 90, metric 310172416, type internal
  Redistributing via eigrp 123
  Last update from 10.0.0.1 on Tunnel0, 00:09:55 ago
  Routing Descriptor Blocks:
  * 10.0.0.1, from 10.0.0.1, 00:09:55 ago, via Tunnel0
      Route metric is 310172416, traffic share count is 1
      Total delay is 1005000 microseconds, minimum bandwidth is 9 Kbit
      Reliability 255/255, minimum MTU 1472 bytes
      Loading 1/255, Hops 2
```

R3 pings “10.0.2.2”, sourcing packet off “10.0.3.3”. Since CEF is disabled, the system performs NHRP lookup to find the NBMA address for “10.0.2.2”. This is opposed to CEF behavior that would only resolve the next-hop for “10.0.2.2″ entry. Naturally, the router forwards NHRP request to R3’s NHS, which is R1. At the same time, R3 forwards the data packet (ICMP echo) via its current next-hop – “10.0.0.1”, that is via the hub.

```
Rack1R3#
NHRP: MACADDR: if_in null netid-in 0 if_out Tunnel0 netid-out 123
NHRP: Checking for delayed event 0.0.0.0/10.0.2.2 on list (Tunnel0).
NHRP: No node found.
NHRP: Sending packet to NHS 10.0.0.1 on Tunnel0
NHRP: Checking for delayed event 0.0.0.0/10.0.2.2 on list (Tunnel0).
```

```
NHRP: No node found.
NHRP: Attempting to send packet via DEST 10.0.2.2
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.1.1
NHRP: Send Resolution Request via Tunnel0 vrf 0, packet size: 81
 src: 10.0.0.3, dst: 10.0.2.2
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 900
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.2.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: 81 bytes out Tunnel0
NHRP: MACADDR: if_in null netid-in 0 if_out Tunnel0 netid-out 123
NHRP: Checking for delayed event 0.0.0.0/10.0.2.2 on list (Tunnel0).
NHRP: No node found.
NHRP: Sending packet to NHS 10.0.0.1 on Tunnel0
```

Resolution Request arrives to R1 (the NHS). Since R1 has no mapping for “10.0.2.2” (R2 only registers the IP address 10.0.0.2 – its own next-hop IP address), the NHS looks up into routing table, to find the next-hop towards 10.0.2.2. Since it happens to be R2’s IP “10.0.0.2”, the NHS then tries to **forward** the resolution request towards the next router on the path to the network requested in resolution message – to R2. Thanks to R2’s NHRP registration with R1, the NHS now knows R2’s NBMA address, and successfully encapsulates the packet. In addition, R1 forwards the data packet from R1 to R2, using its routing table. Obviously, the data packet will arrive to R2 a little bit faster, since NHRP requires more time to process and forward the request.

```
Rack1R1#
NHRP: Receive Resolution Request via Tunnel0 vrf 0, packet size: 81
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 900
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.2.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: netid_in = 123, to_us = 0
NHRP: NAT-check: matched destination address 150.1.3.3
NHRP: nhrp_rtlookup yielded Tunnel0
NHRP: Tu0: Found and skipping dynamic multicast mapping  NBMA: 150.1.3.3
NHRP: netid_out 123, netid_in 123
NHRP: nhrp_cache_lookup_comp returned 0x0
NHRP: Attempting to send packet via DEST 10.0.2.2
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.2.2
```

```
NHRP: Forwarding Resolution Request via Tunnel0 vrf 0, packet size: 101
 src: 10.0.0.1, dst: 10.0.2.2
 (F) afn: IPv4(1), type: IP(800), hop: 254, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 900
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.2.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: 101 bytes out Tunnel0
```

Now the data packet (ICMP echo) has arrived to R2. R2 generates the response (ICMP – echo reply from “10.0.2.2” to “10.0.3.3”) and now R2 needs the NMBA address of “10.0.3.3” (CEF is disabled on R2). As usual, R2 generates a resolutions request to its NHS (R1). At the same time, R2 sends the response packet to R3’s request across the hub, since it does not know the NBMA address of R3.

```
Rack1R2#
NHRP: Send Resolution Request via Tunnel0 vrf 0, packet size: 81
 src: 10.0.0.2, dst: 10.0.3.3
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 919
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.3.3
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: 81 bytes out Tunnel0
```

Soon after the data packet arrived, R2 receives the Resolution Request from R3 forwarded by R1. Since R2 is the egress router on NBMA segment for the network “10.0.2.2”, it may reply to the request.

```
Rack1R2#
NHRP: Receive Resolution Request via Tunnel0 vrf 0, packet size: 101
 (F) afn: IPv4(1), type: IP(800), hop: 254, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 900
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.2.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: netid_in = 123, to_us = 0
NHRP: nhrp_rtlookup yielded Loopback1
NHRP: netid_out 0, netid_in 123
```

```
NHRP: We are egress router for target 10.0.2.2, recevied via Tunnel0
NHRP: Redist mask now 1
NHRP: Attempting to send packet via DEST 10.0.0.3
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.3.3
```

Note that R2 replies with the **full** prefix found in its routing table – “10.0.2.0/24”, not just single host “10.0.2.2/32” (this feature is critical for DMVPN Phase 3). This information is encapsulated inside “(C-1)” part of the NHRP reply packet (Client Information Element 1) which describes a client – network connected to the router (R2). The “prefix” field is “/24” which is exactly the value taken from the routing table.

Also note, that R2 learned R3’s NBMA address from the Resolution Request, and now replies **directly** to R3, bypassing R1. The “stable” flag means that the querying/replying router directly knows the source or destination IP address in the resolution request/reply.

```
Rack1R2#
NHRP: Send Resolution Reply via Tunnel0 vrf 0, packet size: 129
 src: 10.0.0.2, dst: 10.0.0.3   <-- NBMA addresses of R2/R3
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth dst-stable unique src-stable", reqid: 900
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.2.2
 (C-1) code: no error(0)
       prefix: 24, mtu: 1514, hd_time: 360
       addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client NBMA: 150.1.2.2
       client protocol: 10.0.2.2
NHRP: 129 bytes out Tunnel0
```

At this moment, Resolution Request from R2 for network “10.0.3.3″ reaches R1 – the NHS. Since the NHS has no information on “10.0.3.3″, it forwards the request to R3 – the next-hop found via the routing table on path to “10.0.3.3″.

```
Rack1R1#
NHRP: Receive Resolution Request via Tunnel0 vrf 0, packet size: 81
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 919
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.3.3
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: netid_in = 123, to_us = 0
NHRP: NAT-check: matched destination address 150.1.2.2
NHRP: nhrp_rtlookup yielded Tunnel0
```

```
NHRP: Tu0: Found and skipping dynamic multicast mapping  NBMA: 150.1.2.2
NHRP: netid_out 123, netid_in 123
NHRP: nhrp_cache_lookup_comp returned 0x0
NHRP: Attempting to send packet via DEST 10.0.3.3
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.3.3

NHRP: Forwarding Resolution Request via Tunnel0 vrf 0, packet size: 101
 src: 10.0.0.1, dst: 10.0.3.3
 (F) afn: IPv4(1), type: IP(800), hop: 254, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 919
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.3.3
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: 101 bytes out Tunnel0
```

Back to R3. At this point, it received the ICMP reply for the original ICMP echo packet. Now R3 receives the NHRP Resolution Reply to its original Resolution Request directly from R2. This allows R3 to learn that “10.0.2.0/24” is reachable via NMBA IP address “150.1.2.2”. Note that CIE field “(C-1)” in the reply packet, which tells R3 about the whole “10.0.2.0/24” network – the “prefix” is set to “24”.

```
Rack1R3#
NHRP: Receive Resolution Reply via Tunnel0 vrf 0, packet size: 129
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth dst-stable unique src-stable", reqid: 900
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.2.2
 (C-1) code: no error(0)
       prefix: 24, mtu: 1514, hd_time: 360
       addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client NBMA: 150.1.2.2
       client protocol: 10.0.2.2
NHRP: netid_in = 0, to_us = 1
NHRP: NAT-check: matched destination address 150.1.2.2
NHRP: Checking for delayed event 150.1.2.2/10.0.2.2 on list (Tunnel0).
NHRP: No node found.
NHRP: No need to delay processing of resolution event nbma src:150.1.3.3 nbma
dst:150.1.2.2
NHRP: Checking for delayed event 0.0.0.0/10.0.2.2 on list (Tunnel0).
NHRP: No node found.
```

Finally, the Resolution Request from R2, forwarded by R1 (the NHS) arrives to R3. The local router performs lookup for 10.0.3.3 and finds this to be directly connected network, with the prefix

of /24. Therefore, R3 generates a Resolution Reply packet and sends it directly to R2, bypassing R1. This packet tells R2 to map logical IP “10.0.3.0/24” to NBMA address “150.1.3.3”.

```
Rack1R3#
NHRP: Receive Resolution Request via Tunnel0 vrf 0, packet size: 101
 (F) afn: IPv4(1), type: IP(800), hop: 254, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth src-stable", reqid: 919
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.3.3
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 360
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: netid_in = 123, to_us = 0
NHRP: nhrp_rtlookup yielded Loopback1
NHRP: netid_out 0, netid_in 123
NHRP: We are egress router for target 10.0.3.3, recevied via Tunnel0
NHRP: Redist mask now 1
NHRP: Attempting to send packet via DEST 10.0.0.2
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.2.2

NHRP: Send Resolution Reply via Tunnel0 vrf 0, packet size: 129
 src: 10.0.0.3, dst: 10.0.0.2
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth dst-stable unique src-stable", reqid: 919
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.3.3
 (C-1) code: no error(0)
       prefix: 24, mtu: 1514, hd_time: 360
       addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client NBMA: 150.1.3.3
       client protocol: 10.0.3.3
NHRP: 129 bytes out Tunnel0
```

At last, R2 receives the response to its Resolution Request, and everything is stable now. R2 and R3 know how to reach “10.0.3.0/24” and “10.0.2.0/24” respectively.

```
Rack1R2#
NHRP: Receive Resolution Reply via Tunnel0 vrf 0, packet size: 129
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "router auth dst-stable unique src-stable", reqid: 919
     src NBMA: 150.1.2.2
     src protocol: 10.0.0.2, dst protocol: 10.0.3.3
 (C-1) code: no error(0)
       prefix: 24, mtu: 1514, hd_time: 360
```

```
       addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client NBMA: 150.1.3.3
       client protocol: 10.0.3.3
NHRP: netid_in = 0, to_us = 1
NHRP: NAT-check: matched destination address 150.1.3.3
NHRP: Checking for delayed event 150.1.3.3/10.0.3.3 on list (Tunnel0).
NHRP: No node found.
NHRP: No need to delay processing of resolution event nbma src:150.1.2.2 nbma
dst:150.1.3.3
NHRP: Checking for delayed event 0.0.0.0/10.0.3.3 on list (Tunnel0).
NHRP: No node found.
```

Now let's look at NHRP caches of all three routers:

```
Rack1R1#show ip nhrp
10.0.0.2/32 via 10.0.0.2, Tunnel0 created 01:00:47, expire 00:04:02
  Type: dynamic, Flags: authoritative unique registered
  NBMA address: 150.1.2.2
10.0.0.3/32 via 10.0.0.3, Tunnel0 created 01:00:47, expire 00:04:23
  Type: dynamic, Flags: authoritative unique registered
  NBMA address: 150.1.3.3

Rack1R2#show ip nhrp
10.0.0.1/32 via 10.0.0.1, Tunnel0 created 01:56:30, never expire
  Type: static, Flags: authoritative used
  NBMA address: 150.1.1.1
10.0.0.3/32 via 10.0.0.3, Tunnel0 created 00:00:24, expire 00:05:35
  Type: dynamic, Flags: router implicit
  NBMA address: 150.1.3.3
  10.0.2.0/24 via 10.0.2.2, Tunnel0 created 00:00:24, expire 00:05:35
  Type: dynamic, Flags: router authoritative unique local
  NBMA address: 150.1.2.2
    (no-socket)
10.0.3.0/24 via 10.0.3.3, Tunnel0 created 00:00:24, expire 00:05:35
  Type: dynamic, Flags: router
  NBMA address: 150.1.3.3

Rack1R3#show ip nhrp
10.0.0.1/32 via 10.0.0.1, Tunnel0 created 01:56:00, never expire
  Type: static, Flags: authoritative used
  NBMA address: 150.1.1.1
10.0.0.2/32 via 10.0.0.2, Tunnel0 created 00:00:02, expire 00:05:57
  Type: dynamic, Flags: router implicit used
  NBMA address: 150.1.2.2
10.0.2.0/24 via 10.0.2.2, Tunnel0 created 00:00:02, expire 00:05:57
  Type: dynamic, Flags: router used
  NBMA address: 150.1.2.2
```

```
10.0.3.0/24 via 10.0.3.3, Tunnel0 created 00:00:02, expire 00:05:57
  Type: dynamic, Flags: router authoritative unique local
  NBMA address: 150.1.3.3
    (no-socket)
```

The “implicit” flag means that the router learned mapping without explicit request, as a part of other router’s reply or request. The “router” flag means that the mapping is either for the remote router or for a network behind the router. The “(no-socket)” flag means that the local router will not use this entry and trigger IPSec socket creation. The “local” flag means the mapping is for the network directly connected to the local router. The router uses those mappings when it loses connection to the local network, so that the NHC may send a purge request to all other clients, telling that the network has gone and they must remove their mappings.

Here is an example. Ensure R3 has the above-mentioned mappings, and then shut down the Loopback1 interface, observing the debugging command output on R3 and R2. R3 sends purge request directly to R2, since it knows R2 requested that mapping.

```
Rack1R3#
NHRP: Redist callback: 10.0.3.0/24
NHRP: Invalidating map tables for prefix 10.0.3.0/24 via Tunnel0
NHRP: Checking for delayed event 150.1.3.3/10.0.3.3 on list (Tunnel0).
NHRP: No node found.
NHRP: Attempting to send packet via DEST 10.0.0.2
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.2.2
NHRP: Send Purge Request via Tunnel0  vrf 0, packet size: 73
 src: 10.0.0.3, dst: 10.0.0.2
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "reply required", reqid: 36
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.0.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 0
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client protocol: 10.0.3.3
NHRP: 73 bytes out Tunnel0
```

R2 receives Purge Request from R3. Note that the “reply required” flag is set. Hence, R2 must confirm that it deleted the mapping with a Purge Reply packet. R2 will erase the corresponding mapping learned via “10.0.0.3” and generate a response packet

```
Rack1R2#
NHRP: Receive Purge Request via Tunnel0  vrf 0, packet size: 73
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "reply required", reqid: 36
     src NBMA: 150.1.3.3
```

```
     src protocol: 10.0.0.3, dst protocol: 10.0.0.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 0
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client protocol: 10.0.3.3
NHRP: netid_in = 123, to_us = 1
NHRP: Attempting to send packet via DEST 10.0.0.3
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.3.3
```

R2 first tries to send the Purge Reply to R3 directly, using the NBMA address of R3. Note that CIE header mentions the network erased from the local mappings list

```
Rack1R2#
NHRP: Send Purge Reply via Tunnel0 vrf 0, packet size: 73
 src: 10.0.0.2, dst: 10.0.0.3
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "reply required", reqid: 36
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.0.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 0
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client protocol: 10.0.3.3
NHRP: 73 bytes out Tunnel0
NHRP: Invalidating map tables for prefix 10.0.3.0/24 via Tunnel0
NHRP: Attempting to send packet via DEST 10.0.0.1
NHRP: Encapsulation succeeded.  Tunnel IP addr 150.1.1.1
```

R3 receives the reply to its purge request and now it knows that R2 is consistent.

```
Rack1R3#
NHRP: Receive Purge Reply via Tunnel0 vrf 0, packet size: 73
 (F) afn: IPv4(1), type: IP(800), hop: 255, ver: 1
     shtl: 4(NSAP), sstl: 0(NSAP)
 (M) flags: "reply required", reqid: 36
     src NBMA: 150.1.3.3
     src protocol: 10.0.0.3, dst protocol: 10.0.0.2
 (C-1) code: no error(0)
       prefix: 0, mtu: 1514, hd_time: 0
       addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
       client protocol: 10.0.3.3
NHRP: netid_in = 0, to_us = 1
```

Timing out NHRP entries with Process-Switching

The last question is how NHRP times out unused entries in case of process-switching mode. Recall the “used” flag set for NHRP mapping. Every time a packet is process-switched using the respective NHRP entry, it is marked as “used”. The background NHRP process runs every 60

seconds, and check the expire timers for each NHRP entry. If the “used” flag is set and expire timer for the entry is greater than 120 seconds then the process clears the flag (and every new packet will refresh it). If the timer is less than 120 seconds and the flag is set, IOS generates a refreshing NHRP request. However, if the flag is not set, the system allows the entry to expire, unless another packet hits it and makes active.

The above-described behavior of NHRP with process switching allows for one interesting feature. The hub router may now summarize all information sent down to spokes say into one default route. This will not affect the spokes, for they will continue querying next-hop information for every destination prefix sent over the mGRE tunnel interface, and learning the optimal next-hop. It would be great to combine this “summarization” feature with the performance of CEF switching. This is exactly what they implemented with DMVPN Phase 3. However, Phase 3 is subject to a separate discussion.

Integrating IPsec

Haven’t we forgotten something for DMVPN Phase 1/Phase 2? That was IPsec, the components that provides confidentiality and integrity checking to mGRE/NHRP. Now, compared with the complexity of NHRP operations, IPsec integration is straightforward.

First, the hub needs to know how to authentication all the spokes using IKE. The most scalable way is to use X.509 certificates and PKI, but for the simplicity, we will just use the same pre-shared key on all routers. Note that we need to configure the routers with a wild-card pre-shared key, in order to accept IKE negotiation requests from any other dynamic peer.

As for IPsec Phase 2, we need dynamic crypto maps there, since the hub has no idea of the connecting peer IP addresses. Fortunately, Cisco IOS has a cute feature called IPsec profiles, designed for use with tunnel interfaces. The profile attaches to a tunnel interface and automatically considers all traffic going out of the tunnel as triggering the IPsec Phase 2. The IPsec phase proxy identities used by the IPsec profile are the source and destination host IP addresses of the tunnel. It makes sense to use IPSec transport mode with mGRE as the latter already provides tunnel encapsulation. Besides, IOS supports some features, like NAT traversal only with IPSec transport mode.

Let’s review an example below and explain how it works.

mGRE + NHRP Phase 2 + Spoke-to-spoke tunnels + IPsec

```
R1:
crypto isakmp policy 10
 encryption 3des
 authentication pre-share
 hash md5
 group 2
!
```

```
crypto isakmp key 0 CISCO address 0.0.0.0 0.0.0.0
!
crypto ipsec transform-set 3DES_MD5 esp-3des esp-md5-hmac
 mode transport
!
crypto ipsec profile DMVPN
 set transform-set 3DES_MD5
!
interface Tunnel 0
 tunnel protection ipsec profile DMVPN
```

R2 & R3:

```
crypto isakmp policy 10
 encryption 3des
 authentication pre-share
 hash md5
 group 2
!
crypto isakmp key 0 CISCO address 0.0.0.0 0.0.0.0
!
crypto ipsec transform-set 3DES_MD5 esp-3des esp-md5-hmac
 mode transport
!
crypto ipsec profile DMVPN
 set transform-set 3DES_MD5
!
interface Tunnel 0
 tunnel protection ipsec profile DMVPN
```

Start with any spoke, e.g. R3. Since the router uses EIGRP on Tunnel 0 interface, a multicast packet will eventually be send out of the tunnel interface. Thanks to the static NHRP multicast mapping, mGRE will encapsulate the EIGRP packet towards the hub router. The IPsec profile will see GRE traffic going from “150.1.3.3” to “150.1.1.1”. Automatically, ISAKMP negotiation will start with R1, and authentication will use pre-shared keys. Eventually both R1 and R3 will create IPsec SAs for GRE traffic between “150.1.3.3” and “150.1.1.1”. Now R3 may send NHRP resolution request. As soon as R3 tries to send traffic to a network behind R2, it will resolve next-hop “10.0.0.2” to the IP address of 150.1.2.2. This new NHRP entry will trigger ISAKMP negotiation with NBMA address 150.1.2.2 as soon as router tries to use it for packet forwarding. IKE negotiation between R3 and R2 will start and result in formation of new SAs corresponding to IP address pair “150.1.2.2 and 150.1.3.3” and GRE protocol. As soon as the routers complete IPsec Phase 2, packets may flow between R2 and R3 across the shortcut path.

When an unused NHRP entry times out, it will signal the ISAKMP process to terminate the respective IPsec connection. We described the process for timing out NHRP entries before, and

as you remember, it depends on the “hold-time” value set by the routers. Additionally, the systems may expire ISAKMP/IPsec connections due to IPsec timeouts.

This is the crypto system status on the hub from the example with NHRP Phase 2 and process-switching:

```
IPsec Phase 1 has been established with both spokes

Rack1R1#show crypto isakmp sa
dst             src             state          conn-id slot status
150.1.1.1       150.1.2.2       QM_IDLE              1    0 ACTIVE
150.1.1.1       150.1.3.3       QM_IDLE              3    0 ACTIVE
```

IPsec Phase 2 SA entries for both protected connections to R2 and R3 follows. Note that SAs are for GRE traffic between the loopback.

```
Rack1R1#show crypto ipsec sa

interface: Tunnel0
    Crypto map tag: Tunnel0-head-0, local addr 150.1.1.1

   protected vrf: (none)
   local  ident (addr/mask/prot/port): (150.1.1.1/255.255.255.255/47/0)
   remote ident (addr/mask/prot/port): (150.1.2.2/255.255.255.255/47/0)
   current_peer 150.1.2.2 port 500
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 230, #pkts encrypt: 230, #pkts digest: 230
    #pkts decaps: 227, #pkts decrypt: 227, #pkts verify: 227
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0
    #pkts not decompressed: 0, #pkts decompress failed: 0
    #send errors 12, #recv errors 0

     local crypto endpt.: 150.1.1.1, remote crypto endpt.: 150.1.2.2
     path mtu 1514, ip mtu 1514, ip mtu idb Loopback0
     current outbound spi: 0x88261BA3(2284198819)

     inbound esp sas:
      spi: 0xE279A1EE(3799622126)
        transform: esp-3des esp-md5-hmac ,
        in use settings ={Transport, }
        conn id: 2001, flow_id: SW:1, crypto map: Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4472116/2632)
        IV size: 8 bytes
        replay detection support: Y
        Status: ACTIVE
```

```
 spi: 0xB4F6A9E5(3036064229)
   transform: esp-3des esp-md5-hmac ,
   in use settings ={Transport, }
   conn id: 2003, flow_id: SW:3, crypto map: Tunnel0-head-0
   sa timing: remaining key lifetime (k/sec): (4596176/2630)
   IV size: 8 bytes
   replay detection support: Y
   Status: ACTIVE
 spi: 0x1492E4D0(345171152)
   transform: esp-3des esp-md5-hmac ,
   in use settings ={Transport, }
   conn id: 2005, flow_id: SW:5, crypto map: Tunnel0-head-0
   sa timing: remaining key lifetime (k/sec): (4525264/2630)
   IV size: 8 bytes
   replay detection support: Y
   Status: ACTIVE

inbound ah sas:

inbound pcp sas:

outbound esp sas:
 spi: 0x81949874(2173999220)
   transform: esp-3des esp-md5-hmac ,
   in use settings ={Transport, }
   conn id: 2002, flow_id: SW:2, crypto map: Tunnel0-head-0
   sa timing: remaining key lifetime (k/sec): (4472116/2626)
   IV size: 8 bytes
   replay detection support: Y
   Status: ACTIVE
 spi: 0xAA5D21A7(2858230183)
   transform: esp-3des esp-md5-hmac ,
   in use settings ={Transport, }
   conn id: 2004, flow_id: SW:4, crypto map: Tunnel0-head-0
   sa timing: remaining key lifetime (k/sec): (4596176/2627)
   IV size: 8 bytes
   replay detection support: Y
   Status: ACTIVE
 spi: 0x88261BA3(2284198819)
   transform: esp-3des esp-md5-hmac ,
   in use settings ={Transport, }
   conn id: 2006, flow_id: SW:6, crypto map: Tunnel0-head-0
   sa timing: remaining key lifetime (k/sec): (4525265/2627)
   IV size: 8 bytes
   replay detection support: Y
   Status: ACTIVE
```

```
  outbound ah sas:

  outbound pcp sas:

protected vrf: (none)
local  ident (addr/mask/prot/port): (150.1.1.1/255.255.255.255/47/0)
remote ident (addr/mask/prot/port): (150.1.3.3/255.255.255.255/47/0)
current_peer 150.1.3.3 port 500
  PERMIT, flags={origin_is_acl,}
 #pkts encaps: 225, #pkts encrypt: 225, #pkts digest: 225
 #pkts decaps: 226, #pkts decrypt: 226, #pkts verify: 226
 #pkts compressed: 0, #pkts decompressed: 0
 #pkts not compressed: 0, #pkts compr. failed: 0
 #pkts not decompressed: 0, #pkts decompress failed: 0
 #send errors 17, #recv errors 0

  local crypto endpt.: 150.1.1.1, remote crypto endpt.: 150.1.3.3
  path mtu 1514, ip mtu 1514, ip mtu idb Loopback0
  current outbound spi: 0xBEB1D9CE(3199326670)

  inbound esp sas:
   spi: 0x10B44B31(280251185)
     transform: esp-3des esp-md5-hmac ,
     in use settings ={Transport, }
     conn id: 2007, flow_id: SW:7, crypto map: Tunnel0-head-0
     sa timing: remaining key lifetime (k/sec): (4436422/2627)
     IV size: 8 bytes
     replay detection support: Y
     Status: ACTIVE

  inbound ah sas:

  inbound pcp sas:

  outbound esp sas:
   spi: 0xBEB1D9CE(3199326670)
     transform: esp-3des esp-md5-hmac ,
     in use settings ={Transport, }
     conn id: 2008, flow_id: SW:8, crypto map: Tunnel0-head-0
     sa timing: remaining key lifetime (k/sec): (4436424/2627)
     IV size: 8 bytes
     replay detection support: Y
     Status: ACTIVE

  outbound ah sas:
```

```
    outbound pcp sas:
```

Now let’s see how a spoke router establishes a spoke-to-spoke IPsec tunnel:

```
No NHRP mapping for spoke’s network first


Rack1R3#sh ip nhrp
10.0.0.1/32 via 10.0.0.1, Tunnel0 created 02:02:42, never expire
  Type: static, Flags: authoritative used
  NBMA address: 150.1.1.1


ISAKMP negotiated just with R1


Rack1R3#sh crypto isakmp sa
dst             src             state          conn-id slot status
150.1.1.1       150.1.3.3       QM_IDLE              1    0 ACTIVE
```

Generate traffic to network behind R2. Note that the first ping passes through, since it’s routed across the hub, but the second packet is sent directly to R2 and is missed, since IPsec Phase 2 has not yet been established

```
Rack1R3#ping 10.0.2.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.2.2, timeout is 2 seconds:
!.!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 52/121/324 ms


Notice the new NHRP mappings. Note that the tunnel will expire in about 3
 minutes, if no new traffic is going to be generated


Rack1R3#sh ip nhrp
10.0.0.1/32 via 10.0.0.1, Tunnel0 created 02:05:38, never expire
  Type: static, Flags: authoritative used
  NBMA address: 150.1.1.1
10.0.2.0/24 via 10.0.2.2, Tunnel0 created 00:02:44, expire 00:03:15
  Type: dynamic, Flags: router
  NBMA address: 150.1.2.2
```

IOS create IPsec Phase 2 SAs for tunnels between R2-R3 and R1-R3. The tunnel between 2 and R3 is dynamic and is used to send only the data traffic.

```
Rack1R3#show crypto isakmp sa
dst             src             state          conn-id slot status
150.1.1.1       150.1.3.3       QM_IDLE              1    0 ACTIVE
150.1.3.3       150.1.2.2       QM_IDLE              2    0 ACTIVE

Rack1R3#show crypto ipsec sa

interface: Tunnel0
    Crypto map tag: Tunnel0-head-0, local addr 150.1.3.3

   protected vrf: (none)
   local  ident (addr/mask/prot/port): (150.1.3.3/255.255.255.255/47/0)
   remote ident (addr/mask/prot/port): (150.1.1.1/255.255.255.255/47/0)
   current_peer 150.1.1.1 port 500
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 290, #pkts encrypt: 290, #pkts digest: 290
    #pkts decaps: 284, #pkts decrypt: 284, #pkts verify: 284
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0
    #pkts not decompressed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 150.1.3.3, remote crypto endpt.: 150.1.1.1
     path mtu 1514, ip mtu 1514, ip mtu idb Loopback0
     current outbound spi: 0x10B44B31(280251185)

     inbound esp sas:
      spi: 0xBEB1D9CE(3199326670)
        transform: esp-3des esp-md5-hmac ,
        in use settings ={Transport, }
        conn id: 2001, flow_id: SW:1, crypto map: Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4526856/2383)
        IV size: 8 bytes
        replay detection support: Y
        Status: ACTIVE

     inbound ah sas:

     inbound pcp sas:

     outbound esp sas:
      spi: 0x10B44B31(280251185)
        transform: esp-3des esp-md5-hmac ,
        in use settings ={Transport, }
        conn id: 2002, flow_id: SW:2, crypto map: Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4526853/2381)
```

```
      IV size: 8 bytes
      replay detection support: Y
      Status: ACTIVE

  outbound ah sas:

  outbound pcp sas:

protected vrf: (none)
local  ident (addr/mask/prot/port): (150.1.3.3/255.255.255.255/47/0)
remote ident (addr/mask/prot/port): (150.1.2.2/255.255.255.255/47/0)
current_peer 150.1.2.2 port 500
  PERMIT, flags={origin_is_acl,}
 #pkts encaps: 3, #pkts encrypt: 3, #pkts digest: 3
 #pkts decaps: 4, #pkts decrypt: 4, #pkts verify: 4
 #pkts compressed: 0, #pkts decompressed: 0
 #pkts not compressed: 0, #pkts compr. failed: 0
 #pkts not decompressed: 0, #pkts decompress failed: 0
 #send errors 0, #recv errors 0

  local crypto endpt.: 150.1.3.3, remote crypto endpt.: 150.1.2.2
  path mtu 1514, ip mtu 1514, ip mtu idb Loopback0
  current outbound spi: 0x847D8EEC(2222821100)

  inbound esp sas:
   spi: 0xA6851754(2793740116)
     transform: esp-3des esp-md5-hmac ,
     in use settings ={Transport, }
     conn id: 2004, flow_id: SW:4, crypto map: Tunnel0-head-0
     sa timing: remaining key lifetime (k/sec): (4602306/3572)
     IV size: 8 bytes
     replay detection support: Y
     Status: ACTIVE

  inbound ah sas:

  inbound pcp sas:

  outbound esp sas:
   spi: 0x847D8EEC(2222821100)
     transform: esp-3des esp-md5-hmac ,
     in use settings ={Transport, }
     conn id: 2003, flow_id: SW:3, crypto map: Tunnel0-head-0
     sa timing: remaining key lifetime (k/sec): (4602306/3572)
     IV size: 8 bytes
     replay detection support: Y
```

```
        Status: ACTIVE

    outbound ah sas:

    outbound pcp sas:
```

Now you see how all the component of DMVPN work together. We have not covered some other major topics like NAT traversal with NHRP and DMVPN redundancy with multiple hubs. Those advanced topics probably require a separate post, since this one has grown too big already