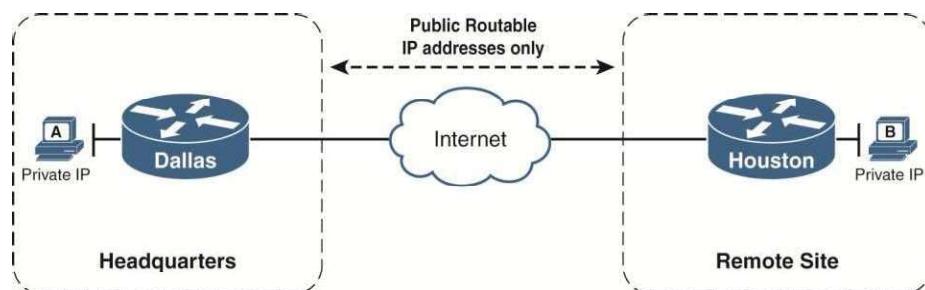


Introduction to DMVPN

One of the challenges many organizations face is dealing with remote site connectivity. Typically, companies start with a single headquarters office. As the company grows and expands into different locations, these remote locations need to have a means of communicating with each other and the original headquarters location. When sites are geographically separated in such a way, a wide area network (WAN) is needed in order to allow them to interact with each other.

A WAN is a network that spans multiple geographic regions and is typically created by a single service provider or a group of service providers sharing a common interest. The best example of a WAN is the public Internet. The public Internet is a WAN composed of many privately owned organizations and service providers collaborating together to create a global network.

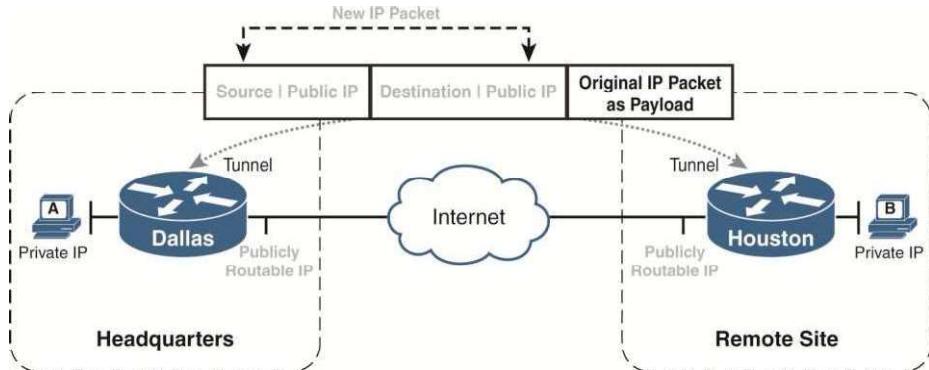
Internet connectivity over the years has moved from being a luxury to being a requirement for continued business operations. It is common for most branch sites to start out with only a single Internet connection in their infancy. Take the example below:



The corporate headquarters office located in Dallas has a remote site located in Houston. Both the headquarters office and remote site have active connections to the public Internet. The company can leverage the existing Internet connection as the WAN transport to exchange information between the networks at the headquarters site and those at the remote site. However, this presents a problem. The public Internet uses public IP addressing, whereas the remote sites use RFC 1918 private addressing. Internet routers are not configured to route RFC 1918 private addresses, which means a packet from Host A in Dallas would not reach Host B in Houston.

Tunneling provides an easy solution to this problem. When the ISP provisions the connection to the Internet, the Dallas and Houston routers are provided with publicly routable IP addressing.

Tunneling allows the Dallas router to encapsulate the original IP packet from Host A into an IP packet sourced from Dallas's public IP address and destined to Houston's public IP address. This happens by adding an extra IP header to the existing IP packet called an outer header. The resulting packet is an IP packet that contains the original IP packet with the internal addressing as payload.



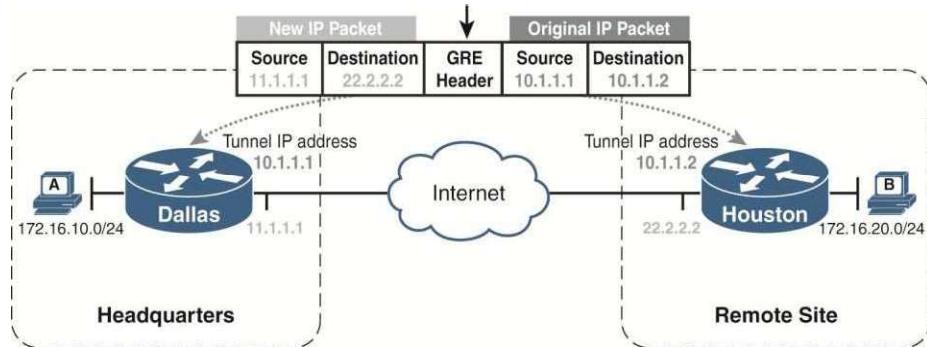
The Internet routers forward the new IP packet just as they would any other traffic, landing it at the Houston router. The Houston router removes the outer IP packet header to expose the original IP packet destined from Host A to Host B. The Houston router can then route the original packet out its LAN interface toward Host B.

What has happened here is that the Dallas and Houston routers have created a virtual network that uses the public Internet as its transport. Such a virtual network is called an **overlay network** because it is dependent on an underlying transport, called the **underlay network**. Traffic flowing through the overlay network is forwarded by the underlay network as opaque data. Without the use of protocol analyzers, the underlay network is unaware of the existence of the overlay. The Dallas and Houston routers are considered the endpoints of the tunnel that forms the overlay network.

Many tunneling technologies can be used to form the overlay network. Probably the most widely used is the **Generic Routing Encapsulation (GRE) tunnel**. A GRE tunnel can support tunneling for a variety of protocols over an IP-based network. It works by inserting an IP and GRE header on top of the original protocol packet, creating a new GRE/IP packet. The resulting GRE/IP packet uses a source/destination pair that is routable over the underlying infrastructure, as described above. The GRE/IP header is known as the **outer header**, and the original protocol header is the **inner header**.

In Cisco IOS, GRE tunnels are configured using tunnel interfaces. A tunnel interface is defined and assigned its own unique IP address, which defines the overlay network. It is then configured with appropriate source and destination IP addresses to use for the GRE tunnel's outer IP header. These addresses should be reachable over the underlay network. For example, if the underlay is provided by the public Internet, then these source and destination addresses should be publicly routable addresses. Typically, the destination IP address should be the address of the remote tunnel endpoint's interface connected to the underlay network. The tunnel overlay addresses of both endpoints should be in the same subnet.

In the following example, the tunnel endpoints on Dallas and Houston form an overlay network in the 10.1.1.0/24 address range. The Dallas router's IP address on the Internet is 11.1.1.1, and the Houston router's is 22.2.2.2. The graphic shows a GRE/IP packet sent from Dallas to Houston.



The following commands are used to configure the GRE tunnel between the two routers:

Dallas:

```
interface tunn1
ip address 10.1.1.1 255.255.255.0 ! Dallas's overlay address
tunnel source 11.1.1.1 ! Dallas's underlay address
tunnel destination 22.2.2.2 ! Houston's underlay address
```

Houston:

```
interface tunn1
ip address 10.1.1.2 255.255.255.0 ! Houston's overlay address
tunnel source 22.2.2.2 ! Houston's underlay address
tunnel destination 11.1.1.1 ! Dallas's underlay address
```

After configuring this, a ping through the overlay network between Dallas and Houston results in the following packet:

```
Internet Protocol Version 4, Src: 11.1.1.1, Dst: 22.2.2.2
Generic Routing Encapsulation (IP)
Internet Protocol Version 4, Src: 10.1.1.1, Dst: 10.1.1.2
Internet Control Message Protocol
```

Notice the top IP header source, 11.1.1.1. It is the same IP address used in Dallas's tunnel source command. The destination IP address is 22.2.2.2, which is the IP address used in Dallas's tunnel destination command. The GRE/IP header uses the values provided from these two commands to build the outer IP header. The next IP header is the original header for ICMP traffic that was

delivered through the overlay. 10.1.1.1 is Dallas's overlay address, and 10.1.1.2 is Houston's overlay address.

The tunnel is fully functional. Now the Dallas and Houston routers need a way to direct traffic through their tunnels. Dynamic routing protocols such as EIGRP are a good choice for this. By simply enabling EIGRP on the tunnel and LAN interfaces on the router, the routing table is populated appropriately:

```
Dallas#show ip route | begin Gate
Gateway of last resort is not set

    10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        10.1.1.0/24 is directly connected, Tunnell1
L        10.1.1.1/32 is directly connected, Tunnell1
    11.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        11.1.1.0/24 is directly connected, Ethernet0/0
L        11.1.1.1/32 is directly connected, Ethernet0/0
22.0.0.0/24 is subnetted, 1 subnets
R        22.2.2.0 [120/1] via 11.1.1.2, 00:00:22, Ethernet0/0
    172.16.0.0/16 is variably subnetted, 3 subnets, 2 masks
C        172.16.10.0/24 is directly connected, Loopback1
L        172.16.10.1/32 is directly connected, Loopback1
D        172.16.20.0/24 [90/27008000] via 10.1.1.2, 00:01:03, Tunnell1
```

The routing entry for the 172.16.20.0/24 network lists the tunnel interface 1 as the exit interface. When Dallas routes a packet destined to this network, it forwards to tunnel interface 1 and performs the encapsulation, as shown previously. The Dallas router then performs an additional routing lookup against the new outer tunnel destination IP address (22.2.2.2 in case of Dallas-to-Houston traffic). The second IP lookup carries the requirement that in order for Dallas to successfully route through the tunnel, it must have a route to the remote tunnel endpoint, Houston. In this example, RIP (shown in red) is being used to exchange underlay routing information between the ISP and the Dallas and Houston routers. Using RIP, Dallas has learned a route to 22.2.2.2, which is Houston's underlay address.

The result is that connectivity between the LANs succeeds, as evidenced by this **traceroute** output:

```
Dallas#traceroute 172.16.20.1
```

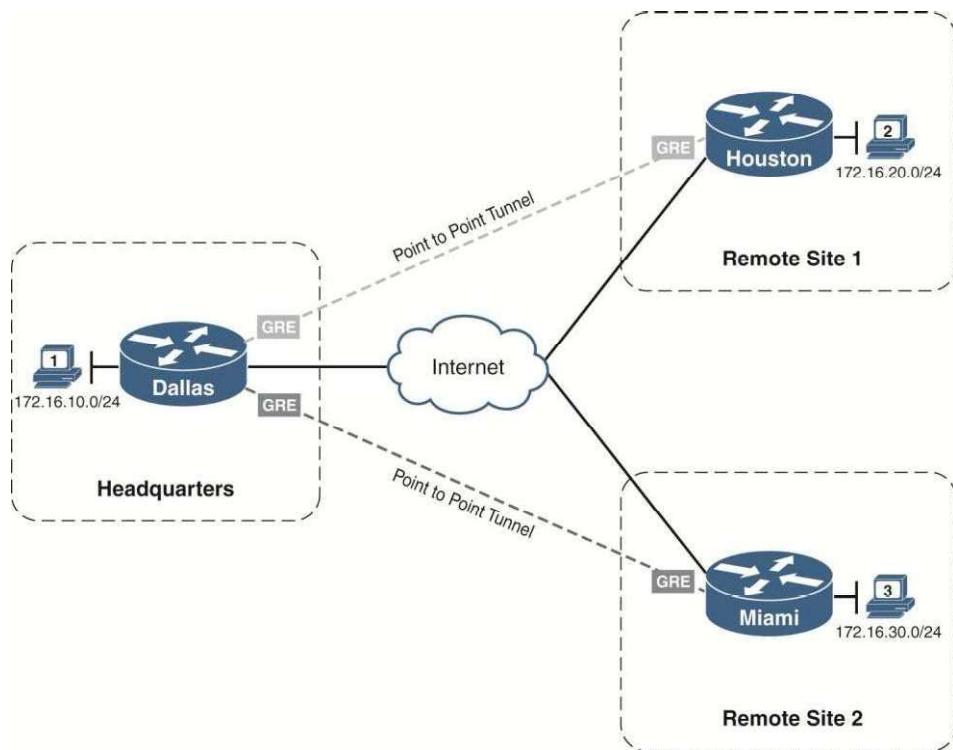
```
Type escape sequence to abort.
```

```
Tracing the route to 172.16.20.1
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

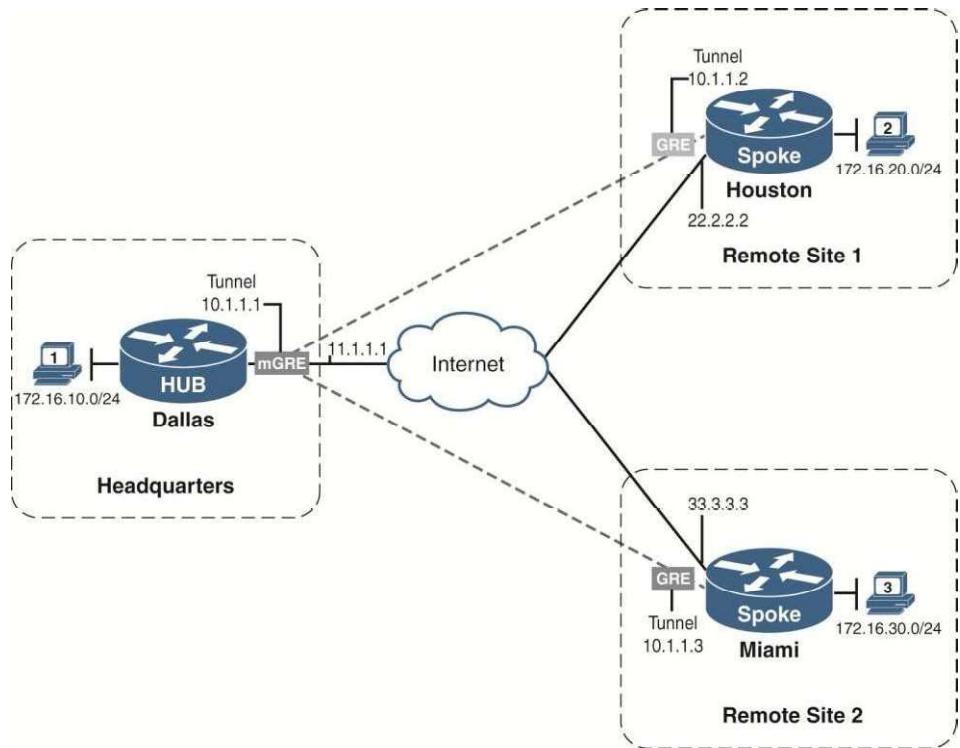
```
1 10.1.1.2 1 msec 5 msec 5 msec
```

Because of the limitation of setting static source and destination addresses, GRE tunnels are by nature point-to-point tunnels. It is not possible to configure a GRE tunnel to have multiple destinations by default. If the goal is only connecting two sites, then a point-to-point GRE tunnel is a great solution. However, as sites are added, additional point-to-point tunnels are required to connect them. Take the example below, where the company in our example has expanded into a second remote site, in Miami.



To configure basic connectivity between the two remote sites and the main headquarters site in Dallas, two point-to-point GRE tunnels would need to be configured. In addition, if Houston and Miami required direct communication, a third point-to-point tunnel would be needed. If more sites are added, the number of tunnels grows along with them. In addition to configuring more tunnels, the company would need to ensure that each tunnel exists in a separate subnet, creating multiple isolated overlay networks.

An alternative to configuring multiple point-to-point GRE tunnels is to use **multipoint GRE tunnels** to provide the connectivity desired.



Multipoint GRE (mGRE) tunnels are similar in construction to point-to-point GRE tunnels with the exception of one command: the **tunnel destination** command. Instead of declaring a static destination, no destination is declared, and instead the **tunnel mode gre multipoint** command is issued, as shown in the example below (on the Dallas router):

Dallas:

```
interface tunn1
ip address 10.1.1.1 255.255.255.0
tunnel source 11.1.1.1
tunnel destination 22.2.2.2 ! static destination removed
tunnel mode gre multipoint
```

How does Dallas know what destination to set for the GRE/IP packet created by the tunnel interface?

The easy answer is that, on its own, there is no way for Dallas to glean the destination address without the help of an additional protocol, such as **Next Hop Resolution Protocol (NHRP)**. NHRP was originally designed to allow routers connected to non-broadcast multiple-access (NBMA) networks to discover the proper next-hop mappings to communicate together. It is specified in RFC 2332. NBMA networks faced a similar issue as mGRE tunnels. In broadcast networks, protocols such as ARP use broadcast or multicast messages to dynamically discover endpoints on the network.

The best example of this is on Ethernet-based networks. Before a station can send to a remote station, it needs to know the MAC address of the remote station to build the Ethernet frame. ARP provides a means by which the sending station can discover and map the Layer 3 address of the receiving station to the receiving station's MAC address, using broadcast packets. This exchange is made possible because Ethernet infrastructure devices (namely switches) provide a means by which a single message can be sent from one station and replicated to all stations connected to the LAN.

In NBMA networks, there is no mechanism for doing this. For example, Dallas cannot simply send a broadcast out of the tunnel interface to dynamically discover the Houston router's address in the underlay for two reasons:

- In order to encapsulate the broadcast using the tunnel interface, Dallas needs to have a destination address for the outer GRE/IP packet.
- Even if Dallas placed a broadcast address in the destination of the IP packet, the underlay network (the public Internet in this case) does not support replication of broadcast packets. The ISP router would simply drop the traffic.

Looking at the two points above, it becomes apparent that by converting the point-to-point GRE tunnel to an mGRE tunnel, the 10.1.1.0/24 overlay network has become an NBMA network. Without the ability to support broadcast communication on the overlay, routers are unable to support broadcast-based network discovery protocols such as ARP, dynamic routing protocol hello messages, and the like. Missing this ability severely stunts the usefulness of mGRE with regard to dynamic scalability. However, with the overlay functioning as an NBMA network, NHRP can step in to glue it back together.

NHRP accomplishes this by keeping a mapping table of Layer 3 overlay addresses to Layer 3 underlay address pairs. For example, the NHRP table on Dallas would include a mapping of 10.1.1.2/22.2.2.2 for the Houston router. When traffic is forwarded out the mGRE interface on Dallas with NHRP configured, Dallas can consult its NHRP table to find the appropriate tunnel destination.

This solution solves the problem of where Dallas can get the information needed to route the packet, but it does not solve how it receives the information. To do so, NHRP organizes the routers connected to the same NHRP NBMA domain into one of two categories. They are either **next hop servers (NHSs)** or **next hop clients (NHCs)**. An NHC is configured with the overlay address of the router acting as the NHS. When a client comes online in an NHRP-enabled NBMA network, it first advertises its own overlay-to-underlay mapping to its configured NHS by using a special **NHRP registration message**. The registration message includes the NHC's overlay and NBMA network layer addressing information. The NHS receives and stores this information in its NHRP table.

All active clients attached to the NHRP-enabled NBMA must register with the NHS. When the client needs to send traffic to another router attached to the NBMA network, the client can send an NHRP resolution message to the NHS router (for which it already has a static mapping), requesting the appropriate mapping information. If the intended router is online, it is in one of two states: It has registered with an NHS (meaning it is a client), or it is an NHS (and contains the appropriate mapping

table). In either case, the sending client can logically consult the NHS for the mapping information with a high probability of receiving a reply.

The NHS forwards this resolution request to the proper client, which responds directly to the querying client.

NOTE:

In previous implementations of NHRP on Cisco IOS, the NHS would respond directly to the requesting client. This functionality was changed in later releases as an enhancement to support hierarchical DMVPN setups.

The collection of routers that share NHRP information is known as an *NHRP domain*. An NHRP domain is identified by its NHRP network identifier, which is a configurable numeric value. All routers must be configured with an **NHRP network identifier** in order to process NHRP messages. Doing so allows the router to correlate received NHRP information on multiple interfaces with a specific NHRP domain. This allows multiple interfaces to participate in the same or different NHRP domains as the design requires. The NHRP network identifier is a locally significant value that does NOT have to match between two routers.

To fix the tunnel configuration on the Dallas router, NHRP should be configured on the Dallas and Houston routers. NHS routers should have connectivity to all spokes they will service. Looking at the diagram, the Miami and Houston routers both have GRE tunnel connectivity to the Dallas router. With this information, it makes sense to designate the Dallas router as the NHS for the NHRP domain.

The Houston router is configured with the **ip nhrp nhs 10.1.1.1** command to designate the Dallas router as the proper NHS. Both routers use the **ip nhrp network-id 1** command to enable NHRP messaging on the tunnel interfaces:

Dallas:

```
interface Tunnel1
  ip address 10.1.1.1 255.255.255.0
  ip nhrp network-id 1
  tunnel source 11.1.1.1
  tunnel mode gre multipoint
end
```

Houston:

```

interface Tunnel1
ip address 10.1.1.2 255.255.255.0
ip nhrp network-id 1
ip nhrp nhs 10.1.1.1 ! Overlay address of Dallas, the NHS
tunnel source 22.2.2.2
tunnel destination 11.1.1.1
end

```

Now a ping through the overlay between Dallas and Houston is successful:

```

Dallas#ping 10.1.1.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.1.2, timeout is 2 seconds:
!!!!!

```

The ping succeeds because Dallas has the necessary overlay-to-underlay mapping information in its NHRP mapping table. The contents of the NHRP mapping table can be displayed using the **show ip nhrp** command:

```

Dallas#show ip nhrp
10.1.1.2/32 via 10.1.1.2 ! Overlay address of Houston
Tunnel1 created 00:05:15, expire 01:55:29
Type: dynamic, Flags: unique registered used nhop
NBMA address: 22.2.2.2 ! Underlay address of Houston

```

NOTE:

Houston's NHRP mapping table will be empty. Because it uses a point-to-point GRE tunnel, there is no need for it to store overlay-to-underlay mappings. Its tunnel can only be routed directly to Dallas.

Communication between Dallas and Houston's overlay address is successful, but the communication between their LAN interfaces is not. The routing table on Dallas shown below no longer lists the 172.16.20.0/24 network. EIGRP is still configured on the interfaces, but EIGRP prefixes are not being exchanged.

```

Dallas#show ip route | begin Gate
Gateway of last resort is not set

```

```
10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C     10.1.1.0/24 is directly connected, Tunnel1
L     10.1.1.1/32 is directly connected, Tunnel1
11.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C     11.1.1.0/24 is directly connected, Ethernet0/0
L     11.1.1.1/32 is directly connected, Ethernet0/0
22.0.0.0/24 is subnetted, 1 subnets
R     22.2.2.0 [120/1] via 11.1.1.2, 00:00:04, Ethernet0/0
33.0.0.0/24 is subnetted, 1 subnets
R     33.3.3.0 [120/1] via 11.1.1.2, 00:00:04, Ethernet0/0
172.16.0.0/16 is variably subnetted, 2 subnets, 2 masks
C     172.16.10.0/24 is directly connected, Loopback1
L     172.16.10.1/32 is directly connected, Loopback1
```

Adding to the difficulties, the EIGRP neighborship between Dallas and Houston is flapping on the Dallas side:

Dallas:

```
%DUAL-5-NBRCHANGE: EIGRP-Ipv4 1: Neighbor 10.1.1.2 (Tunnel1) is up: new adjacency
```

```
%DUAL-5-NBRCHANGE: EIGRP-Ipv4 1: Neighbor 10.1.1.2 (Tunnel1) is down: retry limit exceeded
```

```
%DUAL-5-NBRCHANGE: EIGRP-Ipv4 1: Neighbor 10.1.1.2 (Tunnel1) is up: new adjacency
```

The reason for this is that while NHRP has filled in to provide the tunnel destination, it still has not solved the underlying problem of the overlay being an NBMA network. EIGRP hello messages are multicast between routers. Taking an example of two routers (R1 and R2), the basic EIGRP neighborship process is as follows:

1. R1 and R2 exchange multicast hello messages.

2. R2 receives the hello message and creates a neighbor entry for R1 in its neighbor table in the “pending” state.
3. R2 sends a unicast update message with the “init” bit set to accelerate the hello process.
4. R1 receives the update message with the “init” bit set and sends back an acknowledgment packet.
5. Steps 2–4 are repeated but in reverse.
6. The neighborship is established, and the two routers begin exchanging prefixes.

Through this sequence, both multicast and unicast connectivity is verified between the two EIGRP routers. When the neighborship process above is applied to the mGRE interface on the Dallas router, the Dallas router is unable to tunnel hello packets to the Houston router because the destination address for EIGRP hello packets is 224.0.0.10.

IP addresses in that range are called *multicast IP addresses*. Typically, when EIGRP is enabled on an interface, the router can encode the Ethernet frame with a multicast MAC address that signifies to intermediate devices that this traffic should be forwarded out all switch ports in the same VLAN, much like a broadcast packet. The problem Dallas is facing is that EIGRP is enabled on its mGRE interface. Because the interface is mGRE, Dallas does not have a hard-coded destination address for the GRE/IP header, as explained earlier. As a result, Dallas cannot encode the GRE/IP packet to tunnel the multicast traffic across the underlay. The following events occur:

1. Houston sends a hello packet to Dallas. It can do so because it has a point-to-point GRE tunnel. All traffic through the tunnel goes to Dallas as the statically configured tunnel destination.
2. Dallas receives the hello, initiates the neighbor structure, and sends a unicast update message with the init bit set to Houston.
3. Because Houston has not received a hello packet from Dallas, it does not accept its update packet.
4. Dallas waits to receive an acknowledgment for its update packet sent to Houston and receives none.
5. Dallas clears the neighbor entry for Houston.
6. Houston sends another hello packet to Dallas.

This cycle repeats until Dallas is able to send a multicast hello to Houston. Specifically, because Houston never receives a multicast packet from Dallas (it only receives a unicast packet at step 2), Houston always rejects Dallas’s update packet. Dallas can never send a multicast hello packet to Houston because it does not have sufficient forwarding information (that is, destination NBMA address) to send the multicast through its mGRE interface.

Dallas needs a mechanism by which it can associate multicast traffic with a specific NBMA address in order to send the multicast as unicast over the underlay. NHRP solves this problem by storing a separate NHRP table specifically for multicast traffic. This table lists all of the underlay addresses to which multicast traffic should be sent. When the router needs to forward multicast packets out of an NHRP-enabled interface, it replicates the packet as unicast toward each underlay address in the NHRP multicast table. This process, often referred to as **pseudo-multicasting**, is very similar to Frame Relay pseudo-multicasting in Frame Relay hub-and-spoke topologies.

The NHRP multicast table can be configured statically using **ip nhrp map multicast underlay-ip-address** command. Such a configuration is tedious on an NHS because there is no telling how many NHRP clients will register mappings with the NHS. So instead, the **ip nhrp map multicast dynamic** command tells the NHS to automatically add a dynamic NHRP multicast entry for every client that registers with the NHS. To solve the connectivity issues between Dallas and Houston, this command is configured on Dallas. Then, the tunnel interface on Houston is flapped to force it to re-register with Dallas. The **show ip nhrp multicast** command verifies the multicast entry on the Dallas router. The EIGRP neighbors come up, and pings are successful between the LAN interfaces again:

```
Dallas#show run interface tunnell1

interface Tunnell1
  ip address 10.1.1.1 255.255.255.0
  no ip redirects
  ip nhrp map multicast dynamic
  ip nhrp network-id 1
  tunnel source 11.1.1.1
  tunnel mode gre multipoint

Houston(config)#interface tunnell1
Houston(config-if)#shut

%LINKPROTO-5-UPDOWN: Line protocol on Interface Tunnell1, changed state to
down

%LINK-5-CHANGED: Interface Tunnell1, changed state to administratively down

Houston(config-if)#no shut
```

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnell, changed state to up  
%LINK-3-UPDOWN: Interface Tunnell, changed state to up
```

```
%DUAL-5-NBRCHANGE: EIGRP-Ipv4 1: Neighbor 10.1.1.2 (Tunnell) is up: new  
adjacency
```

```
Dallas#show ip nhrp multicast
```

I/F	NBMA address
Tunnell	22.2.2.2
	Flags: dynamic (Enabled)

```
Dallas#show ip route 172.16.20.0
```

```
Routing entry for 172.16.20.0/24
```

```
Known via "eigrp 1", distance 90, metric 27008000, type internal
```

```
Redistributing via eigrp 1
```

```
Last update from 10.1.1.2 on Tunnell, 00:00:13 ago
```

```
Routing Descriptor Blocks:
```

```
* 10.1.1.2, from 10.1.1.2, 00:00:13 ago, via Tunnell
```

```
    Route metric is 27008000, traffic share count is 1
```

```
    Total delay is 55000 microseconds, minimum bandwidth is 100 Kbit
```

```
    Reliability 255/255, minimum MTU 1476 bytes
```

```
    Loading 1/255, Hops 1
```

```
Dallas#ping 172.16.20.1 source 172.16.10.1
```

```
Type escape sequence to abort.
```

```
Sending 5, 100-byte ICMP Echos to 172.16.20.1, timeout is 2 seconds:
```

```
Packet sent with a source address of 172.16.10.1
```

!!!!

Taking a step back can help you understand what this seemingly complex configuration has accomplished: Before, static point-to-point GRE tunnels, each representing a separate subnet, were required to form the overlay necessary to connect the Dallas and Houston sites. When the Miami site was added, another tunnel was required to connect Miami to Dallas and, if necessary, to connect Miami with Houston.

The mGRE/NHRP configuration outlined allows only a single tunnel interface to be required per router in order to connect Dallas to Houston and Dallas to Miami. All Miami needs is a point-to-point GRE interface configured with Dallas as the NHRP NHS, and connectivity will be established. Through EIGRP, Miami can also receive routes for the networks behind Dallas as well. Most importantly, the tunnel interfaces on Dallas, Houston, and Miami are all part of the same IP subnet and participate in the same NHRP domain.

The total configuration on Miami to join it to the cloud and the appropriate verification commands are as follows:

Miami:

```
interface Tunnel1
ip address 10.1.1.3 255.255.255.0
ip nhrp network-id 1
ip nhrp nhs 10.1.1.1
tunnel source 33.3.3.3
tunnel destination 11.1.1.1
end
!
router eigrp 1
network 10.1.1.0 0.0.0.255
network 172.16.30.0 0.0.0.255
```

```
Miami#show ip route | begin Gate
```

```
Gateway of last resort is not set
```

```

10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C      10.1.1.0/24 is directly connected, Tunnel1
L      10.1.1.3/32 is directly connected, Tunnel1
      11.0.0.0/24 is subnetted, 1 subnets
R      11.1.1.0 [120/1] via 33.3.3.1, 00:00:16, Ethernet0/0
      22.0.0.0/24 is subnetted, 1 subnets
R      22.2.2.0 [120/1] via 33.3.3.1, 00:00:16, Ethernet0/0
      33.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C      33.3.3.0/24 is directly connected, Ethernet0/0
L      33.3.3.3/32 is directly connected, Ethernet0/0
      172.16.0.0/16 is variably subnetted, 3 subnets, 2 masks
D      172.16.10.0/24 [90/27008000] via 10.1.1.1, 00:00:12, Tunnel1
C      172.16.30.0/24 is directly connected, Loopback1
L      172.16.30.1/32 is directly connected, Loopback1

```

```
Miami#ping 172.16.10.1 source 172.16.30.1
```

Type escape sequence to abort.

```

Sending 5, 100-byte ICMP Echos to 172.16.10.1, timeout is 2 seconds:
Packet sent with a source address of 172.16.30.1
!!!!!

```

In the routing table output above, you can see that Miami receives a route for the LAN behind Dallas (the 172.16.10.0/24 network) but not for the Houston LAN (172.16.20.0/24 network). Dallas appears to not be advertising the Houston LAN to Miami. This reason is not unknown to Distance Vector protocols as it is caused by a concept known as **split horizon**. Split horizon prevents a router from advertising a route for a prefix out the same interface the router uses to reach that prefix. In Dallas's case, it learns the 172.16.20.0/24 route from Houston, which is connected to its tunnel interface. It cannot advertise that same network toward Miami out the same tunnel 1 interface.

To alleviate the problem here, split horizon can be disabled on the tunnel interface of Dallas with the **no ip split-horizon eigrp** command. Notice that Miami now learns of the LAN network behind Houston:

```

Dallas(config)#interface tunnell
Dallas(config-if)#no ip split-horizon eigrp 1

%DUAL-5-NBRCHANGE: EIGRP-IPv4 1: Neighbor 10.1.1.3 (Tunnell) is resync:
split horizon changed

Miami#show ip route eigrp | begin Gateway
Gateway of last resort is not set

    172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
D        172.16.10.0/24 [90/27008000] via 10.1.1.1, 00:06:57, Tunnell
D        172.16.20.0/24 [90/28288000] via 10.1.1.1, 00:01:07, Tunnell

```

A ping between the LAN interfaces of Miami and Houston confirms the connectivity:

```

Miami#ping 172.16.20.1 source 172.16.30.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.20.1, timeout is 2 seconds:
Packet sent with a source address of 172.16.30.1
!!!!!

```

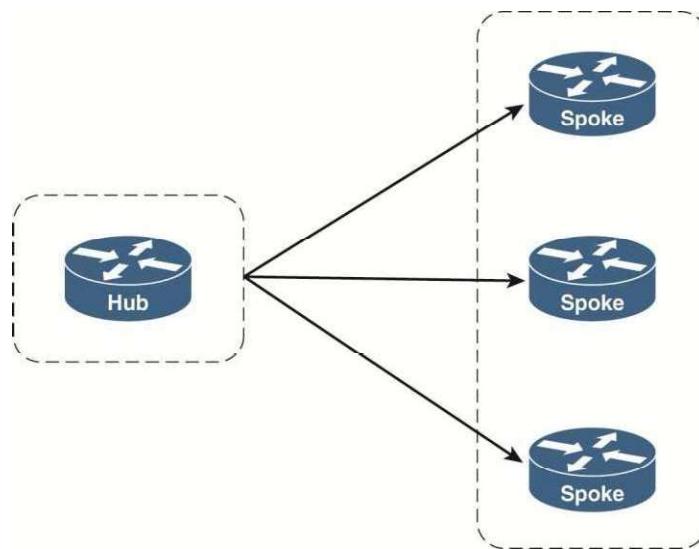
The mGRE/NHRP solution reduces the configuration complexity to something that can be easily scripted out using various tools. Adding a new site to the existing VPN only requires direct configuration of the new devices. The devices come online, register with Dallas, receive their routing updates, and are ready to participate in the overlay network without the need to consult a service provider to provision private WAN circuits.

Using mGRE and NHRP in this way forms the underlying architecture that powers the Dynamic Multipoint VPN (DMVPN) solution. The routers above are actually participating in a DMVPN cloud. It is *dynamic* because routers can be added or removed with minimal configuration. It is *multipoint* because Dallas, as the NHS, can form tunnels to all clients that register with it, using a single multipoint GRE tunnel interface. It is virtually private because it is tunneled over an existing underlay, presenting the illusion of direct connectivity for all of the devices involved.

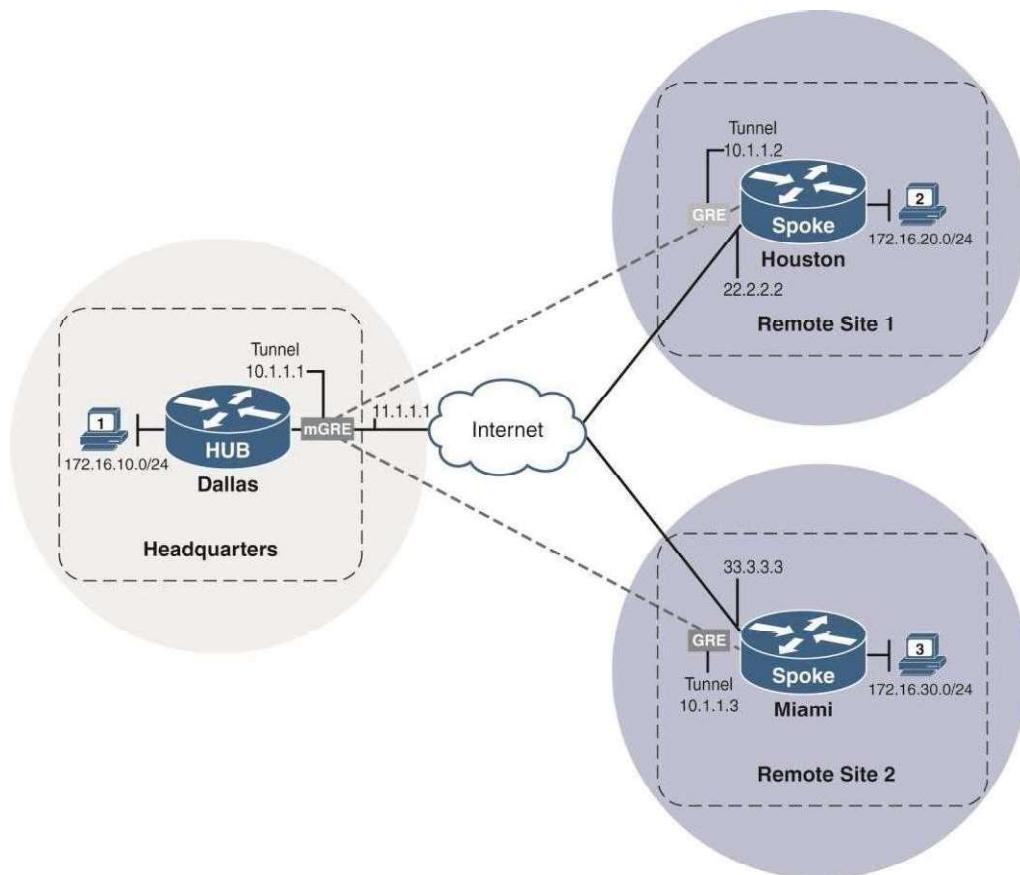
DMVPN Mechanics

The section above outlines the use case that DMVPN was created to solve. It also introduces the symbiotic relationship between NHRP and mGRE that forms the building blocks for a DMVPN cloud. This section delves a bit more deeply into the operation and mechanics of a DMVPN cloud.

Although DMVPN uses NHRP for its address resolution, it does not borrow from NHRP terminology but instead uses its own. DMVPN clouds are constructed as **hub-and-spoke** topologies. In a hub-and-spoke topology, networking devices (the spokes) connect through a common, shared network device that acts as the aggregation point for the topology. This common point is termed the **hub** of the hub-and-spoke network.



DMVPN clouds are, at their core, hub-and-spoke networks. The DMVPN hub router acts as the central aggregation point for all of the spoke routers that connect to it in its DMVPN cloud. In the NHRP network created in the first section, the Dallas router is considered the hub of the DMVPN cloud, and Houston and Miami routers are the spokes.



This distinction is important as hubs and spokes have different responsibilities within the DMVPN cloud. DMVPN hub routers also act as NHRP NHS routers that store and receive mapping information from the NHCs. DMVPN spoke routers are the NHRP NHCs, which send their mapping information to a hub router that acts as the spoke routers' NHSs. In the example above, Houston and Miami are the spoke routers that send their NHRP mapping information to the hub router Dallas, through a process called **NHRP registration**.

Going back in time to a point where both Dallas and Houston were first configured and following the entire registration process will help you gain a better understanding of the NHRP registration process. Recall the following configurations on the Dallas and Houston routers from the section above:

On Hub—Dallas:

```
interface Tunnel1
ip address 10.1.1.1 255.255.255.0
no ip split-horizon eigrp 1
```

```
ip nhrp map multicast dynamic  
ip nhrp network-id 1  
tunnel source 11.1.1.1  
tunnel mode gre multipoint
```

On Spoke—Houston:

```
interface Tunnel1  
ip address 10.1.1.2 255.255.255.0  
ip nhrp network-id 1  
ip nhrp nhs 10.1.1.1  
tunnel source 22.2.2.2  
tunnel destination 11.1.1.1
```

Dallas is configured with an **mGRE** tunnel, while Houston is configured with a **point-to-point** GRE tunnel. EIGRP is also configured to run on the tunnel interfaces of the routers. Finally, EIGRP’s split-horizon feature has been disabled on the tunnel interface on Dallas.

Recall that with Dallas being configured with an mGRE tunnel interface, there is no tunnel destination configured for the GRE tunnel interface to use in its GRE tunnel IP header. Without this information, Dallas cannot forward traffic to Houston in the overlay, as evidenced by the unsuccessful **ping** output below:

```
Dallas#ping 10.1.1.2  
  
Type escape sequence to abort.  
Sending 5, 100-byte ICMP Echos to 10.1.1.2, timeout is 2 seconds:  
.....  
Success rate is 0 percent (0/5)
```

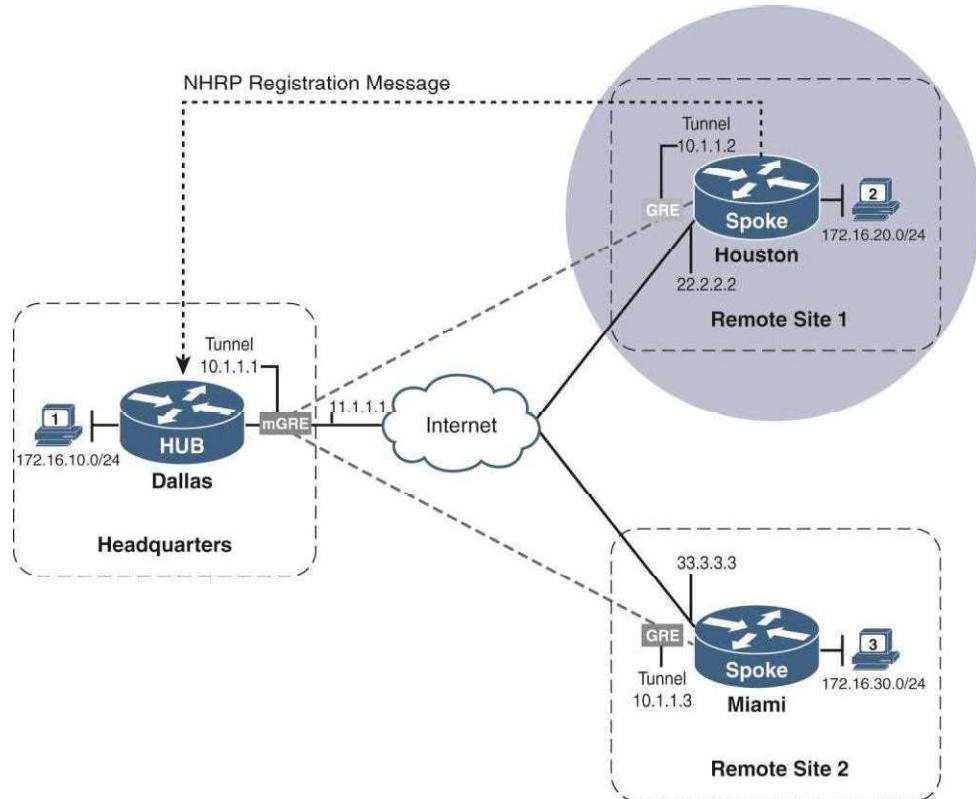
NHRP steps in to provide this information by maintaining an NBMA-to-overlay IP address mapping table on the router. This NHRP mapping table is consulted to fill in the required information when sending packets across the DMVPN network.

Just as static routes can be configured for the IP routing table, the NHRP mapping table can be populated using static NHRP mapping command (**ip nhrp map**). However, populating the mapping

table this way carries the same drawbacks at scale as configuring static routes. The configuration complexity grows as many more spokes are connected to the same hub router.

The designers of DMVPN take advantage of the NHS/NHC relationship to allow the spokes to **dynamically register** their NHRP mapping information with the hub routers. Spokes are configured with the **ip nhrp nhs** command, which identifies which router functions as the NHS.

In the example topology, Dallas is the hub of the DMVPN and therefore is also the NHS for the NHRP network. Once Houston comes online, it first sends an **NHRP registration request** to Dallas in order to register its NBMA-to-overlay address mapping.



With the help of the **debug nhrp packet** and **debug nhrp detail** commands on both Dallas and Houston, the registration process can be observed in real time. Once Houston's tunnel interface comes online, Houston sends an NHRP registration packet to Dallas. It knows Dallas is the NHS because Dallas's IP address is identified in the **ip nhrp nhs 10.1.1.1** command on Houston's tunnel interface. This NHRP registration packet is shown in the debugging output below:

Houston:

```
NHRP: Send Registration Request via Tunnell1 vrf 0, packet size: 88  
src: 10.1.1.2, dst: 10.1.1.1
```

```

(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 88 extoff: 52
(M) flags: "unique nat ", reqid: 11
src NBMA: 22.2.2.2
src protocol: 10.1.1.2, dst protocol: 10.1.1.1
(C-1) code: no error(0)
      prefix: 32, mtu: 17916, hd_time: 7200

```

The first two lines of the debugging output indicate that an NHRP registration request packet is being sent to 10.1.1.1 (NHS). The source of the packet is 10.1.1.2 (Houston's tunnel address). The mapping information is split in the src NBMA and src protocol fields of the registration packet. The src NBMA field indicates the NBMA address of the device that is sending the NHRP registration request. In this case, Houston's NBMA address is 22.2.2.2, the address configured as the tunnel source address of its GRE interface connected to the DMVPN. The src protocol address is the overlay IP address of the device sending the registration request. For Houston, this address is its tunnel interface address 10.1.1.2, which is the tunnel IP address assigned to Houston's GRE interface connected to the DMVPN.

These pieces of information combined are received on the hub router, Dallas, as shown below:

```

NHRP: Receive Registration Request via Tunnell vrf 0, packet size: 88
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 88 extoff: 52
(M) flags: "unique nat ", reqid: 11
src NBMA: 22.2.2.2
src protocol: 10.1.1.2, dst protocol: 10.1.1.1
(C-1) code: no error(0)
      prefix: 32, mtu: 17916, hd_time: 7200
      addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
NHRP: Adding Tunnel Endpoints (VPN: 10.1.1.2, NBMA: 22.2.2.2)
NHRP: Successfully attached NHRP subblock for Tunnel Endpoints (VPN: 10.1.1.2, NBMA: 22.2.2.2)

```

Once Dallas receives the mapping, it adds the information to the NHRP mapping table. Notice that the debugging output above indicates the VPN address as 10.1.1.2 for Houston. In this context, VPN indicates the overlay protocol address for the DMVPN.

The resulting mapping on Dallas can be verified using the **show ip nhrp** and **show dmvpn** commands on Dallas:

```
Dallas#show ip nhrp
```

```
10.1.1.2/32 via 10.1.1.2
```

```
Tunnell created 00:55:33, expire 01:44:27
```

```
Type: dynamic, Flags: unique registered used nhop
```

```
NBMA address: 22.2.2.2
```

```
Dallas#show dmvpn
```

```
Legend: Attrb --> S - Static, D - Dynamic, I - Incomplete
```

```
N - NATed, L - Local, X - No Socket
```

```
# Ent --> Number of NHRP entries with same NBMA peer
```

```
NHS Status: E --> Expecting Replies, R --> Responding, W --> Waiting
```

```
UpDn Time --> Up or Down Time for a Tunnel
```

```
=====
```



```
Interface: Tunnell, IPv4 NHRP Details
```

```
Type:Hub, NHRP Peers:1,
```

```
# Ent Peer NBMA address Peer Tunnel Add State UpDn Tm Attrb
```

```
-----
```



```
1 22.2.2.2          10.1.1.2      UP 01:02:32      D
```

The **show ip nhrp** output lists the mapping information for the 10.1.1.2/32 address on the overlay network. The NBMA address is listed as 22.2.2.2, and it is flagged as a **dynamic** entry because it was added as a result of the NHRP registration process.

The **show dmvpn** output shows the entry with the NBMA address 22.2.2.2 as the Peer NBMA address and the overlay address 10.1.1.2 as the Peer Tunnel Add. It lists the state of the tunnel as UP. The tunnel itself can be in many other states, depending on which part of the NHRP registration process the entry is currently going through. The last column, the Attrb column, signifies that the entry was created dynamically.

In response to the NHRP registration request from Houston, Dallas replies to Houston with an **NHRP registration reply** message. The exchange is shown below:

On Dallas:

```
NHRP: Attempting to send packet through interface Tunnell1 via DEST dst  
10.1.1.2
```

```
NHRP: Send Registration Reply via Tunnell1 vrf 0, packet size: 108
```

```
src: 10.1.1.1, dst: 10.1.1.2
```

```
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
```

```
shtl: 4 (NSAP), sstl: 0 (NSAP)
```

```
pktsz: 108 extoff: 52
```

```
(M) flags: "unique nat ", reqid: 11
```

```
src NBMA: 22.2.2.2
```

```
src protocol: 10.1.1.2, dst protocol: 10.1.1.1
```

```
(C-1) code: no error(0)
```

```
prefix: 32, mtu: 17916, hd_time: 7200
```

```
addr_len: 0 (NSAP), subaddr_len: 0 (NSAP), proto_len: 0, pref: 0
```

```
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:  
22.2.2.2
```

```
NHRP: 132 bytes out Tunnell1
```

On Houston:

```
NHRP: Receive Registration Reply via Tunnell1 vrf 0, packet size: 108
```

```
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
```

```
shtl: 4 (NSAP), sstl: 0 (NSAP)
```

```
pktsz: 108 extoff: 52  
(M) flags: "unique nat ", reqid: 11  
src NBMA: 22.2.2.2  
src protocol: 10.1.1.2, dst protocol: 10.1.1.1  
(C-1) code: no error(0)
```

When the process is complete, Houston has successfully registered with Dallas. Dallas has been configured with an mGRE tunnel interface. There is no tunnel destination configured for the GRE tunnel interface to use in its GRE tunnel IP header. With Houston registered, Dallas now has sufficient information to forward packets out its tunnel interface to Houston. The successful ping below demonstrates communication between Dallas and Houston:

```
Dallas#ping 10.1.1.2  
  
Type escape sequence to abort.  
Sending 5, 100-byte ICMP Echos to 10.1.1.2, timeout is 2 seconds:  
!!!!!  
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/5 ms
```

Adding a new spoke to the DMVPN is as simple as pasting the same configuration template from Houston on router Miami, taking care to change the tunnel IP address and tunnel source address accordingly:

On Miami:

```
interface Tunnell  
ip address 10.1.1.3 255.255.255.0  
ip nhrp network-id 1  
ip nhrp nhs 10.1.1.1  
tunnel source 33.3.3.3  
tunnel destination 11.1.1.1
```

The output of the **show dmvpn** command from the hub Dallas verifies that the spoke Miami has registered with it dynamically as well. A ping from Dallas to Miami's tunnel IP address is also shown to succeed:

```
Dallas#show dmvpn
```

Legend: Attrb --> S - Static, D - Dynamic, I - Incomplete
N - NATed, L - Local, X - No Socket
Ent --> Number of NHRP entries with same NBMA peer
NHS Status: E --> Expecting Replies, R --> Responding, W --> Waiting
UpDn Time --> Up or Down Time for a Tunnel

```
=====
```

Interface: Tunnell, IPv4 NHRP Details

Type:Hub, NHRP Peers:2,

#	Ent	Peer NBMA address	Peer Tunnel Add	State	UpDn	Tm	Attrb
1	22.2.2.2		10.1.1.2	UP	01:10:58		D
1	33.3.3.3		10.1.1.3	UP	00:00:07		D

```
Dallas#ping 10.1.1.3
```

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.1.1.3, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/5 ms

The DMVPN hub router is not only responsible for the NHRP mapping table but also for the exchange of routing information between the spokes. Above, EIGRP is used as the routing protocol of choice and is enabled on the tunnel interfaces. EIGRP multicast hello packets discover neighbors. In the previous section, these multicasts needed a proper NBMA address to be pseudo-multicasted over the overlay.

NHRP fills this gap in as well by maintaining a multicast mapping table. This table can also be populated dynamically or statically. In DMVPN, routing protocol multicasts are sent only between the hub and spoke routers. This means the spokes need an NHRP multicast mapping only for the

hub—and not for the other spokes. It also means the hub router requires an NHRP multicast mapping for every spoke that registers with it on the DMVPN cloud. This information too can be dynamically filled in at the same time the hub creates the dynamic NHRP mapping for NBMA-to-overlay addressing through the **ip nhrp map multicast dynamic** command.

NOTE:

In the case where the spokes are configured with point-to-point GRE tunnels, there is no need to configure the NHRP multicast mapping on the spokes. A point-to-point GRE tunnel has only one destination, to which all packets can be sent: a destination that is hard coded as the **tunnel destination** configuration under the tunnel interface. Therefore, multicast traffic can always be sent out the point-to-point tunnel interfaces to this destination address. This is the case above, where the spokes have point-to-point GRE tunnels configured and the hub uses the mGRE tunnel.

However, multicast mappings would be required in cases where the spokes also use mGRE interfaces—specifically in DMVPN Phase 2 and Phase 3 designs.

With the **ip nhrp map multicast dynamic** command configured on the tunnel interface on Dallas, whenever a spoke registers with Dallas, it automatically adds the spoke's NBMA address to its NHRP multicast table. The output below from **debug nhrp detail** was taken when Houston first registered with Dallas. The **show ip nhrp multicast** command confirms the mapping:

On Dallas:

```
NHRP: Tul: Creating dynamic multicast mapping  NBMA: 22.2.2.2
```

```
NHRP: Added dynamic multicast mapping for NBMA: 22.2.2.2
```

```
Dallas#show ip nhrp multicast
```

I/F	NBMA address
Tunnel1	22.2.2.2
	Flags: dynamic (Enabled)

The same occurs for Miami. With EIGRP configured on the tunnel interfaces, EIGRP packets (multicast and unicast) will now be encapsulated within GRE and unicasted to the NBMA addresses of Houston and Miami by Dallas. The following captures show an example of such packets transmitted by Dallas to the spokes, Houston and Miami.

```
Frame 32: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
Ethernet II, Src: aa:bb:cc:00:01:00 (aa:bb:cc:00:01:00), Dst: aa:bb:cc:00:04:00
(aa:bb:cc:00:04:00)

Internet Protocol Version 4, Src: 11.1.1.1, Dst: 22.2.2.2
Generic Routing Encapsulation (IP)
```

```
Internet Protocol Version 4, Src: 10.1.1.1, Dst: 224.0.0.10
```

```
Cisco EIGRP
```

Frame 33: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0

Ethernet II, Src: aa:bb:cc:00:01:00 (aa:bb:cc:00:01:00), Dst: aa:bb:cc:00:04:00 (aa:bb:cc:00:04:00)

```
Internet Protocol Version 4, Src: 11.1.1.1, Dst: 33.3.3.3
```

Generic Routing Encapsulation (IP)

```
Internet Protocol Version 4, Src: 10.1.1.1, Dst: 224.0.0.10
```

```
Cisco EIGRP
```

EIGRP relationships are then formed between Dallas and the spokes, Houston and Miami. The **show ip eigrp neighbors** command output on Dallas verifies these EIGRP neighborships:

Houston:

```
Houston#show ip eigrp neighbors
```

```
EIGRP-IPv4 Neighbors for AS(1)
```

H	Address	Interface	Hold	Uptime	SRTT	RTO	Q	Seq
			(sec)	(ms)		Cnt	Num	
1	10.1.1.3	Tu1	12	00:09:20		5	1470	0 3
0	10.1.1.2	Tu1	12	00:19:36		13	1470	0 7

On completing the above configurations, the routing tables on Houston and Miami confirm that they have EIGRP routes of each other's LAN networks. A ping is issued between the LANs on Houston and Miami to confirm the communication:

On Houston:

```
Houston#show ip route eigrp | begin Gate
```

Gateway of last resort is not set

172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks

```
D      172.16.10.0/24 [90/27008000] via 10.1.1.1, 00:21:39, Tunnell
```

```
D      172.16.30.0/24 [90/28288000] via 10.1.1.1, 00:11:23, Tunnell
```

On Miami:

```
Miami#show ip route eigrp | begin Gate
```

```
Gateway of last resort is not set
```

```
    172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
```

```
D      172.16.10.0/24 [90/27008000] via 10.1.1.1, 00:11:40, Tunnell
```

```
D      172.16.20.0/24 [90/28288000] via 10.1.1.1, 00:11:40, Tunnell
```

On Houston:

```
Houston#ping 172.16.30.1 source 172.16.20.1
```

```
Type escape sequence to abort.
```

```
Sending 5, 100-byte ICMP Echos to 172.16.30.1, timeout is 2 seconds:
```

```
Packet sent with a source address of 172.16.20.1
```

```
!!!!!
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/5 ms
```

DMVPN Designs

The word *phase* is almost always connected to DMVPN design discussions. *DMVPN phase* refers to the version of DMVPN implemented in a DMVPN design. DMVPN as a solution was rolled out in different stages as the solution became more widely adopted to address performance issues and additional improvised features. There are three main phases for DMVPN:

Phase 1 - Hub-and-spoke

Phase 2 - Spoke-initiated spoke-to-spoke tunnels

Phase 3 - Hub-initiated spoke-to-spoke tunnels

The differences between the DMVPN phases are related to routing efficiency and the ability to create spoke-to-spoke tunnels.

Phase 1: Hub-and-Spoke

Earlier, we discussed the problem of providing WAN connectivity to allow branch sites of a corporation to communicate with each other. To create a scalable and dynamic WAN strategy, a DMVPN solution was implemented as an overlay network. This network used only a single tunnel interface on three routers to create a hub-and-spoke network. The hub router Dallas leverages mGRE and NHRP to dynamically create a tunnel between itself and the branch (spoke) office routers Houston and Miami, which both use traditional point-to-point GRE tunnels.

The resulting overall design mimics a traditional hub-and-spoke network, where all traffic between Miami and Houston first travels through Dallas. In this design, Miami and Houston cannot communicate directly with each other. This design implementation is referred to as **DMVPN Phase 1**.

DMVPN Phase 1 uses mGRE interfaces on the hub router and point-to-point GRE tunnel interfaces on the spoke routers. With point-to-point GRE interfaces on spoke routers, all traffic sent by spokes through the DMVPN overlay is forced through the hub router. In other words, traffic from a spoke to another spoke always traverses the hub. Because traffic is forced through the hub router, there is no reason for the spokes to retain specific routing information for networks behind other spoke routers. The routing tables on the spokes can be optimized to a single default route received from the hub router.

To prove this point, a traceroute is performed below on Houston to the network address 172.16.30.1 on Miami. This network is advertised via EIGRP by the hub router Dallas to Houston. As seen below, the path taken to reach this address is from Houston, then to Dallas, and then to Miami.

```
Houston#show ip route eigrp | begin Gateway

Gateway of last resort is not set

172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
D      172.16.10.0/24 [90/27008000] via 10.1.1.1, 01:00:20, Tunnell1
D      172.16.30.0/24 [90/28288000] via 10.1.1.1, 01:00:19, Tunnell1
```

```
Houston#traceroute 172.16.30.1 source 172.16.20.1
```

Type escape sequence to abort.