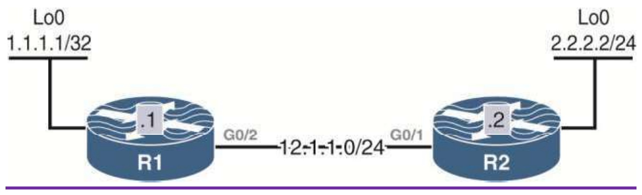
Lab 1
Establishing a BGP Session Using the Correct TTL Value



This lab should be conducted on the Enterprise POD.

BGP Peer Session Overview

BGP, unlike other routing protocols, such as OSPF and EIGRP, does not implement its own transport when forming neighbor relationships (also called peer relationships in BGP terminology) with other BGP-speaking routers. Instead, BGP leverages TCP as its transport protocol, running over the well-known BGP TCP port 179. This means that in order for a BGP peering session to come up between two routers, they must first establish a TCP session with each other. Thus, BGP session establishment is a twostep process, where the first step is establishing a TCP session and the second step is exchanging BGP-specific information to build the BGP peering session.

TCP sessions operate on a client/server model. The server listens for connection attempts on a specific TCP port number. The client attempts to establish TCP sessions to the port number on which the server is listening. The client sends a TCP synchronization (TCP SYN) message to the listening server indicating that it would like to begin sending data to the server. The server responds with a TCP synchronization acknowledgment (TCP SYN ACK) message confirming it received the client's request and is ready to receive data over the connection. The client finally responds with a simple TCP acknowledgment (TCP ACK) message to acknowledge that it received the server’s SYN-ACK packet. From this point on, the client can begin sending data to the server as TCP segments. This process is known as the TCP three-way handshake.

When BGP is enabled on a router, the router begins listening for TCP server connection attempts on port 179. When the router is configured to peer with a particular neighbor (using the neighbor command in BGP router configuration mode), it attempts to establish a TCP connection with the configured neighbor by sending a TCP SYN to the potential neighbor. This process is also known as an active open attempt. The TCP SYN packet is sent with the source IP address of the outgoing interface the router uses to reach the neighbor, the destination IP address of the potential neighbor, and the destination TCP port 179. In this situation, the router is acting as a TCP client, attempting to connect to a TCP server at port 179.

The remote neighbor listens for connections coming in on TCP port 179. When it receives the TCP SYN packet, it checks its own BGP configuration to verify that the connection attempt is being made from an IP address that is designated as a potential BGP peer by using the neighbor command in its own BGP configuration. If it finds a match, it responds with a TCP SYN-ACK message; otherwise, it resets the TCP session. At the same time, the server is also sending its own TCP SYN packets to its configured neighbor in an attempt to establish a BGP peering session with it.

Because BGP routers both passively listen for TCP connections and actively attempt to create TCP sessions to configured neighbors, they act as both TCP clients and servers during the TCP exchange. If two neighbors both receive and send a TCP SYN connection to each other, the one with the higher BGP identifier (or BGP router ID) becomes the TCP client, and the one with the lower BGP router ID becomes the TCP server.

Once the TCP session is established, the routers begin the BGP peering session establishment phase, determining the type of BGP peering and exchanging capabilities. Neighbors are identified by their IP addresses and BGP autonomous system numbers (ASNs):

- If a peer’s ASN matches the local ASN, it is considered to be an internal BGP (iBGP) peer.
- If a peer’s ASN does not match the local ASN, it is considered to be an external BGP (eBGP) peer.

The two peers also exchange BGP capabilities used to negotiate the keepalive (hello) interval, hold timer value, and other session parameters. If the receiving BGP peer finds a parameter unacceptable, then the BGP peering session does not come up.

This is the basic process used to establish a BGP session between two routers. The following labs address the subtleties in how this interaction occurs. The key point to remember is that BGP uses TCP as transport and it is therefore bound by the rules of TCP regarding how it operates. Due to this reliance on TCP, BGP session establishment has two phases: TCP session establishment and BGP peering session establishment.

Task 1

Configure appropriate IP addressing as indicated in the diagram above.

The following first configures IP addressing as specified in the topology diagram:

On R1:

```
Router(config)#hostname R1

R1(config)#interface g0/2
R1(config-if)#ip address 12.1.1.1 255.255.255.0
R1(config-if)#no shut

R1(config)#interface loopback 0
R1(config-if)#ip address 1.1.1.1 255.255.255.255
```

On R2:

```
Router(config)#hostname R2

R2(config)#interface g0/1
R2(config-if)#ip address 12.1.1.2 255.255.255.0
R2(config-if)#no shut

R2(config)#interface loopback 0
R2(config-if)#ip address 2.2.2.2 255.255.255.255
```

Task 2

Configure R1 and R2 to become eBGP neighbors. They should use their Loopback0 interfaces as the peering addresses.

The task indicates that R1 and R2 should establish a BGP peering session with each other using their Loopback 0 interfaces. Using normal routing protocols such as OSPF or EIGRP, this request would be very difficult to complete. This is because protocols like OSPF and EIGRP require that prospective protocol neighbors be directly-connected to each other. BGP, because it uses TCP as transport, has no such requirements on session establishment. All the routers need is to have IP reachability to the target neighbor so the router can route the TCP and BGP control packets to the intended peer.

With the above in mind, before the peering configurations are performed, it is important to provide R1 and R2 reachability to each other's loopback addresses. This reachability can be provided with static routing, or with any IGP. The lab solution below chooses to use OSPF on R1 and R2 to provide reachability. The **network 0.0.0.0 0.0.0.0 area 0** assigns all interfaces on the two routers to Area 0 and enables OSPF to run on them.

On R1 and R2:

```
Rx(config)#router ospf 1
Rx(config-router)#network 0.0.0.0 0.0.0.0 area 0
```

On R1:

You should see the following console message:

```
%OSPF-5-ADJCHG: Process 1, Nbr 2.2.2.2 on GigabitEthernet0/2 from
LOADING to FULL, Loading Done
```

After completing the above, a ping sourced from R1’s loopback address 1.1.1.1 to R2’s address. 2.2.2.2 is issued to verify reachability:

```
R1#ping 2.2.2.2 source 1.1.1.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
Packet sent with a source address of 1.1.1.1
!!!!!
```

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

Now that the stage is set, the BGP configuration can begin. R1 and R2 should be configured to become eBGP peers. The following configuration configures R1 in AS 100 and R2 in AS 200:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#neighbor 2.2.2.2 remote-as 200
R1(config-router)#neighbor 2.2.2.2 update-source lo0
```

On R2:

```
R2(config)#router bgp 200
R2(config-router)#neighbor 1.1.1.1 remote-as 100
R2(config-router)#neighbor 1.1.1.1 update-source lo0
```

The configuration above deserves some explanation. First, the BGP process is initialized on each router using the **router BGP *ASN*** command in global configuration mode. On Cisco IOS, this enables all of the BGP processes on the router. While the command readies those processes they actually are not carried out until at least one neighbor is configured under the BGP process. The **neighbor *neighbor-address* remote-as *ASN*** commands first configure a passive TCP listening socket on the router for the specified neighbor IP address. This socket can be seen using the **show tcp brief all** command as shown below:

```
R2#show tcp brief all

TCB       Local Address              Foreign Address              (state)
120F28B0  0.0.0.0.179               1.1.1.1.*                    LISTEN
```

On R1:

```
R1#show tcp brief all

TCB       Local Address              Foreign Address              (state)
17212710  0.0.0.0.179               2.2.2.2.*                    LISTEN
```

The output above confirms that R1 and R2 have established passive listening ports for the neighbor IP addresses configured under the BGP router configuration. In R1's case, it is only willing to accept TCP packets received on its own TCP port 179 (BGP) from 2.2.2.2 with any source TCP port number. If any other IP address attempts a connection, the router will refuse it. The same logic is applied to R2. It will only accept TCP packets received on its own TCP port 179 from 1.1.1.1 with any source TCP port number.

Because the routers are configured to only accept TCP packets from a specific source IP address, when R1 and R2 attempt to open a BGP peering session with each other, they need to make sure they source the initial TCP SYN packet from the appropriate IP address. In R1's case, since R2 is listening for packets specifically from 1.1.1.1, R1 should send with a source IP address of 1.1.1.1. Likewise, R2, since R1 is listening for packets specifically from 2.2.2.2, should send its own TCP SYN with a source IP address of 2.2.2.2.

NOTE:
These listening addresses were the IP addresses directly configured on R1 and R2's **neighbor neighbor-address remote-as ASN** commands.

The **neighbor *neighbor-address* update-source *interface*** command is used to force the routers to use another interface's IP address as the update source rather than using the default which is whatever IP address is configured on R1 and R2's exit interfaces. Without this command, R1 would send its TCP SYN packet with a source address of 12.1.1.1 because its outgoing interface in its routing table to reach 2.2.2.2 is its G0/2 interface which has an IP address of 12.1.1.1 as shown below:

```
R1#show ip route 2.2.2.2

Routing entry for 2.2.2.2/32
  Known via "ospf 1", distance 110, metric 2, type intra area
  Last update from 12.1.1.2 on GigabitEthernet0/2, 00:15:16 ago
  Routing Descriptor Blocks:
  * 12.1.1.2, from 2.2.2.2, 00:15:16 ago, via GigabitEthernet0/2
      Route metric is 2, traffic share count is 1

R1#show ip interface brief | exclude un

Interface                  IP-Address      OK? Method Status                Protocol
GigabitEthernet0/2         12.1.1.1        YES manual up                    up
Loopback0                  1.1.1.1         YES manual up                    up
```

With the **neighbor update-source** command, R1 will use the IP address assigned to its lo0 (1.1.1.1) as the source IP instead. This way, the source IP R1 uses matches the source IP for which R2 is listening. The same is true on R2 for R1. R2 will use its lo0 interface IP address (2.2.2.2) as the source IP which is the same source IP for which R1 is listening.

After completing the above configuration, no console log messages confirming BGP peerings appear on R1 or R2. The **show ip bgp summary** command output on these routers also reveal that the state is IDLE:

```
R1#show ip bgp summary | begin Neigh

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
2.2.2.2         4   200       0       0        1    0    0 never    Idle
```

On R2:

```
R2#show ip bgp summary | begin Neigh

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
1.1.1.1         4   100       0       0        1    0    0 never    Idle
```

The **debug ip bgp** command on R1 reveals the following log message:

On R1:

```
R1#debug ip bgp
BGP debugging is on for address family: IPv4 Unicast

BGP: 2.2.2.2 Active open failed - no route to peer, open active delayed
12288ms (35000ms max, 60% jitter)
```

The same message appears on R2 except instead of "2.2.2.2" it references "1.1.1.1". The message is an indication that R1 and R2 are unable to open a TCP connection to each other because there is no route to the peering addresses, 2.2.2.2 and 1.1.1.1 respectively. The routing table on R1 and R2, however, reveal that both routers have OSPF routes for each other's peering addresses:

```
R1#show ip route ospf | begin Gate
Gateway of last resort is not set

      2.0.0.0/32 is subnetted, 1 subnets
O        2.2.2.2 [110/2] via 12.1.1.2, 00:25:56, GigabitEthernet0/2
```

On R2:

```
R2#show ip route ospf | begin Gate
Gateway of last resort is not set

      1.0.0.0/32 is subnetted, 1 subnets
O        1.1.1.1 [110/2] via 12.1.1.1, 00:26:18, GigabitEthernet0/1
```

So why does the log message state no route to the peer?

The reason has to do with two things: The **BGP peering type** and the **type of route** that exists in the routing table on R1 and R2.

First, while it is true that BGP neighbors do not have to be directly connected in order to be true BGP peers, the default requirements for establishing those peering relationships vary depending on the BGP peering type. In BGP, there are two types of peers, **external** and **internal**. **External peers** are formed between two BGP routers that have differing ASNs. **Internal peers** are formed between two BGP routers that have the same ASN. External peers are called **eBGP peers** and internal peers are called **iBGP peers**.

When a router attempts to form a BGP peering with a neighbor that is an eBGP peer, the router performs an implicit default check on the peering address, more commonly known as the **connected check**. This check ensures that the local router's route to reach the peer's IP address, the address specified in the neighbor command, appears as a directly-connected route in the local routers routing table. If the peering address is not a directly-connected route, the check fails. As a result, TCP connections cannot be opened, and the router logs the message "**Active open failed - no route to peer**". This is exactly what's happening in this case.

R1 and R2 have OSPF-learned routes to reach each other's peering address. This route is not directly connected and as such is invalid to be used. The result is R1 and R2 do not even attempt to form a TCP connection and the active open fails.

Recall, a similar sanity check is also performed in RIP where a router will only accept a RIP routing update from the neighbor if the source of the update belongs to the same subnet configured on the receiving router interface. In RIP, this sanity check is disabled using the no validate-update-source command in router RIP configuration mode.

The BGP sanity check can be disabled too. This is done using the **neighbor *neighbor-address* disable-connected-check** command. With this command, the router will suspend its sanity check and utilize any route in the routing table to start the active open process except for a default route.

This configuration is tested by configuring the **neighbor *neighbor-address* disable-connected-check** command on both R1 and R2 below. As soon as the configuration is completed, both R1 and R2 form eBGP peerings with each other:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#neighbor 2.2.2.2 disable-connected-check
```

On R2:

```
R2(config)#router bgp 200
R2(config-router)#neighbor 1.1.1.1 disable-connected-check
```

You should see the following console message:

```
%BGP-5-ADJCHANGE: neighbor 1.1.1.1 Up
```

The **show ip bgp neighbor** command on R1 verifies this:

On R1:

```
R1#show ip bgp neighbor 2.2.2.2 | include TTL

Connection is ECN Disabled, Mininum incoming TTL 0, Outgoing TTL 1
```

Notice the TTL value is set to 1 in the outgoing packet. A capture of a TCP packet sent by R1 to R2 verifies that the TTL in the IP header is set to 1 by default:

```
Internet Protocol Version 4, Src: 1.1.1.1, Dst: 2.2.2.2
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x021f (543)
    Flags: 0x4000, Don't fragment
    Time to live: 1
    Protocol: TCP (6)
    Header checksum: 0x70ec [validation disabled]
    [Header checksum status: Unverified]
    Source: 1.1.1.1
    Destination: 2.2.2.2
Transmission Control Protocol, Src Port: 24228, Dst Port: 179, Seq: 1,
Ack: 1, Len: 0
```

This point is used to disprove a commonly-held belief that in order to form an eBGP peering session between the loopback interfaces of two routers, the TTL value would need to be manually set from its default of 1 to a value of 2. This isn't a baseless belief and is rooted in how routers decrement the TTL value of the IP packet. The TTL or **time-to-live** value of an IP packet indicates how many router hops an IP packet can travel. Each router decrements the TTL value by 1 before forwarding on to another router or network segment. If the TTL value is 0, the router will not forward the packet and will send an **ICMP Time Exceeded In-Transit** message back to the source.

The theory is as follows. R1 sends a TCP SYN packet to R2 with a TTL value of 1. This TCP SYN is destined to R2's lo0 interface 2.2.2.2. When R2 receives the packet, it decrements the TTL value to 0. With the TTL value now at zero, R2 would be unable to route the packet to its Lo0 interface preventing the session from being established.

The above, however, disproves this theory. R1 and R2 are able to establish an eBGP peering without changing the TTL value. This is because, even though the peerings are established over loopback interfaces, the TTL value restriction only applies to actual router hops. Packets routed to a loopback interface do not incur this penalty because they are still being consumed by the local router and are not being routed to a remote subnet or device.

Task 3

Reconfigure the above BGP peering such that R1 and R2 are able to become eBGP peers without using the **disable-connected-check** command. Ensure that the TTL value of sent packets is as low as possible. Use a BGP-related command to accomplish this task.

The previous task demonstrated the use of the **neighbor disable-connected-check** command to prevent the router from performing a connected check for the peering address. The task above requires reconfiguring R1 and R2 such that they form an eBGP peering without the use of the **disable-connected-check** command.

To begin, the **neighbor disable connected-check** configuration is first removed from R1 and R2 below:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#no neighbor 2.2.2.2 disable-connected-check
```

On R2:

```
R2(config)#router bgp 200
R2(config-router)#no neighbor 1.1.1.1 disable-connected-check
```

There are situations in eBGP peering where the local router intends to peer with another BGP-speaking router that is more than one router hop away. In such a situation, the local router needs to send its BGP messages with a TTL value higher than 1 to reach the remote peer. The **neighbor ebgp-multihop *hop-count*** command takes in a numerical value that indicates how many router hops away the remote peer is. It also causes the router to set the TTL value in the IP header of sent BGP messages to the indicated value. The value ranges from 1 - 255:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#neighbor 2.2.2.2 ebgp-multihop ?
  <1-255>  maximum hop count
```

With this understanding, the **neighbor *neighbor-address* ebgp-multihop *hop-count*** command sounds like a good candidate for solving this task. Applied to the topology, both R1 and R2 are only a single hop away, thus the **neighbor *neighbor-address* ebgp-multihop 1** command should be used to indicate the routers are only a single hop away. In the below, both R1 and R2 are configured with the **neighbor *neighbor-address* ebgp-multihop 1** command and the **debug ip bgp** is enabled on R1:

```
R1#debug ip bgp
BGP debugging is on for address family: IPv4 Unicast

R1(config)#router bgp 100
R1(config-router)#neighbor 2.2.2.2 ebgp-multihop 1
```

On R2:

```
R2(config)#router bgp 200
R2(config-router)#neighbor 1.1.1.1 ebgp-multihop 1
```

After completing the above, the eBGP peerings between R1 and R2 do not form. A familiar log message “**no route to peer**” can be seen on R1. The same happens on R2 as well.

On R1:

```
BGP: 2.2.2.2 Active open failed - no route to peer, open active delayed
7168ms (35000ms max, 60% jitter)
```

The reason that the R1 and R2 do not form eBGP peerings with each other is because, even though the routers have been told the peer is only a single hop away, the connected check has not been disabled. This is evidenced by the **show ip bgp neighbor 2.2.2.2 | include external|External** command output below:

```
R1#show ip bgp neighbor 2.2.2.2 | include external|External

BGP neighbor is 2.2.2.2,  remote AS 200, external link
  External BGP neighbor not directly connected.
  External BGP neighbor configured for connected checks (single-hop no-
disable-connected-check)
```

The highlighted output above indicates that the external BGP neighbor is still configured to perform connected checks.

The behavior makes sense. If the router is told the peer is only a single hop away, then the peer should be directly-connected to the local router. This means that using the **ebgp-multihop** command with a TTL value of 1 will still require the use of the **disable-connected-check** command. The task explicitly forbids the use of the **disable-connected-check** command, invalidating the option.

There is a hidden effect to the **ebgp-multihop** command. If set to a value of 1 the connected check is retained. However, if set to a value higher than one, the command silently disables the connected check. To verify this, the multihop configuration on R1 and R2 is increased to the value 2:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#neighbor 2.2.2.2 ebgp-multihop 2
```

On R2:

```
R2(config)#router bgp 200
```

```
R2(config-router)#neighbor 1.1.1.1 ebgp-multihop 2
```

You should see the following log messages on R1 and R2:

```
%BGP-5-ADJCHANGE: neighbor 2.2.2.2 Up

%BGP-5-ADJCHANGE: neighbor 1.1.1.1 Up
```

After completing the above, the eBGP peering comes up between R1 and R2. In addition, the capture belows show a TCP packet from R1 to R2 with a TTL value of 2 as specified in the **neighbor 2.2.2.2 ebgp-multihop 2** command:

```
Internet Protocol Version 4, Src: 1.1.1.1, Dst: 2.2.2.2
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x33b9 (13241)
    Flags: 0x4000, Don't fragment
    Fragment offset: 0
    Time to live: 2
    Protocol: TCP (6)
    Header checksum: 0x3e52 [validation disabled]
    [Header checksum  status:  Unverified]
    Source: 1.1.1.1
    Destination: 2.2.2.2
Transmission Control Protocol, Src Port: 179, Dst Port: 39826, Seq: 119,
Ack: 119, Len: 0
     [ -- output omitted -- ]
```

It is important to understand why the eBGP peering session has come up with an eBGP multihop setting of 2. It is not related to how the TTL value was set, as shown in the previous task where no such TTL modification was made and the eBGP peer session between R1 and R2 did come up. In this configuration, as far as BGP is concerned, the neighbor can be up to 2 hops away but, more importantly, the routers have disabled connected checks for the neighbor. This is proven using the **show ip bgp neighbors 2.2.2.2** output:

On R1:

```
R1#show ip bgp neighbors 2.2.2.2

BGP neighbor is 2.2.2.2,  remote AS 200, external link
  BGP version 4, remote router ID 2.2.2.2
  BGP state = Established, up for 00:06:43
  [ -- output omitted -- ]
  Address tracking is enabled, the RIB does have a route to 2.2.2.2
  Connections established 3; dropped 2
  Last reset 00:11:30, due to Active open failed
  External BGP neighbor may be up to 2 hops away.
  External BGP neighbor NOT configured for connected checks (multi-hop
no-disable-connected-check)
  Interface associated: (none) (peering address NOT in same link)
  Transport(tcp) path-mtu-discovery is enabled
  Graceful-Restart is disabled
```

The highlighted output above states "**External BGP neighbor NOT configured for connected checks** ". In other words, R1 is not performing the default connected check on the peering address.

This behavior is logical. If the router is being explicitly told the peer is more than one hop away, then performing a connected check makes no sense. In such a case, the router skips the check and is able to send its BGP messages as normal with the modified TTL value. This configuration also meets the task requirements of using the lowest TTL value possible and not using the disable-connected-check command.

Note: The command **neighbor** neighbor-address **ebgp-multihop** can also be issued without an explicit value specified. If no TTL value is specified, the router defaults to setting the TTL to 255. In both cases with a value of 2 or higher, or no value, the eBGP connected check is disabled on the router.

Task 4

Reconfigure R1 and R2 such that they form an eBGP peering with each other. Ensure that the TTL value of any received BGP packet is no less than 253. Do not use the **disable-connected-check** command.

This task imposes a different requirement on the topology. It explicitly states that the TTL value of the received BGP packet should not be less than 253 in order for the peering session to be established. The current configuration explicitly sets the TTL value to 2. As such, it should be removed before proceeding with the solution:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#no neighbor 2.2.2.2 ebgp-multihop 2
```

On R2:

```
R2(config)#router bgp 200
R2(config-router)#no neighbor 1.1.1.1 ebgp-multihop 2
```

The previous two methods employed two different mechanisms for establishing eBGP peering sessions. The first, made no modifications to the TTL value, instead disabling the eBGP connected check using the **neighbor** x.x.x.x **disable-connected-check** command. The second method utilized the **ebgp-mul**tihop feature to set the TTL to a value higher than 1 and implicitly disable the connected check. Though they function similarly, they do not impose a direct TTL minimum on received BGP packets.

There is a security feature available for BGP peer configuration that does impose a minimum TTL value restriction on BGP peers. This feature is known as the **Generalized TTL Security Feature (GTSM)**. This feature is defined in RFC 5082. The basic operation is that a device is configured to only accept a received protocol packet if it meets or exceeds a minimum TTL value. This feature is implemented to prevent CPU-based denial of service attacks where an attack will flood the router with invalid routing protocol traffic that must be processed by the routers forwarding engine. With GTSM, the router will drop packets at the hardware level due to the TTL being out of scope. It is most effective when used between protocol peers that are supposed to be directly-connected.

RFC 7454 outlines the application of the GTSM to BGP as a security mechanism to prevent BGP session spoofing. GTSM is enabled using the **neighbor** x.x.x.x **ttl-security hops** *hop-count* command. When configured with a hop count, the command sets a minimum expectation on the TTL value in the IP header received from the indicated potential neighbor. If the received TTL value in the IP header is equal to or greater than the value specified on a router, the router accepts and processes the connection attempt. Else, the router rejects the connection attempt.

When GTSM is enabled using the **neighbor** x.x.x.x **ttl-security hops** *hops* command, the router does two things:

1. Sets the TTL of outgoing packets to 255
2. Only accepts the BGP packet if the TTL is equal to or greater than the expected TTL value

The first change is imperative. The GTSM is based on received hop count values. The TTL field in the IP header is basically a hop counter that starts at 255 and counts down. In order for the GTSM to accurately know how many hop counts away the received packet is, then packets originating from a BGP router with GTSM enabled must start with a hop count of 255. The full reason for this is tied directly to the second point above.

In the second point, the router configures its interface not to accept a BGP packet that has a TTL value that indicates the router is further than the configured hop count value in the TTL security command. This TTL value is called the expected TTL value. The expected TTL is calculated by subtracting the hop-count value specified from 255. The logic here is simple. If the router is configured to only peer with another router that is at most 2 hops away, an assumption is made that this router is originating its BGP packets with a TTL of 255 as well. The TTL is decremented at each router hop. Since the TTL is decremented at each hop, the local router should only accept the packet if the received TTL value is 255 - 2 hops or 253 or greater.

By default, BGP routers originate their BGP packets to eBGP neighbors with a hop count of 1. This means if one peer is configured with the **neighbor** x.x.x.x **ttl-security hops 2** command and the other side is not, the peer that does not have GTSM enabled will still originate packets with TTL set to 1. If the peer is only a single hop away, the GTSM-enabled peer will drop the attempt because the received TTL value of 1 is less than the expected TTL value of 254. This is why for proper GTSM operation, both BGP peers should be configured for GTSM. The resulting configuration will automatically force the routers to originate their BGP packets with a TTL of 255, allowing the GTSM process to work properly. This is why the first point in the effects of the TTL security command is important.

The BGP GTSM feature hop count setting works similarly to the **ebgp-mutlihop** setting. If the value is set to 2 or greater, then the router will silently disable the BGP connected check. However, if set to a value of 1, then the router will not disable the BGP connected check. This is shown below where R1 has configured BGP GTSM with a hop count of 1 with **debug ip bgp** enabled:

<u>On R1:</u>

```
R1#debug ip bgp
BGP debugging is on for address family: IPv4 Unicast

R1(config)#router bgp 100
R1(config-router)#neighbor 2.2.2.2 ttl-security hops 1

BGP: 2.2.2.2 Active open failed - no route to peer, open active delayed
```

```
11264ms (35000ms max, 60% jitter)
```

Just as with the **ebgp-multihop** command, setting the value to 1 generates the same active open failure on the router. Once again, the **disable-connected-check** command could be used to mitigate this, but this would not complete the task.

The task states that the BGP packers received by the routers should be received with a TTL value of 253 or greater. Setting the **ttl-security hops** command to 1 would mean the TTL value of received packets should be 254 or greater. The proper setting for this command is to set the **ttl-security hops** to 2. Doing so has two effects: It automatically disables the BGP connected check and sets the minimum received TTL to 253. This is the configuration used in the below:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#neighbor 2.2.2.2 ttl-security hops 2
```

On R2:

```
R2(config-if)#router bgp 200
R2(config-router)#neighbor 1.1.1.1 ttl-security hops 2
```

After configuration the **show ip bgp neighbors 2.2.2.2 | include external|External|Minimum** command confirms the connected check has been disabled. Also notice the minimum TTL value has to be 253, and the outgoing TTL value is set to 255:

On R1:

```
R1#show ip bgp neighbors 2.2.2.2 | include external|External|Minimum

BGP neighbor is 2.2.2.2,  remote AS 200, external link
  External BGP neighbor may be up to 2 hops away.
  External BGP neighbor NOT configured for connected checks (multi-hop no-
disable-connected-check)
Connection is ECN Disabled, Minimum incoming TTL 253, Outgoing TTL 255
```

Task 5

Configure R1 and R2 to become eBGP peers. Do not use **disable-connected-check**, **ebgp-multihop**, or **ttl-security** to accomplish this task. Do not configure any tunneling mechanisms or IRB to accomplish this task.

The previous tasks established an eBGP peering session between R1 and R2 using three methods all revolving around disabling BGP's connected check mechanism. The commands accomplished this goal in different ways and with different effects on the TTL values of the resulting BGP packets. The following summarizes these commands covered so far:

1. **neighbor disable-connected-check** command: Used to disable the BGP connected check when establishing eBGP peers using loopback interfaces
    a. The TTL of the resulting BGP packets are still set to 1
    b. Most useful between directly-connected eBGP peers that use loopback interfaces as their BGP peering addresses
2. **neighbor ebgp-multihop** command: Indicates how many hops away a BGP peer is and sets the TTL value in the BGP packet to that specific value.
    a. If set to a value greater than 1, the command automatically disables the BGP connected check
    b. Most useful when prospective eBGP peers are separated by multiple router hops
    c. Can be used by directly-connected eBGP peers that use loopback interfaces as their peering addresses
        i. In this case, the hop count should be 2 or greater. This disables the BGP connected check and is the true reason behind why this command results in a successful eBGP peering session between the two routers.
3. **neighbor ttl-security hops** command: A security mechanism that sets a Minimum TTL value for received BGP packets from a particular configured neighbor. Packets from the neighbor arriving with a TTL value less than the minimum are discarded.
    a. If the **hops** value is set to a value greater than 1, then this command automatically disables the BGP connected check
    b. Most useful to provide security against BGP spoofing attacks. It drops spoofed packets that arrive with a TTL value lower than the minimum value. It should be noted that this command imposes a maximum hop count. In other words, the configured peer can only be a maximum of **hops** value away from the router on which the command is configured.

c. Should be configured on BOTH eBGP peers for compatibility

d. Can be used by directly-connected eBGP peers that use loopback interfaces as their peering addresses

    i. In this case, the **hops** value should be 2 or greater. This disables the BGP connected check and is the true reason behind why this command results in a successful eBGP peering session between the two routers.

Each of the configuration styles above result in modifications to the base BGP behavior. This task seeks a way to allow R1 and R2 to form an eBGP peering session with each other without modifying BGP's normal behavior. This means none of the commands listed above should be configured as a solution to this problem.

The solution to this task lies in understanding what the original problem was between R1 and R2. R1 and R2 are supposed to be eBGP neighbors. IOS performs a directly-connected check for eBGP neighbors. This means the route used to reach a potential eBGP neighbor's peering address should lead out a directly-connected interface.

R1 and R2, however, are configured to peer using their loopback interfaces (1.1.1.1 and 2.2.2.2 respectively) as peering addresses. The problem is the only way R1 can reach R2's loopback interface is through an OSPF-learned route that recurses to the directly-connected interface to R2. The route itself is not a directly-connected route (appears as "O" in the routing table instead of "C"). The same is true on R2 for R1's loopback interface.

To complete this task, the routers' route to reach the peering address needs to be changed to be a directly-connected address. One way to solve the task is by using various tunneling mechanisms such as L2VPN or GRE tunneling. The task explicitly forbids these methods. One way that is not prohibited by the task is by configuring the point-to-point protocol over Ethernet (PPPoE).

PPPoE is an application of PPP over ethernet interfaces. PPPoE is used to provide individual services to subscribers connected to a common DSL shared connection. The subscriber devices (called CPE devices) dial into a centralized PPPoE server that assigns an IP address to the subscribers and allows the service provider to track statistics for billing and other things.

Applied to this topology, PPPoE will be used to allow R1 and R2 to be directly connected. R1 will be configured as the PPPoE server and R2 will be the PPPoE client. Before beginning the configuration, the BGP process on R1 and R2 is completely removed:

On R1:

```
R1(config)#no router bgp 100
```

On R2:

```
R2(config)#no router bgp 200
```

The above step was used to make sure all BGP-related configurations are removed.

Next, R1 is configured as the PPPoE server. The configuration begins by removing the IP address on the G0/2 interface on R1. Then, a virtual-template interface is created. Virtual-template interfaces are interfaces used to apply configuration information like IP addressing to other interfaces. In this case, the virtual-template will be configured with the **ip unnumbered** feature referencing R1's Loopback 0 interface. Through this configuration, the virtual-template borrows the IP address 1.1.1.1/32 from Loopback 0.

Remove IP address from G0/2

On R1:

```
R1(config)#interface g0/2
R1(config-if)#no ip address
```

Configure virtual-template 12 to borrow loopback0 IP address

```
R1(config)#interface virtual-template 12
R1(config-if)#ip unnumbered lo0
```

After the virtual interface is configured, PPPoE configuration is applied to ethernet interfaces using what is known as a broadband access group (bba-group). The bba-group will inherit the settings from the previously configured virtual-template interface. Bba group "tst" is created with the type **pppoe** and the previously created virtual template is applied to the bba-group.

```
R1(config)#bba-group pppoe tst
R1(config-bba-group)#virtual-template 12
```

After all of the above is configured, the bba-group is applied to the G0/2 interface on R1.

```
R1(config)#interface g0/2
R1(config-if)#pppoe enable group tst
```

Next, R2 needs to be configured as the PPPoE client. This configuration is completed by creating a dialer interface and borrowing the loopback 0 IP address from that interface. The dialer interface is configured with PPP encapsulation and is added to what is known as a dialer pool. The dialer pool number is used to tie the PPP configuration to the ethernet interface. As with R1, the previous IP addressing is removed from R2's G0/1 interface:

Creating the dialer interface with PPP encapsulation and dialer pool

On R2:

```
R2(config)#interface dialer 21
R2(config-if)#ip unnumbered lo0
R2(config-if)#encapsulation ppp
R2(config-if)#dial pool 100
```

Remove IP address from G0/1 and apply the pppoe-client dial pool configuration

```
R2(config)#interface g0/1
R2(config-if)#no ip address
R2(config-if)#pppoe-client dial-pool-number 100
```

After configuring PPPoE, the BGP configuration can be re-applied to the routers. This time, only **neighbor** x.x.x.x **remote-as** *asn* commands are needed:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#neighbor 2.2.2.2 remote 200
```

On R2:

```
R2(config)#router bgp 200
R2(config-router)#neighbor 1.1.1.1 remote 100
```

You should see the following console output on R1 and R2:

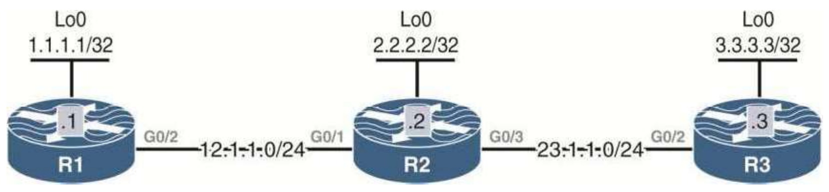
```
%BGP-5-ADJCHANGE: neighbor 2.2.2.2 Up
%BGP-5-ADJCHANGE: neighbor 1.1.1.1 Up
```

NOTE: Because OSPF is still configured to run on all interfaces, the routers may log **"%OSPF-5-ADJCHG: Process 1, Nbr 1.1.1.1 on Dialer21 from EXSTART to DOWN, Neighbor Down: Too many retransmissions**" messages.

This is because an OSPF neighborship is trying to be established between a virtual-access interface on R1 and the Dialer interface on R2. These interface types use different MTU values (1492 for Virtual-Access interfaces and 1500 for Dialer interface). Because of the different MTU values, OSPF's DBD exchange process cannot complete. The solution is to statically set the MTU to match on the two interfaces or use the **ip ospf mtu-ignore** command under the virtual-template configuration on R1 and the Dialer interface configuration on R2.

Task 6

Reload the routers and configure the following topology.



The following configures the network as indicated in the diagram above:

On R1:

```
Router(config)#hostname R1

R1(config)#interface g0/2
R1(config-if)#ip address 12.1.1.1 255.255.255.0
R1(config-if)#no shut

R1(config)#interface lo0
R1(config-if)#ip address 1.1.1.1 255.255.255.255
```

On R2:

```
Router(config)#hostname R2

R2(config)#interface g0/1
R2(config-if)#ip address 12.1.1.2 255.255.255.0
R2(config-if)#no shut

R2(config)#interface g0/3
R2(config-if)#ip address 23.1.1.2 255.255.255.0
```

```
R2(config-if)#no shut

R2(config)#int lo0
R2(config-if)#ip address 2.2.2.2 255.255.255.255
```

On R3:

```
Router(config)#hostname R3

R3(config)#interface g0/2
R3(config-if)#ip address 23.1.1.3 255.255.255.0
R3(config-if)#no shut

R3(config-if)#interface lo0
R3(config-if)#ip address 3.3.3.3 255.255.255.255
```

Task 7

Configure R1 and R3 with an eBGP session using their Loopback0 interfaces.

The task specifies that R1 and R3 should establish an eBGP peering session using their loopback interfaces as the peering addresses. From the earlier sections, it has been established that BGP can establish peering sessions with devices that are multiple hops away. All that is needed is reachability to the peering address.

The first step in solving this task is to provide reachability between the Lo0 interface on R1 to the Loopback0 interface on R3. For this purpose, OSPF for area 0 is enabled on all of the routers as follows:

On R1:

```
R1(config)#router ospf 1
R1(config-router)#network 0.0.0.0 0.0.0.0 area 0
```

On R2:

```
R2(config)#router ospf 1
R2(config-router)#network 0.0.0.0 0.0.0.0 area 0
```

On R3:

```
R3(config)#router ospf 1
R3(config-router)#network 0.0.0.0 0.0.0.0 area 0
```

After configuring OSPF and waiting for the neighbor adjacencies to come up, the routing table on R1 and R3 along with accompany ping output verifies reachability between the loopback interfaces on the routers:

On R1:

```
R1#show ip route ospf | begin Gate
Gateway of last resort is not set

      2.0.0.0/32 is subnetted, 1 subnets
O        2.2.2.2 [110/2] via 12.1.1.2, 00:01:55, GigabitEthernet0/2
      3.0.0.0/32 is subnetted, 1 subnets
O        3.3.3.3 [110/3] via 12.1.1.2, 00:00:12, GigabitEthernet0/2
      23.0.0.0/24 is subnetted, 1 subnets
O        23.1.1.0 [110/2] via 12.1.1.2, 00:01:55, GigabitEthernet0/2


R1#ping 3.3.3.3 source lo0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 3.3.3.3, timeout is 2 seconds:
Packet sent with a source address of 1.1.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/2 ms
```

On R3:

```
R3#show ip route ospf | begin Gate
Gateway of last resort is not set

      1.0.0.0/32 is subnetted, 1 subnets
O        1.1.1.1 [110/3] via 23.1.1.2, 00:01:01, GigabitEthernet0/2
      2.0.0.0/32 is subnetted, 1 subnets
O        2.2.2.2 [110/2] via 23.1.1.2, 00:01:01, GigabitEthernet0/2
      12.0.0.0/24 is subnetted, 1 subnets
O        12.1.1.0 [110/2] via 23.1.1.2, 00:01:01, GigabitEthernet0/2

R3#ping 1.1.1.1 source lo0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 1.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 3.3.3.3
```

```
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/2 ms
```

Now that reachability has been confirmed, an eBGP peering session needs to be configured between the two routers. The first thing that needs to be decided is what ASN the two routers will use. Since the peering is meant to be an eBGP peering, then the two ASNs must be different. The solution chooses ASN 100 for R1 and ASN 300 for R3.

After determining which ASNs will be used, some thought must be given to exactly how the BGP messages will be exchanged between the two routers. In previous examples, the eBGP peering was a single hop away. The tasks examined the implications of single-hop eBGP peering sessions over loopback interfaces. It was established that the default eBGP TTL of 1 did not prevent the peers from forming eBGP peering sessions. The real hurdle was the BGP connected check that needed to be disabled in order to allow the peering session.

With this in mind, R1 and R3 will be configured to form an eBGP peering session using the following configuration commands:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#neighbor 3.3.3.3 remote-as 300
R1(config-router)#neighbor 3.3.3.3 update-source lo0
R1(config-router)#neighbor 3.3.3.3 disable-connected-check
```

On R3:

```
R3(config)#router bgp 300
R3(config-router)#neighbor 1.1.1.1 remote-as 100
R3(config-router)#neighbor 1.1.1.1 update-source lo0
R3(config-router)#neighbor 1.1.1.1 disable-connected-check
```

After applying the configuration above, the show ip bgp summary output lists the potential neighbor as “Idle” on both R1 and R3:

On R1:

```
R1#show ip bgp summary

BGP router identifier 1.1.1.1, local AS number 100
BGP table version is 1, main routing table version 1
```

```
Neighbor        V       AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down
State/PfxRcd
3.3.3.3         4      300       0       0        1    0    0 never    Idle
```

On R3:

```
R3#show ip bgp summary

BGP router identifier 3.3.3.3, local AS number 300
BGP table version is 1, main routing table version 1

Neighbor        V       AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down
State/PfxRcd
1.1.1.1         4      100       0       0        1    0    0 never    Idle
```

In the earlier tasks, a state of "Idle" meant there was a problem with the BGP session establishment process. The debug ip bgp and debug ip bgp outputs on R1 helps to track down the issue:

On R1:

```
BGP: ses global 3.3.3.3 (0x169AFDF8:0) act Reset (Active open failed).
BGP: 3.3.3.3 active went from Active to Idle
BGP: nbr global 3.3.3.3 Active open failed - open timer running
BGP: nbr global 3.3.3.3 Active open failed - open timer running

ICMP: time exceeded rcvd from 12.1.1.2
ICMP: time exceeded rcvd from 12.1.1.2
```

The first output indicates that R1 is attempting to perform an Active open for the eBGP session but the active open is failing. The second part of the output gives an indication of why. R2 is responding to R1 with an ICMP time exceeded message. This message is a result of the default TTL value for eBGP packets being set to 1. The process is as follows:

1. R1 tries to send a TCP SYN packet to 3.3.3.3 sourced from its lo0 interface (1.1.1.1) with a TTL of 1
2. R2 receives the TCP packet and decrements the TTL value from 1 to 0.
3. R2 determines this packet needs to be routed out of another interface. An insufficient TTL value, triggers the TTL time exceeded message.
4. R2 drops the TCP packet and responds with an ICMP time exceeded message
5. R3 never receives the TCP packet resulting in no BGP peer session being established.

The same happens whenever R3 attempts to send a TCP SYN packet to R1 to attempt an active open. With default eBGP TTL settings, the eBGP peering session between R1 and R3 cannot complete. This is where the TTL modification techniques used in the previous tasks come in. There are two ways to modify the TTL, using the **ebgp-multihop** command or the **ttl-security hops** command. Each accomplishes the goal in a separate way.

With **ebgp-multihop** the TTL of the eBGP packets are modified to the specified hop count value. In this case, R3 is an additional router hop away from R1. Thus, the **ebgp-multihop 2** command should be used to manually set the TTL to 2. This way, R2 can route the BGP packet to R3. R3 will receive the BGP packet with a TTL of 1 and will be able to process the request.

The **ttl-security hops** command works by setting the TTL value to the maximum value (255) and imposing a minimum TTL value accepted limit. Applied to the topology, R3 is again an additional router hop away, so the **ttl-security hops 2** command should be used. Doing so will make it such that R1 will send its active open packet with a value of 255. It will be decremented to 254 at R2 and R3 will receive it with a TTL value of 253 (after decrementing it itself). Also, R1 will not accept a BGP packet from R3 that has a TTL value less than 253. For this reason, both R1 and R3 should be configured with the same **ttl-security hops** command.

No matter which method is utilized, because the hop count is 2, the router will automatically disable the BGP connected check rendering the manual **disable-connected-check** command unnecessary. The following uses the **ebgp-multihop** command to solve the task. First, the **disable-connected-check** command is removed from both configurations. Then, the **ebgp-multihop** is added:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#no neighbor 3.3.3.3 disable-connected-check
R1(config-router)#neighbor 3.3.3.3 ebgp-multihop 2
```

On R3:

```
R3(config)#router bgp 300
R3(config-router)#no neighbor 1.1.1.1 disable-connected-check
R3(config-router)#neighbor 1.1.1.1 ebgp-multihop 2
```

After configuration you should see the following message on R1 and R3:

```
R1#
%BGP-5-ADJCHANGE: neighbor 3.3.3.3 Up
```

```
R3#
%BGP-5-ADJCHANGE: neighbor 1.1.1.1 Up
```

Task 8

Reconfigure R1 and R3 with an eBGP session without modifying the TTL value, configuring IRB, or using GRE or IPnIP tunneling mechanisms.

In the previous task, a multihop eBGP peering session was established between R1 and R3's loopback interfaces. In order to make this possible, both routers needed to be configured with the **ebgp-multihop 2** or **ttl-security hops 2** command. Doing so accomplished two goals: disabling BGP's connected check and increasing the TTL value of BGP packets so R2 could successfully route the packets between the routers.

What if the requirement was to have R1 and R3 become eBGP peers but to keep the TTL value at 1? This is the problem that this task seeks to create. The task is to recreate the current eBGP peering session between R1 and R3 without modifying the TTL value or using IRB, GRE, or IPnIP tunneling.

To prepare for the solution, the **ebgp-multihop** configuration applied in task 7 is removed and the **disable-connected-check** configuration is applied again. Additionally, the peers are reset to clear the existing session:

Removing the commands:

On R1:

```
R1(config)#router bgp 100
R1(config-router)#no neighbor 3.3.3.3 ebgp-multihop 2
R1(config-router)#neighbor 3.3.3.3 disable-connected-check
```

On R3:

```
R3(config)#router bgp 300
R3(config-router)#no neighbor 1.1.1.1 ebgp-multihop 2
R3(config-router)#neighbor 1.1.1.1 disable-connected-check
```

Resetting the BGP peering sessions:

On R1:

```
R1#clear ip bgp *

%BGP-3-NOTIFICATION_MANY: sent to 1 sessions 6/4 (Administrative Reset)
for all peers
%BGP-5-ADJCHANGE: neighbor 3.3.3.3 Down User reset

%BGP_SESSION-5-ADJCHANGE: neighbor 3.3.3.3 IPv4 Unicast topology base
removed from session  User reset
```

On R3:

```
R3#clear ip bgp *

%BGP-3-NOTIFICATION_MANY: sent to 1 sessions 6/4 (Administrative Reset)
for all peers
%BGP-5-ADJCHANGE: neighbor 1.1.1.1 Down User reset

%BGP_SESSION-5-ADJCHANGE: neighbor 1.1.1.1 IPv4 Unicast topology base
removed from session  User reset
```

Verifying Idle peering sessions:

```
R3#show ip bgp summary

BGP router identifier 3.3.3.3, local AS number 300
BGP table version is 1, main routing table version 1

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down State/PfxRcd
1.1.1.1         4   100       0       0        1    0    0 00:04:42 Idle
```

On R1:

```
R3#show ip bgp summary

BGP router identifier 1.1.1.1, local AS number 100
BGP table version is 1, main routing table version 1

Neighbor       V     AS MsgRcvd MsgSent   TblVer   InQ OutQ Up/Down  State/PfxRcd
3.3.3.3        4    300       0       0        1    0    0 00:04:43 Idle
```

Now that the peering sessions have been reverted back to their original configurations, the process of solving the task can begin. What is needed is a way to make R1 and R3 believe they are directly connected to